The Thesis committee for Tharit Tangkijwanichakul
Certifies that this is the approved version of the following thesis:

Chain of Operators for Inverse Hessian Estimation in Least-squares Migration

APPROVED BY

SUPERVISING COMMITTEE:

Dr. Sergey Fomel, Supervisor

Dr. Stephen Grand

Dr. Kyle Spikes

Chain of Operators for Inverse Hessian Estimation in Least-squares Migration

by

Tharit Tangkijwanichakul

THESIS

Presented to the Faculty of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

BACHELOR OF SCIENCE WITH HONORS

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2021

Acknowledgments

For my past 4 years at the University of Texas at Austin as an undergraduate student, I have received many supports from people I encountered. First, I would like to thank Dr. Sergey Fomel for his role as both academic advisor and examplary scientist. I benevolently have been influenced by him in many ways. I am thankful for his style of not hand-holding his students. His guidance is always useful but always leaves the room for me to think and self-study to finally come up with conclusions by myself. Most importantly, it is his emphasis on reproducibility of research which encourages the intellectual honesty and transparency. That being said, for working with him, not only I have sharpen my technical expertise but I also have a chance to refine my character.

I also want to thank many professors in Jackson School of Geosciences whom I have encountered. I particularly thank Dr. Kyle Spikes and Thomas Hess who acted as my very first research supervisors and introduced me the field of seismic processing. This led me to discover my interest and eventually meet Dr. Sergey Fomel. I also want to mention Dr. Stephen Grand, Dr. Clark Wilson, and Dr. Luc Lavier for their help in builing my foundation in geophysics.

I am grateful to many graduate students and research scientists I have met along the way: Harpreet Kaur, Ben Gremillion, Nam Pham, Zhicheng Geng, Luke Decker, Rebecca Gao, Hector Corzo, Dr. Yuzhi Shi, Dr. Ray Abma. Interacting with them helps me grow intellectually and personally. Prior to working with the Texas Consortium for Computational Seismology (TCCS), I used to be rather overconfident

student. However, having met and got to know both present and former TCCS students, I became more humble. I realize that there are entirely different playing field out there. This in turn aspired me to never stop learning.

Last but not least, I want to thank my soon-to-be colleage at PTTEP and my family back in Thailand for supporting me for all these years in the United States. Even without their physical presence, I never feel walking alone.

THARIT TANGKIJWANICHAKUL

The University of Texas at Austin

April 2021

Chain of Operators for Inverse Hessian Estimation in Least-squares Migration

Tharit Tangkijwanichakul, B.S.

The University of Texas at Austin, 2021

Supervisor: Dr. Sergey Fomel

I propose a novel way to approximate the inverse Hessian operator by a chain of weights in space and frequency domains. I call this method the Chain of Operators. Fundamentally, we approximate the complex operator (Hessian) via the chain of elementary operators (weights). This method is physically intuitive and provide a simple way to invert for the inverse Hessian. The method can be applied either for compensating migrated images i.e. applying approximated Inverse Hessian directly to migrated image or in the form of a preconditioner inside iterative least-squares reverse-time migration (LSRTM).

Tests on synthetic and real data shows that this approach provides an effective approximation of the Inverse Hessian while having the minimal cost of forward and inverse FFTs (Fast Fourier Transforms). However, the effectiveness of the methods vary quite greatly from dataset to dataset.

When used for compensating migrated image, the method proves noticeably effective in synthetic Marmousi dataset. With real Viking Graben dataset, the method

has difficulties in estimating weights due to muted area of the data. However, with some adjustments, the method noticeably improves the image resolution and amplitude balance. Plus, when examining frequency spectrum, compensating the image by an approximated inverse Hessian emulates the effect of iterative LSRTM.

When used the approximated Inverse Hessian to form a preconditioner in LSRTM, the result with Marmousi dataset shows that it can significantly accelerates the convergence of LSRTM and achieves high-quality imaging results in fewer iterations. However, when applied to the Sigsbee model, the method is only marginally effective. Particularly. yt appears to have difficulties in solving for the weight in freqeuncy domain as migrating the Sigsbee data with RTM tends to produce stronger low-frequency noise.

Overall, my experiments can be considered as a proof of concept that we can approximate the complex operator via the chain of elementary operators in the pariticular application to improving the efficentcy of Least-squares Migration.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

REVIEW OF SEISMIC IMAGING

Seismic Migration as a adjoint operator

The seismic reflection data $\mathbf{d}$ that is acquired in the survey has an approximately linear relationship with the earth reflecitivity model $\mathbf{m}$ as $\mathbf{d} = \mathbf{Lm}$ (**REF HERE**). The contrast of the lithology in the subsurface leads to difference in acoustic impedance and hence reflectivity $\mathbf{m}$. $\mathbf{L}$ is the operator that incorporates the physics of seismic wave propagation involved from the source to the subsurface and back to the receivers. In practice, choices of $\mathbf{L}$ can be classified to 2 categories: ray-based and wave-equation-based (**REF HERE**). The velocity model of the subsurface embedded in $\mathbf{L}$ is the crucial parameter that control the structural accuracy of the migrated image. In the subsequent work presented in this thesis, I am dealing exclusively with the situation (i.e the stage of imaging project) where the velocity model is assumed to be accurately estimated and left unchanged throughout. Otherwise, they problem will be non-linear which requires the different optimization approach (**REF HERE**).

The task of seimic imaging (or migration) is to construct the earth image represented by $\mathbf{m}$ from the acquired data $\mathbf{d}$. It is recognized that seismic migration is an adjoint operator $\mathbf{L^T}$of the its associated forward modeling $\mathbf{L}$ (Claerbout, 1992). In other words, the seismic image is constructed from back-projecting the seismic energy acquired on the surface to the point in the subsurface where that energy comes from.

This process is kinematically equivalent to applying adjoint operator $\mathbf{L^T}$ to the data $\mathbf{d}$.

However, one can see that $\mathbf{m_{mig}} = \mathbf{L^T d}$ is not the solution to the problem $\mathbf{d} = \mathbf{Lm}$. In fact, the least-squares solution is $\mathbf{m} = (\mathbf{L^T L})^{-1}\mathbf{L^T d}$. The missing term in $\mathbf{m_{mig}}$ is $(\mathbf{L^T L})^{-1}$ which is called Inverse Hessian. In other words, the migrated image is when I assume the inverse Hessian is identity matrix which is not generally true.

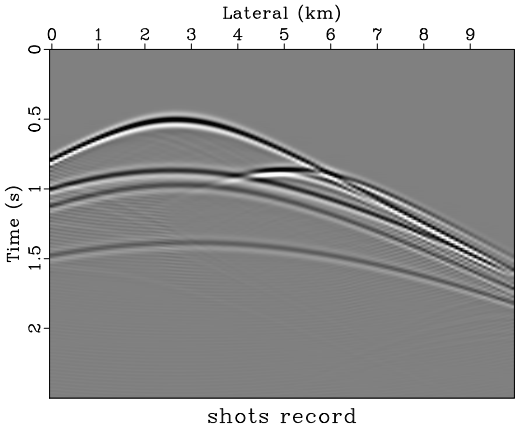
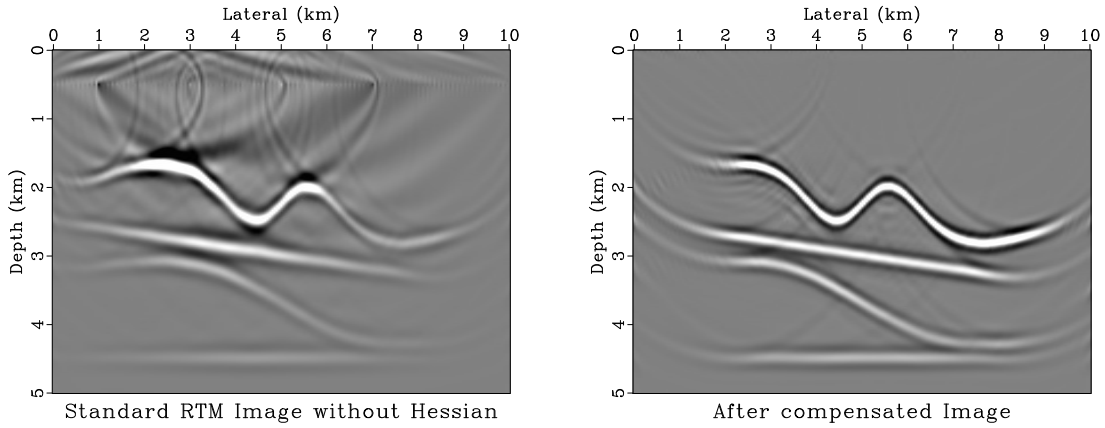


Figure 1.1: Simple Layer - Shot record chapter-intro/../chapter-intro/intro shots

To illustrate the effect of incorporating the inverse Hessian term, consider the synthetic shot record $\mathbf{d}$ in Figure 1.1 generated using finite-differnece wave propagator. If I apply the migration operator $\mathbf{L^T}$, I get the image in Figure 1.2(a). In contrast, the image in Figure 1.2(b) is when I compensated the standard migrated image $\mathbf{m_{mig}} = \mathbf{L^T d}$ with the Inverse Hesssian $(\mathbf{L^T L})^{-1}$ through the iterative inversion. One can see that both images are structurally similar. This is due to the fact that mentioned earlier that it is only the velocity model that controls the structural accuracy of the migrated image. Here, I do not change or make an update to velocity model through the iterative inversion. One can notice obvious differences in terms of



(a) (b)

Figure 1.2: Simple Layer - RTM Image(left) without Inverse Hessian (right) with Inverse Hessian chapter-intro/../chapter-intro/intro mig1,lsrtm

amplitude balancing and resolutions. The $\mathbf{m} = (\mathbf{L^T L})^{-1}\mathbf{L^T d}$ image has more balancing amplitude and improved resolution. This is the simple illustration that having accurately estimated inverse Hessian is beneficial.

DEVELOPMENT IN INVERSE HESSIAN ESTIMATION

Least-squares migration (LSM) is a well-established technique for improving the quality of seismic imaging through inversion (Nemeth et al., 1999; Ronen and Liner, 2000). By formulating seismic imaging as a linear estimation problem, least-squares migration utilizes the power of iterative inversion for recovering the reflectivity of subsurface structures. In recent years, least-squares imaging, in particular LSRTM (least-square reverse-time migration) has found many successful applications (Dai et al., 2012; Wang et al., 2016; Wong et al., 2011).

The least-squares formulation involves the inverse Hessian operator. As recog-

nized by previous researchers, the algorithm can be accelerated if the inverse Hessian matrix can be accurately approximated. Normally, the inverse Hessian is replaced by the identity matrix i.e. when only migration is performed, which is not a correct assumption. This is because it is not practical to form the inverse Hessian explicitly as it involves directly inverting an extremely large matrix.

In the conventional approach, the inverse Hessian is approximated by iterative methods, such as conjugate gradients (Tarantola, 1984; Sun et al., 2016; Xue et al., 2016). This raises the cost of LSRTM to the cost of migration and modeling multiplied by the number of iterations. Only a small number of iterations can be affordable in practice.

To reduce this cost or, alternatively, to reduce the number of iterations by accelerating the iterative convergence, a number of methods have been proposed in the literature for approximating the inverse Hessian using relatively inexpensive computations. Rickett (2003) estimated a simple scaling operator to approximate the inverse Hessian with a diagonal matrix. Guitton (2004) and Greer et al. (2018) extended this approach to matching filters, approximating the inverse Hessian with a nonstationary convolution operator. This approach is also known as *migration deconvolution* (Hu et al., 2001; Yu et al., 2006). Aoki and Schuster (2009) approximated inverse Hessian as a deblurring filter for regularization and precondition scheme. Kaur et al. (2020) adopt the deep learning technique and use generative adversarial networks in a conditional setting (CycleGANs) to approximate inverse Hessian.

This work is motivated mainly by an asymptotic theory (Miller et al., 1987; Bleistein, 1987). The theory shows that the inverse Hessian can be represented as a combination of weights in the time domain and the frequency domain. The result

extends to the case of LSRTM (Hou and Symes, 2015, 2016).

