A multilevel approach for approximating the inverse Hessian in four-dimensional variational data assimilation

Kirsty Lynne Brown

Submitted for
Degree of Doctor of Philosophy
Department of Mathematics and Statistics
University of Strathclyde, Glasgow

November 6, 2018

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

Abstract

Data assimilation methods are routinely employed in many scientific and technological fields. The inverse Hessian, and inverse square root Hessian, are of importance in various aspects of these procedures. The Hessian vector product is usually defined in the context of geophysical and engineering applications by the sequential solution of a tangent linear and adjoint problem. However, there are no readily available routines for computing the inverse Hessian, and inverse square root Hessian, vector products. It is generally necessary, when solving high-dimensional data assimilation problems, to operate in a matrix-free environment. A suitable method for generating a compact representation of the inverse Hessian, and inverse square root Hessian, is required in such cases.

A multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of any given symmetric positive definite matrix with eigenvalues clustered around unity is presented in this thesis. This algorithm is applied to the Hessian in the framework of incremental four-dimensional variational data assimilation (4D-Var), with the standard control variable transform implemented, in order to construct an approximation to the inverse Hessian. A novel decomposition of the Hessian as the sum of a set of local Hessians is introduced in this setting. Two practical variants of the multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of this Hessian are also presented. The accuracy of approximations to the inverse Hessian generated by applying the three algorithms proposed is investigated. The application considered in this thesis focuses on preconditioning the system of linear equations in the inner step of a Gauss-Newton procedure in incremental 4D-Var with an approximation to the inverse Hessian generated using the three algorithms proposed.

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Alison Ramage for her guidance throughout my time as a postgraduate student at Strathclyde. I would also like to thank my second supervisor Dr. Igor Gejadze, now based at the Institut national de recherche en sciences et technologies pour l'environnement et l'agriculture (IRSTEA), for his contribution to this project.

I would also like to acknowledge the Engineering and Physical Sciences Research Council (EPSRC) for funding for this project.

I would like to express my gratitude to the Nolan family for their encouragement and support whilst completing this thesis. Not least, Michael, for all of the adventures, and for always brightening up the day.

Finally, I would like to thank my family for their continued support over the years. I would especially like to thank my parents, Alex and Elizabeth, for encouraging me to pursue higher education, and for enabling me to complete this thesis.

Contents

1 Introduction **1**

1.1 Overview of data assimilation in Numerical Weather Prediction 1

1.2 The Hessian in variational data assimilation 3

1.3 Thesis outline 5

2 Four-dimensional variational data assimilation **7**

2.1 The standard formulation of 4D-Var 8

2.2 The tangent linear model 9

2.3 Linear analysis estimate 10

2.4 The adjoint model 13

2.5 The Gauss-Newton method 14

2.6 The incremental formulation of 4D-Var 16

2.7 The inner loop problem 18

2.8 The Hessian in incremental 4D-Var 19

2.9 The control variable transform 19

2.10 Limited-memory approximation to the Hessian 22

3 Background methods **23**

3.1 The Lanczos method 24

3.2 The QR method 27

3.3 The implicitly restarted Lanczos method 30

3.4 The conjugate gradient method 34

3.5 The preconditioned conjugate gradient method 36

4 A multilevel eigenvalue decomposition algorithm for approximating the inverse Hessian 38

4.1 Multilevel grid structure . . . 39
4.2 Interpolation and restriction operators . . . 40
4.3 Additional grid transfer operators . . . 41
4.4 The multilevel eigenvalue decomposition algorithm . . . 45
4.4.1 Outline . . . 45
4.4.2 Details . . . 45
4.4.3 Preconditioning strategy . . . 47
4.4.4 Summary . . . 48
4.5 Approximating the inverse Hessian . . . 50
4.6 Model problem . . . 50
4.7 Implementation details . . . 52
4.8 Evaluating the accuracy of approximations using the Riemannian distance . . . 53
4.9 Choice of preconditioner . . . 55
4.10 Choice of interpolation method . . . 56
4.11 Executing one shift in ARPACK . . . 58
4.12 Comparing approximations to the inverse Hessian . . . 60
4.13 Conclusion . . . 69

5 Practical algorithms for approximating the inverse Hessian 72

5.1 Modified version of the multilevel eigenvalue decomposition algorithm . . . 73
5.2 Decomposition of the Hessian . . . 76
5.3 The Hessian decomposition algorithm . . . 80
5.3.1 Outline . . . 80
5.3.2 Details . . . 81
5.3.3 Summary . . . 81
5.4 Approximating the inverse Hessian using the Hessian decomposition algorithm . . . 82
5.5 Additional implementation details . . . 83
5.6 Comparing approximations to the inverse Hessian . . . 83

5.7 The reduced memory Hessian decomposition algorithm 89
5.7.1 Outline 89
5.7.2 Details 90
5.7.3 Summary 91
5.8 Approximating the inverse Hessian using the reduced memory Hessian decomposition algorithm 93
5.9 Comparing approximations to the inverse Hessian 93
5.10 Conclusion 98

6 Preconditioning in a Gauss-Newton procedure using approximations to the inverse Hessian 101
6.1 Additional model problems 103
6.2 Implementation details 104
6.3 Evaluating preconditioner performance 105
6.4 Choices of preconditioners 106
6.5 Introducing preconditioning at different stages of the Gauss-Newton method 108
6.6 Comparing preconditioners constructed using the multilevel eigenvalue decomposition algorithm 111
6.7 Comparing preconditioners constructed using the Hessian decomposition and reduced memory Hessian decomposition algorithms 114
6.8 Conclusion 116

7 Conclusions 130

A Tangent linear problem for Burgers' equation 134

List of Figures

2.1 Outline of the steps involved in solving (2.26)-(2.27) using the Gauss-Newton method. 16

2.2 Outline of the main steps of incremental 4D-Var. 17

2.3 Outline of the updated steps of incremental 4D-Var with the control variable transform (2.41)-(2.43) applied. 21

3.1 The QR method. 29

3.2 The shifted QR method. 29

4.1 Outline of the multilevel eigenvalue decomposition algorithm. 49

4.2 Flow evolution of $\varphi(x,t)$ satisfying (4.32)-(4.34) for initial condition $\varphi_1(x,0)$ in (4.35). 51

4.3 Standard deviation $\boldsymbol{\sigma}_k$ plotted as a function of x on $\Omega = [0,1]$. 55

4.4 Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles), $\tilde{A}_0^{-1}$ where P_k^{k-i} has been implemented using cubic spline interpolation (red circles), and $\tilde{A}_0^{-1}$ where P_k^{k-i} has been implemented using linear interpolation (black circles). Two combinations of R_e and N_e presented in Table 4.1 are highlighted. 57

4.5 Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles), $\tilde{A}_0^{-1}$ where the standard version of ARPACK has been applied (red circles), and $\tilde{A}_0^{-1}$ where the proposed modified version of ARPACK has been applied (black circles). Two combinations of R_e and N_e presented in Table 4.2 are highlighted. 59

4.6 Comparison plot showing the first eighty eigenvalues of $\tilde{A}_0^{-1}$ (red circles). The first eighty eigenvalues of A_0^{-1} plotted using blue circles are indistinguishable from the eigenvalues of $\tilde{A}_0^{-1}$. The combination implemented is $R_e = 400$ and $N_e = (400, 0, 0, 0)$. The associated measures of accuracy and computational cost are $D_e = 5.5185e-13$ and $M_e = 401$, respectively. 61

4.7 Comparison plot showing D_e plotted against R_e where the maximum allowable memory ratio is $R_e = 24$. The average of D_e calculated with respect to the 5% of cases of N_e that resulted in the lowest D_e (solid blue line). The true minimum D_e attained (dashed blue line). The result D_e obtained where the case of N_e implemented involved only the finest grid at grid level $k = 0$ (dotted blue line). The value D_e obtained where the case of N_e implemented resulted from applying the doubling strategy (red crosses). 62

4.8 Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles) and $\tilde{A}_0^{-1}$ (red circles). Six combinations of R_e and N_e are highlighted. The results D_e obtained are presented. 64

4.9 Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles) and $\tilde{A}_0^{-1}$ (red circles). Six combinations of R_e and N_e are highlighted. The results D_e obtained are presented. 65

5.1 Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles), $\tilde{A}_0^{-1}$ where the standard version of the multilevel eigenvalue decomposition algorithm has been applied (red circles), and $\tilde{A}_0^{-1}$ where the proposed modified version of the multilevel eigenvalue decomposition algorithm has been applied (black circles). Two combinations of R_e and N_e presented in Table 5.1 are highlighted. 76

5.2 Outline of the Hessian decomposition algorithm. 82

5.3 Comparison plots showing the first eighty eigenvalues of: C_0^{-1} (blue circles) and $\tilde{C}_0^{-1}$ (red circles). The eigenvalues of C_0^{-1} are indistinguishable from the eigenvalues of $\tilde{C}_0^{-1}$ in the plot in subfigure 5.3(a). The four combinations of k and n_k^l presented in Table 5.2 are highlighted. The results D_e obtained are presented. 85

5.4 Comparison plots showing the first eighty eigenvalues of: C_0^{-1} (blue circles) and $\tilde{C}_0^{-1}$ (red circles). Four combinations of n_k^l and N_e presented in Table 5.4 are highlighted. The results D_e obtained are presented. . . 88

5.5 Outline of the reduced memory Hessian decomposition algorithm. 92

5.6 Comparison plot showing the first eighty eigenvalues of $\tilde{C}_0^{-1}$ (red circles). The first eighty eigenvalues of C_0^{-1} plotted using blue circles are indistinguishable from the eigenvalues of $\tilde{C}_0^{-1}$. The combination implemented is $k = 0$ where $n_k^l = 400$, $N_k^l = (400, 0, 0, 0)$, and $N_e = (400, 0, 0, 0)$. The associated measures of accuracy and computational cost are $D_e = 4.3068e - 12$ and $\hat{M}_e = 401$, respectively. 95

5.7 Comparison plots showing the first eighty eigenvalues of: C_0^{-1} (blue circles) and $\tilde{C}_0^{-1}$ (red circles). Four cases of N_k^l presented in Table 5.5 are highlighted. The results D_e obtained are presented. 96

6.1 Flow evolution of $\varphi(x,t)$ satisfying (4.32)-(4.34) for initial condition $\varphi_2(x,0)$ in (6.3). 103

6.2 Convergence diagram for model problem MP1 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioner P1B introduced at outer iteration number j of the Gauss-Newton method: $j = 1$ (solid red line), $j = 2$ (solid blue line), $j = 3$ (solid green line), and $j = 6$ (solid pink line). 110

6.3 Convergence diagram for model problem MP1 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P1A (solid red line), P1B (solid blue line), P1C (solid green line), and P1D (solid pink line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 113

6.4 Convergence diagram for model problem MP1 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 117

6.5 Convergence diagram for model problem MP1 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 118

6.6 Convergence diagram for model problem MP2 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 120

6.7 Convergence diagram for model problem MP2 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 121

6.8 Convergence diagram for model problem MP3 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 123

6.9 Convergence diagram for model problem MP3 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 124

6.10 Convergence diagram for model problem MP4 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 126

6.11 Convergence diagram for model problem MP4 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'. 127

List of Tables

4.1 The values D_e and M_e obtained for four combinations of R_e and N_e, respectively, with P_k^{k-i} implemented using linear or cubic spline interpolation. 57

4.2 The values D_e and M_e obtained for four combinations of R_e and N_e, respectively, with the standard or proposed modified version of ARPACK applied. 59

4.3 The average values D_e^a and M_e^a calculated with respect to the 5% of cases of N_e that resulted in the lowest D_e for six instances of R_e included in the plot in Figure 4.7. 66

4.4 The values D_e and M_e obtained for six cases of R_e included in the plot in Figure 4.7 where N_e resulted in the true minimum D_e. 66

4.5 The values D_e and M_e obtained for six cases of R_e included in the plot in Figure 4.7 where N_e involved only the finest grid at grid level $k = 0$. . 67

4.6 The values D_e and M_e obtained for six cases of R_e included in the plot in Figure 4.7 where N_e followed the doubling strategy. 67

5.1 The values D_e and M_e obtained for four combinations of R_e and N_e, respectively, using the standard or proposed modified version of the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1. 75

5.2 The values D_e and $\hat{M}_e$ obtained for four combinations of k and n_k^l in N_d where $N_e = (400, 0, 0, 0)$. The memory ratio R_d is also tabulated. 85

5.3 The values D_e and $\hat{M}_e$ obtained for the six cases of N_e presented in Table 4.6 where $k = 1$ and $n_k^l = 200$ in N_d. The memory ratio R_d is also tabulated. 87

5.4 The values D_e and $\hat{M}_e$ obtained for the six cases of N_e presented in Table 5.3 where $k = 1$ and n_k^l in N_d is specific to the particular case of N_e implemented. The memory ratio R_d is also tabulated. 87

5.5 The values D_e and $\hat{M}_e$ obtained for six cases of N_k^l where $k = 1$, $n_k^l = 9$ in N_d, and $N_e = (6, 12, 24, 48)$. The memory ratio $\hat{R}_d$ is also tabulated. . 94

6.1 Summary of the four model problems studied. 104

6.2 Details of the four preconditioners generated using the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1. 107

6.3 Details of the preconditioners generated using the Hessian decomposition ('HD') and reduced memory Hessian decomposition ('RMHD') algorithms outlined in Figures 5.2 and 5.5, respectively. 107

6.4 The memory ratios R_d and $\hat{R}_d$ corresponding to the preconditioners presented in Table 6.3 for the four model problems summarised in Table 6.1. 108

6.5 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with preconditioning introduced at outer iteration numbers $j = 1, 2, 3, 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The model problem is MP1 where $N_c = 5$. The preconditioner applied is P1B. 110

6.6 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P1A, P1B, P1C, and P1D, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP1 where $N_c = 5$. 112

6.7 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP1 where $N_c = 5$. . . . 116

6.8 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP1 where $N_c = 10$. . . 119

6.9 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP2 where $N_c = 5$. . . . 119

6.10 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP2 where $N_c = 10$. . . 122

6.11 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP3 where $N_c = 5$. . . . 122

6.12 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP3 where $N_c = 10$. . . 125

6.13 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP4 where $N_c = 5$. . . . 125

6.14 The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP4 where $N_c = 10$. . . 128

Chapter 1

Introduction

1.1 Overview of data assimilation in Numerical Weather Prediction

Data assimilation is the process whereby mathematical models characterising the key elements of a dynamical system are combined with observations in order to obtain an estimate of the system state. Applications of data assimilation arise in many scientific and technological fields, notably within the geosciences; see, for example, [3], [38], [67], [108], [115]. Relevant areas of significant interest include meteorology, oceanography, geology, and hydrology.

Data assimilation is of particular importance in the research and study of weather forecasting problems in Numerical Weather Prediction (NWP). The principal aim in NWP is to utilise all available information about the present state of the atmosphere to produce forecasts predicting the atmospheric state at future time points. For an introduction to the topic of atmospheric data assimilation see [30], [62]. Further useful sources of introductory material include [12], [89], [90]. The history and progression of data assimilation in NWP is reviewed briefly in [87]. An overview of the developments in data assimilation at NWP centres is given in [100].

Data assimilation in NWP involves combining observational data with mathematical models representing important atmospheric processes in order to obtain estimates of current and future atmospheric states. The compilation of regional and global weather forecasts is usually undertaken several times a day at NWP centres. The assimilation

process is conducted in real time to ensure that resulting forecasts provide realistic predictions of future atmospheric states. A data assimilation cycle is executed over a specified time period encompassing a given set of observations. This time interval is referred to as the assimilation window. The time period covered in an operational setting typically ranges from several hours to a few days.

The mathematical models representing the key dynamical processes occurring in the atmosphere are a fundamental component of any NWP system. The number of state variables associated with these models is generally of the order of $10^7 - 10^8$. Observational data is gathered from a variety of sources such as aircraft, ships, weather balloons, and satellites. It is usually the case in NWP forecasting problems that the number of available observations is of the order of $10^5 - 10^6$, notably less than the number of model state variables. This issue is addressed by introducing a background estimate of the true atmospheric state. The background state is typically the result of a previous data assimilation cycle.

An element of uncertainty is present in all NWP forecasting problems due to the accumulation of errors throughout the assimilation process. These discrepancies are incurred primarily through the background, observations, and forecasting models as a consequence of limitations in the data assimilation system. The background error characterises the difference between the background and the true atmospheric state. The observational error is attributed to measuring instrument error, or results from the translation of observations into model appropriate form. Model error is the result of the various approximations and discretisations implemented in an attempt to replicate the dynamical atmospheric processes taking place. It is not possible to represent the true atmospheric state exactly in practice.

The production of forecasts in NWP is a complex task that requires the efficient use of supercomputers and specialised data assimilation methods. The improvement in forecast quality in recent years is mainly attributed to computational advancements such as the development of sophisticated atmospheric models and data assimilation techniques, as well as the increase in availability of measuring instruments for capturing observational data. Operational NWP systems depend on data assimilation methods to integrate the many variants of atmospheric data with the forecasting models. The

choice of method is based on a number of factors such as the specific problem under consideration and the computational resources available. Data assimilation techniques are discussed in the framework of NWP in [81], [117]; see also [9], [46].

1.2 The Hessian in variational data assimilation

Variational methods are often employed in NWP; see, for example, [22], [30], [62]. The implementation of variational data assimilation in large-scale problems is discussed in [69]. The variational approach is to formulate the data assimilation problem as a constrained minimisation problem. The aim is to determine an estimate of the initial model state for a given dynamical model by minimising a non-linear cost function subject to a non-linear model constraint. The solution is referred to as the analysis. Variational data assimilation problems can be formulated as optimal control problems; see, for example, [73], [75]. The first variational method implemented operationally is known as three-dimensional variational data assimilation (3D-Var); see [1], [23], [43], [84], [103]. Four-dimensional variational data assimilation (4D-Var) is an extension of 3D-Var that incorporates observations distributed within a specified time interval. Incremental 4D-Var [24] is currently implemented at NWP centres such as the Met Office [104] and the European Centre for Medium Range Forecasts (ECMWF) [64], [85], [102]. Ensemble 4D-Var is a recently developed version of 4D-Var that has been implemented operationally; see, for example, [16], [20], [80]. The development of operational 4D-Var data assimilation systems for coupled atmospheric and oceanic models is currently of interest; see, for example, [39], [110].

The Hessian is the matrix containing the second-order partial derivatives of the associated cost function in variational data assimilation. The Hessian and inverse Hessian are of importance in various aspects of variational data assimilation procedures. An overview of the properties of the Hessian in the wider context of meteorological and oceanographic modelling is given in [118]. One role of the Hessian is as a coefficient matrix in the inner step of a Gauss-Newton procedure in the framework of incremental 4D-Var [24]. The premise is to solve a system of linear equations in an inner loop in order to determine an update for the current estimate of the initial model state.

The linear system is generally solved in practice using a suitable iterative scheme such as the conjugate gradient method [60]. An approximation to the inverse Hessian, if inexpensive to compute, can be used to precondition this system to accelerate the convergence of the iterative method.

