Efficient Inversion of Large-Scale Problems Exploiting Structure and Randomization

by

Jarom D. Hogue

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2020 by the
Graduate Supervisory Committee:

Rosemary A. Renaut, Chair
Zdzislaw Jackiewicz
Rodrigo B. Platte
Christian Ringhofer
Bruno Welfert

ARIZONA STATE UNIVERSITY

August 2020

ABSTRACT

Dimensionality reduction methods are examined for large-scale discrete problems, specifically for the solution of three-dimensional geophysics problems: the inversion of gravity and magnetic data. The matrices for the associated forward problems have beneficial structure for each depth layer of the volume domain, under mild assumptions, which facilitates the use of the two dimensional fast Fourier transform for evaluating forward and transpose matrix operations, providing considerable savings in both computational costs and storage requirements. Application of this approach for the magnetic problem is new in the geophysics literature. Further, the approach is extended for padded volume domains.

Stabilized inversion is obtained efficiently by applying novel randomization techniques within each update of the iteratively reweighted scheme. For a general rectangular linear system, a randomization technique combined with preconditioning is introduced and investigated. This is shown to provide well-conditioned inversion, stabilized through truncation. Applying this approach, while implementing matrix operations using the two dimensional fast Fourier transform, yields computationally effective inversion, in memory and cost. Validation is provided via synthetic data sets, and the approach is contrasted with the well-known LSRN algorithm when applied to these data sets. The results demonstrate a significant reduction in computational cost with the new algorithm. Further, this new algorithm produces results for inversion of real magnetic data consistent with those provided in literature.

Typically, the iteratively reweighted least squares algorithm depends on a standard Tikhonov formulation. Here, this is solved using both a randomized singular value decomposition and the iterative LSQR Krylov algorithm. The results demonstrate that the new algorithm is competitive with these approaches and offers the advantage that no regularization parameter needs to be found at each outer iteration.

Given its efficiency, investigating the new algorithm for the joint inversion of these data sets may be fruitful. Initial research on joint inversion using the two dimensional fast Fourier transform has recently been submitted and provides the basis for future work. Several alternative directions for dimensionality reduction are also discussed, including iteratively applying an approximate pseudo-inverse and obtaining an approximate Kronecker product decomposition via randomization for a general matrix. These are also topics for future consideration.

ACKNOWLEDGMENTS

My advisor, Rosie Renaut, has been instrumental in helping me with my research. She first introduced me to inverse problems during an undergrad research experience at ASU, and has provided many invaluable insights and suggestions over the years.

I would like to thank Youzuo Lin for mentoring me at Los Alamos National Laboratory and introducing me to randomization techniques. His positive attitude and encouragement have been a source of inspiration.

I would like to thank the members of my committee for their time, invaluable comments and expertise, as well as my other professors for their skillful dissemination of information.

I would like to thank my many friends and fellow graduate students who all contributed to a warm and friendly environment.

Most of all I would like to thank my wife and kids for their continuous love, patience and support. They are a constant source of light and joy in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Technology increasingly enables the collection of massive amounts of data. While this causes calculation of the discrete forward operation $A\boldsymbol{x} = \boldsymbol{b}$ to be more computationally expensive, this data is often of use in determining an approximate solution of the more expensive discrete inverse problem to find $\boldsymbol{x}$ given A and $\boldsymbol{b}$. Inverting to find $\boldsymbol{x}$ often requires some form of matrix decomposition followed by multiple forward multiplications. Even if A is invertible, this also often requires some form of regularization to produce a useful solution. As the amount of data collected increases, the computational cost of the inverse problem increases by a multiplicative factor. Coupled with a similar effect from increasing the number of desired model parameters, solving these large problems has proven difficult for technology to keep up with, Lin et al. [2016], Wang [2015]. To combat this, some techniques for solving large-scale inverse problems utilize dimensionality reduction in some form. The aim of this work is to further develop dimensionality reduction techniques that allow stable and efficient solutions of inverse problems.

Inverse problems are introduced in Chapter 2. A framework for linear ill-posed inverse problems, including discretization, ill-posedness, regularization and regularization parameters is also established. Choice of norm is considered based on desired properties of the unknown solution, and methods are presented that allow an analytical solution to be obtained. The usefulness of the singular value decomposition (SVD) is shown, and computational limitations when the scale becomes large are discussed. As examples of large-scale 3D problems, geophysical applications for inverting `gravity` and `magnetic` data are introduced.

In Chapter 3 it is shown that a block-Toeplitz with Toeplitz blocks (BTTB) matrix can be extended to a block circulant with circulant blocks (BCCB) matrix, for which matrix multiplication can be performed via a convolution transformation using the two-dimensional fast Fourier transform (2DFFT), Vogel [2002]. Moreover, this formulation can be applied to the `gravity` and `magnetic` problems, and also be extended when considering model contributions from outside the discrete domain, for both the symmetric and non-symmetric case. An in-depth formulation is provided for the configurations with and without domain padding. The details of the algorithmic implementation and a computational comparison are provided.

It is shown in Chapter 4 that the application of randomization directly to a matrix system can further reduce the computational requirements. Standard randomized system sketches are discussed. The naive extension of the randomized sketch to the right of the discrete forward operator, via a substitution, fails to produce a viable result, and provides motivation for the LSRN method, Meng et al. [2014], which shows a viable application of a right preconditioning substitution. This idea is extended to produce a dimensionality reduction technique that combines a random left sketch and a right preconditioning substitution, denoted the randomized preconditioning sketch (RPS). This method is shown to produce a viable solution, without many of the limitations of previous methods presented, and numerical results verify the stability of the solution when applied to inversion of `gravity` and `magnetic` data in combination with fast multiplication via the 2DFFT.

A standard approach to obtaining a useful solution is reducing the computational cost by using an approximate SVD. Randomization is applied in Chapter 5 to significantly reduce the computational cost of obtaining an approximate SVD, namely for inversion of `gravity` and `magnetic` data. Computational comparisons are demonstrated for both real and synthetic data.

Chapter 6 contains other applications for the techniques developed, along with other directions for dimensionality reduction of large-scale problems, as well as final conclusion and direction for future research. These include alternate applications of randomization such as applying an approximate pseudo-inverse and obtaining an approximate Kronecker product decomposition. A practical application of the RPS method to a nonlinear transient groundwater transmissivity problem is also presented. It is shown that RPS improves on the accuracy of the solution compared to the randomized geostatistical approach (RGA), which only implements a random left sketch. Section 6.6 provides conclusions and directions for further research. A more detailed outline of these chapters follows.

Outline: Basic linear algebra concepts and inversion techniques are introduced in Chapter 2. Discretization is discussed in Section 2.1, ill-posedness in Section 2.2, and noise in Section 2.3. Regularization techniques are shown in Section 2.4, with truncation in Section 2.4.1, Tikhonov regularization in Section 2.4.2, and general p-norm regularization in Section 2.4.3. The large-scale 3D `gravity` and `magnetic` inversion problems are introduced in Section 2.5.

In Chapter 3 `gravity` and `magnetic` kernels are discussed. Fast multiplication using a circulant extension is introduced in Section 3.1. Matrix structure resulting from spatially invariant kernels is addressed in Section 3.2 with symmetric kernels developed without domain padding in Section 3.2.3, and non-symmetric kernels in Section 3.2.4. Padding is introduced around the domain in Section 3.3, with matrix structure resulting from including domain padding for symmetric kernels discussed in Section 3.3.1, and non-symmetric kernels in Section 3.3.2. Fast multiplication is extended for a padded domain in Section 3.3.3. Optimizations are discussed for generating the kernel matrices in Section 3.4, specifically for the `gravity` kernel in Section 3.4.1 and for the `magnetic` kernel in Section 3.4.2. Numerical results validate the fast multiplication techniques presented in Section 3.5, and available software resources are provided in Section 3.5.1.

Chapter 4 combines the fast multiplication techniques with randomization for inversion of `gravity` and `magnetic` data. Dimensionality reduction is introduced in Section 4.1 via a randomized sketch, with sketching applied directly to a system in Section 4.2. A left sketch applied directly to a system is presented in Section 4.2.1, with a naive application of a right sketch via a substitution presented in Section 4.2.2. Preconditioning and LSRN are presented in Section 4.2.3. A novel randomized preconditioning sketch (RPS) is introduced in Section 4.3, and error analysis is given. Computational analysis is given in Section 4.4. Numerical experiments are presented in Section 4.5, with specifics relating to inversion of `gravity` and `magnetic` data presented in Section 4.5.1, implementation details given in Section 4.5.2, implementation of the synthetic model in Section 4.5.4, and results are presented in Section 4.5.5. Inversion of real-world `magnetic` data is implemented via truncation in Section 4.5.9.

The inversion of `gravity` and `magnetic` data via Tikhonov regularization combined with an approximate SVD is discussed in Chapter 5. Applying randomization to obtain an approximate SVD is presented in Section 5.1. Previously introduced methodology is used to establish a basis for numerical results in Section 5.1.1, with specific algorithmic details given in Section 5.1.2. Computational costs are examined in Section 5.1.3, and numerical experimentation is presented in Section 5.2 with results shown in Section 5.2.2. Inversion of real-world `magnetic` data is implemented via Tikhonov regularization combined with an approximate SVD in Section 5.2.6. Conclusions are drawn in Section 5.3.

Alternative directions for dimensionality reduction are presented in Chapter 6, starting with alternative sketching techniques in Sections 6.1-6.2. A method to obtain an approximate pseudo-inverse is presented in Section 6.1, with an updated iteratively applied pseudo-inverse proposed in Section 6.1.1. An alternative right sketch based on linear interpolation is discussed in Section 6.2. A Kronecker product decomposition is discussed in Section 6.3, with a novel sketch applied to the Kronecker product decomposition and analysis given in

Section 6.3.1. The novel sketching technique is extended and analyzed for a decomposition containing a sum of Kronecker products in Section 6.3.2. Implementation of the Kronecker product decomposition to blocks within a matrix is presented in Section 6.3.3. A nonlinear $2D$ groundwater transmissivity test problem is presented in Section 6.4. The principal component geostatistical approach is presented in Section 6.4.1, and the randomized geostatistical approach is discussed in Section 6.4.2. The randomized preconditioning sketch is applied to groundwater transmissivity in Section 6.4.3, and numerical results are presented in Section 6.4.4. Final conclusions and future work are discussed in Section 6.6.

Necessary mathematical identities for the pseudo-inverse and singular value decomposition (SVD) are provided in Appendix A. Useful referenced information is provided in Appendix B.

Note that MATLAB style notation is used throughout. Specifically, a $\cdot$ before an operation denotes element-wise operations. Further, for column vector $\boldsymbol{x} \in \mathbb{R}^m$ and row vector $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{x} \cdot * \boldsymbol{y} = [y_1\boldsymbol{x}, y_2\boldsymbol{x}, \ldots, y_n\boldsymbol{x}]$, and $\boldsymbol{x} + \boldsymbol{y} = [y_1 + \boldsymbol{x}, y_2 + \boldsymbol{x}, \ldots, y_n + \boldsymbol{x}]$. MATLAB notation is also used to prescribe a set of values ranging from a to b with step size sz as $a : sz : b$, where sz can be negative.

Chapter 2

TECHNIQUES FOR INVERSION

Solution of discrete inverse problems is of interest in many applications. Here, background information is presented on inverse problems. Specifically, discrete systems are introduced in Section 2.1. Ill-posed problems and the discrete Picard condition are discussed in Section 2.2. Noise contamination and regularization methods are presented in Sections 2.3 and 2.4 respectively. Computational limitations when the scale becomes large are motivated by the description of large-scale 3D problems for inversion of `gravity` and `magnetic` data in Section 2.5.

2.1 Discretization

Consider as an example the solution of a Fredholm integral equation of the first kind,

$$\int_{\Omega_t} H(s,t)g(t)dt = f(s), \tag{2.1}$$

where $t \in \Omega_t$, $s \in \Omega_s$, the square integrable kernel H and the continuous right hand side f are known, and continuous g is unknown. This integral equation can be discretized using the trapezoidal rule over discrete points $1 \leq j \leq m$, $1 \leq i \leq n$ and constant $\Delta t_j = t_j - t_{j-1}$ as

$$\int_{\Omega_t} H(s_i,t)g(t)dt \approx \sum_{j=2}^{n} \Delta t_j \left(\frac{H(s_i,t_{j-1})g(t_{j-1}) + H(s_i,t_j)g(t_j)}{2} \right) \tag{2.2}$$

for each i. Combining like terms, with $a_{i,j} = \Delta t_j H(s_i,t_j)$, $x_j = g(t_j)$ and sampled values $b_i = f(s_i)$, provides the discrete system

$$A\boldsymbol{x} \approx \boldsymbol{b}.$$

The first and last columns of A may be scaled by $\frac{1}{2}$ to account for the first and last terms of (2.2), depending on boundary conditions.

2.2 Ill-posed Inverse Problems and the Discrete Picard Condition

The solution of the discrete problem $A\boldsymbol{x} \approx \boldsymbol{b}$ is often ill-posed; small errors in the data can be amplified, creating large errors in the solution. Using the standard 2-norm $\|A\|_2 = \sup_{\|\boldsymbol{x}\|_2=1} \|A\boldsymbol{x}\|_2$, the condition number of A, $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$ for invertible A, is large for ill-posed problems, Hansen [1994].

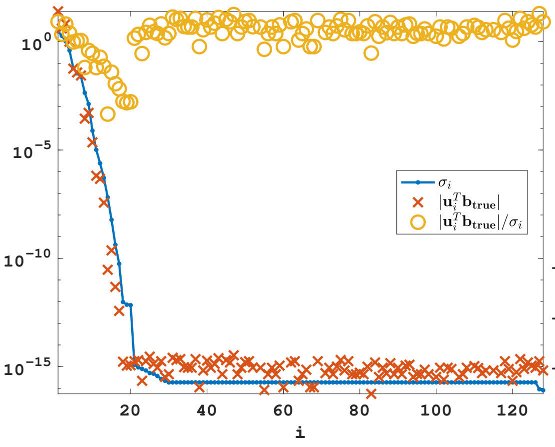
The singular value decomposition (SVD) $A = U\Sigma V^T = \sum_i \boldsymbol{u}_i\sigma_i\boldsymbol{v}_i^T$ is useful for the analysis of ill-conditioned problems, where U and V are orthonormal, Σ is diagonal with elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, and A is rank r. Given $\|A\|_2 = \sigma_1$, then $\kappa_2(A) = \sigma_1/\sigma_r$. Thus, when the condition number is large, there is a large gap between the largest and smallest singular values, and the small singular values are more susceptible to round-off errors.

Regardless of rank, the naive solution can be written in terms of the pseudo-inverse as $\boldsymbol{x} = A^\dagger\boldsymbol{b} = \sum_{\sigma_i\neq 0} \boldsymbol{v}_i\sigma_i^{-1}\boldsymbol{u}_i^T\boldsymbol{b} = \sum_{\sigma_i\neq 0} \frac{\boldsymbol{u}_i^T\boldsymbol{b}}{\sigma_i}\boldsymbol{v}_i$. The pseudo-inverse of A is given by $A^\dagger = V\Sigma^\dagger U^T$, where $\Sigma^\dagger$ has diagonal elements $\sigma_i^\dagger = \sigma_i^{-1}$ if $\sigma_i \neq 0$ and $\sigma_i^\dagger = 0$ if $\sigma_i = 0$. The **discrete Picard condition**, Hansen [1994], is satisfied if the absolute values of coefficients $|\boldsymbol{u}_i^T\boldsymbol{b}|$ decay faster than the singular values. Otherwise, terms that relate to small singular values become over weighted, amplifying small errors.

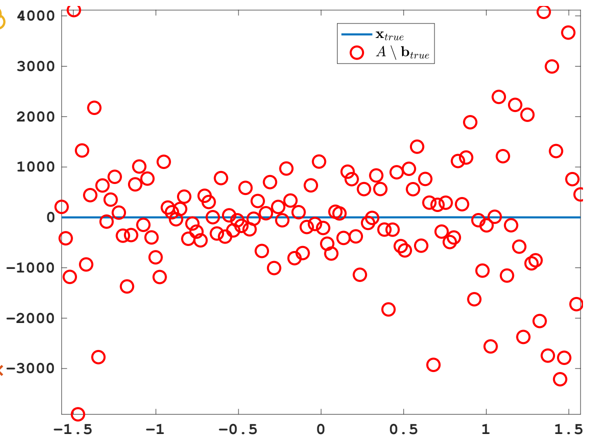
Regularization Tools, Hansen [1994], provides many simple inverse problems in MATLAB, which can be used to illustrate properties of ill-posed problems. For example, problem **shaw** is a one-dimensional model of an image restoration problem defined by the kernel

$$H(s,t) = (\cos(t) + \cos(s))^2 \left(\frac{\sin(\pi(\sin(s) + \sin(t)))}{\pi(\sin(s) + \sin(t))}\right)^2,$$

for $s, t \in [-\pi/2, \pi/2]$. The SVD for matrix A obtained from the discretization of H with $m = n = 128$ is used. The ratios $\frac{|\boldsymbol{u}_i^T\boldsymbol{b}_{\text{true}}|}{\sigma_i}$ are shown in Figure 2.1, with the clean right hand side as $\boldsymbol{b}_{\text{true}}$. It can be seen that the Picard condition fails to be satisfied after approximately 20 terms due to numerical precision. The naive solution reaches magnitudes of approximately $4\text{e}3$, whereas the true solution has an approximate maximum value of 2.



(a) Picard plot



(b) naive solution

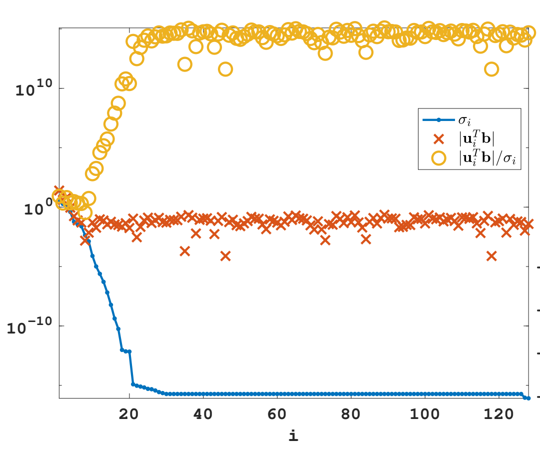
Figure 2.1: The computed scalars $|\boldsymbol{u}_i^T\boldsymbol{b}|/\sigma_i$ are plotted versus i for **shaw** with $n = 128$ for $\boldsymbol{b}_{\text{true}}$ in Figure 2.1a, and the related naive solution using the MATLAB backslash operator is shown in Figure 2.1b. The Picard condition holds approximately for the first 20 components in Figure 2.1a.

2.3 Noise

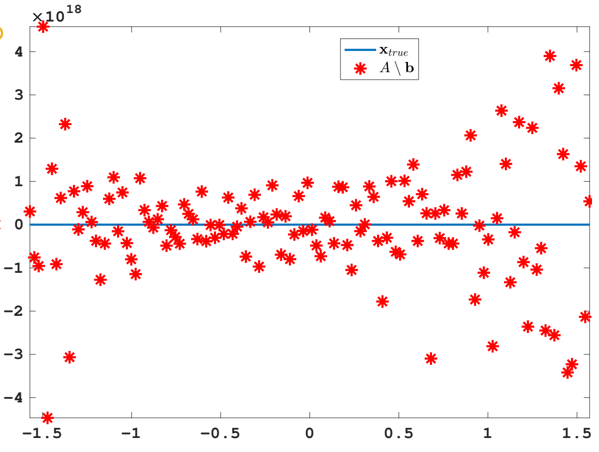
Under the assumption that $\boldsymbol{b}$ is measured data, it is generally contaminated by noise. To simulate contaminated data, normalized white noise is added to $\boldsymbol{b}_{true}$ as

$$\boldsymbol{b} = \boldsymbol{b}_{true} + \eta\frac{\boldsymbol{b}_{true}}{\|\boldsymbol{b}_{true}\|_2} \cdot *\textbf{randn}(n, 1),$$

where $\cdot*$ represents element-wise multiplication. Again using the SVD of A for problem **shaw** with $m = n = 128$ and noise level $\eta = 1$, the Picard condition is violated after approximately the first 8 terms, see Figure 2.2a. The naive solution now reaches an approximate magnitude of $4\text{e}18$, Figure 2.2b. See Figure 2.3 for a comparison of $\boldsymbol{b}_{true}$ and $\boldsymbol{b}$. Since calculating the singular values is limited by machine precision, and any noise in the data can contaminate the Fourier coefficients $\boldsymbol{u}_i^T\boldsymbol{b}$, accurately approximating the full rank of the system is not typically possible. Instead, the numerical rank is the number of accurate singular components. The numerical rank, k, can be estimated by the first 1 through k components for which the Picard condition approximately holds.



(a) Picard plot



(b) naive solution

Figure 2.2: The computed scalars $|\boldsymbol{u}_i^T\boldsymbol{b}|/\sigma_i$ are plotted versus i for **shaw** with $n = 128$ and noise level $\eta = 1$ in Figure 2.2a, and the related naive solution using the MATLAB backslash operator is shown in Figure 2.2b. The Picard condition holds approximately for the first 8 components in Figure 2.2a.

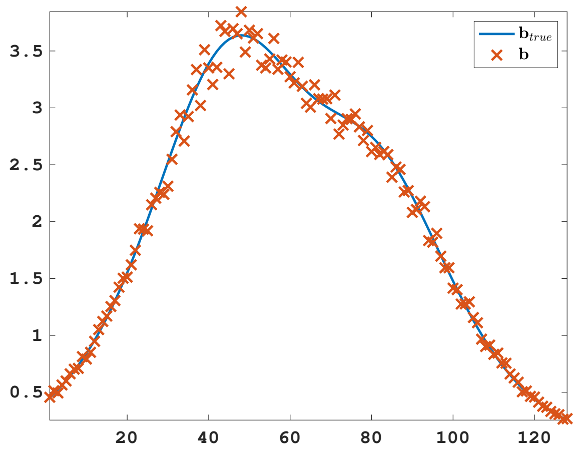


Figure 2.3: $\boldsymbol{b}_{\text{true}}$ and $\boldsymbol{b}$ are shown for **shaw** with $n = 128$ and noise level $\eta = 1$.

2.4 Regularization

2.4.1 Truncation

Solving a linear ill-posed inverse problem accurately requires some form of regularization, since terms for which the Picard condition is violated become overweighted. The simplest regularization method takes the first k terms of the naive solution producing

$$\boldsymbol{x}_k = \sum_{i=1}^{k} \frac{\boldsymbol{u}_i^T \boldsymbol{b}}{\sigma_i} \boldsymbol{v}_i = \sum_{i=1}^{r} \phi_i \frac{\boldsymbol{u}_i^T \boldsymbol{b}}{\sigma_i} \boldsymbol{v}_i, \tag{2.3}$$

where $\phi_i = 1$ for $i \leq k$ and $\phi_i = 0$ for $i > k$.

The truncation parameter k approximates the numerical rank, and is optimized when only the terms with singular values dominated by noise are discarded. Generalized cross validation (GCV), Hansen [1994], is a widely accepted technique for estimating the optimal regularization parameter k. The optimal k is found by minimizing

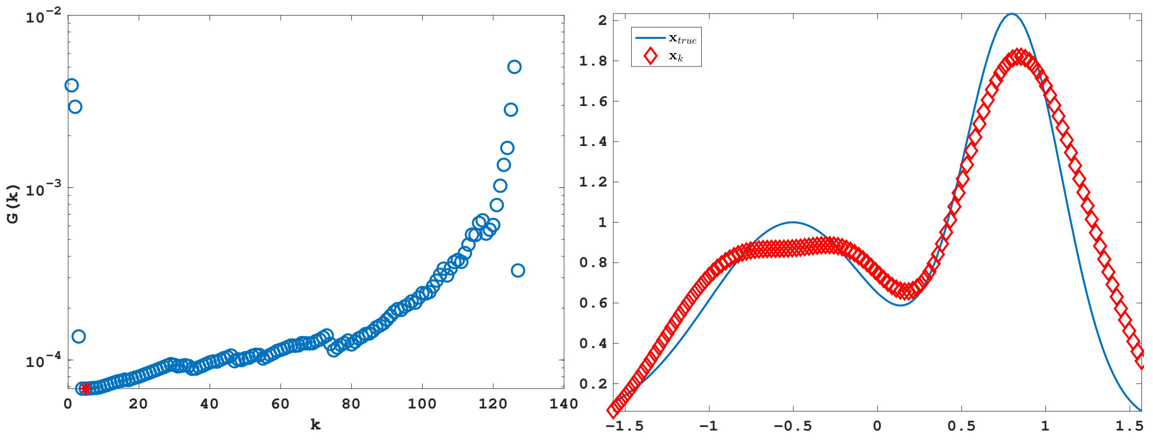
$$G(k) = \frac{\|A\mathbf{x}_k - \mathbf{b}\|_2^2}{(trace(\mathrm{I} - AA_k^\dagger))^2},$$

where $A_k^\dagger = \sum_{i=1}^{k} \boldsymbol{u}_i^T \sigma_i^\dagger \boldsymbol{v}_i$, and is used here for determining the optimal regularization parameter k for the truncation approach. The regularization parameter obtained through GCV often approximately enforces the numerical rank accurately. GCV, however, will also occasionally fail to produce a viable regularization parameter. Figure 2.4 shows the GCV function with the obtained truncation parameter k for **shaw** with $m = n = 128$ and $\eta = 1$ in Figure 2.4a along with the computed solution $\boldsymbol{x}_k$ versus the true solution $\boldsymbol{x}_{\text{true}}$ in Figure 2.4b.

2.4.2 Tikhonov Regularization

Standard Tikhonov regularization, Hansen [1994], defines $\boldsymbol{x}$ as the minimizer of

$$\frac{1}{2}\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \frac{\alpha^2}{2}\|\boldsymbol{x}\|_2^2.$$



(a) GCV function (b) truncated solution

Figure 2.4: The GCV function is plotted versus choice of truncation parameter k for **shaw** with $n = 128$ and noise level $\eta = 1$ in Figure 2.4a, and the related truncated solution $\boldsymbol{x}_k$ is shown along with the true solution $\boldsymbol{x}_{\text{true}}$ in Figure 2.4b.

The regularization parameter α weights the contributions between the two terms. Differentiating with respect to $\boldsymbol{x}$ shows that $\boldsymbol{x}$ is obtained as the solution of the normal equations:

$$\begin{aligned} 0 &= \frac{\partial}{\partial \boldsymbol{x}} \left(\frac{1}{2} \|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \frac{\alpha^2}{2} \|\boldsymbol{x}\|_2^2 \right) \\ &= \frac{\partial}{\partial \boldsymbol{x}} \left(\frac{1}{2} (A\boldsymbol{x} - \boldsymbol{b})^T (A\boldsymbol{x} - \boldsymbol{b}) + \frac{\alpha^2}{2} \boldsymbol{x}^T \boldsymbol{x} \right) \\ &= \frac{1}{2} \frac{\partial}{\partial \boldsymbol{x}} \left(\boldsymbol{x}^T A^T A \boldsymbol{x} - \boldsymbol{b}^T A \boldsymbol{x} - \boldsymbol{x}^T A^T \boldsymbol{b} + \boldsymbol{b}^T \boldsymbol{b} + \alpha^2 \boldsymbol{x}^T \boldsymbol{x} \right) \\ &= A^T A \boldsymbol{x} - A^T \boldsymbol{b} + \alpha^2 \boldsymbol{x}. \end{aligned}$$

The optimal solution is $\boldsymbol{x} = (A^T A + \alpha^2 \mathrm{I})^{-1} A^T \boldsymbol{b}$, ie $\boldsymbol{x}$ solves the normal equations for the augmented system

$$\begin{bmatrix} A \\ \alpha \mathrm{I} \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{0} \end{bmatrix}.$$

Notice that

$$\begin{aligned}(A^TA+\alpha^2\mathbf{I})^\dagger A^T &= (V\Sigma^2V^T+\alpha^2VV^T)^\dagger V\Sigma U^T\\ &= V(\Sigma^2+\alpha^2\mathbf{I})^\dagger\Sigma U^T\\ &= V(\Sigma^2+\alpha^2\mathbf{I})^\dagger\Sigma^2\Sigma^\dagger U^T\\ &= V\Phi\Sigma^\dagger U^T,\end{aligned}$$

where now $\Phi := (\Sigma^2+\alpha^2\mathbf{I})^\dagger\Sigma^2$. Filter factors $\phi_i \in [0,1]$ depend on regularization parameter α, where ϕ_i are diagonal elements $\Phi_{i,i} = \phi_i = \sigma_i^2/(\sigma_i^2+\alpha^2)$.

Define $A_\alpha^\dagger := V\Phi\Sigma^\dagger U^T$. Then

$$\boldsymbol{x}_\alpha := \sum_{\sigma_i\neq 0}\phi_i\frac{\boldsymbol{u}_i^T\boldsymbol{b}}{\sigma_i}\boldsymbol{v}_i = V\Phi\Sigma^\dagger U^T\boldsymbol{b} = A_\alpha^\dagger\boldsymbol{b}. \tag{2.4}$$

The optimal filter factors transition from $\phi_i \approx 1$ for $i < k$ to $\phi_i \approx 0$ for $i > k$ for approximate numerical rank k. Notice that $\phi_i = 1$ for $i \leq k$ and $\phi_i = 0$ for $i > k$ produces the truncation approach given in (2.3). Figure 2.5 illustrates the optimal filter factors obtained through GCV for a truncation approach versus Tikhonov regularization.

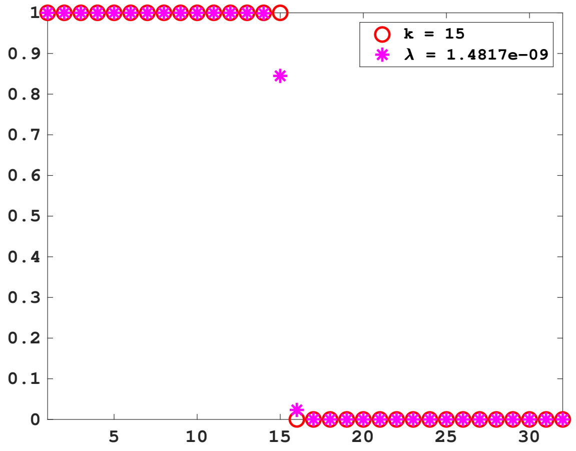


Figure 2.5: Filter factors are shown for **shaw** with $n = 32$ and noise level $\eta = 1\text{e}{-3}$, based on the optimal truncation parameter k for a truncation approach, and also for filter factors dependent on regularization parameter α for Tikhonov regularization.

Setting $\alpha = 0$ produces the standard least squares (LS) solution $\boldsymbol{x} = (A^TA)^{-1}A^T\boldsymbol{b}$ if (A^TA) has full rank. When A is underdetermined, A^TA is guaranteed to be singular, and the substitution $\boldsymbol{x} = A^T\boldsymbol{y}$ is implemented instead. Then $\boldsymbol{y} = (AA^T)^{-1}\boldsymbol{b}$, provided A has full rank, resulting in the solution $\boldsymbol{x} = A^T(AA^T)^{-1}\boldsymbol{b}$. In general, the two solutions are related by the pseudo-inverse $A^\dagger = V\Sigma^\dagger U^T$ since

$$A^T(AA^T)^\dagger = V\Sigma U^T(U\Sigma\Sigma^T U^T)^\dagger = V\Sigma U^T(U^T)^{-1}(\Sigma\Sigma^T)^\dagger(U)^{-1} = V\Sigma^\dagger U^T$$

and

$$(A^TA)^\dagger A^T = (V\Sigma^T\Sigma V^T)^\dagger V\Sigma U^T = (V^T)^{-1}(\Sigma^T\Sigma)^\dagger(V)^{-1}V\Sigma U^T = V\Sigma^\dagger U^T.$$

It has also been shown that implementing a positive definite diagonal weighting matrix W produces the weighted LS problem,

$$\frac{1}{2}\left\|W^{1/2}(A\boldsymbol{x}-\boldsymbol{b})\right\|_2^2. \tag{2.5}$$

which is minimized by the solution of the weighted normal equations $A^TWA\boldsymbol{x} = A^TW\boldsymbol{b}$.

Standard Tikhonov regularization can also be extended to impose properties on the solution by incorporating structured matrix L and minimizing

$$\frac{1}{2}\|A\boldsymbol{x}-\boldsymbol{b}\|_2^2 + \frac{\alpha^2}{2}\|L\boldsymbol{x}\|_2^2$$

where now $(A^TA + \alpha^2 L^TL)\boldsymbol{x} = A^T\boldsymbol{b}$. If L is invertible and $\mathcal{N}(A)\cap\mathcal{N}(L) = 0$, then $(L^T)^{-1}(A^TA+\alpha^2L^TL)L^{-1}L\boldsymbol{x} = ((L^T)^{-1}A^TAL^{-1}+\alpha^2\mathrm{I})L\boldsymbol{x} = (L^T)^{-1}A^T\boldsymbol{b}$, which now fits the standard Tikhonov regularization for $\tilde{A} = AL^{-1}$ with solution $\boldsymbol{x} = L^{-1}(\tilde{A}^T\tilde{A} + \alpha^2\mathrm{I})^{-1}\tilde{A}^T\boldsymbol{b}$. Note that when L is not invertible, an analytic solution can be obtained using a generalized SVD, Golub and Van Loan [2013], which is not considered here.

Tikhonov regularization is easy to implement since it provides a unique solution for each α, but depends on finding an optimal α. Although there are many methods to determine an appropriate regularization parameter, the focus here is on the unbiased predictive risk estimator (UPRE), Vatankhah et al. [2017], Vogel [2002]. Suppose that the noise in the data has variance $\sigma = 1$, then the UPRE minimizes

$$U(\alpha) = \left\| \left(AA_\alpha^\dagger - \mathrm{I}\right) \boldsymbol{b} \right\|_2^2 + 2\mathrm{trace}\left(AA_\alpha^\dagger\right) - m, \tag{2.6}$$

providing an estimate on how well the regularized solution predicts $\boldsymbol{b}_{\text{true}}$. In general, weighting $W = C^{-1}$, based on the covariance C in noise in (2.5), whitens the Gaussian noise in the data, and (2.6) can be applied.

2.4.3 Minimum p-Norm

Minimization using a 2-norm produces a smooth solution, which can lead to over-smoothing. Other choices of norm are often used based on desired characteristics. For example, the 1-norm tends to be better at edge detection than the 2-norm. In general, minimization can be implemented via q and p-norms by minimizing, Rodriguez and Wohlberg [2006],

$$\frac{1}{q} \left\| A\boldsymbol{x} - \boldsymbol{b} \right\|_q^q + \frac{\alpha^2}{p} \left\| \boldsymbol{x} \right\|_p^p. \tag{2.7}$$

Whereas the 2-norm on both terms, $q = p = 2$, provides an analytic solution, this is not available for general p and q. Many methods exist for the solution of this mixed p-q problem given by (2.7). Here the focus is on one approach that recasts (2.7) as a 2-norm problem.

The iteratively reweighted LS (IRLS) method, Rodriguez and Wohlberg [2006], iteratively applies a weighted norm such that $\|\boldsymbol{x}\|_p^p \approx \left\|W^{1/2}x\right\|_2^2$. Specifically,

$$\|\boldsymbol{x}\|_p^p = \sum_{i=1}^m |x_i|^p \approx \sum_{i=1}^m \left(\frac{x_i}{(x_i^2 + \epsilon^2)^{(2-q)/4}} \right)^2 = \|W\boldsymbol{x}\|_2^2,$$

where $w_i = (x_i^2 + \epsilon^2)^{-(2-q)/4}$ are the diagonal elements of W and $\epsilon \ll 1$ is chosen to avoid dividing by values close to zero. Since W is based on the unknown solution, either an initial guess for $\boldsymbol{x}$ is used to produce an estimate $W^{(1)}$, or $W^{(1)} = \mathrm{I}$ is used. $W^{(k)}$ is updated iteratively until a stopping condition is met, ie $\|\boldsymbol{x}\|_p^p \approx \|W(\boldsymbol{x}^{(k-1)})\boldsymbol{x}^{(k)}\|_2^2$ for iteration k.

2.5 Large-Scale Inversion for 3D Problems

Obtaining a solution for an inverse problem can become prohibitively expensive as the size of the problem increases. For example, consider the 3D problem associated with the inversion of `gravity` and `magnetic` data. The 3D Fredholm integral equation of the first kind, which describes these two models, is given by

$$d(a,b,c) = \int\int\int h(a,b,c,x,y,z)\zeta(x,y,z)dx\,dy\,dz. \tag{2.8}$$

Here, consistent with geophysics literature, the discrete forward model is denoted by $\mathbf{d} = G\mathbf{m}$. Thus, for results specific to `gravity` and `magnetic` the general system is replaced by $\mathbf{d} = G\mathbf{m}$. The data measurements for $d(a,b,c)$ are made on the surface with $c = 0$, not necessarily at uniformly-spaced station locations denoted by

$$s_{ij} = (a_{ij}, b_{ij}),\ 1 \le i \le s_x,\ 1 \le j \le s_y. \tag{2.9}$$

The total number of stations at the surface is $m = s_x s_y$. The volume domain is discretized into $n = s_x s_y n_z$ uniform prisms, c_{pqr}, with coordinates

$$\begin{aligned} x_{p-1} &= (p-1)\Delta_x \quad x_p = p\Delta_x, \quad 1 \le p \le s_x, \\ y_{q-1} &= (q-1)\Delta_y \quad y_q = q\Delta_y, \quad 1 \le q \le s_y, \\ z_{r-1} &= (r-1)\Delta_z \quad z_r = r\Delta_z, \quad 1 \le r \le n_z. \end{aligned} \tag{2.10}$$

The geometry is illustrated in Figure 2.6, in which the configuration of station at location (i,j) relative to volume prism pqr is shown. Here the blocks in the z-direction define depth $z \ge 0$ pointing down, and it is assumed there is one station located above each prism, so that there are s_x and s_y blocks in the x and y-directions, respectively.

The entries in G depend on the integral of kernel h and correspond to the unit contribution from a given prism to a particular station. The ordering of the entries depends on the organization of the volume domain into a vector of length, $n = s_x s_y n_z$. The multilayer

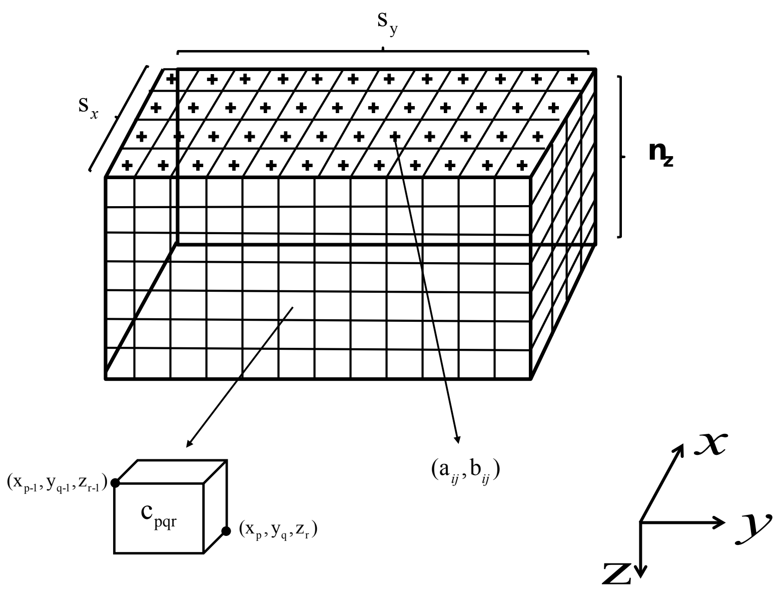


Figure 2.6: The configuration of prism pqr in the volume relative to a station on the surface at location $s_{ij} = (a_{ij}, b_{ij})$. This shows that the stations are located on the surface of the physical domain.

model, in which the volume is organized by slices in depth, yields

$$G = [G^{(1)}, G^{(2)}, \dots, G^{(n_z)}]. \tag{2.11}$$

Each $G^{(r)}$ has size $s_x s_y \times n_x n_y = m \times n_r$, where there are n_r prisms in slice r, and maps from the prisms in depth slice r, with depth coordinates z_{r-1} and z_r, to the station measurements. Further, $G^{(r)}$ decomposes as a block matrix with block entries $G^{(r)}_{jq}$, $1 \leq j \leq s_y$, $1 \leq q \leq n_y$ each of size $s_x \times n_x$. Equivalently, this means that a given slice of the volume with $s_y n_y$ blocks is mapped to a one dimensional vector using **row-major** ordering; the prisms in the slice are swept through for increasing x and fixed y direction. Entry $(G^{(r)})_{k\ell}$, $1 \leq k \leq s_x s_y$, $1 \leq \ell \leq n_x n_y$ represents the contribution from the prism at location $\ell = (q-1)n_x + p$, for $1 \leq q \leq n_y$ and $1 \leq p \leq n_x$, for depth slice r, to the station at $k = (j-1)s_x + i$, $1 \leq j \leq s_y$, $1 \leq i \leq s_x$. Using $\tilde{h}(s_{ij})_{pqr}$ to denote the function that calculates the contribution to station s_{ij} from prism c_{pqr}, then

$$(G^{(r)})_{k\ell} = \tilde{h}(s_{ij})_{pqr}, \; k = (j-1)s_x + i, \; \ell = (q-1)n_x + p. \tag{2.12}$$

Assuming that the discussion is applied for slice r, the dependence of $\tilde{h}$ on depth is removed and $\tilde{h}(s_{ij})_{pq}$ is used to indicate the contribution to station $s_{ij} = (a_{ij}, b_{ij})$ due to block number p in x and q in y. Further, while the discussion is applied under the assumption of a uniform depth interval, Δ_z, the approach applies equally well for problems in which the multilayer coordinate grid has layers of different depths, see e.g. Zhang and Wong [2015]. Further discussion on the calculation of $\tilde{h}$ for the `gravity` and `magnetic` models is given in Chapter 3. The notation used for the solution of `gravity` and `magnetic` problems is provided in Table 2.1.

2.5.1 Computation Limitations for Large-scale 3D Problems

The kernel sizes for `gravity` and `magnetic` problems are based on number of stations in the x direction, number of stations in the y direction, and number of layers of depth, namely n_x, n_y, and n_z respectively. The number of values in the kernel is $(n_x n_y)^2 n_z$. Thus, for $n_x = 100$ and $n_y = 50$ with $n_z = 10$, there are $2.5\text{e}8$ elements in the kernel matrix. Assuming 8 bytes per number, this requires 2GB of memory.

In addition, calculating the SVD requires $\mathcal{O}\left(n_x n_y (n_x n_y n_z)^2\right)$ floating-point operations (flops), Golub and Van Loan [2013]. Using the same example, the SVD would require $\mathcal{O}\left(1.25\text{e}13\right)$ flops. In contrast, matrix-vector multiplication would require approximately $2(n_x n_y)^2 n_z = 5\text{e}8$ flops. Thus, for large-scale problems, mitigating or avoiding the cost of directly calculating the SVD is highly desirable.

Doubling the number of grid points in the y direction produces $n_x = n_y = 100$ and $n_z = 10$, with 10^9 elements in the kernel, requiring 8GB of memory. The SVD now requires $\mathcal{O}\left(10^{14}\right)$ flops, whereas matrix-vector multiplication requires approximately $2\text{e}9$

Table 2.1: Notation Adopted in the Discussion

s_x	# true stations in x
$s_{ij} = (a_{ij}, b_{ij})$	Station location
s_y	# true stations in y
$m = s_x s_y$	# measurements
n_x, n_y, n_z	# coordinate blocks in x,y,z
$\Delta_x, \Delta_y, \Delta_z$	Grid sizes in x,y,z
$p_{x_\mathrm{L}}, p_{x_\mathrm{R}}, p_x$	Left, right, total padding: x
n_x	$n_x = s_x + p_x$
$p_{y_\mathrm{L}}, p_{y_\mathrm{R}}, p_y$	Left, right,total, padding: y
n_y	$n_y = s_y + p_y$
x_p	$x_p = (p - 1 - p_{x_\mathrm{L}})\Delta_x$, $1 \leq p \leq n_x + 1$
$n = n_x n_y n_z$	Volume Dimension
y_q	$y_q = (q - 1 - p_{y_\mathrm{L}})\Delta_y$, $1 \leq q \leq n_y + 1$
$n_r = n_x n_y$	Layer Dimension
z_r	$z_r = (r - 1)\Delta_z$, $1 \leq r \leq n_z + 1$
c_{pqr}	Prism pqr in xyz
d, h, ζ	Forward Model, see (2.8)
$\tilde{h}(s_{ij})_{pqr}$	Projection c_{pqr} to s_{ij}
$G \in \mathcal{R}^{m \times n}$	$(G^{(r)})_{k\ell} = \tilde{h}(s_{ij})_{pqr}$, see (2.12)
$G^{(r)} \in \mathcal{R}^{m \times n_r}$	Depth r Contribution
$G_q^{(r)} \in \mathcal{R}^{s_x \times n_x}$	$G_q^{(r)} = G_{1q}^{(r)}$, $1 \leq q \leq n_y$
$\bar{G}_j^{(r)} \in \mathcal{R}^{s_x \times n_x}$	$\bar{G}_j^{(r)} = \bar{G}_{j1}^{(r)}$, $1 \leq j \leq s_y$
$\mathbf{c}_q$, $\mathbf{r}_q$	$G_q^{(r)} = \textbf{toeplitz}(\mathbf{c}_q, \mathbf{r}_q)$, see (3.12)
$\bar{\mathbf{c}}_j$, $\bar{\mathbf{r}}_j$	$\bar{G}_j^{(r)} = \textbf{toeplitz}(\bar{\mathbf{c}}_j, \bar{\mathbf{r}}_j)$, see (3.21)
BTTB	Block Toeplitz Toeplitz blocks
symBTTB	Symmetric BTTB
BCCB	Block Circulant Circulant blocks
J_m	Exchange matrix, Defn. 3.1.1
$\mathbf{c}_q^{\mathrm{ext}}$, $\mathbf{r}_q^{\mathrm{ext}}$	Defining $(G^{(r)})^{\mathrm{circ}}$
$\bar{\mathbf{c}}_j^{\mathrm{ext}}$, $\bar{\mathbf{r}}_j^{\mathrm{ext}}$	Defining $(\bar{G}^{(r)})^{\mathrm{circ}}$
T	Components of BCCB, see (3.37)
$\hat{T}$	$\textbf{fft2}(T)$: 2DFFT

flops. As further grid refinement is desired, this problem quickly becomes intractable in terms of both memory and computations. The remaining focus of this dissertation is on reducing the dimensionality, and thus memory and computational requirements associated with solving large-scale inverse problems.

Techniques to handle these large-scale problems, with respect to storage and computational cost, will also be the focus of later chapters.

2.6 Conclusions

A brief overview of the most relevant aspects of the solution of inverse problems, to be used in later chapters, has been provided. Also of interest, the large-scale 3D `gravity` and `magnetic` problems have been described in general form.

Chapter 3

USING STRUCTURE TO REDUCE COMPUTATIONS OF FORWARD OPERATIONS FOR LARGE-SCALE GRAVITY AND MAGNETIC KERNELS

Overview of Chapter

This chapter provides a general discussion on Toeplitz and circulant matrices and fast Fourier based multiplication. Block structure is taken into account with the extended two-dimensional Fourier based multiplication. This is followed by a detailed tutorial on the efficient generation of sensitivity matrices with BTTB structure and their efficient forward multiplication using the 2DFFT. The approach depends on the underlying convolution kernel that describes the model. This work, therefore, provides a general tool for the efficient simulation of `gravity` and `magnetic` field data, as well as any formulation which admits a model matrix with BTTB structure. It will be shown that the fast convolution multiplication can be extended for rectangular kernel matrices. Results will validate that there is a considerable reduction in computational cost for generation of these matrices, and for forward and transpose operations with these matrices.The algorithm is open source and available at `https://github.com/renautra/FastBTTB`, along with a full description of the algorithm implementation and example simulations at `https://math.la.asu.edu/~rosie/research/bttb.html`.

Fast computation of geophysics kernel models has been considered by a number of authors, including calculation within the Fourier domain as in Li et al. [2018], Pilkington [1997], Zhao et al. [2018], and through discretization of the operator and calculation in the spatial domain as in Chen and Liu [2018], Zhang and Wong [2015]. Pilkington [1997] introduced the use of the Fast Fourier Transform (FFT) for combining the evaluation of the `magnetic` kernel in the Fourier domain with the conjugate gradient method for solving the inverse problem to determine `magnetic` susceptibility from measured `magnetic` field data. Li et al. [2018] considered the use of the Gauss FFT for fast forward modeling of the `magnetic` kernel on an undulated surface, combined with spline interpolation of the surface data. This work focused on the implementation of the model in the wave number domain and only applied the method for forward modeling. The Gauss FFT was also used by Zhao et al. [2018] for the development of a high accuracy forward modeling approach for the `gravity` kernel.

Bruun and Nielsen [2007], and subsequently, Zhang and Wong [2015], introduced the use of the Block-Toeplitz-Toeplitz-Block (BTTB) structure of the modeling sensitivity matrix for fast three-dimensional inversion of three-dimensional `gravity` and `magnetic` data. For a matrix with BTTB structure, it is possible to embed the information within a matrix of Block-Circulant Circulant-Block (BCCB) structure that facilitates fast forward multiplication using a two-dimensional FFT (2DFFT). For three-dimensional modeling, Zhang and Wong [2015] exploited the two-dimensional multi-layer structure of the kernel, that provides BTTB structure for each layer of the domain, and performed the inverse operation iteratively over all layers of the domain. The technique is flexible to depth layers of variable heights, and permits the inclusion of smoothness stabilizers in the inversion, for each layer of the domain. Moreover, Zhang and Wong [2015] adopted the preconditioning of the BTTB matrix using optimal preconditioning operators as presented in Chan and Jin [2007] for implementing efficient and effective solvers for the inversion.

A fast forward modeling of the gravity field was developed by Chen and Liu [2018], using the three-dimensional modeling of the `gravity` kernel as given in e.g. Boulanger and Chouteau [2001], Haáz [1953]. Their work extends the techniques of Zhang and Wong [2015] for taking advantage of the BTTB structure of the sensitivity matrix associated with a single layer of the `gravity` kernel, but offers greater improvements in the implementation of the forward kernel through the presentation of an optimized calculation of the kernel entries in this matrix. This arises due to the observation that the uniform placement of the measurement stations, in relation to the coordinate grid for the unknown densities, yields significant savings in computation and memory. When the stations are on a grid that is uniformly staggered with respect to the coordinate grid on the top surface of the coordinate domain, redundant operations in the calculation of the sensitivity matrix can be eliminated.

A careful derivation of the forward operators for kernels that are spatially invariant in all dimensions, and are thus convolution operators, is presented. With uniform placement of measurement stations relative to the coordinate domain, namely on staggered grids in the x and y dimensions, the resulting discretizations of the first kind Fredolm integral operators, Zhdanov [2002], yield sensitivity matrices that exhibit BTTB structure. Dependent on the model, the matrices are symmetric BTTB (symBTTB) or BTTB but not symmetric. This depends on the kernel operator, the `gravity` kernel, Boulanger and Chouteau [2001], is symmetric in the distances, but the `magnetic` kernel, Rao and Babu [1991] is not. Thus, the generation of the sensitivity matrix requires further analysis for efficient computation in the case of the `magnetic` kernel as compared to the `gravity` kernel. It is demonstrated that it is feasible to develop an optimized calculation of the entries of the `magnetic` kernel matrix, in a manner similar to that used for the `gravity` case, but due to lack of symmetry the memory and computation requirements are increased. Still, remarkable savings in generating the matrix are achieved. While it is not possible to take advantage of BTTB structure when the stations are not on a uniform grid, the calculation of the underlying kernel matrices can still be optimized for arbitrary stations locations, by reuse of common vectors and arrays for each depth layer of the coordinate volume.

Although the BTTB structure for the gravity kernal has been discussed in the literature, the analysis for the `magnetic` kernel is new. Moreover, this work provides both the careful derivation of the matrices that arise, and the associated implementation of the 2DFFT for fast computation of forward multiplications with the matrix and its transpose. This is described in the submitted paper Hogue et al. [2019], where the intent is to provide a user-friendly environment for the development and validation of kernel operators that yield the underlying BTTB structures which facilitate the fast computation of both forward and transpose operations. Before presenting this discussion for the specific kernels, a general introduction to Toeplitz matrices, BTTB matrices, and the use of the 2DFFT is provided.

Overview of main scientific contributions. The approach implements and extends the BTTB algorithm for the forward modeling with the `magnetic` kernel, and for the inclusion of padding around the domain. Specifically, the main contributions are as follows. (i) A detailed derivation of the implementation of the algorithm presented in Chen and Liu [2018] for the forward modeling of the `gravity` problem is given; (ii) The algorithm is extended to include domain padding in x and y directions, that is not necessarily symmetric with respect to the domain and is no longer limited to a square kernel with square blocks; (iii) The use of the 2DFFT for forward multiplication using the transpose matrix, as is required for the solution of the associated inverse problem is demonstrated; (iv) The algorithm, with and without padding, is further extended for matrices that are not symmetric, as occurs for the `magnetic` kernel; (v) Efficient derivation of the underlying operators to be used without the 2DFFT is also provided, so as to facilitate a realistic performance comparison between the use of the 2DFFT for the forward multiplication and a direct forward multiplication without the use of the 2DFFT; (vi) All components of the discussion are coded for optimal memory, storage and computation as an open source MATLAB code which can be adapted for any convolution kernel which generates a BTTB matrix. The user need only provide the algorithm for the computation of the original matrix components.

The chapter is organized as follows. In Section 3.1 a general discussion is given on the formulation that facilitates the use of the 2DFFT, following the discussion of Vogel [2002]. The general kernel-based forward geophysics model presented in Section 2.5, is extended in Section 3.2, specifically in Section 3.2.1 for convolutional kernels as seen for `gravity` and `magnetic` potential fields. In Section 3.2.2, it is demonstrated how the placement of the measurement stations as uniformly staggered with respect to the coordinate domain yields a distance vector for distances from coordinates to stations that is efficiently stored as a one-dimensional instead of two-dimensional vector. It is shown how operators that are spatially invariant yield matrix operators with BTTB structure, for the symmetric case in Section 3.2.3 and the non-symmetric case in Section 3.2.4. This applies also for the case of the introduction of padding around the domain, Section 3.3, for the symmetric case in Section-3.3.1 and the non-symmetric case in Section 3.3.2. In each case the calculation of the relevant entries of the matrices are carefully explained. Specific examples are given in Section 3.4 for the efficient derivation of the entries in the operators for `gravity` and `magnetic` kernels, following Chen and Liu [2018] and Rao and Babu [1991], in Sections 3.4.1 and 3.4.2, respectively. The improved efficiency of forward operations with the matrix and its transpose for these kernels is demonstrated for domains of increasing size. The presented numerical results in Section 3.5 validate that the given algorithms are efficient and facilitate forward modeling for problems that are significantly larger as compared to the case when the BTTB structure is not utilized for fast computation with the 2DFFT. Software availability is discussed in Section 3.5.1.

3.1 Circulant Operators and the 2DFFT

An introduction of the structured matrices that are useful throughout the discussion is provided. First, assume $B \in \mathbb{R}^{m \times m}$ has block structure with blocks $B_{ij} \in \mathbb{R}^{n_x \times n_x}$ for $1 \leq i \leq n_y$ and $1 \leq j \leq n_y$, and thus $m = n_x n_y$. Now assume each block is Toeplitz.