TECHNICAL CONTRIBUTIONS

I propose to approximate the inverse Hessian by reformulate by simultaneously estimating a chain of weights in the space and frequency domains from initial images. I design this chain to be symmetric to preserve the symmetric property of the Hessian. Our method which has relatively low computational cost compared to the overall Least-squares migration process. Once the weight matrices are estimated, they can efficiently approximate the inverse Hessian. Alternatively, the approximate inverse Hessian can be incorporated in the form of a preconditioner to accelerate the convergence of iterative LSRTM.

THESIS OUTLINE

This undergraduate thesis organized according to the following outline:

- In Chapter 2, I first present the theory of chain of oparators, problem formulation, and an estimation algorithm. In particular, I present the estimation of inverse Hessian applied to the Least-square Migration problem.
- In Chapter 3, I test the accuracy of the proposed approach using zero-offset synthetic data. I start with the synthetic data from the Marmousi model (Versteeg, 1994). Then, I also test with the real data from Viking Graben.
- Subsequently in Chapter 3, I use prestack Marmousi and Sigbee2A data to evaluate the effectiveness of the proposed approximation of Inverse Hessian by using it as a preconditioner for LSRTM.

Chapter 2

Chain of operator: Theory

THEORY

Motivation of chain of operators

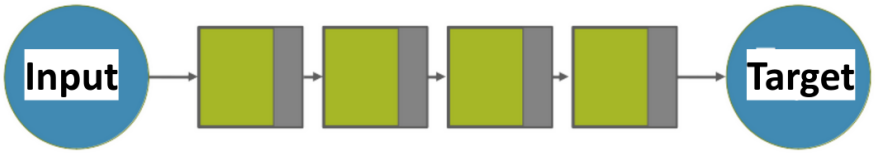


Figure 2.1: Schematic drawing of chain of operators chapter-background/../chapter-background/bg chain

The motivation behind the idea of the chain of operators is that one can approximate the complex operator through the chain of parameterized elementary operators. As illustrated in Figure 2.1, the input source is related to target through the chain of parameterized elemtary operator (green). Between each of these elementary operators are predetermined function (grey) chosen appropriately for the domain of application. In this work as motivated by asymtotic theory mentioned earlier, this predermined function is a Fouurier (and Inverse Fourier) Transform.

Problem formulation

Given surface seismic data $\mathbf{d}$, the first migrated image $\mathbf{m}_1$ is obtained by applying seismic migration as the adjoint of forward linear modeling operator $\mathbf{L}$:

$$\mathbf{m}_1 = \mathbf{L}^T \mathbf{d} \ . \tag{2.1}$$

Next, we can model and migrate again, generating the second image

$$\mathbf{m}_2 = \mathbf{L}^T \mathbf{L} \, \mathbf{m}_1 \tag{2.2}$$

If we can find an effective approximation for the Hessian operator from equation (2.2)

$$\mathbf{H} \approx \mathbf{L}^T \mathbf{L} \tag{2.3}$$

and its inverse, then applying this inverse to the initial image $\mathbf{m}_1$ will provide an effective approximation for the desired least-squares image

$$\mathbf{m}_3 = \mathbf{H}^{-1} \, \mathbf{m}_1 \approx \left(\mathbf{L}^T \mathbf{L} \right)^{-1} \mathbf{L}^T \mathbf{d} \ . \tag{2.4}$$

As discussed earlier that we cannot afford the cost of directly inverting the matrix, it is clear that the desired form of approximated Hessian should be easy, preferably trivial, to invert.

Chain operator in Least-squares Migration

We propose to approximate the connection between $\mathbf{m}_1$ and $\mathbf{m}_2$ in equation (2.2) using a chain of weights alternating between the original domain and the Fourier domain, as follows:

$$\mathbf{m}_2 \approx \mathbf{W}\mathbf{F}^{-1}\mathbf{W}_\mathbf{f}\mathbf{F}\mathbf{W}\mathbf{m}_1 \ , \tag{2.5}$$

where $\mathbf{W}$ and $\mathbf{W_f}$ are diagonal matrices, and $\mathbf{F}$ represents the Fourier transform. $\mathbf{W}$ represents the weighting factor that match the amplitude of the $\mathbf{m_1}$ and that of $\mathbf{m_2}$ in the original domain of the image (time or depth). $\mathbf{W_f}$ $\mathbf{W_f}$ is a weigting factor (or filter) that match the frequency of those two corresponding images. Our goal is to estimate $\mathbf{W}$ and $\mathbf{W_f}$ from the given pairs of $\mathbf{m}_1$ and $\mathbf{m}_2$,

In the digital filtering terminology, the operation of $\mathbf{F}^{-1}\mathbf{W_f}\mathbf{F}$ is equivalent to convolution. Therefore, the proposed chain represents a combination of scaling and convolution in a symmetric setting. The symmetry is important because the Hessian operator ($\mathbf{L^T L}$) , which we are approximating using this representation, is symmetric by definition.

Once we obtain $\mathbf{W}$ and $\mathbf{W_f}$, inverting the chain is straightforward as each weight matrices is a diagonal matrix. The compensated migrated image $\mathbf{m}_3$ from equation (2.4) becomes

$$\mathbf{m}_3 = \mathbf{W}^{-1}\mathbf{F}^{-1}\mathbf{W_f}^{-1}\mathbf{F}\mathbf{W}^{-1}\mathbf{m}_1 \ . \tag{2.6}$$

Solving for W and $\mathbf{W_f}$

To solve the nonlinear representation of chain of operators in equation (2.5), we introduce intermediate vectors $\mathbf{x_1}$ and $\mathbf{x_2}$ and treat them as additional unknown variables, splitting the initial condition into three linear equations, as follows:

$$\begin{aligned} \mathbf{x_1} &\approx \mathbf{W}\mathbf{m_1} \\ \mathbf{x_2} &\approx \mathbf{F}^{-1}\mathbf{W_f}\mathbf{F}\mathbf{x_1} \\ \mathbf{m_2} &\approx \mathbf{W}\mathbf{x_2} \end{aligned} \tag{2.7}$$

In the vector notation,

$$\mathbf{r} = \begin{bmatrix} \mathbf{W} & -\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}^{-1}\mathbf{W_f}\mathbf{F} & -\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{W} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{m_1} \\ \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{m_2} \end{bmatrix} \approx \mathbf{0} \,, \tag{2.8}$$

and the solution is found by minimizing the residual vector $\mathbf{r}$ while updating $\mathbf{w}$, $\mathbf{w_f}$, $\mathbf{x_1}$, and $\mathbf{x_2}$. Linearizing equation (2.8) with respect to perturbations in unknown variables, we arrive at

$$\begin{bmatrix} -\mathbf{I} & \mathbf{0} & \mathbf{diag}[\mathbf{m_1}] & \mathbf{0} \\ \mathbf{F}^{-1}\mathbf{W_f}\mathbf{F} & -\mathbf{I} & \mathbf{0} & \mathbf{F}^{-1}\mathbf{diag}[\mathbf{F}\mathbf{x_1}] \\ \mathbf{0} & \mathbf{W} & \mathbf{diag}[\mathbf{x_2}] & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{\Delta x_1} \\ \mathbf{\Delta x_2} \\ \mathbf{\Delta W} \\ \mathbf{\Delta W_f} \end{bmatrix} \approx -\mathbf{r} \tag{2.9}$$

We solve (2.9) using the conjugate-gradient method with shaping regularization that enforces the smoothness of estimated variables (Fomel, 2007b), then update $\mathbf{W}$, $\mathbf{W_f}$, and recalculate residual $\mathbf{r}$ in (2.8) iteratively until convergence. This corresponds to a regularized Gauss-Newton approach for solving the original nonlinear system.

Noted that to choose the step size in the updating scheme, we use a simple binary-search of step size. After obtaining the update vector from (2.9), we update the variables using step size = 1.0. Then, we check if the residual decreases or not. If does not, we rollback the variables and update them using step size = 0.5. We repeat this fashion until the residual decreases. See the Appendix for complete code implemtation (chain2dfft.c and Mchain2dfft.c)

It is clear that solving (2.5) is not guaranteed to achieve global minimum as the equation is non-convex with respect to $\mathbf{W}$ and $\mathbf{W_f}$. To mitigate the problem of being subjected to local minima, the initial guess $\mathbf{W_0}$ and $\mathbf{W_f 0}$ are supplied to the algorithms. To initialize the initial time/space weight $\mathbf{W}$, we use a smooth division

of $\mathbf{m_2}$ by $\mathbf{m_1}$ (Fomel, 2007a) and then take a square root. We set the initial frequency weight $\mathbf{W_f}$ to 1.0.

Preconditioning with chain weights

To go beyond applying the inverse Hessian in Post-stack image, we propose to apply the weights $\mathbf{W}$ and $\mathbf{W_f}$ to form a preconditioner to speed up the convergence of iterative least-square migration. The original problem is to solve for $\mathbf{m}$ where $\mathbf{d} \approx \mathbf{Lm}$. We apply the change of variables from $\mathbf{m}$ to $\mathbf{y}$ according to $\mathbf{m} = \mathbf{Py}$ where $\mathbf{P}$ is the preconditioning matrix. Then the problem becomes

$$\mathbf{d} \approx \mathbf{LPy} \tag{2.10}$$

We then solve this equation for $\mathbf{y}$ with the least-square objective function

$$J(\mathbf{y}) = \frac{1}{2}||(\mathbf{LP})\mathbf{y} - \mathbf{d})||_2^2 \ , \tag{2.11}$$

which corresponds to the analytical solution

$$\mathbf{y} = (\mathbf{P^T L^T LP})^{-1}\mathbf{P^T L^T d} \ . \tag{2.12}$$

After iteratively inverting for $\mathbf{y}$, we recover the original variable $\mathbf{m} = \mathbf{Py}$.

The iterative inversion of finding $\mathbf{y}$ is accelerated because according to the approximation of Hessian operator $\mathbf{L^T L} \approx \mathbf{WF^{-1}W_f FW}$

$$\begin{aligned} \mathbf{L^T L} &\approx \mathbf{WF^{-1}W_f FW} \\ &= (\mathbf{PP^T})^{-1} \ , \end{aligned} \tag{2.13}$$

the preconditioner $\mathbf{P}$ can be constructed from chain weights as

$$\mathbf{P} = \mathbf{W^{-1}F^{-1}W_f^{-1/2}F} \ . \tag{2.14}$$

This makes the inverted operator in equation (2.12) close to unitary,

$$(\mathbf{LP})^{\mathbf{T}}\mathbf{LP} \approx \mathbf{I}\,, \tag{2.15}$$

which accelerates the convergence when the inversion is carried out by an iterative method.

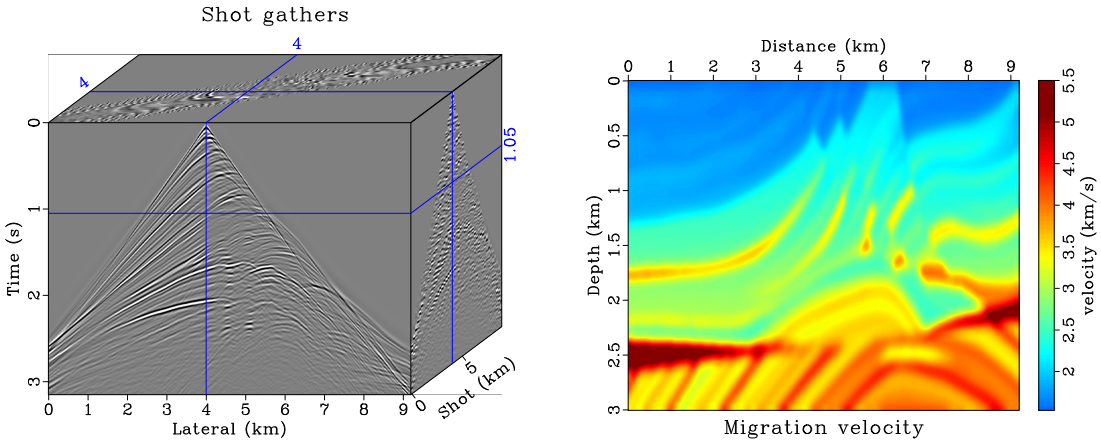
Chapter 3

Chain of operators: Experiments

ZERO-OFFSET EXPERIMENTS

Chain as a Deconvolution Filter: Synthetic Marmousi data

We test the idea of the chain of operators with Marmousi data (Versteeg, 1994). The shot gather volume is shown in Figure 3.1(a) and consists of 45 shots. The migration velocity is shown in Figure 3.1(b) which is obtained from smoothing the stratigraphic slowness. Figure 3.2 shows reverse-time migarted image using a finite-difference wave propagator.