The inverse Hessian and inverse square root Hessian also have different statistical applications in variational data assimilation. Specifically, the inverse Hessian is equal to the analysis error covariance matrix in the case of a linear dynamical model provided that certain statistical conditions are satisfied; see, for example, [101], [118]. If the dynamical model is non-linear, then the inverse Hessian can be used as an approximation to the analysis error covariance matrix; see, for example, [44], [74], [101], [118], [119]. That is, information relating to the analysis error covariance matrix can be inferred from the inverse Hessian. For instance, confidence intervals for the components of the analysis vector can be defined by the corresponding diagonal entries of the inverse Hessian. The analysis probability density function is defined by the analysis and the analysis error covariance matrix. Random functions generated using the inverse square root Hessian can be used as 'particles' of the ensemble of initial states; see [33], [34]. These may be useful in ensemble forecasting; see, for example, [79], [120]. An approximation to the inverse square root Hessian can be used to precondition the non-linear minimisation procedure in the framework of the fully non-linear ensemble method [44] or the randomised maximum likelihood method [19] to accelerate convergence.

In practice, due to the large-scale of variational data assimilation problems in NWP, the Hessian cannot be explicitly represented, or stored, in matrix form. The Hessian vector product is usually defined in such cases by the sequential solution of a tangent linear and adjoint problem (see Section 2.8). However, there are no readily available routines for computing the inverse Hessian, and inverse square root Hessian, vector products. A suitable method for generating a compact representation of the inverse Hessian, and inverse square root Hessian, is therefore required. One approach adopted in operational implementations of incremental 4D-Var [24] at the ECMWF is to use estimates of a specified number of the leading eigenvalues, and corresponding eigenvectors, of the Hessian, following the application of the standard control variable transform, to construct a limited-memory approximation to the inverse Hessian; see, for example,

[36]. These eigenpair estimates are computed by means of the Lanczos method [68]; see also [29].

The concept of employing a multilevel strategy to construct an approximation to the inverse Hessian, and inverse square root Hessian, is applicable in variational data assimilation. Multigrid methods are algorithms based on a hierarchy of grids that are typically used to solve partial differential equations. For an introductory overview of multigrid see, for example, [14], [57], [131]. Multigrid methods can also be used to solve optimisation problems governed by partial differential equations; see [11]. An additional application of multigrid techniques is for solving eigenvalue problems; see, for example, [21], [55], [65]. A multigrid method has been applied in the context of variational data assimilation in [31].

1.3 Thesis outline

Background material is presented in Chapters 2 and 3 of this thesis. An overview of four-dimensional variational data assimilation (4D-Var) is given in Chapter 2. The mathematical techniques employed in Chapters 4, 5, and 6 are introduced in Chapter 3.

The first novel concept introduced in this thesis, a multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of any given symmetric positive definite matrix with eigenvalues clustered around unity, is presented in Chapter 4. This algorithm is applied to the Hessian in the framework of incremental 4D-Var [24], with the standard control variable transform implemented, in order to construct an approximation to the inverse Hessian. The accuracy of approximations to the inverse Hessian generated using the multilevel eigenvalue decomposition algorithm is investigated in Chapter 4.

A second key concept, a novel decomposition of the Hessian as the sum of a set of local Hessians, is introduced in the setting of incremental 4D-Var in Chapter 5. Two practical variants of the multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of this Hessian are presented in Chapter 5. The accuracy of approximations to the inverse Hessian generated using the two algorithms proposed is also investigated.

The application considered in Chapter 6 focuses on preconditioning the system of linear equations in the inner step of a Gauss-Newton procedure in incremental 4D-Var with an approximation to the inverse Hessian generated using the three algorithms introduced in Chapters 4 and 5. A numerical study is presented which shows that computational gains can be achieved by applying the preconditioners in practice.

The novel concepts introduced in Chapters 4 and 5 have been published in [15]. A numerical study focusing on the application considered in Chapter 6 is also presented.

Chapter 2

Four-dimensional variational data assimilation

Four-dimensional variational data assimilation (4D-Var) is often employed in Numerical Weather Prediction (NWP); see, for example, [22], [62], [89], [90]. The aim is to determine an estimate of the initial model state for a forecast model that best fits given observational data distributed over a specified time interval. An overview of 4D-Var is given in this chapter.

The standard formulation of 4D-Var is presented in Section 2.1 in the first instance. The tangent linear model (TLM) typically introduced in operational implementations of 4D-Var is defined in Section 2.2. A linear estimate of the solution to the 4D-Var problem is derived in Section 2.3, and the adjoint model used in operational implementations of 4D-Var is discussed in Section 2.4. The Gauss-Newton method, an iterative technique applicable for solving the 4D-Var problem, is presented in Section 2.5; see also, for example, [32], [48], [91]. Specifically, the application of this method for solving the 4D-Var problem is described.

Incremental 4D-Var [24] is an iterative version of 4D-Var currently implemented at many NWP centres; see, for example, [64], [85], [102], [104]. This procedure is outlined in Section 2.6. The inner loop problem in incremental 4D-Var is discussed in Section 2.7. The Hessian is defined in the context of incremental 4D-Var in Section 2.8. A control variable transform usually applied in incremental 4D-Var is discussed in Section 2.9. Finally, a limited-memory decomposition of the Hessian following this

variable transformation is presented in Section 2.10.

2.1 The standard formulation of 4D-Var

The standard formulation of 4D-Var is as a non-linear least squares problem; see, for example, [89], [90]. The premise is to determine an estimate of the initial model state for a given dynamical model by minimising a non-linear cost function subject to a non-linear model constraint. Let $\mathbf{x}_i \in \mathbb{R}^N$ denote the model state vector at time t_i of the assimilation window $[t_0, t_n]$. The evolution of $\mathbf{x}_i$ from time t_i to time t_{i+1} $(i = 0, \ldots, n-1)$ is defined by a non-linear model operator $\mathcal{M}(t_{i+1}, t_i, .) : \mathbb{R}^N \to \mathbb{R}^N$ as follows:

$$\mathbf{x}_{i+1} = \mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i) \quad (i = 0, \ldots, n-1). \tag{2.1}$$

Suppose that the observations $\mathbf{y}_i \in \mathbb{R}^{p_i}$ at time t_i satisfy the equation

$$\mathbf{y}_i = \mathcal{H}_i(\mathbf{x}_i) + \boldsymbol{\delta}_o^i \quad (i = 0, \ldots, n) \tag{2.2}$$

where $\mathcal{H}_i(.) : \mathbb{R}^N \to \mathbb{R}^{p_i}$ is a non-linear observation operator, and $\boldsymbol{\delta}_o^i \in \mathbb{R}^{p_i}$ denotes the observational error incurred at time t_i. The operator $\mathcal{H}_i(.)$ is required in (2.2) to translate $\mathbf{x}_i$ from the model state space $\mathbb{R}^N$ to the observation space $\mathbb{R}^{p_i}$. Note that the number of available observations p_i at time t_i is usually less than N in practice. This issue is addressed by introducing an estimate of $\mathbf{x}_0$ in the form of the background $\mathbf{x}_0^b \in \mathbb{R}^N$. The error in the background is

$$\boldsymbol{\delta}_b = \mathbf{x}_0^b - \mathbf{x}_0^t \tag{2.3}$$

where $\mathbf{x}_0^t$ denotes the true atmospheric state at time t_0. The specification of covariance matrices allows uncertainties in $\mathbf{y}_i$ and $\mathbf{x}_0^b$ to be represented. Specifically, let $R_i \in \mathbb{R}^{p_i \times p_i}$ and $B \in \mathbb{R}^{N \times N}$ denote symmetric positive definite covariance matrices characterising the errors in $\mathbf{y}_i$ and $\mathbf{x}_0^b$, respectively. The importance of B in variational data assimilation is discussed in [7], [8]. The aim of 4D-Var is then to minimise the

cost function

$$\mathcal{J}(\mathbf{x}_0) = \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}_0^b)^T B^{-1}(\mathbf{x}_0 - \mathbf{x}_0^b) + \frac{1}{2}\sum_{i=0}^{n} \left(\mathcal{H}_i(\mathbf{x}_i) - \mathbf{y}_i\right)^T R_i^{-1} \left(\mathcal{H}_i(\mathbf{x}_i) - \mathbf{y}_i\right) \quad (2.4)$$

with respect to $\mathbf{x}_0$ subject to (2.1). The solution to the 4D-Var problem is known as the analysis $\mathbf{x}_0^a$. The error in the analysis is

$$\boldsymbol{\delta}_a = \mathbf{x}_0^a - \mathbf{x}_0^t.$$

It has been shown in [82] that, under certain statistical conditions, $\mathbf{x}_0^a$ is equal to the maximum a posteriori Bayesian estimate of $\mathbf{x}_0$.

A standard assumption in 4D-Var is that $\mathcal{M}(t_{i+1}, t_i, .)$ is perfect. The incorporation of model error in 4D-Var is discussed in [122], [124]. However, model error is not considered in this thesis. Note that, if $n = 0$, then 4D-Var reduces to three-dimensional variational data assimilation (3D-Var); see, for example, [22], [23], [43], [62], [84].

2.2 The tangent linear model

Simplifications are usually introduced in operational implementations of 4D-Var. The standard approach is to linearise the operators $\mathcal{M}(t_{i+1}, t_i, .)$ in (2.1) and $\mathcal{H}_i(.)$ in (2.2) about $\mathbf{x}_i$. This procedure is dependent on the validity of the so-called tangent linear hypothesis; see, for example, [12], [62]. This states that $\mathcal{M}(t_{i+1}, t_i, .)$ and $\mathcal{H}_i(.)$ can be linearised. The Taylor series expansions of $\mathcal{M}(t_{i+1}, t_i, .)$ and $\mathcal{H}_i(.)$ around $\mathbf{x}_i$ for a perturbation $\delta\mathbf{x}_i \in \mathbb{R}^N$ are

$$\mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i + \delta\mathbf{x}_i) = \mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i) + M_{i,i+1}\delta\mathbf{x}_i + O(\delta\mathbf{x}_i^2), \quad (2.5)$$

$$\mathcal{H}_i(\mathbf{x}_i + \delta\mathbf{x}_i) = \mathcal{H}_i(\mathbf{x}_i) + H_i\delta\mathbf{x}_i + O(\delta\mathbf{x}_i^2) \quad (2.6)$$

where

$$M_{i,i+1} = \left.\frac{\partial \mathcal{M}(t_{i+1}, t_i, .)}{\partial \mathbf{x}}\right|_{\mathbf{x}_i}, \quad H_i = \left.\frac{\partial \mathcal{H}_i(.)}{\partial \mathbf{x}}\right|_{\mathbf{x}_i}. \quad (2.7)$$

The matrices $M_{i,i+1}$ and H_i in (2.7) are the Jacobians of $\mathcal{M}(t_{i+1}, t_i, .)$ and $\mathcal{H}_i(.)$, respectively, calculated with respect to $\mathbf{x}_i$. If the tangent linear hypothesis is valid, then

$\mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i + \delta\mathbf{x}_i)$ in (2.5) and $\mathcal{H}_i(\mathbf{x}_i + \delta\mathbf{x}_i)$ in (2.6) can be approximated by neglecting the higher order terms denoted by $O(\delta\mathbf{x}_i^2)$. The premise is therefore to set

$$\mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i + \delta\mathbf{x}_i) = \mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i) + M_{i,i+1}\delta\mathbf{x}_i \tag{2.8}$$

and

$$\mathcal{H}_i(\mathbf{x}_i + \delta\mathbf{x}_i) = \mathcal{H}_i(\mathbf{x}_i) + H_i\delta\mathbf{x}_i.$$

It follows from (2.1) that

$$\mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i + \delta\mathbf{x}_i) = \mathbf{x}_{i+1} + \delta\mathbf{x}_{i+1}. \tag{2.9}$$

Substituting the expressions for $\mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i)$ in (2.1) and $\mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i + \delta\mathbf{x}_i)$ in (2.9) into (2.8) gives

$$\mathbf{x}_{i+1} + \delta\mathbf{x}_{i+1} = \mathbf{x}_{i+1} + M_{i,i+1}\delta\mathbf{x}_i$$

thus

$$\delta\mathbf{x}_{i+1} = M_{i,i+1}\delta\mathbf{x}_i.$$

The matrix $M_{i,i+1}$ in (2.7) is referred to as the tangent linear model (TLM). Applying the chain rule

$$M_{0,i} = M_{i-1,i}M_{i-2,i-1}\ldots M_{0,1}. \tag{2.10}$$

That is, $M_{0,i}$ is equal to the product of i intermediate matrices pertaining to each time step of the assimilation window $[t_0, t_i]$. In practice, the tangent linear model code can usually be derived from the non-linear model code by applying automatic differentiation techniques; see, for example, [53].

2.3 Linear analysis estimate

In general, explicit solutions to the 4D-Var problem presented in Section 2.1 cannot be determined. However, it is possible to obtain a linear estimate of $\mathbf{x}_0^a$; see, for example, [90]. This derivation is based on the assumption that the difference between $\mathbf{x}_0^a$ and $\mathbf{x}_0^b$

is a linear combination of the innovation vectors

$$\tilde{\mathbf{d}}_\mathbf{i} = \mathbf{y}_i - \mathcal{H}_i(\mathbf{x}_i^b) \quad (i = 0, \ldots, n).$$

The premise is to linearise $\mathcal{J}(\mathbf{x}_0)$ in (2.4) about the non-linear background trajectory $\mathbf{x}_i^b$ $(i = 0, \ldots, n)$ where

$$\mathbf{x}_{i+1}^b = \mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i^b) \quad (i = 0, \ldots, n-1). \tag{2.11}$$

This involves linearising the operators $\mathcal{M}(t_{i+1}, t_i, .)$ in (2.1) and $\mathcal{H}_i(.)$ in (2.2) about $\mathbf{x}_i^b$ by defining

$$M_{i,i+1}^b = \left.\frac{\partial \mathcal{M}\,(t_{i+1}, t_i, .)}{\partial \mathbf{x}}\right|_{\mathbf{x}_i^b}, \quad H_i^b = \left.\frac{\partial \mathcal{H}_i(.)}{\partial \mathbf{x}}\right|_{\mathbf{x}_i^b}. \tag{2.12}$$

The matrices $M_{i,i+1}^b$ and H_i^b in (2.12) are the Jacobians of $\mathcal{M}(t_{i+1}, t_i, .)$ and $\mathcal{H}_i(.)$, respectively, calculated with respect to $\mathbf{x}_i^b$. It follows from (2.4) that the resulting linearised cost function is

$$\tilde{\mathcal{J}}(\delta\mathbf{x}_0) = \frac{1}{2}\delta\mathbf{x}_0^T B^{-1}\delta\mathbf{x}_0 + \frac{1}{2}\sum_{i=0}^{n}(H_i^b M_{0,i}^b \delta\mathbf{x}_0 - \tilde{\mathbf{d}}_\mathbf{i})^T R_i^{-1}(H_i^b M_{0,i}^b \delta\mathbf{x}_0 - \tilde{\mathbf{d}}_\mathbf{i}) \tag{2.13}$$

where

$$\delta\mathbf{x}_0 = \mathbf{x}_0 - \mathbf{x}_0^b. \tag{2.14}$$

The gradient of $\tilde{\mathcal{J}}(\delta\mathbf{x}_0)$ in (2.13) calculated with respect to $\delta\mathbf{x}_0$ is

$$\nabla\tilde{\mathcal{J}}(\delta\mathbf{x}_0) = (B^{-1} + \overline{H}^T\overline{R}^{-1}\overline{H})\delta\mathbf{x}_0 - \overline{H}^T\overline{R}^{-1}\overline{\mathbf{d}} \tag{2.15}$$

where

$$\overline{H} = \begin{pmatrix} H_0^b \\ H_1^b M_{0,1}^b \\ \vdots \\ H_n^b M_{0,n}^b \end{pmatrix}, \quad \overline{R} = \begin{pmatrix} R_0 & 0 & \ldots & 0 \\ 0 & R_1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & R_n \end{pmatrix}, \quad \overline{\mathbf{d}} = \begin{pmatrix} \tilde{\mathbf{d}}_0 \\ \tilde{\mathbf{d}}_1 \\ \vdots \\ \tilde{\mathbf{d}}_n \end{pmatrix}. \tag{2.16}$$

Setting $\nabla\tilde{\mathcal{J}}(\delta\mathbf{x}_0)$ in (2.15) equal to zero and solving for $\delta\mathbf{x}_0$ gives

$$\delta\mathbf{x}_0 = (B^{-1} + \overline{H}^T\overline{R}^{-1}\overline{H})^{-1}\overline{H}^T\overline{R}^{-1}\overline{\mathbf{d}}. \tag{2.17}$$

Let $\overline{\mathbf{x}}_0^a \in \mathbb{R}^N$ denote the linear estimate of $\mathbf{x}_0^a$. This is determined by replacing $\delta\mathbf{x}_0$ in (2.17) with the definition in (2.14), solving for $\mathbf{x}_0$, and setting $\mathbf{x}_0 = \overline{\mathbf{x}}_0^a$. Specifically,

$$\overline{\mathbf{x}}_0^a = \mathbf{x}_0^b + K\overline{\mathbf{d}} \tag{2.18}$$

where

$$K = (B^{-1} + \overline{H}^T\overline{R}^{-1}\overline{H})^{-1}\overline{H}^T\overline{R}^{-1} \equiv B\overline{H}^T(\overline{R} + \overline{H}B\overline{H}^T)^{-1}. \tag{2.19}$$

The matrix K in (2.19) is referred to as the gain or weight matrix of $\overline{\mathbf{x}}_0^a$ in (2.18). If certain statistical conditions are satisfied, then $\overline{\mathbf{x}}_0^a$ in (2.18)-(2.19) is the best linear unbiased estimate (BLUE) of $\mathbf{x}_0^t$; see, for example, [90]. The error in $\overline{\mathbf{x}}_0^a$ is

$$\overline{\boldsymbol{\delta}}_a = \overline{\mathbf{x}}_0^a - \mathbf{x}_0^t.$$

The analysis error covariance matrix associated with $\overline{\mathbf{x}}_0^a$ in this case is

$$\overline{A} = (I_N - K\overline{H})B \tag{2.20}$$

where I_N is the $N \times N$ identity matrix. The matrix $\overline{A}$ in (2.20) characterises the error in $\overline{\mathbf{x}}_0^a$.

It follows from (2.15) that the Hessian of $\tilde{\mathcal{J}}(\delta\mathbf{x}_0)$ in (2.13) calculated with respect to $\delta\mathbf{x}_0$ is

$$\nabla^2\tilde{\mathcal{J}}(\delta\mathbf{x}_0) = B^{-1} + \overline{H}^T\overline{R}^{-1}\overline{H} \tag{2.21}$$

where $\overline{H}$ and $\overline{R}$ are defined in (2.16). Provided that B is non-singular, then $\nabla^2\tilde{\mathcal{J}}(\delta\mathbf{x}_0)$ in (2.21) is symmetric positive definite. The inverse of $\nabla^2\tilde{\mathcal{J}}(\delta\mathbf{x}_0)$ is equal to $\overline{A}$ in (2.20); see [101].

2.4 The adjoint model

An iterative approach is often employed in operational implementations of 4D-Var. The procedure in such cases is to reformulate the 4D-Var problem presented in Section 2.1 as an unconstrained optimisation problem. This is solved in practice using an appropriate optimisation method; see [35]. Applying an optimisation method in 4D-Var involves computing the gradient of $\mathcal{J}(\mathbf{x}_0)$ in (2.4). This is facilitated by the method of Lagrange; see, for example, [89], [90]. The new requirement is that a set of adjoint equations must be satisfied in addition to (2.1).