Specifically,

$$B_{ij} = \begin{bmatrix} a_0 & a_1 & \cdots & a_{n_x-1} \\ a_{-1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_1 \\ a_{1-n_x} & \ddots & a_{-1} & a_0 \end{bmatrix} \tag{3.1}$$

is a Toeplitz matrix with $2n_x - 1$ unique elements. Notice that B_{ij} can be defined solely by its first column and row, namely $\mathbf{c}$ and $\mathbf{r}$ respectively. MATLAB style notation **toeplitz**$(\mathbf{c}, \mathbf{r})$ is used to denote this relationship.

Any Toeplitz matrix can be embedded in a circulant extension,

$$E = \left[\begin{array}{cccc|ccc} a_0 & a_1 & \cdots & a_{n_x-1} & a_{1-n_x} & \cdots & a_{-1} \\ a_{-1} & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_1 & \ddots & \ddots & a_{1-n_x} \\ a_{1-n_x} & \ddots & a_{-1} & a_0 & \ddots & \ddots & a_{n_x-1} \\ \hline a_{n_x-1} & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & a_1 \\ a_1 & \cdots & a_{n_x-1} & a_{1-n_x} & \cdots & a_{-1} & a_0 \end{array}\right],$$

where this circulant matrix also has $2n_x - 1$ unique elements, and can be defined by its first column

$$\begin{bmatrix} a_0 & a_{-1} & \cdots & a_{1-n_x} & a_{n_x-1} & \cdots & a_1 \end{bmatrix}^T.$$

Further, assume B has block Toeplitz structure, so B can be written

$$B = \begin{bmatrix} A_0 & A_1 & \cdots & A_{n_y-1} \\ A_{-1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & A_1 \\ A_{1-n_y} & \ddots & A_{-1} & A_0 \end{bmatrix}, \tag{3.2}$$

where each A_i is Toeplitz. This structure is called block Toeplitz with Toeplitz blocks (BTTB), and B now has $2n_y - 1$ unique blocks. Abusing MATLAB notation, denote this relationship by **toeplitz**(C, R), where C and R contain the defining block column and row respectively.

Similar to the circulant extension of B_{ij}, B can be extended to form a $2n_y - 1 \times 2n_y - 1$ block circulant matrix. The defining first column of blocks for the block circulant extension is given by

$$\left[\begin{array}{cccccccc} E_0^T & E_{-1}^T & \cdots & E_{1-n_x}^T & E_{n_x-1}^T & \cdots & E_1^T \end{array}\right]^T, \tag{3.3}$$

where each E_j is a circulant extension of A_j. This produces a matrix E that is block circulant with circulant blocks (BCCB) that has $2n_y - 1$ defining blocks with $2n_x - 1$ unique defining elements in each block. Thus the first column of E contains all of the unique defining elements.

Specifically, the defining first column of each circulant extension from the first defining column of blocks is stored as $T \in \mathbb{R}^{(2n_x-1)\times(2n_y-1)}$, given by

$$T = \left[\begin{array}{ccccccc} E_0\boldsymbol{e}_1 & E_{-1}\boldsymbol{e}_1 & \cdots & E_{1-n_x}\boldsymbol{e}_1 & E_{n_x-1}\boldsymbol{e}_1 & \cdots & E_1\boldsymbol{e}_1 \end{array}\right],$$

where elementary matrix $\boldsymbol{e}_1$ is the first column of the identity matrix and thus $E_j\boldsymbol{e}_1$ is the first column of E_j.

Compact notation that is useful for defining circulant extensions is now introduced, which depends on the exchange matrix.

Definition 3.1.1 (Exchange matrix)**.** *The* ***exchange*** *matrix is the $m \times m$ matrix J_m which is everywhere 0 except for 1's on the principal counter diagonal.*

Given arbitrary vector $\mathbf{x}$ of length m, with entries x_i, $1 \le i \le m$, then $J_m\mathbf{x} = \mathbf{y}$ where $\mathbf{y}_i = \mathbf{x}_{m-i+1}$, namely it is the the vector with the order of the entries reversed. Equivalently, for matrix A with rows $\mathbf{a}_i$, $1 \le i \le m$, then $J_mA = B$ where B is the matrix with rows in reverse order, $\mathbf{b}_i = \mathbf{a}_{m-i+1}$. Further, multiplying on the right reorders the columns in reverse order. Specifically, $J_m^T = J_m$, and thus $\mathbf{y}^T = (J_m\mathbf{x})^T = \mathbf{x}^TJ_m^T = \mathbf{x}^TJ_m$ and the column entries of $\mathbf{y}$ are in reverse order as compared to $\mathbf{x}$. In the same way, AJ_m gives the matrix with the columns in reverse order. In MATLAB the exchange matrix is implemented using the functions **flipud** and **fliplr**, for "up-down" and "left-right", for multiplication with J_m on the left and right, respectively.

Now the defining first column of each E_j is given by

$$\mathbf{c}_j^{\text{ext}} = \begin{bmatrix} \mathbf{c}_j \\ J_{n_x-1}\mathbf{r}_j(2:n_x) \end{bmatrix}. \tag{3.4}$$

While (3.4) can be used as a defining vector to explicitly generate the extensions E_j as circulant matrices, here the intent is to define the vectors that define the extensions but not to generate the extensions. Note that this definition for generating the extension differs from that used in Li et al. [2018], Vogel [2002], Zhang and Wong [2015]; the extra 0 is omitted for convenience. Also, the circulant extension is defined directly instead of performing a series of transformations.

Consider the right shift matrix

$$S_R = \begin{bmatrix} 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & \ddots & 1 & 0 \end{bmatrix}.$$

Then the circulant extension of each block A_j from BTTB matrix B can also be written using defining elements a_k for $k = 1 - n_x, -n_x, \cdots, n_x - 1$ as

$$E_j = \sum_{k=1-n_x}^{n_x-1} a_k S_R^{-k}. \tag{3.5}$$

Similarly, the BCCB extension of B defined by T can be written

$$E = \sum_{j=1-n_y}^{n_y-1} S_R^{-j} \otimes E_j,$$

where the Kronecker product is defined by

$$D \otimes Z = \begin{bmatrix} d_{11}Z & \cdots & d_{1n}Z \\ \vdots & \ddots & \vdots \\ d_{m1}Z & \cdots & d_{mn}Z \end{bmatrix}.$$

Now matrix S_R is useful for analyzing the convolution. Let F be the Fourier matrix, where $F^T = F$, $F^{-1} = F^*$, and $F_{i1} = 1$ for all i. This provides the similarity transform of S_R.

Lemma 3.1.1. *[Vogel, 2002, Lemma 5.15] Let* $\omega = exp(-2\pi\sqrt{-1}/n)$*. Then*

$$S_R = F^* \textbf{\textit{diag}}(1, \omega, \omega^2, \ldots, \omega^{n-1})F.$$

Proof. $F_{ij} = \omega^{ij}/\sqrt{n}$. Consequently,

$$\left[F^* \textbf{diag}(1, \omega, \omega^2, \ldots, \omega^{n-1})F\right]_{ij} = \frac{1}{n}\sum_{k=0}^{n-1} \bar{\omega}^{ik}\omega^k\omega^{jk} = \frac{1}{n}\sum_{k=0}^{n-1} \omega^{(j+1-i)k}.$$

Thus, $(S_R)_{ij} = 1$ if $i = j-1$, and $(S_R)_{ij} = 0$ if $i \neq j-1$. □

Consequently, as shown in Davis [1979], a circulant matrix is diagonalizable by a Fourier matrix.

Theorem 3.1.2. *[Davis, 1979, Theorem 3.2.3] Let* $\Lambda_j = \textbf{\textit{diag}}([\lambda_1, \lambda_2, \ldots, \lambda_n])$*. Then*

$$E_j = F^*\Lambda_j F \tag{3.6}$$

is circulant.

Extending Lemma 3.1.1 to (3.5),

$$FE_jF^* = F(\sum_{k=1}^{2n_x-1} t_{jk}S_R^{(k-1)})F^* = \sum_{k=1}^{2n_x-1} t_{jk}FS_R^{k-1}F^*.$$

Since $S_R^p = (F^*\Lambda_{S_R}F)^p = F^*\Lambda_{S_R}^pF$ for $\Lambda_{S_R} = \textbf{diag}(1, \omega, \omega^2, \ldots, \omega^{n-1})$, it is immediate that FE_jF^* is a diagonalized matrix.

Now rewriting (3.6) yields

$$F E_j \boldsymbol{e}_1 = \Lambda_j F \boldsymbol{e}_1 = \boldsymbol{\lambda}_j,$$

where $\boldsymbol{\lambda}_j \in \mathbb{R}^{n_x}$ contains diagonal entries of Λ_j. Forward multiplication $E_j \boldsymbol{x}$ can then be computed as

$$E_j \boldsymbol{x} = F^* \Lambda_j F \boldsymbol{x} = F^*(\boldsymbol{\lambda_j} \cdot *(F\boldsymbol{x})),$$

where $\cdot *$ signifies element-wise multiplication.

It remains to be seen how this method applies to the full block matrix. But, following Vogel [2002], the 2DFFT can be written in terms of a Kronecker product. Hence, operations using E are accomplished via the 2DFFT. Define $\boldsymbol{x} = \mathbf{vec}(X)$ such that the columns the columns of $X \in \mathbb{R}^{m \times n}$ are stacked to form $\boldsymbol{x} \in \mathbb{R}^{mn}$, let $X = \mathbf{array}(\boldsymbol{x})$ be the inverse operation, and let **fft2**() denote the 2DFFT.

Proposition 3.1.3. *[Vogel, 2002, Proposition 5.30] Given $X \in \mathbb{C}^{n_x \times n_y}$,*

$$\textbf{\textit{fft2}}(X) = \textbf{\textit{array}}((F_{n_y} \otimes F_{n_x})\boldsymbol{x}),$$

where Fourier matrices $F_{n_x} \in \mathbb{C}^{n_x \times n_x}$, $F_{n_y} \in \mathbb{C}^{n_y \times n_y}$.

Then, $\mathbf{array}((F_{n_y} \otimes F_{n_x})\boldsymbol{x}) = F_{n_x} X F_{n_y}^T = F_{n_x} X F_{n_y}$. Now considering BCCB matrix E,

$$(F_{n_y} \otimes F_{n_x}) E (F_{n_y} \otimes F_{n_x})^* = (F_{n_y} \otimes F_{n_x})(\sum_{j=1}^{2n_y - 1} S_R^{j-1} \otimes E_j)(F_{n_y}^* \otimes F_{n_x}^*).$$

Given $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$,

$$\sum_{j=1}^{2n_y-1} (F_{n_y} \otimes F_{n_x})(S_R^{j-1} \otimes E_j)(F_{n_y}^* \otimes F_{n_x}^*) = \sum_{j=1}^{2n_y-1} (F_{n_y} S_R^{j-1} F_{n_y}^*) \otimes (F_{n_x} E_j F_{n_x}^*).$$

Since $(F_{n_y} S_R^{j-1} F_{n_y}^*)$ and $(F_{n_x} E_j F_{n_x}^*)$ have both been shown to be diagonal and the Kronecker product of two diagonal matrices is also diagonal, this reduces to a sum of diagonal matrices. Thus

$$(F_{n_y} \otimes F_{n_x}) E (F_{n_y}^* \otimes F_{n_x}^*) = \Lambda$$

is a diagonal matrix, and BCCB matrix E is diagonalized by a 2DFFT. In addition,

$$(F_{n_y} \otimes F_{n_x})E\boldsymbol{e}_1 = \Lambda(F_{n_y} \otimes F_{n_x})\boldsymbol{e}_1 = \boldsymbol{\lambda},$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{n_x n_y}$ contains the diagonal elements of Λ. Note that $(F_{n_y} \otimes F_{n_x})E\boldsymbol{e}_1 = \textbf{vec}(F_{n_x}\textbf{array}(E\boldsymbol{e}_1)F_{n_y})$, where

$$\textbf{array}(E\boldsymbol{e}_1) \in \mathcal{R}^{n_x \times n_y} = [E_0\boldsymbol{e}_1, E_{-1}\boldsymbol{e}_1, \cdots, E_{1-n_x}\boldsymbol{e}_1, E_{n_x-1}\boldsymbol{e}_1, \cdots, E_1\boldsymbol{e}_1].$$

Now, $T = \textbf{array}(E\boldsymbol{e}_1)$ with $\hat{T} = \textbf{array}((F_{n_y} \otimes F_{n_x})E\boldsymbol{e}_1)$. Then for $W \in \mathbb{R}^{(2n_x-1)\times(2n_y-1)}$ with $\mathbf{w} = \textbf{vec}(W)$,

$$E\boldsymbol{w} = (F^*_{n_y} \otimes F^*_{n_x})\Lambda(F_{n_y} \otimes F_{n_x})\boldsymbol{w} = (F^*_{n_y} \otimes F^*_{n_x})(\boldsymbol{\lambda} \cdot *((F_{n_y} \otimes F_{n_x})\boldsymbol{x})).$$

Notice that $(F_{n_y} \otimes F_{n_x})\boldsymbol{w} = \text{vec}(F_{n_x} W F_{n_y})$. Removing the vectorization from the element-wise multiplication yields

$$E\boldsymbol{w} = (F^*_{n_y} \otimes F^*_{n_x})(\hat{T} \cdot *(F_{n_x} W F_{n_y})).$$

Now for $U \in \mathbb{R}^{n_x \times n_y}$ with $\boldsymbol{u} = \textbf{vec}(U)$, define W by

$$W = \begin{bmatrix} U & 0_{n_x(n_y-1)} \\ 0_{(n_x-1)n_y} & 0_{(n_x-1)(n_y-1)} \end{bmatrix}, \tag{3.7}$$

using 0_{mn} to denote a matrix of zeros of size $m \times n$, and compute

$$\textbf{array}(E\boldsymbol{w}) = T \star W = \textbf{ifft2}(\hat{T} \cdot * \hat{W}). \tag{3.8}$$

Here $\star$ denotes convolution, **ifft2** denotes the inverse `2DFFT`, and $\hat{W} = \textbf{fft2}(W)$. But now to obtain $B\mathbf{u}$ from this product, notice that each B_{ij} is in the upper left block of the corresponding E_j and $\boldsymbol{u}_{(j)}$ is segmented such that it multiplies with the first n_x entries in block column j. Hence,

$$\begin{bmatrix} B_{ij} & \dagger \\ \dagger & \dagger \end{bmatrix} \begin{bmatrix} \mathbf{u}_{(j)} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} B_{ij}\mathbf{u}_{(j)} \\ \dagger \end{bmatrix},$$

where † shows an omission of entries.

This form can be extended to E as a whole, producing

$$E\boldsymbol{w} = \mathbf{vec}\left(\left[\begin{array}{ccccccc} \sum_j B_{1j}\mathbf{u}_{(j)} & \sum_j B_{2j}\mathbf{u}_{(j)} & \cdots & \sum_j B_{n_yj}\mathbf{u}_{(j)} & \dagger & \cdots & \dagger \\ \dagger & \dagger & \cdots & \dagger & \dagger & \cdots & \dagger \end{array}\right]\right).$$

Thus, $B\mathbf{u}$ can be extracted as the top left $n_x \times n_y$ block of $E\boldsymbol{w}$ before vectorizing the result.

It is immediate that a set of equivalent steps can be used to calculate $B^T\mathbf{v}$, where $\mathbf{v} = \mathbf{vec}(V)$ for matrix V, since B^T is also BTTB. In addition, B^T is embedded in E^T. The matrix $\tilde{T}$ defining the BCCB extension of B^T uses T with columns 2 through $2n_y - 1$ swapped left to right, and rows 2 through $2n_x - 1$ swapped top to bottom. As shown by Bruun and Nielsen [2007], however, the complex conjugate can be used to produce the transpose operation. For $\boldsymbol{z} \in \mathbb{R}^{n_xn_y}$,

$$E^T\boldsymbol{z} = \Big((F_{n_y}^* \otimes F_{n_x}^*)\bar{\Lambda}(F_{n_y} \otimes F_{n_x})\Big)\,\boldsymbol{z} = (F_{n_y}^* \otimes F_{n_x}^*)(\bar{\hat{T}} \cdot *(F_{n_x}ZF_{n_y})),$$

and

$$\begin{aligned} \boldsymbol{z}^TE &= \boldsymbol{z}^T(F_{n_y}^* \otimes F_{n_x}^*)\Lambda(F_{n_y} \otimes F_{n_x}) \\ &= ((F_{n_y}^* \otimes F_{n_x}^*)\bar{\Lambda}(F_{n_y} \otimes F_{n_x})\boldsymbol{z})^* \\ &= ((F_{n_y}^* \otimes F_{n_x}^*)(\bar{\hat{T}} \cdot *(F_{n_x}ZF_{n_y})))^T. \end{aligned}$$

Thus, when $\hat{T}$ is available, the transpose operations can be performed using the complex conjugate of $\hat{T}$. Since B^T has blocks $A_j^T \in \mathbb{R}^{n_x \times n_x}$ with block dimensions $n_y \times n_y$, $B^T\mathbf{v}$ can also be extracted as the top left $n_x \times n_y$ block of $\mathbf{array}(E^T\boldsymbol{z})$ before vectorizing the result.

Now suppose the multiplication $D_LBD_R\boldsymbol{u}$ is needed, where D_L and D_R are diagonal with diagonal elements in vectors $\boldsymbol{d}_L$ and $\boldsymbol{d}_R$ respectively. The product $D_LBD_R\boldsymbol{u}$ is accomplished via $D_L(B(D_R\boldsymbol{u}))$, where now $B(D_R\boldsymbol{u})$ is obtained via the fast `2DFFT` based multiplication using a BCCB extension of B, and computational cost is further reduced by using element-wise multiplication $\boldsymbol{d}_L \cdot *(B(\boldsymbol{d}_R \cdot *\boldsymbol{u}))$. Specifically, using BCCB extension E of BTTB matrix B with corresponding extension $\boldsymbol{w}$ of $\boldsymbol{d}_R \cdot *\boldsymbol{u}$, compute $E\boldsymbol{w} = (F^*_{n_y} \otimes F^*_{n_x})\boldsymbol{\lambda} \cdot *(F_{n_y} \otimes F_{n_x})\boldsymbol{w}$, and extract $\boldsymbol{b} = B(D_R\boldsymbol{u})$ vectorized from the upper-left $n_x \times n_y$ block of $E\boldsymbol{w}$. Finally, $D_L B D_R \boldsymbol{u} = \boldsymbol{d}_L \cdot *\boldsymbol{b}$. Moreover, a similar approach is used for $D_R B^T D_L \boldsymbol{v}$, but using the complex conjugate as detailed above.

```
   Input: That for $\hat{T}$: see Table 3.1;
   $\mathbf{x}$ : vector for forward or transpose multiplication;
   $t$ : 1 or 2 for forward or transpose multiplication, respectively;
   prob_params : required parameters see Table 3.1
   Output: vector: $\mathbf{b}$ of size $m$ or $n$, for $t = 1, 2$, respectively.
1  Extract parameters from prob_params ;
2  Initialize zero array for $\mathbf{b}$ and $W$;
3  if $t == 2$ then
4      Initialize $W$ according to (3.39) ;
5      Take transform of $W$: $\hat{W} = \mathbf{fft2}(W)$;
6  end
7  for $j = 1 : n_z$ % For all layers of domain do
8      switch $t$ do
9          case 1 do
10             Initialize $W$ according to (3.38);
11             Take transform of $W$: $\hat{W} = \mathbf{fft2}(W)$;
12             Form convolution (3.8): $W = \mathbf{real}(\mathbf{ifft2}(\mathrm{That}(:,:,j) \cdot *\hat{W}))$;
13             Extract and accumulate top left block:
                 $\mathbf{b} = \mathbf{b} + \mathbf{reshape}(W(1:s_x, 1:s_y), m, 1)$;
14         end
15         case 2 do
16             Form convolution (3.8): $Z = \mathbf{real}(\mathbf{ifft2}(\mathbf{conj}(\mathrm{That}(:,:,j) \cdot *\hat{W})))$;
17             Extract top left block and assign to output:
                 $\mathbf{b}((j-1)n_r + 1 : jn_r) = \mathbf{reshape}(Z(1:n_x, 1:n_y), n_r, 1)$;
18         end
19     end
20 end
```

Algorithm 1: $\mathbf{b} = \texttt{mult\_BTTB}(\mathrm{That}, \mathbf{x}, t, \texttt{prob\_params})$
This algorithm calculates the forward and transpose multiplication, $G\mathbf{x}$, or $G^T\mathbf{x}$ as described in Section 3.1 using the embedding of the BTTB matrix in a BCCB matrix and the 2DFFT. The transform of (3.36) or (3.37) for symBTTB and BTTB, respectively, is precomputed and provided in That. See Table 3.1 for definitions of input parameters.

Now suppose also that G is a column block matrix,

$$G = \begin{bmatrix} G^{(1)}, & G^{(2)}, & \cdots & G^{(n_z)} \end{bmatrix},$$

and each $G^{(r)}$ is BTTB. Then it is also efficient to apply the 2DFFT based multiplication for $G\boldsymbol{u}$ and $G^T\boldsymbol{v}$, where $\boldsymbol{u} \in \mathbb{R}^{mn_z}$ and $\boldsymbol{v} \in \mathbb{R}^m$. Specifically, partition $\boldsymbol{u}$ into $\boldsymbol{u}_r \in \mathbb{R}^m$ for $1 \leq r \leq n_z$. Then

$$G\mathbf{u} = \sum_{r=1}^{n_z} G^{(r)}\mathbf{u}_r, \text{ and } G^T\mathbf{v} = \begin{bmatrix} \left(G^{(1)}\right)^T \mathbf{v} \\ \left(G^{(2)}\right)^T \mathbf{v} \\ \vdots \\ \left(G^{(n_z)}\right)^T \mathbf{v} \end{bmatrix}. \tag{3.9}$$

Now each $G^{(r)}\mathbf{u}_r$ and $\left(G^{(r)}\right)^T \mathbf{v}$ can be obtained via the 2DFFT based multiplication for each respective circulant extension. This leads to the specific case of application for the rest of this chapter, namely `gravity` and `magnetic` forward operations. Matrix operations using BCCB extensions for a BTTB matrix are demonstrated in Algorithm 1.

3.2 Matrix Structure for Spatially Invariant Kernels

The structure of the matrices is considered that arise for spatially invariant kernels, and the associated computations that are required. The discussion is presented first for domains without padding in Section 3.2.3, and then the modifications that are required when domain padding is introduced, Section 3.3.1. This will show that the kernels generate matrices that contain BTTB structure.

3.2.1 Spatially Invariant Kernels

This discussion focuses on kernels that are spatially invariant in all dimensions:

$$h(a, b, c, x, y, z) = h(x - a, y - b, z - c).$$

Then, considering a single slice at depth z, the calculation of (2.12) depends on the differences $(x-a)$ and $(y-b)$ for all station and prism coordinates, (2.9) and (2.10), respectively. Using the matrices

$$\left.\begin{array}{ll}(DX)_{ij,p} = (x_p - a_{ij}) & 0 \le p \le n_x \\ (DY)_{ij,q} = (y_q - b_{ij}) & 0 \le q \le n_y\end{array}\right\} 1 \le i \le s_x,\ 1 \le j \le s_y,$$

the distances for block pq, $1 \le p \le n_x$ and $1 \le q \le n_y$, are obtained from distance matrices $(DX)_{p-1}$ and $(DX)_p$ in x and likewise from $(DY)_{q-1}$ and $(DY)_q$ in y. Now, these matrices are independent of the slice coordinate r, and, under the assumption that the prisms are uniform in the $x-$ and $y-$ dimensions,

$$(DX)_p = (DX)_{p-1} + \Delta_x,\ 1 \le p \le n_x,\ (DY)_q = (DY)_{q-1} + \Delta_y,\ 1 \le q \le n_y.$$

Thus, all matrices $(DX)_p$ and $(DY)_q$ can be obtained directly from $(DX)_0$ and $(DY)_0$ and they are independent of the slice, regardless of the locations of the stations relative to the prisms. When the stations are on a uniform grid so that a_{ij} is independent of j and b_{ij} is independent of i, then the sizes of matrices $(DX)_0$ and $(DY)_0$ are reduced in the first dimension to s_x and s_y, respectively. It is practical, therefore, to store $(DX)_0$ and $(DY)_0$ entirely, and update an entire slice of the domain without recalculating $(DX)_0$ and $(DY)_0$ across slices, regardless of the station locations. Still, greater optimization is achieved when the stations are located also on a uniform grid.

3.2.2 Placement of the Stations at the Center of the Cells

Now, following Boulanger and Chouteau [2001], Chen and Liu [2018], suppose that the stations are placed at the centers of the cells such that $a_i = (i - \frac{1}{2})\Delta_x$, $1 \le i \le s_x$ and $b_j = (j - \frac{1}{2})\Delta_y$, $1 \le j \le s_y$. Then, the two coordinate systems for the stations and the volume domain, are uniformly staggered in the $x - y$ plane. Thus the distances between stations and coordinates are uniform,

$$\begin{aligned}
(DX)_{i,p} &= x_p - a_i &&= (p-1)\Delta_x - (i-\tfrac{1}{2})\Delta_x &&= (p-i-\tfrac{1}{2})\Delta_x, \quad 1 \le p \le n_x+1\\
(DY)_{j,q} &= y_q - b_j &&= (q-1)\Delta_y - (j-\tfrac{1}{2})\Delta_y &&= (q-j-\tfrac{1}{2})\Delta_y, \quad 1 \le q \le n_y+1,
\end{aligned}$$

and for all pairs of indices (i,p) and (j,q), the possible paired distances are obtained from the vectors

$$\begin{aligned}
X_\ell &= (\ell - s_x - \tfrac{1}{2})\Delta_x, \quad 1 \le \ell \le 2s_x\\
Y_k &= (k - s_y - \tfrac{1}{2})\Delta_y, \quad 1 \le k \le 2s_y.
\end{aligned} \tag{3.10}$$

Under these assumptions, the associated system matrix is then BTTB.

3.2.3 Symmetric Kernel Matrices with Toeplitz Block Structure

Consider first the case in which the matrix $G^{(r)}$ is symmetric BTTB. Specifically, suppose that the slice matrix $G^{(r)}$ has a symmetric block structure and is defined by its first row (and column) $G^{(r)}_{1q} = G^{(r)}_q$, $1 \le q \le n_y$. Then, with $n_x = s_x$ and $n_y = s_y$,

$$G^{(r)} = \begin{bmatrix}
G^{(r)}_1 & G^{(r)}_2 & G^{(r)}_3 & \dots & G^{(r)}_{n_y}\\
G^{(r)}_2 & G^{(r)}_1 & G^{(r)}_2 & \dots & G^{(r)}_{n_y-1}\\
\vdots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \vdots\\
G^{(r)}_{s_y} & G^{(r)}_{s_y-1} & G^{(r)}_{s_y-2} & \dots & G^{(r)}_1
\end{bmatrix}, \tag{3.11}$$

where each $G^{(r)}_q$ is symmetric and defined by its first row (and column),

$$G^{(r)}_q = \begin{bmatrix}
g_{1q} & g_{2q} & g_{3q} & \cdots & g_{n_xq}\\
g_{2q} & g_{1q} & g_{2q} & \cdots & g_{(n_x-1)q}\\
\vdots & \ddots & \ddots & \ddots & \vdots\\
\vdots & \ddots & \ddots & \ddots & \vdots\\
g_{s_xq} & g_{(s_x-1)q} & g_{(s_x-2)q} & \cdots & g_{1q}
\end{bmatrix}.$$

MATLAB notation can be used to write these matrices compactly in terms of the defining first row (column). Specifically, using MATLAB notation,

$$G_q^{(r)} = \textbf{toeplitz}(\mathbf{r}_q), \;\; \mathbf{r}_q = [g_{1q}, g_{2q}, g_{3q}, \dots, g_{n_x q}]^T, \tag{3.12}$$

and, with abuse of the same notation as applied to matrices,

$$G^{(r)} = \textbf{toeplitz}(R), \;\; R = \left[(G_1^{(r)})^T, (G_2^{(r)})^T, (G_3^{(r)})^T, \dots, (G_{n_y}^{(r)})^T\right]^T. \tag{3.13}$$

From (3.12) and (3.13) it is immediate, as discussed in Boulanger and Chouteau [2001] and Chen and Liu [2018], that the generation of $G^{(r)}$ requires only the calculation of its first row. But the first row represents the contributions of all prisms to the first station. Thus, to find $G^{(r)}$ requires only the calculation of

$$(G^{(r)})_{1\ell} = \tilde{h}(s_{11})_{pq}, \;\; \ell = (q-1)n_x + p, \;\; 1 \le p \le n_x, \;\; 1 \le q \le n_y \quad \text{or} \tag{3.14}$$

$$(G_1^{(r)})_{1:n_x n_y} = \left[\begin{array}{c|c|c|c} \mathbf{r}_1^T & \mathbf{r}_2^T & \dots & \mathbf{r}_{n_y}^T \end{array}\right] \quad \text{where}$$

$$\mathbf{r}_q = \left[\tilde{h}(s_{11})_{1q}, \tilde{h}(s_{11})_{2q}, \tilde{h}(s_{11})_{3q}, \dots, \tilde{h}(s_{11})_{n_x q}\right]^T, \;\; 1 \le q \le n_y. \tag{3.15}$$

Equivalently, it is sufficient to calculate only the distances $(DX)_{1,p} = (p - 3/2)\Delta_x$ and $(DY)_{1,q} = (q - 3/2)\Delta_y$, for $1 \le p \le n_x + 1$ and $1 \le q \le n_y + 1$, and (3.10) is replaced by

$$\begin{aligned} X_\ell &= (\ell - \tfrac{3}{2})\Delta_x, \quad 1 \le \ell \le (n_x + 1) \\ Y_k &= (k - \tfrac{3}{2})\Delta_y, \quad 1 \le k \le (n_y + 1). \end{aligned} \tag{3.16}$$

3.2.4 Nonsymmetric Kernel Matrices with Block Structure

Consider now the non symmetric BTTB matrix given by

$$G^{(r)} = \begin{bmatrix} G_1^{(r)} & G_2^{(r)} & G_3^{(r)} & \dots & \dots & G_{n_y}^{(r)} \\ \bar{G}_2^{(r)} & G_1^{(r)} & G_2^{(r)} & \dots & \dots & G_{n_y-1}^{(r)} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \bar{G}_{s_y}^{(r)} & \bar{G}_{s_y-1}^{(r)} & \bar{G}_{s_y-2}^{(r)} & \dots & \dots & G_1^{(r)} \end{bmatrix}, \tag{3.17}$$

where, without padding, $n_y = s_y$. This matrix depends on the first block row and column only, and, again using the abuse of the Toeplitz notation, is given by

$$G^{(r)} = \textbf{toeplitz}(C, R), \tag{3.18}$$
$$C = \left[(G_1^{(r)})^T, (\bar{G}_2^{(r)})^T, \dots, (\bar{G}_{s_y}^{(r)})^T\right]^T, \; R = \left[(G_1^{(r)})^T, (G_2^{(r)})^T, \dots, (G_{s_y}^{(r)})^T\right]^T.$$

Here $\bar{G}_j^{(r)}$ is used to denote the contributions below the diagonal, and $G_q^{(r)}$ for the contributions above the diagonal. None of the block matrices are symmetric and, therefore, to calculate $G^{(r)}$ it is necessary to calculate columns and rows that define the entries in C and R. Calculating R uses just the first row entries $G_q^{(r)}$, but since each of these is not symmetric the first columns $\mathbf{c}_q$ of each block in $G_q^{(r)}$ are also needed.

Using (2.12), the $G_q^{(r)}$ are given by (3.24) with

$$\mathbf{r}_q = \left[\tilde{h}(s_{11})_{1q}, \tilde{h}(s_{11})_{2q}, \dots, \tilde{h}(s_{11})_{n_xq}\right]^T, \; 1 \le q \le n_y \tag{3.19}$$
$$\mathbf{c}_q = \left[\tilde{h}(s_{11})_{1q}, \tilde{h}(s_{21})_{1q}, \dots, \tilde{h}(s_{s_x1})_{1q}\right]^T, \; 1 \le q \le n_y. \tag{3.20}$$

Effectively, rather than calculating all entries in the first block row of $G^{(r)}$, $(s_xn_x)n_y$ entries, for each matrix of the block, just its first row and column are calculated, for a total of $(s_x + n_x)n_y$ entries.

This leaves the calculation of the $\bar{G}_j^{(r)}$, $2 \le j \le s_y$, which by the Toeplitz structure only use the entries of the first block column of $G^{(r)}$. They are given by

$$\bar{G}_j^{(r)} = \textbf{toeplitz}(\bar{\mathbf{c}}_j, \bar{\mathbf{r}}_j), \; 2 \le j \le s_y, \tag{3.21}$$
$$\bar{\mathbf{c}}_j = \left[\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{2j})_{11}, \dots, \tilde{h}(s_{s_xj})_{11}\right]^T, \; 2 \le j \le s_y, \text{ and}$$
$$\bar{\mathbf{r}}_j = \left[\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{1j})_{21}, \dots, \tilde{h}(s_{1j})_{n_x1}\right]^T, \; 2 \le j \le s_y. \tag{3.22}$$

It is also immediately clear that this requires not only all distances between the first station and all prism coordinates, as in (3.16), but also for all stations and the first coordinate block (for the first column of $G^{(r)}$) which uses $X_\ell = (\ell + \frac{1}{2})\Delta_x$, $-s_x \le \ell \le 0$, and likewise for Y_k. Thus, the full set of differences (3.10) are required.

3.3 Introducing Padding Around the Domain

Building upon the structure established in Section 2.5, consider the more general configuration for the `gravity` and `magnetic` problems on which padding is imposed. Now that padding is introduced around the domain, with an extra $p_{x_{\mathrm{L}}}$ and $p_{x_{\mathrm{R}}}$ blocks in the x-direction, the $x-$coordinates extend from $(-p_{x_{\mathrm{L}}} : (s_x + p_{x_{\mathrm{R}}}))\Delta_x$ for a total of s_x coordinate blocks within the domain but a total number of blocks $n_x = (s_x + p_{x_{\mathrm{L}}} + p_{x_{\mathrm{R}}})$, where blocks 1 to $p_{x_{\mathrm{L}}}$ are in the padded region to the left of the domain, and blocks $s_x + p_{x_{\mathrm{L}}} + 1$ to n_x are within the padded region to the right. Thus, the coordinates of block p are adjusted to $(p - p_{x_{\mathrm{L}}} - 1)\Delta_x$ to $(p - p_{x_{\mathrm{L}}})\Delta_x$, consistent with (2.10) for $p_{x_{\mathrm{L}}} = 0$. Likewise, the y coordinates extend from $-p_{y_{\mathrm{L}}} : (s_y + p_{y_{\mathrm{R}}})$ and $n_y = (s_y + p_{y_{\mathrm{L}}} + p_{y_{\mathrm{R}}})$, see Figure 3.1. Hence, (3.10) is replaced by

$$\begin{aligned} X_\ell &= (\ell - (s_x + p_{x_{\mathrm{L}}}) - \tfrac{1}{2})\Delta_x, \quad 1 \le \ell \le 2s_x + p_{x_{\mathrm{L}}} + p_{x_{\mathrm{R}}} = n_x + s_x \\ Y_k &= (k - (s_y + p_{y_{\mathrm{L}}}) - \tfrac{1}{2})\Delta_y, \quad 1 \le k \le 2s_y + p_{y_{\mathrm{L}}} + p_{y_{\mathrm{R}}} = n_y + s_y. \end{aligned} \tag{3.23}$$

If all possible paired distances are needed for calculating $\tilde{h}$, vectors X and Y only need to be stored, as given by (3.23), which is negligible as compared to the entire storage of the mn entries in the matrix G. Note that (3.23) explicitly assumes no stations in the padded regions.

3.3.1 Symmetric Kernel Matrices with Toeplitz Block Structure and Domain Padding

Suppose now that the domain is padded in the x and y directions, with no real stations within the padded region. To illustrate the impact of the padding on the generation of the matrix consider a one-dimensional example with $s_x = 4$, $p_{x_{\mathrm{L}}} = 2$ and $p_{x_{\mathrm{R}}} = 1$. Suppose first that there are artificial stations in the first two blocks and in the final block, namely for

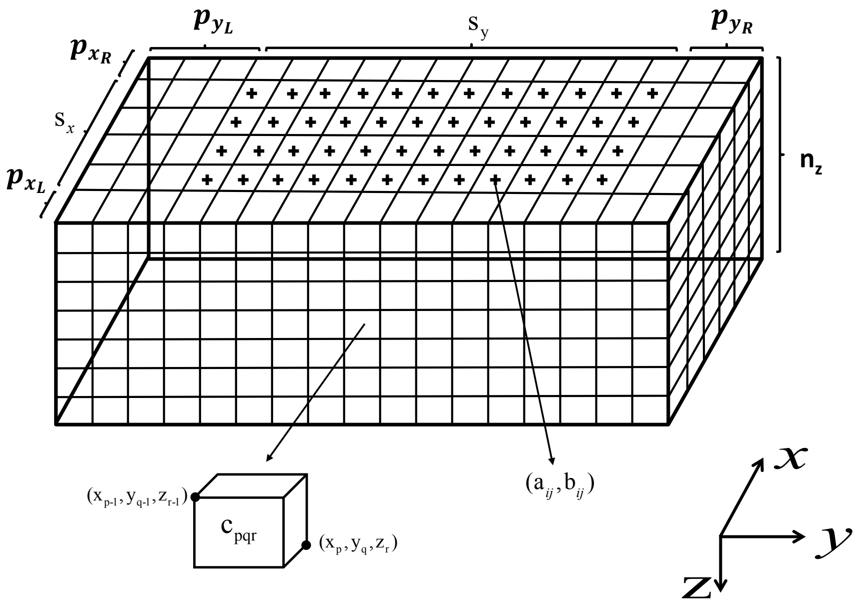


Figure 3.1: The configuration of the volume domain with padding.

blocks 1, 2, and 7. Then the single square and symmetric Toeplitz that defines $G^{(r)}$ is

$$G_1^{(r)} = \left[\begin{array}{cc|cccc|c|l}
g_1 & g_2 & g_3 & g_4 & g_5 & g_6 & g_7 & \text{Station } 1(\text{Artificial}) \\
g_2 & g_1 & g_2 & g_3 & g_4 & g_5 & g_6 & \text{Station } 2(\text{Artificial}) \\
\hline
g_3 & g_2 & g_1 & g_2 & g_3 & g_4 & g_5 & \text{Station } 3 = p_{x_\mathrm{L}} + 1 \\
g_4 & g_3 & g_2 & g_1 & g_2 & g_3 & g_4 & \text{Station } 4 \\
g_5 & g_4 & g_3 & g_2 & g_1 & g_2 & g_3 & \text{Station } 5 \\
g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & g_2 & \text{Station } 6 = p_{x_\mathrm{L}} + s_x \\
\hline
g_7 & g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & \text{Station } 7(\text{Artificial})
\end{array}\right].$$

This depends on

$$\mathbf{r} = (g_1, g_2, g_3, g_4, g_5, g_6, g_7)^T = (\tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \tilde{h}(s_1)_3, \ldots, \tilde{h}(s_1)_7)^T.$$

But the contribution to the first real station due to all prisms is given by the third row, row $p_{x_\mathrm{L}} + 1$ of $G_1^{(r)}$, which is

$$(G_1^{(r)})_3 = (g_3, g_2, g_1, g_2, g_3, g_4, g_5)\,,$$

and the contributions for the real stations are determined, using symmetry, by

$$(G_1^{(r)})(p_{x_{\mathrm{L}}}+1:p_{x_{\mathrm{L}}}+s_x,:) = \begin{bmatrix} g_3 & g_2 & g_1 & g_2 & g_3 & g_4 & g_5 \\ g_4 & g_3 & g_2 & g_1 & g_2 & g_3 & g_4 \\ g_5 & g_4 & g_3 & g_2 & g_1 & g_2 & g_3 \\ g_6 & g_5 & g_4 & g_3 & g_2 & g_1 & g_2 \end{bmatrix}$$
$$= \textbf{toeplitz}(\mathbf{c}, \mathbf{r}).$$

Here

$$\mathbf{c} = (g_3, g_4, g_5, g_6) \text{ and } \mathbf{r} = (g_3, g_2, g_1, g_2, g_3, g_4, g_5)^T.$$

More generally, for one dimension only,

$$\mathbf{c} = (g_{p_{x_{\mathrm{L}}}+1}, g_{p_{x_{\mathrm{L}}}+2}, \ldots, g_{p_{x_{\mathrm{L}}}+s_x})^T = (\tilde{h}(s_1)_{p_{x_{\mathrm{L}}}+1}, \tilde{h}(s_1)_{p_{x_{\mathrm{L}}}+2}, \ldots, \tilde{h}(s_1)_{p_{x_{\mathrm{L}}}+s_x})^T \text{ and}$$
$$\mathbf{r} = (g_{p_{x_{\mathrm{L}}}+1}, \ldots, g_2, g_1, g_2, \ldots, g_{n_x-p_{x_{\mathrm{L}}}})^T$$
$$= (\tilde{h}(s_1)_{p_{x_{\mathrm{L}}}+1}, \ldots, \tilde{h}(s_1)_2, \tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \ldots, \tilde{h}(s_1)_{n_x-p_{x_{\mathrm{L}}}})^T.$$

Extending to the two-dimensional case, and assuming that the first artificial station is in the $(1,1)$ block of the padded domain, then $G_q^{(r)}$, for any q, is also Toeplitz and is given by

$$G_q^{(r)} = \textbf{toeplitz}(\mathbf{c}_q, \mathbf{r}_q), \ 1 \le q \le n_y, \tag{3.24}$$
$$\mathbf{c}_q = (\tilde{h}(s_{11})_{(p_{x_{\mathrm{L}}}+1)q}, \tilde{h}(s_{11})_{(p_{x_{\mathrm{L}}}+2)q}, \ldots, \tilde{h}(s_{11})_{(p_{x_{\mathrm{L}}}+s_x)q})^T \text{ and} \tag{3.25}$$
$$\mathbf{r}_q = (\tilde{h}(s_{11})_{(p_{x_{\mathrm{L}}}+1)q}, \ldots, \tilde{h}(s_{11})_{2q}, \tilde{h}(s_{11})_{1q}, \tilde{h}(s_{11})_{2q}, \ldots, \tilde{h}(s_{11})_{(n_x-p_{x_{\mathrm{L}}})q})^T. \tag{3.26}$$

This is consistent with (3.14) - (3.15) for the unpadded case. But notice, also, that the maximum distance between station and coordinates in the $x-$coordinate is $\textbf{max}(n_x - p_{x_{\mathrm{L}}}, n_x - p_{x_{\mathrm{R}}})\Delta_x = \textbf{max}(s_x + p_{x_{\mathrm{R}}}, s_x + p_{x_{\mathrm{L}}})\Delta_x$.

It remains to apply the same argument to the structure of the matrix $G^{(r)}$, as to the structure of its individual components, to determine the structure of the symBTTB matrix when padding is applied. Then, consistent with (3.24)-(3.26), (3.13) is replaced by

$$G^{(r)} = \textbf{toeplitz}(C, R), \tag{3.27}$$

$$C = ((G^{(r)}_{p_{y_{\mathrm{L}}}+1})^T, \dots, (G^{(r)}_{p_{y_{\mathrm{L}}}+s_y})^T)^T \text{ and} \tag{3.28}$$

$$R = ((G^{(r)}_{p_{y_{\mathrm{L}}}+1})^T, \dots, (G^{(r)}_2)^T, (G^{(r)}_1)^T, (G^{(r)}_2)^T, (G^{(r)}_3)^T, \dots, (G^{(r)}_{n_y-p_{y_{\mathrm{L}}}})^T)^T. \tag{3.29}$$

Moreover, since this matrix depends on the first row of the symmetric matrix, defined with respect to the artificial station at s_{11}, it is sufficient to still use (3.16) for the calculation of the relevant distances between the first station and all coordinate blocks. But, from (3.25) - (3.26), and (3.28)-(3.29), just as all entries g_j in $G^{(r)}_q$ are not calculated, all the blocks $G^{(r)}_q$ are also not calculated, rather the blocks needed are for $q = 1 : \textbf{max}(n_y - p_{y_{\mathrm{L}}}, n_y - p_{y_{\mathrm{R}}})$. Thus, while (3.16) may be used to calculate the relevant distances, in practice some savings in memory and computation can be made, when padding is significant relative to s_x and s_y, by using

$$\begin{aligned} X_\ell &= (\ell - \tfrac{3}{2})\Delta_x, \quad 1 \le \ell \le s_x + \textbf{max}(p_{x_{\mathrm{R}}}, p_{x_{\mathrm{L}}}) + 1 \\ Y_k &= (k - \tfrac{3}{2})\Delta_y, \quad 1 \le k \le s_y + \textbf{max}(p_{y_{\mathrm{R}}}, p_{y_{\mathrm{L}}}) + 1. \end{aligned} \tag{3.30}$$

3.3.2 Nonsymmetric Kernel Matrices with Block Structure and Domain Padding

As for the discussion of the impact of the domain padding on the symmetric kernel in Section 3.3.1, consider first an example using one dimension, namely fixed y-coordinate. Again assume $s_x = 4$, $p_{x_{\mathrm{L}}} = 2$ and $p_{x_{\mathrm{R}}} = 1$ and suppose that there are artificial stations in the first two blocks and in the final block, namely for blocks 1, 2, and 7. Then, the single square but non-symmetric Toeplitz matrix that defines $G^{(r)}$ is

$$G_1^{(r)} = \left[\begin{array}{cc|cccc|c} g_1 & g_2 & g_3 & g_4 & g_5 & g_6 & g_7 \\ \gamma_2 & g_1 & g_2 & g_3 & g_4 & g_5 & g_6 \\ \hline \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 & g_5 \\ \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 \\ \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 \\ \gamma_6 & \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 \\ \hline \gamma_7 & \gamma_6 & \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 \end{array}\right] \quad \begin{array}{|l} \text{Station } 1(\text{Artificial}) \\ \text{Station } 2(\text{Artificial}) \\ \hline \text{Station } 3 = p_{x_\text{L}} + 1 \\ \text{Station } 4 \\ \text{Station } 5 \\ \text{Station } 6 = p_{x_\text{L}} + s_x \\ \hline \text{Station } 7(\text{Artificial}) \end{array}$$

This depends on

$$\mathbf{r} = (g_1, g_2, g_3, g_4, g_5, g_6, g_7)^T = (\tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \dots, \tilde{h}(s_1)_6, \tilde{h}(s_1)_7)^T$$
$$\mathbf{c} = (g_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6, \gamma_7)^T = (\tilde{h}(s_1)_1, \tilde{h}(s_2)_1, \dots, \tilde{h}(s_6)_1, \tilde{h}(s_7)_1)^T.$$

But again the required rows of $G_1^{(r)}$ are those that correspond to the actual real stations

$$(G_1^{(r)})(p_{x_\text{L}} + 1 : p_{x_\text{L}} + s_x, :) = \begin{bmatrix} \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 & g_5 \\ \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 & g_4 \\ \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 & g_3 \\ \gamma_6 & \gamma_5 & \gamma_4 & \gamma_3 & \gamma_2 & g_1 & g_2 \end{bmatrix}$$
$$= \textbf{toeplitz}(\mathbf{c}, \mathbf{r}) \text{ where}$$
$$\mathbf{c} = (\gamma_3, \gamma_4, \gamma_5, \gamma_6)^T \text{ and } \mathbf{r} = (\gamma_3, \gamma_2, g_1, g_2, g_3, g_4, g_5)^T.$$

More generally,

$$\mathbf{c} = (\gamma_{p_{x_\text{L}}+1}, \gamma_{p_{x_\text{L}}+2}, \dots, \gamma_{p_{x_\text{L}}+s_x})^T = (\tilde{h}(s_{p_{x_\text{L}}+1})_1, \tilde{h}(s_{p_{x_\text{L}}+2})_1, \dots, \tilde{h}(s_{p_{x_\text{L}}+s_x})_1)^T \text{ and}$$
$$\mathbf{r} = (\gamma_{p_{x_\text{L}}+1}, \gamma_{p_{x_\text{L}}}, \dots, \gamma_2, g_1, g_2, \dots, g_{n_x - p_{x_\text{L}}})^T$$
$$= (\tilde{h}(s_{p_{x_\text{L}}+1})_1, \tilde{h}(s_{p_{x_\text{L}}})_1, \dots, \tilde{h}(s_2)_1, \tilde{h}(s_1)_1, \tilde{h}(s_1)_2, \dots, \tilde{h}(s_1)_{n_x - p_{x_\text{L}}})^T.$$

Extending to the two-dimensional case, with the same assumptions as in Section 3.3.1, $G_q^{(r)}$ is obtained as

$$G_q^{(r)} = \textbf{toeplitz}(\mathbf{c}_q, \mathbf{r}_q),\ 1 \le q \le n_y, \tag{3.31}$$
$$\mathbf{c}_q = (\tilde{h}(s_{(p_{x_\mathrm{L}}+1)1})_{1q}, \tilde{h}(s_{(p_{x_\mathrm{L}}+2)1})_{1q}, \ldots, \tilde{h}(s_{(p_{x_\mathrm{L}}+s_x)1})_{1q})^T \text{ and}$$
$$\mathbf{r}_q = (\tilde{h}(s_{(p_{x_\mathrm{L}}+1)1})_{1q}, \tilde{h}(s_{p_{x_\mathrm{L}}1})_{1q}, \ldots, \tilde{h}(s_{21})_{1q}\tilde{h}(s_{11})_{1q}, \ldots, \tilde{h}(s_{11})_{(n_x-p_{x_\mathrm{L}})q})^T.$$

This is consistent with (3.19) - (3.20) for the unpadded case.

Turning to the column block entries, first observe that $\bar{G}_1^{(r)} = G_1^{(r)}$, and so examine $\bar{G}_j^{(r)}$ which represents stations 1 to n_x (both real and artificial) in the x-direction for a fixed j coordinate in the y-direction. Then, with the same example for choices of s_x, p_{x_L} and p_{x_R},

$$\bar{G}_j^{(r)} = \left[\begin{array}{cc|cccc|c} \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 & \bar{g}_5 & \bar{g}_6 & \bar{g}_7 \\ \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 & \bar{g}_5 & \bar{g}_6 \\ \hline \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & g_4 & \bar{g}_5 \\ \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 \\ \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 \\ \bar{\gamma}_6 & \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 \\ \hline \bar{\gamma}_7 & \bar{\gamma}_6 & \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & g_1 \end{array}\right] \quad \begin{array}{|l} \text{Station } 1(\text{Artificial}) \\ \text{Station } 2(\text{Artificial}) \\ \hline \text{Station } 3 = p_{x_\mathrm{L}} + 1 \\ \text{Station } 4 \\ \text{Station } 5 \\ \text{Station } 6 = p_{x_\mathrm{L}} + s_x \\ \hline \text{Station } 7(\text{Artificial}) \end{array}$$

This depends on

$$\bar{\mathbf{r}} = (\bar{g}_1, \bar{g}_2, \bar{g}_3, \bar{g}_4, \bar{g}_5, \bar{g}_6, \bar{g}_7)^T = (\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{1j})_{21}, \ldots, \tilde{h}(s_{1j})_{61}, \tilde{h}(s_{1j})_{71})^T$$
$$\bar{\mathbf{c}} = (\bar{g}_1, \bar{\gamma}_2, \bar{\gamma}_3, \bar{\gamma}_4, \bar{\gamma}_5, \bar{\gamma}_6, \bar{\gamma}_7)^T = (\tilde{h}(s_{1j})_{11}, \tilde{h}(s_{2j})_{11}, \ldots, \tilde{h}(s_{6j})_{11}, \tilde{h}(s_{7j})_{11})^T.$$

But again, since stations 1 to 2 and 7 are artificial, the only needed elements are

$$(\bar{G}_j^{(r)})(p_{x_{\rm L}}+1:p_{x_{\rm L}}+s_x,:) = \begin{bmatrix} \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 & \bar{g}_5 \\ \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 & \bar{g}_4 \\ \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 & \bar{g}_3 \\ \bar{\gamma}_6 & \bar{\gamma}_5 & \bar{\gamma}_4 & \bar{\gamma}_3 & \bar{\gamma}_2 & \bar{g}_1 & \bar{g}_2 \end{bmatrix}$$

$$= \textbf{toeplitz}(\mathbf{c},\mathbf{r}) \text{ where}$$

$$\mathbf{c} = (\bar{\gamma}_3,\bar{\gamma}_4,\bar{\gamma}_5,\bar{\gamma}_6)^T \text{ and } \mathbf{r} = (\bar{\gamma}_3,\bar{\gamma}_2,\bar{g}_1,\bar{g}_2,\bar{g}_3,\bar{g}_4,\bar{g}_5)^T.$$

Thus, in two dimensions, the first column block entries are $\bar{G}_j^{(r)}$, $1 \le j \le n_y$, with

$$\begin{aligned} \bar{G}_j^{(r)} &= \textbf{toeplitz}(\bar{\mathbf{c}}_j,\bar{\mathbf{r}}_j),\ 1 \le j \le s_y + p_{y_{\rm L}} + p_{y_{\rm R}}, \\ \bar{\mathbf{c}}_j &= (\tilde{h}(s_{(p_{x_{\rm L}}+1)j})_{11}, \tilde{h}(s_{(p_{x_{\rm L}}+2)j})_{11}, \dots, \tilde{h}(s_{(p_{x_{\rm L}}+s_x)j})_{11})^T \text{ and} \\ \bar{\mathbf{r}}_j &= (\tilde{h}(s_{(p_{x_{\rm L}}+1)j})_{11}, \tilde{h}(s_{p_{x_{\rm L}}j})_{11}, \dots, \tilde{h}(s_{2j})_{11}, \tilde{h}(s_{1j})_{11}, \dots, \tilde{h}(s_{1j})_{(n_x-p_{x_{\rm L}})1})^T. \end{aligned} \tag{3.32}$$

But now (3.18) is replaced by the block Toeplitz matrix

$$\begin{aligned} G^{(r)} &= \textbf{toeplitz}(C,R), \\ C &= ((\bar{G}^{(r)}_{p_{y_{\rm L}}+1})^T, \dots, (\bar{G}^{(r)}_{p_{y_{\rm L}}+s_y})^T)^T \text{ and} \\ R &= ((\bar{G}^{(r)}_{p_{y_{\rm L}}+1})^T, (\bar{G}^{(r)}_{p_{y_{\rm L}}})^T, \dots, (\bar{G}^{(r)}_{2})^T, (G^{(r)}_{1})^T, \dots, (G^{(r)}_{n_y-p_{y_{\rm L}}})^T)^T. \end{aligned} \tag{3.33}$$

Here each block matrix is the subset of rows corresponding to the real stations, as noted in (3.31) and (3.32). Moreover, we conclude that (3.31) is applied only for $1 \le q \le n_y - p_{y_{\rm L}} = s_y + p_{y_{\rm R}}$ and (3.32) for $1 \le j \le p_{y_{\rm L}} + s_y$, reducing the dimension of the required Y in the $y-$direction. Likewise, X is reduced because of the padding impacting the required entries for generating both $G_q^{(r)}$ and $\bar{G}_j^{(r)}$. Thus while the required vectors are given by (3.10), with n_x replacing s_x and n_y replacing s_y, their lengths can be reduced as for the symmetric case, (3.30), by using

$$\begin{aligned} X_\ell &= (\ell - (s_x + \textbf{max}(p_{x_{\rm R}}, p_{x_{\rm L}})) - \tfrac{1}{2})\Delta_x, \quad 1 \le \ell \le 2(s_x + \textbf{max}(p_{x_{\rm R}}, p_{x_{\rm L}})) \\ Y_k &= (k - (s_y + \textbf{max}(p_{y_{\rm R}}, p_{y_{\rm L}})) - \tfrac{1}{2})\Delta_y, \quad 1 \le k \le 2(s_y + \textbf{max}(p_{y_{\rm R}}, p_{y_{\rm L}})). \end{aligned} \tag{3.34}$$

This effectively assumes the calculation of $\bar{G}_1^{(r)}$ as well as $G_1^{(r)}$, whereas only one is calculated in practice, since $\bar{G}_1^{(r)} = G_1^{(r)}$.