(a) (b)

Figure 3.1: (a) Marmousi - Shot gathers (b) Migration velocity for prestack migration
chapter-lsrtm/../chapter-lsrtm/pre mmbshots45,velmig

This work EAGE CITE was done under the supervision of Dr. Sergey Fomel.

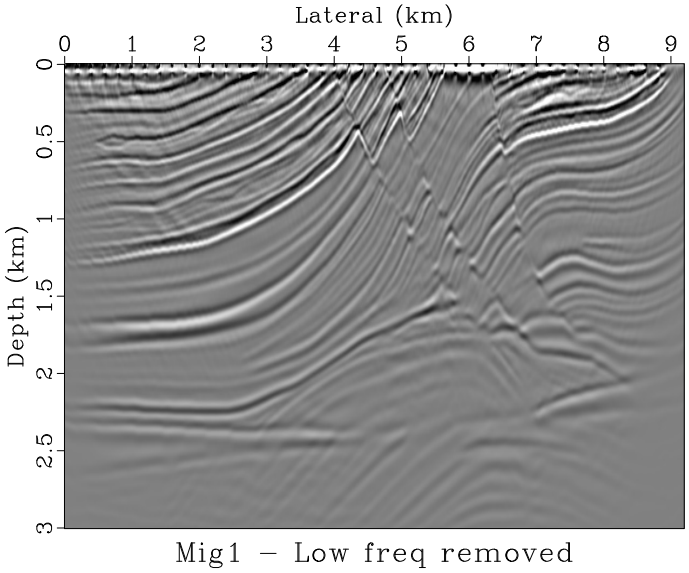
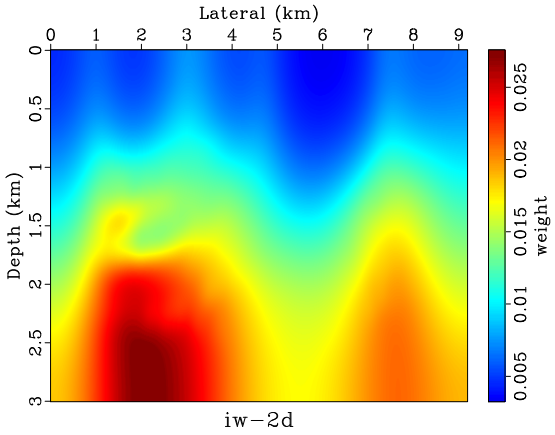


Figure 3.2: Marmousi - RTM Image chapter-lsrtm/../chapter-lsrtm/pre mmbmig1

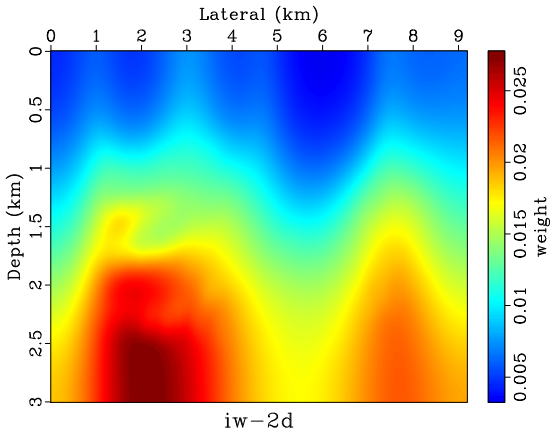
After we obtain the sencond migrated image $\mathbf{m_2}$ by remodeling and remigartion exercise ($\mathbf{L^T L}$) to $\mathbf{m_1}$, we form the initial weights $\mathbf{W_0}$ by taking the square-root of smooth division $\frac{\mathbf{m_2}}{\mathbf{m_1}}$ as mentioned earlier.

We run the chain solver to estimate $\mathbf{W}$ and $\mathbf{W_f}$ for different numbers of iterations. The residual is about 1.9 %, 0.23 %, 0.21 % of the zeroth iteration (i.e. with $\mathbf{W} = \mathbf{W_0}$ and $\mathbf{W_f} = \mathbf{I}$) after 2, 5, 10 iterations respectively. These weights are shown in Figure 3.3 and 3.4. Notice that the weights in space domain $\mathbf{W^{-1}}$ does not change much after more iterations, contrasting to the behavior of the weight in frequency domain $\mathbf{W_f^{-1}}$

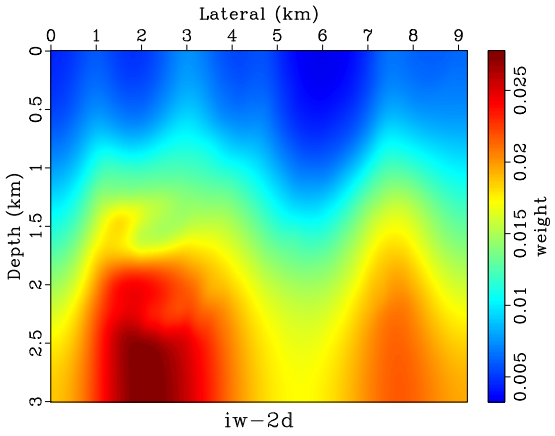
Subsequently, we used chain weights to perform poststack deconvolution as prescribed in equation (2.6). The result (Figure 3.5) shows an immediate improvement in resolution over that of the initial RTM image. The deconvolved image with 5 iterations has higher resolution compared to the one run with 2 iterations. However,



(a)

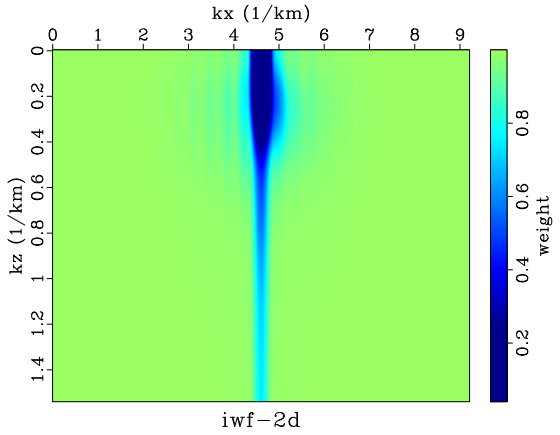


(b)

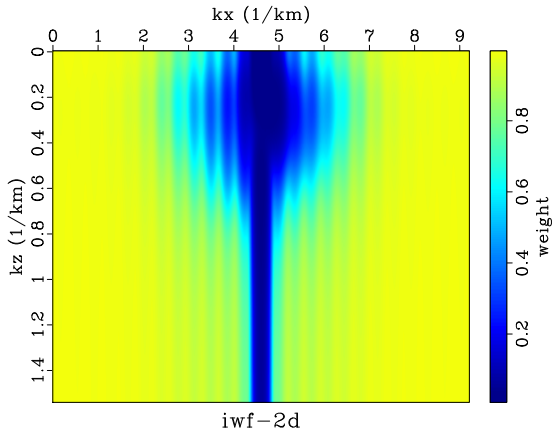


(c)

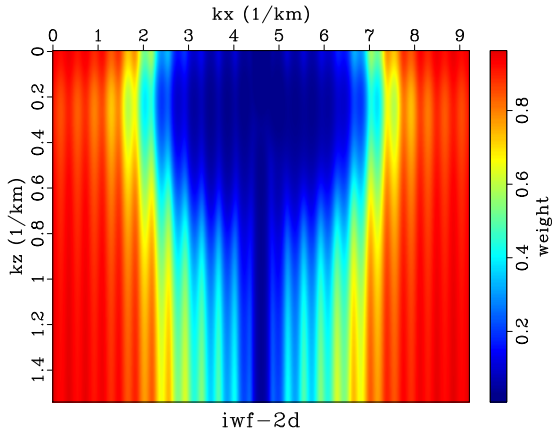
Figure 3.3: Marmousi - $\mathbf{W^{-1}}$ 2, 5, 10 iteration of update chapter-lsrtm/../chapter-lsrtm/pre iw,iw5,iw10



(a)



(b)



(c)

Figure 3.4: Marmousi - $\mathbf{W_f^{-1}}$ 2, 5, 10 iteration of update chapter-lsrtm/../chapter-lsrtm/pre iwf,iwf5,iwf10

the image with 5 iterations gives a smoother image. This may be from the the fact that we get smoother freqeuncy weight $\mathbf{Wf^{-1}}$ in the area where data mostly reside in Fourier domain i.e. 90° dip in (kx,kz) space.

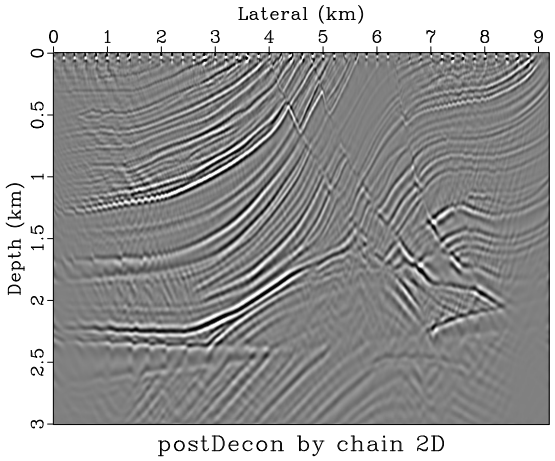
Chain as a Deconvolution Filter: Real Viking Graben data

In this experiment, we compare the result of using the chain of operator as a deconvolution filter applied to a zero-offset migrated image as in equation (2.6) with the traditional zero-offset iterative least-square migration. The zero-offset data (Fig 3.6(a)) comes from the Viking Graben Field. Its corresponding migration velocity is in (3.6(b)). We migrated the data using zero-offset RTM.

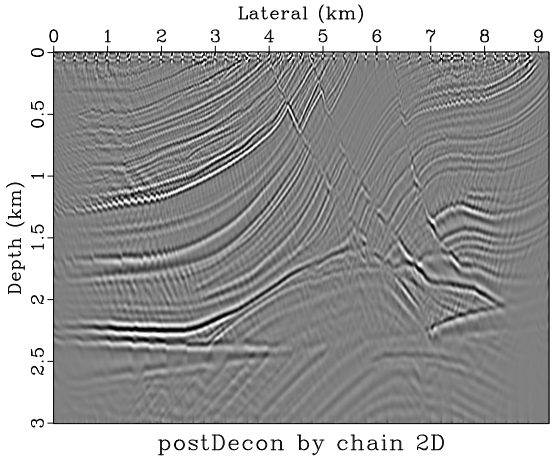
Noted that in the zero-offset data, we have area that we muted during the pre-processing (the area above 0.4s) Thus, the migrated image will have muted area in space domain near the surface too.The experiment shows that if we solve for $\mathbf{W}$, $\mathbf{Wf}$ using whole image, the result after applied deconvolution operator will have high frequency artifacts in the area where the data was muted originally.

Hence, we will test the effectiveness of deconvolution filter only the area where we have completed image. In other words, the weights are solved using input two windowed stacks of the first and second migrated image. In Fig 3.7(a), 3.7(b), 3.7(c) we show windowed section of the migrated image, deconvolved image by chain weights, and traditional iterative least-square migration repectively. To be clear, the image from iterative LS comes from using the whole image as an input still. We can see the amplitude range of deconvolutioned image and LSM image are about the same while that of standard migrated image is in different range.