Suppose that $M_{i,i+1}$ and H_i in (2.7) are defined. The adjoints of $M_{i,i+1}$ and H_i are included in the adjoint equations. These adjoints are the transpose matrices $M_{i,i+1}^T$ and H_i^T, respectively; see, for example, [62]. The matrix $M_{i,i+1}^T$ is referred to as the adjoint model. Taking the transpose of $M_{0,i}$ in (2.10) gives

$$M_{0,i}^T = M_{0,1}^T \dots M_{i-2,i-1}^T M_{i-1,i}^T.$$

Hence $M_{0,i}^T$ is equal to the product of i intermediate transpose matrices pertaining to each time step of the assimilation window $[t_0, t_i]$. The adjoint equations to be solved are

$$\boldsymbol{\lambda}_{n+1} = 0, \tag{2.22}$$

$$\boldsymbol{\lambda}_i = M_{i,i+1}^T \boldsymbol{\lambda}_{i+1} + H_i^T R_i^{-1} \left(\mathbf{y}_i - \mathcal{H}_i(\mathbf{x}_i)\right) \quad (i = n, \dots, 0) \tag{2.23}$$

where $\boldsymbol{\lambda}_i \in \mathbb{R}^N$ $(i = 0, \dots, n)$ denote the adjoint variables. A measure of the sensitivity of $\mathcal{J}(\mathbf{x}_0)$ in (2.4) to variations in $\mathbf{x}_i$ is provided by $\boldsymbol{\lambda}_i$.

The gradient of $\mathcal{J}(\mathbf{x}_0)$ in (2.4) is computed by solving (2.22)-(2.23). Note that the gradient of $\mathcal{J}(\mathbf{x}_0)$ calculated with respect to $\mathbf{x}_0$ is

$$\begin{aligned}
\nabla \mathcal{J}(\mathbf{x}_0) &= B^{-1}(\mathbf{x}_0 - \mathbf{x}_0^b) + \sum_{i=0}^{n} M_{0,i}^T H_i^T R_i^{-1} (\mathcal{H}_i(\mathbf{x}_i) - \mathbf{y}_i) \\
&= B^{-1}(\mathbf{x}_0 - \mathbf{x}_0^b) + H_0^T R_0^{-1} (\mathcal{H}_0(\mathbf{x}_0) - \mathbf{y}_0) + \sum_{i=1}^{n} M_{0,i}^T H_i^T R_i^{-1} (\mathcal{H}_i(\mathbf{x}_i) - \mathbf{y}_i) \\
&= B^{-1}(\mathbf{x}_0 - \mathbf{x}_0^b) - \boldsymbol{\lambda}_0
\end{aligned} \tag{2.24}$$

where

$$\boldsymbol{\lambda}_0 = M_{0,1}^T \boldsymbol{\lambda}_1 + H_0^T R_0^{-1}(\mathbf{y}_0 - \mathcal{H}_0(\mathbf{x}_0)). \tag{2.25}$$

The aim is determine $\mathbf{x}_0$ such that $\nabla \mathcal{J}(\mathbf{x}_0) = 0$. If this condition is not satisfied, then the computed gradient provides a search direction for updating the current estimate of $\mathbf{x}_0$. In practice, the adjoint model code can be derived directly from the tangent linear model code; see [47].

2.5 The Gauss-Newton method

The Gauss-Newton method is an adaptation of Newton's method for solving non-linear least squares problems; see, for example, [32], [48], [91]. This iterative technique is applicable for solving the 4D-Var problem presented in Section 2.1. The Gauss-Newton method is applied in the numerical studies presented in Chapter 6.

Recall that the aim of 4D-Var is to minimise $\mathcal{J}(\mathbf{x}_0)$ in (2.4) with respect to $\mathbf{x}_0$ subject to (2.1). Solving this minimisation problem using the Gauss-Newton method is now described. The starting point is to observe that the 4D-Var problem can be formulated as follows:

$$\min_{\mathbf{x}_0 \in \mathbb{R}^N} \mathcal{J}(\mathbf{x}_0) \;=\; \frac{1}{2} \left\| \mathbf{f}(\mathbf{x}_0) \right\|_2^2 \;\equiv\; \frac{1}{2} \mathbf{f}(\mathbf{x}_0)^T \mathbf{f}(\mathbf{x}_0) \tag{2.26}$$

where

$$\mathbf{f}(\mathbf{x}_0) = \begin{pmatrix} B^{-\frac{1}{2}}(\mathbf{x}_0 - \mathbf{x}_0^b) \\ R_0^{-\frac{1}{2}}(\mathcal{H}_0(\mathbf{x}_0) - \mathbf{y}_0) \\ \vdots \\ R_n^{-\frac{1}{2}}(\mathcal{H}_n(\mathbf{x}_n) - \mathbf{y}_n) \end{pmatrix} \tag{2.27}$$

and $\|.\|_2$ represents the ℓ_2-norm. The premise is to approximate (2.26) by solving a sequence of linearised least squares problems. This is conducted in the framework of an inner-outer loop procedure.

Suppose that $M_{i,i+1}$ and H_i in (2.7) are defined. The gradient of $\mathcal{J}(\mathbf{x}_0)$ in (2.4) calculated with respect to $\mathbf{x}_0$ is $\nabla \mathcal{J}(\mathbf{x}_0)$ in (2.24)-(2.25). Note that $\nabla \mathcal{J}(\mathbf{x}_0)$ can be

defined in terms of $\mathbf{f}(\mathbf{x}_0)$ in (2.27) as follows:

$$\nabla \mathcal{J}(\mathbf{x}_0) = J(\mathbf{x}_0)^T \mathbf{f}(\mathbf{x}_0) \tag{2.28}$$

where

$$J(\mathbf{x}_0) = \begin{pmatrix} B^{-\frac{1}{2}} \\ R_0^{-\frac{1}{2}} H_0 \\ R_1^{-\frac{1}{2}} H_1 M_{0,1} \\ \vdots \\ R_n^{-\frac{1}{2}} H_n M_{0,n} \end{pmatrix}. \tag{2.29}$$

The matrix $J(\mathbf{x}_0)$ in (2.29) is the Jacobian of $\mathbf{f}(\mathbf{x}_0)$ in (2.27) calculated with respect to $\mathbf{x}_0$. It follows from (2.28) that the Hessian of $\mathcal{J}(\mathbf{x}_0)$ calculated with respect to $\mathbf{x}_0$ is

$$\nabla^2 \mathcal{J}(\mathbf{x}_0) = J(\mathbf{x}_0)^T J(\mathbf{x}_0) + \sum_{i=1}^{n+2} \mathbf{f}_i(\mathbf{x}_0) \nabla^2 \mathbf{f}_i(\mathbf{x}_0) \tag{2.30}$$

where $\mathbf{f}_i(\mathbf{x}_0)$ denotes the ith entry of $\mathbf{f}(\mathbf{x}_0)$. Computing $\nabla^2 \mathcal{J}(\mathbf{x}_0)$ in (2.30) is not usually feasible in practice. The Gauss-Newton method is applicable since this technique does not require $\nabla^2 \mathcal{J}(\mathbf{x}_0)$ to be computed directly. The idea is to approximate $\nabla^2 \mathcal{J}(\mathbf{x}_0)$ by setting

$$\nabla^2 \mathcal{J}(\mathbf{x}_0) = J(\mathbf{x}_0)^T J(\mathbf{x}_0).$$

The steps involved in solving (2.26)-(2.27) using the Gauss-Newton method are outlined in Figure 2.1.

Solving (2.31) directly is not always practical. The solution $\delta\hat{\mathbf{x}}_0^{(i)}$ to (2.31) is determined in such cases by solving the following linearised least squares problem with respect to $\delta\hat{\mathbf{x}}_0^{(i)}$ in an inner loop:

$$\min_{\delta\hat{\mathbf{x}}_0^{(i)} \in \mathbb{R}^N} \frac{1}{2} \| J(\mathbf{x}_0^{(i)}) \delta\hat{\mathbf{x}}_0^{(i)} + \mathbf{f}(\mathbf{x}_0^{(i)}) \|_2^2. \tag{2.32}$$

Theoretical results pertaining to the convergence of the Gauss-Newton method in the context of variational data assimilation are presented in [51].

1. Define an initial guess $\mathbf{x}_0^{(1)}$

2. Perform outer iteration for $i = 1, 2, \ldots$ as required. Determine an increment $\delta\hat{\mathbf{x}}_0^{(i)}$ by solving the following system in an inner loop:

$$J(\mathbf{x}_0^{(i)})^T J(\mathbf{x}_0^{(i)})\delta\hat{\mathbf{x}}_0^{(i)} = -J(\mathbf{x}_0^{(i)})^T \mathbf{f}(\mathbf{x}_0^{(i)}). \tag{2.31}$$

3. Update the current guess $\mathbf{x}_0^{(i)}$ using

$$\mathbf{x}_0^{(i+1)} = \mathbf{x}_0^{(i)} + \delta\hat{\mathbf{x}}_0^{(i)}.$$

Figure 2.1: Outline of the steps involved in solving (2.26)-(2.27) using the Gauss-Newton method.

2.6 The incremental formulation of 4D-Var

Incremental 4D-Var [24] is an iterative version of 4D-Var. This is currently implemented at NWP centres such as the Met Office [104], and the European Centre for Medium Range Forecasts (ECMWF) [64], [85], [102]. The premise is to obtain an estimate of the solution to the 4D-Var problem presented in Section 2.1, namely, the analysis $\mathbf{x}_0^a$, by solving a sequence of simpler linearised problems. This is conducted in the framework of an inner-outer loop routine. The main steps of incremental 4D-Var are outlined in Figure 2.2. Note that the matrices $\tilde{M}_{i,i+1}$ and $\tilde{H}_i$ in (2.36) are the Jacobians of $\mathcal{M}(t_{i+1}, t_i, .)$ in (2.1) and $\mathcal{H}_i(.)$ in (2.2), respectively, calculated with respect to $\mathbf{x}_i^{(j)}$.

Incremental 4D-Var is restricted by computational limitations and time constraints at NWP centres. The number of inner and outer loop iterations to perform is chosen carefully in practice. Operational implementations typically involve executing a small number of outer loop iterations. The validity of the tangent linear model (TLM) $\tilde{M}_{i,i+1}$ in (2.36) is discussed in [121]. The outer loop convergence of incremental 4D-Var based on the ECMWF setup has been considered in [123]. Incremental 4D-Var convergence when the TLM is approximated has been investigated in [70]. An advantage of incremental 4D-Var is that the inner loop problem (2.34)-(2.35) can be solved in a lower dimensional space; see [72].

1. Perform outer iteration for $j = 1, 2, \ldots$ as required. Define an initial guess $\mathbf{x}_0^{(j)}$ at time t_0. If $j = 1$, then set $\mathbf{x}_0^{(1)} = \mathbf{x}_0^b$.

2. Compute $\mathbf{x}_i^{(j)}$ at time t_i $(i = 1, \ldots, n)$ using

$$\mathbf{x}_{i+1}^{(j)} = \mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i^{(j)}) \quad (i = 0, \ldots, n-1) \tag{2.33}$$

   and calculate the innovation vectors

$$\mathbf{d}_i^{(j)} = \mathbf{y}_i - \mathcal{H}_i(\mathbf{x}_i^{(j)}) \quad (i = 0, \ldots, n).$$

3. Minimise the following linearised cost function with respect to $\delta\mathbf{x}_0^{(j)}$ in an inner loop:

$$\begin{aligned} \hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)}) &= \frac{1}{2}\left\{\delta\mathbf{x}_0^{(j)} - (\mathbf{x}_0^b - \mathbf{x}_0^{(j)})\right\}^T B^{-1}\left\{\delta\mathbf{x}_0^{(j)} - (\mathbf{x}_0^b - \mathbf{x}_0^{(j)})\right\} \\ &\quad + \frac{1}{2}\sum_{i=0}^{n}(\tilde{H}_i\delta\mathbf{x}_i^{(j)} - \mathbf{d}_i^{(j)})^T R_i^{-1}(\tilde{H}_i\delta\mathbf{x}_i^{(j)} - \mathbf{d}_i^{(j)}) \end{aligned} \tag{2.34}$$

   subject to

$$\delta\mathbf{x}_i^{(j)} = \tilde{M}_{0,i}\delta\mathbf{x}_0^{(j)} \quad (i = 0, \ldots, n) \tag{2.35}$$

   where

$$\tilde{M}_{i,i+1} = \left.\frac{\partial\mathcal{M}\,(t_i, t_{i+1}, .)}{\partial\mathbf{x}}\right|_{\mathbf{x}_i^{(j)}}, \quad \tilde{H}_i = \left.\frac{\partial\mathcal{H}_i(.)}{\partial\mathbf{x}}\right|_{\mathbf{x}_i^{(j)}}. \tag{2.36}$$

4. Update the current guess $\mathbf{x}_0^{(j)}$ using

$$\mathbf{x}_0^{(j+1)} = \mathbf{x}_0^{(j)} + \delta\mathbf{x}_0^{(j)}.$$

5. Repeat process until a prescribed convergence criterion has been satisfied, or a specified number of outer iterations have been executed. The analysis estimate is given by $\hat{\mathbf{x}}_0^a = \mathbf{x}_0^{(m)}$, where m is the total number of outer iterations performed.

Figure 2.2: Outline of the main steps of incremental 4D-Var.

2.7 The inner loop problem

The inner loop problem (2.34)-(2.35) in incremental 4D-Var is typically solved using a suitable optimisation method; see [35]. The choice of method is usually limited to techniques that require the first, but not the second, derivative of $\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ in (2.34) to be computed. The gradient of $\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ calculated with respect to $\delta\mathbf{x}_0^{(j)}$ is

$$\nabla\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)}) = (B^{-1} + \hat{H}^T\hat{R}^{-1}\hat{H})\delta\mathbf{x}_0^{(j)} - B^{-1}(\mathbf{x}_0^b - \mathbf{x}_0^{(j)}) - \hat{H}^T\hat{R}^{-1}\hat{\mathbf{d}} \tag{2.37}$$

where

$$\hat{H} = \begin{pmatrix} \tilde{H}_0 \\ \tilde{H}_1\tilde{M}_{0,1} \\ \vdots \\ \tilde{H}_n\tilde{M}_{0,n} \end{pmatrix}, \quad \hat{R} = \begin{pmatrix} R_0 & 0 & \dots & 0 \\ 0 & R_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R_n \end{pmatrix}, \quad \hat{\mathbf{d}} = \begin{pmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_n \end{pmatrix}. \tag{2.38}$$

Setting $\nabla\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ in (2.37) equal to zero and rearranging terms gives

$$(B^{-1} + \hat{H}^T\hat{R}^{-1}\hat{H})\delta\mathbf{x}_0^{(j)} = B^{-1}(\mathbf{x}_0^b - \mathbf{x}_0^{(j)}) + \hat{H}^T\hat{R}^{-1}\hat{\mathbf{d}}. \tag{2.39}$$

The solution $\delta\mathbf{x}_0^{(j)}$ to (2.39) is the minimiser of $\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ in (2.34). An appropriate iterative method is generally used to solve (2.39). At the start of an outer iteration, it is necessary to perform n forward integrations of the non-linear model $\mathcal{M}(t_{i+1}, t_i, .)$ in (2.33) to calculate the model states $\mathbf{x}_i^{(j)}$ $(i = 1, \dots, n)$. The tangent linear model (TLM) $\tilde{M}_{i,i+1}$ in (2.36) is used to compute $\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$. The adjoint model $\tilde{M}_{i,i+1}^T$ is involved in calculating $\nabla\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ in (2.37). These procedures are computationally demanding in practice. The number of evaluations of $\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ and $\nabla\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ permitted is therefore usually restricted.

It has been shown in [70] that solving the 4D-Var problem using the Gauss-Newton method (as described in Section 2.5) is equivalent to applying incremental 4D-Var outlined in Figure 2.2. A stopping criterion for the inner loop in incremental 4D-Var is described in [71].

2.8 The Hessian in incremental 4D-Var

The Hessian is of importance in incremental 4D-Var; see, for example, [54]. It follows from (2.37) that the Hessian of $\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ in (2.34) calculated with respect to $\delta\mathbf{x}_0^{(j)}$ is

$$\nabla^2\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)}) = B^{-1} + \hat{H}^T\hat{R}^{-1}\hat{H} \tag{2.40}$$

where $\hat{H}$ and $\hat{R}$ are defined in (2.38). To simplify notation, henceforth we will denote $\nabla^2\hat{\mathcal{J}}(\delta\mathbf{x}_0^{(j)})$ in (2.40) by the real $N \times N$ matrix F. Provided that B is non-singular, then F is symmetric positive definite. In practice, due to memory constraints, the Hessian cannot be explicitly represented in matrix form. The Hessian vector product is usually evaluated by executing the tangent linear model (TLM) $\tilde{M}_{i,i+1}$ in (2.36) and the associated adjoint model $\tilde{M}_{i,i+1}^T$. This procedure is computationally demanding.

The condition number of the Hessian provides a measure of the conditioning of the inner loop problem (2.34)-(2.35). A large condition number indicates that the problem is ill-conditioned. This signifies that the solution $\delta\mathbf{x}_0^{(j)}$ to (2.34)-(2.35), or equivalently (2.39), is sensitive to small perturbations in the input data. If the condition number of the Hessian is large, then the convergence rate of the iterative method used to solve (2.39) may be slow. It has been highlighted in [83] that the Hessian is usually ill-conditioned in practice. A theoretical examination of the conditioning of the variational data assimilation problem is presented in [54]; see also [116]. If certain statistical conditions are satisfied, then the inverse Hessian is equal to the analysis error covariance matrix; see [24], [101].

2.9 The control variable transform

A control variable transform is usually applied in incremental 4D-Var. This typically involves $B^{1/2}$, that is, the symmetric square root of the background error covariance matrix B; see, for example, [8], [83], [90]. The premise is to redefine the inner loop problem (2.34)-(2.35) in terms of the new variables

$$\delta\mathbf{z}_i^{(j)} = B^{-1/2}\delta\mathbf{x}_i^{(j)} \quad (i = 0, \ldots, n) \tag{2.41}$$

where

$$\mathbf{z}_i^{(j)} = B^{-1/2}\mathbf{x}_i^{(j)} \quad (i = 0, \ldots, n) \tag{2.42}$$

and

$$\mathbf{z}_0^b = B^{-1/2}\mathbf{x}_0^b. \tag{2.43}$$

Substituting the definitions of $\delta\mathbf{x}_i^{(j)}$, $\mathbf{x}_i^{(j)}$, and $\mathbf{x}_0^b$ given by (2.41)-(2.43) into (2.34)-(2.35) results in the transformed inner loop problem

$$\begin{aligned}\hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)}) &= \frac{1}{2}\left\{\delta\mathbf{z}_0^{(j)} - (\mathbf{z}_0^b - \mathbf{z}_0^{(j)})\right\}^T \left\{\delta\mathbf{z}_0^{(j)} - (\mathbf{z}_0^b - \mathbf{z}_0^{(j)})\right\} \\ &\quad + \frac{1}{2}\sum_{i=0}^{n} (\tilde{H}_i B^{1/2}\delta\mathbf{z}_i^{(j)} - \mathbf{d}_i^{(j)})^T R_i^{-1} (\tilde{H}_i B^{1/2}\delta\mathbf{z}_i^{(j)} - \mathbf{d}_i^{(j)}),\end{aligned} \tag{2.44}$$

$$\delta\mathbf{z}_i^{(j)} = B^{-1/2}\delta\mathbf{x}_i^{(j)} = B^{-1/2}\tilde{M}_{0,i}\delta\mathbf{x}_0^{(j)} \quad (i = 0, \ldots, n). \tag{2.45}$$

The aim now is to minimise (2.44) with respect to $\delta\mathbf{z}_0^{(j)}$ subject to (2.45). Applying the control variable transform (2.41)-(2.43) in incremental 4D-Var removes the explicit representation of the matrix B in the inner loop cost function. In practice, due to memory constraints, the matrix B cannot be explicitly represented in matrix form. It is instead defined implicitly in (2.44) through (2.41)-(2.43). The updated steps of incremental 4D-Var with the control variable transform (2.41)-(2.43) applied are outlined in Figure 2.3.