3.3.3 Convolution with Domain Padding

For the padded domain, assume that the indices for $\mathbf{r}_p$ and $\mathbf{c}_q$ are from the first row and column of the padded domain. Then, for the case of the symBTTB matrix, (3.4) is replaced by

$$\mathbf{r}_q^{\text{ext}} = \begin{bmatrix} \mathbf{r}_q \\ J_{s_x-1}\mathbf{c}_q(2:s_x) \end{bmatrix}, \quad \mathbf{c}_q^{\text{ext}} = \begin{bmatrix} \mathbf{c}_q \\ J_{n_x-1}\mathbf{r}_q(2:n_x) \end{bmatrix}, \tag{3.35}$$

here using the definitions (3.25) and (3.26) for $\mathbf{r}_q$ and $\mathbf{c}_q$. But now since $\mathbf{r}_q$ is defined by $\mathbf{c}_q$ for the symmetric case only $\mathbf{c}_q^{\text{ext}}$ is needed. Each vector is of length s_x for $\mathbf{c}_q$ and $n_x - 1$ for $J_{n_x-1}(\mathbf{r}_q(2:n_x))$. Then using (3.28), the BCCB extension is defined by the matrix

$$T = \left[\begin{array}{ccccccc|ccc} \mathbf{c}_{1+p_{y_\text{L}}}^{\text{ext}} & \cdots & \mathbf{c}_{s_y+p_{y_\text{L}}}^{\text{ext}} & \mathbf{c}_{s_y+p_{y_\text{R}}}^{\text{ext}} & \cdots & \mathbf{c}_2^{\text{ext}} & & \mathbf{c}_1^{\text{ext}} & \cdots & \mathbf{c}_{p_{y_\text{L}}}^{\text{ext}} \end{array}\right], \tag{3.36}$$

of size $(s_x + n_x - 1) \times (s_y + n_y - 1)$. For the nonsymmetric case (3.36) is replaced by

$$T = \left[\begin{array}{ccccccc|ccc} \bar{\mathbf{c}}_{1+p_{y_\text{L}}}^{\text{ext}} & \cdots & \bar{\mathbf{c}}_{s_y+p_{y_\text{L}}}^{\text{ext}} & \mathbf{c}_{s_y+p_{y_\text{R}}}^{\text{ext}} & \cdots & \mathbf{c}_2^{\text{ext}} & & \bar{\mathbf{c}}_1^{\text{ext}} & \cdots & \bar{\mathbf{c}}_{p_{y_\text{L}}}^{\text{ext}} \end{array}\right], \tag{3.37}$$

where $\bar{\mathbf{c}}_j^{\text{ext}}$ is obtained as in (3.35) but using $\bar{\mathbf{c}}_j$ and $\bar{\mathbf{r}}_j$ from (3.32).

Figure 3.2 shows the complete BTTB slice $G^{(r)}$ with domain padding ($s_y = 3$, $n_y = 4$), with defining elements given by T that is generated using the corresponding row and column entries.

Note that in any case in which $p_{y_\text{L}} = 0$ then the end block is removed. Moreover, due to $G^{(r)}\mathbf{u}$ of size $s_x s_y$, when $\mathbf{u}$ is of size $n_x n_y$, the definition of W in (3.7) is replaced by

$$W = \begin{bmatrix} U & 0_{n_x(s_y-1)} \\ 0_{(s_x-1)n_y} & 0_{(s_x-1)(s_y-1)} \end{bmatrix}, \tag{3.38}$$

with $U \in \mathcal{R}^{n_x \times n_y}$.

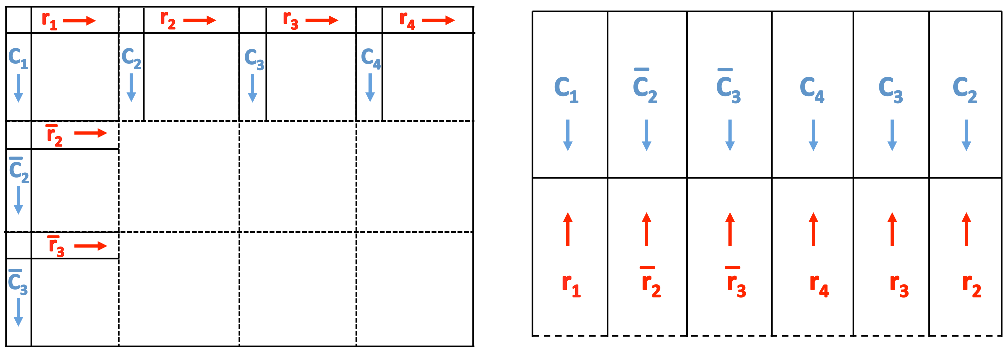


Figure 3.2: Required unique entries from a BTTB matrix $G^{(r)}$ in order to form the corresponding configuration of T, where the arrow denotes the direction of the vector in ascending order. The dotted line indicates that the first elements of each $\boldsymbol{r}_q$ and $\bar{\boldsymbol{r}}_q$ are omitted in the construction of T.

The transpose operation can still be applied via $\tilde{T}$ directly. But notice that for $\mathbf{v}$ of size $s_x s_y$, $(G^{(r)})^T\mathbf{v}$ is of size $n_x n_y$, and in this case (3.7) is replaced by

$$\tilde{W} = \begin{bmatrix} V & 0_{s_x(n_y-1)} \\ 0_{(n_x-1)s_y} & 0_{(n_x-1)(n_y-1)} \end{bmatrix}, \tag{3.39}$$

with V of size $s_x \times s_y$.

3.4 Optimizing the Calculations for Specific Kernels

Chen and Liu [2018] demonstrated that considerable savings are realized in the generation of the entries for the matrices T through optimized calculations of the entries necessary for finding T for forward modeling of the `gravity` problem. Here the focus is on both improving that optimization and with the generation of an optimized and stable calculation for the `magnetic` kernel following the derivation of Rao and Babu [1991]. Relevant parameters are given in Table 3.1.

Table 3.1: Parameters and variables in the codes. The parameters are defined in Table 2.1.

`prob_params`	$s_x, s_y, n_z, p_{x_\mathrm{L}}, p_{x_\mathrm{R}}, p_{y_\mathrm{L}}, p_{y_\mathrm{R}}, n_x, n_y, m, n, n_r, p_x, p_y$
gsx, gsy, gsz	Grid sizes Δ_x, Δ_y and Δ_z
That	That.forward $= \hat{T}$
z_blocks	Depth coordinates, increasing, z_r
D	Declination of geomagnetic field and magnetization vector
I	Inclination of geomagnetic field and magnetization vector
F	Intensity of the geomagnetic field in nT ($10^{-9}F$ in T)
$H = \frac{10^{-9}F}{4\pi}$	Magnetic field intensity (A/m) in SI units
$\tilde{H} = 10^9 H = \frac{F}{4\pi}$	Assumes the field is measured in nT

3.4.1 Gravity Kernel Calculation

The `gravity` kernel generates a symBTTB matrix for each slice in depth (z-direction). According to Boulanger and Chouteau [2001], and as used in Chen and Liu [2018], the contribution of the kernel from the prism at point (p, q) on the volume grid, (where x points East and y points North), to the station at location (a, b, c), and here assume $c = 0 = z_0$, is given by

$$\tilde{h}(a,b,c)_{pq} = \gamma \sum_{i=1}^{2}\sum_{j=1}^{2}\sum_{k=1}^{2}(-1)^i(-1)^j(-1)^k \left(\mathrm{Z}_k \arctan\frac{\mathrm{X}_i \Upsilon_j}{\mathrm{Z}_k \mathrm{P}_{ij}^k} - \mathrm{X}_i \ln\left(\mathrm{P}_{ij}^k + \Upsilon_j\right) - \Upsilon_j \ln\left(\mathrm{P}_{ij}^k + \mathrm{X}_i\right)\right).$$

Here γ is the gravitational constant and

$$\begin{aligned} \mathrm{X}_1 &= x_{p-1} - a, & \mathrm{X}_2 &= x_p - a \\ \Upsilon_1 &= y_{q-1} - b & \Upsilon_2 &= y_q - b \\ \mathrm{Z}_1 &= z_{r-1} - c, & \mathrm{Z}_2 &= z_r - c \\ R_{ij}^2 &= \mathrm{X}_i^2 + \Upsilon_j^2 & \mathrm{P}_{ij}^k &= \sqrt{R_{ij}^2 + \mathrm{Z}_k^2}. \end{aligned} \tag{3.40}$$

```
   **Input:** See Table 3.1 for details;
   $gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
   prob_params required parameters
   **Output:** symBTTB real matrix $G$ of size $m \times n$.
1  Extract parameters from prob_params ;
2  Initialize zero arrays: $G$, $Gr$, $Grq$;
3  Sizes: $nX = s_x + \mathbf{max}(p_{x_\mathrm{L}}, p_{x_\mathrm{R}})$, $nY = s_y + \mathbf{max}(p_{y_\mathrm{L}}, p_{y_\mathrm{R}})$;
4  Form distance arrays $X$ and $Y$ according to (3.30);
5  Form $X2 = X{\cdot}^2$, $Y2 = Y{\cdot}^2$, $XY = X(:) \cdot *Y$ and $R = X2(:) + Y2$;
6  **for** $r = 1 : n_z$ **do**
7      Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
8      Calculate slice response at first station: $g = \mathtt{gravity}(z_1, z_2, X, Y, XY, R)$;
9      **for** $q = 1 : nY$ **do**
10         Extract $\mathbf{c}_q$, $\mathbf{r}_q$ from $g$ : use (3.25), (3.26);
11         Generate: $Grq = \mathbf{toeplitz}(\mathbf{c}_q, \mathbf{r}_q)$: use (3.24) ;
12     **end**
13     **for** $j = [p_{y_L} + 1 : -1 : 2, 1 : s_y + p_{y_R}]$ **do**
14         Build first row of $Gr$ using (3.29);
15     **end**
16     **for** $j = 2 : s_y$ **do**
17         Build $j^{\mathrm{th}}$ row of $Gr$ using (3.27) and (3.28);
18     **end**
19     Assign: $Gr$ to $r^{\mathrm{th}}$ block of $G$;
20 **end**
```

Algorithm 2: $G = \mathtt{sym\_BTTB}(gsx, gsy, z\_blocks, \mathtt{prob\_params})$
Entries of padded symBTTB matrix. Function `gravity`.

Specifically, the summation needed is given by

$$\sum_i \sum_j (-1)^{i+j+1} \left(\left(\mathrm{Z}_1 \arctan \frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_1 (R_1)_{ij}} - \mathrm{Z}_2 \arctan \frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_2 (R_2)_{ij}} \right) - \right.$$

$$\left. \mathrm{X}_i \left(\ln((R_1)_{ij} + \Upsilon_j) - \ln((R_2)_{ij} + \Upsilon_j) \right) - \Upsilon_j \left(\ln((R_1)_{ij} + \mathrm{X}_i) - \ln((R_2)_{ij} + \mathrm{X}_i) \right) \right).$$

Here

$$R_1 = \sqrt{R\,{\cdot}^2 + \mathrm{Z}_1{\cdot}^2}, \quad R_2 = \sqrt{R\,{\cdot}^2 + \mathrm{Z}_2{\cdot}^2},$$

and operations involve element-wise powers and multiplications as indicated by the $\cdot$. Using the notation in Chen and Liu [2018], and ignoring $\sum_j (-1)^{i+j+1}$, the summand is

$$\left((\mathrm{Z}_1 (CM5)_{ij} - \mathrm{Z}_2 (CM6)_{ij}) - \mathrm{X}_i \left((CM3)_{ij} - (CM4)_{ij} \right) - \Upsilon_j \left((CM1)_{ij} - (CM2)_{ij} \right) \right).$$

Now notice that $(CM3)_{ij} - (CM4)_{ij}$ is a logarithmic difference (and also for $(CM1)_{ij} - (CM2)_{ij}$). Thus, the differences can be replaced by

$$CMX = \ln \frac{\mathrm{X} + R_1}{\mathrm{X} + R_2}, \text{ and } CMY = \ln \frac{\Upsilon + R_1}{\Upsilon + R_2}. \tag{3.41}$$

Moreover,

$$CM5Z = \mathrm{Z}_1 \arctan \frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_1 (R_1)_{ij}}, \text{ and } CM6Z = \mathrm{Z}_2 \arctan \frac{(\mathrm{X}\Upsilon)_{ij}}{\mathrm{Z}_2 (R_2)_{ij}}.$$

Hence, the summand of the triple sum is replaced by

$$(((CM5Z)_{ij} - (CM6Z)_{ij}) - \mathrm{X}_i (CMY)_{ij} - \Upsilon_j (CMX)_{ij}) = CM_{ij}.$$

```
   Input: See Table 3.1 for details;
   $gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
   prob_params required parameters
   Output: Array That
1  Extract parameters from prob_params ;
2  Initialize zero arrays: $T$ and That for $T$ and $\hat{T}$;
3  Sizes: $nX = s_x + \mathbf{max}(p_{x_{\mathrm{L}}}, p_{x_{\mathrm{R}}})$, $nY = s_y + \mathbf{max}(p_{y_{\mathrm{L}}}, p_{y_{\mathrm{R}}})$;
4  Form distance arrays $X$ and $Y$ according to (3.30);
5  Form $X2 = X.^2$, $Y2 = Y.^2$, $XY = X(:).*Y$ and $R = X2(:) + Y2$;
6  for $r = 1 : n_z$ do
7      Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
8      Calculate slice response at first station: $g = \mathtt{gravity}(z_1, z_2, X, Y, XY, R)$;
9      for $j = [1 + p_{y_L} : s_y + p_{y_L}.s_y + p_{y_R} : -1 : 21 : p_{y_L}]$ do
10         Extract $\mathbf{r}_j$ from $g$ : use (3.26);
11         Augment column of $T$, use (3.36) ;
12     end
13     Take 2DFFT of $T$: $\mathrm{That}(:,:,r) = \mathbf{fft2}(T)$ ;
14 end
```

Algorithm 3: That = sym_BTTBFFT(gsx, gsy, z_blocks, prob_params) Transform of padded symBTTB matrix. Function gravity.

```
Input: Depth coordinates $z_1$ and $z_2$ for the slice;
$X$: Distances of $x-$coordinates from station 1 size nx+1;
$Y$: Distances of $y-$coordinates from station 1 size ny+1;
$XY$: the product $X(:)\cdot *Y$ which is a matrix of size $(nx+1)\times(ny+1)$;
$R$: the matrix of size $(nx+1)\times(ny+1)$ of entries $X(:)\cdot^2$ and $Y\cdot^2$;
Output: Response vector $g$ of length $nxny$;
$[p,q] = \textbf{size}(R)$;
$nx = p-1; ny = q-1$;
$R_1 = \textbf{sqrt}(R+z_1^2)$;
$R_2 = \textbf{sqrt}(R+z_2^2)$;
$CMX = (\textbf{log}((X(:)+R_1)\cdot/(X(:)+R_2)))\cdot *Y$;
$CMY = (\textbf{log}((Y+R_1)\cdot/(Y+R_2)))\cdot *X(:)$;
$CM5Z = \textbf{atan2}(XY, R_1 z_1) z_1$;
$CM6Z = \textbf{atan2}(XY, R_2 z_2) z_2$;
$CM56 = CM5Z - CM6Z$;
$CM = (CM56 - CMY - CMX)\gamma$;
$g = -(CM(1:nx, 1:ny) - CM(1:nx, 2:q) - CM(2:p, 1:ny) + CM(2:p, 2:q))$;
```

Algorithm 4: $g = \texttt{gravity}(z_1, z_2, X, Y, XY, R)$
Entries of sensitivity matrix G for the `gravity` problem.

Clearly X_1, X_2 are entries from X, and Υ_1, Υ_2 are entries from Y. Thus, given the definition (3.23), and assuming that X and Y are stored in row vectors, matrices $XY = X(:)\cdot *Y$ and $R^2 = X(:)\cdot^2 + Y\cdot^2$ can be formed. These are of size $(n_x+1)\times(n_y+1)$, are independent of the z coordinates, and substantial computation is saved by only calculating XY and R^2 once for all slices. Each slice only requires the calculation of one row of the matrix. Since these are based on matrices, the double sum for multiple coordinates is calculated by shifting each ij matrix to the right in i and right in j with the appropriate sign, and obtaining the entire sum in one line by correct indexing into the matrices. Suppose that CM has size $(n_x+1)\times(n_y+1)$. Then the matrix

$$g = CM(1:n_x, 1:n_y) - CM(1:n_x, 2:n_y+1) - CM(2:n_x+1, 1:n_y) + CM(2:n_x+1, 2:n_x+1)$$

can be reshaped as a row vector

$$\tilde{h}(a_1, b_1, 0) = -\gamma g(:).$$

Thus, the first row of depth block r is calculated by an evaluation of (2.12) for all coordinate contributions to station 1 in one step. Note that the simplification (3.41) is a further optimization of the calculation of entries for G as compared to that given in Chen and Liu [2018]. The details of the use and application of the of the `gravity` problem in the context of forward algorithms for symBTTB matrices are provided in Algorithms 2-3 with the `gravity` function in Algorithm 4.

3.4.2 Magnetic Kernel

The improvement of the calculation of the `gravity` kernel as discussed in Chen and Liu [2018] is now extended to the calculation of the `magnetic` kernel, under the assumption that there is no remanence magnetization or self-demagnetization, so that the magnetization vector is parallel to the Earth's magnetic field. The total field is assumed to be measured in nano Teslas; introducing a scaling factor 10^9 in the definitions. Although the `magnetic` kernel is not symmetric, its discretization does lead to a BTTB matrix, and hence the discussion of Section 3.2.4 is relevant. Applying the simplifications of Rao and Babu [1991] for the evaluation of the `magnetic` kernel contribution, and using their notation [Rao and Babu, 1991, eq. (3)]

$$\tilde{h}(a, b, 0)_{pq} = \tilde{H}(G_1 \ln F_1 + G_2 \ln F_2 + G_3 \ln F_3 + G_4 F_4 + G_5 F_5). \tag{3.42}$$

The constants $g_i = \tilde{H} G_i$ depend on the volume orientation and `magnetic` constants, Rao and Babu [1991], and it is now assumed that x points North and y points East. Taking advantage of the notation in (3.40), and noting all terms are arrays, then with element-wise

```
    Input: See Table 3.1 for details;
    $gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
    prob_params required parameters
    Output: BTTB real matrix $G$ of size $m \times n$.
1   Extract parameters from prob_params ;
2   Calculate constants $g_i = \tilde{H} G_i$ for (3.42), [Rao and Babu, 1991, (3)];
3   Initialize zero arrays: $G$, $Gr$ and row and column cell arrays, Figure 3.2;
4   Sizes: $nX = s_x + \mathbf{max}(p_{x_\mathrm{L}}, p_{x_\mathrm{R}})$, $nY = s_y + \mathbf{max}(p_{y_\mathrm{L}}, p_{y_\mathrm{R}})$;
5   Form distance arrays $X$ and $Y$ according to (3.34);
6   Form $X2 = X.^2$, $Y2 = Y.^2$, and $R = X2(:) + Y2$;
7   for $r = 1 : n_z$ do
8       Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
9       Calculate $grow\{1\} = \tilde{h}(11)_{pq}$, $1 \le p \le nX, 1 \le q \le nY$;
10      for $j = 2 : s_y + p_{y_L}$ do
11          Calculate $grow\{j\} = \tilde{h}(1j)_{p1}$, $1 \le p \le nX$;
12      end
13      Calculate : $gcol\{1\} = \tilde{h}(ij)_{11}$, $1 \le i \le nX, 1 \le j \le nY$;
14      for $q = 2 : s_y + p_{y_R}$ do
15          Calculate : $gcol\{q\} = \tilde{h}(i1)_{1q}$, $1 \le i \le nX$;
16      end
17      for $j = p_{y_L} + 1 : -1 : 2$ do
18          Generate $Gjr$: using $gcol\{1\}$ and $grow\{j\}$, (3.32) for $R$ in (3.33);
19      end
20      for $q = 1 : s_y + p_{y_R}$ do
21          Generate $Gqr$: using $gcol\{q\}$ and $grow\{1\}$, (3.31) for $R$ in (3.33);
22      end
23      for $j = p_{y_L} + 2 : -1 : 2, 1 : s_y + p_{y_L}$ do
24          Generate $Grj$: using $gcol\{1\}$ and $grow\{j\}$, (3.32) for $C$ in (3.33);
25      end
26      Build $Gr$ in (3.33) using $C$ and $R$;
27      Assign: $Gr$ to $r^{\mathrm{th}}$ block of $G$;
28  end
```

Algorithm 5: $G =$ BTTB$(gsx, gsy, z_blocks,$ prob_params$, D, I, H)$
Entries of padded BTTB matrix, Figure 3.2. Function magnetic.

```
Input: See Table 3.1 for details;
$gsx$, $gsy$, $z\_blocks$ : grid spacing in $x$ and $y$ and $z$ coordinates;
prob_params required parameters;
$D$, $I$, $H$ declination, inclination and intensity of magnetization
Output: Array $\mathrm{That}$
1  Extract parameters from prob_params ;
2  Calculate constants $g_i = \tilde{H}G_i$ for (3.42), [Rao and Babu, 1991, (3)];
3  Initialize zero arrays: $T$ and $\mathrm{That}$ for $T$ and $\hat{T}$;
4  Initialize zero arrays for and row and column cell arrays, see Figure 3.2;
5  Sizes: $nX = s_x + \mathbf{max}(p_{x_\mathrm{L}}, p_{x_\mathrm{R}})$, $nY = s_y + \mathbf{max}(p_{y_\mathrm{L}}, p_{y_\mathrm{R}})$;
6  Form distance arrays $X$ and $Y$ according to (3.34);
7  Form $X2 = X\cdot^2$, $Y2 = Y\cdot^2$, and $R = X2(:) + Y2$;
8  for $r = 1 : n_z$ do
9      Set $z_1 = z\_blocks(r)$, $z_2 = z\_blocks(r+1)$;
10     Calculate $grow\{1\} = \tilde{h}(11)_{pq}$, $1 \le p \le nX$, $1 \le q \le nY$;
11     for $j = 2 : s_y + p_{y_L}$ do
12         Calculate $grow\{j\} = \tilde{h}(1j)_{p1}$, $1 \le p \le nX$;
13     end
14     Calculate : $gcol\{1\} = \tilde{h}(ij)_{11}$, $1 \le i \le nX$, $1 \le j \le nY$;
15     for $q = 2 : s_y + p_{y_R}$ do
16         Calculate : $gcol\{q\} = \tilde{h}(i1)_{1q}$, $1 \le i \le nX$;
17     end
18     for $j = p_{y_L} + 1 : p_{y_L} + s_y$ do
19         Augment column of $T$, $gcol\{1\}$ and $grow\{j\}$, (3.32) with (3.37) ;
20     end
21     for $q = s_y + p_{y_R} : -1 : 2$ do
22         Augment column of $T$, $gcol\{q\}$ and $grow\{1\}$, (3.31) with (3.37) ;
23     end
24     for $j = 1 : p_{y_L}$ do
25         Augment column of $T$, $gcol\{1\}$ and $grow\{j\}$, (3.32) with (3.37) ;
26     end
27     Take 2DFFT of $T$: $\mathrm{That}(:,:,r) = \mathbf{fft2}(T)$;
28 end
```

Algorithm 6: $\mathrm{That} = \mathtt{BTTBFFT}(gsx, gsy, z\_blocks, \texttt{prob\_params}, D, I, H)$ Transform of padded BTTB matrix, Figure 3.2. Function `magnetic`.

```
**Input:** Depth coordinates $z_1$ and $z_2$ for the slice;
$X$: Distances of $x-$coordinates from station ;
$Y$: Distances of $y-$coordinates from station ;
$R$: Matrix of entries $X(:)\cdot^2$ and $Y\cdot^2$;
$gc$ vector of constants, [Rao and Babu, 1991, 3];
**Output:** Response vector $g$ of length $(\ell+1)(k+1)$;
$\ell = \textbf{length}(X) - 1$;$k = \textbf{length}(Y) - 1$;
$R_1 = \textbf{sqrt}(R + z_1^2)$;
$R_2 = \textbf{sqrt}(R + z_2^2)$;
$F_1 = ((R_2(1:\ell,1:k) + X(1:\ell)) \cdot /(R_1(1:\ell,1:k) + X(1:\ell))).*((R_1(2:\ell+1,1:k) + X(2:\ell+1)) \cdot /(R_2(2:\ell+1,1:k) + X(2:\ell+1))) \cdot *((R_1(1:\ell+1,2:k+1) + X(1:\ell)) \cdot /(R_2(1:\ell+1,2:k+1) + X(1:\ell))) \cdot *((R_2(2:\ell+1,2:k+1) + X(2:\ell+1)) \cdot /(R_1(2:\ell+1,2:k+1) + X(2:\ell+1)))$;
$F_2 = ((R_2(1:\ell,1:k) + Y(1:k)) \cdot /(R_1(1:\ell,1:k) + Y(1:k))) \cdot *((R_1(2:\ell+1,1:k) + Y(1:k)) \cdot /(R_2(2:\ell+1,1:k) + Y(1:k))) \cdot *((R_1(1:\ell+1,2:k+1) + Y(2:k+1)) \cdot /(R_2(1:\ell+1,2:k+1) + Y(2:k+1))) \cdot *((R_2(2:\ell+1,2:k+1) + Y(2:k+1)) \cdot /(R_1(2:\ell+1,2:k+1) + Y(2:k+1)))$;
$F_3 = ((R_2(1:\ell,1:k) + z2) \cdot /(R_1(1:\ell,1:k) + z1)) \cdot *((R_1(2:\ell+1,1:k) + z1) \cdot /(R_2(2:\ell+1,1:k) + z2)) \cdot *((R_1(1:\ell+1,2:k+1) + z1) \cdot /(R_2(1:\ell+1,2:k+1)+z2)) \cdot *((R_2(2:\ell+1,2:k+1)+z2) \cdot /(R_1(2:\ell+1,2:k+1)+z1))$;
$F_4 = \textbf{atan2}(X(2:\ell+1)z_2, R_2(2:\ell+1,2:k+1) \cdot *Y(2:k+1)) - \textbf{atan2}(X(1:\ell)z_2, R_2(1:\ell+1,2:k+1) \cdot *Y(2:k+1)) - \textbf{atan2}(X(2:\ell+1)z_2, R_2(2:\ell+1,1:k) \cdot *Y(1:k)) + \textbf{atan2}(X(1:\ell)z_2, R_2(1:\ell,1:k) \cdot *Y(1:k)) - \textbf{atan2}(X(2:\ell+1)z_1, R_1(2:\ell+1,2:k+1) \cdot *Y(2:k+1)) + \textbf{atan2}(X(1:\ell)z_1, R_1(1:\ell+1,2:k+1) \cdot *Y(2:k+1)) + \textbf{atan2}(X(2:\ell+1)z_1, R_1(2:\ell+1,1:k) \cdot *Y(1:k)) - \textbf{atan2}(X(1:\ell)z_1, R_1(1:\ell,1:k) \cdot *Y(1:k))$;
$F_5 = \textbf{atan2}(Y(2:k+1)z_2, R_2(2:\ell+1,2:k+1) \cdot *X(2:\ell+1)) - \textbf{atan2}(Y(2:k+1)z_2, R_2(1:\ell+1,2:k+1) \cdot *X(1:\ell)) - \textbf{atan2}(Y(1:k)z_2, R_2(2:\ell+1,1:k) \cdot *X(2:\ell+1)) + \textbf{atan2}(Y(1:k)z_2, R_2(1:\ell,1:k) \cdot *X(1:\ell)) - \textbf{atan2}(Y(2:k+1)z_1, R_1(2:\ell+1,2:k+1) \cdot *X(2:\ell+1)) + \textbf{atan2}(Y(2:k+1)z_1, R_1(1:\ell+1,2:k+1) \cdot *X(1:\ell)) + \textbf{atan2}(Y(1:k)z_1, R_1(2:\ell+1,1:k) \cdot *X(2:\ell+1)) - \textbf{atan2}(Y(1:k)z_1, R_1(1:\ell,1:k) \cdot *X(1:\ell))$;
$g = (gc(1) * \textbf{log}(F_1) + gc(2) * \textbf{log}(F_2) + gc(3) * \textbf{log}(F_3) + gc(4) * F_4 + gc(5) * F_5)$;
$g = g(:)$;
```

Algorithm 7: $g = \text{magnetic}(z_1, z_2, X, Y, R, gc))$
Entries of sensitivity matrix G for the `magnetic` problem.

operations as appropriate the variables F_i are given by

$$\begin{aligned}
F_1 &= \frac{(\mathrm{P}^2_{11}+\mathrm{X}_1)(\mathrm{P}^1_{21}+\mathrm{X}_2)(\mathrm{P}^1_{12}+\mathrm{X}_1)(\mathrm{P}^2_{22}+\mathrm{X}_2)}{(\mathrm{P}^1_{11}+\mathrm{X}_1)(\mathrm{P}^2_{21}+\mathrm{X}_2)(\mathrm{P}^2_{12}+\mathrm{X}_1)(\mathrm{P}^1_{22}+\mathrm{X}_2)}\\
F_2 &= \frac{(\mathrm{P}^2_{11}+\Upsilon_1)(\mathrm{P}^1_{21}+\Upsilon_1)(\mathrm{P}^1_{12}+\Upsilon_2)(\mathrm{P}^2_{22}+\Upsilon_2)}{(\mathrm{P}^1_{11}+\Upsilon_1)(\mathrm{P}^2_{21}+\Upsilon_1)(\mathrm{P}^2_{12}+\Upsilon_2)(\mathrm{P}^1_{22}+\Upsilon_2)}\\
F_3 &= \frac{(\mathrm{P}^2_{11}+\mathrm{Z}_2)(\mathrm{P}^1_{21}+\mathrm{Z}_1)(\mathrm{P}^1_{12}+\mathrm{Z}_1)(\mathrm{P}^2_{22}+\mathrm{Z}_2)}{(\mathrm{P}^1_{11}+\mathrm{Z}_2)(\mathrm{P}^2_{21}+\mathrm{Z}_1)(\mathrm{P}_{12}\cdot^2+\mathrm{Z}_1)(\mathrm{P}^1_{22}+\mathrm{Z}_2)}\\
F_4 = \arctan\frac{\mathrm{X}_2\mathrm{Z}_2}{\mathrm{P}^2_{22}\Upsilon_2} &- \arctan\frac{\mathrm{X}_1\mathrm{Z}_2}{\mathrm{P}^2_{12}\Upsilon_2} - \arctan\frac{\mathrm{X}_2\mathrm{Z}_2}{\mathrm{P}^2_{21}\Upsilon_1} + \arctan\frac{\mathrm{X}_1\mathrm{Z}_2}{\mathrm{P}^2_{11}\Upsilon_1} -\\
&\arctan\frac{\mathrm{X}_2\mathrm{Z}_1}{\mathrm{P}^1_{22}\Upsilon_2} + \arctan\frac{\mathrm{X}_1\mathrm{Z}_1}{\mathrm{P}^1_{12}\Upsilon_2} + \arctan\frac{\mathrm{X}_2\mathrm{Z}_1}{\mathrm{P}^1_{21}\Upsilon_1} - \arctan\frac{\mathrm{X}_1\mathrm{Z}_1}{\mathrm{P}^1_{11}\Upsilon_1}\\
F_5 = \arctan\frac{\Upsilon_2\mathrm{Z}_2}{\mathrm{P}^2_{22}\mathrm{X}_2} &- \arctan\frac{\Upsilon_2\mathrm{Z}_2}{\mathrm{P}^2_{12}\mathrm{X}_1} - \arctan\frac{\Upsilon_1\mathrm{Z}_2}{\mathrm{P}^2_{21}\mathrm{X}_2} + \arctan\frac{\Upsilon_1\mathrm{Z}_2}{\mathrm{P}^2_{11}\mathrm{X}_1} -\\
&\arctan\frac{\Upsilon_2\mathrm{Z}_1}{\mathrm{P}^1_{22}\mathrm{X}_2} + \arctan\frac{\Upsilon_2\mathrm{Z}_1}{\mathrm{P}^1_{12}\mathrm{X}_1} + \arctan\frac{\Upsilon_1\mathrm{Z}_1}{\mathrm{P}^1_{21}\mathrm{X}_2} - \arctan\frac{\Upsilon_1\mathrm{Z}_1}{\mathrm{P}^1_{11}\mathrm{X}_1}.
\end{aligned}$$

As for the `gravity` kernel, the calculation of (3.42) can, therefore, use (3.40) to calculate X, Y and $R\cdot^2$ once for all slices. Minor computational savings may be made by calculating for example $R_1 + \mathrm{X}_1$ within the calculations for $\tilde{h}$ but these are not calculations that can be made independent of the given slice. It may appear also that one could calculate ratios $\mathrm{X}\cdot/\Upsilon$, with modification of the calculations for F_4 and F_5, but the stable calculation of the $\arctan$ requires the ratios as given. Otherwise sign changes in the numerator or denominator passed to $\arctan$ can lead to changes in the obtained angle. In MATLAB we use **atan2** rather than **atan** for improved stability in the calculation of the angle. The details of the use and application of the of the `magnetic` problem in the context of the forward algorithms for BTTB matrices are provided in Algorithms 5-6 with the `magnetic` function in Algorithm 7.

3.5 Numerical Validation

The fast and efficient methods for generating both the symmetric and non symmetric kernels relating to `gravity` and `magnetic` problems are now validated. Specifically, the computational cost of direct calculation of the entries of the matrix G that are required for matrix multiplications is compared with the entries that are required for the transform implementation of the multiplications; Algorithms 2 and 3 are compared with all entries calculated using Algorithm 4, and Algorithms 5 and 6 with all entries calculated using Algorithm 7, for the symmetric gravity, and non symmetric `magnetic` kernels, respectively. The 12 problem sizes considered are detailed in Table 3.2. They are generated by taking the smallest size with $(s_x, s_y, , n_z) = (25, 15, 2)$, and then scaling each dimension by 1 to 12 for the test cases. For padding cases with $p = 0\%$ and $p = 5\%$ across x and y dimensions, and rounded to the nearest integer, are compared. Thus, $m = s_x s_y$, and $n = \lfloor(1+p)s_x\rfloor\lfloor(1+p)s_y\rfloor n_z$. All computations use MATLAB release 2019b implemented on a desktop computer with an Intel(R) Xeon (R) Gold 6138 processor (2.00GHz) and 256GB RAM.

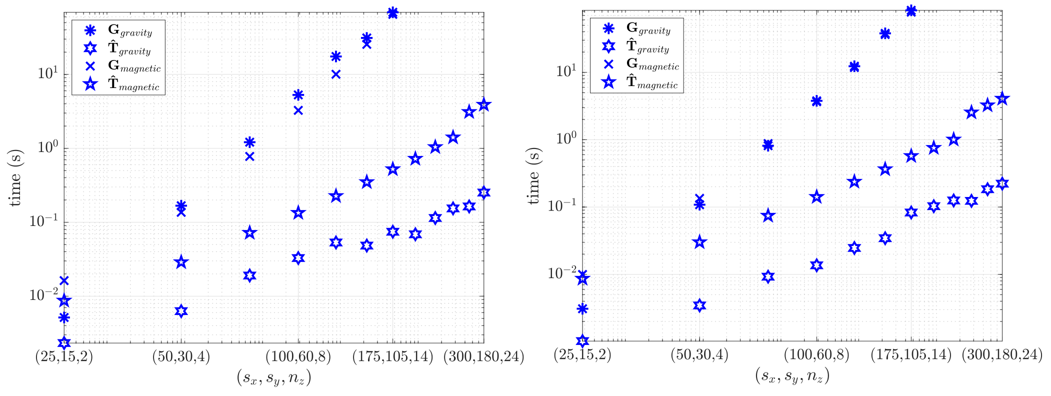
Also, note that the last two columns of Table 3.2 report the estimated memory requirement to store the arrays G and $\hat{T}$, which is independent of whether this is for the `gravity` or the `magnetic` problem. These are the results without padding, but the difference between the padded and unpadded case is insignificant in comparison to the memory requirements for each of these arrays. For the problem of size $m \times n$, matrix G has mn entries, corresponding to $8mn$ bytes and complex array $\hat{T}$ uses approximately $8m$ entries for each depth layer, for a total of $8mn_z = 8n$ entries or $64n$ bytes, here using that one floating point number uses 8 bytes and noting that 1 byte is 10^{-9}GB.

In the results, the kernels generated by Algorithms 2, 5, 3, and 6 are referenced as $G_{\texttt{gravity}}$, $G_{\texttt{magnetic}}$, $T_{\texttt{gravity}}$, and $T_{\texttt{magnetic}}$ respectively. These values are plotted on a "log-log" scale in Figure 3.3, without padding in Figure 3.3a and with padding in Figure 3.3b, in which the problem sizes are given as relevant triples on the $x-$axis. The problem cases from 8 to 12 for the direct calculation of G are too large to fit in memory on the given computer. It can be seen that the generation of G is effectively indepen-

Table 3.2: Dimensions of the volume used in the experiments labeled as problems 1 to 12 corresponding to scaling each dimension in $(25, 15, 2)$ by the problem number (Prob.) and increasing m by a factor 8 for each row. In the last two columns the memory requirements, in Gigabytes (GB), calculated in MATLAB, where the entries for rows 7 to 12 are the estimates given by MATLAB using try zeros(m,n), which gives an exception for matrices that are too large for storage in the given environment and reports the estimated requirements in GB.

Prob.	$(s_x, s_y, , n_z)$	m	n	n ($p = 0.05\%$)	GB G	GB $\hat{T}$
1	$(25, 15, 2)$	375	750	918	.000225	.000005
2	$(50, 30, 4)$	1500	6000	7616	.007200	.000037
3	$(75, 45, 6)$	3375	20250	24402	.054675	.000127
4	$(100, 60, 8)$	6000	48000	58080	.230400	.000303
5	$(125, 75, 10)$	9375	93750	113710	.703125	.000594
6	$(150, 90, 12)$	13500	162000	199200	17.4960	.001028
7	$(175, 105, 14)$	18375	257250	310730	35.2	.001634
8	$(200, 120, 16)$	24000	384000	464640	68.7	.002441
9	$(225, 135, 18)$	30375	546750	662450	123.7	.003478
10	$(250, 150, 20)$	37500	750000	916320	209.5	.004774
11	$(275, 165, 22)$	45375	998250	1206500	337.5	.006358
12	$(300, 180, 24)$	54000	1296000	1568200	521.4	.008258

dent of the `gravity` or `magnetic` kernels; $G_{\texttt{gravity}}$, $G_{\texttt{magnetic}}$ are comparable. But the requirement to calculate extra entries for the non-symmetric `magnetic` kernel is also seen; $T_{\texttt{gravity}} < T_{\texttt{magnetic}}$. On the other hand, the significant savings in generating just the transform matrices, as indicated by timings $T_{\texttt{gravity}}$, and $T_{\texttt{magnetic}}$, as compared to $G_{\texttt{gravity}}$, and $G_{\texttt{magnetic}}$ is evident. There is a considerable computational advantage to the use of the transform for calculating the required components that are needed for evaluating matrix-vector products for these structured kernel matrices.



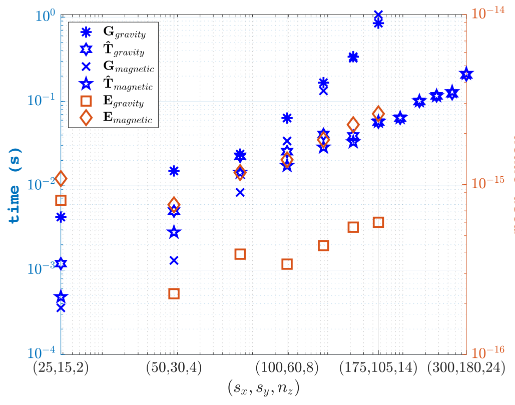
(a) $p = 0$ (b) $p = 0.05$

Figure 3.3: Running times for generating G using Algorithms 2 and 5 and $\hat{T}$ using Algorithms 3 and 6 with 0% padding in Figure 3.3a, and with 5% padding in Figure 3.3b.

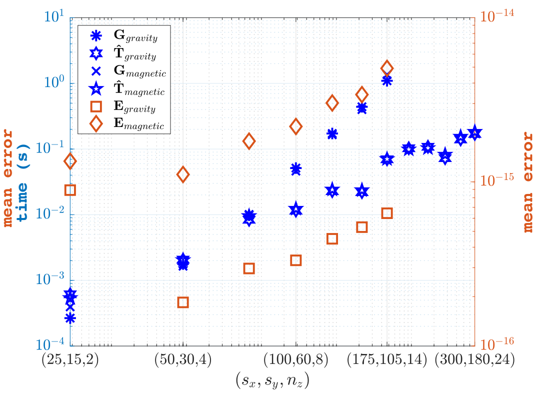
Of greater significance is the comparison of the computational cost of direct matrix multiplications, $\mathbf{b} = G\mathbf{u}$ and $\mathbf{d} = G^T\mathbf{v}$, as compared with the transform implementations for these products. As noted already, $\mathbf{u} \in \mathcal{R}^{n_x n_y n_z}$ is consistently partitioned into n_z blocks, $\mathbf{u}_r \in \mathcal{R}^{n_x n_y}$, $1 \leq r \leq n_z$, with

$$G\mathbf{u} = \sum_{r=1}^{n_z} G^{(r)}\mathbf{u}_r, \text{ and } G^T\mathbf{v} = \begin{bmatrix} \left(G^{(1)}\right)^T \mathbf{v} \\ \left(G^{(2)}\right)^T \mathbf{v} \\ \vdots \\ \left(G^{(n_z)}\right)^T \mathbf{v} \end{bmatrix},$$

where $\mathbf{v} \in \mathcal{R}^{s_x s_y}$. 100 copies of vectors $\mathbf{u} \in \mathcal{R}^n$ and $\mathbf{v} \in \mathcal{R}^m$ are randomly generated and the mean times for calculating the products over all 100 trials, for each problem size, are recorded. The differences over all trials in the generation of $\mathbf{b}$ and $\mathbf{d}$ obtained directly for $G_{\texttt{gravity}}$ and $G_{\texttt{magnetic}}$, and by Algorithm 1 for $T_{\texttt{gravity}}$ and $T_{\texttt{magnetic}}$, are also recorded. Then, $E_{\texttt{gravity}}$ and $E_{\texttt{magnetic}}$ are the mean values of the relative 2-norm of the difference between the results produced by $G_{\texttt{gravity}}$ versus $T_{\texttt{gravity}}$, and for $G_{\texttt{magnetic}}$ versus $T_{\texttt{magnetic}}$, respectively, for both forward and transpose operations. The results are illustrated in Figures 3.4 and 3.5 for the generation of $G\mathbf{u}$ and $G^T\mathbf{v}$, respectively. In each case



(a) $p = 0$

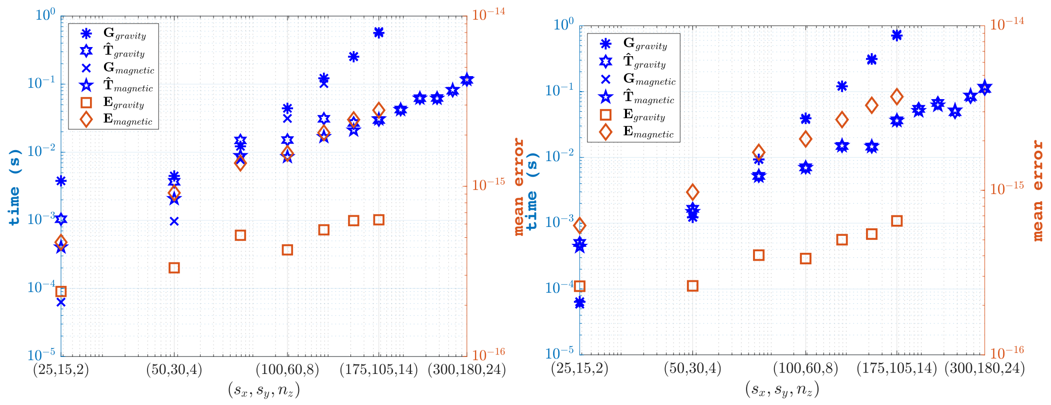


(b) $p = 0.05$

Figure 3.4: Mean running times over 100 trials for forward multiplication using $G_{\texttt{gravity}}$, $G_{\texttt{magnetic}}$, $T_{\texttt{gravity}}$, and $T_{\texttt{magnetic}}$ (left y-axis) and mean errors over 100 trials $E_{\texttt{gravity}}$ and $E_{\texttt{magnetic}}$ for forward multiplication using $G_{\texttt{gravity}}$ versus $T_{\texttt{gravity}}$, and $G_{\texttt{magnetic}}$ versus $T_{\texttt{magnetic}}$ respectively (right y-axis) are shown for 0% padding in Figure 3.4a, and 5% padding in Figure 3.4b.

the timing is reported on the left $y-$axis and the error on the right y-axis. Again all plots are on the "log-log" scale, and Figures 3.4a and 3.5a are without padding, but Figures 3.4b and 3.5b are with padding. Figures 3.4 and Figure 3.5 show significant reductions in mean running time when implemented without the direct calculation of the matrices. Moreover, the results are comparable, $E_{\texttt{gravity}} \lesssim 10\epsilon$ for both forward and transpose operations, and $E_{\texttt{magnetic}} \lesssim 10^2\epsilon$, where ϵ is the machine accuracy. Thus, in all cases, $T_{\texttt{gravity}}$ and $T_{\texttt{magnetic}}$ show a significant reduction in mean running time for large problems, and allow much larger systems to be represented. Indeed, the largest test case for $T_{\texttt{gravity}}$ and $T_{\texttt{magnetic}}$ is by no means a limiting factor, and it is possible to represent much larger kernels.

Remark 3.5.1 (MATLAB **fft2**). *Note that the* MATLAB ***fft2*** *function determines an optimal algorithm for a given problem size. On the first call for a given problem size,* ***fft2*** *uses the function* ***fftw*** *to determine optimal parameters for the Fourier transform. Thus, the first time* ***fft2*** *is called generally takes longer than subsequent instances. This effect is mitigated*



(a) $p = 0$ (b) $p = 0.05$

Figure 3.5: Mean running times over 100 trials for transpose multiplication using $G_{\texttt{gravity}}$, $G_{\texttt{magnetic}}$, $T_{\texttt{gravity}}$, and $T_{\texttt{magnetic}}$ (left y-axis) and mean errors over 100 trials $E_{\texttt{gravity}}$ and $E_{\texttt{magnetic}}$ for transpose multiplication using $G_{\texttt{gravity}}$ versus $T_{\texttt{gravity}}$, and $G_{\texttt{magnetic}}$ versus $T_{\texttt{magnetic}}$ respectively (right y-axis) are shown for 0% padding in Figure 3.5a, and 5% padding in Figure 3.5b.

*by first removing the variable **dwisdom** within **fftw**, and then setting the planner within **fftw** to exhaustive. After running a single call to **fft2**, the resulting **dwisdom** is then saved. Then for each trial, the appropriate stored values for **dwisdom** are loaded before each use of **fft2**. Hence the results are not contaminated by artificially high costs of the first run of **fftw** for each problem case.*

3.5.1 Data Availability and Software Package

The software consists of the main functions to calculate the BTTB and symBTTB matrices, with padding, and the circulant matrices T that are needed for the 2DFFT. Also provided is a simple script to test the algorithms using the `gravity` and `magnetic` kernels. All the software for the algorithms is open source and available at `https://github.com/renautra/FastBTTB`, and described at `https://math.la.asu.edu/~rosie/research/bttb.html`. Provided are the scripts that are used to generate the results presented in the paper. The variables used in the codes are described in

Table 2.1 and Table 3.1. The `TestingScript`.m is easily modified to generate new examples and can be tested within different hardware configurations and versions of MATLAB. A safety test for memory usage in generating large scale examples is provided at the initialization of each problem size, so that problems too large to fit in memory will not be used in generating the matrix G directly. The presented implementation assumes uniform grid sizes in all dimensions, but using depth layers of different heights is an easy modification, through the change in the input coordinate vector z_blocks.

3.6 Conclusions

A general discussion on Toeplitz and circulant structure relating to fast Fourier based multiplication has been presented. It was shown that careful derivation of `gravity` and `magnetic` models admitts BTTB structure, and provides efficient generation of the kernel matrices for both the symmetric and non-symmetric cases with padding around the domain. It was shown that the fast convolution multiplication can be extended for the resulting rectangular kernel matrices. The presented results validate that there is a considerable reduction in computational cost for generation of these matrices, and for forward and transpose operations with these matrices. Software implementing the algorithms has been made available.

Chapter 4

A RANDOMIZED PRECONDITIONING SKETCH

Overview of Chapter

In this chapter, sketching of matrices using randomization techniques is introduced. A new algorithm, RPS, will be shown to significantly reduce computational time compared to the `LSRN` method by combining dimensionality reduction via a left randomized sketch with a right preconditioning substitution. Analysis will show that using the `RPS` method creates an extremely well-conditioned system, along with the associated error bounds for the resulting solution. The `RPS` method will also be shown to provide regularization via truncation based on choice of truncation parameter, and demonstrations will validate the use of the `RPS` algorithm in comparison to `LSRN`. These methods will be applied to real data, and for this practical data set a preferable estimate for the truncation parameter will be shown for the `RPS` and `LSRN` algorithms. The `LSRN` algorithm, however, will require approximately three times as much computational time compared to the `RPS` algorithm for this truncation size.

While randomization techniques have been around for decades, as reviewed in the text of Motwani and Raghavan [1995], algorithms toward dimensionality reduction via randomization have only more recently gained attention and application, Bárdossy and Hörning [2016], Gower and Richtárik [2015], Halko et al. [2011], Lin et al. [2017], Mahoney [2011], Meng et al. [2014], Vatankhah et al. [2018a,b], Woodruff [2014]. Sketching is a technique that takes advantage of recent developments in randomization, and has been introduced as a technique for dimensionality reduction, Halko et al. [2011], Wang [2015]. Two major directions for applying dimensionality reduction through sketching have been considered.

The first is sketching the matrix A, generally to obtain a rank-k decomposition of A that allows the problem to be solved more efficiently, Gower and Richtárik [2017], Halko et al. [2011], Mahoney [2011], Wang [2015], Vatankhah et al. [2018a,b]. The second method applies the sketch directly to the linear system $A\boldsymbol{x} \approx \boldsymbol{b}$ to obtain a reduced size surrogate model, Gower and Richtárik [2015], Lin et al. [2017].

Here, a random Gaussian left sketch, S, is introduced. Shortfalls are addressed when attempting to apply a random Gaussian sketch via the transform $S\boldsymbol{y} = \boldsymbol{x}$, and possible alternatives are presented. It is shown how a randomized left sketching technique can be combined with right preconditioning methods to reduce the dimensions of both the model parameter and measurement data spaces. Theoretical results supporting the new left sketch with right preconditioning are presented.

As noted in Chapter 3, the sensitivity matrices for `gravity` and `magnetic` problems exhibit block Toeplitz Toeplitz block (`BTTB`) structure provided that the data measurement positions are uniform and carefully related to the grid defining the volume discretization. The `2DFFT` can be used to provide fast multiplication as shown. Hence, here the focus is on a method that takes advantage of this fast matrix multiplication.

Note that Zhang and Wong [2015] have used the `BTTB` structure for fast computations with the sensitivity matrix, and employed this within an algorithm for the inversion of `gravity` data using a smoothing regularization, allowing for variable heights of the individual depth layers in the domain. They also applied optimal preconditioning for the `BTTB` matrices using the approach of Chan and Jin [2007]. Their approach was optimized by Chen and Liu [2018] but only for efficient forward `gravity` modeling and with a slight modification in the way that the matrices for each depth layer of the domain are defined using the approximation of the forward integral equation. Here the extension of the use of the FFT approach to the inversion of `magnetic` data is considered. Both inversion problems are solved using a truncation method based on randomization. This method is then validated in combination with BTTB multiplication via the `2DFFT` for the solution of the linear systems that arise in focused inversion of `gravity` and `magnetic` data using both synthetic and real data.

Overview of main scientific contributions. This chapter provides a comprehensive study of the application of the `2DFFT` in focusing inversion algorithms for `gravity` and `magnetic` potential field data sets in combination with the development of new randomization techniques. Specifically, the main contributions are as follows. (i) A detailed review of randomization techniques in application to a discrete linear system by left multiplication; (ii) The extension of this randomization technique to include a right preconditioning substitution to produce a novel randomized preconditioning sketch (`RPS`); (iii) Presentation of theoretical analysis and computational properties for the `RPS` method; (iv) The extension of the `RPS` method to utilize the `2DFFT` for all forward multiplications with the sensitivity matrix, or its transpose; (v) Presentation of numerical experiments that confirm that the `RPS` algorithm is more efficient for the inversion of `gravity` and `magnetic` data sets than an existing method, `LSRN`, Meng et al. [2014]; (vi) Finally, all conclusions are confirmed by application on a practical data set, demonstrating that the methodology is suitable for focusing inversion of large scale data sets and can provide parameter reconstructions with more than 1M variables.

4.1 Sketching A

Rapid assimilation of data and the trend to increase the number of model parameters have led to increasingly large problems. Large inverse problems are prohibitively computationally expensive to solve, both in terms of memory and time. Applying dimensionality reduction can reduce a problem to a manageable size. A sketch, Halko et al. [2011], Wang [2015], reduces dimensionality while attempting to represent the dominant information of a matrix. If $A \in \mathbb{R}^{m\times n}$ and **sketching matrix** $S \in \mathbb{R}^{n\times s}$, where $s \ll \mathbf{min}(m, n)$, then

$C = AS$ is a **sketch** of A. The sketch size $s = k + p$ uses an oversampling parameter p to ensure C provides a rank k approximation of A. Note that a sketch can also be applied by left multiplication for $S \in \mathbb{R}^{m\times s}$ as $C = S^TA$ and as $C = S_1^TAS_2$, Gower and Richtárik [2017], Wang [2015], for consistent sketching matrices S_1 and S_2. The sketch C then contains a combination of either the rows and/or columns of A.

Suppose the SVD of A is given by $A = U\Sigma V^T$, and $A_k = U_k\Sigma_k V_k^T$ where Σ_k contains the first k columns and rows of Σ. It is well known, Hansen [1987], Hansen et al. [2013], that A_k is the closest rank-k approximation to A with $\|A - A_k\|_2 = \sigma_{k+1}$. The solution $\mathbf{x}_k = V_k\Sigma_k^\dagger U_k^T\boldsymbol{b}$ of $A_k\mathbf{x}_k \approx \mathbf{b}$ is then easy to obtain. Computing the SVD of a large matrix can, however, become very computationally expensive, and in some cases altogether impossible. Suppose $C \in \mathbb{R}^{m\times s}$ is a matrix that satisfies the **low-rank approximation** property, Mahoney [2011],

$$\|CC^\dagger A - A\|_2 \leq (1+\epsilon)\|A_k - A\|_2, \tag{4.1}$$

where $C^\dagger$ is the pseudo-inverse of C, and $1 + \epsilon \geq 1$ is small, generally close to 1. Then, C is a good rank-k approximation of A if the dominant k singular terms of A are represented in C. The oversampling parameter p is chosen to ensure statistically that (4.1) holds with high probability, [Sarlos, 2006, Theorem 14].