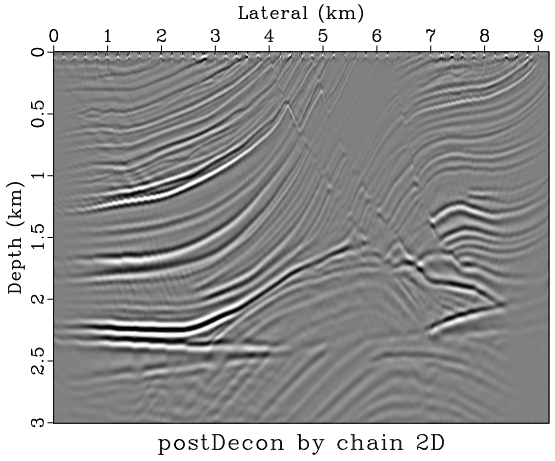
Fig 3.8 shows the zoom-in portion of these images. We can see that the layers



(a)

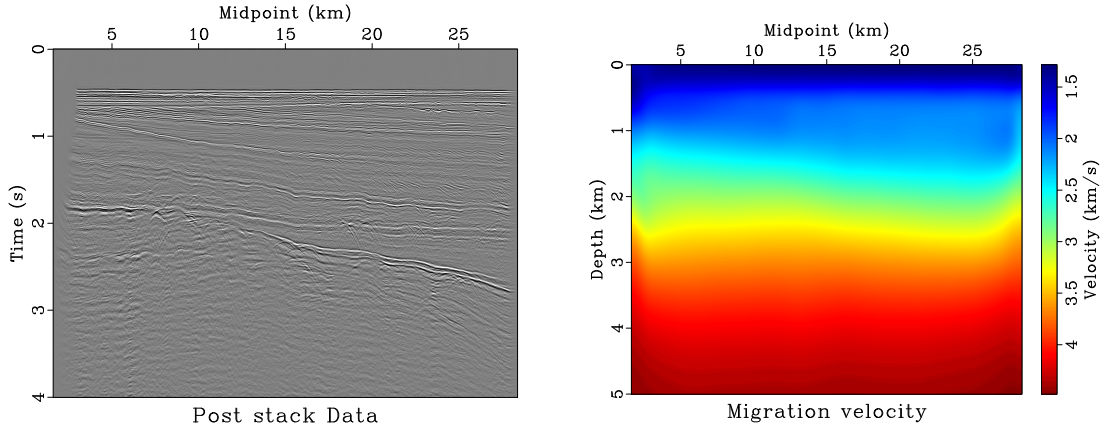


(b)



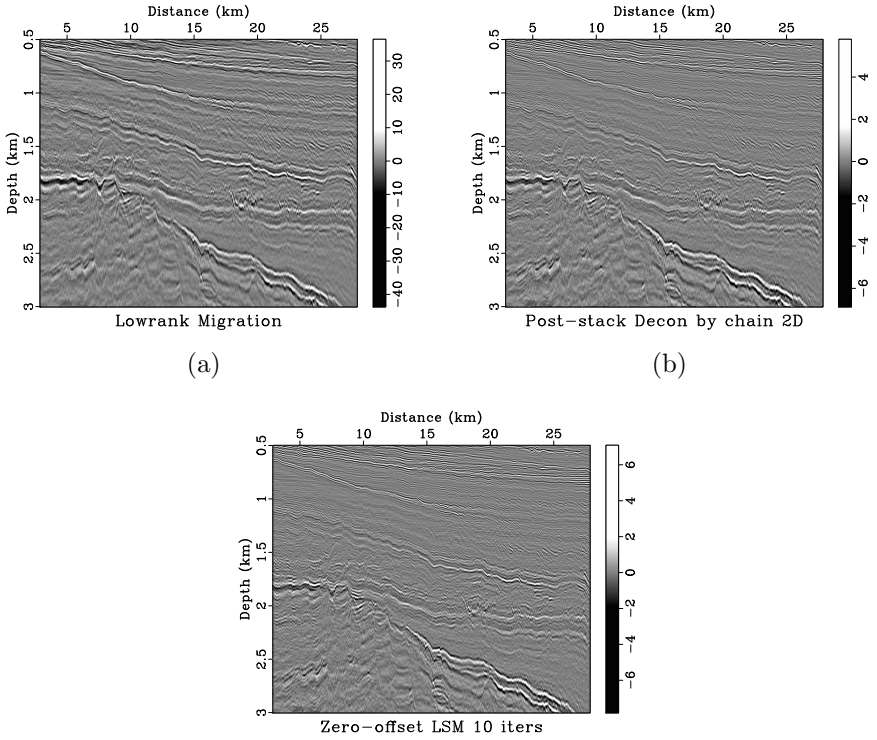
(c)

Figure 3.5: Marmousi - Poststack Deconvolution Image 2, 5, 10 iteration of update chapter-lsrtm/../chapter-lsrtm/pre decon2,decon5,decon10



(a) (b)

Figure 3.6: Viking - (a) Zero-offset shot (b) Dix Velocity for Migration chapter-lsrtm/../chapter-lsrtm/vk zodata,veldix



(a) (b)

(c)

Figure 3.7: Viking - (left) standard Zero-offset migration (mid) Deconvolved image by chain (right) Iterative LSM chapter-lsrtm/../chapter-lsrtm/vk cmig1,cdecon,lsm0

in deconvolutioned image looks more similar to LSM image in terms of amplitude balancing.

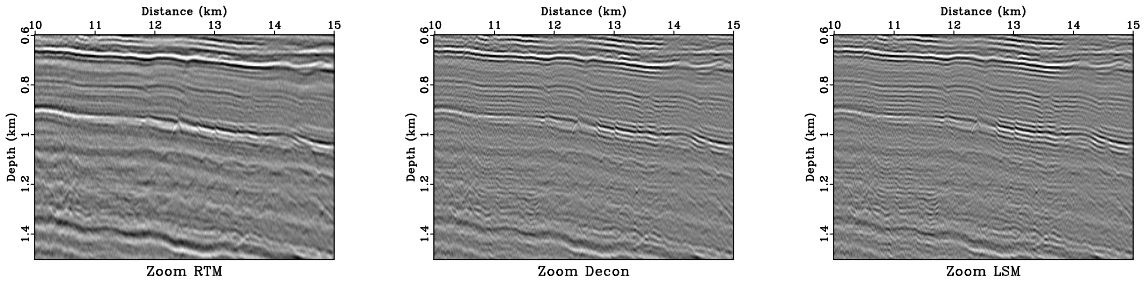


Figure 3.8: Viking - Zoom-in comparison chapter-lsrtm/../chapter-lsrtm/vk zooms

In Fig 3.9, we use the average frequency spectrum as one of the available quantitative tools to compare the image resolution. Deconvolution by chain can recover high frequency but not as good as iterative LSM, which has higher computational cost. The the pattern of frequency spectrum of deconvolved image by chain and that of iterative LSM appear similar which suggests that in this case the chain deconvolution filter derived from the chain emulates the inverse Hessian estimated by iterative LSM.

So far, estimating the inverse Hessian operator by chain of operators shows a promising result on the synthetic data. On the real Viking Graben data, we still gain the improvement of the image from using the method. However, the experiment shows that imperfects of the real data can limit the usability of the method.

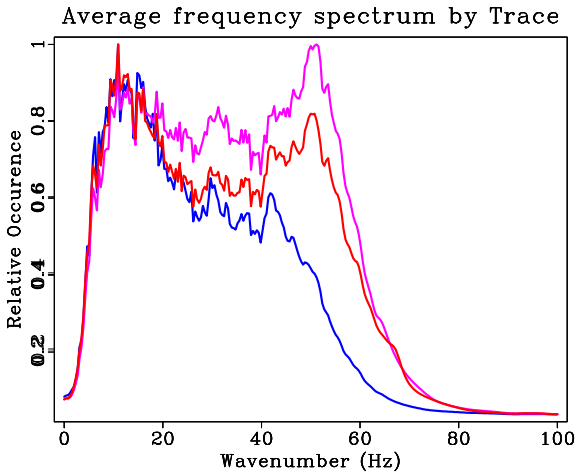


Figure 3.9: Viking - Frequency spectrum of - blue: standard ZORTM, red: Deconvolved by chain, pink: Iterative LSM chapter-lsrtm/../chapter-lsrtm/vk vkspec

PRE-STACK EXPERIMENTS

Chain as a Preconditioner for Least-square RTM

We use the weights in Figure 3.3 and 3.4) to form a preconditioner according to equation (2.14). In particular, we chose the weights obtained after 2 update iterations. This choice was not made based on any preference. Rather, it was due to availibity of results at that time when this subsequent experiment was performed.

In Figure 3.10, we show the change made to the generic conjugate-gradient algorithms to incorporate the preconditioning matrix $\mathbf{P}$ according to equation 2.14. The operator $\mathbf{L}$ and $\mathbf{L^T}$ can be any modeling/migration pair. In this case, we use Reverse-Time Migration (RTM). The illustration is modified from (Fomel et al., 2013). Here, we can see that incorporating preconditioning operator $\mathbf{P}$ incurs a negligible cost to the overall cost of LSRTM. In particular, the preconditioner costs 2 FFTs O(nlogn) while the cost of RTM significantly surpasses that. See the Appendix for the complete code of preconditioner (Mtf2dprec.c).

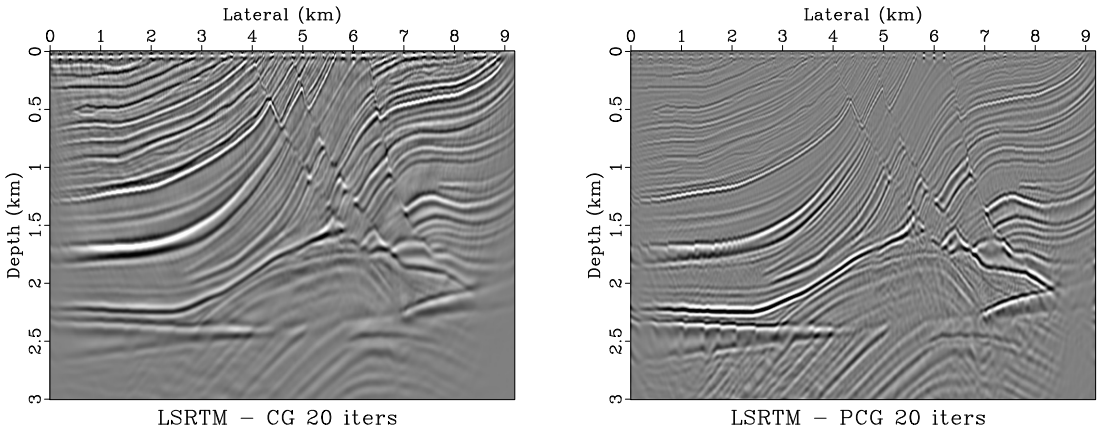
Figure 3.10: Changes in Conjugate-gradient algorithms to incorporate preconditioner chapter-lsrtm/../chapter-lsrtm/pre conj

Once we have all the code ready, we perform least-square reverse time migration using conjugate gradients for 20 iterations. The result without and without preconditioner is shown in Figure 3.11. The LSRTM image with precondtioner leads to a noticeable improvement in resolution. The amplitudes appear more balance, and the reflectors at greater depth are better illuminated.

The resolution improvement can be verified by examining the average frequency spectrum of each image (Figure 3.12). LSRTM with preconditioner (pink curve) has the broadest bandwidth compared to LSRTM without preconditioner (red curve) and RTM image (blue curve).