The gradient of $\hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)})$ in (2.46) calculated with respect to $\delta\mathbf{z}_0^{(j)}$ is

$$\nabla\hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)}) = (I_N + B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{H}B^{1/2})\delta\mathbf{z}_0^{(j)} - (\mathbf{z}_0^b - \mathbf{z}_0^{(j)}) - B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{\mathbf{d}} \tag{2.48}$$

where $\hat{H}$, $\hat{R}$ and $\hat{\mathbf{d}}$ are defined in (2.38). Setting $\nabla\hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)})$ in (2.48) equal to zero and rearranging terms gives

$$(I_N + B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{H}B^{1/2})\delta\mathbf{z}_0^{(j)} = \mathbf{z}_0^b - \mathbf{z}_0^{(j)} + B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{\mathbf{d}}. \tag{2.49}$$

The solution $\delta\mathbf{z}_0^{(j)}$ to (2.49) is the minimiser of $\hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)})$ in (2.46). It follows from (2.48)

1. Perform outer iteration for $j = 1, 2, \ldots$ as required. Define an initial guess $\mathbf{x}_0^{(j)}$ at time t_0. If $j = 1$, then set $\mathbf{x}_0^{(1)} = \mathbf{x}_0^b$.

2. Compute $\mathbf{x}_i^{(j)}$ at time t_i $(i = 1, \ldots, n)$ using

$$\mathbf{x}_{i+1}^{(j)} = \mathcal{M}(t_{i+1}, t_i, \mathbf{x}_i^{(j)}) \quad (i = 0, \ldots, n-1)$$

and calculate the innovation vectors

$$\mathbf{d}_i^{(j)} = \mathbf{y}_i^0 - \mathcal{H}_i(\mathbf{x}_i^{(j)}) \quad (i = 0, \ldots, n).$$

3. Minimise the following cost function with respect to $\delta\mathbf{z}_0^{(j)}$ in an inner loop:

$$\begin{aligned} \hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)}) &= \frac{1}{2}\left\{\delta\mathbf{z}_0^{(j)} - (\mathbf{z}_0^b - \mathbf{z}_0^{(j)})\right\}^T \left\{\delta\mathbf{z}_0^{(j)} - (\mathbf{z}_0^b - \mathbf{z}_0^{(j)})\right\} \\ &\quad + \frac{1}{2}\sum_{i=0}^{n}\left\{\tilde{H}_i B^{1/2}\delta\mathbf{z}_i^{(j)} - \mathbf{d}_i^{(j)}\right\}^T R_i^{-1}\left\{\tilde{H}_i B^{1/2}\delta\mathbf{z}_i^{(j)} - \mathbf{d}_i^{(j)}\right\} \end{aligned} \tag{2.46}$$

subject to

$$\delta\mathbf{z}_i^{(j)} = B^{-1/2}\delta\mathbf{x}_i^{(j)} = B^{-1/2}\tilde{M}_{0,i}\delta\mathbf{x}_0^{(j)} \quad (i = 0, \ldots, n) \tag{2.47}$$

where

$$\mathbf{z}_0^b = B^{-1/2}\mathbf{x}_0^b,$$
$$\mathbf{z}_i^{(j)} = B^{-1/2}\mathbf{x}_i^{(j)} \quad (i = 0, \ldots, n),$$

and

$$\tilde{M}_{i,i+1} = \frac{\partial\mathcal{M}\,(t_i, t_{i+1}, .)}{\partial\mathbf{x}}\,|_{\mathbf{x}_i^{(j)}}, \quad \tilde{H}_i = \frac{\partial\mathcal{H}_i(.)}{\partial\mathbf{x}}\,|_{\mathbf{x}_i^{(j)}}\,.$$

4. Reverse the control variable transform to recover the increment

$$\delta\mathbf{x}_0^{(j)} = B^{1/2}\delta\mathbf{z}_0^{(j)}.$$

5. Update the current guess $\mathbf{x}_0^{(j)}$ using

$$\mathbf{x}_0^{(j+1)} = \mathbf{x}_0^{(j)} + \delta\mathbf{x}_0^{(j)}.$$

6. Repeat process until a prescribed convergence criterion has been satisfied, or a specified number of outer iterations have been executed. The analysis estimate is given by $\hat{\mathbf{x}}_0^a = \mathbf{x}_0^{(m)}$, where m is the total number of outer iterations performed.

Figure 2.3: Outline of the updated steps of incremental 4D-Var with the control variable transform (2.41)-(2.43) applied.

that the Hessian of $\hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)})$ in (2.46) calculated with respect to $\delta\mathbf{z}_0^{(j)}$ is

$$C \equiv \nabla^2\hat{\mathcal{J}}(\delta\mathbf{z}_0^{(j)}) = I_N + B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{H}B^{1/2} \tag{2.50}$$

where $\hat{H}$ and $\hat{R}$ are defined in (2.38). Note that the matrix $B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{H}B^{1/2}$ in (2.50) is positive semi-definite. However, this matrix does not have full rank since the number of observations is usually less than N in practice; see [54]. Hence C is symmetric positive definite. In particular, the eigenvalues of C are greater than or equal to one. The matrix C is relevant to the numerical studies presented in Chapters 4, 5, and 6.

2.10 Limited-memory approximation to the Hessian

Applying the control variable transform discussed in Section 2.9 has been shown to improve the conditioning of the inner loop problem in incremental 4D-Var; see [83]. The application of this control variable transform is referred to as first-level preconditioning. However, further preconditioning is usually necessary to improve the conditioning of the transformed inner loop problem (2.46)-(2.47); see [36]. This generally involves the Hessian C in (2.50). One approach adopted at the ECMWF is to use estimates of a specified number of the leading eigenvalues, and corresponding eigenvectors, of C to construct a limited-memory approximation to C^{-1}; see [36]. These eigenpair estimates are computed by means of the Lanczos method [68]; see also [29]. Specifically, for any $\gamma \in \mathbb{R}$,

$$C^\gamma \simeq I_N + \sum_{i=1}^{r}(\lambda_i^\gamma - 1)\mathbf{u}_i\mathbf{u}_i^T \tag{2.51}$$

where $\{\lambda_i, \mathbf{u}_i\}$ $(i = 1, \ldots, r)$ denote estimates of r eigenpairs of C, with $r \ll N$. A limited-memory approximation to C^{-1} of the form defined in (2.51) is used to precondition the minimisation of the transformed inner loop problem (2.46)-(2.47) in the ECMWF operational implementation of incremental 4D-Var; see [36]. It has been shown in [125] that this preconditioner belongs to a class of limited-memory preconditioners that are applicable in incremental 4D-Var. The limited-memory approximation to C^γ defined in (2.51) is relevant to the multilevel eigenvalue decomposition algorithm introduced in Chapter 4.

Chapter 3

Background methods

An introductory overview of the mathematical techniques employed in Chapters 4, 5, and 6 is given in this chapter.

The methods discussed initially are applicable for solving the eigenvalue problem

$$M\mathbf{z} = \mu\mathbf{z} \tag{3.1}$$

where $M \in \mathbb{R}^{N\times N}$ is symmetric, $\mathbf{z} \in \mathbb{R}^N$, and $\mu \in \mathbb{R}$. The symmetric eigenvalue problem is discussed in detail in [97]. Solving large-scale eigenvalue problems requires specialised techniques; see, for example, [18], [50], [107], [112], [114], [129], [132]. The Lanczos method [68] is particularly relevant if the coefficient matrix M in (3.1) is sparse. This is described in Section 3.1. The key idea is to first simplify (3.1) by reducing M to tridiagonal form. Estimates of the eigenvalues of M are computed on termination of the Lanczos procedure using an appropriate technique such as the QR method [40], [41], [66] (summarised in Section 3.2). The corresponding eigenvectors, if required, can also be computed. However, numerical issues limit the application of the Lanczos method in practice. The implicitly restarted Lanczos method [111] is a variant of the Lanczos method that is suitable for solving large-scale eigenvalue problems of the form (3.1). This method, discussed in Section 3.3, is applied in the numerical studies presented in Chapters 4, 5, and 6.

The final two methods discussed in this chapter are relevant for solving systems of

linear equations of the form

$$K\mathbf{q} = \mathbf{f} \tag{3.2}$$

where $K \in \mathbb{R}^{N\times N}$ is symmetric positive definite, $\mathbf{q} \in \mathbb{R}^N$, and $\mathbf{f} \in \mathbb{R}^N$. For an introduction to solving systems of linear equations using iterative methods see, for example, [5], [56], [106]. The conjugate gradient method [60] is applicable for solving (3.2). This is described in Section 3.4. An overview of conjugate direction and gradient methods in optimisation is given in [59]. The conjugate gradient method can be derived from the Lanczos method; see, for example, [50]. A preconditioner is usually applied to (3.2) in practice to accelerate the convergence of the conjugate gradient method. Preconditioned iterative methods for solving large, sparse systems of linear equations are surveyed in [4], [10], [13]. The preconditioned version of the conjugate gradient method is described in Section 3.5. Note that the preconditioned conjugate gradient method is applied in the numerical studies presented in Chapter 6.

3.1 The Lanczos method

The Lanczos method [68] is a technique for reducing a Hermitian, or in the real case symmetric, matrix to tridiagonal form. The equivalent procedure in the non Hermitian, or non symmetric case, is Arnoldi's method [2]. This is not discussed further since only the symmetric case is relevant to the applications considered in this thesis.

The aim is to solve (3.1) by first using the Lanczos method to reduce the coefficient matrix M to tridiagonal form. Let $\mathbf{v}_1 \in \mathbb{R}^N$ denote any given starting vector such that $\|\mathbf{v}_1\|_2 = 1$, where $\|.\|_2$ represents the ℓ_2-norm. Applying the Lanczos method to M in (3.1) generates a sequence of symmetric tridiagonal matrices of the form

$$T_j = V_j^T M V_j \quad (j = 1, \ldots, N) \tag{3.3}$$

where $V_j \in \mathbb{R}^{N\times j}$ $(j = 1, \ldots, N)$ is orthogonal. It follows from (3.3) that M and T_j are similar, that is, T_j has the same eigenvalues as M. Constructing (3.3) is equivalent to solving

$$MV_j = V_j T_j \quad (j = 1, \ldots, N) \tag{3.4}$$

since $V_j^T V_j = I_j$, where I_j denotes the $j \times j$ identity matrix. Let

$$V_j = (\mathbf{v}_1 \mid \ldots \mid \mathbf{v}_j) \tag{3.5}$$

and

$$T_j = \begin{pmatrix} \alpha_1 & \beta_1 & & \ldots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{j-1} \\ 0 & \ldots & & \beta_{j-1} & \alpha_j \end{pmatrix}. \tag{3.6}$$

Substituting (3.5) and (3.6) into (3.4) gives the set of equations

$$M\mathbf{v}_j = \beta_{j-1}\mathbf{v}_{j-1} + \alpha_j \mathbf{v}_j + \beta_j \mathbf{v}_{j+1} \quad (j = 1, \ldots, N-1) \tag{3.7}$$

where $\beta_0 \mathbf{v}_0 \equiv 0$, which can be solved to compute the entries of T_j in (3.6). Note that the columns of V_j in (3.5) form an orthonormal basis of the Krylov subspace

$$\mathcal{K}_j(M, \mathbf{v}_1) = \text{span}\left\{\mathbf{v}_1, M\mathbf{v}_1, \ldots, M^{j-1}\mathbf{v}_1\right\}, \tag{3.8}$$

see, for example, [50]. The vectors $\mathbf{v}_j$ $(j = 1, \ldots, N)$ are referred to as Lanczos vectors. The Lanczos method is outlined in Algorithm 3.1.

```
1: procedure LANCZOS(M, v_1)
2:     Set β_0 = ||v_1||_2, v_0 = 0, and r_0 = v_1
3:     while β_j ≠ 0 do
4:         v_j = (1/β_{j-1}) r_{j-1}
5:         α_j = v_j^T M v_j
6:         r_j = (M − α_j I_N) v_j − β_{j-1} v_{j-1}
7:         β_j = ||r_j||_2
8:     end while
9: end procedure
```

Algorithm 3.1: The Lanczos method.

One iteration of Algorithm 3.1 involves computing one matrix vector product of the form $M\mathbf{v}_j$. The matrix M is accessed solely for this purpose, and remains unchanged afterwards. It is necessary to store the two latest Lanczos vectors $\mathbf{v}_j$ and $\mathbf{v}_{j-1}$ in order

to compute $\mathbf{r}_j$. If $\beta_j = 0$, or equivalently, if $\mathbf{r}_j = \mathbf{0}$, then this indicates that $\mathbf{v}_1$ is contained in an invariant subspace of M. Thus $\beta_j = 0$, or equivalently, $\mathbf{r}_j = \mathbf{0}$, only if the columns of V_j in (3.5) span an invariant subspace of M. In theory, Algorithm 3.1 terminates at iteration number $j = \hat{j}$ where

$$\hat{j} = \text{rank}(\mathcal{K}_N(M, \mathbf{v}_1))$$

and $\mathcal{K}_N(M, \mathbf{v}_1)$ has the form (3.8) with $j = N$. It follows that

$$MV_j = V_j T_j + \mathbf{r}_j \mathbf{e}_j^T \quad (j = 1, \ldots, \hat{j}) \tag{3.9}$$

where V_j has orthonormal columns that span $\mathcal{K}_j(M, \mathbf{v}_1)$, T_j has the form (3.6), and $\mathbf{e}_j \in \mathbb{R}^j$ denotes the jth axis vector; see [50, p. 474, Theorem 9.1.1].

The columns of V_j in (3.5) form an orthonormal basis of $\mathcal{K}_j(M, \mathbf{v}_1)$ provided that exact arithmetic is employed. However, this is not the case in finite precision arithmetic due to the introduction of rounding error. The first documentation of round off error influence on the Lanczos method is attributed to [93]. An error analysis of the Lanczos method is given in [94], [95]; see also [96]. Algorithm 3.1 is usually terminated in practice once $\beta_j < t_\beta$, where t_β denotes a user specified tolerance. The difficulties associated with implementing Algorithm 3.1 are typically addressed by introducing a reorthogonalisation scheme. An error analysis of the Lanczos method with reorthogonalisation in the symmetric case is presented in [92]. Selective reorthogonalisation is discussed in [98]; see also [109]. An implementation of the Lanczos method with no reorthogonalisation is outlined in [27]. A review of practical Lanczos schemes applicable for solving large-scale symmetric eigenvalue problems is given in [28]; see also [29]. The block Lanczos method is discussed in [25], [26], [49], [126].

The standard procedure following the termination of Algorithm 3.1 is to obtain estimates of the eigenvalues of T_j in (3.6) using a suitable method. These are, in turn, approximations to the eigenvalues of M, since M and T_j are similar. The resulting eigenvalue problem at iteration number j has the form

$$T_j \mathbf{d}_j^i = \theta_j^i \mathbf{d}_j^i \quad (i = 1, \ldots, j) \tag{3.10}$$

where $\mathbf{d}_j^i \in \mathbb{R}^j$ $(i = 1, \ldots, j)$, and $\theta_j^i \in \mathbb{R}$ $(i = 1, \ldots, j)$. The values θ_j^i $(i = 1, \ldots, j)$ satisfying (3.10) are referred to as the Ritz values of M. The corresponding Ritz vectors are given by

$$\hat{\mathbf{d}}_j^i = V_j \mathbf{d}_j^i \quad (i = 1, \ldots, j) \tag{3.11}$$

where $\mathbf{d}_j^i$ satisfies (3.10). Note that the Lanczos vectors $\mathbf{v}_i$ $(i = 1, \ldots, j)$ must be stored in order to recover $\hat{\mathbf{d}}_j^i$ in (3.11). The accuracy of a Ritz pair $\{\theta_j^i, \hat{\mathbf{d}}_j^i\}$ improves as j increases. It follows from (3.9)-(3.11) that the residual of $\{\theta_j^i, \hat{\mathbf{d}}_j^i\}$ is given by

$$\begin{aligned} M\hat{\mathbf{d}}_j^i - \theta_j^i \hat{\mathbf{d}}_j^i &= MV_j \mathbf{d}_j^i - \theta_j^i V_j \mathbf{d}_j^i \\ &= (MV_j - V_j T_j)\mathbf{d}_j^i \\ &= \mathbf{r}_j \mathbf{e}_j^T \mathbf{d}_j^i \end{aligned} \tag{3.12}$$

thus

$$\|M\hat{\mathbf{d}}_j^i - \theta_j^i \hat{\mathbf{d}}_j^i\|_2 = \|\beta_j \tilde{d}_{j,i} \mathbf{v}_{j+1}\|_2 = |\beta_j||\tilde{d}_{j,i}| \tag{3.13}$$

where $\tilde{d}_{j,i}$ denotes the jth component of $\mathbf{d}_j^i$. Computing (3.13) is useful for tracking the convergence of a Ritz value θ_j^i. An estimate of (3.13) is obtained in practice without calculating $\hat{\mathbf{d}}_j^i$ directly. If (3.13) satisfies a prescribed tolerance, then θ_j^i is flagged as a converged Ritz value. The corresponding Ritz vector, if required, can be computed using (3.11). It is not possible to determine in advance the number of iterations required to compute the Ritz values to within a specified accuracy, although, the eigenspectrum of M, and the starting vector $\mathbf{v}_1$, are important factors in this respect. Problems with storing V_j in (3.5) arise as j becomes large. Theoretical results pertaining to the convergence of the Ritz values are presented in [63], [93]; see [105], [134] for extensions of this work.