An alternative criterium that ensures that the dominant features of A are represented in C is the **subspace embedding property**, Wang [2015], Woodruff [2014]. Specifically, this requires preservation of the length of a vector under projection

$$\|S^TA\mathbf{x}\|_2 \approx \|A\mathbf{x}\|_2 \quad \text{or} \quad \|S^TA^T\mathbf{z}\|_2 \approx \|A^T\mathbf{z}\|_2, \tag{4.2}$$

for any $\mathbf{x} \in \mathbb{R}^n$ or $\mathbf{z} \in \mathbb{R}^m$. This property becomes less straightforward when the sketch $C = S_1^TAS_2$ is used. Since both dimensions of C have been reduced the transformations $S_2\mathbf{y} = \mathbf{x}$ or $S_1^T\mathbf{w} = \mathbf{z}$, for some $\mathbf{y}, \mathbf{w} \in \mathbb{R}^s$ are needed.

Several different sketching matrices have been shown to enforce properties (4.1) and (4.2) with high probability. These include the use of random Gaussian matrices, randomized Hadamard transforms, and count sketches Wang [2015]. The **random Gaussian sketch matrix** $S = \frac{1}{\sqrt{s}}\Omega$, where Ω contains randomly sampled values $\omega_{ij} \sim \mathcal{N}(0, 1)$, is simple to implement and allows a flexible choice of s that is typically smaller in comparison to other methods, Mahoney [2011], Wang [2015]. Here, $\sim N(0, 1)$ denotes a random variable sampled from a standard normalized Gaussian distribution with mean 0 and variance 1. A randomized singular value decomposition (RSVD) is covered in more detail relating to inversion of `gravity` and `magnetic` data in Chapter 5

4.2 Directly Applying Sketching to Solve Systems

Many methods, Gower and Richtárik [2017], Halko et al. [2011], Mahoney [2011], Wang [2015], Vatankhah et al. [2018a,b], Voronin et al. [2015], use the sketch of A to obtain a surrogate system that is more computationally efficient to solve. Given the linear system

$$A\mathbf{x} \approx \mathbf{b}, \tag{4.3}$$

consider the left, right and full sketches given by $S^TA\mathbf{x} \approx S^T\mathbf{b}$, $AS\mathbf{y} \approx \mathbf{b}$, and $S_1^TAS_2\mathbf{y} \approx S_1^T\mathbf{b}$, respectively, yielding the general form

$$C\mathbf{y} \approx \mathbf{c}, \tag{4.4}$$

where $\boldsymbol{c} = S^T\boldsymbol{b}$ for left and full sketches. Regardless of the choice of s, C is not guaranteed to be square and nonsingular, or even well-conditioned. To find a suitable estimate for $\mathbf{y}$, it may, therefore, be necessary to either right precondition C, use regularization, or both precondition and regularize. Further, the resulting sketched system may still be computationally expensive to solve. Although $\boldsymbol{y}$ may be accurate as a solution of (4.4), the accuracy of $\boldsymbol{x}$ as a solution of (4.3), when using right and full sketches, still depends on the accuracy of the projection $\mathrm{x} = S\mathrm{y}$ from $\mathbb{R}^s$ to $\mathbb{R}^n$. For example, it is immediate that if S is a random Gaussian sketch matrix, then the projection to $\mathbb{R}^n$ is random and inaccurate.

4.2.1 Left Sketch

First consider a left sketch applied to the system $A\boldsymbol{x} = \boldsymbol{b}$, producing the surrogate system $S^T A\boldsymbol{x} = S^T\boldsymbol{b}$. Algorithm 8 shows an implementation of the left sketch in MATLAB using a Gaussian sketch matrix and solving the surrogate system with the MATLAB implementation of **lsqr**, which is based on Paige and Saunders [1982].

```
Input: $A$, $\boldsymbol{b}$, and sketch size $s$,
       with $[m, n] = \textbf{size}(A)$
$S = \textbf{randn}(m, s)$;
$\boldsymbol{x} = \textbf{lsqr}(S^T A, S^T\boldsymbol{b})$;
```

Algorithm 8: Left sketch

```
Input: $A$, $\boldsymbol{b}$, and sketch size $s$,
       with $[m, n] = \textbf{size}(A)$
$S = \textbf{randn}(n, s)$;
$\boldsymbol{y} = \textbf{lsqr}(AS, \boldsymbol{b})$;
$\boldsymbol{x} = S\boldsymbol{y}$;
```

Algorithm 9: Right sketch

Although one dimension of the system is reduced when a sketch is applied on the left of A, the solution $\boldsymbol{x}$ is computed in its original dimension.

4.2.2 Naive Right Sketch

Next consider $AS\boldsymbol{y} \approx \boldsymbol{b}$ obtained by the substitution $\boldsymbol{x} = S\boldsymbol{y}$. While this allows implementation of the sketch $C = AS$, the projection $\mathrm{x} = S\boldsymbol{y}$ is disastrous when using a random Gaussian sketch matrix. Algorithm 9 presents a simple MATLAB implementation of the naive right sketch.

4.2.3 LSRN

One method that gives insight into reducing the model dimensionality and providing an accurate projection is LSRN, Meng et al. [2014]. For matrix A which is either significantly overdetermined ($m \gg n$) or underdetermined ($m \ll n$), the LSRN algorithm uses a projection based preconditioning approach. The sketch size $s > \textbf{min}(m, n)$ is chosen to ensure the full rank of A is represented in C, [Meng et al., 2014, Theorem 4.1]. For example, for $n \ll m$, a left sketch $C = S^T A$, of size $(n+p) \times n$, is produced using a random Gaussian sketch matrix S, with $s = n+p$. Next, the SVD $C = U_C \Sigma_C V_C^T$ is produced, and the rank-k SVD is used to form the preconditioning matrix $N = (V_C)_k (\Sigma_C)_k^{-1}$. While CN is used to obtain an initial vector for use with **lsqr**, Paige and Saunders [1982], the extremely overdetermined problem $AN\mathbf{y} \approx \mathbf{b}$ is solved, and the projection $\mathbf{x}_{LSRN} = N\mathbf{y}$ provides the desired solution. LSRN is shown in Algorithm 10 for $m \gg n$, and Algorithm 11 for $m \ll n$. Due to the concentration of the extreme singular values, [Meng et al., 2014, Theorem 4.4], in high probability the system is well-conditioned, with $\kappa_2(AN) \approx 1$. Note that a similar preconditioning approach is used by [Drineas et al., 2012] to obtain statistical leverage scores. Further, as noted in [Meng et al., 2014, Lemma 3.1], range$(NN^TC^T) =$ range(C^T) ensures that a least squares solution is still found. Thus, LSRN provides a viable method that combines sketching and preconditioning techiques. Since LSRN requires $m \ll n$ or $n \ll m$, if m and n are both large LSRN requires more computational resources than a standard regularization approach, and is thus of limited applicability.

```
Input: $A \in \mathbb{R}^{m \times n}$ with rank $k$,
       $\boldsymbol{b} \in \mathbb{R}^m$, and oversampling
       factor $p$, with $m \gg n$
$s = n + p$;
$S =$ randn$(m, s)$;
$C = S^T A \in \mathbb{R}^{s \times n}$;
$[\tilde{}, \Sigma, V] =$ svd$(C,$ 'econ'$)$;
$N = V_k \Sigma_k^{-1}$;
$\boldsymbol{y} =$ lsqr$(AN, \boldsymbol{b})$;
$\boldsymbol{x} = N\boldsymbol{y}$;
```

Algorithm 10: LSRN ($m \gg n$)

```
Input: $A \in \mathbb{R}^{m \times n}$ with rank $k$,
       $\boldsymbol{b} \in \mathbb{R}^m$, and oversampling
       factor $p$, with $m \ll n$
$s = m + p$;
$S =$ randn$(n, s)$;
$C = AS \in \mathbb{R}^{m \times s}$;
$[U, \Sigma, \tilde{}] =$ svd$(C,$ 'econ'$)$;
$M = U_k \Sigma_k^{-1}$;
$\boldsymbol{x} =$ lsqr$(M^T A, M^T \boldsymbol{b})$;
```

Algorithm 11: LSRN ($m \ll n$)

4.3 Randomized Preconditioning Sketch

In order to reduce the model dimensionality and keep the computational costs down we propose a randomized preconditioning sketch (RPS), Hogue and Renaut [2020], with sketch size $s = k + p$, and $k \ll \mathbf{min}(m, n)$. As with LSRN, a random Gaussian left sketch is applied to give $C = S^T A \in \mathbb{R}^{s\times n}$. Then in contrast to choosing $N = (V_C)_k(\Sigma_C)_k^{-1}$ from the truncated SVD of C, we take the truncated SVD of CC^T to form N based on the identity $C^\dagger = C^T(CC^T)^\dagger = C^T V_{CC^T}\Sigma_{CC^T}^\dagger U_{CC^T}^T$ derived from (A.1). Since $s \ll n$ and $CC^T \in \mathbb{R}^{s\times s}$, its SVD, $U_{CC^T}\Sigma_{CC^T}V_{CC^T}^T$, is found efficiently. Hence assuming rank$(CC^T) \geq k$, we can choose $N = C^T(V_{CC^T})_k(\Sigma_{CC^T})_k^{-1}$, producing the overdetermined preconditioned system

$$CN\mathbf{y} \approx S^T\mathbf{b}. \tag{4.5}$$

The solution $\boldsymbol{x} = N\boldsymbol{y}$ is obtained at significantly reduced cost because $CN \in \mathbb{R}^{s\times k}$. Further, multiplying from the right only requires matrix-vector multiplications, where now $\boldsymbol{x} = C^T(V_{CC^T})_k(\Sigma_{CC^T})_k^{-1}\mathbf{y}$. Pseudo-code for RPS is given in Algorithm 12. Computational analysis is discussed carefully in Section 4.4.

```
Input: $A$, $\boldsymbol{b}$, sketch size $s$, and power iteration
       parameter $q$, with $[m, n] = \mathbf{size}(A)$
$S = \mathbf{randn}(m, s)$;
Calculate $C \in \mathbb{R}^{s\times n}$: $C = S^T A$;
$\boldsymbol{c} = \boldsymbol{b}$;
for $i = 1 : q$ do
    $C = (CA)A^T$;        // power iteration
    $\boldsymbol{c} = A(A^T\boldsymbol{c})$;
end
Calculate SVD for $CC^T \in \mathbb{R}^{s\times s}$:
 $[U, \Sigma, \tilde{}\,] = \mathbf{svd}(CC^T)$;
Solve for $\boldsymbol{y} \in \mathbb{R}^k$: $\boldsymbol{y} = U_k^T(S^T\boldsymbol{c})$;
Compute $\boldsymbol{x} \in \mathbb{R}^n$: $\boldsymbol{x} = (C^T(U_k(\Sigma_k^{-1}\boldsymbol{y})))$;
```

Algorithm 12: RPS

Note that the preconditioned surrogate model is solved directly by a single matrix multiplication, provided through Lemma 4.3.1 and Theorem 4.3.2. Error bounds are also presented for the projected solution. Since the error is shown to rely on the decay of singular values, we also include a power iteration such that $S^T(AA^T)^qA\boldsymbol{x} = S^T(AA^T)^q\boldsymbol{b}$. In the following analysis, the impact of floating-point errors is in evaluating terms is ignored and negligible.

Lemma 4.3.1. *Hogue and Renaut [2020] Suppose $A \in \mathbb{R}^{m\times n}$ has rank k and $S \in \mathbb{R}^{m\times s}$ is a standard Gaussian sketch matrix with $s = k+p \ll \textbf{min}(m,n)$. Suppose $U_{CC^T}\Sigma_{CC^T}V_{CC^T}^T$ is the SVD of CC^T where $C = S^TA \in \mathbb{R}^{s\times n}$ and let $N = C^T(V_{CC^T})_k(\Sigma_{CC^T})_k^{-1} \in \mathbb{R}^{n\times k}$. Then $CN = (U_C)_k$ is column orthogonal, where $U_C\Sigma_C V_C^T$ is the SVD of C.*

Proof. Since $V_{CC^T} = U_{CC^T} = U_C$ and $\Sigma_{CC^T} = \Sigma_C^2$ by Lemma A.0.1,

$$\begin{aligned} CN &= CC^T(V_{CC^T})_k(\Sigma_{CC^T})_k^{-1} = U_C\Sigma_C^2U_C^T(U_C)_k(\Sigma_C)_k^{-2} = U_C\Sigma_C^2\begin{bmatrix} \mathrm{I}_k \\ 0 \end{bmatrix}(\Sigma_C)_k^{-2} \\ &= U_C\begin{bmatrix} (\Sigma_C)_k^2 \\ 0 \end{bmatrix}(\Sigma_C)_k^{-2} = U_C\begin{bmatrix} \mathrm{I}_k \\ 0 \end{bmatrix} = (U_C)_k. \end{aligned}$$

Hence CN is column orthogonal. □

Theorem 4.3.2. *Hogue and Renaut [2020] Suppose $A \in \mathbb{R}^{m\times n}$ has rank at least $k \ll \textbf{min}(m,n)$, $S \in \mathbb{R}^{m\times s}$ is a standard Gaussian sketch matrix with $s = k+p \ll \textbf{min}(m,n)$, with $p \geq 4$, and as in Lemma 4.3.1 $N = C^T(V_{CC^T})_k(\Sigma_{CC^T})_k^{-1}$. Let $\boldsymbol{x}_k = A_k^\dagger\boldsymbol{b}$ be the least squares solution of $A_k\boldsymbol{x}_k \approx \boldsymbol{b}$, where A_k is the best rank k approximation of A, and $\tilde{\boldsymbol{x}}_k = N\boldsymbol{y}$ be the projection of the least squares solution of the right preconditioned system $S^TAN\boldsymbol{y} \approx S^T\boldsymbol{b}$. Defining $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ and $(\Delta A)_k = A - (AP)_k$ for the projector $P = C^T(CC^T)^{-1}C$, and using $\sigma_1 > \sigma_2 > \cdots > \sigma_k > \sigma_{k+1}$ as the first $k+1$ singular values of A, then*

1. *The overdetermined least squares problem $CN\boldsymbol{y} \approx S^T\boldsymbol{b}$ with system matrix $CN \in \mathbb{R}^{s\times k}$ is well-conditioned, and $\kappa_2(CN) = 1$ and $\boldsymbol{y} = (U_{CC^T})_k^TS^T\boldsymbol{b}$.*

2. *The solution $\tilde{\boldsymbol{x}}_k = N\boldsymbol{y} = C_k^\dagger S^T \boldsymbol{b}$ is a least squares solution of* (4.3)*, and satisfies*

$$\frac{\|\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k\|_2}{\|\boldsymbol{x}_k\|_2} \leq \frac{\|(\Delta A)_k\|_2}{\sigma_k - \|(\Delta A)_k\|_2} + \frac{\|(\Delta A)_k\|_2}{\sigma_k - \|(\Delta A)_k\|_2 - \sigma_{k+1}} \left(1 + \frac{\sigma_1 \|\boldsymbol{r}\|_2}{(\sigma_k - \|(\Delta A)_k\|_2)\|A\boldsymbol{x}_k\|_2}\right)$$

with probability greater than $1 - 6e^{-p}$, where

$$\|(\Delta A)_k\|_2 \leq \left(2 + 16\sqrt{1 + \frac{k}{p+1}}\right)\sigma_{k+1} + \frac{8\sqrt{s}}{p+1}\left(\sum_{j \geq k+1} \sigma_j^2\right)^{\frac{1}{2}}. \tag{4.6}$$

Proof.

1. It is immediate by Lemma 4.3.1 that $\kappa_2((CN)^T CN) = \kappa_2((U_C)_k^T (U_C)_k) = 1$. Thus the least squares problem given by (4.5) is well-conditioned, by [Golub and Van Loan, 2013, Theorem 5.3.1 and related discussion], summarized in Theorem B.1.1. By column orthogonality and (4.6), $\boldsymbol{y} = (U_C)_k^T S^T \boldsymbol{b} = (U_{CC^T})_k^T S^T \boldsymbol{b}$.

2. Using $N = C^T (V_{CC^T})_k (\Sigma_{CC^T})_k^\dagger$ and Lemma A.0.1,

$$N(U_{CC^T})_k^T = V_C \Sigma_C U_C^T (U_C)_k ((\Sigma_C)_k^\dagger)^2 (U_C)_k^T = (V_C)_k (\Sigma_C)_k^\dagger (U_C)_k^T = C_k^\dagger,$$

   where the last equality holds by (A.2). Thus, $\tilde{\boldsymbol{x}}_k = N\boldsymbol{y} = N(U_{CC^T})_k^T S^T \boldsymbol{b} = C_k^\dagger S^T \boldsymbol{b}$ and $\tilde{\boldsymbol{x}}_k$ is a least squares solution of $C_k \tilde{\boldsymbol{x}}_k \approx S^T \boldsymbol{b}$, or $(S^T A)_k \tilde{\boldsymbol{x}}_k \approx S^T \boldsymbol{b}$.

   Now, using the projector $P = C^T (CC^T)^{-1} C$, and noting that the truncated SVD $(AP)_k$ is the closest rank k approximation to AP, we apply [Halko et al., 2011, Theorem 9.3], summarized in Theorem B.2.1, using $Y = C^T = A^T S$ to show

$$\|A - (AP)_k\|_2 \leq \sigma_{k+1} + \|A(I - P)\|_2,$$

   since for a projector $P^T = P$, $I - P$ is the orthogonal projector to P, and $\|A^T\|_2 = \|A\|_2$. Furthermore, by [Halko et al., 2011, Corollary 10.9], summarized in Corollary B.2.2,

$$\|A(I - P)\|_2 \leq \left(1 + 16\sqrt{1 + \frac{k}{p+1}}\right)\sigma_{k+1} + \frac{8\sqrt{s}}{p+1}\left(\sum_{j \geq k+1} \sigma_j^2\right)^{\frac{1}{2}}.$$

Combining these two results yields the given bound (4.6) for $\|(\Delta A)_k\|_2$.

Finally, the relative least squares error, which is given by [Hansen, 1987, Theorems 3.2-3.4], can be written as

$$\begin{aligned}\frac{\|\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k\|_2}{\|\boldsymbol{x}_k\|_2} \leq &\frac{\|(\Delta A)_k\|_2}{\sigma_k - \|(\Delta A)_k\|_2} \\ &+ \frac{\|(\Delta A)_k\|_2}{\sigma_k - \|(\Delta A)_k\|_2 - \sigma_{k+1}}\left(1 + \frac{\sigma_1\|\boldsymbol{r}\|_2}{(\sigma_k - \|(\Delta A)_k\|_2)\|A\boldsymbol{x}_k\|_2}\right)\end{aligned}$$

as long as there is a distinct gap between σ_k and σ_{k+1}, and thus the result follows.

$\square$

Note that using $N = C^T V_k \Sigma_k^{-1}$ with $\boldsymbol{y} = U_k^T S^T \boldsymbol{b}$ is equivalent to the truncation approach $\boldsymbol{y}_k = \Phi U^T S^T b$, where Φ is diagonal with elements $\phi_i = 1$ for $i \leq k$ and $\phi_i = 0$ for $i > k$. Since Theorem 4.3.2 requires a distinct gap $\sigma_k > \sigma_{k+1}$, consider the case where $\sigma_k \gtrapprox \sigma_{k+1}$. Notice that applying the power iteration produces $(AA^T)^q A = U\Sigma^q V^T$ and thus produces $\sigma_k^q > \sigma_{k+1}^q$ for some q. Now the system $(AA^T)^q A\boldsymbol{x} = (AA^T)^q \boldsymbol{b}$ has a distinct gap between σ_k^q and σ_{k+1}^q, and thus Theorem 4.3.2 is applicable.

4.4 Computational Analysis

The RPS algorithm requires the calculation of $C = S^T A$, the product CC^T and the generation of the SVD for CC^T. These are dominated by $\mathcal{O}(mns)$, $\mathcal{O}(ns^2)$ and $\mathcal{O}(s^3)$ `flops`, respectively, Wang [2015]. Using q power iterations adds an additional $\mathcal{O}(2qmns)$ `flops`. Then solution $\tilde{\boldsymbol{x}}_k = N\mathbf{y} = C^T(V_{CC^T})_k(\Sigma_{CC^T})_k^{-1}\mathbf{y}$ is obtained without directly forming matrix N, using k scalar operations for $(\Sigma_{CC^T})_k^{-1}\mathbf{y}$, s inner products of length k for action by $(V_{CC^T})_k$ and n inner products of length s for action by C^T. The dominant cost is then $\mathcal{O}((n+k)s)$. Similar arguments can be made for LSRN, but with $\mathbf{min}(m,n) < s = \mathbf{min}(m,n) + p$, where p is the oversampling factor. In addition, unlike RPS where the preconditioned system is obtained for free, the matrix multiplication for preconditioning in

LSRN is taken into account. LSRN also requires an iterative solver. Supposing that the standard LSQR algorithm is used, the dominant computational cost per iteration is $\mathcal{O}(mn)$ arising from the matrix vector operations $A^T\boldsymbol{u}$ and $A\boldsymbol{v}$, for $\boldsymbol{u} \in \mathbb{R}^m$ and $\boldsymbol{v} \in \mathbb{R}^n$, Paige and Saunders [1982]. Thus the total cost of an $i-$step LSQR is $\mathcal{O}(2mni_1)$. Since m and n are large, the $2mni$ operations dominate the cost of LSQR. These counts are summarized in Table 4.1. Note in all instances that the first order costs of inner products and scalar vector multiplications are ignored.

Table 4.1: The dominant computational cost comparison of solving $A\boldsymbol{x} \approx \boldsymbol{b}$ for $A \in \mathbb{R}^{m\times n}$ with numerical rank k, sketch size $s \ll$ **min**$(m, n) = \tilde{m}$, **max**$(m, n) = \tilde{\tilde{m}}$, oversampling factor p, q power iteration within RPS, and $i \leq k$ iterations of LSQR. Note that the projection is only applicable for LSRN when $m \gg n$.

	Sketch	Precondition	Solution	Projection	Overall
LSRN	$\mathcal{O}((\tilde{m}+p)nm)$	$\mathcal{O}(mnk)$	$\mathcal{O}(\tilde{\tilde{m}}ki)$	$\mathcal{O}(nk)$	$\mathcal{O}((\tilde{m}+p)nm)$
RPS	$\mathcal{O}((2q+1)mns)$	0	$\mathcal{O}(sk)$	$\mathcal{O}(n(s+k))$	$\mathcal{O}((2q+1)mns)$

The memory cost is also reduced through RPS, as shown in Table 4.2. When the size of the problem is too large to be stored in local memeory, the randomized sketch can be applied as a series of vector operations, thereby only holding an $s \times n$ matrix in memory to be updated iteratively, after which the full size system is no longer needed. The largest matrix needed is then either the $s \times n$ sketch C or the $n \times k$ preconditioning matrix N for the final projection. LSRN, however, requires a matrix larger than A by a factor of p.

4.5 Application to Inversion of Geophysics Data

The fast and efficient methods for inversion of potential field data using the `BTTB` structure of the `gravity` and `magnetic` kernel matrices are now validated.

Table 4.2: The dominant memory cost comparison for solving $A\boldsymbol{x} \approx \boldsymbol{b}$ with $A \in \mathbb{R}^{m\times n}$ with numerical rank k, sketch size $s \ll$ **min**$(m,n) = \tilde{m}$, **max**$(m,n) = \tilde{\tilde{m}}$, and oversampling factor p. Note that the projection is only applicable for LSRN when $m \gg n$.

	Sketch	Precondition	Solution	Projection	Overall
LSRN	$\mathcal{O}((\tilde{m}+p)\tilde{\tilde{m}})$	$\mathcal{O}(\tilde{\tilde{m}}k)$	$\mathcal{O}(\tilde{\tilde{m}}k)$	$\mathcal{O}(nk)$	$\mathcal{O}((\tilde{m}+p)m)$
RPS	$\mathcal{O}(2m+ns)$	$\mathcal{O}(sk)$	$\mathcal{O}(sk)$	$\mathcal{O}(ns)$	$\mathcal{O}(2m+ns)$

4.5.1 Stabilized Inversion

The solution of (2.8) is an ill-posed problem; even if G is well-conditioned the problem is underdetermined because $m \ll n$. There is a considerable literature on the solution of this ill-posed problem, see Vatankhah et al. [2020c] for a relevant overview, and specifically the use of the unifying framework for determining an acceptable solution of $\boldsymbol{d} = G\boldsymbol{m}$ by stabilization. Briefly, $\mathbf{m}^*$ is estimated as the minimizer of the nonlinear objective function $\Phi_\alpha(\mathbf{m})$ subject to bound constraints $\mathbf{m}_{\text{min}} \le \mathbf{m} \le \mathbf{m}_{\text{max}}$

$$\mathbf{m}^* = \underset{\mathbf{m}_{\min}\le\mathbf{m}\le\mathbf{m}_{\max}}{\arg\min}\{\Phi_\alpha(\mathbf{m})\} = \underset{\mathbf{m}_{\min}\le\mathbf{m}\le\mathbf{m}_{\max}}{\arg\min}\{\Phi_{\text{d}}(\mathbf{m}) + \alpha^2\Phi_{\text{S}}(\mathbf{m})\}. \tag{4.7}$$

Here α is a regularization parameter which trades off the relative weighting of the two terms $\Phi_{\text{d}}(\mathbf{m})$ and $\Phi_{\text{S}}(\mathbf{m})$, which are respectively the weighted data misfit and stabilizer, given by

$$\Phi_{\text{d}}(\mathbf{m}) = \|\mathbf{W_d}(G\mathbf{m} - \mathbf{d}_{\text{obs}})\|_2^2, \text{ and } \Phi_{\text{S}}(\mathbf{m}) = \|\mathbf{W}_{\text{z}}\mathbf{W}_{\text{L}}(\mathbf{m} - \mathbf{m}_{\text{apr}})\|_2^2.$$

The weighting matrices $\mathbf{W_d}$, $\mathbf{W}_{\text{z}}$ and $\mathbf{W}_{\text{L}}$ are all diagonal. The prior information given by $\mathbf{m}_{\text{apr}}$ is taken to be zero, $\mathbf{m}_{\text{apr}} = \mathbf{0}$, but when initial estimates for the parameter are available, perhaps from physical measurements, note that these can be incorporated into $\mathbf{m}_{\text{apr}}$ as an initial estimate for $\mathbf{m}$. The weighting matrix $\mathbf{W_d}$ has entries $(\mathbf{W_d})_{ii} = 1/\sigma_i$ when the measured data can be given by $\mathbf{d}_{\text{obs}} = \mathbf{d}_{\text{exact}} + \boldsymbol{\eta}$, where $\mathbf{d}_{\text{exact}}$ is the exact but unknown data, and $\boldsymbol{\eta}$ is a noise vector drawn from uncorrelated Gaussian data with variance components σ_i^2.

Whereas stabilizer matrix $\mathbf{W}_{\mathrm{L}}$ in $\mathbf{W} = \mathbf{W}_{\mathrm{z}}\mathbf{W}_{\mathrm{L}}$ depends on $\mathbf{m}$, $\mathbf{W}_{\mathrm{z}}$ is a constant depth weighting matrix and is routinely used in the context of potential field inversion and is imposed to counteract the natural decay of the kernel with depth. With the same column structure as G, $\mathbf{W}_{\mathrm{z}} = \texttt{blockdiag}(\mathbf{W}_{\mathrm{z}}{}^{(1)}, \dots, \mathbf{W}_{\mathrm{z}}{}^{(n_z)})$ where $\mathbf{W}_{\mathrm{z}}{}^{(r)} = (.5(z_r + z_{r-1}))^{-\beta} I_{n_r \times n_r}$, $.5(z_r + z_{r-1})$ is the average depth for depth level r, and β is a parameter that depends on the data set, [Li and Oldenburg, 1996]. Now, diagonal matrix $\mathbf{W}_{\mathrm{L}}$ depends on the parameter vector $\mathbf{m}$ via i^{th} entry given by

$$(\mathbf{W}_{\mathrm{L}})_{ii} = \left((\mathbf{m}_i - (\mathbf{m}_{\text{apr}})_i)^2 + \epsilon^2\right)^{\frac{\lambda-2}{4}}, \quad i = 1 \dots n,$$

where parameter λ determines the form of the stabilization, and focusing parameter $0 < \epsilon \ll 1$ is chosen to avoid division by zero. Using $\lambda = 1$ yields an approximation to the L_1 norm as described in [Wohlberg and Rodríguez, 2007], and is preferred for inversion of potential field data, although the implementation makes it easy to switch to $\lambda = 0$, yielding a solution which is compact, or $\lambda = 2$ for a smooth solution. Based on prior studies $\epsilon^2 = 1e - 9$ is used, [Vatankhah et al., 2017].

4.5.2 Algorithmic Details for Truncation Approaches

Let $W = \mathbf{W}_{\mathrm{z}}\mathbf{W}_{\mathrm{L}}$ and $\boldsymbol{y} = W(\mathbf{m} - \mathbf{m}_{\text{apr}})$. Then $\mathbf{m} = \mathbf{m}_{\text{apr}} + W^{-1}\boldsymbol{y}$ and $\boldsymbol{y}$ provides an update that can be applied iteratively. Now using $\mathbf{W_d}(G\mathbf{m} - \mathbf{d}_{\text{obs}}) = \mathbf{W_d}(G\mathbf{m}_{\text{apr}} + GW^{-1}\boldsymbol{y} - \mathbf{d}_{\text{obs}})$ with $\boldsymbol{r} = G\mathbf{m}_{\text{apr}} - \mathbf{d}_{\text{obs}}$, produces $\mathbf{W_d}GW^{-1}\boldsymbol{y} = \mathbf{W_d}\boldsymbol{r}$ for solving (4.7) without dependance on α. This comprises the iteratively reweighted least squares (IRLS) method introduced in Section 2.4.3, and is summarized in Algorithm 13. The IRLS method is applied for (2.7) with $q = 2$ and $p = 1$.

As shown in Section 2.4.2, this formulation is equivalent to finding the filtered solution

$$\mathbf{y} = \mathbf{W}^{-1}\tilde{V}\Phi\tilde{\Sigma}^{\dagger}\tilde{U}^{T}\tilde{\mathbf{r}} = \mathbf{W}^{-1}\tilde{G}_{\kappa}^{\dagger}\tilde{\mathbf{r}}, \tag{4.8}$$

where $\tilde{G} = \mathbf{W}_{\mathbf{d}}G\mathbf{W}^{-1} = \tilde{U}\tilde{\Sigma}\tilde{V}^{T}$ and Φ is a diagonal matrix containing filter factors. Here truncation filtering is implemented with $\Phi_{ii} = 1, i \leq \kappa$ and $\Phi_{ii} = 0, i > \kappa$. Hence, solution of (4.7) is obtained via truncation for Algorithm 13, whereas standard Tikhonov style regularization, dependent on α, is examined in Chapter 5.

```
Input: $G$, $\mathbf{d}_{\text{obs}}$, $\mathbf{m}_{\text{min}}$, $\mathbf{m}_{\text{max}}$, $W_d$, $\mathbf{W}_{\text{z}}$, and $K_{\text{max}}$
initialize $k = 0$, $\boldsymbol{m}^{(1)} = \mathbf{0}$, $\boldsymbol{y}^{(1)} = \mathbf{1}$, and stopping conditions;
store $W_dG$, $W_d\mathbf{d}_{\text{obs}}$;
while stopping conditions not met do
    $k = k + 1$;
    $\boldsymbol{r}^{(k)} = G\mathbf{m}^{(k)} - \mathbf{d}_{\text{obs}}$;
    $({\mathbf{W}_{\text{L}}}^{(k)})_{ii} = \textbf{diag}((\boldsymbol{y}_{ii}^2 + \epsilon^2)^{-1/4})$;
    $W = {\mathbf{W}_{\text{L}}}^{(k)}\mathbf{W}_{\text{z}}$;
    solve $W_dGW^{-1}\boldsymbol{y} = W_d\boldsymbol{r}^{(k)}$ for $\boldsymbol{y}$;
    $\boldsymbol{m}^{(k+1)} = \boldsymbol{m}^{(k)} + W^{-1}\boldsymbol{y}$;
    enforce $\mathbf{m}_{\text{min}} \leq \boldsymbol{m}^{(k+1)} \leq \mathbf{m}_{\text{max}}$;
    update stopping conditions;
end
```

Algorithm 13: IRLS

The two methods compared for computing (4.8) via truncation are RPS, Algorithm 12 with 1 power iteration, and LSRN, Algorithm 11 (since $m \ll n$). Computations involving the kernel G with BTTB slices are computed using the FFT approach established in Section 3.1. To use the MATLAB **lsqr** function for LSRN, a parameterized anonymous function is used in place of the kernel that makes use of the FFT approach. In the algorithm specifically, superscript k is used to indicate a variable at an iteration k. Truncation parameter κ is replaced by $\kappa^{(k)}$, $\mathbf{W}_{\text{L}}$ by matrix $\mathbf{W}_{\text{L}}^{(k)}$ with entries $(\mathbf{W}_{\text{L}}^{(k)})_{ii} = \left((\mathbf{m}_i^{(k-1)} - \mathbf{m}_i^{(k-2)})^2 + \epsilon^2\right)^{\frac{\lambda-2}{4}}$ and $\mathbf{m} - \mathbf{m}_{\text{apr}}$ by $\mathbf{m} - \mathbf{m}^{(k-1)}$, initialized with ${\mathbf{W}_{\text{L}}}^{(1)} = I$, and $\mathbf{m}^{(0)} = \mathbf{m}_{\text{apr}}$ respectively. Then $\mathbf{y}^{(k)}$ is found as the solution given by (4.8), and $\mathbf{m}^{(k)}$ is the restriction of $\mathbf{y}^{(k)} + \mathbf{m}^{(k-1)}$ to the bound constraints.

In addition, the MATLAB **eig** function is used in place of the **svd** function since only U and Σ are needed. Specifically, given $AA^T = Q\Lambda Q^T$ and $A = U\Sigma V^T$ for orthonormal U and Q and diagonal Λ and Σ, provides the relation $Q\Lambda Q^T = AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = U\Sigma^2 U^T$. Then $Q\Lambda Q^T$ can be transformed to provide U and Σ by sorting the values in Λ.

Also note that for all simulations, the `IRLS` algorithm is iterated to convergence as determined by the χ^2 test for the predicted data,

$$\|\mathbf{W_d}(G\mathbf{m}^{(k)} - \mathbf{d}_{\text{obs}})\|_2^2 \leq m + \sqrt{2m},$$

or

$$\frac{\|\mathbf{W_d}(G\mathbf{m}^{(k)} - \mathbf{d}_{\text{obs}})\|_2^2}{m + \sqrt{2m}} \leq 1. \tag{4.9}$$

If this is not attained for $k \leq K_{\max}$, the iteration is terminated. Noisy data are generated for observed data $\mathbf{d}_{\text{obs}} = \mathbf{d}_{\text{exact}} + \boldsymbol{\eta}$ using

$$\boldsymbol{\eta}_i = (\tau_1|(\mathbf{d}_{\text{exact}})_i| + \tau_2\|\mathbf{d}_{\text{exact}}\|_\infty)\mathbf{e}_i \tag{4.10}$$

where $\mathbf{e}$ is drawn from a Gaussian normal distribution with mean 0 and variance 1. The pairs (τ_1, τ_2) are chosen to provide a signal to noise ratio (`SNR`), as calculated by

$$\texttt{SNR} = 20\log_{10}\frac{\|\mathbf{d}_{\text{exact}}\|_2}{\|\mathbf{d}_{\text{obs}} - \mathbf{d}_{\text{exact}}\|_2}, \tag{4.11}$$

that is approximately constant across the increasing resolutions of the problem. Recorded for all simulations are (i) the values of the relative error $\texttt{RE}^{(k)}$, as defined by

$$\texttt{RE} = \frac{\|\mathbf{m}_{\text{exact}} - \mathbf{m}^{(k)}\|_2}{\|\mathbf{m}_{\text{exact}}\|_2}, \tag{4.12}$$

(ii) the number of iterations to convergence K which is limited to 25 in all cases, (iii) the scaled χ^2 estimate given by (4.9) at the final iteration, and (iv) the time to convergence measured in seconds, or to iteration 25 when convergence is not achieved.

The `IRLS` algorithm needs to use operations with $\tilde{G} = \mathbf{W_d} G \mathbf{W}^{-1}$ rather than G. But this is handled immediately by using suitable component-wise multiplications of the diagonal matrices and vectors as noted in Section 3.1. Specifically,

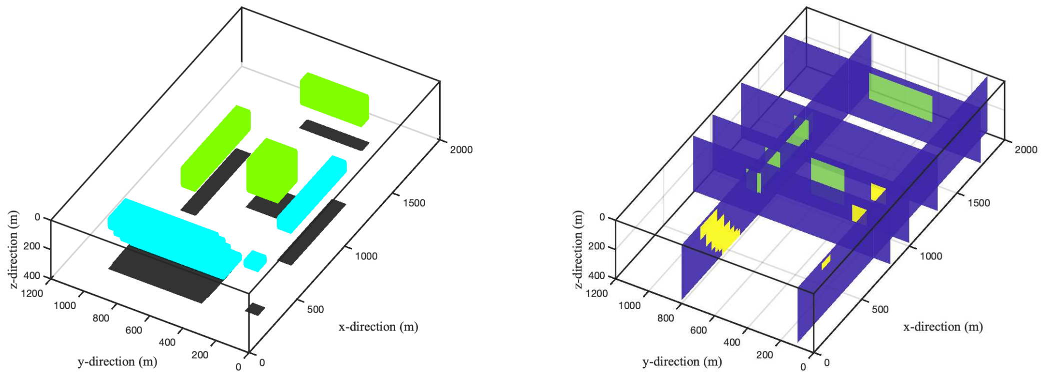
$$\tilde{G}\mathbf{x} = \mathbf{W_d}(G(\mathbf{W}^{-1}\mathbf{x}))$$

and the `2DFFT` is applied for the evaluation of $G\mathbf{w}$ where $\mathbf{w} = \mathbf{W}^{-1}\mathbf{x}$. Then, given $\mathbf{z} = G\mathbf{w}$, a second component-wise multiplication, $\mathbf{W_d}\mathbf{z}$, is applied to complete the process. Within the algorithms, matrix-matrix operations are also required but, clearly, operations $\tilde{G}^{(k)}X$, $(\tilde{G}^{(k)})^T Z$, $Z^T\tilde{G}^{(k)}$ are just loops over the relevant columns (or rows) of the matrices X and Z, with the appropriate weighting matrices provided before and after application of the `2DFFT`.

4.5.3 Implementation Parameter Choices

Diagonal depth weighting matrix $\mathbf{W}_{\mathrm{z}}$ uses $\beta = 0.8$ for the `gravity` problem, and $\beta = 1.4$ for the `magnetic` problem, consistent with recommendations in Li and Oldenburg [1998] and Pilkington [1997], respectively. In order to contrast the performance and computational cost of the `RPS` and `LSRN` algorithms with increasing problem size m, different sizes t of the projected space for the solution are obtained using $t = \textbf{floor}(m/s)$, $s = 40$, 25, 20, 8, 6, and 4, corresponding to increasing t, but also limited by 5000. Truncation parameter $\kappa^{(k)}$ is found using the `GCV` method for $k > 1$, but initialized with appropriately small $\kappa^{(1)} = t/2$. This follows the practice implemented in Vatankhah et al. [2018a], Renaut et al. [2017] for studies using Tikhonov style regularization, and which was based on the recommendation to use a large value for the first regularization parameter, [Farquharson and Oldenburg, 2004], corresponding to a smaller κ when applied for truncation. For comparison, both methods are also implemented without using `GCV`, instead with $\kappa = t$.

4.5.4 Synthetic Data



(a) Iso-surface of the volume structure. (b) Cross-section of the volume structure.

Figure 4.1: The basic volume structure within the domain of size $2000 \times 1200 \times 400$. The extent of each structure is shown by the shadow on the base of the volume. The same structure is used for the results using the padded domain.

A volume structure with a number of boxes of different dimensions, and a six-layer dipping dike is used for the validation of the algorithms. The same structure is used for generation of the `gravity` and `magnetic` potential field data. For `gravity` data the densities of all aspects of the structure are set to 1, with the homogeneous background set to 0. For the `magnetic` data, the dipping dike, one extended well and one very small well have susceptibilities $.06$. The three other structures have susceptibilities set to $.04$. The distinction between these structures with different susceptibilities is illustrated in the illustration of the iso-structure in Figure 4.1a and the cross-section in Figure 4.1b.

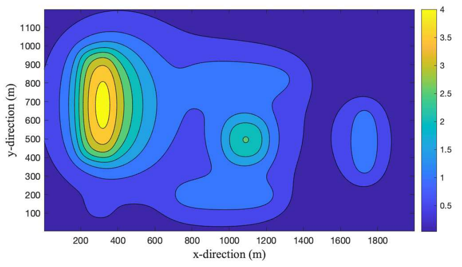
The domain volume is discretized in x, y and z into the number of blocks as indicated by triples (s_x, s_y, n_z) with increasing resolution for increasing values of these triples. They are generated by taking $(s_x, s_y, n_z) = (25, 15, 2)$, and then scaling each dimension by scaling factor $\ell \geq 4$ for the test cases, correspondingly, $s_x s_y = 375$ is scaled by ℓ^2 with increasing ℓ, yielding a minimum problem size with $m = 6000$ and $n = 48000$. The grid sizes are

Table 4.3: Dimensions of the volume used in the experiments with scaling of the small problem size $(25, 15, 2)$ by scale factor ℓ in each dimension. m and n are the dimensions of the measurement vector and the volume domain, respectively, $G \in \mathcal{R}^{m\times n}$. Here $m = s_x s_y = 375\ell^2$ and $n = mn_z$ where $n_x = s_x$ and $n_y = s_y$ without padding. Here, $n_{\text{pad}} = n_x n_y n_z$ is used to denote the volume dimension n with 5% padding, using $n_x = s_x + 2\,\textbf{round}(\text{pad}\, s_x)$ and $n_y = s_y + 2\,\textbf{round}(\text{pad}\, s_y)$ for padding obtained using a percentage, pad, on each side of the domain so that $p_{x_\text{L}} = p_{x_\text{R}} = \textbf{round}(\text{pad}\, s_x)$, and similarly for s_y.

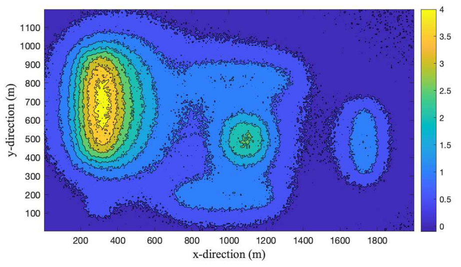
ℓ	(s_x, s_y, n_z)	m	n	n_{pad}	τ_2^{g}	τ_2^{m}	$\texttt{SNR}^{\text{g}}$	$\texttt{SNR}^{\text{m}}$
4	$(100, 60, 8)$	6000	48000	58080	.0138	.0081	24.0	24.0
5	$(125, 75, 10)$	9375	93750	113710	.0147	.0083	24.0	24.0
6	$(150, 90, 12)$	13500	162000	199200	.0133	.0074	24.0	24.0
7	$(175, 105, 14)$	18375	257250	310730	.0133	.0070	24.0	24.0
8	$(200, 120, 16)$	24000	384000	464640	.0133	.0071	24.0	24.1
9	$(225, 135, 18)$	30375	546750	662450	.0133	.0069	24.0	24.0
10	$(250, 150, 20)$	37500	750000	916320	.0132	.0070	24.0	24.0
11	$(275, 165, 22)$	45375	998250	1206500	.0135	.0075	24.0	24.0
12	$(300, 180, 24)$	54000	1296000	1568160	.0135	.0075	24.0	24.0

thus given by the triples $(\Delta_x, \Delta_y, \Delta_z) = (2000/s_x, 1200/s_y, 400/n_z)$. The problem sizes considered for each simulation are detailed in Table 4.3. Padding is given by $\text{pad} = 0\%$ and $\text{pad} = 5\%$ across x and y dimensions. These are rounded to the nearest integer yielding $p_{x_\text{L}} = p_{x_\text{R}} = \textbf{round}(\text{pad}\, s_x)$, and $n_x = s_x + 2\,\textbf{round}(\text{pad}\, s_x)$. n_y is calculated in the same way, yielding $n = (s_x + 2\,\textbf{round}(\text{pad}\, s_x))(s_y + 2\,\textbf{round}(\text{pad}\, s_y))n_z$. Certainly, the choice to use $\text{pad} = 5\%$ is quite large, but is chosen to demonstrate that the solutions obtained using the `2DFFT` are robust to boundary conditions, and thus not impacted by the restriction due to lack of padding or very small padding.

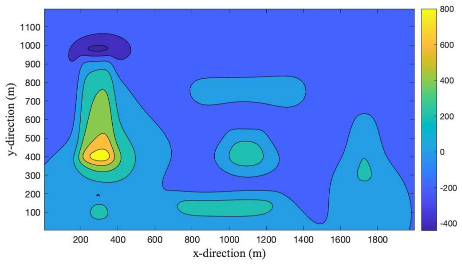
For these structures and resolutions, noisy data are generated as given in (4.10) to yield an `SNR` of approximately 24 across all scales as calculated using (4.11). This results in different choices of τ_1 and τ_2 for each problem size and dependent on the `gravity` or `magnetic` data case, denoted by $(\tau_1^{\mathrm{g}}, \tau_2^{\mathrm{g}})$ and $(\tau_1^{\mathrm{m}}, \tau_2^{\mathrm{m}})$, respectively. In all cases $\tau_1^{\mathrm{g}} = \tau_1^{\mathrm{m}} = .02$ and τ_2 is adjusted. The simulations for the choices of τ_2^{g} and τ_2^{m} for increasing problem sizes are detailed in Table 3.2. Figure 4.2 illustrates the true and noisy data for `gravity` and `magnetic` data, when $\ell = 12$.



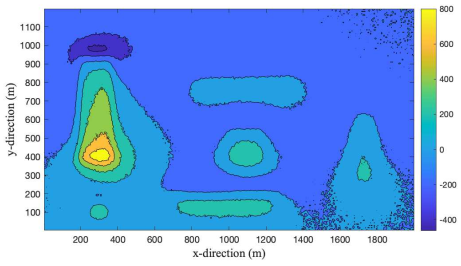
(a) True `gravity` anomaly



(b) Noisy `gravity` anomaly



(c) True `magnetic` anomaly



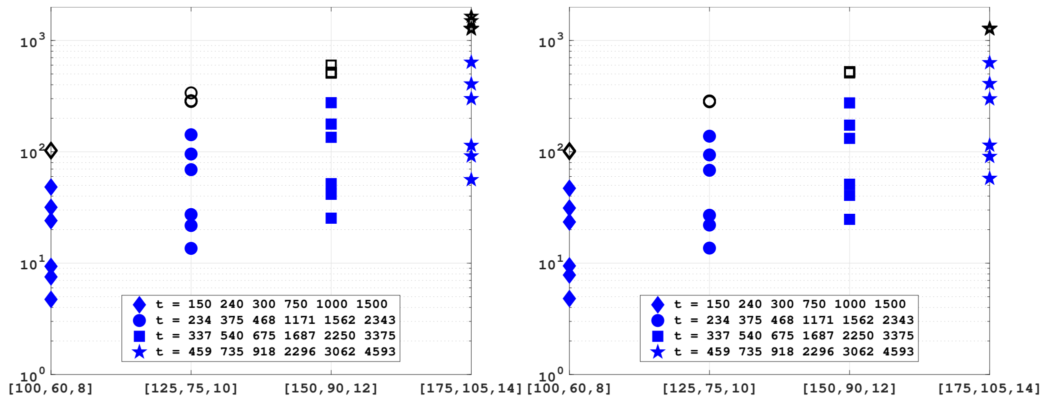
(d) Noisy `magnetic` anomaly

Figure 4.2: The calculated true and noisy anomalies for the volume structure given in Figure 4.1a, where the units are mGal and nT for `gravity` and `magnetic` data, respectively. The anomalies used for the inversion using the padded domain are exactly the same as given here.

4.5.5 Numerical Results

The validation and analysis of the algorithms for the inversion of the potential field data is presented in terms of (i) the cost per iteration of the algorithm (Section 4.5.6), (ii) the total cost to convergence of the algorithm (Section 4.5.7), and (iii) the quality of the obtained solutions, (Section 4.5.8). Supporting quantitative data that summarize the illustrated results are also presented as Tables.

4.5.6 Comparative Cost of RPS and LSRN Algorithms Per IRLS Iteration



(a) Running time `magnetic`. (b) Running time `gravity`

Figure 4.3: Running time in seconds for one iteration of the inversion algorithm for the inversion of `magnetic` and `gravity` data using the `2DFFT`, without padding the volume domain. Problems are of increasing size, as indicated by the $x-$axis for triples $[n_x, n_y, n_z]$ and increasing projection size t ($y-$axis using log scale) determined by fractions of $m = s_x s_y$. In Figures 4.3a and 4.3b the running time for the `LSRN` algorithm and the `RPS` algorithm with one power iteration using $t_p = \textbf{floor}(1.05t)$ (an oversampling percentage 5%), for the `magnetic` and `gravity` problems respectively. `GCV` is used to obtain κ in all cases. In these plots the blue solid symbols represent the timing for one iteration of the `RPS` algorithm and the open black symbols represent the timing for the same simulation using the `LSRN` algorithm. Note that the projection for `LSRN` does not rely on t, and the timings tends to overlap.

Table 4.4: Results for inversion of `gravity` potential field data for the simulations described in Table 4.3 without padding for problem sizes up to $\ell = 7$. The maximum number of iterations is set to 25 in all cases. t_p is the size of the projected space for the `RPS` implementation, whereas the `LSRN` implementation uses the projected space of $1.1m$. Reported are the number of iterations to convergence, K, for convergence as defined by (4.9). The calculated relative error `RE` for the given K are also given.

`gravity`			`RPS (GCV)`		`RPS` (t)		`LSRN (GCV)`		`LSRN` (t)	
ℓ	t	t_p	K	`RE`	K	`RE`	K	`RE`	K	RE
4	150	157	7	0.73	6	0.71	5	0.62	6	0.72
	240	252	5	0.66	5	0.68	5	0.62	4	0.66
	300	315	5	0.66	4	0.65	5	0.62	4	0.66
	750	787	5	0.66	5	0.68	4	0.62	5	0.68
	1000	1050	5	0.66	9	0.79	4	0.62	7	0.76
	1500	1575	9	0.76	10	0.80	4	0.62	12	0.82
5	234	245	7	0.73	6	0.71	6	0.62	6	0.72
	375	393	6	0.69	6	0.71	7	0.96	6	0.72
	468	491	7	0.73	6	0.71	6	0.62	6	0.72
	1171	1229	8	0.74	8	0.77	8	0.71	11	0.81
	1562	1640	9	0.76	10	0.80	7	0.96	11	0.81
	2343	2460	12	0.78	13	0.82	7	0.96	14	0.83
6	337	353	7	0.73	5	0.68	4	0.62	5	0.68
	540	567	6	0.69	6	0.71	4	0.62	5	0.68
	675	708	5	0.66	6	0.71	5	0.62	5	0.68
	1687	1771	10	0.77	12	0.81	6	0.62	12	0.82
	2250	2362	11	0.78	13	0.82	6	0.62	12	0.82
	3375	3543	14	0.80	15	0.83	7	0.96	16	0.84
7	459	481	7	0.73	6	0.71	7	0.96	7	0.76
	735	771	7	0.73	7	0.75	6	0.62	7	0.76
	918	963	7	0.73	7	0.75	6	0.62	7	0.76
	2296	2410	12	0.78	12	0.81	7	0.96	13	0.83
	3062	3215	13	0.79	13	0.82	7	0.96	14	0.83
	4593	4822	17	0.81	16	0.83	14	0.76	17	0.84

Table 4.5: Timing results to convergence for inversion of `gravity` potential field data for the simulations described in Table 4.3 without padding, for problem sizes up to $\ell = 7$. In the last two columns the relative costs of `LSRN` as compared to `RPS`. Values greater than 1 indicate that `RPS` is overall faster. In general, `RPS` is faster for inversion of `gravity` data. As projection size increases, however, the relative efficiency of `RPS` decreases and $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$ decreases towards 1. Results for relative errors and number of iterations are presented in Table 4.4 for `gravity` data.

			$\texttt{Cost}_{\texttt{RPS}}$		$\texttt{Cost}_{\texttt{LSRN}}$		$\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$	
ℓ	t	t_p	GCV	t	GCV	t	GCV	t
4	150	157	40	31	580	600	15	19
	240	252	43	43	574	400	13	9
	300	315	53	42	572	408	11	10
	750	787	136	136	451	506	3	4
	1000	1050	180	314	456	704	3	2
	1500	1575	495	499	469	1217	1	2
5	234	245	123	92	2371	1652	19	18
	375	393	167	150	2723	1640	16	11
	468	491	239	187	2348	1653	10	9
	1171	1229	685	615	3082	3016	5	5
	1562	1640	1039	1037	2737	3024	3	3
	2343	2460	2054	2075	2767	3876	1	2
6	337	353	217	148	2941	2585	14	17
	540	567	299	290	2893	2505	10	9
	675	708	303	364	3780	2496	12	7
	1687	1771	1543	1821	4255	6000	3	3
	2250	2362	2171	2649	3680	5998	2	2
	3375	3543	4177	4399	3505	8109	1	2
7	459	481	444	339	8628	8762	19	26
	735	771	725	642	7369	8774	10	14
	918	963	844	814	7408	8662	9	11
	2296	2410	3911	3546	8672	16032	2	5
	3062	3215	5901	5184	8665	17279	1	3
	4593	4822	12049	10019	17267	21100	1	2

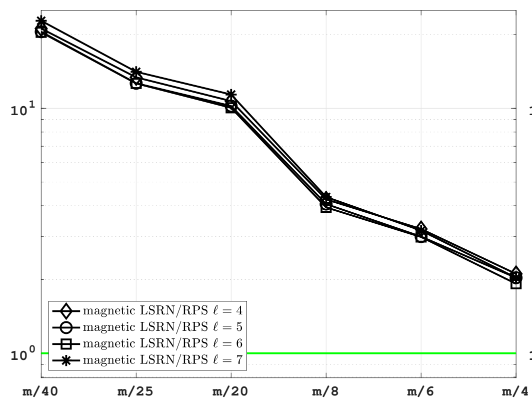
Table 4.6: Results for inversion of `magnetic` potential field data for the simulations described in Table 4.3 without padding for problem sizes up to $\ell = 7$. The maximum number of iterations is set to 25 in all cases. t_p is the size of the projected space for the `RPS` implementation, whereas the `LSRN` implementation uses the projected space of $1.1m$. Reported are the number of iterations to convergence, K, for convergence as defined by (4.9), with $K = 25$ indicating that the simulation did not converge to the given tolerance, marked with $*$. The calculated relative error `RE` for the given K are also given.