For closer inspection, the selected zoom-in sections are also provided in Figures 3.13,3.14,3.15,3.16

Subsequently, we run LSRTM for 100 iterations. To quantify the improvement, the misfit in data domain (Figure 3.17) is calculated using the L_2-norm of data residual $||\mathbf{d_k} - \mathbf{d}||_\mathbf{2}$ normalized by $||\mathbf{d}||_\mathbf{2}$. Here we also include the result of using only space



(a) (b)

Figure 3.11: Marmousi - (a) LSRTM Image without and (b) with Preconditioner after 20 iterations chapter-lsrtm/../chapter-lsrtm/pre cgrad20,pgrad20

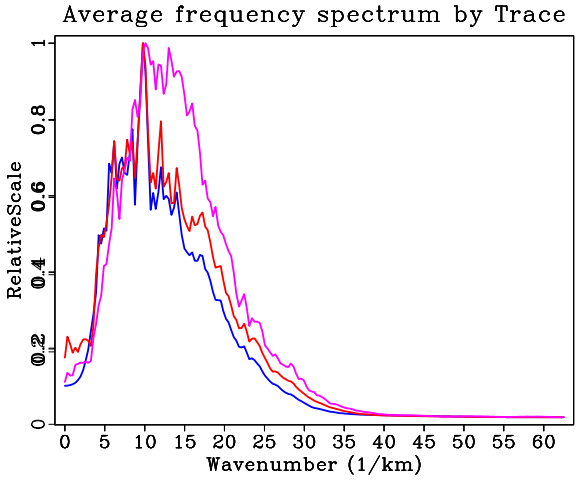


Figure 3.12: Marmousi - Frequency spectrum of migrated image - blue: standard RTM, red: LSRTM, pink: Precondition LSRTM chapter-lsrtm/../chapter-lsrtm/pre spec

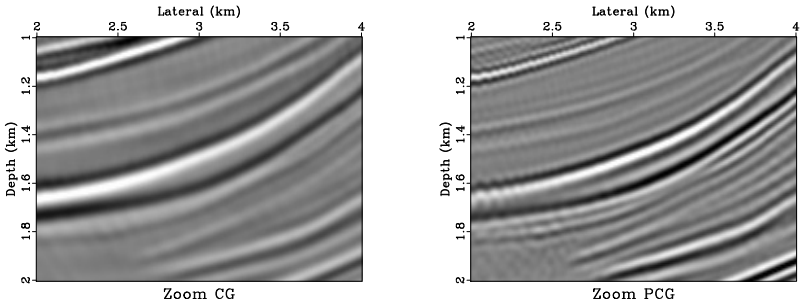


Figure 3.13: Marmousi - Zoom-in comparison 1
chapter-lsrtm/../chapter-lsrtm/pre zm1

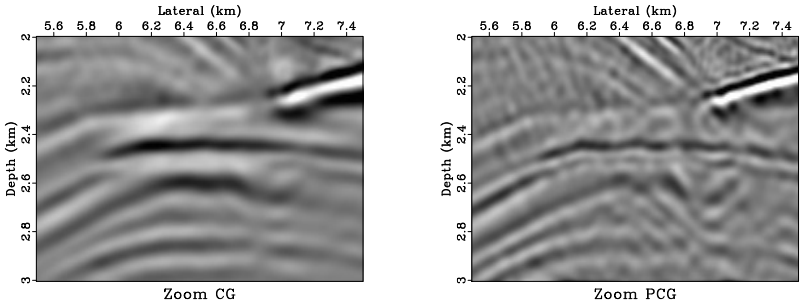


Figure 3.14: Marmousi - Zoom-in comparison 2
chapter-lsrtm/../chapter-lsrtm/pre zm2

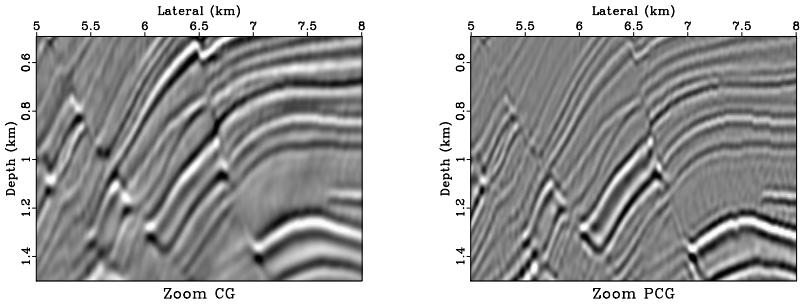


Figure 3.15: Marmousi - Zoom-in comparison 3
chapter-lsrtm/../chapter-lsrtm/pre zm3

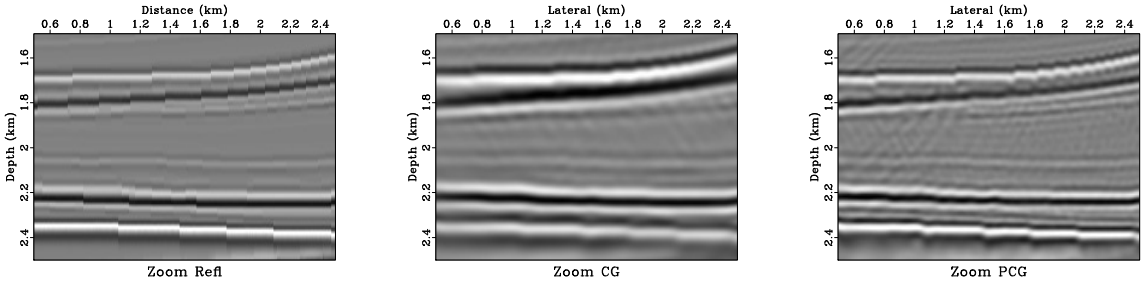


Figure 3.16: Marmousi - Zoom-in comparison 4 chapter-lsrtm/../chapter-lsrtm/pre zm4

weight $\mathbf{W}$ obtained from taking the square root of smooth division of $\frac{m_2}{m_1}$ while leaving $\mathbf{W_f}$ as $\mathbf{I}$ as precondtioner. This is the simplest form of preconditioner one comes up with. The plot confirms that the chain preconditioner leads to faster convergence while using space-only weight shows slight improvement in convergence. The LSRTM images after 100 iteration are shown in Figures 3.18.

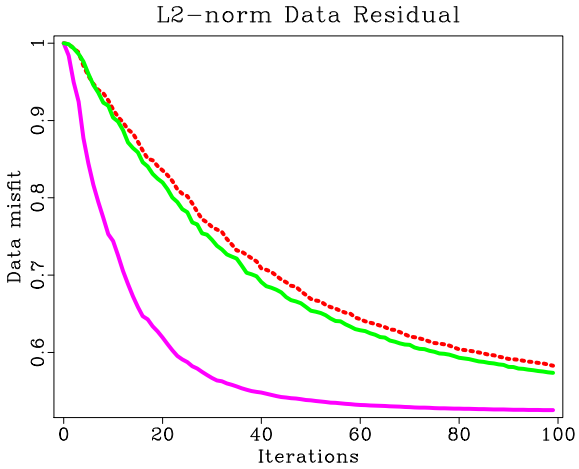


Figure 3.17: Marmousi - Normalize data misfit dash=without preconditioner solid(green)=with weight in space only solid(purple)=with chain preconditioner chapter-lsrtm/../chapter-lsrtm/pre dres100

In terms of amplitude balancing, LSRTM without preconditioner after 100

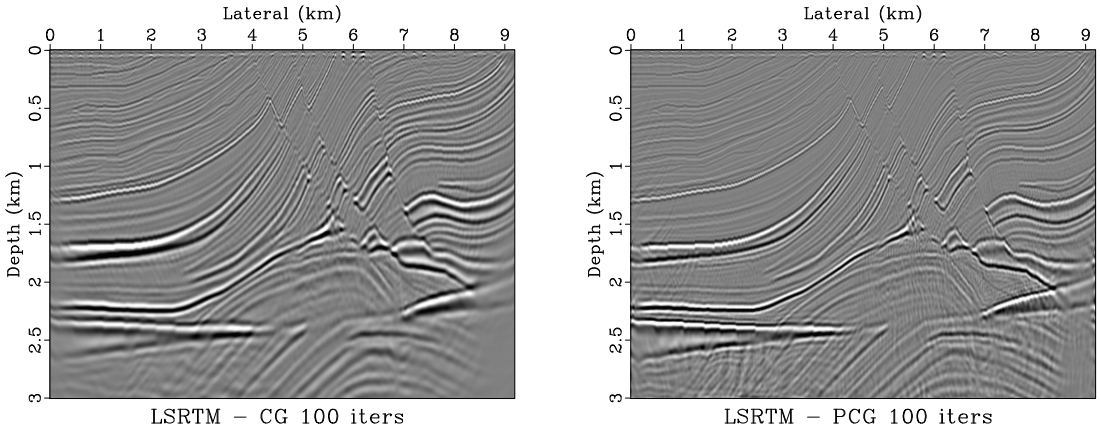


Figure 3.18: Marmousi - (left)LSRTM Image without Precondtitioner (mid) with Preconditioner (**W**,**Wf**) (right) Preconditioner (**W** only)after 100 iterations chapter-lsrtm/../chapter-lsrtm/pre cgrad100,pgrad100,wwpgrad100

iterations gives similar image to the image from LSRTM with precondtioner after only 20 iterations.

Next, we are still testing our method with synthetic data, but now we are using the data with a different geological setting. In the following experiment, we use the Sigsbee2A model which emulates the geological setting of the Gulf of Mexico. It is characterized by presences of nearly flat layers and a large salt body as shown by the stratigraphic velocity in Fig 3.19. The model was developed by The Subsalt Multiples Attenuation and Reduction Technology Joint Venture(SMAART JV).

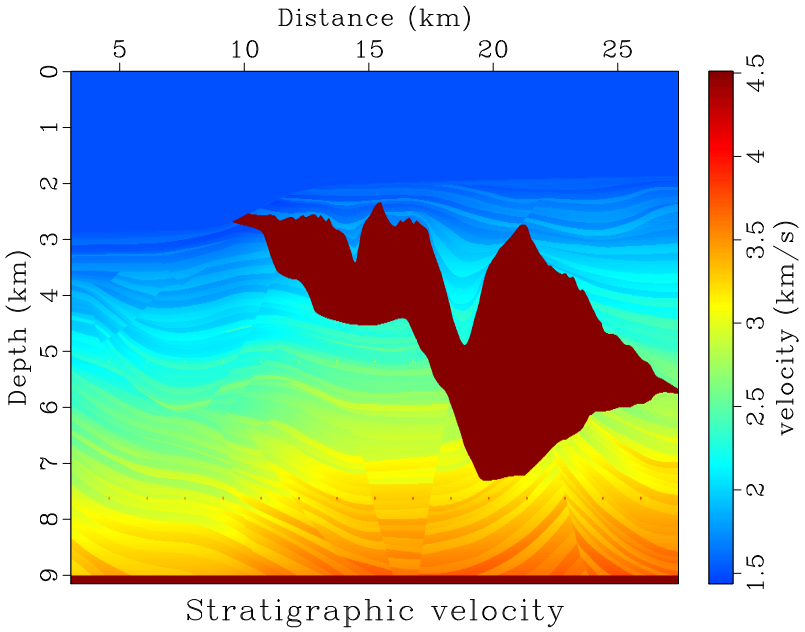


Figure 3.19: Sigsbee - Stratigraphic velocity chapter-lsrtm/../chapter-lsrtm/sigs veltrue

Starting with the shot record in Fig 3.20, we proceed as before by perform the Reverse-Time Migration. The result prior to removing low freqeuncy noise is shown in Fig 3.21(a) while the image after removeing it is shown in Fig 3.21(b) . From this initial assessment, we can expect in this dataset, the low frequency predominates the

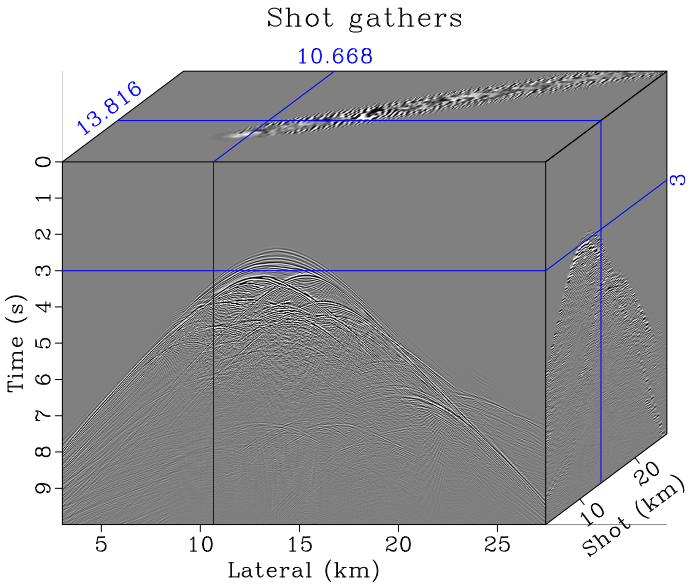
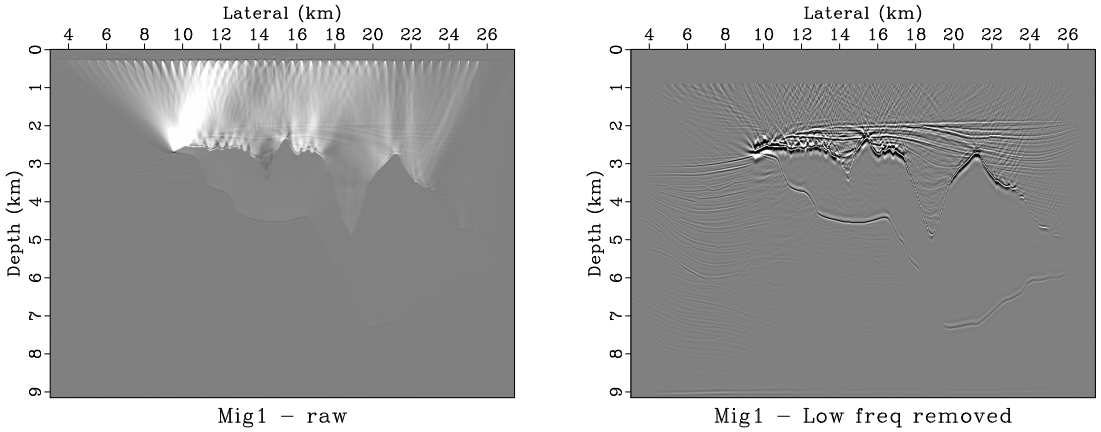


Figure 3.20: Sigsbee - Synthetic shots chapter-lsrtm/../chapter-lsrtm/sigs bshots45

migrated image and likely to cause difficulties to the chain algorithm in later step as we are trying to find the frequency weight $\mathbf{W_f}$.