3.2 The QR method

The QR method [40], [41], [66] is an eigenvalue computation technique. An overview of this procedure is given in [128], [130]; see also [50], [114]. The QR method is applicable for obtaining estimates of the eigenvalues of M in (3.1). However, applying the QR method to T_j in (3.6) on termination of Algorithm 3.1 is advantageous in practice

since the form of T_j simplifies the computational steps. The premise is to construct a sequence of similarity transformations that converge to the Schur decomposition of T_j, namely,

$$D^T T_j D = \text{diag}(\theta_j^1, \theta_j^2, \ldots, \theta_j^j) \tag{3.14}$$

where $D \in \mathbb{R}^{j \times j}$ is orthogonal. Let $\mathbf{d}_j^i \in \mathbb{R}^j$ denote the ith column of D. Thus (3.10) follows from (3.14); see [50, p 393, Theorem 8.1.1]. Applying the QR method to T_j in (3.6) generates a sequence of symmetric tridiagonal matrices

$$\{T_j^{(0)}, T_j^{(1)}, \ldots, T_j^{(i)}\}.$$

The first step is to construct a factorisation of the form

$$T_j^{(i-1)} = \tilde{D}_i \tilde{U}_i \quad (i = 1, 2, \ldots) \tag{3.15}$$

where $\tilde{D}_i \in \mathbb{R}^{j \times j}$ $(i = 1, 2, \ldots)$ is orthogonal, and $\tilde{U}_i \in \mathbb{R}^{j \times j}$ $(i = 1, 2, \ldots)$ is upper triangular. An iterate is generated by multiplying the factors $\tilde{D}_i$ and $\tilde{U}_i$ in the reverse order of (3.15). That is,

$$T_j^{(i)} = \tilde{U}_i \tilde{D}_i \quad (i = 1, 2, \ldots). \tag{3.16}$$

Combining (3.15) and (3.16) gives

$$T_j^{(i)} = \tilde{D}_i^T T_j^{(i-1)} \tilde{D}_i \tag{3.17}$$

since $\tilde{D}_i^T \tilde{D}_i = I_j$. Thus $T_j^{(i-1)}$ and $T_j^{(i)}$ are similar. It follows from (3.17) that

$$T_j^{(i)} = \hat{D}^T T_j \hat{D} \tag{3.18}$$

where

$$\hat{D} = \tilde{D}_1 \tilde{D}_2 \ldots \tilde{D}_i. \tag{3.19}$$

The QR method is terminated in practice once the off diagonal entries of $T_j^{(i)}$ in (3.16) are sufficiently small. Estimates of the eigenvalues of M are contained along the main diagonal of $T_j^{(i)}$. The corresponding eigenvectors, if required, can be computed analo-

gously to (3.11) using $\hat{D}$ in (3.19). The QR method is outlined in Figure 3.1.

procedure QR(M)
    Construct $T_j = V_j^T M V_j$ using Algorithm 3.1
    Set $T_j^{(0)} = T_j$ and $\tilde{D}_0 = I_j$
    for $i = 1, 2, \ldots,$ **do**
        $T_j^{(i-1)} = \tilde{D}_i \tilde{U}_i$
        $T_j^{(i)} = \tilde{U}_i \tilde{D}_i$
        $\hat{D} = \tilde{D}_{i-1} \tilde{D}_i$
    end for
end procedure

Figure 3.1: The QR method.

A useful variant of the QR method is obtained by introducing a shift $\omega_i \in \mathbb{R}$ in (3.15) and (3.16); see [40]. Specifically,

$$T_j^{(i-1)} - \omega_i I_j = \tilde{D}_i \tilde{U}_i \quad (i = 1, 2, \ldots), \tag{3.20}$$

$$T_j^{(i)} = \tilde{U}_i \tilde{D}_i + \omega_i I_j \quad (i = 1, 2, \ldots). \tag{3.21}$$

The matrix $T_j^{(i)}$ in (3.21) is symmetric tridiagonal. Note that combining (3.20) and (3.21) gives (3.17). It follows that $T_j^{(i)}$ has the form (3.18)-(3.19) in this case. A shift is introduced in the QR method to accelerate the convergence of this procedure. The rate of convergence is dependent on the choice of shift. Shift selection strategies are discussed in [50]; see also [114]. The shifted QR method is outlined in Figure 3.2.

procedure QRSHIFT(M)
    Construct $T_j = V_j^T M V_j$ using Algorithm 3.1
    Set $T_j^{(0)} = T_j$ and $\tilde{D}_0 = I_j$
    for $i = 1, 2, \ldots,$ **do**
        Determine a shift $\omega_i \in \mathbb{R}$
        $T_j^{(i-1)} - \omega_i I_j = \tilde{D}_i \tilde{U}_i$
        $T_j^{(i)} = \tilde{U}_i \tilde{D}_i + \omega_i I_j$
        $\hat{D} = \tilde{D}_{i-1} \tilde{D}_i$
    end for
end procedure

Figure 3.2: The shifted QR method.

3.3 The implicitly restarted Lanczos method

The implicitly restarted Lanczos method [111] is an adaptation of the Lanczos method for solving large-scale Hermitian or symmetric eigenvalue problems. The equivalent Arnoldi variant is applicable in non-Hermitian and non-symmetric cases; see [111]. The purpose of introducing implicit restarting in the Lanczos method is to overcome the numerical issues associated with this procedure. An important modification is that the number of Lanczos iterations to be performed is specified in advance. This ensures that the number of Lanczos vectors generated is predetermined. The following discussion relates to the application of the implicitly restarted Lanczos method for solving (3.1).

Suppose that estimates of a specified number q of the eigenvalues of M in (3.1) are required, and let s denote a positive integer such that $f = q + s < N$. Let $\mathbf{v}_1 \in \mathbb{R}^N$ denote any given starting vector such that $\|\mathbf{v}_1\|_2 = 1$, where $\|.\|_2$ represents the ℓ_2-norm. It follows from (3.9) that terminating Algorithm 3.1 after $j = f$ iterations gives

$$MV_f = V_f T_f + \mathbf{r}_f \mathbf{e}_f^T. \tag{3.22}$$

The premise here is to apply an analogy of the shifted QR method outlined in Figure 3.2 to T_f in (3.22). The first step is to construct

$$T_f - \omega_i I_f = \tilde{D}_i \tilde{U}_i, \tag{3.23}$$

$$T_f^+ = \tilde{U}_i \tilde{D}_i + \omega_i I_f \tag{3.24}$$

analogously to (3.20) and (3.21). The matrix T_f^+ in (3.24) is an update of T_f in (3.23). The shift ω_i is introduced to (3.22) as follows:

$$(M - \omega_i I_N)V_f - V_f(T_f - \omega_i I_f) = \mathbf{r}_f \mathbf{e}_f^T. \tag{3.25}$$

Substituting (3.23) in (3.25) gives

$$(M - \omega_i I_N)V_f - V_f \tilde{D}_i \tilde{U}_i = \mathbf{r}_f \mathbf{e}_f^T. \tag{3.26}$$

The starting vector $\mathbf{v}_1$ is updated through this procedure as demonstrated by post

multiplying (3.26) by the axis vector $\mathbf{e}_1 \in \mathbb{R}^f$. It follows that

$$(M - \omega_i I_N)\mathbf{v}_1 = \mathbf{v}_1^+ \hat{u}$$

where

$$\mathbf{v}_1^+ = V_f \tilde{D}_i \mathbf{e}_1, \quad \hat{u} = \mathbf{e}_1^T \tilde{U}_i \mathbf{e}_1.$$

The vector $\mathbf{v}_1^+$ is an update of $\mathbf{v}_1$. The next step is to post multiply (3.26) by $\tilde{D}_i$ to obtain

$$(M - \omega_i I_N)(V_f \tilde{D}_i) - (V_f \tilde{D}_i)(\tilde{U}_i \tilde{D}_i) = \mathbf{r}_f \mathbf{e}_f^T \tilde{D}_i. \tag{3.27}$$

Rearranging terms in (3.27) gives

$$M(V_f \tilde{D}_i) - (V_f \tilde{D}_i)(\tilde{U}_i \tilde{D}_i + \omega_i I_f) = \mathbf{r}_f \mathbf{e}_f^T \tilde{D}_i. \tag{3.28}$$

The shift strategy is extended by applying s shifts ω_i $(i = 1, \ldots, s)$ in (3.22). It follows from (3.28) that (3.22) has the updated form

$$MV_f^+ = V_f^+ T_f^+ + \mathbf{r}_f \mathbf{e}_f^T \hat{D}^+ \tag{3.29}$$

where

$$V_f^+ = V_f \hat{D}^+, \quad T_f^+ = (\hat{D}^+)^T T_f \hat{D}^+, \tag{3.30}$$

and

$$\hat{D}^+ = \tilde{D}_1 \tilde{D}_2 \ldots \tilde{D}_s. \tag{3.31}$$

The matrices defined in (3.30) are updates of V_f and T_f. An important property is that

$$(V_f^+)^T V_f^+ = I_f$$

since $V_f^T V_f = I_f$ and $(\hat{D}^+)^T \hat{D}^+ = I_f$. Partitioning V_f^+ and T_f^+ in (3.30) such that

$$V_f^+ = (V_q^+, \hat{V}_s), \quad T_f^+ = \begin{pmatrix} T_q^+ & \hat{\beta}_q \mathbf{e}_q \mathbf{e}_1^T \\ \hat{\beta}_q \mathbf{e}_1 \mathbf{e}_q^T & \hat{T}_s \end{pmatrix}, \tag{3.32}$$

the term $\mathbf{r}_f \mathbf{e}_f^T \hat{D}^+$ in (3.29) simplifies as follows

$$\mathbf{r}_f \mathbf{e}_f^T \hat{D}^+ = \beta_f \mathbf{v}_{f+1} \mathbf{e}_f^T \hat{D}^+ = \mathbf{v}_{f+1} \mathbf{h} \tag{3.33}$$

where

$$\mathbf{h} = (\underbrace{0, 0, \ldots, \tilde{\beta}_f}_{q}, \underbrace{\overline{\mathbf{h}}^T}_{s}). \tag{3.34}$$

Substituting (3.32)-(3.34) into (3.29) gives

$$M(V_q^+, \hat{V}_s) = (\underbrace{V_q^+ T_q^+ + \hat{\beta}_q \hat{V}_s \mathbf{e}_1 \mathbf{e}_q^T + \tilde{\beta}_f \mathbf{v}_{f+1} \mathbf{e}_q^T}_{q}, \underbrace{\hat{\beta}_q V_q^+ \mathbf{e}_q \mathbf{e}_1^T + \hat{V}_s \hat{T}_s + \mathbf{v}_{f+1} \overline{\mathbf{h}}^T}_{s}). \tag{3.35}$$

The first q columns in (3.35) satisfy

$$M V_q^+ = V_q^+ T_q^+ + \mathbf{r}_q^+ \mathbf{e}_q^T \tag{3.36}$$

where

$$\mathbf{r}_q^+ = \hat{\beta}_q \hat{V}_s \mathbf{e}_1 + \tilde{\beta}_f \mathbf{v}_{f+1}, \tag{3.37}$$

$$\mathbf{v}_{q+1}^+ = \frac{1}{\beta_q^+} \mathbf{r}_q^+, \quad \beta_q^+ = \|\mathbf{r}_q^+\|_2. \tag{3.38}$$

It follows that (3.36)-(3.38) is a Lanczos factorisation of M since

$$(V_q^+)^T \mathbf{r}_q^+ = \hat{\beta}_q (V_q^+)^T \hat{V}_s \mathbf{e}_1 + \tilde{\beta}_f (V_q^+)^T \mathbf{v}_{f+1} = \mathbf{0}$$

thus

$$(V_q^+)^T \mathbf{v}_{q+1}^+ = \frac{1}{\beta_q^+} (V_q^+)^T \mathbf{r}_q^+ = \mathbf{0}.$$

A new factorisation of the form (3.22) is obtained at this stage by performing $j = s$ iterations of Algorithm 3.1 starting from (3.36)-(3.38), with $\mathbf{v}_1$ replaced by $\mathbf{v}_1^+$. The process continues in a recursive manner with the alternative application of s shifts, and the execution of s iterations of Algorithm 3.1, until a given convergence criterion is satisfied. Estimates of the q requested eigenvalues of M are contained along the main diagonal of T_q^+ in (3.36) on termination of the procedure. The corresponding eigenvectors, if required, can be computed using V_q^+ in (3.36) and $\hat{D}^+$ in (3.31). The

implicitly restarted Lanczos method is outlined in Algorithm 3.2.

procedure IRLM($M, \mathbf{v}_1, q, s$)
  Set $f = q + s$
  Perform $j = f$ iterations of Algorithm 3.1 to obtain
    $MV_f = V_f T_f + \mathbf{r}_f \mathbf{e}_f^T$
  for $i = 1, 2, \ldots$ **do**
    Determine the shifts ω_j $(j = 1, \ldots, s)$
    Set $\tilde{D}_0 = I_f$
    for $j = 1, \ldots, s$ **do**
      $T_f - \omega_j I_f = \tilde{D}_j \tilde{U}_j$
      $T_f^+ = \tilde{U}_j \tilde{D}_j + \omega_j I_f$
      $\hat{D}^+ = \tilde{D}_{j-1} \tilde{D}_j$
    end for
    Set $V_q^+ = V_f^+ \hat{D}^+(:, 1:q)$ and $T_q^+ = T_f^+(1:q, 1:q)$
    Perform $j = s$ iterations of Algorithm 3.1 starting from
      $MV_q^+ = V_q^+ T_q^+ + \mathbf{r}_q^+ \mathbf{e}_q^T$
    with $\mathbf{v}_1 = \mathbf{v}_1^+$ to obtain a Lanczos factorisation of the form
      $MV_f = V_f T_f + \mathbf{r}_f \mathbf{e}_f^T$
  end for
end procedure

Algorithm 3.2: The implicitly restarted Lanczos method.

The implementation of Algorithm 3.2 involves computing f matrix vector products of the form $M\mathbf{v}_j$. The procedure outlined in Algorithm 3.2 is a polynomial acceleration scheme. The first application of polynomial acceleration for solving eigenvalue problems is attributed to [37]. An overview of polynomial acceleration techniques in this context is given in [107]. Applying s shifts ω_i $(i = 1, \ldots, s)$ to T_f in (3.22) implicitly updates the starting vector $\mathbf{v}_1$ as follows

$$\mathbf{v}_1^+ = \frac{1}{\tau} \psi_s(M) \mathbf{v}_1$$

where

$$\psi_s(M) = (M - \omega_s I_N)(M - \omega_{s-1} I_N) \ldots (M - \omega_1 I_N) \tag{3.39}$$

and $\tau \in \mathbb{R}$ denotes a scaling factor included to ensure that $\|\mathbf{v}_1^+\|_2 = 1$. The polynomial $\psi_s(M)$ in (3.39) is of degree s. The shifts ω_i $(i = 1, \ldots, s)$ are the zeros of $\psi_s(M)$. It

therefore follows from (3.1) and (3.39) that

$$\psi_s(\mu) = \prod_{i=1}^{s} (\mu - \omega_i). \tag{3.40}$$

The premise is to define ω_i $(i = 1, \ldots, s)$ in (3.40) such that $\psi_s(M)\mathbf{v}_1$ dampens the undesirable components of $\mathbf{v}_1$; see [107]. Knowledge of the eigenspectrum of M is useful in this respect. If no such information is available, then the recommended procedure is to implement exact shifts; see [111]. This involves using the s unwanted eigenvalue estimates of T_f as shifts. Alternative shift selection strategies are described in [6], [17]. The convergence of Algorithm 3.2 is discussed in [111]. Deflation techniques designed to accelerate the convergence of Algorithm 3.2 are presented in [76], [77].

The standard implementation of Algorithm 3.2 is available through the ARPACK software; see [78]. Note that this is applied in the numerical studies presented in Chapters 4, 5, and 6. Suppose that θ_j^i $(i = 1, \ldots, j)$ satisfying (3.10) is a Ritz value of M in (3.1) computed at iteration number j of Algorithm 3.1 as per step 13 of Algorithm 3.2. The corresponding Ritz vector is $\hat{\mathbf{d}}_j^i$ in (3.11). The convergence of θ_j^i is determined in ARPACK using the residual norm defined in (3.13). The criterion employed is

$$\|M\hat{\mathbf{d}}_j^i - \theta_j^i \hat{\mathbf{d}}_j^i\|_2 \leq t_e \hat{\varepsilon} \quad (i = 1, \ldots, j) \tag{3.41}$$

where

$$\hat{\varepsilon} = \max(\varepsilon, |\theta_j^i|)$$

with t_e a user specified tolerance and ε a machine constant. The default tolerance used in the numerical studies presented in Chapters 4, 5, and 6 is $t_e = 10^{-2}$.

3.4 The conjugate gradient method

The conjugate gradient (CG) method [60] is an iterative technique applicable for solving systems of linear equations of the form (3.2). The starting point is to observe that (3.2) can be formulated as the minimisation problem

$$\min_{\mathbf{q} \in \mathbb{R}^N} \phi(\mathbf{q}) \tag{3.42}$$

where

$$\phi(\mathbf{q}) = \frac{1}{2}\mathbf{q}^T K\mathbf{q} - \mathbf{q}^T\mathbf{f}. \tag{3.43}$$

The gradient of $\phi(\mathbf{q})$ in (3.43) calculated with respect to $\mathbf{q}$ is

$$\nabla\phi(\mathbf{q}) = K\mathbf{q} - \mathbf{f}. \tag{3.44}$$

The solution $\mathbf{q}$ to $\nabla\phi(\mathbf{q}) = \mathbf{0}$ is therefore the minimiser of (3.42)-(3.43). Clearly, setting $\nabla\phi(\mathbf{q})$ in (3.44) equal to zero and rearranging terms gives (3.2), so the minimiser $\mathbf{q}$ of (3.42)-(3.43) is also the solution of (3.2). The idea of CG is to solve (3.42)-(3.43) iteratively instead of solving (3.2) directly. Let $\mathbf{q}_0 \in \mathbb{R}^N$ denote any given initial estimate of $\mathbf{q}$ in (3.2). The steps involved in solving (3.2) using the conjugate gradient method are outlined in Algorithm 3.3.

```
1:  procedure CG(K, f, q_0)
2:      Set r̂_0 = f − K q_0
3:      while r̂_i ≠ 0 do
4:          if i = 1 then
5:              p_1 = r̂_0
6:          else
7:              η_i = r̂_{i−1}^T r̂_{i−1} / r̂_{i−2}^T r̂_{i−2}
8:              p_i = r̂_{i−1} + η_i p_{i−1}
9:          end if
10:         ν_i = r̂_{i−1}^T r̂_{i−1} / p_i^T K p_i
11:         q_i = q_{i−1} + ν_i p_i
12:         r̂_i = r̂_{i−1} − ν_i K p_i
13:     end while
14: end procedure
```

Algorithm 3.3: The conjugate gradient method.

One iteration of Algorithm 3.3 involves computing one matrix vector product of the form $K\mathbf{p}_i$. The set of vectors $\{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_i\}$ generated by applying Algorithm 3.3 to (3.2) are conjugate with respect to the matrix K, that is,

$$\mathbf{p}_i^T K\mathbf{p}_j = 0, \quad \forall\, i \neq j,$$

hence the method's name. The convergence rate of Algorithm 3.3 depends on the eigenvalue distribution of K: specifically, convergence is faster when the eigenvalues of K

are clustered in a small number of distinct groups. In theory, Algorithm 3.3 terminates after at most N iterations; see, for example, [50, p. 530, Theorem 10.2.5]. However, this is not guaranteed in practice due to the introduction of rounding error. The behaviour of the conjugate gradient method in finite precision arithmetic is discussed in [52], [133]; see also [61], [113]. The convergence rate of the conjugate gradient method in exact arithmetic has been considered in [127]. Algorithm 3.3 is generally terminated once $\|\hat{\mathbf{r}}_i\|_2 < t_c\|\hat{\mathbf{r}}_0\|_2$, where $\|.\|_2$ represents the ℓ_2-norm and t_c denotes a user specified tolerance.