`magnetic`			`RPS` (`GCV`)		`RPS` (t)		`LSRN` (`GCV`)		`LSRN` (t)	
ℓ	t	t_p	K	`RE`	K	`RE`	K	`RE`	K	RE
4	150	157	25$*$	0.75	25$*$	0.80	13	0.71	25$*$	0.77
	240	252	25$*$	0.75	25$*$	0.80	12	0.70	25$*$	0.77
	300	315	25$*$	0.75	23	0.80	9	0.68	25$*$	0.77
	750	787	12	0.68	10	0.68	11	0.69	11	0.68
	1000	1050	11	0.67	10	0.68	12	0.70	10	0.67
	1500	1575	10	0.66	9	0.66	11	0.69	8	0.68
5	234	245	25$*$	0.75	25$*$	0.80	14	0.72	25$*$	0.77
	375	393	25$*$	0.75	25$*$	0.80	14	0.72	25$*$	0.77
	468	491	25$*$	0.75	20	0.80	15	0.73	16	0.75
	1171	1229	14	0.70	12	0.69	24	0.75	12	0.69
	1562	1640	13	0.69	11	0.69	25$*$	0.75	11	0.68
	2343	2460	12	0.68	10	0.68	14	0.72	10	0.67
6	337	353	25$*$	0.75	25$*$	0.80	15	0.73	23	0.70
	540	567	25$*$	0.75	17	0.74	15	0.73	16	0.75
	675	708	23	0.75	15	0.73	15	0.73	15	0.74
	1687	1771	14	0.70	11	0.69	25$*$	0.75	11	0.68
	2250	2362	11	0.67	9	0.66	25$*$	0.75	10	0.67
	3375	3543	9	0.66	9	0.66	25$*$	0.75	9	0.67
7	459	481	25$*$	0.75	25$*$	0.80	25$*$	0.75	22	0.81
	735	771	25$*$	0.75	18	0.74	16	0.74	16	0.75
	918	963	25$*$	0.75	15	0.73	25$*$	0.75	15	0.74
	2296	2410	14	0.70	12	0.69	25$*$	0.75	12	0.69
	3062	3215	13	0.69	12	0.69	25$*$	0.75	12	0.69
	4593	4822	13	0.69	11	0.69	25$*$	0.75	12	0.69

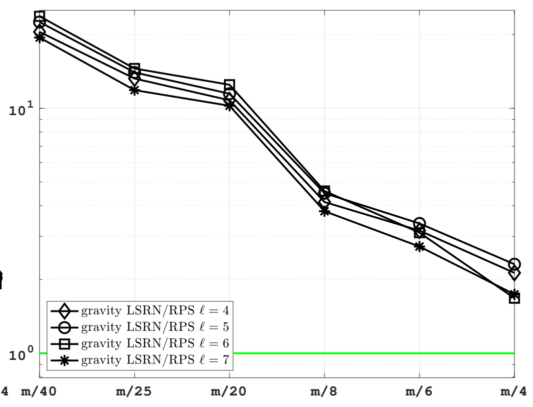
Table 4.7: Timing results to convergence for inversion of `magnetic` potential field data for the simulations described in Table 4.3 without padding, for problem sizes up to $\ell = 7$. Entries with $*$ indicate that the algorithm did not converge. In the last two columns the relative costs of `LSRN` as compared to `RPS`. Values greater than 1 indicate that `RPS` is overall faster. In all cases, `RPS` is faster for inversion of `gravity` data. As projection size increases, however, the relative efficiency of `RPS` decreases and $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$ decreases towards 1. Results for relative errors and number of iterations are presented in Table 4.6 for `magnetic` data.

			$\texttt{Cost}_{\texttt{RPS}}$		$\texttt{Cost}_{\texttt{LSRN}}$		$\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$	
ℓ	t	t_p	GCV	t	GCV	t	GCV	t
4	150	157	117$*$	115$*$	1287	2474$*$	11$*$	21$*$
	240	252	189$*$	187$*$	1212	2473$*$	6$*$	13$*$
	300	315	234$*$	213	902	2477$*$	4$*$	12$*$
	750	787	281	236	1093	1090	4	5
	1000	1050	340	312	1194	998	4	3
	1500	1575	471	427	1092	809	2	2
5	234	245	335$*$	334$*$	3856	7027$*$	11$*$	21$*$
	375	393	545$*$	543$*$	3860	7008$*$	7$*$	13$*$
	468	491	681$*$	536	4172	4392	6$*$	8
	1171	1229	945	807	6586	3300	7	4
	1562	1640	1194	1008	6875$*$	3041	6$*$	3
	2343	2460	1633	1368	3873	2754	2	2
6	337	353	616$*$	615$*$	7553	11549	12$*$	19$*$
	540	567	997$*$	682	7558	8133	8$*$	12
	675	708	1151	747	7538	7571	7	10
	1687	1771	1773	1393	12477$*$	5578	7$*$	4
	2250	2362	1859	1507	12603$*$	5082	7$*$	3
	3375	3543	2357	2358	12583$*$	4528	5$*$	2
7	459	481	1363$*$	1364$*$	31012$*$	27514	23$*$	20$*$
	735	771	2194$*$	1591	19785	19850	9$*$	12
	918	963	2721$*$	1638	30961$*$	18643	11$*$	11
	2296	2410	4038	3401	31325$*$	14807	8$*$	4
	3062	3215	5069	4690	30899$*$	14810	6$*$	3
	4593	4822	7903	6677	30906$*$	14836	4$*$	2

The computational cost is investigated, as measured in seconds, for one iteration of the inversion algorithm using the circulant embedding for the resolutions up to $\ell = 7$ that are indicated in Table 4.3, using both the `RPS` and `LSRN` algorithms, and for both `gravity` and `magnetic` data. For fair comparison, all the timing results that are reported use MATLAB release 2019b implemented on the same cores using an Intel(R) Xeon(R) Gold 6138 processor (2.00GHz) and 256GB RAM. The details of the timing results for one step of the `IRLS` algorithm using `GCV` to obtain truncation parameter κ are illustrated in Figure 4.3, with the specific values for the `gravity` data case, given in Tables 4.4-4.5, and `magnetic` data case, given in Tables 4.6-4.7.



(a) `magnetic` data: $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$.



(b) `gravity` data: $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$.

Figure 4.4: The relative computational cost for one iteration of the `IRLS` algorithm for inversion using the `LSRN` algorithm as compared to the `RPS` algorithm ($\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$), for given ℓ and projected size t. In each case the given plots for a fixed ℓ are for increasing projection size t as given by m/s. The horizontal line at $y = 1$ represents the data for which the costs are the same, independent of whether using the `RPS` or `LSRN` algorithms. `GCV` is used to obtain κ in all cases. The `RPS` algorithm is more efficient when t is maintained small, $s = 40$, 25 and 20, but is still more efficient than `LSRN` when $s = 4$. While the gain in using `RPS` is shown to depend on t, the effect on the relative cost across ℓ is insignificant. This coincides with the computational analysis given in Section 4.4.

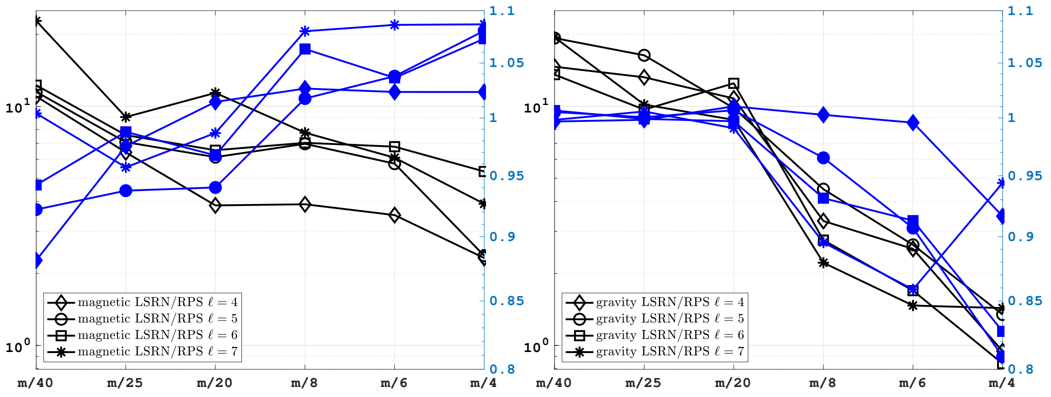
Figure 4.3 provides an overview of the computational cost with increasing projection size t, for a given m, when the algorithm is implemented using the `2DFFT`. These costs exclude the cost of generating $\hat{T}$. In these plots, the black open symbols indicate calculations using the `LSRN` algorithm and blue solid symbols indicate using the `RPS` algorithm. The same symbols are used for each choice of t and ℓ. An initial observation, confirming expectation, is that the timings for equivalent problems and methods, are almost independent of whether the potential field data are `gravity` or magnetic, comparing Figure 4.3a with Figure 4.3b. With increasing ℓ, (increasing values of the triples along the $x-$axis), it can also be observed that the open symbols are less spread out vertically, confirming that the `LSRN` algorithm is more expensive for problems at all resolutions, since the sketch size is based on m instead of t. Indeed, the comparative cost is almost identical for all problem sizes.

Figure 4.4 shows the relative computational costs, $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$ for a single iteration of `IRLS` when using `GCV` to obtain truncation parameter κ. These plots demonstrate that the relative costs for a single iteration are not constant across all t with `RPS` generally cheaper for smaller t. Further, the computational benefit is approximately equal across ℓ. These results confirm the analysis of the computational cost in terms of `flops` provided in Section 4.4. The relative computational costs increase from roughly 1.9 to 22 for `gravity` and from roughly 2 to 23 for `magnetic`, with decreasing t and without regard for ℓ.

4.5.7 Comparative Cost of RPS and LSRN Algorithms to Convergence

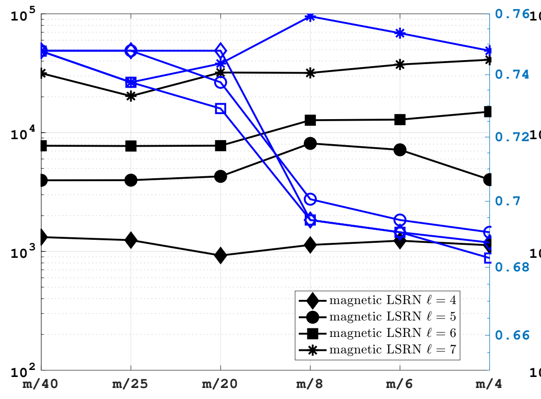
The computational cost of the `IRLS` algorithm for solving the inversion problem to convergence depends on the choice of t, the choice of `LSRN` or `RPS` algorithms, and whether solving the `magnetic` or the `gravity` problem. Tables 4.4-4.7 report the timing results for the inversion of `gravity` and `magnetic` data for problems of increasing size ℓ and projected spaces of sizes t_p. The relative total computational costs to convergence, $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$, (the last two columns in Tables 4.5 and 4.7) are illustrated via Figures 4.5a-4.5b, for the `magnetic` and `gravity` results, respectively. The results in Figures 4.5a and 4.5b for the `magnetic` and `gravity` problems respectively demonstrate a strong preference for the use of the `RPS` algorithm, except for large t, $t = \textbf{floor}(m/4)$ for the `gravity` problem. Furthermore, if based entirely on the calculated `RE`, the results suggest that good results can be achieved for relatively small t as compared to m, certainly $s \gtrsim 8$ leads to generally acceptable error estimates, and the errors using the `LSRN` are generally larger for comparable choices of t.

Figure 4.6 shows the computational cost to convergence for `LSRN` and `RPS`, $\texttt{Cost}_{\texttt{LSRN}}$ and $\texttt{Cost}_{\texttt{RPS}}$, as well as the corresponding `RE` upon convergence. These plots demonstrate that while the costs with the `RPS` algorithm are generally less for smaller t, the `RE` decreases for larger t. The `RE` upon convergence is generally lower when using `RPS` as compared to `LSRN`, with values ranging from roughly 0.66 to 0.75 for `RPS` and from roughly 0.68 to 0.76 for `LSRN`.

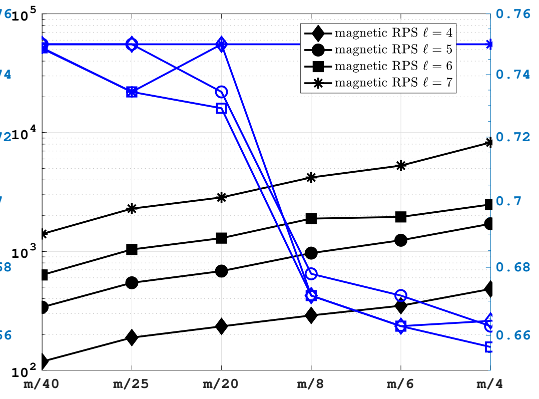


(a) `magnetic` data: $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$. (b) `gravity` data: $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$.

Figure 4.5: Computational cost to convergence of the `IRLS` algorithm for inversion using `LSRN` as compared to `RPS`, $\texttt{Cost}_{\texttt{LSRN}}/\texttt{Cost}_{\texttt{RPS}}$, for `magnetic` and `gravity` problems respectively, in Figures 4.5a-4.5b, black open symbols, and `RE`, corresponding blue solid symbols.

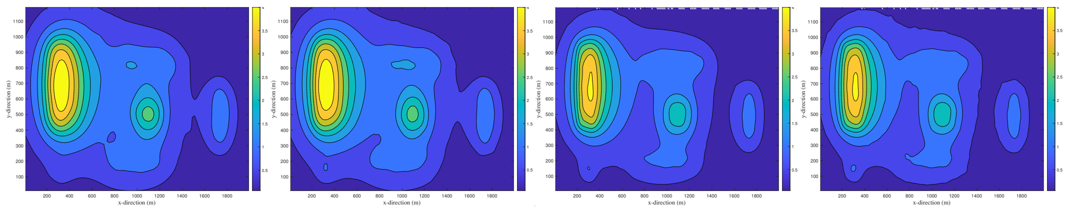


(a) `magnetic` data: `LSRN`.



(b) `magnetic` data: `RPS`.

Figure 4.6: The total computational cost of the `IRLS` algorithm for inversion (left y-axis, open symbols) and corresponding converged `RE` (right y-axis, solid symbols), for given ℓ and projected size t. In each case the given plots for a fixed ℓ are for increasing projection size t as given by m/s. The `RPS` algorithm is more efficient when t is maintained small, $s = 40$, 25 and 20. The `RE` in using `RPS`, however, decreases as t increases. The `RE` also decreases across t when using `LSRN`, but is generally larger than when using `RPS`.

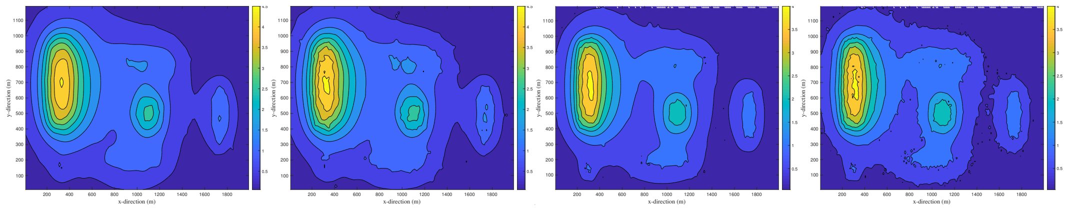


(a) `LSRN`: $\ell = 4$, $t = 300$, $(5, 572\text{s})$.

(b) $\ell = 4$, $t = 750$, $(4, 451\text{s})$.

(c) $\ell = 7$, $t = 918$, $(6, 7408\text{s})$.

(d) $\ell = 7$, $t = 2296$, $(7, 8672\text{s})$.



(e) `RPS`: $(5, 53\text{s})$.

(f) $(5, 136\text{s})$.

(g) $(7, 844\text{s})$.

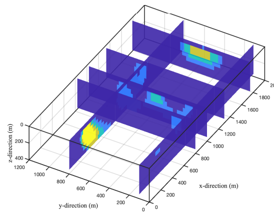
(h) $(12, 3911\text{s})$.

Figure 4.7: For `gravity` data the predicted anomalies (mGal) obtained using `LSRN` in Figures 4.7a-4.7d and `RPS` in Figures 4.7e-4.7h. The first row for `LSRN` indicates the choices of ℓ and t in each column: $t = 300$ and $t = 750$ for $\ell = 4$, and $t = 918$ and $t = 2296$ for $\ell = 7$, corresponding to $t = \textbf{floor}(m/20)$ and $t = \textbf{floor}(m/8)$ for $(m, n) = (6000, 48000)$ and $(18375, 257250)$, respectively. In the captions are pairs $(K, \texttt{Costs})$, (number of iterations to convergence and computational cost in seconds). See Tables 4.4 and 4.5.

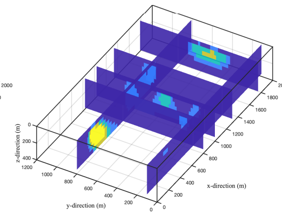
4.5.8 Illustrating Solutions with Increasing ℓ and t

Figure 4.7 illustrates the predicted anomalies and reconstructed volumes for `gravity` data inverted by both algorithms, with resolutions given by $\ell = 4$ and $\ell = 7$ with $t =$ **floor**$(m/20)$ and $t =$ **floor**$(m/8)$. For the cases using $\ell = 4$ it can be seen that the predicted anomalies are generally less accurate than with $\ell = 7$. Moreover, there is more deterioration present in the anomaly predictions when using $t =$ **floor**$(m/20)$ instead of $t =$ **floor**$(m/8)$. It is apparent from consideration of the reconstructed volumes shown in Figures 4.8a-4.8h that the `LSRN` and `RPS` algorithms yield similar results in most cases, and specifically the high resolution $\ell = 7$ results are very good, even using $t =$ **floor**$(m/20)$. When including the consideration of the computational cost, it is clear that if using $\ell = 7$ it is sufficient to use $t =$ **floor**$(m/20)$ and the `RPS` algorithm, but that a reasonable result may even be obtained using the same algorithm but with $\ell = 4$ and requiring less than 1 minute of computational time.

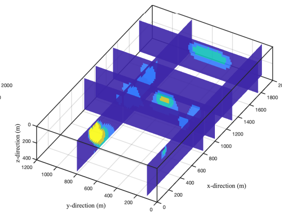
The results for the inversion of the `magnetic` data are illustrated in Figures 4.9 and 4.10 for the same cases as for the inversion of `gravity` data in illustrated in Figure 4.8. Now, in contrast to the `gravity` results, the predicted anomalies are in good agreement with the true data for the results obtained using the `GKB` algorithm, with apparently greater accuracy for the lower resolution solutions, $\ell = 4$ for both choices of t. On the other hand, the predicted `magnetic` anomalies are less satisfactory for small ℓ and t but acceptable for large ℓ. Then, considering the reconstructed volumes, there is a lack of resolution for $\ell = 4$ which is evidenced by the loss of the small well near the surface, which is seen when $\ell = 7$ for both cases of t. The other structures in the domain are also resolved better with $\ell = 7$, but using $t =$ **floor**$(m/8)$ produces a lower `RE` for `RPS` over $t =$ **floor**$(m/20)$. Then, considering the reconstructions obtained using the `RPS` algorithm, while it is clear that the result with $\ell = 4$ and small t is unacceptable, the anomaly and reconstructed volume with



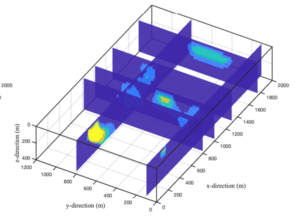
(a) LSRN: $\ell = 4$, $t = 300$, $(.66, .99)$.



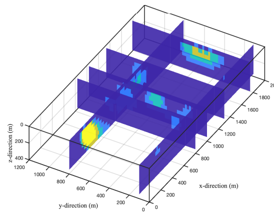
(b) $\ell = 4$, $t = 750$, $(.66, .99)$.



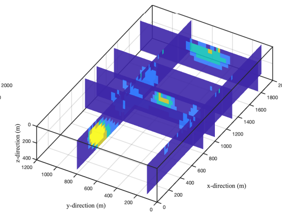
(c) $\ell = 7$, $t = 918$, $(.60, .98)$.



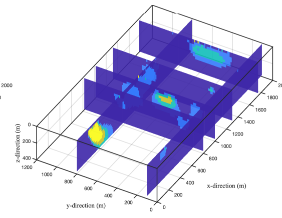
(d) $\ell = 7$, $t = 2296$, $(.60, .97)$.



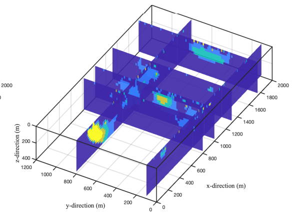
(e) RPS: $(.65, .96)$.



(f) $(.65, .94)$.



(g) $(.61, .95)$.

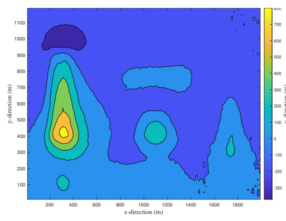


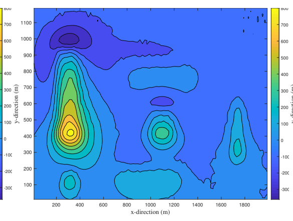
(h) $(.67, .93)$.

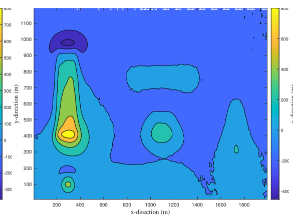
Figure 4.8: For `gravity` data the reconstructed volumes obtained using LSRN in Figures 4.8a-4.8d and RPS in Figures 4.8e-4.8h, corresponding to Figures 4.7a-4.7d and Figures 4.7e-4.7h, respectively. The first row for LSRN indicates the choices of ℓ and t in each column: $t = 300$ and $t = 750$ for $\ell = 4$, and $t = 918$ and $t = 2296$ for $\ell = 7$, corresponding to $t = \textbf{floor}(m/20)$ and $t = \textbf{floor}(m/8)$ for $(m, n) = (6000, 48000)$ and $(18375, 257250)$, respectively. In the captions are pairs $(\texttt{RE}, \chi^2/(m + \sqrt{2m}))$. Results for all cases are summarized in Table 4.4 with timings in Table 4.5.

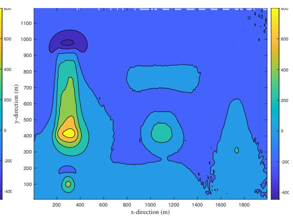
$\ell = 4$ and $t = \textbf{floor}(m/8)$ is acceptable and achieved in reasonable time, approximately 5 minutes, far faster than using LSRN. Thus, the RPS algorithm requires a fraction of the computational time compared to LSRN within the `magnetic` data inversion algorithm for reasonable choice of $t \geq \textbf{floor}(m/8)$.

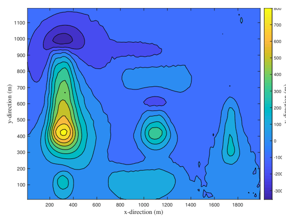
Further, for truncation parameter choice estimation, the GCV function is preferable in general for `gravity` for both RPS and LSRN, although when $\ell = 7$, $\kappa = t$ produces similar timing and number of iterations for RPS. In contrast, the choice $\kappa = t$ is preferable in general for the `magnetic` problem for both RPS and LSRN, and GCV performs poorly when applied to LSRN when ℓ is large, in particular $\ell = 7$. Although the choice $\kappa = t$ is

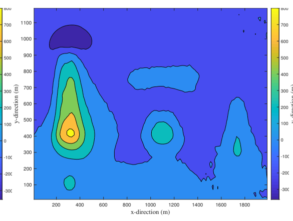

(a) LSRN: $\ell = 4$, $t = 300$, $(9, 902\text{s})$.

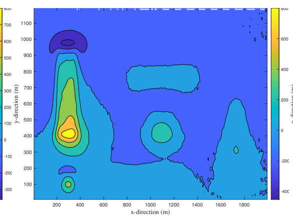

(b) $\ell = 4$, $t = 750$, $(11, 1093\text{s})$.


(c) $\ell = 7$, $t = 918$, $(25, 30961\text{s})$.


(d) $\ell = 7$, $t = 2296$, $(25, 31325\text{s})$.


(e) RPS: $(25, 234\text{s})$.


(f) $(12, 281\text{s})$.


(g) $(25, 2721\text{s})$.

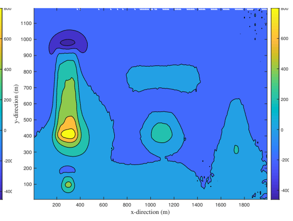
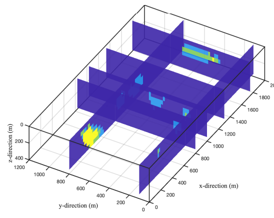

(h) $(14, 4038\text{s})$.

Figure 4.9: For `magnetic` data the predicted anomalies (nT) obtained using `LSRN` in Figures 4.9a-4.9d and `RPS` in Figures 4.9e-4.9h. The first row for `LSRN` indicates the choices of ℓ and t in each column: $t = 300$ and $t = 750$ for $\ell = 4$, and $t = 918$ and $t = 2296$ for $\ell = 7$, corresponding to $t = \textbf{floor}(m/20)$ and $t = \textbf{floor}(m/8)$ for $(m, n) = (6000, 48000)$ and $(18375, 257250)$, respectively. In the captions are pairs $(K, \texttt{Costs})$, (number of iterations to convergence and computational cost in seconds). Results for all cases are summarized in Table 4.6 with timings in Table 4.7.

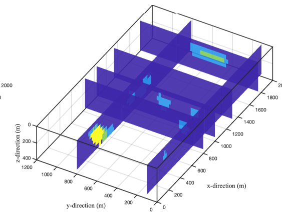
preferable, the `RPS` method performs well for these `magnetic` test cases when $t \geq 10$, regardless of parameter choice method.

4.5.9 Real Data

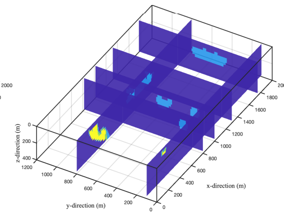
For validation of the simulated results on a practical data set the `RPS` algorithm is applied using $\kappa = t$ as the truncation parameter for the inversion of a magnetic field anomaly that was collected over a portion of the Wuskwatim Lake region in Manitoba, Canada. This data set was discussed in Pilkington [2009] and also used in Vatankhah et al. [2020a]. Further details of the geological relevance of this data set is given in these references. Moreover, its use makes for direct comparison with these existing results. The grid uses



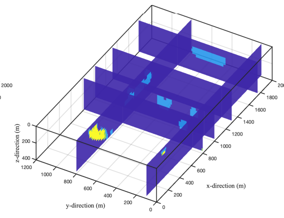
(a) LSRN: $\ell = 4$, $t = 300$, $(.69, .86)$.



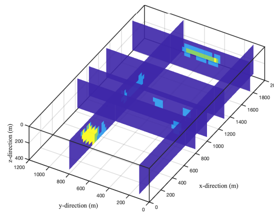
(b) $\ell = 4$, $t = 750$, $(.65, .97)$.



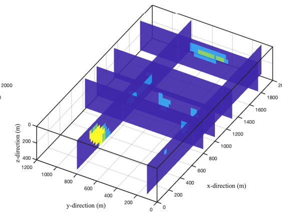
(c) $\ell = 7$, $t = 918$, $(.74, 1.02)$.



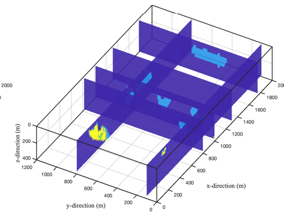
(d) $\ell = 7$, $t = 2296$, $(.76, 1.05)$.



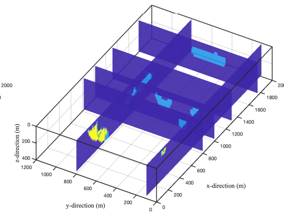
(e) RPS: $(.68, 1.17)$.



(f) $(.63, .96)$.



(g) $(.75, 1.03)$.

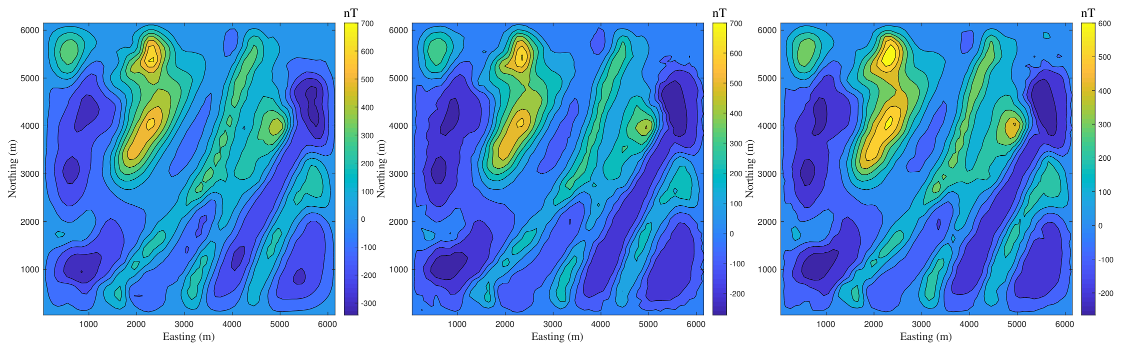


(h) $(.70, .97)$.

Figure 4.10: For magnetic data the reconstructed volumes obtained using LSRN in Figures 4.10a-4.10d and RPS in Figures 4.10e-4.10h, corresponding to Figures 4.9a-4.9d and Figures 4.9e-4.9h, respectively. The first row for LSRN indicates the choices of ℓ and t in each column: $t = 300$ and $t = 750$ for $\ell = 4$, and $t = 918$ and $t = 2296$ for $\ell = 7$, corresponding to $t = \textbf{floor}(m/20)$ and $t = \textbf{floor}(m/8)$ for $(m, n) = (6000, 48000)$ and $(18375, 257250)$, respectively. In the captions are the pairs $(\texttt{RE}, \chi^2/(m + \sqrt{2m}))$. Results for all cases are summarized in Table 4.6 with timings in Table 4.7.

$62 \times 62 = 3184$ measurements at 100m intervals in the East-North direction with padding of 5 cells yielding a horizontal cross section of size 72×72 cells in the East-North directions. The depth dimension is discretized with $\Delta z = 100$m, yielding a regular cube, to $\Delta z = 8$m for rectangular prisms with a smaller edge length in the depth dimension for a total depth of 2000m, and providing increasing values of n from 103680 to 1238976 as detailed in Table 4.8. The given magnetic anomaly is illustrated in Figure 4.11a.

In each inversion the RPS algorithm is run with $t = 384$, corresponding to $t = \textbf{floor}(m/10)$, where $m = 3844$ and oversampled projected space of size 403, and a noise distribution based on (4.10) is employed using $\tau_1 = .02$ and $\tau_2 = .018$. The computational cost measured in seconds is also given in Table 4.8 and demonstrates that it is feasible



(a) Given anomaly. (b) Predicted: $n = 103680$. (c) Predicted: $n = 1238976$.

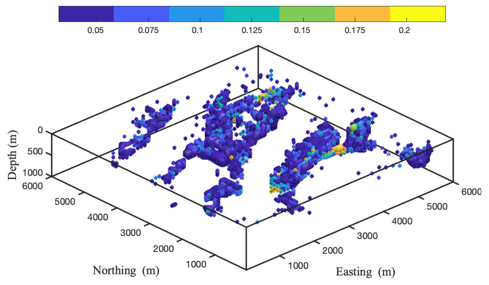
Figure 4.11: The given `magnetic` anomaly in Figure 4.11a and the obtained predicted anomalies for inversion using the parameters for the first and last lines of data in Table 4.8 for the `RPS` method using t as the truncation parameter in Figures 4.11b-4.11c, respectively.

to invert for large parameter volumes, in times ranging from just under 5 minutes for the coarsest resolution, to just over 73 minutes for the volume with the highest resolution. Table 4.8 also shows results for the `RPS` algorithm using `GCV` and for the `LSRN` algorithm using both truncation parameter estimation methods.

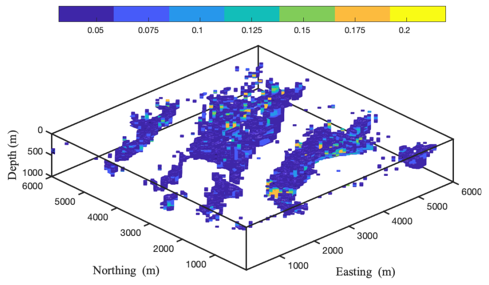
Results of inversion, for the coarsest and finest resolutions are presented in Figures 4.11, 4.12 and 4.13, for anomalies, reconstructed volumes, and depth slices through the volume domain, respectively. It can be seen from Figures 4.11b-4.11c that the predicted anomalies provide better agreement to the measured anomaly with larger n_z. Moreover, the increased resolution provides greater detail in Figure 4.12b as compared to Figure 4.12a. Here the volumes are presented for the depth from 0 to 1000m only, but it is seen in Figures 4.13e and 4.13j, which are the slices at depth 1100m, that there is little structure evident at greater depth. Comparing the depth slices for increasing depth, it is immediate that the use of the higher resolution leads to more structure at increased depth.

Table 4.8: Inversion of practical `magnetic` data using `RPS` with the `GCV` approach, with t as the truncation parameter as illustrated in Figure 4.11, and using `LSRN` with both truncation approaches, for $m = 3844$ on a grid of 62×62 stations, with $\Delta x = \Delta y = 100$m and padding of 5 cells in both x and y-directions, yielding blocks of size $n_r = 5184$. The inversion uses $t = 384$ (**floor**$(m/10)$) and $t_p = 403$. The noise in the algorithm uses (4.10) as given for the simulations with $\tau_1 = .02$ and $\tau_2 = .018$.

method	n	n_z	Δz	K	$\chi^2/(m+\sqrt{2m})$	Cost(s)
`RPS` (`GCV`)	103680	20	100	27$*$	1.41	454
	207360	40	50	27$*$	1.32	912
	305856	59	33	27$*$	1.15	1334
	414720	80	25	27$*$	1.11	1850
	518400	100	20	27$*$	1.12	2300
	616896	119	16	27$*$	1.11	2738
	829440	160	12	27$*$	1.13	3703
	1036800	200	10	27$*$	1.06	4734
	1238976	239	8	27$*$	1.06	5588
`RPS` t	103680	20	100	27$*$	1.06	454
	207360	40	50	27	0.99	909
	305856	59	33	27	0.98	1343
	414720	80	25	24	0.98	1637
	518400	100	20	27	1.03	2284
	616896	119	16	26	0.99	2633
	829440	160	12	23	0.98	3157
	1036800	200	10	24	0.99	4186
	1238976	239	8	25	1.00	5161
`LSRN` (`GCV`)	103680	20	100	18	0.79	1471
	207360	40	50	18	0.92	2942
	305856	59	33	19	0.79	4613
	414720	80	25	18	0.84	5835
	518400	100	20	18	0.89	7257
	616896	119	16	18	0.93	8694
	829440	160	12	18	0.85	11555
	1036800	200	10	19	0.82	15291
	1238976	239	8	18	0.92	17331
`LSRN` t	103680	20	100	27$*$	1.06	2200
	207360	40	50	27$*$	1.03	4446
	305856	59	33	26	1.00	6201
	414720	80	25	26	0.99	8400
	518400	100	20	25	0.98	10088
	616896	119	16	27$*$	1.01	12907
	829440	160	12	26	0.98	16661
	1036800	200	10	26	0.99	20897
	1238976	239	8	27	1.00	25994

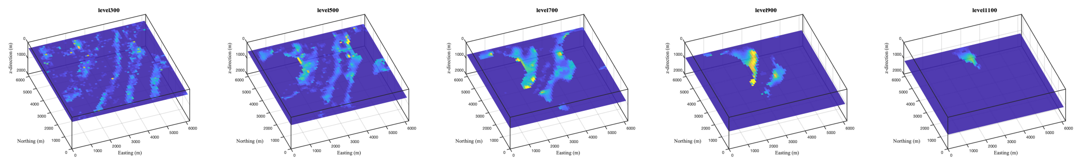


(a) Iso-surface using $n = 103680$.

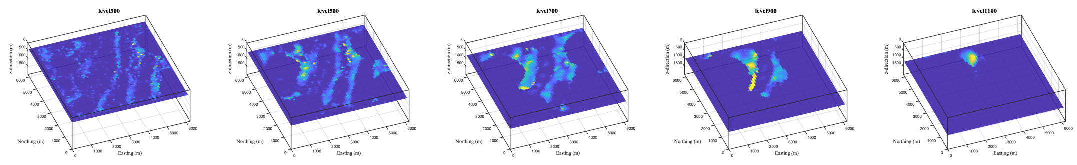


(b) Iso-surface using $n = 1238976$.

Figure 4.12: The reconstructed volumes showing parameters $\kappa > 0.05$ and depth from 0 to 1000, corresponding to the predicted anomalies in Figure 4.11.



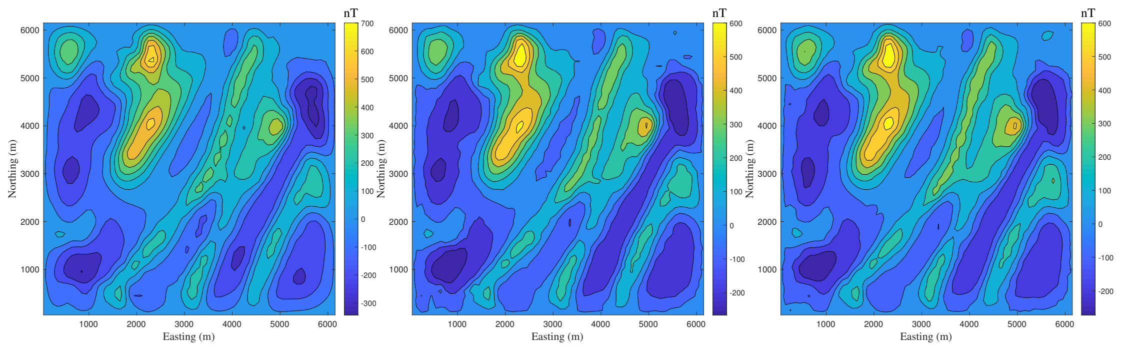
(a) 300m (b) 500m (c) 700m (d) 900m (e) 1100m



(f) 300m (g) 500m (h) 700m (i) 900m (j) 1100m

Figure 4.13: Slices through the volumes illustrated in Figure 4.12 for depths 300, 500, 700, 900 and 1100, for $n = 103680$ in Figures 4.13a-4.13e and for $n = 1238976$ in Figures 4.13f-4.13j.

As shown in Table 4.8, the `RPS` algorithm converges in a fraction of the computational time compared to the `LSRN` algorithm when using $\kappa = t$ to approximate the truncation parameter, but fails to reach convergence when using `GCV`. The χ^2 error upon convergence, however, tends to be larger for the `RPS` method. Similar to the results in Table 4.7, the `RPS` algorithm is slightly faster with lower χ^2 error upon convergence when using $\kappa = t$ to estimate the truncation parameter as compared to applying GCV. In contrast, the opposite is true for the LSRN algorithm, where applying GCV results in a significant reduction in number of iterations to convergence compared to using $\kappa = t$ as the truncation parameter, with smaller χ^2 error and time to convergence. Note that when using GCV, the LSRN method does not rely on t or t_p.



(a) Given anomaly. (b) Predicted: $n = 103680$. (c) Predicted: $n = 1238976$.

Figure 4.14: The given magnetic anomaly in Figure 4.14a and the obtained predicted anomalies for inversion using the parameters for the first and last lines of data in Table 4.9 for the RPS method using t as the truncation parameter in Figures 4.14b-4.14c, respectively.

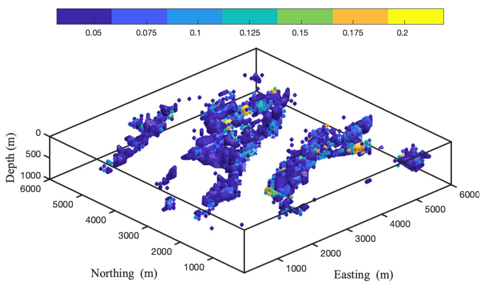
Contrary to the results for simulated data, convergence conditions were not met for many runs for both LSRN and RPS and thus an increase to $t = 480$ is implemented, corresponding to $t = \textbf{floor}(m/8)$, and oversampled projected space of size 504. The updated computational cost measured in seconds is given in Table 4.9 and demonstrates that it is feasible to invert for large parameter volumes, in times ranging from just under 5 minutes for the coarsest resolution, to just over 73 minutes for the volume with the highest resolution.

Results of inversion, for the coarsest and finest resolutions are presented in Figures 4.14, 4.15 and 4.16, for anomalies, reconstructed volumes, and depth slices through the volume domain, respectively. Figures 4.14b-4.14c illustrate that the predicted anomalies provide better agreement to the measured anomaly with larger n_z. Moreover, the increased resolu-

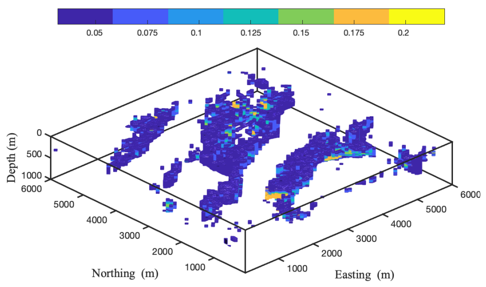
Table 4.9: Inversion of `magnetic` data using `RPS` with the `GCV` approach, with t as the truncation parameter as illustrated in Figure 4.14, and using `LSRN` with both truncation approaches, for $m = 3844$ on a grid of 62×62 stations, with $\Delta x = \Delta y = 100$m and padding of 5 cells in both x and y-directions, yielding blocks of size $n_r = 5184$. The inversion uses $t = 480$ (**floor**$(m/8)$) and $t_p = 504$. The noise in the algorithm uses (4.10) as given for the simulations with $\tau_1 = .02$ and $\tau_2 = .018$.

method	n	n_z	Δz	K	$\chi^2/(m+\sqrt{2m})$	Cost(s)
`RPS` (`GCV`)	103680	20	100	21	0.99	442
	207360	40	50	19	0.99	805
	305856	59	33	19	0.93	1210
	414720	80	25	18	0.97	1548
	518400	100	20	19	0.98	2025
	616896	119	16	19	0.94	2449
	829440	160	12	18	0.93	3176
	1036800	200	10	19	0.91	4160
	1238976	239	8	20	0.99	5299
`RPS` t	103680	20	100	19	0.98	398
	207360	40	50	17	0.99	716
	305856	59	33	17	0.94	1082
	414720	80	25	17	0.97	1472
	518400	100	20	17	0.99	1806
	616896	119	16	17	0.93	2246
	829440	160	12	17	0.94	3031
	1036800	200	10	17	0.97	3729
	1238976	239	8	17	0.99	4485
`LSRN` (`GCV`)	103680	20	100	18	0.79	1471
	207360	40	50	18	0.92	2942
	305856	59	33	19	0.79	4613
	414720	80	25	18	0.84	5835
	518400	100	20	18	0.89	7257
	616896	119	16	18	0.93	8694
	829440	160	12	18	0.85	11555
	1036800	200	10	19	0.82	15291
	1238976	239	8	18	0.92	17331
`LSRN` t	103680	20	100	18	0.99	1467
	207360	40	50	18	0.93	2933
	305856	59	33	17	0.98	4053
	414720	80	25	18	0.93	5860
	518400	100	20	18	0.96	7286
	616896	119	16	18	0.94	8636
	829440	160	12	17	0.99	10954
	1036800	200	10	18	0.92	14600
	1238976	239	8	17	0.98	16419

tion provides greater detail in Figure 4.15b as compared to Figure 4.15a. Here the volumes are presented for the depth from 0 to 1000m only, but it is seen in Figures 4.16e and 4.16j, which are the slices at depth 1100m, that there is little structure evident at greater depth. Again, comparing the depth slices for increasing depth, it is clear that the use of the higher resolution leads to more structure at increased depth.

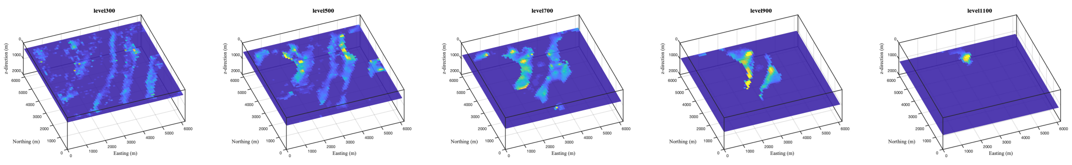


(a) Iso-surface using $n = 103680$.

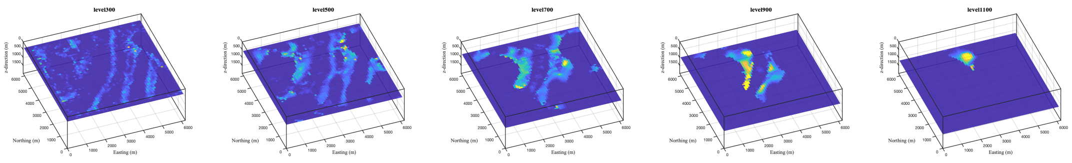


(b) Iso-surface using $n = 1238976$.

Figure 4.15: The reconstructed volumes showing parameters $\kappa > 0.05$ and depth from 0 to 1000, corresponding to the predicted anomalies in Figure 4.14.



(a) 300m (b) 500m (c) 700m (d) 900m (e) 1100m



(f) 300m (g) 500m (h) 700m (i) 900m (j) 1100m

Figure 4.16: Slices through the volumes illustrated in Figure 4.15 for depths 300, 500, 700, 900 and 1100, for $n = 103680$ in Figures 4.16a-4.16e and for $n = 1238976$ in Figures 4.16f-4.16j.

As shown in Table 4.9, the RPS algorithm converges in a fraction of the computational time compared to the LSRN algorithm for both truncation parameter evaluation methods. The χ^2 error upon convergence, however, tends to be larger for the RPS method. Similar to the results in Table 4.7, the RPS algorithm is slightly faster with lower χ^2 error upon convergence when using t to estimate the truncation parameter as compared to applying GCV. With $t = m/8$, the same is now also true for the LSRN algorithm, and using $t = m/8$ results in a significant reduction in number of iterations to convergence compared to using $t = m/10$, with smaller χ^2 error and time to convergence even thought the projection size is larger.

4.6 Conclusions

The RPS method has been shown to significantly reduce computational time compared to the LSRN method by combining dimensionality reduction via a left randomized sketch with a right preconditioning substitution. Analysis has also been given showing that using the RPS method creates an extremely well-conditioned system, along with the associated error bounds for the resulting solution. The RPS method has also been shown to provide regularization via truncation based on choice of truncation parameter, and demonstrations validated the use of the RPS algorithm in comparison to LSRN. The choice of truncation parameter based on a suitable projection size was shown to be preferable for the magnetic problems for both RPS and LSRN. While the GCV method was preferable for the gravity problems, its use was shown to be unnecessary for RPS, particularly as the size of the problems increased. These methods were applied to real data, and for this practical data set the estimate for the truncation parameter of $t = m/8$ was preferable for the RPS and LSRN algorithms. The LSRN algorithm, however, required approximately three times as much computational time compared to the RPS algorithm. The RPS method is easy to implement, and can be applied to a wide range of problems where dimensionality reduction is desired.

Chapter 5

INVERSION OF LARGE-SCALE GRAVITY AND MAGNETIC DATA USING APPROXIMATE SINGULAR VALUE DECOMPOSITIONS

Overview of Chapter

In this chapter, two algorithms, `GKB` and `RSVD`, for the focused inversion of potential field data with all operations for the sensitivity matrix G implemented using a fast `2DFFT` algorithm will be developed and validated for the inversion of both `gravity` and `magnetic` data sets. The results will show that it is distinctly more efficient to use the `2DFFT` for operations with matrix G rather than direct multiplication. This is independent of algorithm and data set, for all large scale implementations considered. Moreover, the implementation using the `2DFFT` makes it feasible to solve these large scale problems on a standard desktop computer without any code modifications to handle multiple cores or GPUs, which is not possible due to memory constraints when m and n increase. While both algorithms are improved with this implementation, the results show that the impact on the `GKB` efficiency is greater than that on the `RSVD` efficiency. When considering the computational cost to convergence for both algorithms, the results will confirm earlier published results that it is more efficient to use `RSVD`, with $t \geq \textbf{floor}(m/8)$ for inversion of `gravity` data. Moreover, generally larger projected spaces are required when using `RSVD` for the inversion of `magnetic` data. On the other hand, prior published work did not contrast `GKB` with `RSVD` for the inversion of `magnetic` data. Here, results will show that `GKB` is more efficient for these large-scale problems and can use also $t \approx \textbf{floor}(m/8)$ rather than larger spaces for use with `RSVD`. Further, `GKB` is also more efficient than `RPS`, with RPS more computationally efficient than `RSVD`. When the implementations use padding, fewer iterations to convergence will be shown to be required. RPS is shown to provide a viable truncation approach for the inversion of real magnetic data. Although GKB is preferable, RPS provides similar timing and similar number of iterations to convergence for all cases, without dependance on either regularization parameter α or truncation parameter κ, provided an appropriate projection space is used.

Associated with the development of a focusing inversion algorithm, is the choice of solver within the inversion algorithm, the choice of regularizer for focusing the subsurface structures, and a decision on determination of suitable regularization parameters. With respect to the solver, small scale problems can be solved using the full singular value decomposition (SVD) of the sensitivity matrix, which is not feasible for the large scale. Moreover, the use of the SVD for focusing inversion has been well-investigated in the literature, see for example [Vatankhah et al., 2014b,a], while choices and implementation details for focusing inversion are reviewed in [Vatankhah et al., 2020c]. Furthermore, methods that yield useful approximations of the SVD, hence enabling automatic but efficient techniques for choice of the regularization parameters have also been discussed in [Renaut et al., 2017, Vatankhah et al., 2017] when considered with iterative Krylov methods based on the Golub-Kahan Bidiagonalization (GKB) algorithm, [Paige and Saunders, 1982], and in Vatankhah et al. [2018a, 2020a] when adopted using the randomized singular value decomposition (RSVD), [Halko et al., 2011]. Recommendations for the application of the RSVD with power iteration, and the sizes of the projected spaces to be used for both GKB and RSVD were presented, but only within the context of problems that can be solved without the use of the 2DFFT. Thus, a complete validation of these algorithms for the solution of the large scale focusing inversion problem, with considerations contrasting the effectiveness of these algorithms in the large scale, is still important, and is addressed here.

An alternative approach for the comparison of RSVD and GKB algorithms was discussed by Luiken and van Leeuwen [2020]. The focus there, on the other hand, was on the effective determination of both the size of the projected space and the determination of the optimal regularization parameter, using these algorithms. Their RSVD algorithm used the range finder suggested in [Halko et al., 2011, Algorithm 4], rather than the power iteration. They concluded with their one rather small example for an under-determined sensitivity matrix of size 400 by 2500 that this was not successful. The test for the GKB approach was successful for this problem, but it is still rather small scale as compared to the problems considered here. Instead as stated, we return to the problem of assessing a suitable size of the projected space to be used for large scale inversion of magnetic and gravity data, using the techniques that provide an approximate SVD and hence efficient and automatic estimation of the regularization parameter concurrently with solving large scale problems. We use the method of Unbiased Predictive Risk Estimation (UPRE) for automatically estimating the regularization parameters, as extensively discussed elsewhere, Vogel [2002].

Overview of main scientific contributions. This chapter provides a comprehensive study of the application of the 2DFFT in regularized focusing inversion algorithms for gravity and magnetic potential field data sets, and summarizes joint research in Renaut et al. [2020]. Specifically, the main contributions are as follows. (i) A detailed review of the mechanics for the inversion of potential field data using focusing inversion algorithms based on the iteratively regularized least squares algorithm in conjunction with the solution of linear systems using GKB or RSVD algorithms; (ii) The extension of these approaches for the use of the 2DFFT for all forward multiplications with the sensitivity matrix, or its transpose; (iii) Comparison of the computational cost when using the 2DFFT as compared to the sensitivity matrix, or its transpose, directly, when implemented within the inversion algorithm, and dependent on the sizes of the projected spaces adopted for the inversion; (iv) Presentation of numerical experiments that confirm that the RSVD algorithm is more efficient than the GKB for the inversion of gravity data sets, for larger problems than previously considered; (v) A new comparison of the use of GKB as compared for RSVD for the inversion of `magnetic` data sets, showing that `GKB` is to be preferred; (vi) All conclusions are confirmed by application on a practical data set, demonstrating that the methodology is suitable for focusing inversion of large scale data sets and can provide parameter reconstructions with more than 1M variables using a laptop computer; (vii) Comparison is given for `RPS` compared to `GKB` and `RSVD` for the inversion of `magnetic` data, which shows that while `GKB` is preferred, `RPS` is a viable alternative.

The chapter is organized as follows. The general methodology used for approximation of the `SVD` is presented in Section 5.1. Details for the numerical solution of the inversion formulation are provided in Section 5.1.1 and the algorithms are in Section 5.1.2. The estimated computational cost of each algorithm, in terms of the number of floating point operations `flops` is given in Section 5.1.3. Numerical results applying the presented algorithms to synthetic and practical data are described in Section 5.2, with the details that apply to all computational implementations given in Section 5.2.1. Results assessing comparison of computational costs for one iteration of the algorithm for use with, and without, the `2DFFT` are discussed in Section 5.2.3. The convergence of the `2DFFT`-based algorithms for problems of increasing size is discussed in Section 5.2.4. Validating results for the inversion of real `magnetic` data obtained over a portion of the Wuskwatim Lake region in Manitoba, Canada are provided in Section 5.2.6 and conclusions in Section 5.3.

5.1 Approximate Singular Value Decomposition

The SVD can be approximated using the sketch $C = AS$, Halko et al. [2011], Vatankhah et al. [2018a]. First, a QR decomposition provides orthogonal $Q \in \mathbb{R}^{m\times s}$ and upper triangular R. Then, the SVD $\tilde{U}\Sigma V$ of $Q^TA \in \mathbb{R}^{s\times n}$ is less computationally expensive since $s \ll m$. Since $QQ^T = \mathrm{I}$, this provides $U \approx Q\tilde{U}$. $U_k\Sigma_kV_k^T$ is then returned, where $k < s$. The sketch $C = S^TA$ can be used to approximate a rank k SVD of A in a similar way.

Either way, the resulting rank k decomposition can then be used to approximate $A^\dagger$, Halko et al. [2011], Vatankhah et al. [2018a].

As shown in Halko et al. [2011],

$$\|A - QQ^TA\|_2 \leq \left[1 + 11\sqrt{s\mathbf{min}(m,n)}\right]\sigma_{k+1}$$

with failure at most $6p^{-p}$ when $p \geq 5$. Since this error bound is based on the separation of singular values, the error may be high when singular values decay slowly. A power iteration can then be used to decrease the error, since $(AA^T)^qA = (U\Sigma^2U^T)^qU\Sigma V^T = U\Sigma^{2q}U^TU\Sigma V^T = U\Sigma^{2q+1}V^T$, and thus the decay of singular values is increased. This method is developed in specific relation to inversion of `gravity` and `magnetic` data in Section 5.1.2.