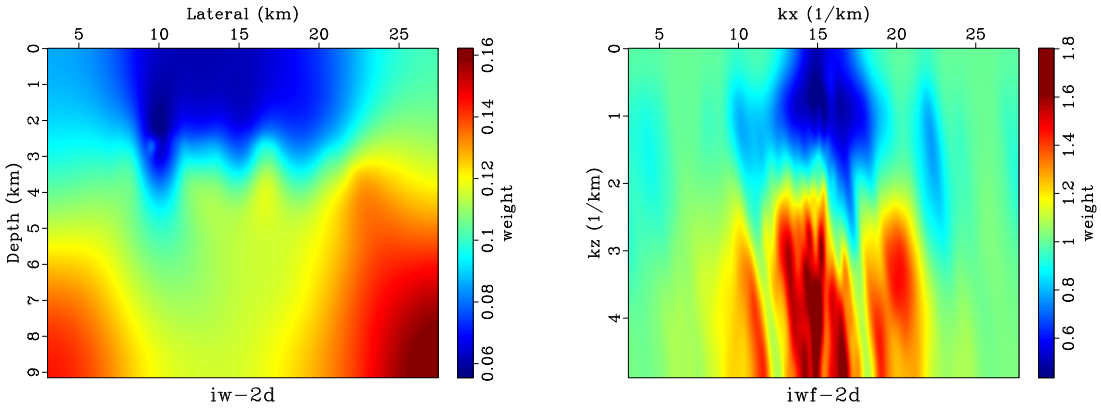
We solve for weight in space domain $\mathbf{W}$ and frequency domain $\mathbf{W_f}$ shown in Fig 3.22(a), 3.22(b) respectively. We can see that the weight in frequency domain looks unrealistic because with the presence of flat layers, the output of 2D Fourier Transform of the image should appear to be 90° dip in (kx,kz) domain. In fact, this is what we see in 3.4 in case of Marmousi model. We think this is because the artifacts of RTM (low frequency noise) makes it harder for the algorithms to find the true/resonable weights.

We carry out the experiment and perform Least-square RTM. Noted that in the process of LSRTM, we did not apply low-pass filter to remove low-frequency artifacts in any step of conjugate gradient. The result of LSRTM without preconditioner and with preconditioner is shown in Fig 3.23 and 3.24 respectively.



(a) (b)

Figure 3.21: Sigsbee - RTM image (a) before remove low-frequency noise (b) after remove low-frequency noise chapter-lsrtm/../chapter-lsrtm/sigs smig1,bmig1



(a) (b)

Figure 3.22: Sigsbee - (a) Weight in space domain $\mathbf{W}$ (b) in Frequency domain $\mathbf{W_f}$ chapter-lsrtm/../chapter-lsrtm/sigs iw-2d,iwf-2d

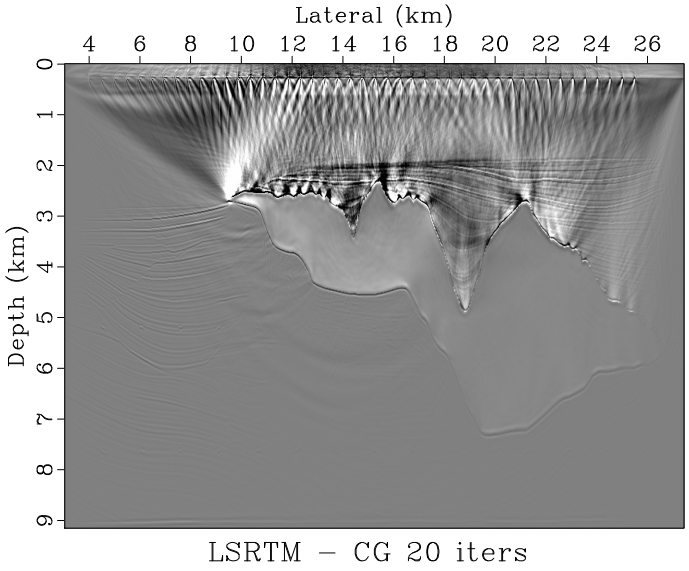


Figure 3.23: Sigsbee - LSRTM image without preconditioner chapter-lsrtm/../chapter-lsrtm/sigs cgrad-out0

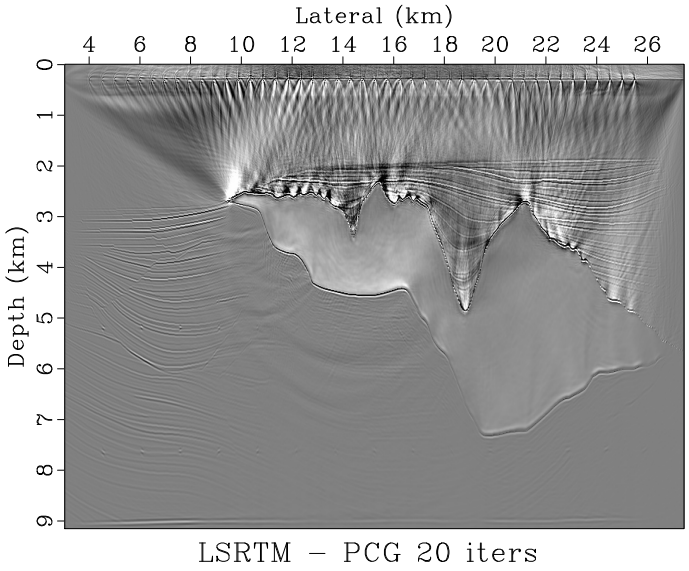


Figure 3.24: Sigsbee - LSRTM image with preconditioner chapter-lsrtm/../chapter-lsrtm/sigs pcgrad-out0

From these Figures 3.23 and 3.24, we can see that Least-squares RTM has trouble getting attenuating the low-frequency noise in the area above the salt body. However, to the left of the image where there is no salt body, LSRTM with precondtioner gives a better illumation. In fact, below the salt body around 12-16 km, preconditioned LSRTM also gives slightly better illumationation.

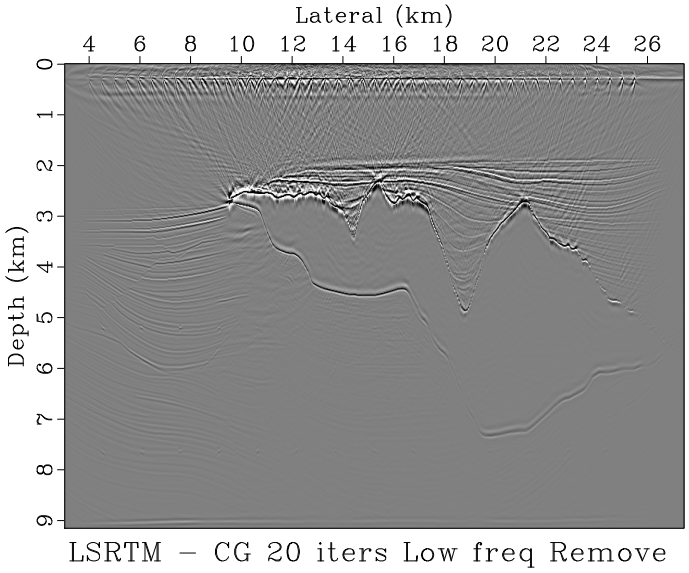


Figure 3.25: Sigsbee - LSRTM image without preconditioner after remove low frequency chapter-lsrtm/../chapter-lsrtm/sigs lsrtm0

Fig 3.25 and 3.26 show the LSRTM after remove the low frequency noise. Here, we see that the preconditioner has a very small positive effect on the final image, far less effective than the case of Marmousi data set. As mentioned above, we attribute ineffectiveness to the difficult in dealing with low-frequency noise originated from nature of RTM algorithms itself along with the presence of salt body. Given Sigsbee's complex geological setting, we do not practically have any alternatives to RTM as we must rely on two-way wavefield migration to correctly and suffienctly image geological features.

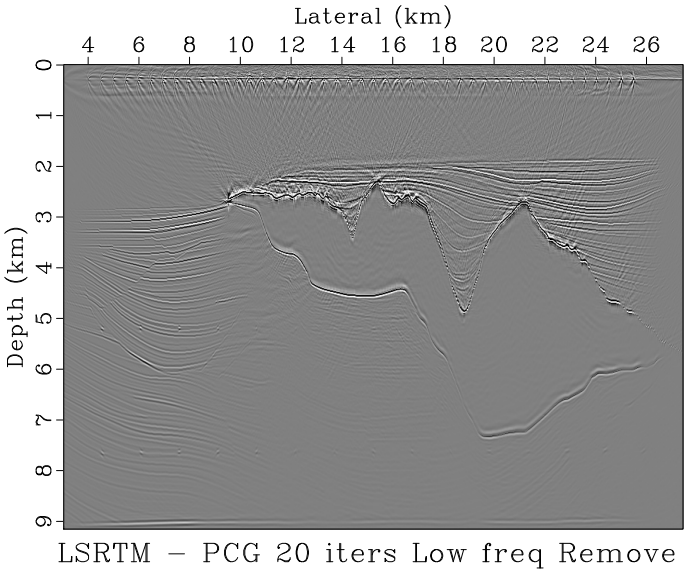


Figure 3.26: Sigsbee - LSRTM image with preconditioner after remove low frequency chapter-lsrtm/../chapter-lsrtm/sigs plsrtm0

Chapter 4

Conclusions

In this work, we propose a novel approach to approximating the inverse Hessian operator in least-squares migration. The method is data-driven and involves solving for weights in the original and Fourier domains that match two subsequent migrated images using the method called "chain of operators". The approximated inverse Hessian can be apply either as a decovolution filter to the migrated image or used to form a preconditioner for iterative prestack least-squares migration.

Our synthetic example shows the promising results in some cases and also reveals the challenges to the usability and limit of the algorithm. In the case that the algorithm work nicely, the results in the final image have improved resolution, balanced amplitudes, and a better data fit.

Overall, this work serves as a proof of concept of our proposed method "Chain of Operators". In other words, the complex operator can be effectively approximated through the chain of parameterized elementary operators - at least in the application of seismic least-square migration.

FUTURE WORK

For this particular application, we hope to further investigate the effectiveness of the algorithm with the real field-size prestack dataset. It is also worth studying how we

can come up with other scheme of solving for the chain weights beside the regularized Gauss-Newton approach.

In the field of seismic processing, the idea of chain of operators has many other promising applications. One area is to find the seismic shift between to two seismic traces. We can represent a small shift in seismic trace by one elemetary operator. Then, the bigger shift will become the chain of this small shift operators. In fact, we can also incorporate this shift to the approximation of the Hessian operator too.

It is also worth emphasizing that the idea of the chain of operators is not limited to the field of seismic processing. As we can see from the motivation and the generality of the hypothesis that "the complex operator can be effectively approximated through the chain of parameterized elementary operators".

Appendix

CODE

Below is the code for the code of the programs that are newly devoloped for this project. In particular, Mchain2dfft.c and chain2dfft.c are the program for solving for weights in space and frequency domain. Mtf2dprec.c and tf2dprec.c are the subroutine that is used to implement a preconditioning in a Conjugate-gradient solver. The library used in those can be found at Madagascar open-source software environment for reproducible computational experiments (Fomel et al., 2013). The package is available at `http://www.ahay.org/`.