3.5 The preconditioned conjugate gradient method

A preconditioner is usually applied to (3.2) in order to accelerate the convergence of the conjugate gradient method. Suppose that $W \in \mathbb{R}^{N\times N}$ is symmetric positive definite. The theoretical idea is to precondition (3.2) with W^{-1} as follows:

$$W^{-1}K\mathbf{q} = W^{-1}\mathbf{f}. \tag{3.45}$$

Note that solving (3.45) is equivalent to solving (3.2). The aim is to define W such that the eigenvalues of $W^{-1}K$ are more clustered than the eigenvalues of K, resulting in faster CG convergence. However, $W^{-1}K$ must be symmetric in order to apply the conjugate gradient method to (3.45). This is not generally the case, so in practice it is usual to set $W = G^2$ and precondition (3.2) symmetrically. This gives

$$\hat{K}\hat{\mathbf{q}} = \hat{\mathbf{f}} \tag{3.46}$$

where $\hat{K} = G^{-1}KG^{-1}$, $\hat{\mathbf{q}} = G\mathbf{q}$, and $\hat{\mathbf{f}} = G^{-1}\mathbf{f}$. The conjugate gradient method can now be applied to (3.46) since $\hat{K}$ is symmetric positive definite. Note that $\hat{K}$ and $\hat{\mathbf{q}}$ in (3.46) are not constructed in practice, but $W^{-1}\tilde{\mathbf{q}}$ must be computable for any given vector $\tilde{\mathbf{q}} \in \mathbb{R}^N$. The matrices $W^{-1}K$ and $\hat{K}$ are similar since

$$GW^{-1}KG^{-1} = G(G^{-1}G^{-1})KG^{-1} = G^{-1}KG^{-1} = \hat{K}.$$

Thus $W^{-1}K$ has the same eigenvalues as $\hat{K}$. Let $\mathbf{q}_0 \in \mathbb{R}^N$ denote any given initial estimate of $\mathbf{q}$ in (3.2). The steps involved in solving (3.45) using the preconditioned conjugate gradient method are outlined in Algorithm 3.4.

```
1: procedure PCG(K, f, q_0)
2:     Set r̂_0 = f − K q_0
3:     while r̂_i ≠ 0 do
4:         Solve W c_{i−1} = r̂_{i−1}
5:         if i = 1 then
6:             p_1 = c_0
7:         else
8:             η_i = r̂_{i−1}^T c_{i−1} / r̂_{i−2}^T c_{i−2}
9:             p_i = c_{i−1} + η_i p_{i−1}
10:        end if
11:        ν_i = r̂_{i−1}^T c_{i−1} / p_i^T K p_i
12:        q_i = q_{i−1} + ν_i p_i
13:        r̂_i = r̂_{i−1} − ν_i K p_i
14:    end while
15: end procedure
```

Algorithm 3.4: The preconditioned conjugate gradient method.

One iteration of Algorithm 3.4 still involves computing only one matrix vector product of the form $K\mathbf{p}_i$. However, Algorithm 3.4 requires additional computational work compared to Algorithm 3.3 since the system of linear equations $W\mathbf{c}_{i-1} = \hat{\mathbf{r}}_{i-1}$ must be solved at step 4. Of course, this should be feasible in practice. Setting $W = I_N$ in Algorithm 3.4 gives the standard conjugate gradient method (with no change to the spectrum of K). If $W = K$, then Algorithm 3.4 converges in one iteration, but solving the linear system at step 4 is just as computationally expensive as solving (3.2). The sensible approach for choosing W involves balancing the computational cost of solving the linear system at step 4 of Algorithm 3.4 with the desired enhanced convergence rate of the method.

The preconditioned conjugate gradient method is applied in the numerical studies presented in Chapter 6. The tolerance used in the stopping criterion in these numerical studies is $t_c = 10^{-2}$. The number of iterations permitted is restricted, as is generally the case in NWP applications. Two cases are considered in this thesis, namely, $N_c = 5$ and $N_c = 10$, where N_c denotes the maximum allowable number of PCG iterations.

Chapter 4

A multilevel eigenvalue decomposition algorithm for approximating the inverse Hessian

A multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of any given symmetric positive definite matrix $A \in \mathbb{R}^{N\times N}$ with eigenvalues clustered around unity is presented in this chapter. It is assumed that A can be approximated using a specified number of its leading eigenvalues, and corresponding eigenvectors, by means of a limited-memory approximation of the form defined in (2.51). That is, for any $\gamma \in \mathbb{R}$,

$$A^{\gamma} \simeq I_N + \sum_{i=1}^{r} (\lambda_i^{\gamma} - 1)\mathbf{u}_i\mathbf{u}_i^T \tag{4.1}$$

where $\{\lambda_i, \mathbf{u}_i\}$ $(i = 1, \ldots, r)$ denote r eigenpairs of A, with $r \ll N$. Note that $A\mathbf{w}$ must be computable for any given vector $\mathbf{w} \in \mathbb{R}^N$.

The discussions in Sections 4.1-4.4 focus on the multilevel eigenvalue decomposition algorithm. The multilevel grid structure employed is described in Section 4.1 in the first instance. The interpolation and restriction operators required to transfer information between grids in this multilevel construction are discussed in Section 4.2. Additional

specialised grid transfer operators used in the multilevel eigenvalue decomposition algorithm are defined in Section 4.3. An outline of the multilevel eigenvalue decomposition algorithm is presented in Section 4.4.1, and the algorithm is described in detail in Section 4.4.2. The preconditioning strategy adopted is discussed in Section 4.4.3. Finally, a summary of the complete algorithm is presented in Section 4.4.4.

The particular case of the matrix A considered in this chapter is the Hessian C in (2.50) (see Section 4.5). The numerical studies presented in this chapter involve approximating the inverse Hessian for the specific model problem outlined in Section 4.6. The implementation details pertaining to this model problem are discussed in Section 4.7. The numerical measure employed to evaluate the accuracy of approximations constructed using the multilevel eigenvalue decomposition algorithm is defined in Section 4.8. The choices of preconditioner applied in the multilevel eigenvalue decomposition algorithm, and interpolation method used to implement the interpolation operator, are considered in Sections 4.9 and 4.10, respectively. The software package ARPACK [78] discussed in Section 3.3 is applied in the numerical studies presented in Sections 4.10 and 4.12. A modified version of ARPACK that permits only one shift to be executed is considered in Section 4.11. The numerical study presented in Section 4.12 focuses on comparing various approximations to the inverse Hessian constructed using the multilevel eigenvalue decomposition algorithm.

4.1 Multilevel grid structure

The multilevel eigenvalue decomposition algorithm is outlined in this chapter for the one-dimensional case. However, the concepts discussed are also valid in two- and three-dimensional settings. Suppose that the matrix A represents an operator based on a grid defined over any given spatial domain Ω. A sequence of grids over Ω, assumed here to be $\Omega = [0, 1]$, is required for the multilevel eigenvalue decomposition algorithm. Suppose that the base grid contains $m_0 = m + 1$ uniformly distributed grid points. Specifically, m must be a positive integer that is divisible by $2^{n_l - 1}$, where n_l is the total number of grid levels required in the multilevel construction. The base grid will be the finest grid in the multilevel setup, denoted using the level counter $k = 0$. The

next grid is constructed by removing a grid point from between each pair of existing grid points on the finest grid. This generates a grid at grid level $k = 1$ that is composed of $m_1 = m/2 + 1$ uniformly distributed grid points. The grid coarsening process is continued in order to construct a sequence of grids at grid levels $k = 0, 1, 2, \ldots, k_c$, where $k = k_c$ denotes the coarsest grid level and $k_c = n_l - 1$. Note that the grid at a general grid level k in this setup contains $m_k = m/2^k + 1$ grid points. In what follows, the notation I_k represents the $m_k \times m_k$ identity matrix associated with grid level k in the multilevel construction.

4.2 Interpolation and restriction operators

Interpolation and restriction operators are required to transfer information between grids in the multilevel construction described in Section 4.1. The interpolation operator which prolongs any given vector $\mathbf{w}_k \in \mathbb{R}^{m_k}$ defined at grid level k to the finer grid at grid level $k - i$ $(0 \leq i \leq k)$ is introduced in the first instance. Let $P_k^{k-i} \in \mathbb{R}^{m_{k-i} \times m_k}$ $(0 \leq i \leq k)$ denote the interpolation matrix that prolongs $\mathbf{w}_k$ from grid level k to grid level $k - i$. That is,

$$\tilde{\mathbf{w}}_{k-i} = P_k^{k-i} \mathbf{w}_k, \quad P_k^k = I_k \quad (0 \leq i \leq k). \tag{4.2}$$

The restriction operator performs the opposite procedure of restricting any given vector $\mathbf{w}_{k-i} \in \mathbb{R}^{m_{k-i}}$ defined at grid level $k - i$ to the coarser grid at grid level k. Let $(P_k^{k-i})^T \in \mathbb{R}^{m_k \times m_{k-i}}$ denote the restriction matrix that restricts $\mathbf{w}_{k-i}$ from grid level $k - i$ to grid level k. Specifically,

$$\hat{\mathbf{w}}_k = (P_k^{k-i})^T \mathbf{w}_{k-i}, \quad (P_k^k)^T = I_k \quad (0 \leq i \leq k). \tag{4.3}$$

If $\mathbf{w}_{k-i} = \tilde{\mathbf{w}}_{k-i}$, then the definition of $\tilde{\mathbf{w}}_{k-i}$ in (4.2) can be substituted into (4.3). This gives

$$\hat{\mathbf{w}}_k = (P_k^{k-i})^T \tilde{\mathbf{w}}_{k-i} = (P_k^{k-i})^T P_k^{k-i} \mathbf{w}_k. \tag{4.4}$$

It follows from (4.4) that

$$\hat{\mathbf{w}}_k = \mathbf{w}_k \tag{4.5}$$

only if

$$(P_k^{k-i})^T P_k^{k-i} = I_k. \tag{4.6}$$

Provided that P_k^{k-i} is orthogonal, then the condition in (4.6) holds. The specific form of P_k^{k-i} in (4.2) used in the numerical studies presented in this chapter is discussed in Sections 4.7 and 4.10. Note that the orthogonality condition in (4.6) is not imposed in these cases.

4.3 Additional grid transfer operators

We now consider how the matrix A can be represented at the various grid levels in the multilevel construction described in Section 4.1. Suppose that A is constructed based on the finest grid at grid level $k = 0$: we will write $A \equiv A_0$. More generally, let $A_k \in \mathbb{R}^{m_k \times m_k}$ denote a representation of the matrix A at grid level k ($k = 0, \ldots, k_c$). Although the matrix A_0 may not be constructed in practice, it is assumed that $A_0\mathbf{w}_0$ is computable for any given vector $\mathbf{w}_0 \in \mathbb{R}^{m_0}$ defined at grid level $k = 0$. A strategy for computing $A_k\mathbf{w}_k$ ($k = 1, \ldots, k_c$), where $\mathbf{w}_k \in \mathbb{R}^{m_k}$ denotes any given vector defined at grid level k, is required for the multilevel eigenvalue decomposition algorithm.

Now suppose that A_k is defined at grid level k ($k = 1, \ldots, k_c$). Let $\mathbf{w}_{k-i} \in \mathbb{R}^{m_{k-i}}$ denote any given vector defined at a finer grid level $k - i$ ($0 < i \leq k$). The aim is to compute $\tilde{S}_{k-i}(A_k)\mathbf{w}_{k-i}$, where $\tilde{S}_{k-i}(.) : \mathbb{R}^{m_k \times m_k} \to \mathbb{R}^{m_{k-i} \times m_{k-i}}$ denotes a grid transfer operator required to prolong A_k from grid level k to grid level $k-i$. An obvious strategy would be to define $\tilde{S}_{k-i}(A_k)\mathbf{w}_{k-i}$ as follows:

$$\tilde{S}_{k-i}(A_k)\mathbf{w}_{k-i} = P_k^{k-i} A_k (P_k^{k-i})^T \mathbf{w}_{k-i} \quad (0 < i \leq k) \tag{4.7}$$

where

$$\tilde{S}_k(A_k) = P_k^k A_k (P_k^k)^T = I_k A_k I_k = A_k$$

using (4.2) and (4.3). Now suppose that A_{k-i} is defined at grid level $k-i$ ($0 < i \leq k_c$). Let $\mathbf{w}_k \in \mathbb{R}^{m_k}$ denote any given vector defined at a coarser grid level k. The aim in this case is to compute $\tilde{Q}_k(A_{k-i})\mathbf{w}_k$, where $\tilde{Q}_{k-i}(.) : \mathbb{R}^{m_{k-i} \times m_{k-i}} \to \mathbb{R}^{m_k \times m_k}$ denotes a grid transfer operator required to restrict A_{k-i} from grid level $k - i$ to grid level k.

Note that $\tilde{Q}_k(A_{k-i})\mathbf{w}_k$ can be defined analogously to $\tilde{S}_{k-i}(A_k)\mathbf{w}_{k-i}$ in (4.7) as follows:

$$\tilde{Q}_k(A_{k-i})\mathbf{w}_k = (P_k^{k-i})^T A_{k-i} P_k^{k-i}\mathbf{w}_k, \quad (0 < i \leq k_c) \tag{4.8}$$

where

$$\tilde{Q}_k(A_k) = (P_k^k)^T A_k P_k^k = I_k A_k I_k = A_k$$

using (4.2) and (4.3).

In this thesis, alternative grid transfer operators are used instead of $\tilde{S}_{k-i}(.)$ in (4.7) and $\tilde{Q}_k(.)$ in (4.8). These are derived by considering the process of restricting $\mathbf{w}_{k-i}$ to grid level k in detail. The first step is to decompose $\mathbf{w}_{k-i}$ into two vectors as follows:

$$\mathbf{w}_{k-i} = \mathbf{w}_{k-i}^a + \mathbf{w}_{k-i}^b \tag{4.9}$$

where

$$\mathbf{w}_{k-i}^a = (I_{k-i} - P_k^{k-i}(P_k^{k-i})^T)\mathbf{w}_{k-i} \tag{4.10}$$

and

$$\mathbf{w}_{k-i}^b = P_k^{k-i}(P_k^{k-i})^T\mathbf{w}_{k-i}. \tag{4.11}$$

Let $\overline{\mathbf{w}}_k \in \mathbb{R}^{m_k}$ denote the vector obtained by restricting $\mathbf{w}_{k-i}$ in (4.9)-(4.11) to grid level k using $(P_k^{k-i})^T$ in (4.3). Specifically,

$$\overline{\mathbf{w}}_k = \overline{\mathbf{w}}_k^a + \overline{\mathbf{w}}_k^b$$

where

$$\begin{aligned}
\overline{\mathbf{w}}_k^a &= (P_k^{k-i})^T\mathbf{w}_{k-i}^a \\
&= (P_k^{k-i})^T(I_{k-i} - P_k^{k-i}(P_k^{k-i})^T)\mathbf{w}_{k-i} \\
&= ((P_k^{k-i})^T - (P_k^{k-i})^T P_k^{k-i}(P_k^{k-i})^T)\mathbf{w}_{k-i}
\end{aligned} \tag{4.12}$$

and

$$\begin{aligned}\overline{\mathbf{w}}_k^b &= (P_k^{k-i})^T \mathbf{w}_{k-i}^b \\ &= (P_k^{k-i})^T P_k^{k-i} (P_k^{k-i})^T \mathbf{w}_{k-i}.\end{aligned}$$

Assuming that (4.6) holds, then $\overline{\mathbf{w}}_k^a$ in (4.12) simplifies to $\overline{\mathbf{w}}_k^a = \mathbf{0}$ and $\overline{\mathbf{w}}_k = \overline{\mathbf{w}}_k^b$ where $\overline{\mathbf{w}}_k^b = (P_k^{k-i})^T \mathbf{w}_{k-i}$. This indicates that $\mathbf{w}_{k-i}^a$ in (4.10) includes components of $\mathbf{w}_{k-i}$ that cannot be represented on the grid at grid level k. Thus $\mathbf{w}_{k-i}^a$ should not be restricted to grid level k. The premise of our novel grid transfer operator is therefore to restrict $\mathbf{w}_{k-i}^b$ in (4.11) to grid level k, apply A_k, and prolong the result to grid level $k-i$. That is,

$$\begin{aligned}S_{k-i}(A_k)\mathbf{w}_{k-i} &= \mathbf{w}_{k-i}^a + \tilde{S}_{k-i}(A_k)\mathbf{w}_{k-i}^b \\ &= \mathbf{w}_{k-i}^a + P_k^{k-i} A_k (P_k^{k-i})^T \mathbf{w}_{k-i}^b \\ &= (I_{k-i} - P_k^{k-i}(P_k^{k-i})^T)\mathbf{w}_{k-i} + P_k^{k-i} A_k (P_k^{k-i})^T P_k^{k-i} (P_k^{k-i})^T \mathbf{w}_{k-i} \\ &= (I_{k-i} - P_k^{k-i}(P_k^{k-i})^T)\mathbf{w}_{k-i} + P_k^{k-i} A_k (P_k^{k-i})^T \mathbf{w}_{k-i} \\ &= \left[P_k^{k-i}(A_k - I_k)(P_k^{k-i})^T + I_{k-i}\right]\mathbf{w}_{k-i} \qquad (4.13)\end{aligned}$$

using (4.6) and (4.7). The grid transfer operator $S_{k-i}(.) : \mathbb{R}^{m_k \times m_k} \to \mathbb{R}^{m_{k-i} \times m_{k-i}}$ in (4.13) is used in the multilevel eigenvalue decomposition algorithm instead of $\tilde{S}_k(.)$ in (4.7). Specifically, $S_{k-i}(A_k)$ is defined as follows:

$$S_{k-i}(A_k) = P_k^{k-i}(A_k - I_k)(P_k^{k-i})^T + I_{k-i} \quad (0 < i \leq k) \qquad (4.14)$$

where

$$S_k(A_k) = P_k^k(A_k - I_k)(P_k^k)^T + I_k = I_k(A_k - I_k)I_k + I_k = A_k$$

using (4.2) and (4.3). The transpose of $S_{k-i}(.)$ in (4.14) is also required. It follows

from (4.14) that

$$\begin{aligned} S_{k-i}^T(A_k) &= \left[P_k^{k-i}(A_k - I_k)(P_k^{k-i})^T + I_{k-i}\right]^T \\ &= P_k^{k-i}(A_k^T - I_k)(P_k^{k-i})^T + I_{k-i} \\ &= S_{k-i}(A_k^T). \end{aligned}$$

In a similar way, the grid transfer operator $Q_k(.) : \mathbb{R}^{m_{k-i} \times m_{k-i}} \rightarrow \mathbb{R}^{m_k \times m_k}$ is used in the multilevel eigenvalue decomposition algorithm instead of $\tilde{Q}_k(.)$ in (4.8). Note that $Q_k(A_{k-i})$ is defined analogously to $S_{k-i}(A_k)$ in (4.14) as follows:

$$Q_k(A_{k-i}) = (P_k^{k-i})^T(A_{k-i} - I_{k-i})P_k^{k-i} + I_k \quad (0 < i \leq k_c) \tag{4.15}$$

where

$$Q_k(A_k) = (P_k^k)^T(A_k - I_k)P_k^k + I_k = A_k$$

using (4.2) and (4.3). It follows from (4.15) that

$$\begin{aligned} Q_k^T(A_{k-i}) &= \left[(P_k^{k-i})^T(A_{k-i} - I_{k-i})P_k^{k-i} + I_k\right]^T \\ &= (P_k^{k-i})^T(A_{k-i}^T - I_{k-i})P_k^{k-i} + I_k \\ &= Q_k(A_{k-i}^T). \end{aligned}$$

Since we have assumed that (4.6) holds, then $S_{k-i}(A_k)$ in (4.14) has the property

$$S_{k-i}(A_k) = S_{k-i}(A_k^{1/2})S_{k-i}((A_k^{1/2})^T).$$

An additional result is that $Q_k(A_{k-i})$ in (4.15) simplifies to

$$Q_k(A_{k-i}) = (P_k^{k-i})^T A_{k-i} P_k^{k-i}$$

however

$$Q_k(A_{k-i}) \neq Q_k(A_{k-i}^{1/2})Q_k((A_{k-i}^{1/2})^T).$$

4.4 The multilevel eigenvalue decomposition algorithm

4.4.1 Outline

Suppose that A_0 is available on the finest grid at grid level $k = 0$ in the multilevel construction described in Section 4.1 in the form of $A_0\mathbf{w}_0$, where $\mathbf{w}_0 \in \mathbb{R}^{m_0}$ denotes any given vector defined on the finest grid. Full details of the multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to $A_0^{-1}\mathbf{w}_0$ and $A_0^{-1/2}\mathbf{w}_0$ are described in Section 4.4.2: we present here an outline of the algorithm to aid understanding. The steps involved in the multilevel eigenvalue decomposition algorithm are as follows:

1. Represent A_0 on the coarsest grid at grid level $k = k_c$ in the multilevel setup using the grid transfer operator $Q_k(.)$ defined in (4.15).