5.1.1 Numerical Solution

Following the basic setup established in Section 4.5.1 and incorporating regularization parameter α, then the solution $\mathbf{m}^*$ of (4.7) without the bound constraints is given analytically by

$$\mathbf{m} = \mathbf{m}_{\text{apr}} + (G^T\mathbf{W_d}^T\mathbf{W_d}G + \alpha^2\mathbf{W}^T\mathbf{W})^{-1}G^T\mathbf{W_d}^T\mathbf{W_d}(\mathbf{d}_{\text{obs}} - G\mathbf{m}_{\text{apr}}). \tag{5.1}$$

Equivalently, assuming that $\mathbf{W}$ is invertible, and defining $\tilde{G} = \mathbf{W_d}G\mathbf{W}^{-1}$, $\tilde{\mathbf{r}} = \mathbf{W_d}(\mathbf{d}_{\text{obs}} - G\mathbf{m}_{\text{apr}})$ and $\mathbf{y} = \mathbf{m} - \mathbf{m}_{\text{apr}}$, then $\mathbf{y}$ solves the normal equations

$$\mathbf{y} = \mathbf{W}^{-1}(\tilde{G}^T\tilde{G} + \alpha^2I)^{-1}\tilde{G}^T\tilde{\mathbf{r}}, \tag{5.2}$$

and $\mathbf{m}^*$ can be found by restricting $\mathbf{y} + \mathbf{m}_{\text{apr}}$ to lie within the bound constraints.

Now, (5.2) can be used to obtain the iterative solution for (4.7) using `IRLS`, but with dependance on α, as described in Vatankhah et al. [2020c]. Specifically, again superscript k is used to indicate a variable at an iteration k, and α is replaced by $\alpha^{(k)}$, $\mathbf{W}_{\text{L}}$ by matrix $\mathbf{W}_{\text{L}}^{(k)}$

with entries $(\mathbf{W}_{\mathrm{L}}^{(k)})_{ii} = \left((\mathbf{m}_i^{(k-1)} - \mathbf{m}_i^{(k-2)})^2 + \epsilon^2\right)^{\frac{\lambda-2}{4}}$ and $\mathbf{m} - \mathbf{m}_{\mathrm{apr}}$ by $\mathbf{m} - \mathbf{m}^{(k-1)}$, initialized with ${\mathbf{W}_{\mathrm{L}}}^{(1)} = I$, and $\mathbf{m}^{(0)} = \mathbf{m}_{\mathrm{apr}}$ respectively. Then $\mathbf{y}^{(k)}$ is found as the solution of the normal equations (5.2), and $\mathbf{m}^{(k)}$ is the restriction of $\mathbf{y}^{(k)} + \mathbf{m}^{(k-1)}$ to the bound constraints.

This use of the `IRLS` algorithm for the incorporation of the stabilization term Φ_{S} contrasts the implementation discussed in [Zhang and Wong, 2015] for the inversion of potential field `gravity` data. In their presentation, they considered the solution of the general smoothing Tikhonov formulation described by (5.2) for general fixed smoothing operator D replacing $\mathbf{W}_{\mathrm{L}}$. For the solver, they used the re-weighted regularized conjugate solver for iterations to improve $\mathbf{m}^{(k)}$ from $\mathbf{m}_{\mathrm{apr}}$. They also included a penalty function to impose positivity in $\mathbf{m}^{(k)}$, depth weighting to prevent the accumulation of the solution at the surface, and adjustment of α with iteration k to encourage decrease in the data fit term. Moreover, they showed that it is possible to pick approximations D which also exhibit `BTTB` structure for each depth layer, so that $D\mathbf{x}$ can also be implemented by layer using the 2DFFT. Although we do not consider the unifying stabilization framework here with general operator D as described in Vatankhah et al. [2020c], it is a topic for future study, and a further extension of the work, therefore, of Zhang and Wong [2015] for the more general stabilizers. In the earlier work of the use of the `BTTB` structure arising in potential field inversion, Bruun and Nielsen [2007] investigated the use of a truncated `SVD` and the conjugate gradient least squares method for the minimization of the data fit term without regularization. They also considered the direct solution of the constant Tikhonov function with $\mathbf{W}_{\mathrm{L}} = I$ and a fixed regularization parameter α, for which the solution uses the filtered `SVD` in the small scale. Here, not only do we use the unifying stabilization framework, but we also estimate α at each iteration of the `IRLS` algorithm. The `IRLS` algorithm is implemented with two different solvers that yield effective approximations of a truncated SVD. One is based on the randomized singular value decomposition (RSVD), and the second uses the Golub Kahan Bidiagonalization (GKB).

5.1.2 Algorithmic Details

The IRLS algorithm relies on the use of an appropriate solver for finding $\mathbf{y}^{(k)}$ as the solution of the normal equations (5.2) for each update k, and a method for estimating the regularization parameter $\alpha^{(k)}$. While any suitable computational scheme can be used to update $\mathbf{m}^{(k)}$, the determination of $\alpha^{(k)}$ automatically can be challenging. But if the solution technique generates the SVD for $\tilde{G}^{(k)}$, or an approximation to the SVD, such as by use of the RSVD or GKB factorization for $\tilde{G}^{(k)}$, then there are many efficient techniques that can be used such as the unbiased predictive risk estimator (UPRE) or generalized cross validation (GCV). The obtained estimate for $\alpha^{(k)}$ depends on the estimator used and there is extensive literature on the subject, e.g. Hansen [2010]. Thus, here, consistent with earlier studies on the use of the GKB and RSVD for stabilized inversion, we use the UPRE, denoted by $U(\alpha)$, for all iterations $k > 1$, and refer to Renaut et al. [2017] and Vatankhah et al. [2018a, 2020a] for the details on the UPRE. The GKB and RSVD algorithms play, however, a larger role in the discussion and thus for clarity are given here as Algorithms 14 and 15, respectively.

For the use of the GKB we note that Algorithm 14 uses the factorization $\tilde{G}A_{t_p} = H_{t_p+1}B_{t_p}$, where $A_{t_p} \in \mathcal{R}^{n\times t_p}$ and $H_{t_p+1} \in \mathcal{R}^{m\times t_p+1}$. Steps 6 and 11 of Algorithm 14 apply the modified Gram-Schmidt re-orthogonalization to the columns of A_{t_p} and H_{t_p+1}, as is required to avoid the loss of column orthogonality. This factorization is then used in Step 15 to obtain the rank t_p approximate SVD given by $\tilde{G} = (H_{t_p+1}U_{t_p})\Sigma_{t_p}(A_{t_p}V_{t_p})^T$. The quality of this approximation depends on the conditioning of $\tilde{G}$, [Paige and Saunders, 1982]. In particular, the projected system of the GKB algorithm inherits the ill-conditioning of the original system, rather than just the dominant terms of the full SVD expansion. Thus, the approximate singular values include dominant terms that are good approximations to

```
Input: $\tilde{\mathbf{r}} \in \mathcal{R}^m$, $\tilde{G} \in \mathcal{R}^{m\times n}$, a target rank $t$ and size of oversampled projected
        problem $t_p$, $t < t_p \ll m$.
Output: $\alpha$ and $\mathbf{y}$.
1  Set $\mathbf{a} = \mathbf{zeros}(n,1)$, $B = \texttt{sparse}(\mathbf{zeros}(t_p+1,t_p))$, $H = \mathbf{zeros}(m,t_p+1)$,
     $A = \mathbf{zeros}(n,t_p)$;
2  Set $\beta = \|\tilde{\mathbf{r}}\|_2$, $\mathbf{h} = \tilde{\mathbf{r}}/\beta$, $H(:,1) = \mathbf{h}$;
3  for $i = 1 : t_p$ do
4      $\mathbf{b} = \tilde{G}^T\mathbf{h} - \beta\mathbf{a}$ ;
5      for $j = 1 : i-1$ do
6          $\mathbf{b} = \mathbf{b} - (A(:,j)^T\mathbf{b})A(:,j)$ (modified Gram-Schmidt (MGS))
7      end
8      $\gamma = \|\mathbf{b}\|_2$, $\mathbf{a} = \mathbf{b}/\gamma$, $B(i,i) = \gamma$, $A(:,i) = \mathbf{a}$;
9      $\mathbf{c} = \tilde{G}\mathbf{a} - \gamma\mathbf{h}$ ;
10     for $j = 1 : i$ do
11         $\mathbf{c} = \mathbf{c} - (H(:,j)^T\mathbf{c})H(:,j)$ (MGS)
12     end
13     $\beta = \|\mathbf{c}\|_2$, $\mathbf{h} = \mathbf{c}/\beta$, $B(i+1,i) = \beta$, $H(:,i+1) = \mathbf{h}$;
14 end
15 SVD for sparse matrix: $U_{t_p}\Sigma_{t_p}V_{t_p}^T = \mathbf{svds}(B,t_p)$;
16 Apply UPRE to find $\alpha$ using $U_{t_p}(:,1:t)$ and $\Sigma_{t_p}(1:t,1:t)$;
17 Solution $\mathbf{y} = \|\tilde{\mathbf{r}}\|_2 A_{t_p} V_{t_p} \Gamma(\alpha,\Sigma_{t_p}) U_{t_p}(1,:)^T$;
```

Algorithm 14: Use GKB algorithm for factorization $\tilde{G}A_{t_p} = H_{t_p+1}B_{t_p}$ and obtain solution $\mathbf{y}$ of (5.2).

the dominant singular values of the original system, as well as very small singular values that approximate the tail of the singular spectrum of the original system. The accuracy of the dominant terms increases quickly with increasing t_p, Paige and Saunders [1982]. Therefore, to effectively regularize the dominant spectral terms from the rank t_p approximation, in Step 16 we use the truncated UPRE that was discussed and introduced in Vatankhah et al. [2017]. Specifically, a suitable choice for $\alpha^{(k)}$ is found using the truncated SVD of B_{t_p} with t terms. Then, in Step 17, $\mathbf{y}^{(k)}$ is found using all terms in the expansion of B_{t_p}. The matrix $\Gamma(\alpha,\Sigma)$ in Step 17 is the diagonal matrix with entries $\sigma_i/(\sigma_i^2+\alpha^2)$. In the simulations we use $t_p = \mathbf{floor}(1.05\,t)$ corresponding to 5% increase in the space obtained. This contrasts to using just t terms and will include terms from the tail of the spectrum. Note, furthermore, that the top t terms, from the projected space of size $t_p > t$ will be more accurate estimates of the true dominant t terms than if obtained with $t_p = t$. Effectively, by using a 5% increase of t in the calculation of t_p, we assume that the first t terms from the t_p approximation provide good approximations of the dominant t spectral components of the original matrix $\tilde{G}$. We reiterate that the presented algorithm depends on parameters t_p and t. At Step 16 in Algorithm 14 $\alpha^{(k)}$ is found using the projected space of size t but the update for $\mathbf{y}$ in Step 17 uses the oversampled projected space of size t_p. The results presented for the synthetic tests will demonstrate that this uniform choice for t_p is a suitable compromise between taking t_p too small and contaminating the solutions by components from the less accurate approximations of the small components, and a reliable, but larger, choice for t_p that provides a good approximation of the dominant terms within reasonable computational cost.

The algorithm presented in Algorithm 15, denoted as `RSVD`, includes a single power iteration in Steps 3 to 6. Without the use of the power iteration in the `RSVD` it is necessary to use larger projected systems in order to obtain a good approximation of the singular space of the original system, Halko et al. [2011]. Further, it was shown in [Vatankhah et al., 2020a], that when using `RSVD` for potential field inversion, it is better to apply a power iteration. Skipping the power iteration steps leads to a less accurate approximation of the dominant singular space. Moreover, the gain from taking more than one power iteration is insignificant as compared to the increased computational time required. As with the `GKB`, the `RSVD`, with and without power iteration, depends on two parameters t and t_p, where here t is the target rank and t_p is size of the oversampled system, $t_p > t$. For given t and t_p the algorithm uses an eigen decomposition with t_p terms to find the `SVD` approximation of $\tilde{G}$ with t_p terms. Hence, the total projected space is of size t_p, the size of the oversampled system, which is then restricted to size t for estimating the approximation of $\tilde{G}$. Using $(Y + Y^T)/2$ in Step 10 of Algorithm 15, rather than Y, assures that the matrix is symmetric which is important for the efficiency of **eig**.

```
Input: $\tilde{\mathbf{r}} \in \mathcal{R}^m$, $\tilde{G} \in \mathcal{R}^{m\times n}$, a target matrix rank $t$ and size of oversampled
        projected problem $t_p$, $t < t_p \ll m$.
Output: $\alpha$ and $\mathbf{y}$.
1  Generate a Gaussian random matrix $\Omega \in \mathcal{R}^{t_p\times m}$ ;
2  $Y = \Omega\tilde{G} \in \mathcal{R}^{t_p\times n}$ ;
3  $[Q, \sim] = \mathbf{qr}(Y^T, 0)$, $Q \in \mathcal{R}^{n\times t_p}$. (economic QR decomposition) ;
4  $Y = \tilde{G}Q \in \mathcal{R}^{m\times t_p}$ ;
5  $[Q, \sim] = \mathbf{qr}(Y, 0)$, $Q \in \mathcal{R}^{m\times t_p}$ ;
6  $Y = Q^T\tilde{G}$, $Y \in \mathcal{R}^{t_p\times n}$ ;
7  $[Q, \sim] = \mathbf{qr}(Y^T, 0)$, $Q \in \mathcal{R}^{n\times t_p}$ ;
8  $B = \tilde{G}Q \in \mathcal{R}^{m\times t_p}$ ;
9  Compute $Y = B^TB \in \mathcal{R}^{t_p\times t_p}$ ;
10 Eigen-decomposition of $B^TB$: $[\tilde{V}, D] = \mathbf{eig}((Y + Y^T)/2)$ ;
11 $S = \mathbf{diag}(\sqrt{|\mathbf{real}(D)|})$, $[S, \mathtt{indsort}] = \mathbf{sort}(S, 'descend')$;
12 $\tilde{\Sigma}_t = \mathbf{diag}(S(1:t))$, $\tilde{V} = \tilde{V}(:, \mathtt{indsort}(1:t))$, $\tilde{U} = \tilde{V}./(S(1:t)^T)$;
13 Apply UPRE to find $\alpha$ using $\tilde{U}$, $\tilde{\Sigma}_t$, and $B^T\tilde{\mathbf{r}}$;
14 Solution $\mathbf{y} = Q\tilde{V}\Gamma(\alpha, \tilde{\Sigma}_t)\tilde{U}^T(B^T\tilde{\mathbf{r}})$;
15 Note if $\tilde{V}_t = Q\tilde{V}$ is formed; and $\tilde{U}_t = B\tilde{U}\ \tilde{\Sigma}_t^{-1}$, then $\tilde{U}_t\tilde{\Sigma}_t\tilde{V}_t^T$ is a $t$-rank
     approximation of matrix $\tilde{G}$ ;
```

Algorithm 15: Use RSVD with one power iteration to compute an approximate SVD of $\tilde{G}$ and obtain solution $\mathbf{y}$ of (5.2)

It is clear that the RSVD and GKB algorithms provide approximations for the spectral expansion of $\tilde{G}$, with the quality of this approximation dependent on both t and t_p, and hence the quality of the obtained solutions $\mathbf{y}^{(k)}$ at a given iteration is dependent on these choices for t and t_p. As noted, the GKB algorithm inherits the ill-conditioning of $\tilde{G}$ but the RSVD approach provides the dominant terms, and is not impacted by the tail of the spectrum. Thus, the use of the same choices may not be expected for the pairs t and t_p for these algorithms. Vatankhah et al. [2020a] investigated the choices for t and t_p for both gravity and magnetic kernels. When using RSVD with the single power iteration they showed that suitable choices for t, when $t_p = t + 10$, are $t \gtrsim m/s$, where $s \approx 8$ for the gravity problem and $s \approx 4$ for magnetic data inversion. This contrasts using $s \approx 6$ and $s \approx 2$ without power iteration, for gravity and magnetic data inversion, respectively. On the other hand, results presented in [Vatankhah et al., 2017] suggest using $t_p \gtrsim m/s$ where $s \lesssim 20$ for the inversion of `gravity` data using the `GKB` algorithm. This leads to the range of t used in the simulations to be discussed in Section 5.2, $s = 40$, 25, 20, 8, 6, 4 and 3. This permits a viable comparison of cost and accuracy for `GKB` and `RSVD`. Observe that, for the large scale cases considered here, testing with least $s = 3$ was chosen rather than $s = 2$. Indeed, using $s = 2$ generates a large overhead of testing for a wide range of parameter choices, and suggests that relatively large subspaces would be needed defined by $t = m/2$, offering limited gain in speed and computational cost.

5.1.3 Computational Costs

Of interest is the computational cost of (i) the practical implementations of the `GKB` or `RSVD` algorithms for finding the parameter vector $\mathbf{y}^{(k)}$ when operations with matrix G are implemented using the `2DFFT`, and (ii) the associated impact of the choices of t_p on the comparative costs of these algorithms with increasing m and n. In the estimates the focus is on the dominant costs in terms of `flops`, recalling that the underlying cost of a dot product of two vectors of length m is assumed to be $2m$. Further, the costs ignore any overheads of data movement and data access.

First, the evaluation of matrix products with $\tilde{G}$ or $\tilde{G}^T$ is addressed, required at Steps 4 and 9 of Algorithm 14 and 2, 4, 6 and 8 of Algorithm 15. Matrix operations with G, rather than $\tilde{G}$, use the `2DFFT`, as described in Section 3.1 for $G\mathbf{x}$, $G^T\mathbf{y}$ and $\mathbf{y}^TG$, based on the discussion in [Vogel, 2002]. The cost of a single matrix vector operation in each case is $4n_xn_yn_z\log_2(4n_xn_y) = 4n\log_2(4n_r)$. This includes the operation of the `2DFFT` on the reshaped components of $\mathbf{x}_r \in \mathcal{R}^{n_xn_y}$ and the inverse `2DFFT` of the component-wise product of $\hat{\mathbf{x}}_r$ with $\hat{G}^{(r)}$, for $r = 1 : n_z$, but ignores the lower cost of forming the component-wise products and summations over vectors of size n_r. Thus, multiplication with a matrix of size $n \times t_p$ has dominant cost

$$4nt_p \log_2(4n_r),$$

in place of $2mnt_p$. The IRLS algorithm uses operations with $\tilde{G} = \mathbf{W}_{\mathbf{d}} G \mathbf{W}^{-1}$ via component-wise multiplication, and loops over matrix-vector multiplication instead of matrix-matrix multiplication as demonstrated in Section 4.5.2.

Now, to determine the impact of the choices for t (and t_p) the dominant costs are estimated for finding the solution of (5.2) using the GKB and RSVD algorithms. This is the major cost of the IRLS algorithm. The assumptions for the dominant costs of standard algorithms, given in Table 4.3, are quoted from Golub and Van Loan [2013]. But note that the cost for **eig** depends significantly on problem size and symmetry. Here t can be quite large, when m is large, but the matrix is symmetric, hence the estimate $9t^3$ is used, [Golub and Van Loan, 2013, Algorithm 8.3.3]. Note that **svds** for the sparse bidiagonal matrix B is achieved at cost which is at most quadratic in the variables. A comment on the cost of the **qr** operation is also required. Generally, in forming the QR factorization of a matrix the information on the Householder reflectors would be maintained that are used in the reduction of the matrix to upper triangular form, rather than accumulating the matrix Q. The cost is reduced significantly if Q is not accumulated. But, as seen from Steps 2, 4, 6 and 8 of Algorithm 15, evaluation of products of Q with $\tilde{G}$ or its transpose are needed. To take advantage of the 2DFFT evaluating a product of Q with a diagonal scaling matrix is needed first, which amounts to accumulation of matrix Q. Experiments, that are not reported here, show that it is more efficient to accumulate Q as given in Algorithm 15, rather than to to first evaluate the product of Q with a diagonal scaling matrix without pre accumulation. Then, the cost for accumulating Q is $2t^2(m - t/3)$ for a matrix of size $m \times t$, [Golub and Van Loan, 2013, page 255] yielding a total cost for the **qr** step of $4t^2(m - t/3)$, as also reported in Xiang and Zou [2013].

Table 5.1: Computational costs for standard operations. Matrix $G \in \mathcal{R}^{m\times n}$, $X \in \mathcal{R}^{n\times t}$, $Y \in \mathcal{R}^{m\times t}$, sparse bidiagonal $B \in \mathcal{R}^{t+1\times t}$, $A^TA \in \mathcal{R}^{t\times t}$, and $Z \in \mathcal{R}^{m\times t}$. The modified Gram-Schmidt for $C \in \mathcal{R}^{m\times i}$ is repeated for $i = 1 : t$, yielding the given estimate. These costs use the basic unit that the inner product $\mathbf{x}^T\mathbf{x}$ for $\mathbf{x}$ of length n requires $2n$ operations.

GX	G^TY	**svds**(B)	`MGS`(C)	**eig**(A^TA)	$[Q,\sim] =$ **qr**(Z)
$2mnt$	$2mnt$	$6t(m+t)$	$2mt^2$	$9t^3$	$4t^2(m-t/3)$

Results in Table 5.1 estimate the dominant costs of Algorithms 14 and 15. The estimates do not distinguish between costs based on t_p or t, noting $t_p = \textbf{floor}(1.05\,t)$ and $t = m/s$. The distinction between m and n_r is also ignored, where $n_r > m$ for padded domains. Moreover, the cost of finding $\alpha^{(k)}$ and then evaluating $\mathbf{y}^{(k)}$ is of lower order than the dominant costs involved with finding the needed factorizations. Using LOT to indicate the lower order terms that are ignored, and assuming the calculation without the use of the `2DFFT`, the most significant terms yield

$$\begin{aligned}
\texttt{CostG}_{\texttt{GKB}} &= 4nmt + 2t^2(n+m) + LOT \\
\texttt{CostG}_{\texttt{RSVD}} &= 8nmt + 4t^2(2n+m-t) + 2mt^2 + 9t^3 + LOT \\
&= 8nmt + 4t^2(2n+3/2m) + 5t^3 + LOT.
\end{aligned}$$

When using the `2DFFT`, the first two entries $2mnt$ in Table 5.1 are replaced by $4nt\log_2(4n_r)$. Then, using $m \approx n_r$, it is just the first term in each estimate that is replaced leading to the costs with the `2DFFT` as

$$\texttt{Cost}_{\texttt{GKB}} = 8nt\log_2(4m) + 2t^2(n+m) + LOT \tag{5.3}$$

$$\texttt{Cost}_{\texttt{RSVD}} = 16nt\log_2(4m) + 4t^2(2n+3/2m) + 5t^3 + LOT. \tag{5.4}$$

Both pairs of equations suggest, just in terms of `flop` count, that $\texttt{Cost}_{\texttt{RSVD}} > 2\,\texttt{Cost}_{\texttt{GKB}}$. Thus, in order to obtain a comparable cost it would require the use of a smaller t for the

RSVD, than for the GKB. This expectation contradicts earlier experiments contrasting these algorithms for the inversion of gravity data, using the RSVD without power iteration, as discussed in Vatankhah et al. [2018a]. Alternatively, it would be desired that the RSVD should converge in the IRLS far faster than the GKB. Further, theoretically, the gain of using the 2DFFT is that the major terms are $8t^2n$ and $2t^2n$ for the RSVD and GKB, respectively. as compared to $8nmt > 8t^2n$ and $4mnt > 2t^2n$, noting $t < m$. Specifically, even though the costs should go up with order nt^2 eventually with the 2DFFT, this is still far slower than the increase mnt that arises without taking advantage of the structure.

Now, as discussed in Xiang and Zou [2013], measuring the computational cost just in terms of the flop count can be misleading. It was noted by Xiang and Zou [2013] that a distinction between the GKB and RSVD algorithms, where the latter is without the power iteration, is that the operations required in the GKB involve many BLAS2 (matrix-vector) operations, requiring repeated access to the matrix or its transpose, as compared to BLAS3 (matrix-matrix) operations for RSVD implementations. On the other hand, within the **qr** algorithm, the Householder operations also involve BLAS2 operations. Hence, when using MATLAB, the major distinction should be between the use of functions that are builtin and compiled, or are not compiled. In particular, the functions **qr** and **eig** are builtin and hence optimized, but all other operations that are used in the two algorithms do not use any compiled code. Specifically, there is no compiled option for the MGS used in steps 6 and 11 of Algorithm 14, while almost all operations in Algorithm 15 use builtin functions or BLAS3 operations for matrix products that do not involve the matrices with BTTB structure. Thus, in the evaluation of the two algorithms in the MATLAB environment, computational costs will be directly considered, rather than just the estimates given by (5.3) -(5.4). On the other hand, the estimates of the flop counts should be relevant for higher-level programming environments, and are thus relevant more broadly. In all implementations none of the results quoted will use multiple cores or GPUs.

5.2 Numerical Experiments

The fast and efficient methods are now validated for inversion of potential field data using the `BTTB` structure of the `gravity` and `magnetic` kernel matrices. Moreover, the same setup is used as for examples in Chapter 4.

5.2.1 Implementation Parameter Choices

For clarity, the parameter choices used in Chapter 4 are repeated. Diagonal depth weighting matrix $\mathbf{W}_{\mathrm{z}}$ uses $\beta = 0.8$ for the `gravity` problem, and $\beta = 1.4$ for the `magnetic` problem, consistent with recommendations in Li and Oldenburg [1998] and Pilkington [1997], respectively. Diagonal $\mathbf{W}_{\mathbf{d}}$ is determined by the noise in the data, and hard constraint matrix $\mathbf{W}_{\mathrm{h}}$ is taken to be the identity. Moreover, $\mathbf{m}_{\mathrm{apr}} = 0$ is used, indicating no imposition of prior information on the parameters. Regularization parameter $\alpha^{(k)}$ is found using the `UPRE` method for $k > 1$, but initialized with appropriately large $\alpha^{(1)}$ given by

$$\alpha^{(1)} = \left(\frac{n}{m}\right)^{3.5} \frac{\sigma_1}{\mathtt{mean}(\sigma_i)}. \tag{5.5}$$

Here σ_i are the estimates of the ordered singular values for $\mathbf{W}_{\mathbf{d}} G \mathbf{W}^{-1}$ given by the use of the `RSVD` or `GKB` algorithm, and the mean value is taken only over $\sigma_i > 0$. This follows the practice implemented in Vatankhah et al. [2018a], Renaut et al. [2017] for studies using the `RSVD` and `GKB`, and which was based on the recommendation to use a large value for $\alpha^{(1)}$, [Farquharson and Oldenburg, 2004]. In order to contrast the performance and computational cost of the `RSVD` and `GKB` algorithms with increasing problem size m, different sizes t of the projected space for the solution are obtained using $t = \textbf{floor}(m/s)$. Generally, the `GKB` is successful with larger values for s (smaller t) as compared to that needed for the `RSVD` algorithm, $s = 40$, 25, 20, 8, 6, 4 and 3, corresponding to increasing t, but also limited by 5000.

For all simulations, the `IRLS` algorithm is iterated to convergence as determined by the χ^2 test for the predicted data, given by (4.9). If this is not attained for $k \leq K_{\max}$, the iteration is terminated. Noisy data are generated for observed data $\mathbf{d}_{\text{obs}} = \mathbf{d}_{\text{exact}} + \boldsymbol{\eta}$ using (4.10). Recorded for all simulations are (i) the values of the relative error $\texttt{RE}^{(k)}$, as defined by (4.12), (ii) the number of iterations to convergence K which is limited to 25 in all cases, (iii) the scaled χ^2 estimate given by (4.9) at the final iteration, and (iv) the time to convergence measured in seconds, or to iteration 25 when convergence is not achieved.

5.2.2 Numerical Results

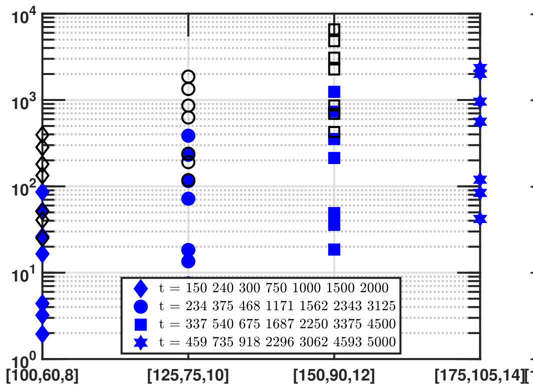
The validation and analysis of the algorithms for the inversion of the potential field data is presented in terms of (i) the cost per iteration of the algorithm (Section 5.2.3), (ii) the total cost to convergence of the algorithm (Section 5.2.4), and (iii) the quality of the obtained solutions, (Section 5.2.5). Supporting quantitative data that summarize the illustrated results are also presented as Tables.

5.2.3 Comparative Cost of RSVD and GKB Algorithms Per IRLS Iteration

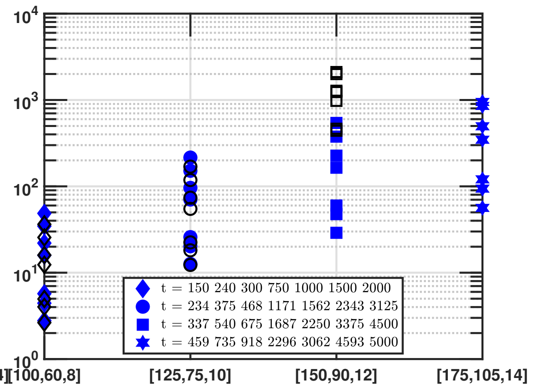
The computational cost is investigated, as measured in seconds, for one iteration of the inversion algorithm using both the direct multiplications using matrix G, respectively, G^T, and the circulant embedding, for the resolutions up to $\ell = 6$ that are indicated in Table 4.3, using both the `RSVD` and `GKB` algorithms, and for both `gravity` and `magnetic` data. For fair comparison, all the timing results that are reported use MATLAB release 2019b implemented on the same iMac 4.2GHz Quad-Core Intel Core i7 with 32GB RAM. In this environment, the size of the matrix G is too large for effective memory usage when $\ell > 6$. The details of the timing results for one step of the `IRLS` algorithm are illustrated in Figures 5.1-3.5, with the specific values for the `magnetic` data case, given in Table 5.2.

Table 5.2: Timing results in seconds for one step of the inversion algorithm applied to `magnetic` potential field data for the simulations described in Table 4.3 without padding and with padding (indicated by `P`), and for problem sizes up to $\ell = 7$. $t_p = \textbf{floor}(1.05t)$ is the size of the oversampled projected space for `GKB` and `RSVD` implementations. The columns under `Direct use of` G do not use the `2DFFT`. These results are illustrated in Figures 5.1-3.5, along with the equivalent set of results for the inversion of `gravity` data.

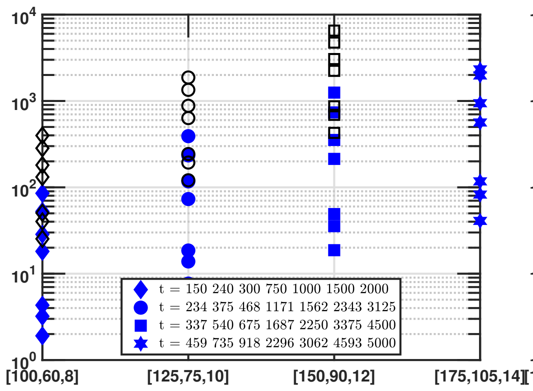
magnetic			WITH 2DFFT				Direct use of G			
ℓ	t	t_p	GKB	RSVD	PGKB	PRSVD	GKB	RSVD	PGKB	PRSVD
4	150	157	2	3	2	2	25	3	31	3
	240	252	3	4	3	4	40	4	49	5
	300	315	4	6	4	5	51	5	62	6
	750	787	16	16	17	14	132	12	161	15
	1000	1050	26	22	28	20	180	16	220	19
	1500	1575	52	35	56	32	283	26	344	31
	2000	2100	87	49	95	46	393	36	477	45
5	234	245	8	13	7	11	120	12	143	14
	375	393	14	20	14	18	193	18	232	22
	468	491	18	26	18	24	244	22	294	27
	1171	1229	72	70	77	68	633	55	765	66
	1562	1640	115	96	125	90	862	74	1044	89
	2343	2460	230	151	257	144	1347	118	1633	142
	3125	3281	389	215	435	208	1869	169	2278	211
6	337	353	19	29	16	20	430	440	532	1597
	540	567	36	48	32	35	689	1996	831	2985
	675	708	49	60	46	45	867	977	1050	2821
	1687	1771	213	164	224	127	2255	465	2739	1301
	2250	2362	351	227	382	182	3068	1235	3738	2425
	3375	3543	733	376	818	315	4798	1279	5890	2834
	4500	4725	1259	542	1413	475	6666	2108	61661	3487
7	459	481	41	56	54	72	NA	NA	NA	NA
	735	771	84	94	104	117	NA	NA	NA	NA
	918	963	117	121	145	150	NA	NA	NA	NA
	2296	2410	554	346	674	433	NA	NA	NA	NA
	3062	3215	944	496	1136	601	NA	NA	NA	NA
	4593	4822	1999	854	2409	1061	NA	NA	NA	NA
	5000	5250	2317	949	2868	1192	NA	NA	NA	NA



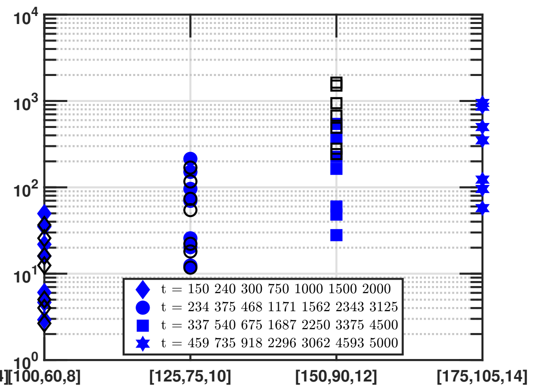
(a) Running time `magnetic`: `GKB`.



(b) Running time `magnetic`: `RSVD`.



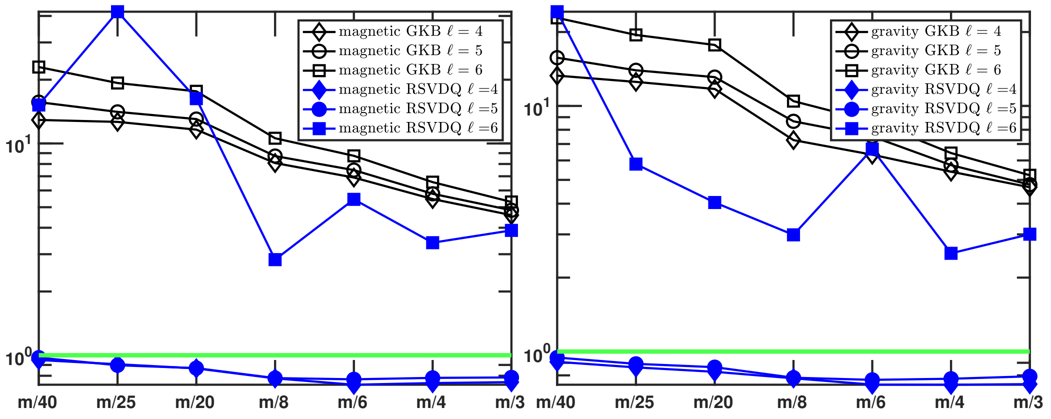
(c) Running time `gravity`: `GKB`.



(d) Running time `gravity`: `RSVD`.

Figure 5.1: Running time in seconds for one iteration of the inversion algorithm for the inversion of `magnetic` and `gravity` data, without padding the volume domain. Problems are of increasing size, as indicated by the $x-$axis for triples $[n_x, n_y, n_z]$ and increasing projection size t ($y-$axis using log scale) determined by fractions of $m = s_x s_y$. In Figures 5.1a and 5.1c the running time for the `GKB` algorithm using $t_p = \textbf{floor}(1.05t)$ (an oversampling percentage 5%), for the `magnetic` and `gravity` problems respectively. In Figures 5.1b and 5.1d the equivalent running times using the `RSVD` algorithm with one power iteration. In these plots the solid symbols represent the timing for one iteration of the algorithm using the `2DFFT` and the open symbols represent the timing for the same simulation using G directly. Matrix G for problem size $\ell = 7$, which corresponds to triple $[175, 105, 14]$, requires too much memory for implementation in the specific computing environment.

Figure 5.1 provides an overview of the computational cost with increasing projection size t, for a given m, when the algorithm is implemented using G directly, or using the `2DFFT`. These costs exclude the cost of generating G. In these plots, the open symbols indicate calculations using G and solid symbols indicate using the `2DFFT`. The same symbols are used for each choice of t and ℓ. An initial observation, confirming expectation, is that the timings for equivalent problems and methods, are almost independent of whether the potential field data are `gravity` or magnetic, comparing Figures 5.1a-5.1b with Figures 5.1c and 5.1d. The lack of entries for triple $[175, 105, 14]$ indicates that the matrix G is too large for the operations, $\ell = 7$. With increasing ℓ, (increasing values of the triples along the $x-$axis), it can also be observed that the open symbols are more spread out vertically, confirming that the algorithms using G directly are more expensive for problems at these resolutions.

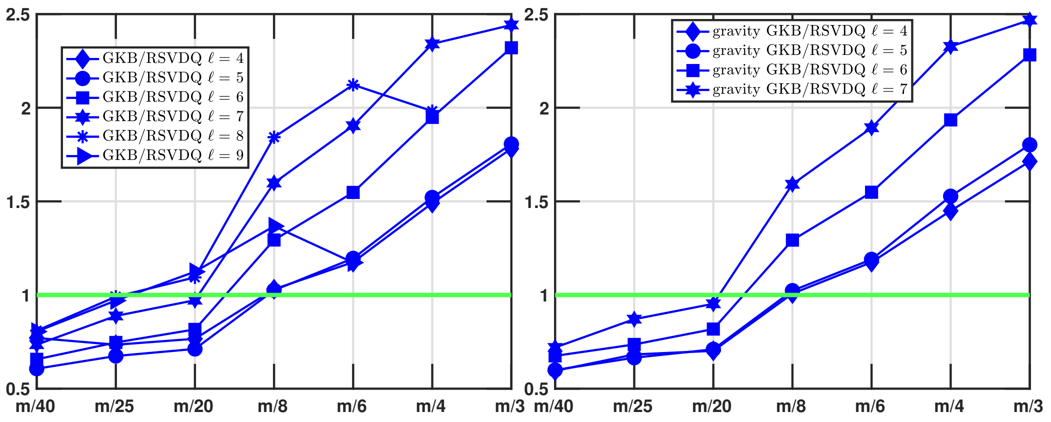


(a) `magnetic` data: $\texttt{Cost}_{\texttt{G}}/\texttt{Cost}_{\texttt{2DFFT}}$. (b) `gravity` data: $\texttt{Cost}_{\texttt{G}}/\texttt{Cost}_{\texttt{2DFFT}}$.

Figure 5.2: Relative computational cost for one iteration of the `IRLS` algorithm using G directly as compared to the `2DFFT`, as indicated by $\texttt{Cost}_{\texttt{G}}/\texttt{Cost}_{\texttt{2DFFT}}$, for the data presented in Figure 5.1, for the `magnetic` and `gravity` problems, Figures 5.2a-5.2b. Here, the values for the relative cost that are less than 1, below the horizontal line at $y = 1$, indicate that it is more efficient to use G directly. Values that are greater than 1 indicate that it is more efficient to use the `2DFFT`. Open symbols indicate the `GKB` algorithm and solid symbols the `RSVD` algorithm. In each case the given plots for a fixed ℓ are for increasing projection size t as given by m/s for the selections of t as used in Figure 5.1.

Figure 5.2 shows the relative computational costs for one iteration of the `IRLS` algorithm using the matrix G as compared to the algorithm using the `2DFFT`, as indicated by $\texttt{Cost}_{\texttt{G}}/\texttt{Cost}_{\texttt{2DFFT}}$, for the data presented in Figure 5.1. The size t used for the projected problem in terms of the ratio m/s is given along the $x-$axis. The lines with solid blue symbols are for results using the RSVD algorithm, and the open black symbols are for the GKB algorithm. Here, the values for the relative cost that are less than 1, below the horizontal green line at $y = 1$, indicate that for the specific algorithm it is more efficient to use G directly. Values that are greater than 1 indicate that it is more efficient to use the 2DFFT for the given algorithm and problem size. It is apparent that it is not beneficial to use the 2DFFT for the smaller scale implementation of the RSVD algorithm, when $\ell = 4$ or 5. But the situation is completely reversed using the GKB algorithm for all choices of ℓ and the RSVD algorithm for $\ell \geq 6$. Thus, the relative gain in reduced computational cost, by using the 2DFFT depends on the algorithm used within the IRLS inversion algorithm. The decrease in efficiency for a given size problem, fixed ℓ but increasing size t (in the $x-$axis), is explained by the theoretical discussion relating to equations (5.3)-(5.4). As t increases the impact of the efficient matrix multiplication using the 2DFFT is reduced. Again the gravity and magnetic data results are comparable.

Figure 5.1 provides no information on the relative costs of the GKB and RSVD algorithms with increasing ℓ, independent of the use of the 2DFFT. Figure 5.3 shows the relative computational costs, $\mathtt{Cost}_{\mathtt{GKB}}/\mathtt{Cost}_{\mathtt{RSVD}}$. Note that Figure 5.3a also includes results for larger problems. These plots demonstrate that the relative costs for a single iteration are not constant across all t with the GKB generally cheaper for smaller t, and the RSVD cheaper for larger t. These results confirm the analysis of the computational cost in terms of flops provided in (5.3)-(5.4) for small t. The relative computational costs increase from roughly 0.6 to 2.5, increasing with both ℓ and t. Still, this improved relative performance of RSVD with increasing ℓ and t appears to violate the flop count analysis in (5.3)-(5.4). As discussed in Section 5.1.3, this is a feature of the implementation. While RSVD is implemented using the MATLAB builtin function **qr** which uses compiled code for faster implementation, GKB only uses builtin operations for performing the MGS re-

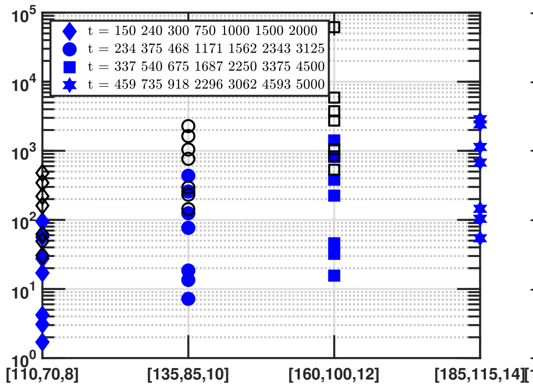


(a) `magnetic` data: $\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$. (b) `gravity` data: $\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$.

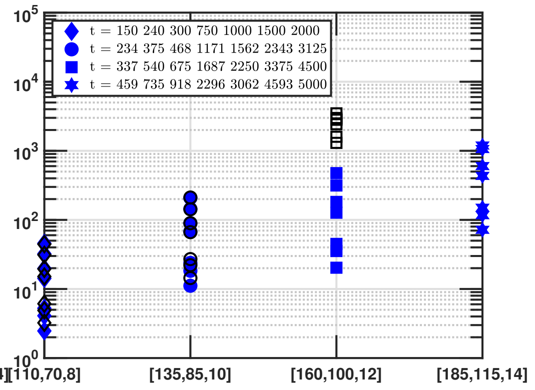
Figure 5.3: The relative computational cost for one iteration of the `IRLS` algorithm for inversion using the `GKB` as compared to the `RSVD` algorithm ($\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$), for given ℓ and projected size t. In each case the given plots for a fixed ℓ are for increasing projection size t as given by m/s as in Figure 5.2. The horizontal line at $y = 1$ represents the data for which the costs are the same, independent of whether using the `RSVD` or `GKB` algorithms. The `GKB` is more efficient when t is maintained small, $s = 40$, 25 and 20. The gain in using the `GKB` decreases, however, as ℓ increases. For small ℓ and t, the estimates confirm the computational cost estimates in (5.3)-(5.4), but for larger projection sizes t, the `RSVD` is more efficient. In Figure 5.3a the relative costs are also included for $\ell = 8$ and $\ell = 9$, where $t \leq 5000$.

orthogonalization of the basis matrices A_{t_p} and H_{t_p}. Once again results are comparable for inversion of both `gravity` and `magnetic` data sets.

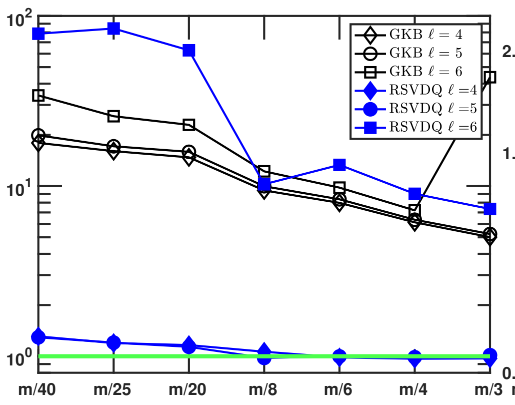
Figure 5.4 summarizes `magnetic` data timing results from Table 5.2 for domains which are padded with 5% padding in x and y directions. Data illustrated in Figures 5.4a-5.4b are equivalent to the results presented in Figures 5.1a-5.1b, but with padded volume domains. Again these results show the open symbols are more spread out vertically, for increasing ℓ, confirming that the algorithms using G directly are more expensive for problems at these resolutions, with greater impact when using the `GKB` algorithm for small ℓ. This is further confirmed in Figure 5.4c, equivalent to Figure 5.2a, showing that the computational cost of performing one step of the `IRLS` algorithm using matrix G directly, is always greater than that using the `2DFFT`. This is more emphasized for the `GKB` algorithm.



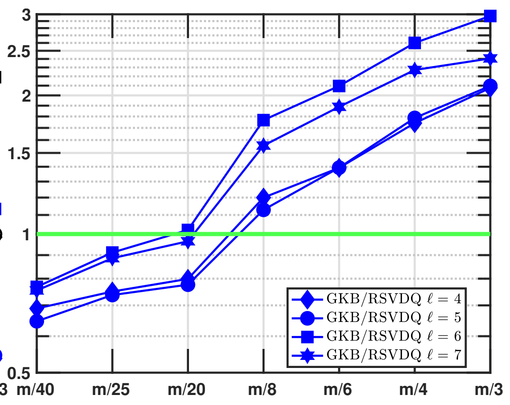
(a) Running time (padded): GKB.



(b) Running time (padded): RSVD.



(c) $\text{Cost}_{\text{G}}/\text{Cost}_{\text{2DFFT}}$ (Padded).

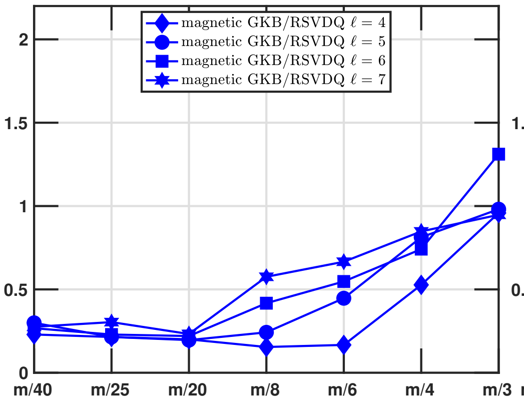


(d) $\text{Cost}_{\text{GKB}}/\text{Cost}_{\text{RSVD}}$ (Padded).

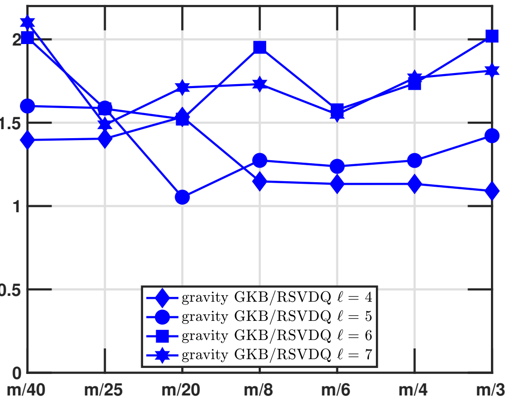
Figure 5.4: In Figures 5.4a-5.4b the running time in seconds for one iteration of the inversion algorithm for the inversion of magnetic data, for the same problems as in Figure 5.1a and 5.1b but with padding, $\text{pad} = 5\%$, added to the volume domain. Problems are of increasing size, as indicated by the $x-$axis for triples $[n_x, n_y, n_z]$ and increasing projection size t ($y-$axis using $\log$ scale) determined by fractions of $m = s_x s_y$. In these plots the solid symbols represent the timing for one iteration of the algorithm using the 2DFFT and the open symbols represent the timing for the same simulation without using the 2DFFT for the kernel operations. In Figure 5.4c the relative costs for these results, as also provided in Figure 5.2a for the case without padding, and in Figure 5.4d the relative costs of the two algorithms with the 2DFFT, as in Figure 5.3a without padding.

The relative costs shown in Figure 5.4d, equivalent to Figure 5.3a, again shows that the GKB algorithm is cheaper for small t when ℓ is small. But as the problem size increases and the projected problem size also increases, it is more efficient to use the RSVD algorithm, consistent with the observations for the unpadded domains.

5.2.4 Comparative Cost of RSVD and GKB Algorithms to Convergence



(a) `magnetic` $\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$.



(b) `gravity` $\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$.

Figure 5.5: Computational cost to convergence of the `IRLS` algorithm for inversion using the `GKB` as compared to the `RSVD` algorithm, $\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$, for the `magnetic` and `gravity` problems respectively, in Figures 5.5a-5.5b.

The computational cost of the `IRLS` algorithm for solving the inversion problem to convergence depends on the choice of t, the choice of `GKB` or `RSVD` algorithms, and whether solving the `magnetic` or the `gravity` problem. Table 5.3 reports the timing results for the inversion of `gravity` and `magnetic` data for problems of increasing size ℓ and projected spaces of sizes t_p. The relative total computational costs to convergence, $\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$, (the last two columns in Table 5.3) are illustrated via Figures 5.5a-5.5b, for the `magnetic` and `gravity` results, respectively. There is a distinct difference between the two problems. The results in Figure 5.5a for the `magnetic` problem demonstrate a strong preference for the use of the `GKB` algorithm, except for large t, $t = \textbf{floor}(m/3)$. In contrast, the `RSVD` algorithm is always most efficient for the solution of the `gravity` problem, which is consistent with the conclusion presented in Vatankhah et al. [2018a] for `RSVD` without power iteration. Moreover, the data presented in Table 5.4 for the `gravity` problem, indicate that the `RSVD` algorithm generally converges more quickly and yields a smaller relative error. Furthermore, if based entirely on the calculated

Table 5.3: Timing results to convergence for inversion of `gravity` and `magnetic` potential field data for the simulations described in Table 4.3 without padding, for problem sizes up to $\ell = 7$. Entries with $*$ indicate that the algorithm did not converge. In the last two columns the relative costs of `GKB` as compared to `RSVD`. Values greater than 1, less than 1, indicate that the `RSVD` is overall faster, slower, respectively. In general `RSVD` is faster for inversion of `gravity` data but slower for inversion of `magnetic` data. Still, as problem size increases, the relative efficiency of `GKB` for the `magnetic` data decreases, $\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$ increases towards 1. Results for relative errors and number of iterations are presented in Tables 5.5-5.4, for `magnetic` and `gravity` data, respectively.

			`gravity`		`magnetic`		$\texttt{Cost}_{\texttt{GKB}}/\texttt{Cost}_{\texttt{RSVD}}$	
ℓ	t	t_p	`GKB`	`RSVD`	`GKB`	`RSVD`	`gravity`	`magnetic`
4	150	157	78	56	40	172*	1.40	0.23
	240	252	111	79	60	282*	1.40	0.21
	300	315	153	100	70	351*	1.53	0.20
	750	787	248	216	136	883*	1.15	0.15
	1000	1050	323	285	197	1182*	1.13	0.17
	1500	1575	494	436	343	650	1.13	0.53
	2000	2100	641	588	739	771	1.09	0.96
5	234	245	265	166	152	509*	1.60	0.30
	375	393	411	259	174	811*	1.59	0.21
	468	491	342	325	199	1014*	1.05	0.20
	1171	1229	1064	835	626	2582*	1.27	0.24
	1562	1640	1235	997	948	2121	1.24	0.45
	2343	2460	1899	1492	1728	2126	1.27	0.81
	3125	3281	2918	2052	2915	2971	1.42	0.98
6	337	353	595	296	246	923*	2.01	0.27
	540	567	671	424	347	1514*	1.58	0.23
	675	708	802	527	413	1877*	1.52	0.22
	1687	1771	2704	1385	1077	2581	1.95	0.42
	2250	2362	2518	1597	1608	2937	1.58	0.55
	3375	3543	4308	2483	3071	4142	1.73	0.74
	4500	4725	6925	3429	6699	5109	2.02	1.31
7	459	481	1427	679	594	2157*	2.10	0.28
	735	771	1642	1104	1070	3524*	1.49	0.30
	918	963	2084	1218	1026	4413*	1.71	0.23
	2296	2410	5732	3311	3809	6618	1.73	0.58
	3062	3215	6959	4490	5639	8469	1.55	0.67
	4593	4822	12347	6979	10979	12949	1.77	0.85
	5000	5250	13975	7711	12544	13239	1.81	0.95

RE, the results suggest that good results can be achieved for relatively small t as compared to m, certainly $s \gtrsim 8$ leads to generally acceptable error estimates, and in contrast to the case without the power iteration, here with power iteration, the errors using the GKB are generally larger for comparable choices of t.

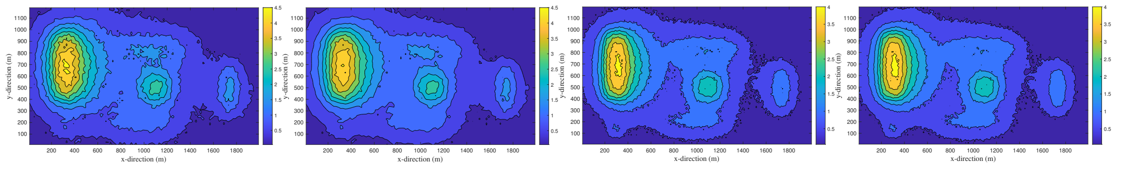
For the magnetic data, the results in Table 5.5 demonstrate that the RSVD algorithm generally requires more iterations than the GKB algorithm, and that the obtained relative errors are then comparable, or slightly larger. This is then reflected in Figure 5.5a that the GKB algorithm is most efficient. Referring back to Table 5.5, it is the case that the RSVD algorithm often reaches the maximum number of iterations, $K = 25$, without convergence, when GKB has converged in less than half the number of iterations, when t is small relative to m, $t =$ **floor**(m/s) with $s = 40$, 25 and 20. This verifies that the RSVD needs to take a larger projected subspace t in order to capture the required dominant spectral space when solving the magnetic problem, as compared to the gravity problem, and confirms the conclusions presented in Vatankhah et al. [2020a]. On the other hand, the use of the GKB as compared to the RSVD was not discussed in Vatankhah et al. [2020a]. These results now lead to a new conclusion concerning these two algorithms for solving the magnetic data inversion problem. In particular, the results suggest that the GKB algorithm be adopted for inversion of magnetic data. Further, the results suggest that the relative error obtained using the GKB generally decreases with increasing t, and that it is necessary to use subspaces with t at least as large as **floor**$(m/8)$. It remains to verify these assertions by illustrating the results of the inversions and the predicted anomalies for a selection of cases.

Table 5.4: Results for inversion of `gravity` potential field data for the simulations described in Table 4.3 without padding and with padding and for problem sizes up to $\ell = 7$. The maximum number of iterations is set to 25 in all cases. t_p is the size of the projected space for `GKB` and `RSVD` implementations. Reported are the number of iterations to convergence, K, for convergence as defined by (4.9), with $K = 25$ indicating that the simulation did not converge to the given tolerance. The calculated relative error `RE` for the given K are also given, for both unpadded and padded cases respectively.