Listing 1: chapter-lsrtm/code/Mchain2dfft.c

```
/* Find a symmetric chain of 2D-Fourier weighting and scaling with movies*/
/*
  Copyright (C) 2004 University of Texas at Austin

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

#include <rsf.h>
#include "chain2dfft.h"
#include "twosmooth2.h"
#include "fft2.h"



int main(int argc, char* argv[])
{
    int i, n, nk, n2, iter, niter, liter, nt, nx, nt1, nt2, nx2;
    int rect1, rect2, frect1, frect2;
    float dt,dx,x0;
    float l2_r,l2_r_new,alpha;
    float *w, *dw, *x, *y, *r, *p, *r_new, *w_prev;
    sf_file wht, fwht, src, tgt, mch;
    sf_file w0, wf0;
    /* For fft2 */
    bool isCmplx = false;
    int pad = 1;

```

```
    sf_init(argc,argv);
    src = sf_input("in");
    wht = sf_output("out");
    /* space weight */

    tgt = sf_input("target");
    /* target */

    w0 = sf_input("init_w");
    wf0 = sf_input("init_wf");

    fwht = sf_output("fweight");
    /* frequency weight */
    mch = sf_output("match");
    /* matched */


    if (!sf_histint(src,"n1",&nt)) sf_error("No n1= in input");
    if (!sf_histint(src,"n2",&nx)) sf_error("No n2= in input");
    if (!sf_histfloat(src,"d1",&dt)) sf_error("No d1= in input");
    if (!sf_histfloat(src,"d2",&dx)) sf_error("No d2= in input");
    if (!sf_histfloat(src,"o2",&x0)) x0=0.;

    if (!sf_getint("niter",&niter)) niter=0;
    /* number of iterations */
    if (!sf_getint("liter",&liter)) liter=50;
    /* number of linear iterations */


    n = nt*nx;

    nk = fft2_init(isCmplx, pad, nt, nx, &nt1, &nx2);
    nt2 = nk/nx2;

    n2 = 3*n+nk;
    sf_putint(fwht,"n1",nt2);
    sf_putint(fwht,"n2",nx2);

    w = sf_floatalloc(n2);
    dw = sf_floatalloc(n2);
    w_prev = sf_floatalloc(n2);

    x = sf_floatalloc(n);
    y = sf_floatalloc(n);
    r = sf_floatalloc(3*n);
    r_new = sf_floatalloc(3*n);


    if (!sf_getint("rect1",&rect1)) rect1=1;
    /* smoothing in time dim1*/
    if (!sf_getint("rect2",&rect2)) rect2=1;
    /* smoothing in time dim2*/
    if (!sf_getint("frect1",&frect1)) frect1=1;
    /* smoothing in frequency dim1 */
    if (!sf_getint("frect2",&frect2)) frect2=1;
    /* smoothing in frequency dim2 */

    twosmooth2_init(n,nk,nt,nt2,
            rect1,rect2,
            frect1,frect2,
            2*n);
```

```
101
    sf_floatread(x,n,src); /* source */
103 sf_floatread(y,n,tgt); /* target */

105 sfchain2d_init(nt,nx,nt1,nx2,nk,
       w+2*n,w+3*n,w,w+n,x);
107
    sf_conjgrad_init(n2, n2, 3*n, 3*n, 1., 1.e-6, true, false);
109

111 p = sf_floatalloc(n2);

113 /* initialize w [time w and freqz w] */
    for (i=0; i < 2*n; i++) {
115 w[i] = 0.0f;
    }
117
    sf_floatread(w+2*n, n, w0);
119 sf_floatread(w+3*n, nk, wf0);

121 for (iter=0; iter < niter; iter++) {
    sf_warning("Start %d",iter);
123
    for (i=0; i < n2; i++) {
125     w_prev[i] = w[i];
        }
127 alpha=1.0;

129 sfchain2d_res(y,r);
    l2_r = cblas_snrm2(3*n,r,1);
131 sf_warning("(Before update) L2 norm of res: %g", l2_r);

133 sf_warning("Residual %d",iter);

135 sf_conjgrad(NULL, sfchain2d_lop,twosmooth2_lop,p,dw,r,liter);


137
    for (i=0; i < n2; i++) {
139     w[i] += alpha*dw[i];
    }
141
    sfchain2d_res(y,r_new);
143
    l2_r_new = cblas_snrm2(3*n,r_new,1);
145
    sf_warning("(After update) L2 norm of res: %g, alpha = %g", l2_r_new,alpha);
147


149 while(l2_r_new>l2_r && alpha > 0.00001){
        sf_warning("Too big step !");
151
        for (i=0; i < n2; i++) {
153         w[i] = w_prev[i];
        }
155
        alpha *=0.5;
157
        for (i=0; i < n2; i++) {
159         w[i] += alpha*dw[i];
        }
161
```

```
        sfchain2d_res(y,r_new);
163
        l2_r_new = cblas_snrm2(3*n,r_new,1);
165     sf_warning("(After update) L2 norm of res: %g, alpha = %g", l2_r_new,alpha);
167
    }
169 sf_warning("Pass now !");
171
173
175 }/* End of iteration */
177 sf_floatwrite(w+2*n,n,wht);
    sf_floatwrite(w+3*n,nk,fwht);
179 sfchain2d_apply(y);
    sf_floatwrite(y,n,mch);
181
    exit(0);
183 }
```

Listing 2: chapter-lsrtm/code/chain2dfft.c

```
1  /* Chain of symmetric 2D-Fourier weighting and scaling */
   /*
3    Copyright (C) 2004 University of Texas at Austin

5    This program is free software; you can redistribute it and/or modify
     it under the terms of the GNU General Public License as published by
7    the Free Software Foundation; either version 2 of the License, or
     (at your option) any later version.
9
     This program is distributed in the hope that it will be useful,
11   but WITHOUT ANY WARRANTY; without even the implied warranty of
     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
13   GNU General Public License for more details.

15   You should have received a copy of the GNU General Public License
     along with this program; if not, write to the Free Software
17   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
   */
19
   #include <rsf.h>
21 #include "chain2dfft.h"
   #include "fft2.h"
23

25 static int n, nk, nw, nt, nx, nt1,nt2,nx2;
   static float *w, *wf, **tmp1, **tmp2, *x1, *x2, *s;
27 static sf_complex *ctmp1, *ctmp2; /* for 2D-fft */

29 void sfchain2d_init(int  n1     /* trace length */,
           int  n2     /* number of traces */,
31         int  n1pad  /* dim1 for 2-D fft input */,
           int  n_fftx /* dim2 for 2-D fft input */,
33         int  n_out_fft     /* dim for 2-D FFT output */,
           float *w1   /* weight [n1*n2] */,
35         float *wf1  /* Fourier weight [nk = nk*n2 (from fft2)) ] */,
           float *y1   /* intermediate [n1*n2] */,
```

```
        float *y2   /* intermediate [n1*n2] */,
        float *src  /* source [n1*n2] */)
/*< initialize >*/
{
    nt = n1;
    nx = n2;
    nt1 = n1pad;
    nx2 = n_fftx;
    nk = n_out_fft;
    nt2 = nk/n_fftx;
    nw = nt2;

    /*Model parameters*/
    n = n1*n2;
    w = w1;
    wf = wf1;
    x1 = y1;
    x2 = y2;
    s = src;

    tmp1 = sf_floatalloc2(nt1,nx2);
    tmp2 = sf_floatalloc2(nt1,nx2);

    /*for 2D fft*/
    ctmp1 = sf_complexalloc(nk);
    ctmp2 = sf_complexalloc(nk);

}

void sfchain2d_close(void)
/*< clean allocated storage >*/
{
    free(*tmp1);
    free(*tmp2);
    free(tmp1);
    free(tmp2);
    free(ctmp1);
    free(ctmp2);
}

void sfchain2d_res(const float *t /* target */,
     float *r       /* residual */)
/*< apply the chain operator >*/

{
    int i, i1, i2, ik;

    /* pad with zeros */
    for (i2=0; i2 < nx; i2++) {
  for (i1=0; i1 < nt; i1++) {
      tmp2[i2][i1] = x1[i1+i2*nt];
  }
  for (i1=nt; i1 < nt1; i1++) {
      tmp2[i2][i1] = 0.0f;
  }
    }
    for (i2=nx; i2 < nx2; i2++) {
  for (i1=0; i1 < nt1; i1++) {
      tmp2[i2][i1] = 0.0f;
  }
    }
```

```
    /* forward FFT */

    fft2_allocate(ctmp2);
    fft2(tmp2[0],ctmp2);

    /* frequency weight */
    for (ik=0; ik < nk; ik++) {
    ctmp2[ik] *= wf[ik];
  }
    /* inverse FFT */
    ifft2(tmp2[0],ctmp2);
    /* Compute residual r */
    for(i=0; i<nt*nx; i++){
    i1 = i%nt;
    i2 = i/nt;

    r[i]     = x1[i]-w[i]*s[i];
    r[n+i]   = x2[i]-tmp2[i2][i1];
    r[2*n+i] = t[i]-w[i]*x2[i];
    }
}

void sfchain2d_apply(float *y)
/*< apply the chain operator >*/

{
    int i, ik, i1, i2;

   /* pad with zeros */
    for (i2=0; i2 < nx; i2++) {
  for (i1=0; i1 < nt; i1++) {
      i = i1+i2*nt;
      tmp2[i2][i1] = w[i]*s[i];
  }
  for (i1=nt; i1 < nt1; i1++) {
      tmp2[i2][i1] = 0.0f;
  }
    }
    for (i2=nx; i2 < nx2; i2++) {
  for (i1=0; i1 < nt1; i1++) {
      tmp2[i2][i1] = 0.0f;
  }
    }

    /* forward FFT */
    fft2_allocate(ctmp2);
    fft2(tmp2[0],ctmp2);

    /* frequency weight */
    for (ik=0; ik < nk; ik++) {
  ctmp2[ik] *= wf[ik];
    }
    /* inverse FFT */
    ifft2(tmp2[0],ctmp2);

    for (i=0; i < n; i++) {
  i1 = i%nt;
  i2 = i/nt;

  y[i] = w[i]*tmp2[i2][i1];
```

```
    }
}



void sfchain2d_lop (bool adj, bool add, int nxx, int nyy, float* x, float* y)
/*< linear operator >*/
{
    int i, ik, i1, i2;
    int *dc_id, *nyq_id;
    int id;
    float scale;
    dc_id = sf_intalloc(nx);
    nyq_id = sf_intalloc(nx);


    if (nxx != 3*n+nk || nyy != 3*n) sf_error("%s: Wrong size",__FILE__);

    sf_adjnull(adj,add,nxx,nyy,x,y);

    if (adj) {
  for (i=0; i < n; i++) {
      i1 = i%nt;
      i2 = i/nt;

      tmp1[i2][i1] = y[n+i];
      tmp2[i2][i1] = x1[i];

      x[n+i] += w[i]*y[2*n+i] - y[n+i];
      x[2*n+i] += x2[i]*y[2*n+i];
  }

  /* pad with zeros */
  for (i2=0; i2 < nx; i2++) {
      for (i1=nt; i1 < nt1; i1++) {
    tmp1[i2][i1] = 0.0f;
    tmp2[i2][i1] = 0.0f;
      }
  }
  for (i2=nx; i2 < nx2; i2++) {
      for (i1=0; i1 < nt1; i1++) {
    tmp1[i2][i1] = 0.0f;
    tmp2[i2][i1] = 0.0f;
      }
  }

    fft2_allocate(ctmp2);
  fft2(tmp2[0],ctmp2);
    fft2_allocate(ctmp1);
  fft2(tmp1[0],ctmp1);

    for(id = 0; id < nx; id++){
        dc_id[id] = id*nw;
        nyq_id[id] = (id+1)*nw - 1;
    }

  for (ik=0; ik < nk; ik++) {
        scale = 2.0;
        for(id=0; id<nx; id++)
```

```
        {
            if(ik==dc_id[id] || ik == nyq_id[id]){
                scale = 1.0;
                break;
            }
        }

    x[3*n+ik] +=
    scale*crealf(ctmp1[ik]*conjf(ctmp2[ik])/(nt1*nx2));

    ctmp1[ik] *= wf[ik];
}
ifft2(tmp1[0],ctmp1);

for (i=0; i < n; i++) {
    i1 = i%nt;
    i2 = i/nt;

    x[2*n+i] += s[i]*y[i];
    x[i] += tmp1[i2][i1] - y[i];
}
  } else { /*Forward*/

for (i=0; i < n; i++) {
    i1 = i%nt;
    i2 = i/nt;

    y[i] += s[i]*x[2*n+i] - x[i];
    tmp1[i2][i1] = x[i];
    tmp2[i2][i1] = x1[i];
}
/* pad with zeros */
for (i2=0; i2 < nx; i2++) {
    for (i1=nt; i1 < nt1; i1++) {
  tmp1[i2][i1] = 0.0f;
  tmp2[i2][i1] = 0.0f;
    }
}
for (i2=nx; i2 < nx2; i2++) {
    for (i1=0; i1 < nt1; i1++) {
  tmp1[i2][i1] = 0.0f;
  tmp2[i2][i1] = 0.0f;
    }
}

  fft2_allocate(ctmp1);
fft2(tmp1[0],ctmp1);
for(ik=0; ik<nk;ik++){
  ctmp1[ik] *=wf[ik];
}
ifft2(tmp1[0],ctmp1);
  fft2_allocate(ctmp2);
fft2(tmp2[0],ctmp2);
for(ik=0; ik<nk;ik++){
  ctmp2[ik] *=x[3*n+ik];
}
ifft2(tmp2[0],ctmp2);

for (i=0; i < n; i++) {
    i1 = i%nt;
    i2 = i/nt;
```

```
      y[n+i] += tmp1[i2][i1] + tmp2[i2][i1] - x[n+i];
      y[2*n+i] += x2[i]*x[2*n+i] + w[i]*x[n+i];
  }
    }
}
```