2. Apply a local preconditioner to $Q_k(A_0)$ in order to cluster its eigenvalues around unity.

3. Compute estimates of a specified number (n_k, say) of the largest eigenvalues, and corresponding eigenvectors, of the preconditioned matrix using the Lanczos method and store in memory. If $k = 0$, then terminate the procedure following this step.

4. Construct a limited-memory approximation to the inverse of the preconditioned matrix of the form defined in (4.1) to be used for preconditioning at the next finest grid level $k - 1$.

5. Progress to the next finest grid level $k - 1$ and repeat the procedure.

An approximation to $A_0^{-1}\mathbf{w}_0$ or $A_0^{-1/2}\mathbf{w}_0$ can be computed on termination of the multilevel eigenvalue decomposition algorithm using the n_k eigenpairs calculated at each grid level k ($k = 0, \ldots, k_c$) in the multilevel construction.

4.4.2 Details

The multilevel eigenvalue decomposition algorithm is now described in terms of the procedure at grid level k ($k = 1, \ldots, k_c$) in the multilevel construction outlined in

Section 4.1. The first step is to represent A_0 on the grid at grid level k using the grid transfer operator $Q_k(.)$ in (4.15) as follows:

$$Q_k(A_0) = (P_k^0)^T(A_0 - I_0)P_k^0 + I_k. \tag{4.16}$$

The next step is to precondition $Q_k(A_0)$ in (4.16) to obtain

$$\tilde{Q}_k(A_0) = (Z_{k+1}^k)^T Q_k(A_0) Z_{k+1}^k \tag{4.17}$$

where $Z_{k+1}^k \in \mathbb{R}^{m_k \times m_k}$ $(k = 0, \ldots, k_c)$ denotes a preconditioner defined on the grid at grid level k. The premise is to construct Z_{k+1}^k such that the eigenvalues of $\tilde{Q}_k(A_0)$ in (4.17) are closer to unity than those of $Q_k(A_0)$ in (4.16). The specific preconditioning strategy adopted, that is, the choice of Z_{k+1}^k in (4.17), will be discussed in Section 4.4.3. The notation Z_{k+1}^k highlights that this preconditioner is constructed at grid level $k+1$ then prolonged on to grid level k. The Lanczos method is used to compute estimates of a specified number n_k of the largest eigenvalues, and corresponding eigenvectors, of $\tilde{Q}_k(A_0)$ in (4.17). The resulting n_k eigenpairs $\{\lambda_k^i, \mathbf{u}_k^i\}$ $(i = 1, \ldots, n_k)$ are required in order to construct a limited-memory approximation to $\tilde{Q}_k^{-1}(A_0)$ or $\tilde{Q}_k^{-1/2}(A_0)$ of the form defined in (4.1). Specifically,

$$\hat{Q}_k^{-1}(A_0) = I_k + \sum_{i=1}^{n_k} ((\lambda_k^i)^{-1} - 1)\mathbf{u}_k^i(\mathbf{u}_k^i)^T \tag{4.18}$$

and

$$\hat{Q}_k^{-1/2}(A_0) = I_k + \sum_{i=1}^{n_k} ((\lambda_k^i)^{-1/2} - 1)\mathbf{u}_k^i(\mathbf{u}_k^i)^T. \tag{4.19}$$

It therefore follows from (4.17), (4.18), and (4.19) that

$$Q_k^{-1}(A_0) = Z_{k+1}^k \tilde{Q}_k^{-1}(A_0)(Z_{k+1}^k)^T \simeq Z_{k+1}^k \hat{Q}_k^{-1}(A_0)(Z_{k+1}^k)^T \tag{4.20}$$

and

$$Q_k^{-1/2}(A_0) = Z_{k+1}^k \tilde{Q}_k^{-1/2}(A_0) \simeq Z_{k+1}^k \hat{Q}_k^{-1/2}(A_0). \tag{4.21}$$

An approximation to $Q_k^{-1}(A_0)\mathbf{w}_k$ or $Q_k^{-1/2}(A_0)\mathbf{w}_k$, for any given vector $\mathbf{w}_k \in \mathbb{R}^{m_k}$ defined at grid level k, is therefore computable using (4.20) or (4.21).

The procedure on the finest grid at grid level $k = 0$ differs since A_0 is available in the form $A_0\mathbf{w}_0$. Thus $Q_0(A_0) \equiv A_0$ in this case. The first step is then to precondition A_0 directly to obtain

$$\tilde{Q}_0(A_0) = (Z_1^0)^T A_0 Z_1^0. \tag{4.22}$$

The Lanczos method is used to compute estimates of a specified number n_0 of the largest eigenvalues, and corresponding eigenvectors, of $\tilde{Q}_0(A_0)$ in (4.22). The resulting n_0 eigenpairs $\{\lambda_0^i, \mathbf{u}_0^i\}$ $(i = 1, \ldots, n_0)$ are required in order to construct a limited-memory approximation to $\tilde{Q}_0^{-1}(A_0)$ or $\tilde{Q}_0^{-1/2}(A_0)$ of the form defined in (4.18) or (4.19). An approximation to $A_0^{-1}\mathbf{w}_0$ can be computed at this stage by means of (4.20). That is,

$$A_0^{-1}\mathbf{w}_0 \simeq Q_0^{-1}(A_0)\mathbf{w}_0 = Z_1^0 \hat{Q}_0^{-1}(A_0)(Z_1^0)^T\mathbf{w}_0. \tag{4.23}$$

An approximation to $A_0^{-1/2}\mathbf{w}_0$, if required, can be computed using (4.21). Specifically,

$$A_0^{-1/2}\mathbf{w}_0 \simeq Q_0^{-1/2}(A_0)\mathbf{w}_0 = Z_1^0 \hat{Q}_0^{-1/2}(A_0)\mathbf{w}_0 \tag{4.24}$$

where $\hat{Q}_0^{-1/2}(A_0)$ has the form (4.19).

In what follows, it is useful to represent the parameters n_k $(k = 0, \ldots, k_c)$ in vector form by setting

$$N_e = (n_0, n_1, \ldots, n_{k_c}), \quad \hat{N}_e = \sum_{k=0}^{k_c} n_k \tag{4.25}$$

where $\hat{N}_e$ is the sum of the entries in N_e. The accuracy of an approximation to $A_0^{-1}\mathbf{w}_0$ or $A_0^{-1/2}\mathbf{w}_0$ constructed using the multilevel eigenvalue decomposition algorithm is dependent on the choice of N_e in (4.25). That is, the values n_k in N_e are key in determining the quality of the approximation obtained. This will be explored later.

4.4.3 Preconditioning strategy

The preconditioner Z_{k+1}^k in (4.17) is defined based on the assumption that $Q_{k+1}(A_0)$ is a good approximation to $Q_k(A_0)$. If this is the case then, recalling definition (4.14), it

is reasonable to expect that $S_k(Q_{k+1}^{-1}(A_0))$ can be used to precondition $Q_k(A_0)$. This expectation will be considered for a particular example in Section 4.9. Specifically, we expect the matrix

$$\overline{Q}_k(A_0) \simeq S_k(Q_{k+1}^{-\frac{1}{2}}(A_0))Q_k(A_0)S_k(Q_{k+1}^{-\frac{1}{2}}(A_0)) \tag{4.26}$$

to have eigenvalues clustered more closely around unity. In practice, the preconditioner Z_{k+1}^k at grid level k is constructed using information prolonged from the next coarsest grid level $k+1$. The procedure on the coarsest grid at grid level $k = k_c$ is to set

$$Z_{k_c+1}^{k_c} = I_{k_c}$$

since grid level $k = k_c + 1$ does not exist. At other grid levels, the preconditioner Z_{k+1}^k and its transpose $(Z_{k+1}^k)^T$ are defined as follows:

$$Z_{k+1}^k = \begin{cases} S_k(Z_{k+2}^{k+1}\hat{Q}_{k+1}^{-1/2}(A_0)), & \text{if } k = 0, 1, \ldots, k_c - 1 \\ I_{k_c}, & \text{if } k = k_c \end{cases}, \tag{4.27}$$

$$(Z_{k+1}^k)^T = \begin{cases} S_k(\hat{Q}_{k+1}^{-1/2}(A_0)(Z_{k+2}^{k+1})^T), & \text{if } k = 0, 1, \ldots, k_c - 1 \\ I_{k_c}, & \text{if } k = k_c \end{cases} \tag{4.28}$$

where $\hat{Q}_{k+1}^{-1/2}(A_0)$ is of the form defined in (4.19).

4.4.4 Summary

The multilevel eigenvalue decomposition algorithm described in Section 4.4.2 is outlined in Figure 4.1. Note that the matrices involved in this procedure are not explicitly constructed in practice. Specifically, only matrix vector products are computed. The inputs to the multilevel eigenvalue decomposition algorithm are A_0 in the form of $A_0\mathbf{w}_0$, where $\mathbf{w}_0 \in \mathbb{R}^{m_0}$ denotes any given vector defined at grid level $k = 0$, and N_e in (4.25). The outputs are the following two vectors

$$\Lambda_0 = [\lambda_{k_c}^1, \ldots, \lambda_{k_c}^{n_{k_c}}, \lambda_{k_c-1}^1, \ldots, \lambda_{k_c-1}^{n_{k_c-1}}, \ldots, \lambda_0^1, \ldots, \lambda_0^{n_0}], \tag{4.29}$$

1: $[\Lambda_0, U_0] = MLEVD(A_0, N_e)$
2: **for** $k = k_c, k_c - 1, \ldots, 0$ **do**
3: Construct $Q_k(A_0)$ in (4.16)
4: **if** $k = k_c$ **then**
5: Set $Z_{k_c+1}^{k_c} = I_{k_c}$ and $(Z_{k_c+1}^{k_c})^T = I_{k_c}$
6: **else**
7: Construct Z_{k+1}^k in (4.27) and $(Z_{k+1}^k)^T$ in (4.28)
8: **end if**
9: Precondition $Q_k(A_0)$ to obtain $\tilde{Q}_k(A_0)$ in (4.17)
10: Compute $\{\lambda_k^i, \mathbf{u}_k^i\}$ $(i = 1, \ldots, n_k)$ by applying the Lanczos method to $\tilde{Q}_k(A_0)$ and store in memory
11: **end for**

Figure 4.1: Outline of the multilevel eigenvalue decomposition algorithm.

$$U_0 = [\mathbf{u}_{k_c}^1, \ldots, \mathbf{u}_{k_c}^{n_{k_c}}, \mathbf{u}_{k_c-1}^1, \ldots, \mathbf{u}_{k_c-1}^{n_{k_c-1}}, \ldots, \mathbf{u}_0^1, \ldots, \mathbf{u}_0^{n_0}]. \tag{4.30}$$

The entries of Λ_0 in (4.29) are the eigenvalue estimates λ_k^i $(i = 1, \ldots, n_k)$ of $\tilde{Q}_k(A_0)$ in (4.17) computed at grid levels $k = 0, \ldots, k_c$. The corresponding eigenvectors $\mathbf{u}_k^i$ $(i = 1, \ldots, n_k)$ are contained in U_0 in (4.30). The vectors Λ_0 and U_0 in are constructed as the procedure progresses from the coarsest grid level $k = k_c$ to the finest grid level $k = 0$. An approximation to $A_0^{-1}\mathbf{w}_0$ or $A_0^{-1/2}\mathbf{w}_0$ can be computed on termination of the multilevel eigenvalue decomposition algorithm by means of (4.23) or (4.24) using Λ_0 and U_0.

The vector Λ_0 in (4.29) contains a total of $\hat{N}_e$ entries, where $\hat{N}_e$ is defined in (4.25). The total number of entries in U_0 in (4.30) is therefore

$$\sum_{k=0}^{k_c} n_k m_k = \left(\sum_{k=0}^{k_c} \frac{n_k}{2^k} \right) m + \hat{N}_e$$

since $m_k = m/2^k + 1$. Given that the finest grid at grid level $k = 0$ contains $m_0 = m + 1$ grid points, an estimate of the ratio of memory required for the multilevel eigenvalue decomposition algorithm in terms of m is obtained by calculating

$$R_e = \sum_{k=0}^{k_c} \frac{n_k}{2^k}. \tag{4.31}$$

Note that the memory ratio R_e in (4.31) is dependent on N_e in (4.25). This quantity

will play a key role in the numerical study presented in Section 4.12.

4.5 Approximating the inverse Hessian

The multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 is proposed as a suitable method for constructing a limited-memory approximation to the inverse (and inverse square root) of any given symmetric positive definite matrix $A \in \mathbb{R}^{N\times N}$ with eigenvalues clustered around unity. It is assumed that A can be approximated using a specified number of its leading eigenvalues, and corresponding eigenvectors, by means of a limited-memory approximation of the form defined in (4.1). The particular case of the matrix A considered in the remainder of this chapter is the Hessian C in (2.50). The numerical studies presented in Sections 4.10-4.12 involve approximating the inverse Hessian for the specific model problem outlined in Section 4.6.

4.6 Model problem

Burgers' equation is often used in data assimilation applications as a simple model characterising elements of atmospheric flow; see, for example, [42]. The model problem implemented in the numerical studies presented in this chapter is a one-dimensional Burgers' equation incorporating a non-linear viscous term. That is,

$$\frac{\partial\varphi}{\partial t} + \frac{1}{2}\frac{\partial}{\partial x}(\varphi^2) = \frac{\partial}{\partial x}\left(\mu(\varphi)\frac{\partial\varphi}{\partial x}\right), \quad \varphi = \varphi(x,t), \quad x \in (0,1), \quad t \in (0,T) \tag{4.32}$$

subject to the Neumann boundary conditions

$$\left.\frac{\partial\varphi}{\partial x}\right|_{x=0} = 0, \quad \left.\frac{\partial\varphi}{\partial x}\right|_{x=1} = 0 \tag{4.33}$$

with viscosity coefficient

$$\mu(\varphi) = \rho_0 + \rho_1\left(\frac{\partial\varphi}{\partial x}\right)^2 \tag{4.34}$$

where ρ_0 and ρ_1 are positive constants. The particular values used in this thesis are $\rho_0 = 10^{-4}$ and $\rho_1 = 10^{-5}$ in (4.34) for all computations. The tangent linear problem associated with (4.32)-(4.34) is presented in Appendix A. The initial condition for

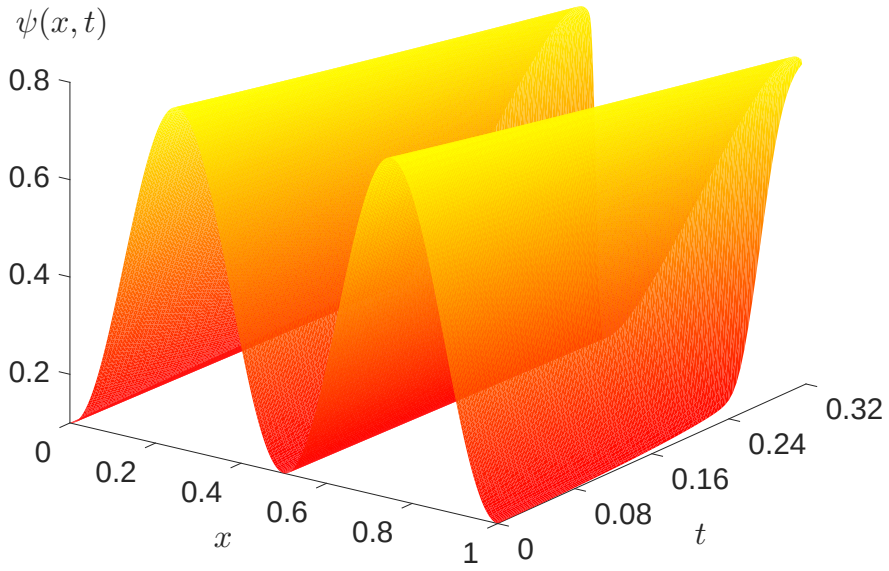


Figure 4.2: Flow evolution of $\varphi(x,t)$ satisfying (4.32)-(4.34) for initial condition $\varphi_1(x,0)$ in (4.35).

(4.32)-(4.34) is defined as follows:

$$\varphi_1(x,0) = 0.1 + 0.35\left[1 + \sin\left(4\pi x + \frac{3\pi}{2}\right)\right], \quad (0 < x < 1)\,. \tag{4.35}$$

The flow evolution of $\varphi(x,t)$ satisfying (4.32)-(4.34) where $\varphi(x,0) = \varphi_1(x,0)$ is plotted in Figure 4.2.

An implicit time discretisation is used, and the power law first-order finite volume scheme for spatial discretisation; details can be found in [99]. The procedure at each time step is to perform non-linear iterations to converge on non-linear coefficients. The observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ in (2.2) included in the 4D-Var cost function $\mathcal{J}(\mathbf{x}_0)$ in (2.4) are typically provided by sensors. The observational data for this model problem is generated randomly based on the sensor configuration scheme S_1. This comprises of seven stationary sensors. A stationary sensor has a fixed position in space and provides observational data captured at this point. That is, $x_j(t) = x_j(0)$ $(i = 1, \ldots, 7)$ for all $t \in [0, T]$, where $x_j(t)$ denotes the position of sensor j in the spatial domain Ω at time t. Note that $\Omega = [0, 1]$. Specifically, $x_1(t) = 0.3$, $x_2(t) = 0.4$, $x_3(t) = 0.45$, $x_4(t) = 0.5$, $x_5(t) = 0.55$, $x_6(t) = 0.6$, and $x_7(t) = 0.7$ for all $t \in [0, T]$ in this case.