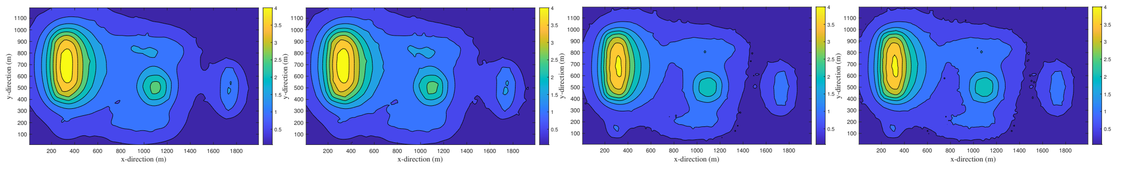
`gravity`			`GKB`		`RSVD`		`PGKB`		`PRSVD`	
ℓ	t	t_p	K	`RE`	K	`RE`	K	`RE`	K	RE
4	150	157	19	1.00	8	0.56	20	1.00	8	0.56
	240	252	16	0.97	7	0.56	17	0.97	7	0.56
	300	315	17	0.97	7	0.56	17	0.96	7	0.56
	750	787	9	0.76	6	0.57	10	0.76	6	0.57
	1000	1050	8	0.66	6	0.57	8	0.66	6	0.57
	1500	1575	7	0.64	6	0.57	7	0.64	6	0.57
	2000	2100	6	0.62	6	0.57	7	0.64	7	0.58
5	234	245	21	1.05	8	0.49	21	1.05	9	0.51
	375	393	19	1.01	8	0.50	21	1.03	9	0.53
	468	491	12	0.82	8	0.50	13	0.84	8	0.53
	1171	1229	12	0.78	8	0.53	11	0.79	8	0.57
	1562	1640	9	0.66	7	0.53	9	0.68	8	0.58
	2343	2460	8	0.65	7	0.55	8	0.67	8	0.59
	3125	3281	8	0.64	7	0.57	8	0.66	7	0.60
6	337	353	24	1.03	8	0.56	23	1.03	8	0.60
	540	567	15	0.90	7	0.58	15	0.93	7	0.61
	675	708	14	0.89	7	0.58	14	0.91	7	0.62
	1687	1771	13	0.85	7	0.61	12	0.85	6	0.64
	2250	2362	8	0.70	6	0.62	8	0.71	6	0.64
	3375	3543	7	0.69	6	0.63	8	0.71	6	0.64
	4500	4725	7	0.69	6	0.63	8	0.70	6	0.64
7	459	481	24	1.07	8	0.56	25	1.08	8	0.59
	735	771	16	0.95	8	0.57	16	0.95	8	0.59
	918	963	15	0.93	7	0.58	15	0.94	7	0.60
	2296	2410	11	0.75	7	0.60	11	0.75	7	0.60
	3062	3215	9	0.72	7	0.60	10	0.73	7	0.61
	4593	4822	8	0.70	7	0.61	9	0.71	7	0.61
	5000	5250	8	0.70	7	0.61	9	0.71	7	0.61

Table 5.5: Results for inversion of `magnetic` potential field data for the simulations described in Table 4.3 without padding and with padding and for problem sizes up to $\ell = 7$. The maximum number of iterations is set to 25 in all cases. t_p is the size of the projected space for `GKB` and `RSVD` implementations. Reported are the number of iterations to convergence, K, for convergence as defined by (4.9), with $K = 25$ indicating that the simulation did not converge to the given tolerance. The calculated relative error `RE` for the given K are also given, for both unpadded and padded cases respectively.

`magnetic`			`GKB`		`RSVD`		`PGKB`		`PRSVD`	
ℓ	t	t_p	K	`RE`	K	`RE`	K	`RE`	K	RE
4	150	157	10	0.71	25	0.72	5	0.63	25	0.79
	240	252	9	0.68	25	0.69	5	0.63	25	0.72
	300	315	8	0.66	25	0.69	5	0.63	25	0.74
	750	787	5	0.63	25	0.64	4	0.63	25	0.65
	1000	1050	5	0.63	25	0.63	4	0.63	19	0.65
	1500	1575	5	0.63	9	0.63	5	0.62	8	0.64
	2000	2100	7	0.61	8	0.63	6	0.60	7	0.63
5	234	245	12	0.81	25	0.82	6	0.71	25	0.89
	375	393	8	0.72	25	0.78	6	0.69	25	0.82
	468	491	7	0.70	25	0.77	6	0.69	25	0.80
	1171	1229	7	0.66	25	0.70	6	0.66	25	0.71
	1562	1640	7	0.66	15	0.70	6	0.66	12	0.71
	2343	2460	7	0.65	10	0.67	6	0.66	9	0.68
	3125	3281	8	0.65	10	0.67	8	0.66	9	0.68
6	337	353	10	0.74	25	0.73	5	0.67	25	0.77
	540	567	8	0.69	25	0.69	5	0.67	25	0.73
	675	708	7	0.67	25	0.68	5	0.67	25	0.71
	1687	1771	5	0.64	13	0.66	5	0.65	10	0.68
	2250	2362	5	0.64	11	0.65	5	0.66	10	0.68
	3375	3543	5	0.64	10	0.64	5	0.67	9	0.66
	4500	4725	7	0.62	9	0.63	5	0.67	9	0.66
7	459	481	11	0.78	25	0.80	6	0.69	25	0.80
	735	771	10	0.74	25	0.75	6	0.69	25	0.76
	918	963	7	0.69	25	0.73	6	0.69	25	0.75
	2296	2410	7	0.67	14	0.70	5	0.72	12	0.72
	3062	3215	7	0.68	13	0.70	6	0.69	11	0.71
	4593	4822	7	0.68	13	0.69	6	0.70	11	0.72
	5000	5250	7	0.68	12	0.68	6	0.70	11	0.72



(a) GKB: $\ell = 4$, $t = 750$, $(9, 248\text{s})$. (b) $\ell = 4$, $t = 1500$, $(7, 494\text{s})$. (c) $\ell = 7$, $t = 2296$, $(11, 5732\text{s})$. (d) $\ell = 7$, $t = 4593$, $(8, 12347\text{s})$.

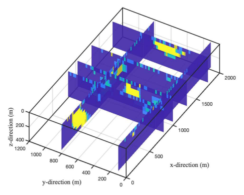


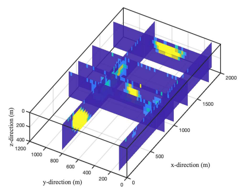
(e) RSVD: $(6, 216\text{s})$. (f) $(6, 436\text{s})$. (g) $(7, 3311\text{s})$. (h) $(7, 6979\text{s})$.

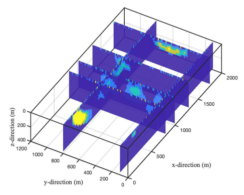
Figure 5.6: For `gravity` data the predicted anomalies (mGal) obtained using GKB in Figures 5.6a-5.6d and RSVD in Figures 5.6e-5.6h. The first row for GKB indicates the choices of ℓ and t in each column: $t = 750$ and $t = 1500$ for $\ell = 4$, and $t = 2296$ and $t = 4593$ for $\ell = 7$, corresponding to $t = \textbf{floor}(m/8)$ and $t = \textbf{floor}(m/4)$ for $(m, n) = (6000, 48000)$ and $(18375, 257250)$, respectively. Captions contain pairs $(K, \texttt{Costs})$, (number of iterations to convergence and computational cost in seconds). See Tables 5.4 and 5.3.

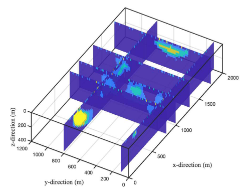
5.2.5 Illustrating Solutions with Increasing ℓ and t

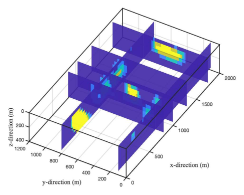
First, a comparison is given for a set of solutions for which the timing results were compared in Section 5.2.4. Figures 5.6 and 5.6b illustrate the predicted anomalies and reconstructed volumes for `gravity` data inverted by both algorithms, with resolutions given by $\ell = 4$ and $\ell = 7$ with $t = \textbf{floor}(m/8)$ and $t = \textbf{floor}(m/4)$. For the cases using $\ell = 4$ it can be seen that the predicted anomalies are generally less accurate than with $\ell = 7$. Moreover, there is little deterioration in the anomaly predictions when using $t = \textbf{floor}(m/8)$ instead of $t = \textbf{floor}(m/4)$, except that the results with the GKB show more residual noise. On the other hand, it is more apparent from consideration of the reconstructed volumes shown in Figures 5.7a-5.7h that the RSVD algorithm does yield better results in all cases, and specifically the high resolution $\ell = 7$ results are very good, even using $t = \textbf{floor}(m/8)$. When including the consideration of the computational cost, it is clear that if using $\ell = 7$ it is sufficient to use $t = \textbf{floor}(m/8)$ and the RSVD algorithm, but that a reasonable result may even be obtained using the same algorithm but with $\ell = 4$ and requiring less than 5 minutes of compute time.

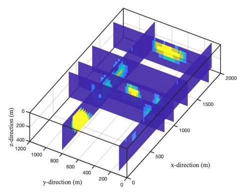

(a) GKB: $(.76, .90)$.

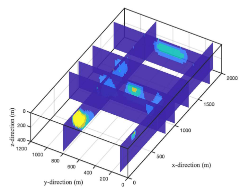

(b) $(.64, .96)$.


(c) $(.75, .81)$.


(d) $(.70, 1.00)$.


(e) RSVD: $(.57, .96)$.


(f) $(.57, .93)$.


(g) $(.60, .95)$.

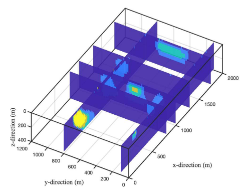
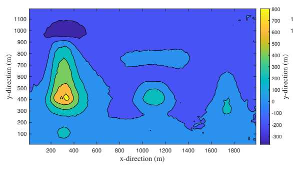
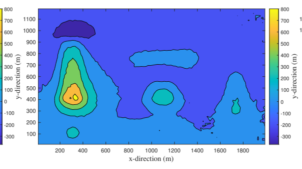
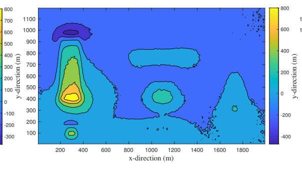
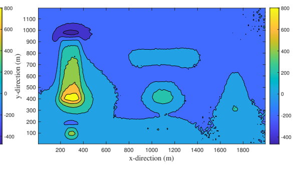
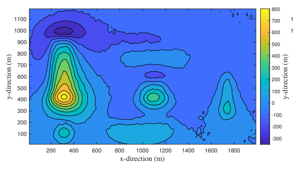
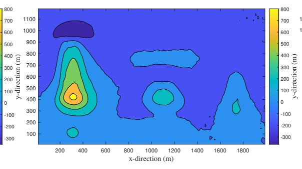
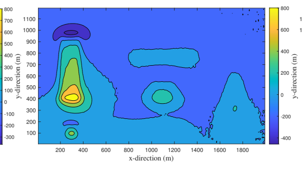
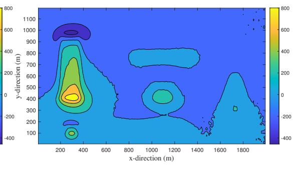

(h) $(.61, .91)$.

Figure 5.7: For gravity data the reconstructed volumes obtained using GKB in Figures 5.7a-5.7d and RSVD in Figures 5.7e-5.7h, corresponding to Figures 5.6a-5.6d and Figures 5.6e-5.6h, respectively. The first row for GKB indicates the choices of ℓ and t in each column as provided in Figure 5.6. In the captions are pairs $(\text{RE}, \chi^2/(m + \sqrt{2m}))$. Results for all cases are summarized in Table 5.4 with timings in Table 5.3.

The results for the inversion of the magnetic data are illustrated in Figure 5.8 for the same cases as for the inversion of gravity data in illustrated in Figure 5.6. Now, in contrast to the gravity results, the predicted anomalies are in good agreement with the true data for the results obtained using the GKB algorithm, with apparently greater accuracy for the lower resolution solutions, $\ell = 4$ for both choices of t. On the other hand, the predicted magnetic anomalies are less satisfactory for small ℓ and t but acceptable for large ℓ. Then, considering the reconstructed volumes, there is a lack of resolution for $\ell = 4$ which is evidenced by the loss of the small well near the surface, which is seen when $\ell = 7$ for both cases of t, when using the GKB. The other structures in the domain are also resolved better with $\ell = 7$, but there is little gain from using $t = \textbf{floor}(m/4)$ over $t = \textbf{floor}(m/8)$. Then, considering the reconstructions obtained using the `RSVD` algorithm, while it is clear that the result with $\ell = 4$ and small t is unacceptable, the anomaly and reconstructed volume with $\ell = 4$ and $t = \textbf{floor}(m/4)$ is acceptable and achieved in reasonable time, approximately 11 minutes, far faster than using $\ell = 7$ with `GKB`. Thus, this may contradict the conclusion that one should use the `GKB` algorithm within the `magnetic` data inversion algorithm. If there is a large amount of data and a high resolution volume is required, then it is important to use `GKB` in order to limit computational cost. Otherwise, it can be sufficient to use the `RSVD` provided $t \geq \textbf{floor}(m/8)$ for a coarser resolution solution obtained at reasonable computational cost.

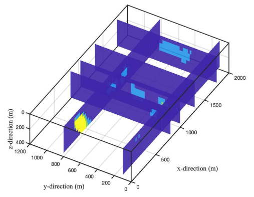





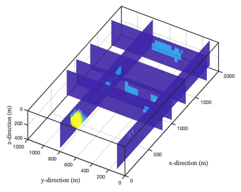
(a) `GKB`: $\ell = 4, t = 750$, $(5, 136\text{s})$. (b) $\ell = 4, t = 1500$, $(5, 343\text{s})$. (c) $\ell = 7, t = 2296$, $(7, 3809\text{s})$. (d) $\ell = 7, t = 4593$, $(7, 10979\text{s})$.

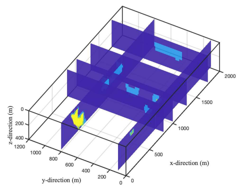





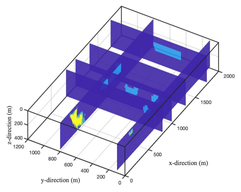
(e) `RSVD`: $(25, 883\text{s})$. (f) $(9, 650\text{s})$. (g) $(14, 6618\text{s})$. (h) $(13, 12949\text{s})$.

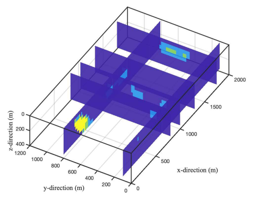
Figure 5.8: For `magnetic` data the predicted anomalies (nT) obtained using `GKB` in Figures 5.8a-5.8d and `RSVD` in Figures 5.8e-5.8h. The first row for `GKB` indicates the choices of ℓ and t in each column: $t = 750$ and $t = 1500$ for $\ell = 4$, and $t = 2296$ and $t = 4593$ for $\ell = 7$, corresponding to $t = \textbf{floor}(m/8)$ and $t = \textbf{floor}(m/4)$ for $(m, n) = (6000, 48000)$ and $(18375, 257250)$, respectively. In the captions are pairs $(K, \texttt{Costs})$, (number of iterations to convergence and computational cost in seconds). Results for all cases are summarized in Table 5.5 with timings in Table 5.3.

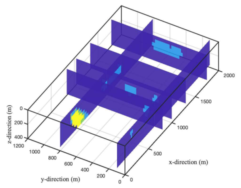

(a) GKB: $\ell = 4, t = 750$, $(.63, .90)$.

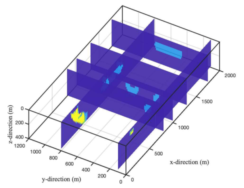

(b) $\ell = 4, t = 1500$, $(.63, .91)$.


(c) $\ell = 7, t = 2296$, $(.67, .90)$.


(d) $\ell = 7, t = 4593$, $(.68, .92)$.


(e) RSVD: $(.64, 1.11)$.


(f) $(.63, .93)$.


(g) $(.70, .99)$.

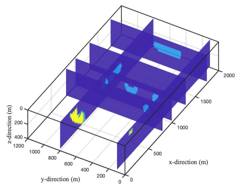
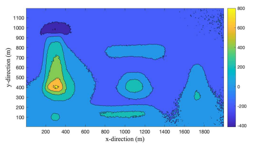
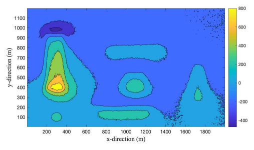
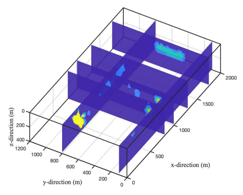

(h) $(.69, .90)$.

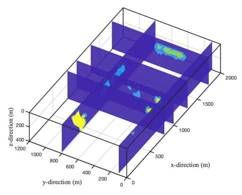
Figure 5.9: For magnetic data the reconstructed volumes obtained using GKB in Figures 5.9a-5.9d and RSVD in Figures 5.9e-5.9h, corresponding to Figures 5.8a-5.8d and Figures 5.8e-5.8h, respectively. The first row for GKB indicates the choices of ℓ and t in each column, as provided in Figure 5.8. In the captions are pairs $(\texttt{RE}, \chi^2/(m + \sqrt{2m}))$. Results for all cases are summarized in Table 5.5 with timings in Table 5.3.

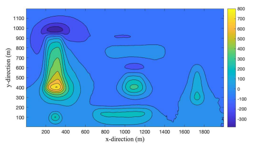
The quality of solutions obtained is now investigated for magnetic data using higher resolution data sets, and both GKB and RSVD algorithms to assess which algorithm is best suited for such larger problems. In these cases $t = \textbf{floor}(m/20)$ is chosen, to assess quality with a necessarily restricted subspace size as compared to the size of the given data set. Results using $\ell = 11$ with $t = 2268$ and $\ell = 12$ with $t = 2700$, corresponding to $m = 45375$ and $n = 998250$, and $m = 54000$ and $n = 1296000$, respectively, are illustrated in Figure 5.10. For these large scale problems, the memory becomes too large for implementation on the environment with just 32GB RAM. Thus, these timings are for an implementation using a desktop computer with the Intel(R) Xeon (R) Gold 6138 CPU 2.00GHz chip and with MATLAB release 2019b. Comparing the results between $m = 45375$ and $m = 54000$ ($\ell = 11$ and $\ell = 12$) it can be seen that the predicted anomalies are always better for the larger problem, and in particular the result shown in Figure 5.10e shows greater artifacts when using RSVD. The obtained reconstruction for this case, shown in Figure 5.10g is, however, acceptable. Overall, trading off between computational cost and solution quality, there seems little gain in using $\ell = 12$ and the results with $\ell = 11$ obtained with the GKB algorithm in 227 minutes (nearly 4 hours) are suitable. These results also show that it is sufficient to use a relatively smaller projected space, $t = \textbf{floor}(m/20)$ when m is larger. Indeed, notice that even in these cases the largest matrix required by both algorithms is of size $n \times t_p$ and requires 17.7GB and 27.4GB, for $\ell = 11$ and $\ell = 12$, respectively. Effectively, it is this large memory requirement that limits the given implementation using either GKB or RSVD for larger size problems.

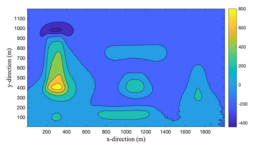

(a) $11 : (9, 13595\text{s})$.

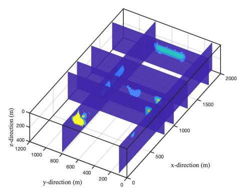

(b) $12 : (8, 21649\text{s})$.


(c) $11 : (.74, .98)$.


(d) $12 : (.74, .92)$.


(e) $(15, 21266\text{s})$.


(f) $(16, 41981\text{s})$.


(g) $(.74, .99)$.

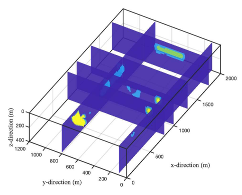
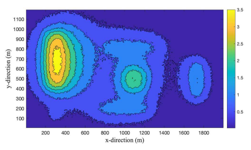
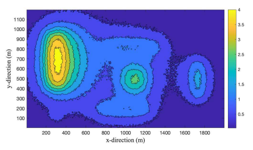

(h) $(.73, .98)$.

Figure 5.10: The magnetic anomalies (nT) and reconstructed volumes using the GKB and RSVD algorithms in Figures 5.10a-5.10d and 5.10e-5.10h, respectively. The first row indicates the choice of $\ell = 11$, for which $t = 2268 = \textbf{floor}(m/20)$, $m = 45375$ and $n = 998250$ and oversampled projected problem of size 2381, or $\ell = 12$, with $t = 2700 = \textbf{floor}(m/20)$, $m = 54000$ and $n = 1296000$, and oversampled projected problem of size 2835. In the captions are pairs $(K, \texttt{Costs})$, (number of iterations to convergence and computational cost in seconds) for the anomalies, and $(\text{RE}, \chi^2/(m + \sqrt{2m}))$ for the reconstructions.

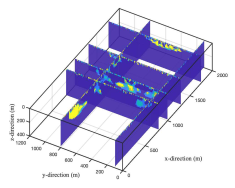
Numerical experiments for the inversion of `gravity` data, similar to the testing for the `magnetic` data, demonstrates that indeed the `RSVD` algorithm with power iteration is to be preferred for the inversion of `gravity` data, yielding acceptable solutions at lower cost than when using the `GKB` algorithm. Representative results are detailed in Figure 5.11 for the same parameter settings as given in Figure 5.10 for the `magnetic` problem.



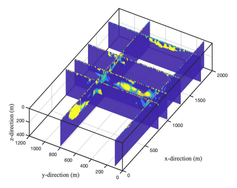
(a) $11 : (21, 30833\text{s})$.



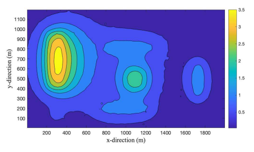
(b) $12 : (21, 59837\text{s})$.



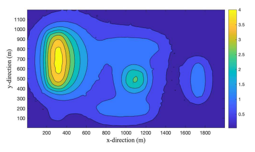
(c) $11 : (1.02, .76)$.



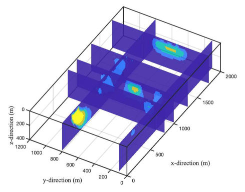
(d) $12 : (1.00, .78)$.



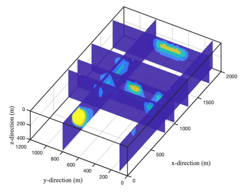
(e) $(8, 11779\text{s})$.



(f) $(8, 30212\text{s})$.



(g) $(.61, 1.00)$.



(h) $(.59, .99)$.

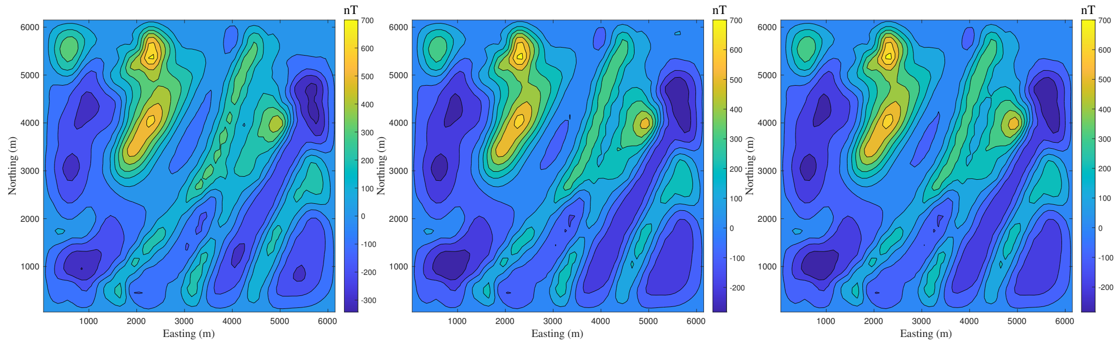
Figure 5.11: The `gravity` anomalies (mGal) and reconstructed volumes using the `GKB` and `RSVD` algorithms in Figures 5.11a-5.11d and 5.11e-5.11h, respectively. The first row indicates the choice of $\ell = 11$, for which $t = 2268 = \textbf{floor}(m/20)$, $m = 45375$ and $n = 998250$ and oversampled projected problem of size 2381, or $\ell = 12$, with $t = 2700 = \textbf{floor}(m/20)$, $m = 54000$ and $n = 1296000$, and oversampled projected problem of size 2835. In the captions are pairs $(K, \texttt{Costs})$, (number of iterations to convergence and computational cost in seconds) for the anomalies, and $(\texttt{RE}, \chi^2/(m+\sqrt{2m}))$ for the reconstructions.

5.2.6 Real Data

Validation of the simulated results on a practical data set is applied by the `GKB` algorithm for the inversion of a `magnetic` field anomaly that was collected over a portion of the Wuskwatim Lake region in Manitoba, Canada. This data set was discussed in Pilkington [2009] and also used in Vatankhah et al. [2020a] for inversion using the `RSVD` algorithm with a single power iteration. Further details of the geological relevance of this data set is given in these references. Moreover, its use makes for direct comparison with these existing results. Here a grid is used of $62 \times 62 = 3184$ measurements at 100m intervals in the East-North direction with padding of 5 cells yielding a horizontal cross section of size 72×72 in the East-North directions. The depth dimension is discretized with $\Delta z = 100$m, yielding a regular cube, to $\Delta z = 8$m for rectangular prisms with a smaller edge length in the depth dimension for a total depth of 2000m, and providing increasing values of n from 103680 to 1238976 as detailed in Table 5.6. The given `magnetic` anomaly is illustrated in Figure 5.12a.

In each inversion the `GKB` algorithm is run with $t = 480$, corresponding to $t = \textbf{floor}(m/8)$, where $m = 3184$ and oversampled projected space of size 504, and a noise distribution based on (4.10) is employed using $\tau_1 = .02$ and $\tau_2 = .018$. All inversions converge to the tolerance $\chi^2/(m + \sqrt{2m})) < 1$ in no more than 19 iterations for all problem sizes, as given in Table 5.6. The computational cost measured in seconds is also given in Table 5.6 and demonstrates that it is feasible to invert for large parameter volumes, in times ranging from just under 5 minutes for the coarsest resolution, to just over 73 minutes for the volume with the highest resolution. Here the computations are performed on a MacBook Pro laptop with 2.5 GHz Dual-Core Intel Core i7 chip and 16GB memory. Figure 5.13a shows that the `UPRE` function has a well-defined minimum at the final iteration for all resolutions, and Figure 5.13b shows that the convergence of the scaled χ^2 value is largely independent of n. The final regularization parameter $\alpha^{(K)}$ decreases with increasing n, while the initial α found using (5.5) increases with n, as reported in Table 5.6.

Results of the inversion, for the coarsest and finest resolutions are presented in Figures 5.12, 5.14 and 5.15, for anomalies, reconstructed volumes, and depth slices through the volume domain, respectively. First, from Figures 5.12b-5.12c, as compared to Figure 16b in Vatankhah et al. [2020a], it is evident that the predicted anomalies provide better

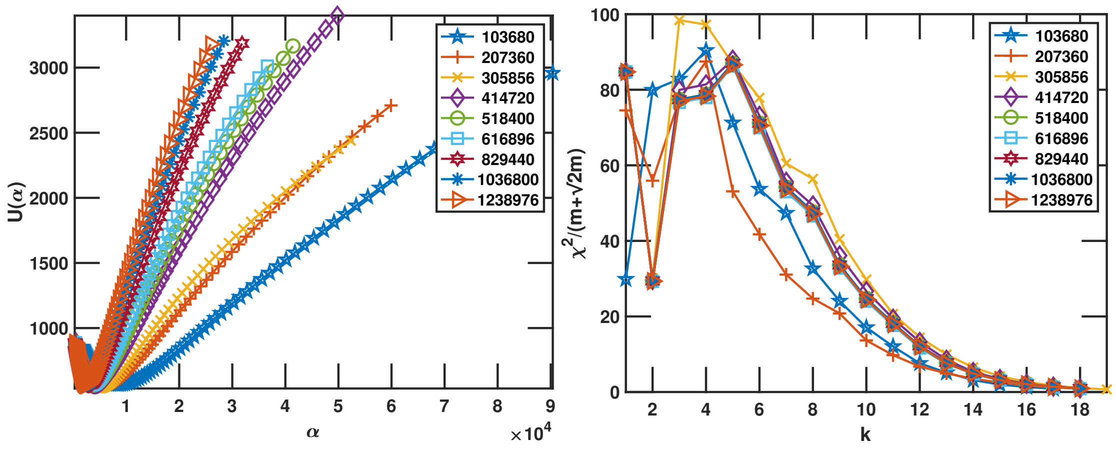


(a) Given anomaly. (b) Predicted: $n = 103680$. (c) Predicted: $n = 1238976$.

Figure 5.12: The given `magnetic` anomaly in Figure 5.12a and the obtained predicted anomalies for the inversion using the parameters for the first and last lines of data in Table 5.6 in Figures 5.12b-5.12c, respectively.

Table 5.6: Inversion of `magnetic` data as illustrated in Figure 5.12 for $m = 3844$ on a grid of 62×62 stations, with $\Delta x = \Delta y = 100$m and padding of 5 cells in both x and y-directions, yielding blocks of size $n_r = 5184$. The inversion uses the `GKB` algorithm with $t = 480$ (**floor**$(m/8)$) and $t_p = 504$. The noise in the algorithm uses (4.10) as given for the simulations with $\tau_1 = .02$ and $\tau_2 = .018$. These results are obtained using a MacBook Pro laptop with 2.5 GHz Dual-Core Intel Core i7 chip and 16GB memory.

n	n_z	Δz	K	$\alpha^{(1)}$	$\alpha^{(K)}$	$\chi^2/(m+\sqrt{2m})$	Cost(s)
103680	20	100	17	$4.60e+05$	8558	0.87	334
207360	40	50	18	$5.36e+06$	5887	0.90	754
305856	59	33	19	$2.07e+07$	4930	0.70	1126
414720	80	25	18	$6.09e+07$	4116	0.95	1513
518400	100	20	18	$1.33e+08$	3701	0.94	2018
616896	119	16	18	$2.43e+08$	3386	0.90	2095
829440	160	12	18	$6.90e+08$	2933	0.95	3091
1036800	200	10	18	$1.51e+09$	2627	0.95	3690
1238976	239	8	18	$2.80e+09$	2396	0.96	4389

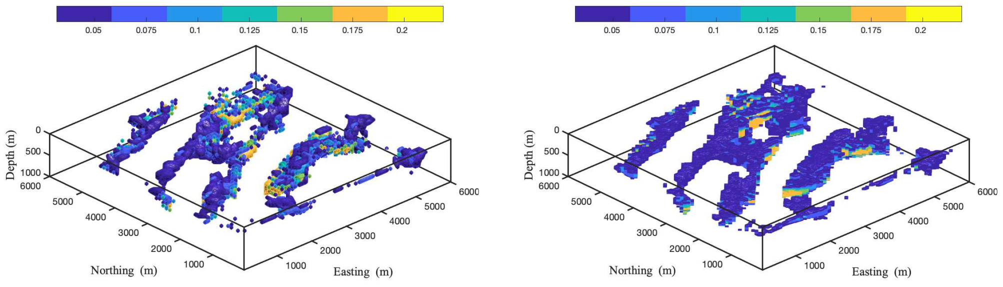


(a) Regularization function for K.

(b) χ^2 with k.

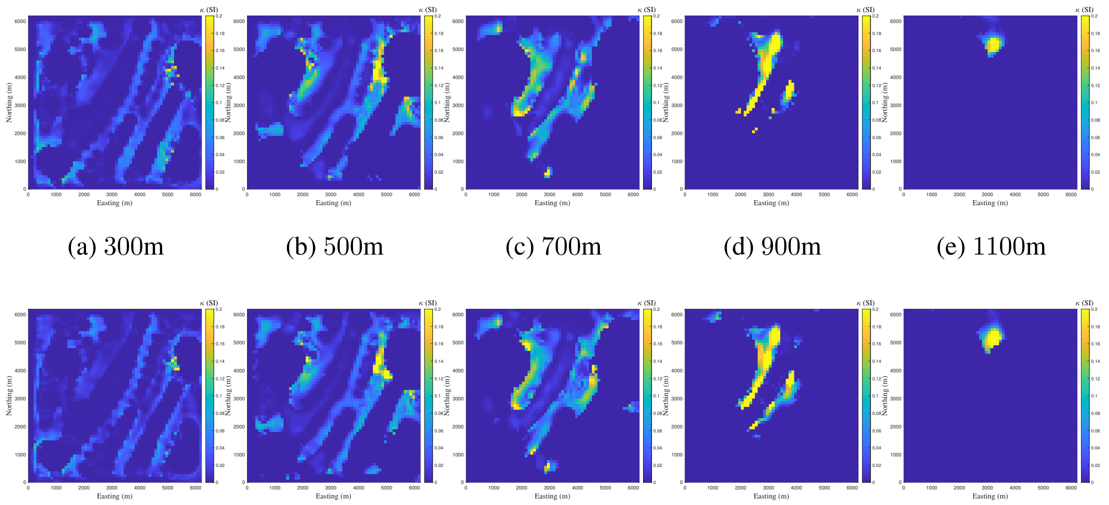
Figure 5.13: The plot of the regularization function $U(\alpha)$ for the UPRE algorithm, at the final iteration K for increasing values of n as indicated in Table 5.6 in Figure 5.13a and the progression of the scaled χ^2 estimate as a function of iteration k and for increasing n in Figure 5.13b.

agreement to the measured anomaly, with respect to structure and the given values. Moreover, more structure is seen in the volumes presented in Figures 5.14a-5.14b as compared to Figure19 Vatankhah et al. [2020a], and the increased resolution provides greater detail in Figure 5.14bas compared to Figure 5.14a. Here the volumes are presented for the depth from 0 to 1000m only, but it is seen in Figures 5.15e and 5.15j, which are the slices at depth 1100m, that there is little structure evident at greater depth. Comparing the depth slices for increasing depth shows that the use of the higher resolution leads to more structure at increased depth. Moreover, the results are consistent with those presented in Vatankhah et al. [2020a] for the use of the RSVD for a projected size $t = 1100$ as compared to $t = 480$ used here. It should also be noted that the RSVD algorithm with one power iteration does not converge within 50 steps, under the same configurations for m, n and t.



(a) Iso-surface using $n = 103680$. (b) Iso-surface using $n = 1238976$.

Figure 5.14: The reconstructed volumes showing parameters $\kappa > 0.05$ and depth from 0 to 1000, corresponding to the predicted anomalies in Figure 5.12.



(a) 300m (b) 500m (c) 700m (d) 900m (e) 1100m

(f) 300m (g) 500m (h) 700m (i) 900m (j) 1100m

Figure 5.15: Slices through the volumes illustrated in Figure 5.14 for depths 300, 500, 700, 900 and 1100, for $n = 103680$ in Figures 5.15a-5.15e and for $n = 1238976$ in Figures 5.15f-5.15j.

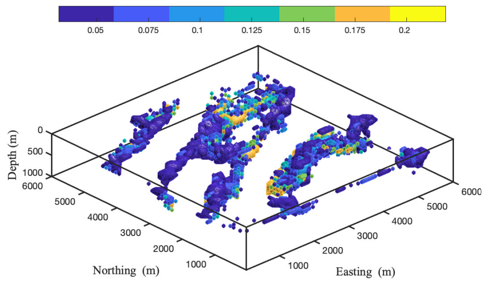
5.2.7 Comparison to RPS

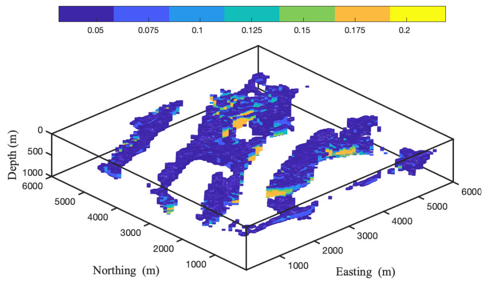
Here MATLAB release 2019b is implemented on the same cores as Section 4.5.5 using an Intel(R) Xeon(R) Gold 6138 processor (2.00GHz) and 256GB RAM for all results.

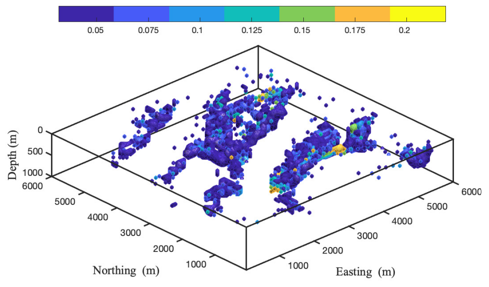
Comparisons in Table 5.7 show that the RPS algorithm reaches convergence faster and with a similar number of iterations compared to the RSVD algorithm, but requires more computational time and number of iterations compared to the GKB algorithm.

Table 5.7: Inversion of magnetic data using RPS with t as the truncation parameter as illustrated in Figure 4.11, GKB and using RSVD with one power iteration, for $m = 3844$ on a grid of 62×62 stations, with $\Delta x = \Delta y = 100$m and padding of 5 cells in both x and y-directions, yielding blocks of size $n_r = 5184$. The inversion uses $t = 384$ (**floor**$(m/10)$) and $t_p = 403$. The noise in the algorithm uses (4.10) as given for the simulations with $\tau_1 = .02$ and $\tau_2 = .018$.

method	n	n_z	Δz	K	$\chi^2/(m+\sqrt{2m})$	Cost(s)
RPS t	103680	20	100	19	0.98	398
	207360	40	50	17	0.99	716
	305856	59	33	17	0.94	1082
	414720	80	25	17	0.97	1472
	518400	100	20	17	0.99	1806
	616896	119	16	17	0.93	2246
	829440	160	12	17	0.94	3031
	1036800	200	10	17	0.97	3729
	1238976	239	8	17	0.99	4485
GKB	103680	20	100	17	0.87	376
	207360	40	50	18	0.90	749
	305856	59	33	19	0.70	1156
	414720	80	25	18	0.95	1469
	518400	100	20	18	0.94	1790
	616896	119	16	18	0.90	2092
	829440	160	12	18	0.95	2785
	1036800	200	10	18	0.95	3323
	1238976	239	8	18	0.96	4001
RSVD	103680	20	100	27	1.19	903
	207360	40	50	21	0.99	1401
	305856	59	33	23	0.99	2306
	414720	80	25	25	1.00	3423
	518400	100	20	22	1.00	3725
	616896	119	16	24	1.00	4923
	829440	160	12	25	1.00	7007
	1036800	200	10	23	0.99	8012
	1238976	239	8	27	1.00	11258


(a) GKB: Iso-surface using $n = 103680$.


(b) Iso-surface using $n = 1238976$.


(c) RPS: Iso-surface using $n = 103680$.

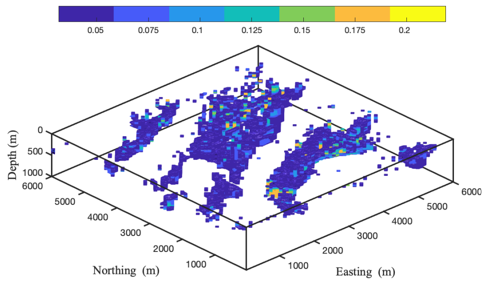

(d) Iso-surface using $n = 1238976$.

Figure 5.16: The reconstructed volumes showing parameters $\kappa > 0.05$ and depth from 0 to 1000, corresponding to the predicted anomalies in Figure 5.12.

5.3 Conclusions

Two algorithms, GKB and RSVD, for the focused inversion of potential field data with all operations for the sensitivity matrix G implemented using a fast 2DFFT algorithm have been developed and validated for the inversion of both gravity and magnetic data sets. The results show first that it is distinctly more efficient to use the 2DFFT for operations with matrix G rather than direct multiplication. This is independent of algorithm and data set, for all large scale implementations considered. Moreover, the implementation using the 2DFFT makes it feasible to solve these large scale problems on a standard desktop computer without any code modifications to handle multiple cores or GPUs, which is not possible due to memory constraints when m and n increase. While both algorithms are improved with this implementation, the results show that the impact on the GKB efficiency is greater than that on the RSVD efficiency. A theoretical analysis of the computational cost of each algorithm for a single iterative step demonstrates that the GKB should be faster, but this is not always realized in practice as the problem size increases, with commensurate increase in the size of the projected space. Then, the efficiency of GKB deteriorates, and the advantage of using builtin routines from MATLAB for the RSVD algorithm is crucial.

When considering the computational cost to convergence for both algorithms, which also then includes the cost due to the requiring projected spaces that are of reasonable size relative to m, the results confirm earlier published results that it is more efficient to use RSVD, with $t \geq$ **floor**$(m/8)$ for inversion of gravity data. Moreover, generally larger projected spaces are required when using RSVD for the inversion of magnetic data. On the other hand, prior published work did not contrast GKB with RSVD for the inversion of magnetic data. Here, results contribute a new conclusion to the literature, namely that GKB is more efficient for these large-scale problems and can use also $t \approx$ **floor**$(m/8)$ rather than larger spaces for use with RSVD. Further, GKB is also more efficient than RPS, with RPS more computationally efficient than LSRN. Critically, which algorithm to use is determined by the spectral space for the underlying problem-specific sensitivity matrix G, as discussed in Vatankhah et al. [2020a]. Moreover, the restriction can be relaxed, $t \approx$ **floor**$(m/8)$; indeed satisfactory results are achieved using $t \approx m/20$ for large problems, for the inversion of magnetic data.

It should be noted that equivalent conclusions can be made when the implementations use padding, only that generally fewer iterations to convergence are required. Furthermore, all the implementations use the automatic determination of the regularization parameter using the UPRE function. The suitability of the UPRE function was demonstrated in earlier references, and is thus not reproduced here, but results that are not reported here demonstrated that the earlier results still hold for these large scale problems and algorithms. In contrast, `RPS` provides a viable truncation approach for the inversion of real `magnetic` data. Although `GKB` is preferable, `RPS` provides similar timing and similar number of iterations to convergence for all cases, without dependance on either regularization parameter α or truncation parameter κ, provided an appropriate projection space is used.

Chapter 6

OTHER DIRECTIONS IN DIMENSIONALITY REDUCTION FOR INVERSION OF LARGE-SCALE PROBLEMS, CONCLUSIONS AND FUTURE WORK

Overview of Chapter

In this chapter some alternative directions and applications of dimensionality reduction techniques will be reviewed. These have been considered in developing the major topics that were described in Chapters 3-5. An iterative method for approximating the pseudo-inverse will be shown to be much more computationally efficient when applied directly to the system within each iteration. Using a linear interpolation matrix as a dimensionality reduction substitution is shown to produce a viable piecewise linear solution upon projection. Sketching is shown to provide an approximate Kronecker product decomposition via a SVD. Further, the same SVD is shown to provide a decomposition for a sum of approximate Kronecker products. RPS will be shown to produce viable results when applied to a nonlinear groundwater transmissivity problem at reduced computational error and time. Conclusions for this work will be presented, and avenues for future research will be discussed for joint inversion of gravity and magnetic data as well as validation of the sketched Kronecker product decomposition.

The chapter is organized as follows. First, alternative sketching techniques are considered in Sections 6.1-6.2. An existing iterative method for obtaining an approximate pseudo-inverse is discussed in Section 6.1. The iterative method is extended and directly applied to the matrix system in Section 6.2 to reduce computation cost while obtaining a solution. An alternative right sketch based on linear interpolation is developed in Section 6.2. Sketching is applied in a novel way to obtain a Kronecker product decomposition

(KPD) in Section 6.3. Considerations are also given for producing a sum of Kronecker products, producing Kronecker product decompositions for blocks within a matrix, as well as matrix multiplication using Kronecker products for BTTB matrices. `RPS` is applied to a nonlinear groundwater transmissivity application in Section 6.4, and numerical results are given showing a reduction in both error and computational time on a single-core processor in comparison to the randomized geostatistical approach (`RGA`). This provides further validation of the `RPS` method. Conclusions and future work are discussed in Section 6.6.

6.1 Pseudo-Inverse Approximation

Consider an approach that uses an iterative process to obtain $X \approx A^\dagger$, Gower and Richtárik [2017]. The solution can then be estimated as $\mathbf{x} \approx X\mathbf{b}$. Using $A^T = A^T A A^\dagger \approx A^T A X$, and left multiplying by $C^{(i)} = AS^{(i)}$ yields $S^{(i)T}A^T \approx S^{(i)T}A^T A X$ or $C^{(i)T} \approx C^{(i)T}AX$. Minimizing the difference $X^{(i+1)} - X^{(i)}$ subject to this formulation provides the update,

$$X^{(i+1)} = X^{(i)} - A^T C^{(i)}(C^{(i)T}AA^T C^{(i)})^\dagger C^{(i)T}(AX^{(i)} - I).$$

Writing $B^{(i)} = C^{(i)T}A = S^{(i)T}A^T A$ allows this update to be written more concisely as

$$X^{(i+1)} = X^{(i)} - B^{(i)T}(B^{(i)}B^{(i)T})^\dagger(B^{(i)}X^{(i)} - C^{(i)T}). \tag{6.1}$$

Note that at each iteration, X is updated using a different sketch matrix, and X keeps the same dimensions as $A^\dagger$. Thus, even though the sketch helps reduce the cost of computing $A^\dagger$, most of the arithmetic is done using full size matrices. Although this provides a viable result, the computational cost can be just as high as using a direct method on the full size problem.

Note that the right sketch $C = AS$ is left multiplied on A. This creates a sketch on the normal equations $S^T A^T A\mathbf{x} = S^T A^T \mathbf{b}$. Other methods also inherently avoid directly solving a right sketch when applied to a system.

6.1.1 Applying an Approximate Pseudo-Inverse

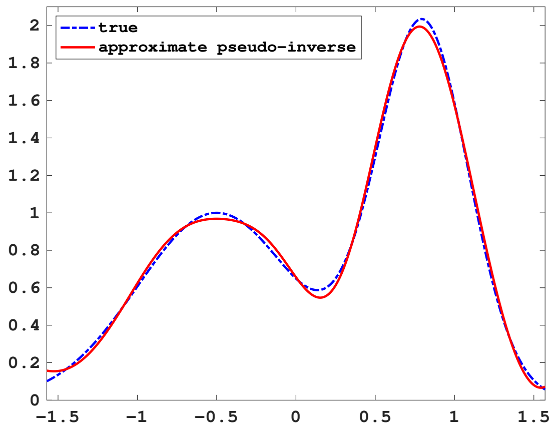
An immediate motivation for applying sketches directly to a system can be shown by re-examining the approximate pseudo-inverse. Since the current interest is in obtaining $X \approx A^{\dagger}$ to obtain the approximate solution $\boldsymbol{x} \approx X\boldsymbol{b}$, as shown in Section 6.1, the iterative pseudo-inverse approximation process can be simplified. An updated iterative method is proposed using $\mathbf{x}^{(i)} = X^{(i)}\mathbf{b}$. Thus, right multiplying (6.1) by $\boldsymbol{b}$ provides the update

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - B^{(i)T}(B^{(i)}B^{(i)T})^{\dagger}(B^{(i)}\mathbf{x}^{(i)} - C^{(i)T}\boldsymbol{b}),$$

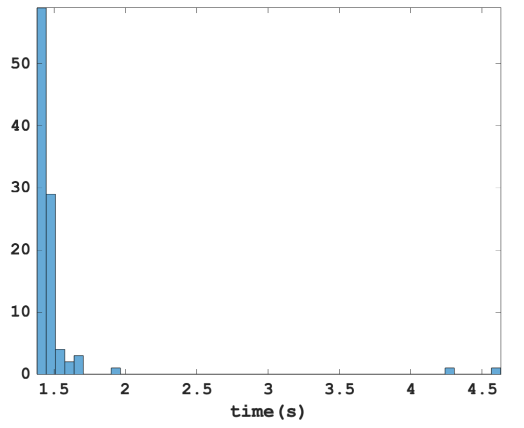
where now the approximate pseudo-inverse is applied iteratively. In addition to $(B^{(i)}B^{(i)T})^{\dagger}$ being performed at reduced computational cost since $B^{(i)}B^{(i)T} \in \mathbb{R}^{s\times s}$, multiplying from right to left also allows the rest of the update to now be performed on a matrix-vector basis, further reducing the computational cost. A new sketch is still formed within each iteration, however, requiring additional computational resources. These results are demonstrated in Figure 6.1 for **shaw** with $m = n = 1\text{e}4$, $s = 20$ and $\eta = 1\text{e}{-3}$. Notice that this shows that all times for the iteratively applied pseudo-inverse are less than a quarter of the time for finding an approximate pseudo-inverse and then calculating the solution.

6.2 Interpolating Sketch

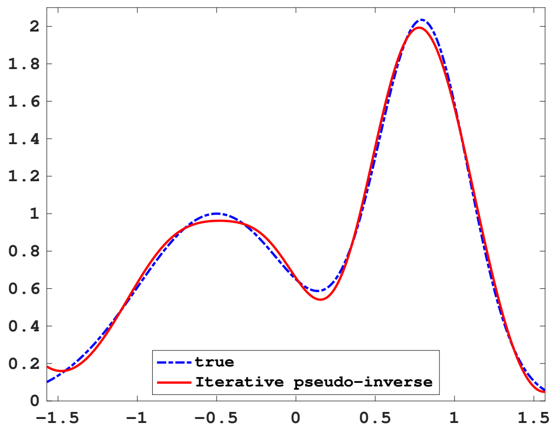
Applying a right random Gaussian sketch fails to provide an accurate approximation of $\mathbf{x}$. Thus, shifting the focus to accurately projecting $\mathbf{y}$ into a larger space leads to a more structured and reliable right sketching matrix. A simple linear interpolation matrix is presented that can be utilized to project $\mathbf{y}$ into an n-length vector. Here, the sketch matrix S is a block diagonal matrix, with each block containing two column vectors. The first vector is a set of linearly spaced points in descending order, and the second vector contains the same points in ascending order. S is then a sparse matrix, which often decreases the



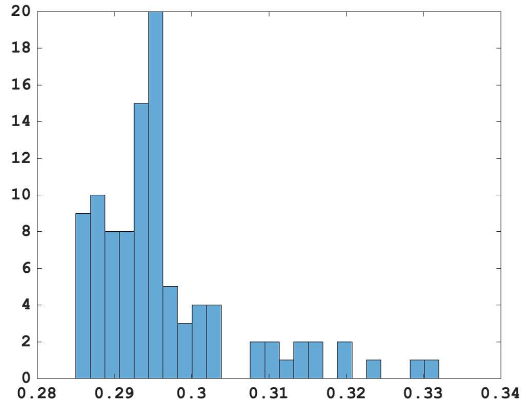
(a) approximate pseudo-inverse



(b) approximate pseudo-inverse



(c) iteratively applied pseudo-inverse



(d) iteratively applied pseudo-inverse

Figure 6.1: The solution of **shaw** with $m = n = 1\text{e}4$ with $s = 20$ and $\eta = 1\text{e}{-3}$ using an approximate pseudo-inverse is shown in Figure 6.1a, and using an iteratively applied pseudo-inverse is shown in Figure 6.1c. The timings for 100 trials using an approximate pseudo-inverse and using the iteratively applied pseudo-inverse are shown in Figure 6.1b and Figure 6.1d respectively.

computational time for matrix multiplication. A simple example with $S \in \mathbb{R}^{5\times 3}$ produces

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}.$$

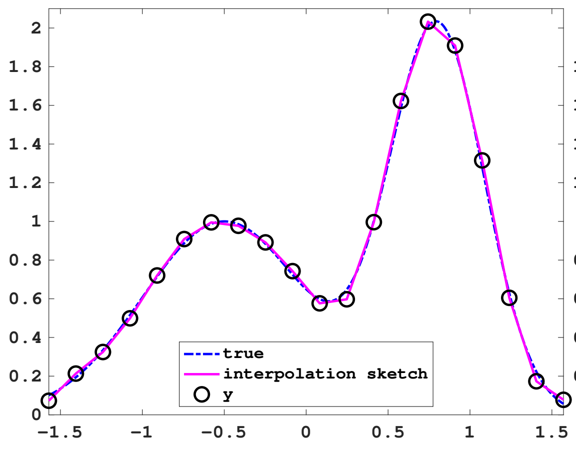
Now, after taking the sketch $C = AS$ and solving $C\boldsymbol{y} = \boldsymbol{b}$, the projection $\boldsymbol{x} = S\boldsymbol{y}$ provides a linear interpolation. This process is summarized in Algorithm 16.

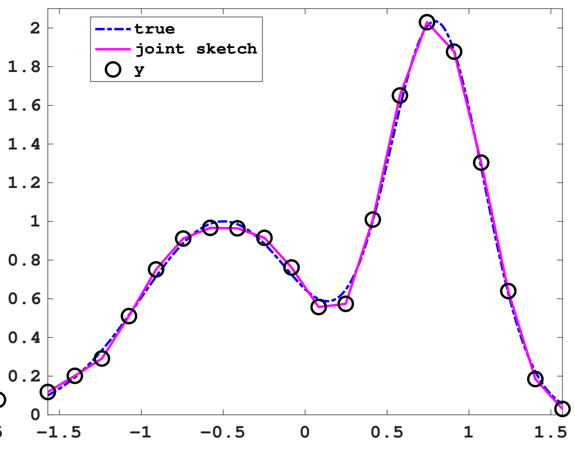
```
**Input:** $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, sketch size $s \ll \textbf{min}(m, n)$
$S = 0 \in \mathbb{R}^{n \times s}$;
$\alpha_2 = 1$;
**for** $j = 1$ **to** $s - 1$ **do**
    $\alpha_1 = \alpha_2$;
    $\alpha_2 = \text{floor}(jn/(s-1))$;
    $\boldsymbol{v} = \text{linspace}(0, 1, \alpha_2 - \alpha_1 + 1)$;
    $S(\alpha_2 : -1 : \alpha_1, j) = \boldsymbol{v}$;
    $S(\alpha_1 : \alpha_2, j+1) = \boldsymbol{v}$;
**end**
$C = AS$;
$\boldsymbol{y} = C_\alpha^\dagger \boldsymbol{b}$;
$\boldsymbol{x} = S\boldsymbol{y}$;
```

Algorithm 16: Interpolating sketch

The projection $\mathbf{x} = S\boldsymbol{y}$ produces a much more stable solution, and the accuracy at a smaller subset of points is based on solving for $\mathbf{y}$. While using linear interpolation allows for a more stable projection, a piecewise linear solution based on $\mathbf{y}$ is not typically the best solution of $A\boldsymbol{x} \approx \boldsymbol{b}$. Similarly, higher order approximations that account for first derivative, second derivative, and so forth, are not guaranteed to provide an accurate solution for $A\boldsymbol{x} \approx \boldsymbol{b}$. The higher order approximations merely enforce a higher degree of smoothness between the points in $\boldsymbol{y}$. The sparsity of S also diminishes as higher oder approximations are implemented.

Note that when using a viable projection, it is possible to implement a joint sketch $S_1 A S_2 \boldsymbol{y} = \boldsymbol{b}$ and maintain the same level of accuracy. S_1 is used as a random Gaussian sketch matrix and S_2 as a linear interpolation matrix in Figure 6.2 for **shaw** with $m = n = 1\text{e}4$ and $s = 20$, which shows that in practice, the values of $\mathbf{y}$ typically fall along the solution regardless of whether a left sketch is also implemented.





(a) right interpolation (b) left Gaussian and right interpolation

Figure 6.2: Solution of **shaw** with $m = n = 1\text{e}4$ and $s = 20$. Only a linear interpolation matrix is used as a right sketch to produce the solution in Figure 6.2a, and a randomized left sketch is also used to produce the solution in Figure 6.2b. In both cases the solution y obtained before the final projection is also shown.