Listing 3: chapter-lsrtm/code/Mtf2dprec.c

```
/* TF Weights Preconditioner for Real input as linear operator*/
/*
  Copyright (C) 2004 University of Texas at Austin

  This program is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 2 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/
#include <math.h>
#include <rsf.h>
#include "tf2dprec.h"
#include "fft2.h"

int main(int argc, char* argv[])
{
    int nz, nx, nz2, nx2, nk, nzx, n_left, i;
    int nk_rfft, nx_rfft;
    bool adj;
    float *ww, *ff;
    float *pp, *qq;
    sf_file src, out, w, wf;

    sf_init(argc,argv);

    src = sf_input("in");
    out = sf_output("out");
    w = sf_input("w");
    wf = sf_input("wf");

    if (!sf_histint(src,"n1",&nz)) sf_error("No n1= in input");
    if (!sf_histint(src,"n2",&nx)) sf_error("No n2= in input");
    /* dim from frequency weight - derived from real fft2 */
    if (!sf_histint(wf,"n1",&nk_rfft)) sf_error("No n1= in wf");
    if (!sf_histint(wf,"n2",&nx_rfft)) sf_error("No n2= in wf");


    nzx = nz*nx;
    n_left = sf_leftsize(src,2);

    nk = fft2_init(false, 1, nz, nx, &nz2, &nx2);

    if(nk_rfft*nx_rfft != nk) sf_error("FFT dimension error (nk)");
    if(nx_rfft != nx2) sf_error("FFT dimension error (nx)");
```

```
54
56      pp = sf_floatalloc(nzx);
        qq = sf_floatalloc(nzx);
58
        ww = sf_floatalloc(nz*nx);
60      sf_floatread(ww,nzx,w);
        sf_fileclose(w);
62
        ff = sf_floatalloc(nk);
64      sf_floatread(ff,nk,wf);
        sf_fileclose(wf);
66
        tf2dprec_init(nz, nx, nk, nz2, nx2, ww, ff);
68
        sf_floatread(pp,nzx,src);
70
        if (!sf_getbool("adj",&adj)) adj=false;
72
        for (i=0; i < n_left; i++) {
74        if (adj) {
              tf2dprec_lop(true,false,nzx,nzx,qq,pp);
76        } else {
              tf2dprec_lop(false,false,nzx,nzx,pp,qq);
78        }
        }
80
        sf_floatwrite(qq,nzx,out);
82
        exit(0);
84  }
```

Listing 4: chapter-lsrtm/code/tf2dprec.c

```
/* TF Weights Preconditioner for Real input as linear oper. */
2   #include <rsf.h>
    #include "tf2dprec.h"
4   #include "fft2.h"


6
    static int nz, nx, nz2, nx2, nk;
8   static float *w, *wf, **tmp1;
    static sf_complex *ctmp1; /* for 2D-fft */
10
12  void tf2dprec_init(int n1, /* trace length */
          int n2, /* number of length */
14        int nk_in, /* total Fourier size [n1*n2] */
          int nz2_in, /* Fourier dim1 */
16        int nx2_in, /* Fourier dim2 */
           float* ww /* [n1*n2] time weight */,
18         float* ff /* [nk] frequency weight */)
    /*< initialize >*/
20
    {
22      nz = n1;
        nx = n2;
24    nk = nk_in;
        nz2 = nz2_in;
26      nx2 = nx2_in;
        w = ww;
```

```
    wf = ff;
    /*for 2D fft*/
    tmp1 = sf_floatalloc2(nz2,nx2);
    ctmp1 = sf_complexalloc(nk);
}



void tf2dprec_close(void)
/*< clean allocated storage >*/
{
    free(*tmp1);
    free(tmp1);
    free(ctmp1);
}

void tf2dprec_lop(bool adj, bool add, int nxx, int nyy, float* x, float* y)
/*< linear operator >*/
{

  int i, i1, i2, ik;
    if (nxx != nz*nx || nyy != nz*nx) sf_error("%s: Wrong size",__FILE__);

    sf_adjnull(adj,add,nxx,nyy,x,y);

    if (adj){ /* Adjoint*/

     /* pad with zeros */
      for (i2=0; i2 < nx; i2++) {
    for (i1=0; i1 < nz; i1++) {
        i = i1+i2*nz;
        tmp1[i2][i1] = w[i]*y[i];
    }
    for (i1=nz; i1 < nz2; i1++) {
        tmp1[i2][i1] = 0.0f;
    }
      }
      for (i2=nx; i2 < nx2; i2++) {
    for (i1=0; i1 < nz2; i1++) {
        tmp1[i2][i1] = 0.0f;
    }
      }

      /* forward FFT */
      fft2_allocate(ctmp1);
      fft2(tmp1[0],ctmp1);

      /* frequency weight */
      for (ik=0; ik < nk; ik++) {
    ctmp1[ik] *= wf[ik];
      }
      /* inverse FFT */
      ifft2(tmp1[0],ctmp1);

      for (i=0; i < nz*nx; i++) {
    i1 = i%nz;
    i2 = i/nz;
    x[i] += tmp1[i2][i1];
      }

    } else{ /* Forward */
```

```
90      /* pad with zeros */
        for (i2=0; i2 < nx; i2++) {
92      for (i1=0; i1 < nz; i1++) {
            i = i1+i2*nz;
94          tmp1[i2][i1] = x[i];
        }
96      for (i1=nz; i1 < nz2; i1++) {
            tmp1[i2][i1] = 0.0f;
98      }
        }
100     for (i2=nx; i2 < nx2; i2++) {
        for (i1=0; i1 < nz2; i1++) {
102         tmp1[i2][i1] = 0.0f;
        }
104     }

106     /* forward FFT */
        fft2_allocate(ctmp1);
108     fft2(tmp1[0],ctmp1);

110     /* frequency weight */
        for (ik=0; ik < nk; ik++) {
112     ctmp1[ik] *= wf[ik];
        }
114     /* inverse FFT */
        ifft2(tmp1[0],ctmp1);
116
        for (i=0; i < nz*nx; i++) {
118     i1 = i%nz;
        i2 = i/nz;
120     y[i] += w[i]*tmp1[i2][i1];
        }
122     }
}
```

Bibliography

Aoki, N., and G. T. Schuster, 2009, Fast least-squares migration with a deblurring filter: Geophysics, **74**, WCA83–WCA93.

Bleistein, N., 1987, On the imaging of reflectors in the earth: Geophysics, **52**, 931–942.

Claerbout, J. F., 1992, Earth soundings analysis: Processing versus inversion, **6**.

Dai, W., P. Fowler, and G. T. Schuster, 2012, Multi-source least-squares reverse time migration: Geophysical Prospecting, **60**, 681–695.

Fomel, S., 2007a, Local seismic attributes: Geophysics, **72**, A29–A33.

———, 2007b, Shaping regularization in geophysical-estimation problems: Geophysics, **72**, R29–R36.

Fomel, S., P. Sava, I. Vlad, Y. Liu, and V. Bashkardin, 2013, Madagascar: open-source software project for multidimensional data analysis and reproducible computational experiments: Journal of Open Research Software, **1**, e8.

Greer, S., Z. Xue, and S. Fomel, 2018, Improving migration resolution by approximating the least-squares hessian using nonstationary amplitude and frequency matching, *in* SEG Technical Program Expanded Abstracts 2018: Society of Exploration Geophysicists, 4261–4265.

Guitton, A., 2004, Amplitude and kinematic corrections of migrated images for nonunitary imaging operators: Geophysics, **69**, 1017–1024.

Hou, J., and W. W. Symes, 2015, An approximate inverse to the extended born modeling operatoran approximate inverse operator: Geophysics, **80**, R331–R349.

———, 2016, Accelerating extended least-squares migration with weighted conjugate gradient iteration: Geophysics, **81**, S165–S179.

Hu, J., G. T. Schuster, and P. A. Valasek, 2001, Poststack migration deconvolution: Geophysics, **66**, 939–952.

Kaur, H., N. Pham, and S. Fomel, 2020, Improving resolution of migrated images by approximating the inverse hessian using deep learning: Geophysics, **85**, 1–62.

Miller, D., M. Oristaglio, and G. Beylkin, 1987, A new slant on seismic imaging: Migration and integral geometry: Geophysics, **52**, 943–964.

Nemeth, T., C. Wu, and G. T. Schuster, 1999, Least-squares migration of incomplete reflection data: Geophysics, **64**, 208–221.

Rickett, J. E., 2003, Illumination-based normalization for wave-equation depth migration: Geophysics, **68**, 1371–1379.

Ronen, S., and C. L. Liner, 2000, Least-squares dmo and migration: Geophysics, **65**, 1364–1371.

Sun, J., S. Fomel, T. Zhu, and J. Hu, 2016, Q-compensated least-squares reverse time migration using low-rank one-step wave extrapolation: Geophysics, **81**, S271–S279.

Tarantola, A., 1984, Inversion of seismic reflection data in the acoustic approximation: Geophysics, **49**, 1259–1266.

Versteeg, R., 1994, The marmousi experience: Velocity model determination on a synthetic complex data set: The Leading Edge, **13**, 927–936.

Wang, P., A. Gomes, Z. Zhang, and M. Wang, 2016, Least-squares rtm: Reality and possibilities for subsalt imaging, *in* SEG Technical Program Expanded Abstracts 2016: Society of Exploration Geophysicists, 4204–4209.

Wong, M., S. Ronen, and B. Biondi, 2011, Least-squares reverse time migration/in-

version for ocean bottom data: A case study, *in* SEG Technical Program Expanded Abstracts 2011: Society of Exploration Geophysicists, 2369–2373.

Xue, Z., Y. Chen, S. Fomel, and J. Sun, 2016, Seismic imaging of incomplete data and simultaneous-source data using least-squares reverse time migration with shaping regularization: Geophysics, **81**, S11–S20.

Yu, J., J. Hu, G. T. Schuster, and R. Estill, 2006, Prestack migration deconvolution: Geophysics, **71**, S53–S62.

Vita

Tharit Tangkijwanichakul graduated from Triam Udom Suksa High School in Thailand. He received a scholarship from PTT Exploration and Production (PTTEP), Thailand state-owned E&P Company, to study exploration geophysics in the USA. He pursued a Bachelor of Science degree in Geophysics from the Jackson School of Geosciences along with getting the certificate in Compuational Science and Engineering from Oden Institute of Computational Science and Engineering. After graduation, he will go back to Thailand and work full-time as a geophysicist for PTTEP.

Permanent address: 908 Poplar St. Austin, TX USA 78705

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.