6.3 Kronecker Product Decomposition

The Kronecker product, defined by

$$D \otimes B = \begin{bmatrix} d_{11}B & \cdots & d_{1n}B \\ \vdots & \ddots & \vdots \\ d_{m1}B & \cdots & d_{mn}B \end{bmatrix},$$

is a dimensionality reduction technique that takes advantage of structure within a matrix, and has been growing in popularity since around 1858, as summarized in Schäcke [2013]. Obtaining an approximate Kronecker product decomposition (KPD) $B \otimes D \approx A \in \mathbb{R}^{m \times n}$, however, can be computationally expensive. If $B \in \mathbb{R}^{m_1 \times n_1}$ and $D \in \mathbb{R}^{m_2 \times n_2}$ can be found, with $m_1 m_2 = m$ and $n_1 n_2 = n$, the cost of matrix operations can be reduced using B and D. In particular, given $\text{vec}(X) = \boldsymbol{x}$, then $A\boldsymbol{x} \approx (B \otimes D)\boldsymbol{x} = \text{vec}(DXB^T)$. In addition, $(B \otimes D)^\dagger = B^\dagger \otimes D^\dagger$, enabling the system $A\boldsymbol{x} \approx \boldsymbol{b}$ to be solved more efficiently.

The SVD of a rearrangement of A provides the necessary components of the KPD, Van Loan and Pitsianis [1993]. Given

$$A = \left[\begin{array}{c|c|c} A_{11} & \cdots & A_{1n_1} \\ \hline \vdots & \ddots & \vdots \\ \hline A_{m_1 1} & \cdots & A_{m_1 n_1} \end{array}\right],$$

and with $\text{vec}(A) = [A_1^T A_2^T \cdots A_n^T]^T$, define the rearrangement of A as

$$\tilde{A} = \begin{bmatrix} \text{vec}(A_{11})^T \\ \vdots \\ \text{vec}(A_{m_1 1})^T \\ \vdots \\ \text{vec}(A_{1n_1})^T \\ \vdots \\ \text{vec}(A_{m_1 n_1})^T \end{bmatrix}.$$

A simple example with

$$A = \left[\begin{array}{cc|cc} a_1 & a_3 & a_9 & a_{11} \\ a_2 & a_4 & a_{10} & a_{12} \\ \hline a_5 & a_7 & a_{13} & a_{15} \\ a_6 & a_8 & a_{14} & a_{16} \end{array}\right]$$

has the rearrangement

$$\tilde{A} = \left[\begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \\ \hline a_5 & a_6 & a_7 & a_8 \\ \hline a_9 & a_{10} & a_{11} & a_{12} \\ \hline a_{13} & a_{14} & a_{15} & a_{16} \end{array}\right].$$

The following results implement the rearrangement of A to obtain a KPD.

Theorem 6.3.1. *[Van Loan and Pitsianis, 1993, Theorem 2.1] Assume that $A \in \mathbb{R}^{m\times n}$ with $m = m_1m_2$ and $n = n_1n_2$. If $B \in \mathbb{R}^{m_1\times m_2}$ and $D \in \mathbb{R}^{m_2\times n_2}$, then*

$$\|A - B \otimes D\|_F = \|\tilde{A} - vec(B)vec(D)^T\|_F.$$

Corollary 6.3.2. *[Van Loan and Pitsianis, 1993, Corollary 2.2] Assume that $A \in \mathbb{R}^{m\times n}$ with $m = m_1m_2$ and $n = n_1n_2$. If the rearrangement of A has the SVD*

$$\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T,$$

where $\tilde{\sigma}_1$ is the largest singular value, and $\tilde{\mathbf{u}}_1$ and $\tilde{\boldsymbol{v}}_1$ are the corresponding singular vectors, then the matrices $B \in \mathbb{R}^{m_1\times m_2}$ and $D \in \mathbb{R}^{m_2\times n_2}$ defined by $vec(B) = \tilde{\sigma}_1\tilde{\mathbf{u}}_1$ and $vec(D) = \tilde{\boldsymbol{v}}_1$ minimize $\|A - B \otimes D\|_F$.

Now, by Theorem 6.3.1 and corollary 6.3.2, a rank 1 approximation of $\tilde{A}$ provides B and D. This result is immediate for Kronecker rank r A by applying $\|A\|_F^2 = \sum_{i=1}^r \sigma_i^2$, [Golub and Van Loan, 2013, Corollary 2.4.3], and thus $\|A - B \otimes D\|_F^2 = \sum_{i=2}^r \sigma_i^2$.

6.3.1 Sketched KPD

Since the elementary technique uses a rank 1 SVD, sketching can be implemented to significantly reduce the computational cost. First, the dimensionality of the rearrangement of A is reduced by the sketch $\tilde{C} = \tilde{A}S$, with random Gaussian $S \in \mathbb{R}^{N\times s}$ and sketch size $s = k + p$, where $k = 1$ is a truncation parameter and p is an oversampling factor. Note that the sketch can be applied iteratively across each row of $\tilde{C}$ by sketching each vectorized block of A one at a time, reducing memory requirements when A is large.

Lemma 6.3.3. *Given $A \in \mathbb{R}^{m\times n}$ with $m_1m_2 = m$ and $n_1n_2 = n$, rearrangement $\tilde{A} \in \mathbb{R}^{m_1n_1\times m_2n_2}$, Gaussian sketch matrix $S \in \mathbb{R}^{m_2n_2\times s}$ with $s = 1 + p$ for oversampling factor p, and sketch $\tilde{C} = \tilde{A}S = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, then*

$$\|(I - P_{\tilde{C}})\tilde{A}\|_2 \leq \left(1 + 17\sqrt{1 + 1/p}\right)\sigma_2 + \frac{8}{\sqrt{s}}\left(\sum_{j>1}\sigma_j^2\right)^{1/2}$$

with failure probability at most $6e^{-p}$*, and*

$$\|(I - P_{\tilde{C}})\tilde{A}\|_2 \leq \left(1 + 8\sqrt{sp\log p}\right)\sigma_2 + 3\sqrt{s}\left(\sum_{j>1}\sigma_j^2\right)^{1/2}$$

with failure probability at most $6p^{-p}$.

Proof. By [Halko et al., 2011, Corollary 10.9], given $p \geq 4$ and random Gaussian sketch matrix S, error bounds

$$\|(I - P_{\tilde{C}})\tilde{A}\|_2 \leq \left(1 + 17\sqrt{1 + k/p}\right)\sigma_{k+1} + \frac{8\sqrt{s}}{p+1}\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$$

for the projection onto the range of $\tilde{C}$ have failure probability at most $6e^{-p}$. □

Note that $P_{\tilde{C}} = \tilde{C}(\tilde{C}^T\tilde{C})^{-1}\tilde{C}^T$ defines the projector onto the range of $\tilde{C}$. Thus, it is assumed $s \geq 5$ for a rank 1 approximation.

```
   Input: $A \in \mathbb{R}^{M\times N}$, $m_1, m_2, n_1, n_2, p$
 1 $s = 1 + p$;
 2 $S = $ randn$(N, s)$;
   // Compute $\tilde{C} = \tilde{A}S$
 3 ind = 1;
 4 for $j = 1$ to $n_2$ do
 5     for $i = 1$ to $m_2$ do
 6         $\tilde{C}_{(\text{ind})(:)} = \text{vec}(A_{ij})^T S$;
 7         ind = ind + 1;
 8     end
 9 end
10 $\tilde{U}\tilde{\Sigma}V = \tilde{C}$;
   // Compute $\tilde{V}^T = \tilde{\Sigma}^{-1}\tilde{U}^T\tilde{A}$
11 ind = 1;
12 for $j = 1$ to $n_2$ do
13     for $i = 1$ to $m_2$ do
14         $\tilde{\boldsymbol{v}}_{\text{ind}}^T = \tilde{\Sigma}^{-1}\tilde{U}^T[\text{vec}(A_{11})_{ind} \cdots \text{vec}(A_{m_2n_2})_{ind}]$;
15         ind = ind + 1;
16     end
17 end
   // Form $B$ and $D$
18 vec$(B) = \tilde{\sigma}_1\tilde{\boldsymbol{u}}_1$;
19 vec$(D) = \tilde{\boldsymbol{v}}_1$;
```

Algorithm 17: SKPD

Then, the SVD $\tilde{U}\tilde{\Sigma}V = \tilde{C}$ produces vec$(B) = \tilde{\sigma}_1\tilde{\boldsymbol{u}}_1$ with high probability. Note that the $V \in \mathbb{R}^{s\times s}$ is unneeded. Since $\tilde{\Sigma}_k^{-1}\tilde{U}_k^T\tilde{A} \approx \tilde{\Sigma}_k^{-1}\tilde{U}_k^T\tilde{U}\tilde{\Sigma}\tilde{V}^T = \tilde{V}_k^T$, set vec$(D)^T = \tilde{\Sigma}_1^{-1}\tilde{U}_1^T\tilde{A}$. This sketched KPD (SKPD) is summarized in Algirithm 17.

Note that the memory cost is optimized for $m_1 \approx m_2$ and $n_1 \approx n_2$. Since $\tilde{A} \in \mathbb{R}^{m_1n_1\times m_2n_2}$, the computational cost of the SVD is reduced by passing A^T to Algorithm 17 when $m_1n_1 > m_2n_2$.

6.3.2 Rank r SKPD

While using a sum of terms for a Kronecker rank r matrix constructed by $A = \sum_{i=1}^r B_i\otimes D_i$ was briefly introduced in Van Loan and Pitsianis [1993], it can be extended to the minimization problem for general matrix A. Notice that $\|A - B\otimes D\|_F$ is reduced further by considering additional terms of the SVD $\sum_{i=1}^r \tilde{\boldsymbol{u}}_i\tilde{\sigma}_i\tilde{\boldsymbol{v}}_i^T = \tilde{A}$.

Lemma 6.3.4. *Suppose $A \in \mathbb{R}^{m\times n}$ with Kronecker rank r defined by rearrangement $\tilde{A} \in \mathbb{R}^{m_1n_1\times m_2n_2}$ with $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, then taking vec$(B_i) = \tilde{\sigma}_i\tilde{\boldsymbol{u}}_i$ and vec$(D_i) = \tilde{\boldsymbol{v}}_i$ produces*

$$\left\|A - \sum_{i=1}^r B_i\otimes D_i\right\|_F = 0.$$

Proof. Since $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T = \sum_{i=1}^r \tilde{\boldsymbol{u}}_i\tilde{\sigma}_i\tilde{\boldsymbol{v}}_i^T$, then by Theorem 6.3.1,

$$\left\|A - \sum_{i=1}^r B_i\otimes D_i\right\|_F = \left\|\tilde{A} - \sum_{i=1}^r \text{vec}(B_i)\text{vec}(D_i)^T\right\|_F = \left\|\tilde{A} - \sum_{i=1}^r \tilde{\boldsymbol{u}}_i\tilde{\sigma}_i\tilde{\boldsymbol{v}}_i^T\right\|_F = 0$$

□

In addition, given an estimate of $\tilde{\Sigma}$, the minimum error for the KPD can still be obtained.

Lemma 6.3.5. *Given A with Kronecker rank r defined by rearrangement $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, target approximate rank $\tilde{r} \leq r$, and diagonal matrices $\Gamma \approx \tilde{\Sigma}$ with diagonal elements $\gamma_i > 0$ and Δ with diagonal elements δ_i such that $\Gamma\Delta = \tilde{\Sigma}$, then vec$(B_i) = \gamma_i\tilde{\boldsymbol{u}}_i$ and vec$(D_i)^T = \frac{1}{\gamma_i}\tilde{\boldsymbol{u}}_i^T\tilde{A}$ minimize $\|A - \sum_{i=1}^{\tilde{r}} B_i\otimes D_i\|_F$.*

Proof. Since $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$ and $\tilde{\Sigma} = \Gamma\Delta$, then $\Delta\tilde{V}^T = \Gamma^{-1}\tilde{U}^T\tilde{A}$. Now,

$$\left\| A - \sum_{i=1}^{\tilde{r}} B_i \otimes D_i \right\|_F^2 = \left\| \tilde{A} - \sum_{i=1}^{\tilde{r}} \tilde{\boldsymbol{u}}_i \gamma_i \delta_i \tilde{\boldsymbol{v}}_i^T \right\|_F^2 = \sum_{i=\tilde{r}}^{r} \tilde{\sigma}_i.$$

$\square$

Algorithm 17 can then be updated with a target approximate Kronecker rank, $\tilde{r}$. The updated rank r SKPD is given in Algorithm 18.

```
   Input: $A \in \mathbb{R}^{M\times N}$, $m_1$, $m_2$, $n_1$, $n_2$, $p$, $\tilde{r}$
1  $s = \tilde{r} + p$;
   // $\vdots$
   // Form $B_i$ and $D_i$
18 for $i = 1$ to $\tilde{r}$ do
19     vec$(B_i) = \tilde{\sigma}_i \tilde{\boldsymbol{u}}_i$;
20     vec$(D_i)^T = \tilde{\boldsymbol{v}}_i$;
21 end
```

Algorithm 18: rank $\tilde{r}$ SKPD

Now by Lemma 6.3.5, given a small misrepresentation of the singular values of $\tilde{A}$ due to sketching, then computing each D_i provides self-correction.

6.3.3 Block Kronecker Product Decomposition

Assuming the matrix A contains some block structure, the Kronecker rank is typically much smaller for each block than for A in whole. In addition, partitioning A and taking a KPD for each partition allows more accuracy.

Theorem 6.3.6. *Suppose $A \in \mathbb{R}^{m\times n}$ with partition $A = [A_1|A_2|\cdots|A_p]$ with $A_i \in \mathbb{R}^{m\times\nu_i}$, $B \in \mathbb{R}^{m_1\times n_1}$ and $D \in \mathbb{R}^{m_2\times n_2}$ with $m_1m_2 = m$ and $n_1n_2 = n$ such that $\|A - B\otimes D\|_F$ is minimized, and $E_i \in \mathbb{R}^{m_1\times q_i}$ with $\sum_i q_i = n_1$ and $G_i \in \mathbb{R}^{m_2\times n_2}$ such that $\|A_i - E_i\otimes G_i\|_F$ is minimized for $i \in [1, p]$.*

Then $\|A - [E_1\otimes G_1|E_2\otimes G_2|\cdots|E_p\otimes G_p]\|_F \leq \|A - B\otimes D\|_F$.

Proof. Partition B such that

$$B = [B_1|B_2|\cdots|B_p]$$

with $B_i \in \mathbb{R}^{m_1 \times q_i}$. Since each $\|A_i - E_i \otimes G_i\|_F$ is minimized and

$$\|A - [E_1 \otimes G_1|E_2 \otimes G_2|\cdots|E_p \otimes G_p]\|_F = \sum_{i=1}^{p} \|A_i - E_i \otimes G_i\|_F,$$

it is then immediate that

$$\sum_{i=1}^{p} \|A_i - E_i \otimes G_i\|_F \leq \sum_{i=1}^{p} \|A_i - B_i \otimes D\|_F = \|A - B \otimes D\|_F.$$

$\square$

A simple example with $m_1 = n_1 = 2$ and $q_1 = q_2 = 1$ allows

$$\left\| A - \begin{bmatrix} e_{1_1}G_1 & e_{2_1}G_2 \\ e_{1_2}G_1 & e_{2_2}G_2 \end{bmatrix} \right\|_F \leq \left\| A - \begin{bmatrix} b_{11}D & b_{12}D \\ b_{21}D & b_{22}D \end{bmatrix} \right\|_F .$$

After taking the KPD, methods that utilize only matrix vector multiplication, such as LSQR, can easily be modified to benefit from the reduced computational cost of Kronecker multiplication. Namely, with $\text{vec}(U_i) = \boldsymbol{u}_i \in \mathbb{R}^{\nu_i}$ for partition $\boldsymbol{u} = [\boldsymbol{u}_1^T|\cdots|\boldsymbol{u}_p^T]^T$ based on the partition of A, then $A\boldsymbol{u} = \sum_{i=1}^{p} A_i\boldsymbol{u}_i = \sum_{i=1}^{p} G_iU_iE_i^T$. Further, with $\text{vec}(V) = \boldsymbol{v}$, then $A^T\boldsymbol{v} = [(A_1^T\boldsymbol{v})^T|\cdots|(A_p^T\boldsymbol{v})^T]^T = [\text{vec}(G_1^TVE_1)^T|\cdots|\text{vec}(G_p^TVE_p)^T]^T$. Using a block KPD allows more potential accuracy, but requires more memory. When A is too large to store in memory, however, this allows a KPD to be formed one block of A at a time.

In general, when A can be represented by a small approximate Kronecker rank, Kronecker product multiplication has potential to significantly reduce computational requirements. Thus the SKPD is a valuable method that can be validated through future research.

6.4 RPS Applied to Groundwater Transmissivity

A brief discussion is now given on work from a summer internship at Los Alamos National Laboratory in 2017, which motivated the development of the RPS method.

An understanding of the subsurface transmissivity is desired in many applications such as estimating the dispersion pathways of water and hazardous materials, extracting valuable resources such as oil and gas, and estimating the effectiveness of geothermal reservoirs, Bárdossy and Hörning [2016], Lin et al. [2017]. Trends in groundwater modeling have lead to increasing numbers of unknown model parameters and measurement data, Kourakos and Mantoglou [2012], Lin et al. [2016], Vermeulen et al. [2005].

Many methods have been developed to alleviate the increasing computational burdens of determining the unknown parameters for these increasingly large problems, Asher et al. [2015], Bárdossy and Hörning [2016], Kitanidis [1997], Lee and Kitanidis [2014], Lin et al. [2017], Zimmerman et al. [1998]. The associated parameter dependent nonlinear transient flow equation can be solved by applying a linearization and then solving the linear system, Kitanidis [1997]. A large portion of the computational cost can then be attributed to solving linear systems of equations.

6.4.1 Principal Component Geostatistical Approach (PCGA)

The subsurface transmissivity can be modeled by the transient groundwater flow equation $\mathbf{h} = f(T) + \epsilon$, giving the hydraulic head, $\mathbf{h}$, in terms of the transmissivity, T, and unknown error, ϵ. Suppose the vectorized transmissivity is given by $\mathbf{m} \sim N(X\boldsymbol{\beta}, Q)$ and that $\mathbf{h} \sim N(\hat{\mathbf{h}}, R)$, and that the standard definition $\|\mathbf{d}\|_D^2 = \mathbf{d}^T D \mathbf{d}$ is used. Then $[\mathbf{m}^T, \boldsymbol{\beta}^T]^T$ can be found as the solution of the non-linear ill-posed problem

$$\begin{bmatrix} \mathbf{m} \\ \boldsymbol{\beta} \end{bmatrix} = \arg\min_{\mathbf{m},\boldsymbol{\beta}} \left\{ \frac{1}{2}\|\mathbf{h} - f(\mathbf{m})\|_{R^{-1}}^2 + \frac{1}{2}\|\mathbf{m} - X\boldsymbol{\beta}\|_{Q^{-1}}^2 \right\} = \arg\min_{\mathbf{m},\boldsymbol{\beta}} \left\{ J(\mathbf{m},\boldsymbol{\beta}) \right\}.$$

Solving for $\nabla J = 0$, using $\nabla_{\mathbf{m}} J = H^T R^{-1} f(\mathbf{m}) - H^T R^{-1}\mathbf{h} + Q^{-1}\mathbf{m} - Q^{-1}X\boldsymbol{\beta}$ and $\nabla_{\boldsymbol{\beta}} J = X^T Q^{-1} X\boldsymbol{\beta} - XQ^{-1}\mathbf{m}$, where H is the Jacobian of f, gives

$$Q^{-1}\mathbf{m} = Q^{-1}X\boldsymbol{\beta} + H^T R^{-1}(\mathbf{h} - f(\mathbf{m})), \text{ and } X^T Q^{-1}\mathbf{m} = X^T Q^{-1} X\boldsymbol{\beta}, \tag{6.2}$$

and

$$X^T H^T R^{-1}(\mathbf{h} - f(\mathbf{m})) = \mathbf{0}. \tag{6.3}$$

But now, following Kitanidis [1997], Lee and Kitanidis [2014], Lin et al. [2017] this can be linearized about the current update for $\boldsymbol{m}^{(i)}$, giving $f(\mathbf{m}^{(i+1)}) \approx f(\mathbf{m}^{(i)}) + H(\mathbf{m}^{(i+1)} - \mathbf{m}^{(i)})$. This can be used to replace $(\mathbf{h} - f(\mathbf{m}))$ in (6.3) by $(\mathbf{h} - f(\mathbf{m}^{(i)}) - H(\mathbf{m}^{(i+1)} - \mathbf{m}^{(i)}))$. Together with introducing $\boldsymbol{\xi} = R^{-1}(\mathbf{h} - f(\mathbf{m}^{(i)}) - H(\mathbf{m}^{(i+1)} - \mathbf{m}^{(i)}))$ this replaces (6.3) by

$$(HX)^T\boldsymbol{\xi} = \mathbf{0}. \tag{6.4}$$

Further, using $\boldsymbol{\xi}$ in (6.2) directly, gives $Q^{-1}\boldsymbol{m} = Q^{-1}X\boldsymbol{\beta} + H^T\boldsymbol{\xi}$ or the update

$$\boldsymbol{m}^{(i+1)} = X\boldsymbol{\beta} + QH^T\boldsymbol{\xi}. \tag{6.5}$$

Multiplying (6.5) through by H along with (6.4) provides the linear block system comprising the PCGA method,

$$\begin{bmatrix} HQH^T + R & HX \\ (HX)^T & 0 \end{bmatrix} \begin{bmatrix} \xi \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} h - f(m^{(i)}) + Hm^{(i)} \\ 0 \end{bmatrix}. \tag{6.6}$$

In terms of $A\boldsymbol{x} = \boldsymbol{b}$, (6.6) can be broken down as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\beta} \end{bmatrix} \quad A = \begin{bmatrix} HQH^T + R & HX \\ (HX)^T & 0 \end{bmatrix} \quad \boldsymbol{b} = \begin{bmatrix} h - f(m^{(i)}) + Hm^{(i)} \\ 0 \end{bmatrix}.$$

6.4.2 Randomized Geostatistical Approach (RGA)

In obtaining the PCGA solution of (6.6), Lee and Kitanidis [2014], the computational time is reduced by utilizing a finite difference approximation in place of matrix H, and the rank-k approximation $Q = \sum_{i=1}^{k} \zeta_i \zeta_i^T$. The RGA approach, Lin et al. [2017], yields

$$\begin{bmatrix} \bar{S}^T HQH^T \bar{S} + \bar{S}^T R \bar{S} & \bar{S}^T HX \\ (\bar{S}^T HX)^T & 0 \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{\xi}} \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} \bar{S}^T (\mathbf{h} - f(\mathbf{m}^{(i)}) + H\mathbf{m}^{(i)}) \\ 0 \end{bmatrix},$$

which corresponds to applying a left block sketch

$$\tilde{S} = \begin{bmatrix} \bar{S} & 0 \\ 0 & I \end{bmatrix}$$

to (6.6) and the right preconditioning $\boldsymbol{\xi} = \bar{S}\tilde{\boldsymbol{\xi}}$, where now the update is given by $\boldsymbol{m}^{(i+1)} = X\boldsymbol{\beta} + QH^T \bar{S} \tilde{\boldsymbol{\xi}}$. Although it was shown in Lin et al. [2017] that RGA produces comparable results to PCGA with reduced computational time this formulation is not immediately amenable to the analysis leading to Theorem 4.3.2.

6.4.3 Application of RPS

In contrast, if RPS is implemented directly on the full block system (6.6), the modified block system

$$S^T \begin{bmatrix} HQH^T + R & HX \\ (HX)^T & 0 \end{bmatrix} N\mathbf{y} \approx S^T \begin{bmatrix} \mathbf{h} - f(\mathbf{m}^{(i)}) + H\mathbf{m}^{(i)} \\ 0 \end{bmatrix}$$

is obtained, which now fits the formulation of Theorem 4.3.2.

6.4.4 Numerical Results

As the number of points in the discretized domain increases, the size of the problem increases quadratically, relying on a $N = 2N_x N_y$ length vector, where N_x and N_y represent the number of points used in the x and y domains respectively. A sketch size $s = 2k = 256$ is used for grids ranging from $N_x = 50$ and $N_y = 51$ to $N_x = 200$ and $N_y = 201$, producing linear problems ranging from 5100×5100 to 80400×80400. 4 hydraulic heads

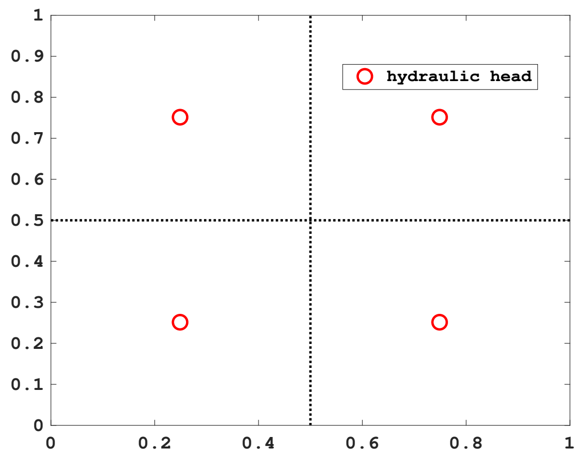


Figure 6.3: Demonstration of spacing for a 4×4 grid of hydraulic heads. x and y are then discretized over $[0, 1]$ using N_x and N_y points respectively.

are simulated on a 2×2 grid for each problem size, see Figure 6.3. For direct comparison, each simulation is performed with a maximum of 5 outer iterations. The resulting root mean squared errors of both the RGA method and RPS are given in Figure 6.4. RPS consistently produces results with less error while maintaining similar computational time on a single core processor.

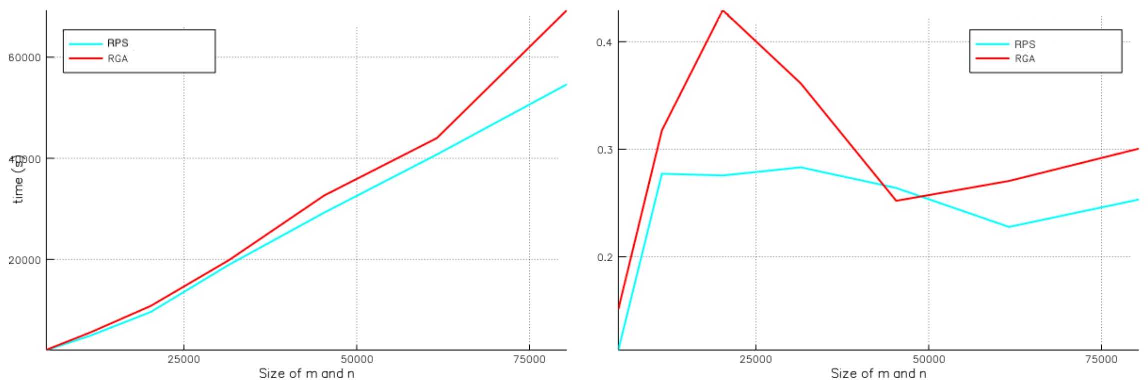


Figure 6.4: Comparison of computational time and root mean squared errors obtained by RPS versus RGA

The horizontal transmissivity is shown on a log scale with a 100×101 grid for both methods in Figure 6.5, giving a visual representation of the results after 5 outer iterations are performed. This provides visual confirmation of the results produced by RPS.

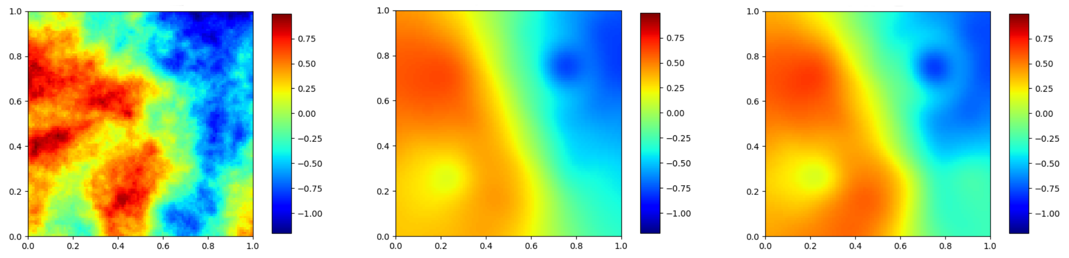

(a) True (b) RGA (c) RPS

Figure 6.5: True log-transmissivity field Figure 6.5a, and the log-transmissivity field resulting from `RGA` Figure 6.5b and `RPS` Figure 6.5c.

6.4.5 Groundwater Transmissivity Conclusions

RPS shows great versatility due to the simplicity and performance gains from preconditioning mixed with the computational efficacy gains from sketching. This method is applied to the framework established by the PCGA method directly and uses properties of the pseudo-inverse to add preconditioning in comparison to the dimensionality reduction of the RGA method.

Through theoretical and computational cost analysis, it is shown that the efficiency of solving the linearized portion of the transient flow equation can be significantly increased by applying these multi-dimensionality reduction techniques, resulting in a well-conditioned system.The improved conditioning achieved by the RPS method helps the solution gain increased accuracy at each iteration.Thus an invaluable method for solving large linear systems by inversion has been provided.

In summary, this method improves accuracy while satisfying the need for computational efficiency caused by the trend to increase the number of both model parameters and data. This contribution consists of extending the dimensionality reduction by a simple, novel construction of a preconditioning matrix based on properties of the pseudo-inverse, and thereby increasing the accuracy of the solution. Due to the simplicity of the approach, this method is by no means limited to hydrogeological inverse problems, but can easily be implemented in a variety of applications across multiple fields of interest.

6.5 Conclusions

An approximate pseudo-inverse method was iteratively applied with significantly reduced computational time in comparison to directly calculating an approximate pseudo-inverse. It was also showed that using a linear interpolation matrix as a right sketch provides a stable method of dimensionality reduction. Although using a linear interpolation matrix produces undesirable characteristics in the projection, it shows an alternative direction for dimensionality reduction that could be extended to higher order interpolation.

In addition, randomization was implemented in a novel way to obtain a series of $\tilde{r}$ approximate Kronecker product decompositions based on an approximate Kronecker rank $\tilde{r}$. If the system matrix can be approximated well for small $\tilde{r}$, this method has potential to significantly reduce computational requirements, but has yet to be validated. Thus, the SKPD provides a promising topic for further study.

6.6 Conclusions and Future Work

As the trend to collect massive amounts of data leads to more computationally expensive regularization methods for inverse problems, solving these large problems becomes difficult for technology to keep up with, Lin et al. [2016], Wang [2015]. To combat this, techniques for solving large-scale inverse problems were presented based on dimensionality reduction.

In Chapter 3, special case problems were introduced that benefit exceptionally from dimensionality reduction. Namely, a block-Toeplitz with Toeplitz blocks (BTTB) matrix was shown to be extendable to a block circulant with circulant blocks (BCCB) matrix, for which matrix multiplication can be performed via a convolution transformation using the

2DFFT. This analysis was extended for the gravity and magnetic problems, and included extensions for domains with model contributions from outside the discrete domain. Fast multiplication using the 2DFFT was also extended for use with BTTB matrices for the case where domain padding is desired.

Chapter 4 shows how applying randomization directly to a matrix system can further reduce computational requirements by combining a random left sketch and a right preconditioning substitution, denoted the randomized preconditioning sketch (RPS). Numerical results validated the stability of the solution when applied to inversion of gravity and magnetic data in combination with fast multiplication via the 2DFFT, and showed a significant reduction in computational time in comparison to the LSRN algorithm.

The RSVD approach for obtaining an approximate SVD was discussed in Chapter 5 for inversion of gravity and magnetic data. Computational comparisons for both real and synthetic data with and without domain padding were also provided.

Chapter 6 provided alternate applications for the techniques developed, along with other directions for dimensionality reduction of large-scale problems. These included alternate applications of randomization such as applying an approximate pseudo-inverse and obtaining an approximate Kronecker product decomposition. Section 6.4 then presented a practical application of the RPS method to a transient groundwater transmissivity problem where it was shown that RPS improves on the accuracy of the solution compared to the randomized geostatistical approach (RGA), which only implements a random left sketch.

In short, extensions for novel formulations of randomization techniques combined with a preconditioning substitution, useful for general inverse problems, have been discussed.

There are many directions for further research in dimensionality reduction. First, the RPS method can be applied in combination with the fast 2DFFT based multiplication for the joint inversion of gravity and magnetic data. An extension to the inversion of gravity and magnetic data by implementing joint inversion has been considered.

The system matrix is structured into a block system containing both the `gravity` and `magnetic` matrices, along with a coupling term. Since this produces a much larger system than considering either method alone, dimensionality reduction shows great potential in providing a viable solution via joint inversion. A paper, Vatankhah et al. [2020b], has been submitted for the implementation of joint inversion using BTTB structure to provide fast matrix multiplication via the `2DFFT`, but without consideration for randomization techniques. Thus, joint inversion of `gravity` and `magnetic` data combining fast `2DFFT` based multiplication with randomization techniques is a topic for future work.

Specifically, joint inversion is implemented by alternating between `magnetic` and `gravity` models for $\mathbf{d}_{\text{obs}}$, G and $\mathbf{m}$, at each step minimizing

$$\|\mathbf{W}_{\mathbf{d}}(\mathbf{d}_{\text{obs}} - G\mathbf{m})\|_2^2 + \alpha^2 \|WD(\mathbf{m} - \mathbf{m}_{\text{apr}})\|_2^2 + \beta^2 \ \|\boldsymbol{t}(\mathbf{m})\|_2^2,$$

with regularization parameters α and β, and the cross-gradient function $\boldsymbol{t}(\mathbf{m}) \in \mathcal{R}^{3n}$ is the link between the `gravity` and `magnetic` models. Given results demonstrated in Sections 4.5.5 and 4.5.9, the `RPS` algorithm provides a significant reduction in computational time when solving either the `gravity` or `magnetic` model. Further, `RPS` has been shown to provide regularization through truncation, effectively removing the reliance on α. Thus, `RPS` provides a viable candidate for simplifying and efficiently solving this joint method.

A basis for applying randomization to obtain an approximate Kronecker product decomposition has been considered. The SKPD was also extended to produce a sum of Kronecker products based on the Kronecker rank. This allows matrix multiplication to be carried out at reduced computational cost for each Kronecker product, but the reduction can become outweighed by repeating the matrix multiplication repeatedly when the Kronecker rank is large. Thus, the SKPD is most efficacious and has yet to be validated for an application where the system matrix has a small Kronecker rank.

REFERENCES

M. J. Asher, B. F. W. Croke, A. J. Jakeman, and L. J. M. Peeters. A review of surrogate models and their application to groundwater modeling. *Water Resources Research*, 51 (8):5957–5973, 2015. ISSN 1944-7973. doi: 10.1002/2015WR016967. URL http://dx.doi.org/10.1002/2015WR016967.

András Bárdossy and Sebastian Hörning. Gaussian and non-Gaussian inverse modeling of groundwater flow using copulas and random mixing. *Water Resources Research*, 52 (6):4504–4526, 2016. ISSN 1944-7973. doi: 10.1002/2014WR016820. URL http://dx.doi.org/10.1002/2014WR016820.

Olivier Boulanger and Michel Chouteau. Constraints in 3D gravity inversion. *Geophysical Prospecting*, 49(2):265–280, 2001. ISSN 1365-2478. doi: 10.1046/j.1365-2478.2001.00254.x. URL http://dx.doi.org/10.1046/j.1365-2478.2001.00254.x.

Christian Eske Bruun and Trine Brandt Nielsen. Algorithms and software for large-scale geophysical reconstructions. Master's thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark, 2007.

Raymond Hon-Fu Chan and Xiao-Qing Jin. *An Introduction to Iterative Toeplitz Solvers*. Society for Industrial and Applied Mathematics, 2007. doi: 10.1137/1.9780898718850. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718850.

Longwei Chen and Lanbo Liu. Fast and accurate forward modelling of gravity field using prismatic grids. *Geophysical Journal International*, 216(2):1062–1071, 11 2018. ISSN 0956-540X. doi: 10.1093/gji/ggy480. URL https://doi.org/10.1093/gji/ggy480.

Philip J. Davis. *Circulant Matrices*. Monographs and textbooks in pure and applied mathematics. Wiley, 1979. ISBN 9780471057710.

Petros Drineas, Malik Magdon-Ismail, Michael W. Mahoney, and David P. Woodruff. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13(111):3475–3506, 2012. URL http://jmlr.org/papers/v13/drineas12a.html.

Colin G Farquharson and Douglas W Oldenburg. A comparison of automatic techniques for estimating the regularization parameter in non-linear inverse problems. *Geophysical Journal International*, 156(3):411–425, 2004. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-246X.2004.02190.x.

Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. JHU Press, fourth edition, 2013. ISBN 1421407949 9781421407944. URL http://www.cs.cornell.edu/cv/GVL4/golubandvanloan.htm.

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

Robert M. Gower and Peter Richtárik. Linearly convergent randomized iterative methods for computing the pseudoinverse. 28 pages, 10 figures, January 2017. URL `https://hal.inria.fr/hal-01430489`.

István Béla Haáz. Relations between the potential of the attraction of the mass contained in a finite rectangular prism and its first and second derivatives. *Geophysical Transactions II*, 7:57–66, 1953.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

Per Christian Hansen. The truncated SVD as a method for regularization. *BIT*, 27(4): 534–553, October 1987. ISSN 0006-3835. doi: 10.1007/BF01937276. URL `http://dx.doi.org/10.1007/BF01937276`.

Per Christian Hansen. Regularization tools a Matlab package for analysis and solution of discrete ill-posed problems. *Numerical Algorithms*, 1994.

Per Christian Hansen. *Discrete Inverse Problems*. Society for Industrial and Applied Mathematics, 2010. doi: 10.1137/1.9780898718836. URL `https://epubs.siam.org/doi/abs/10.1137/1.9780898718836`.

Per Christian Hansen, Víctor Pereyra, and Godela Scherer. *Least squares data fitting with applications*. JHU Press, 2013.

Jarom D Hogue and Rosemary A Renaut. A randomized preconditioning sketch for large ill-posed inverse problems. in preparation, 2020.

Jarom D Hogue, Rosemary A Renaut, and Saeed Vatankhah. A tutorial and open source software for the efficient evaluation of gravity and magnetic kernels, 2019. URL `https://arxiv.org/abs/1912.06976`. submitted.

Peter K. Kitanidis. The minimum structure solution to the inverse problem. *Water Resources Research*, 33(10):2263–2272, 1997. ISSN 1944-7973. doi: 10.1029/97WR01619. URL `http://dx.doi.org/10.1029/97WR01619`.

George Kourakos and Aristotelis Mantoglou. Inverse groundwater modeling with emphasis on model parameterization. *Water Resources Research*, 48(5), 2012. ISSN 1944-7973. doi: 10.1029/2011WR011068. URL `http://dx.doi.org/10.1029/2011WR011068`. W05540.

J. Lee and P. K. Kitanidis. Large-scale hydraulic tomography and joint inversion of head and tracer data using the principal component geostatistical approach (PCGA). *Water Resources Research*, 50(7):5410–5427, 2014. ISSN 1944-7973. doi: 10.1002/2014WR015483. URL `http://dx.doi.org/10.1002/2014WR015483`.

Kun Li, Long-Wei Chen, Qing-Rui Chen, Shi-Kun Dai, Qian-Jiang Zhang, Dong-Dong Zhao, and Jia-Xuan Ling. Fast 3D forward modeling of the magnetic field and gradient tensor on an undulated surface. *Applied Geophysics*, 15(3):500–512, Sep 2018.

ISSN 1993-0658. doi: 10.1007/s11770-018-0690-9. URL https://doi.org/10.1007/s11770-018-0690-9.

Yaoguo Li and Douglas W. Oldenburg. 3-D inversion of magnetic data. *Geophysics*, 61 (2):394–408, 1996. doi: 10.1190/1.1443968. URL https://doi.org/10.1190/1.1443968.

Yaoguo Li and Douglas W. Oldenburg. 3-D inversion of gravity data. *Geophysics*, 63(1): 109–119, 1998. doi: 10.1190/1.1444302. URL https://doi.org/10.1190/1.1444302.

Youzuo Lin, Daniel O'Malley, and Velimir V. Vesselinov. A computationally efficient parallel Levenberg-Marquardt algorithm for highly parameterized inverse model analyses. *Water Resources Research*, 52(9):6948–6977, 2016. ISSN 1944-7973. doi: 10.1002/2016WR019028. URL http://dx.doi.org/10.1002/2016WR019028.

Youzuo Lin, Ellen B. Le, Daniel O'Malley, Velimir V. Vesselinov, and Tan Bui-Thanh. Large-scale inverse model analyses employing fast randomized data reduction. *Water Resources Research*, 53(8):6784–6801, 2017. ISSN 1944-7973. doi: 10.1002/2016WR020299. URL http://dx.doi.org/10.1002/2016WR020299.

Nick Luiken and Tristan van Leeuwen. Comparing RSVD and Krylov methods for linear inverse problems. *Computers & Geosciences*, 137:104427, 2020.

Michael W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011. ISSN 1935-8237. doi: 10.1561/2200000035. URL http://dx.doi.org/10.1561/2200000035.

Xiangrui Meng, Michael A. Saunders, and Michael W. Mahoney. LSRN: A parallel iterative solver for strongly over- or underdetermined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, New York, NY, USA, 1995. ISBN 0-521-47465-5, 9780521474658.

Christopher C. Paige and Michael A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM transactions on Mathematical Software*, 8(1): 43–71, 1982.

Mark Pilkington. 3-D magnetic imaging using conjugate gradients. *Geophysics*, 62(4): 1132–1142, 08 1997. ISSN 0016-8033. doi: 10.1190/1.1444214. URL https://doi.org/10.1190/1.1444214.

Mark Pilkington. 3D magnetic data-space inversion with sparseness constraints. *Geophysics*, 74(1):L7–L15, 2009. doi: 10.1190/1.3026538. URL https://doi.org/10.1190/1.3026538.

D. Bhaskara Rao and N. Ramesh Babu. A rapid method for three-dimensional modeling of magnetic anomalies. *Geophysics*, 56(11):1729–1737, November 1991.

Rosemary A Renaut, Saeed Vatankhah, and Vahid E Ardestani. Hybrid and iteratively reweighted regularization by unbiased predictive risk and weighted GCV for projected systems. *SIAM Journal on Scientific Computing*, 39(2):B221–B243, 2017.

Rosemary A Renaut, Jarom D Hogue, and Saeed Vatankhah. A fast methodology for large-scale focusing inversion of gravity and magnetic data using the structured model matrix and the 2D fast Fourier transform, 2020. URL `https://arxiv.org/abs/2004.13904`. submitted.

Paul Rodriguez and Brendt Wohlberg. An iteratively reweighted norm algorithm for total variation regularization. In *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, pages 892–896. IEEE, 2006.

Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.

Kathrin Schäcke. On the Kronecker product. 2013.

Charles F Van Loan and Nikos Pitsianis. Approximation with Kronecker products. In *Linear algebra for large scale and real-time applications*, pages 293–314. Springer, 1993.

Saeed Vatankhah, Vahid E Ardestani, and Rosemary A Renaut. Application of the χ^2 principle and unbiased predictive risk estimator for determining the regularization parameter in 3-D focusing gravity inversion. *Geophysical Journal International*, 200(1):265–277, Nov 2014a. ISSN 0956-540X. doi: 10.1093/gji/ggu397. URL `http://dx.doi.org/10.1093/gji/ggu397`.

Saeed Vatankhah, Vahid E Ardestani, and Rosemary A Renaut. Automatic estimation of the regularization parameter in 2D focusing gravity inversion: application of the method to the Safo manganese mine in the northwest of Iran. *Journal of Geophysics and Engineering*, 11(4):045001, 2014b.

Saeed Vatankhah, Rosemary A Renaut, and Vahid E Ardestani. 3-D Projected L1 inversion of gravity data using truncated unbiased predictive risk estimator for regularization parameter estimation. *Geophysical Journal International*, 210(3):1872–1887, 06 2017. ISSN 0956-540X. doi: 10.1093/gji/ggx274. URL `https://doi.org/10.1093/gji/ggx274`.

Saeed Vatankhah, Rosemary A Renaut, and Vahid E Ardestani. A fast algorithm for regularized focused 3D inversion of gravity data using randomized singular-value decomposition. *GEOPHYSICS*, 83(4):G25–G34, 2018a. doi: 10.1190/geo2017-0386.1. URL `https://doi.org/10.1190/geo2017-0386.1`.

Saeed Vatankhah, Rosemary A. Renaut, and Vahid E. Ardestani. Total variation regularization of the 3-D gravity inverse problem using a randomized generalized singular value decomposition. *Geophysical Journal International*, 213(1):695–705, 2018b. doi: 10.1093/gji/ggy014. URL `http://dx.doi.org/10.1093/gji/ggy014`.

Saeed Vatankhah, Shuang Liu, Rosemary A Renaut, Xiangyun Hu, and Jamaledin Baniamerian. Improving the use of the randomized singular value decomposition for the inversion of gravity and magnetic data, 2020a. URL `https://doi.org/10.1190/geo2019-0603.1`. recently accepted.

Saeed Vatankhah, Shuang Liu, Rosemary A Renaut, Xiangyun Hu, Jarom D Hogue, and Mostafa Gharloghi. An efficient alternating algorithm for the L_p-norm cross-gradient joint inversion of gravity and magnetic data using the 2D fast Fourier transform, 2020b. URL `https://arxiv.org/abs/2001.03579`. original version on arXiv.

Saeed Vatankhah, Rosemary A Renaut, and Shuang Liu. Research note: A unifying framework for the widely used stabilization of potential field inverse problems. *Geophysical Prospecting*, 68(4):1416–1421, May 2020c. ISSN 0016-8025. doi: 10.1111/1365-2478.12926. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/1365-2478.12926`.

P. T. M. Vermeulen, A. W. Heemink, and J. R. Valstar. Inverse modeling of groundwater flow using model reduction. *Water Resources Research*, 41(6):n/a–n/a, 2005. ISSN 1944-7973. doi: 10.1029/2004WR003698. URL `http://dx.doi.org/10.1029/2004WR003698`. W06003.

C. Vogel. *Computational Methods for Inverse Problems*. Society for Industrial and Applied Mathematics, 2002. doi: 10.1137/1.9780898717570. URL `https://epubs.siam.org/doi/abs/10.1137/1.9780898717570`.

Sergey Voronin, Dylan Mikesell, and Guust Nolet. Compression approaches for the regularized solutions of linear systems from large-scale inverse problems. *GEM - International Journal on Geomathematics*, 6(2):251–294, Nov 2015. ISSN 1869-2680. doi: 10.1007/s13137-015-0073-9. URL `http://dx.doi.org/10.1007/s13137-015-0073-9`.

Shusen Wang. A practical guide to randomized matrix computations with MATLAB implementations. *CoRR*, abs/1505.07570, 2015. URL `http://arxiv.org/abs/1505.07570`.

Brendt Wohlberg and Paul Rodríguez. An iteratively reweighted norm algorithm for minimization of total variation functionals. *Signal Processing Letters, IEEE*, 14(12):948–951, 2007.

David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(12):1–157, 2014. ISSN 1551-305X. doi: 10.1561/0400000060. URL `http://dx.doi.org/10.1561/0400000060`.

Hua Xiang and Jun Zou. Regularization with randomized SVD for large-scale discrete inverse problems. *Inverse Problems*, 29(8):085008, 2013. URL `http://stacks.iop.org/0266-5611/29/i=8/a=085008`.

Yile Zhang and Yau Shu Wong. BTTB-based numerical schemes for three-dimensional gravity field inversion. *Geophysical Journal International*, 203(1):243–256, 2015.

Guangdong Zhao, Bo Chen, Longwei Chen, Jianxin Liu, and Zhengyong Ren. High-accuracy 3D Fourier forward modeling of gravity field based on the Gauss-FFT technique. *Journal of Applied Geophysics*, 150:294 – 303, 2018. ISSN 0926-9851. doi: https://doi.org/10.1016/j.jappgeo.2018.01.002. URL `http://www.sciencedirect.com/science/article/pii/S0926985117301751`.

Michael S. Zhdanov. *Geophysical Inverse Theory and Regularization Problems*, volume 36. Elsevier, Amsterdam, 2002.

D. A. Zimmerman, G. de Marsily, C. A. Gotway, M. G. Marietta, C. L. Axness, R. L. Beauheim, R. L. Bras, J. Carrera, G. Dagan, P. B. Davies, D. P. Gallegos, A. Galli, J. Gómez-Hernández, P. Grindrod, A. L. Gutjahr, P. K. Kitanidis, A. M. Lavenue, D. McLaughlin, S. P. Neuman, B. S. RamaRao, C. Ravenne, and Y. Rubin. A comparison of seven geostatistically based inverse approaches to estimate transmissivities for modeling advective transport by groundwater flow. *Water Resources Research*, 34 (6):1373–1413, 1998. ISSN 1944-7973. doi: 10.1029/98WR00003. URL `http://dx.doi.org/10.1029/98WR00003`.

APPENDIX A

PSEUDO-INVERSE : KEY RESULTS

For arbitrary matrix $A \in \mathbb{R}^{m \times n}$, the pseudo-inverse $A^\dagger$ of A is the unique matrix that satisfies the Moore-Penrose conditions, [Golub and Van Loan, 2013, Section 5.5.4]

$$(i)\ AA^\dagger A = A,\ (ii)\ A^\dagger AA^\dagger = A^\dagger,\ (iii)\ (AA^\dagger)^T = AA^\dagger,\ (iv)\ (A^\dagger A)^T = A^\dagger A. \quad \text{(A.1)}$$

When A is of full rank and square $A^\dagger = A^{-1}$. Note, in general $AA^\dagger \neq A^\dagger A \neq I$. Given the SVD of A, $A = U\Sigma V^T$, and A of rank k, $k \leq \textbf{min}(m, n)$,

$$(i)\ A^\dagger = V\Sigma^\dagger U^T,\ (ii)\ A_k = U_k\Sigma_k V_k^T,\ (iii)\ A_k^\dagger = V_k\Sigma_k^\dagger U_k^T. \quad \text{(A.2)}$$

Lemma A.0.1. *Suppose $A = U\Sigma V^T$ and $AA^T = U_{AA^T}\Sigma_{AA^T}V_{AA^T}^T$ then $(V_{AA^T}) = (U_{AA^T}) = U$ and $(\Sigma_{AA^T}) = \Sigma^2$.*

Proof. It is immediate that $AA^T = U\Sigma^2U^T$ and by the uniqueness of the SVD the result follows. □

Lemma A.0.2. *Assume that matrix A has at least rank k then $A^T(A_k^\dagger)^TA_k^\dagger = A_k^\dagger$*

Proof. It is immediate using the truncated SVD of A that

$$\begin{aligned} A^T(A_k^\dagger)^TA_k^\dagger &= V\Sigma U^TU_k\Sigma_k^{-2}U_k^T = V\Sigma \begin{bmatrix} I_k \\ 0 \end{bmatrix} \Sigma_k^{-2}U_k^T \\ &= V \begin{bmatrix} \Sigma_k \\ 0 \end{bmatrix} \Sigma_k^{-2}U_k^T = V \begin{bmatrix} \Sigma_k^{-1} \\ 0 \end{bmatrix} U_k^T = V_k\Sigma_k^{-1}U_k^T = A_k^\dagger. \end{aligned}$$

□

APPENDIX B

RELEVANT MATHEMATICAL RESULTS

B.1 Conditioning of Least Squares Problem

Theorem B.1.1 shows the dependance of the perturbed least squares solution on the conditioning number.

Theorem B.1.1. *[Golub and Van Loan, 2013, Theorem 5.3.1] Suppose $\boldsymbol{x}_{LS}$, $\boldsymbol{r}_{LS}$, $\hat{\boldsymbol{x}}_{LS}$, and $\hat{\boldsymbol{r}}_{LS}$ satisfy*

$$\begin{aligned} \|A\boldsymbol{x}_{LS} - \boldsymbol{b}\|_2 &= \min & \boldsymbol{r}_{LS} &= \boldsymbol{b} - A\boldsymbol{x}_{LS} \\ \|(A+\delta A)\hat{\boldsymbol{x}}_{LS} - (\boldsymbol{b}+\boldsymbol{\delta b})\|_2 &= \min & \hat{\boldsymbol{r}}_{LS} &= (\boldsymbol{b}+\boldsymbol{\delta b}) - (A+\delta A)\hat{\boldsymbol{x}}_{LS}, \end{aligned}$$

where A has rank n and $\|\delta A\|_2 < \sigma_n(A)$. Assume that $\boldsymbol{b}$, $\boldsymbol{r}_{LS}$ and $\boldsymbol{x}_{LS}$ are not zero. Let θ_{LS} be defined by

$$\sin(\theta_{LS}) = \frac{\boldsymbol{r}_{LS}}{\|\boldsymbol{b}\|_2}.$$

If

$$\epsilon = \boldsymbol{max}\left\{\frac{\|\delta A\|_2}{\|A\|_2}, \frac{\|\boldsymbol{\delta b}\|_2}{\|\boldsymbol{b}\|_2}\right\}$$

and

$$\nu_{LS} = \frac{A\boldsymbol{x}_{LS}}{\sigma_n(A)\|\boldsymbol{x}_{LS}\|_2},$$

then

$$\frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \le \epsilon\left\{\frac{\nu_{LS}}{\cos(\theta_{LS})} + [1 + \nu_{LS}\tan(\theta_{LS})\kappa_2(A)\right\} + \mathcal{O}(\epsilon^2)$$

and

$$\frac{\|\hat{\boldsymbol{r}} - \boldsymbol{r}\|_2}{\|\boldsymbol{b}\|_2} \le \epsilon\left\{\frac{1}{\sin(\theta_{LS})} + \left[\frac{1}{\nu_{LS}\tan(\theta_{LS})} + 1\right]\kappa_2(A)\right\} + \mathcal{O}(\epsilon^2).$$

B.2 Rank k SVD of a Projected Matrix

Next, Theorem B.2.1 and Corollary B.2.2 show an error bound for the truncated SVD under projection.

Theorem B.2.1. *[Halko et al., 2011, Theorem 9.3] Let A be an $m \times n$ matrix with singular values $\sigma_1 \ge \sigma_2 \ge \sigma_3 \ge \dots$, and let Z be an $m \times l$ matrix, where $l \ge k$. Suppose that $\hat{A}_{(k)}$ is the best rank k approximation of $P_Z A$ with respect to the spectral norm. Then*

$$\|A - \hat{A}_{(k)}\|_2 \le \sigma_{k+1} + \|(\mathrm{I} - P_Z)A\|_2.$$

Corollary B.2.2. *[Halko et al., 2011, Corollary 10.9] Suppose that A is a real $m \times n$ matrix with singular values $\sigma_1 \ge \sigma_2 \ge \sigma_3 \ge \dots$. Choose a target rank $k \ge 2$ and an oversampling parameter $p \ge 4$, where $k + p \le$ **min**(m, n). Draw an $n \times (k+p)$ standard Gaussian matrix Ω, and construct the sample matrix $Y = A\Omega$. Then*

$$\|(\mathrm{I} - P_Y)A\|_2 \le \left(1 + 17\sqrt{1 + k/p}\right)\sigma_{k+1} + \frac{8\sqrt{k+p}}{p+1}\left(\sum_{j>k}\sigma_j^2\right)^{1/2},$$

with failure probability at most $6e^{-p}$*. Moreover,*

$$\|(\mathrm{I} - P_Y)A\|_2 \leq \left(1 + 8\sqrt{(k+p)p\log p}\right)\sigma_{k+1} + 3\sqrt{k+p}\left(\sum_{j>k}\sigma_j^2\right)^{1/2},$$

with failure probability at most $6p^{-p}$*.*