4.7 Implementation details

The model problem outlined in Section 4.6 is implemented in FORTRAN. The tangent linear and adjoint models associated with (4.32)-(4.34) are generated using the automatic differentiation engine TAPENADE; see [58]. For this model problem, the multilevel construction described in Section 4.1 consists of four grids. The finest grid at grid level $k = 0$ contains $m_0 = 401$ grid points and the coarsest grid at grid level $k = 3$ is composed of $m_3 = 51$ grid points. The interpolation matrix P_k^{k-i} in (4.2) is constructed using cubic spline interpolation. This choice of interpolation method is discussed in Section 4.10. The restriction matrix $(P_k^{k-i})^T$ in (4.3) is generated using TAPENADE.

The background error covariance matrix B in (2.4) is defined based on the assumption that the background error $\boldsymbol{\delta}_b$ in (2.3) belongs to the Sobelev space $W_2^2[0,1]$; see [45, §5.1] for details. The observation error $\boldsymbol{\delta}_o^i$ in (2.2) is Gaussian and uncorrelated with standard deviation $\sigma_o = 0.016$. The observation error covariance matrix R_i in (2.4) is diagonal with uniform variance $(R_i)_{j,j} = \sigma_o^2$.

The eigenvalue problems involving $\tilde{Q}_k(A_0)$ $(k = 0, 1, 2, 3)$ in (4.17) are solved using the standard version of ARPACK; see [78]. Specifically, estimates of the n_k largest, in terms of magnitude, eigenvalues of $\tilde{Q}_k(A_0)$, and the corresponding eigenvectors, are computed using ARPACK. The particular technique applied is the implicitly restarted Lanczos method (see Section 3.3). The default tolerance used in the convergence criterion in (3.41) in ARPACK is $t_e = 10^{-2}$. A modified version of ARPACK is considered in Section 4.11. The eigenvalue computations must be conducted in double precision. However, it is acceptable to store the resulting eigenpairs in single precision. As the eigenpairs computed at grid level k $(k = 1, 2, 3)$ in the multilevel construction are used to precondition $Q_{k-1}(A_0)$ in (4.16) at grid level $k - 1$, any round-off errors incurred at grid level k are mostly compensated for at grid level $k - 1$.

4.8 Evaluating the accuracy of approximations using the Riemannian distance

A numerical measure is required to evaluate the accuracy of approximations constructed using the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1. The Riemannian distance is used in the numerical studies presented in this chapter; see, for example, [86]. Let $G_1, G_2 \in \mathbb{R}^{N\times N}$ denote symmetric positive definite matrices. The Riemannian distance between G_1 and G_2 is defined as follows:

$$\delta_R(G_1, G_2) = \| \ln(G_1^{-1}G_2)\|_F = \left(\sum_{i=1}^{N} \ln^2(\xi_i) \right)^{1/2} \tag{4.36}$$

where $\|.\|_F$ represents the Frobenius norm and ξ_i $(i = 1, \ldots, N)$ are the eigenvalues of $G_1^{-1}G_2$. A statistical interpretation of the Riemannian distance is as a symmetric measure of the difference between two Gaussian probability distributions having equal modes. As the inverse Hessian can be used as an approximation to the analysis error covariance matrix in 4D-Var, it is appropriate to use the Riemannian distance to measure the difference between the inverse Hessian and an approximation to this matrix constructed using the multilevel eigenvalue decomposition algorithm. An important property is that $\delta_R(G_1, G_2)$ in (4.36) remains unchanged on applying symmetric preconditioning to the input matrices G_1 and G_2. This is relevant since applying the control variable transform discussed in Section 2.9 implicitly preconditions the Hessian F defined in Section 2.8 symmetrically with $B^{1/2}$, that is, the symmetric square root of the background error covariance matrix B. Specifically,

$$B^{1/2}FB^{1/2} = B^{1/2}(B^{-1} + \hat{H}^T\hat{R}^{-1}\hat{H})B^{1/2} = I_N + B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{H}B^{1/2} = C \tag{4.37}$$

where C is the Hessian defined in (2.50). In terms of the Riemannian distance, this means that

$$\delta_R(\tilde{F}, F) = \delta_R(\tilde{C}, C) \tag{4.38}$$

where $\tilde{F} \in \mathbb{R}^{N\times N}$ denotes a symmetric positive definite matrix and $\tilde{C} = B^{1/2}\tilde{F}B^{1/2}$.

Let $A_0 \in \mathbb{R}^{401\times 401}$ denote the Hessian C in (2.50) for the specific model problem

outlined in Section 4.6. This is defined on the finest grid at grid level $k = 0$ in the multilevel construction described in Section 4.1, and is available in the form of $A_0\mathbf{w}_0$, where $\mathbf{w}_0 \in \mathbb{R}^{401}$ denotes any given vector defined at grid level $k = 0$. Note that, since we are dealing with a small model problem, A_0^{-1} is also available. This is used to evaluate the accuracy of approximations to A_0^{-1} constructed using the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1. Let $\tilde{A}_0^{-1} \in \mathbb{R}^{401\times 401}$ denote an approximation generated in this way. The cases of $\tilde{A}_0^{-1}$ considered in this chapter are symmetric positive definite. The accuracy of $\tilde{A}_0^{-1}$ is evaluated in the numerical studies presented in Sections 4.10-4.12 by calculating the normalised Riemannian distance

$$D_e = \frac{\delta_R(\tilde{A}_0^{-1}, A_0^{-1})}{\delta_R(I_0, A_0^{-1})} \tag{4.39}$$

where I_0 is the 401×401 identity matrix associated with the finest grid level $k = 0$ in the multilevel construction. The accuracy of $\tilde{A}_0^{-1}$ is dependent on N_e in (4.25). The ideal outcome would be to identify the optimal N_e for any given problem. However, this is non-trivial in practice. For instance, it may be desirable to construct the best possible approximation given a fixed amount of allocated memory. Alternatively, it may be necessary to determine the approximation that results in the largest reduction in computational cost to construct $\tilde{A}_0^{-1}\mathbf{w}_0$. The approach adopted in the numerical study presented in Section 4.12 is to fix the memory allocated to the multilevel eigenvalue decomposition algorithm. This involves fixing R_e in (4.31) and specifying N_e in (4.25) accordingly. The ARPACK software discussed in Section 4.7 is used in the numerical studies presented in this chapter to compute estimates of a specified number n_k of the largest eigenvalues, and corresponding eigenvectors, of $\tilde{Q}_k(A_0)$ in (4.17). This procedure involves computing Hessian vector products of the form $A_0\mathbf{w}_0$ on the finest grid at grid level $k = 0$ in the multilevel construction. A Hessian vector product evaluation is computationally demanding since it requires the tangent linear and adjoint models associated with (4.32)-(4.34) to be executed. The total number of Hessian vector products computed at the finest grid level $k = 0$ is evaluated in the numerical studies

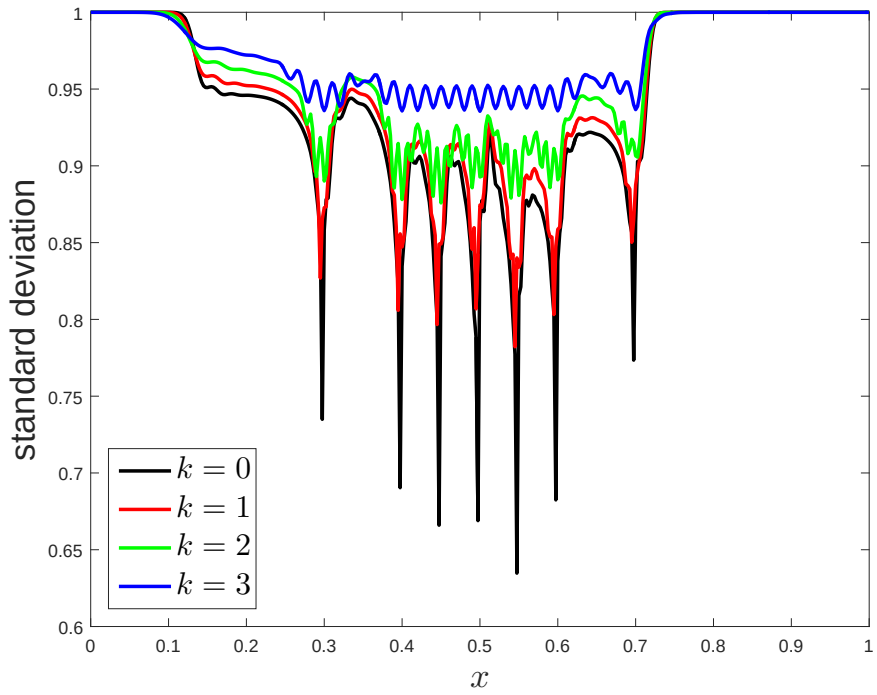


Figure 4.3: Standard deviation $\boldsymbol{\sigma}_k$ plotted as a function of x on $\Omega = [0, 1]$.

presented in this chapter by calculating

$$M_e = \sum_{k=0}^{3} \tilde{n}_k \tag{4.40}$$

where $\tilde{n}_k$ denotes the number of Hessian vector products computed at the finest grid level $k = 0$ in order to solve the eigenvalue problem involving $\tilde{Q}_k(A_0)$ in (4.17) at grid level k $(k = 0, 1, 2, 3)$.

4.9 Choice of preconditioner

The choice of preconditioner Z_{k+1}^k in (4.17) applied at grid level k $(k = 0, \ldots, k_c)$ in the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 is considered for the case of the Hessian C associated with the model problem outlined in Section 4.6. The inverse Hessian is interpreted here as an approximation to the analysis error covariance matrix. Let $C_k \in \mathbb{R}^{m_k \times m_k}$ denote a representation of the matrix C at a coarser grid level k $(k = 1, 2, 3)$ (this will be defined precisely in (5.5)).

The standard deviation pertaining to C_k^{-1} is defined here as the vector $\boldsymbol{\sigma}_k \in \mathbb{R}^{401}$

containing the square roots of the diagonal entries of $S_0(C_k^{-1})$, using the grid transfer operator $S_{k-i}(.)$ in (4.14). Specifically, the ith entry of $\boldsymbol{\sigma}_k$ is defined as follows:

$$(\boldsymbol{\sigma}_k)_i = \sqrt{\left(S_0(C_k^{-1})\right)_{i,i}} \quad (i = 1, \ldots, 401) \tag{4.41}$$

where $\left(S_0(C_k^{-1})\right)_{i,i}$ denotes the (i,i)th entry of $S_0(C_k^{-1})$. The standard deviation $\boldsymbol{\sigma}_0$ associated with C_0^{-1} is used as a reference function for comparing $\boldsymbol{\sigma}_k$. The plot in Figure 4.3 shows $\boldsymbol{\sigma}_k$ plotted as a function of x on $\Omega = [0, 1]$. The key observation based on the plot in Figure 4.3 is that $\boldsymbol{\sigma}_{k+1}$ captures the fundamental components of $\boldsymbol{\sigma}_k$. Thus it is reasonable to expect that $\boldsymbol{\sigma}_{k+1}$ is a good approximation to $\boldsymbol{\sigma}_k$. This substantiates the idea discussed in Section 4.4.3 that $S_k(Q_{k+1}^{-1}(A_0))$, using the grid transfer operator $Q_k(.)$ in (4.15), is a good choice of preconditioner for $Q_k(A_0)$.

4.10 Choice of interpolation method

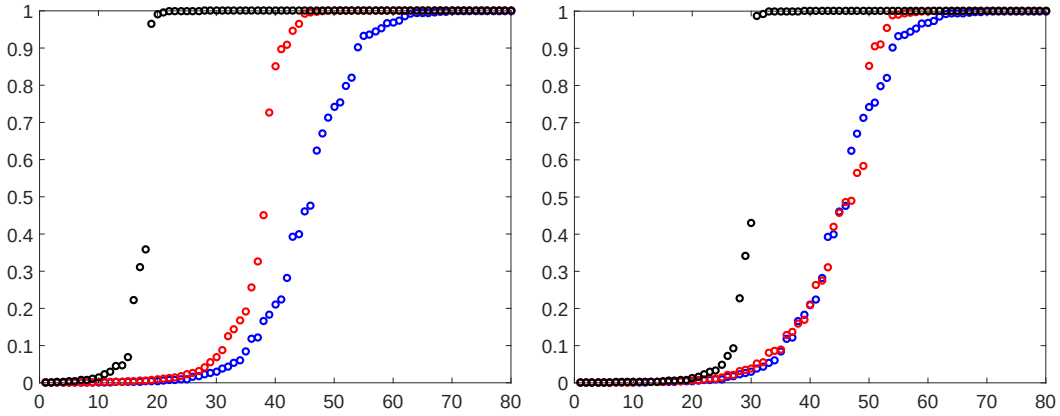
An interpolation method is required to implement the interpolation operator P_k^{k-i} in (4.2). The two methods considered in this section are linear and cubic spline interpolation. These methods are compared by evaluating the accuracy of $\tilde{A}_0^{-1}$ in each case using D_e in (4.39). Note that M_e in (4.40) is also calculated. This provides an indication of computational cost.

The results obtained for four combinations of R_e in (4.31) and N_e in (4.25) are tabulated in Table 4.1. Note that the choices of values used in N_e will be discussed in Section 4.12. The key observation based on the results presented in Table 4.1 is that implementing P_k^{k-i} using linear interpolation resulted in larger values of D_e, that is, less accurate approximations to A_0^{-1}, in all instances. However, this approach involved computing a smaller number of Hessian vector products at the finest grid level $k = 0$ in almost all cases (excluding $R_e = 8$), as shown by the values of M_e tabulated.

The eigenvalues of A_0^{-1} and $\tilde{A}_0^{-1}$ for two combinations of R_e and N_e presented in Table 4.1 are plotted in Figure 4.4. The plots in subfigures 4.4(a) and 4.4(b) relate to the cases of $R_e = 8$ with $N_e = (0, 0, 16, 32)$ and $R_e = 16$ with $N_e = (4, 8, 16, 32)$, respectively. The first eighty eigenvalues of A_0^{-1} are plotted in each case using blue circles. The remaining eigenvalues of A_0^{-1} are close to one and have been omitted. The

		D_e		M_e	
R_e	N_e	Linear	Cubic spline	Linear	Cubic spline
8	$(0, 0, 16, 32)$	$9.6358e-1$	$2.7997e-1$	69	62
12	$(0, 8, 16, 32)$	$7.8938e-1$	$2.0623e-1$	89	116
16	$(4, 8, 16, 32)$	$7.1718e-1$	$1.7247e-1$	108	159
24	$(6, 12, 24, 48)$	$5.4323e-1$	$1.2217e-1$	210	262

Table 4.1: The values D_e and M_e obtained for four combinations of R_e and N_e, respectively, with P_k^{k-i} implemented using linear or cubic spline interpolation.



(a) $R_e = 8,\ N_e = (0, 0, 16, 32)$ (b) $R_e = 16,\ N_e = (4, 8, 16, 32)$

Figure 4.4: Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles), $\tilde{A}_0^{-1}$ where P_k^{k-i} has been implemented using cubic spline interpolation (red circles), and $\tilde{A}_0^{-1}$ where P_k^{k-i} has been implemented using linear interpolation (black circles). Two combinations of R_e and N_e presented in Table 4.1 are highlighted.

first eighty eigenvalues of $\tilde{A}_0^{-1}$ where P_k^{k-i} in (4.2) has been implemented using cubic spline interpolation are plotted using red circles. The first eighty eigenvalues of $\tilde{A}_0^{-1}$ where P_k^{k-i} has been implemented using linear interpolation are plotted using black circles. The eigenvalues of $\tilde{A}_0^{-1}$ in the plots in subfigures 4.4(a) and 4.4(b) are closer to the eigenvalues of A_0^{-1} with P_k^{k-i} implemented using cubic spline interpolation. This corresponds to the smaller distances D_e tabulated in Table 4.1.

The results presented demonstrate that implementing P_k^{k-i} using linear interpolation produces a less accurate approximation to A_0^{-1} than achievable by employing cubic spline interpolation. However, constructing an approximation to A_0^{-1} with P_k^{k-i}

implemented using linear interpolation has been shown to be less computationally expensive in almost all cases studied, as substantiated by the values M_e obtained in these instances. This indicates that employing linear interpolation is a viable strategy for reducing the computational cost. However, as we are interested in obtaining more accurate approximations, cubic spline interpolation is used in the numerical studies presented in the remainder of this thesis.

4.11 Executing one shift in ARPACK

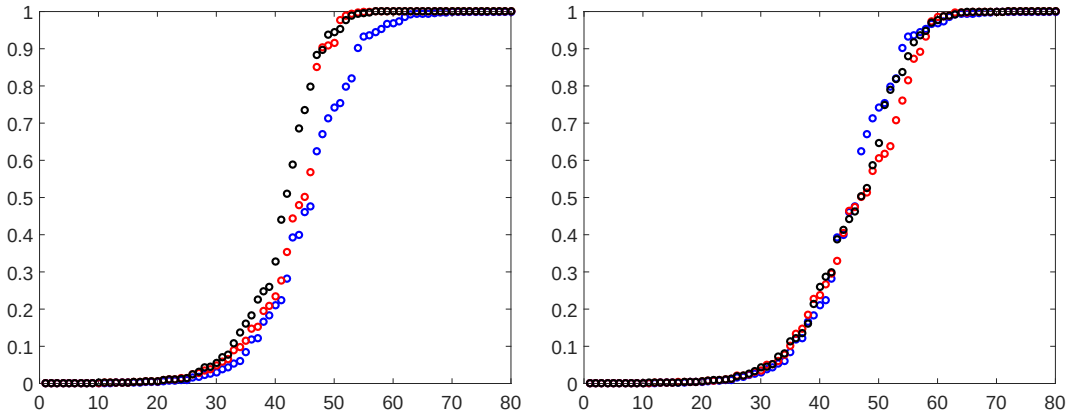
The standard version of ARPACK is used to solve the eigenvalue problems involving $\tilde{Q}_k(A_0)$ $(k = 0, 1, 2, 3)$ in (4.17). The implicitly restarted Lanczos method is the specific technique applied (see Section 3.3). The procedure employed involves executing the necessary number of shifts such that the convergence criterion in (3.41) is satisfied for each of the n_k requested eigenvalue estimates. In this section, a modified version of ARPACK that permits only one shift to be executed is considered. This is compared with the standard version of ARPACK by evaluating the accuracy of $\tilde{A}_0^{-1}$ in each case using D_e in (4.39). Note that M_e in (4.40) is also calculated. This provides an indication of computational cost.

The results obtained for the same four combinations of R_e in (4.31) and N_e in (4.25) presented in Table 4.1 are tabulated in Table 4.2. The key observation based on the results presented is that applying the proposed modified version of ARPACK resulted in larger values of D_e, that is, less accurate approximations to A_0^{-1}, in all instances. However, this approach involved computing a smaller number of Hessian vector products at the finest grid level $k = 0$ in all cases, as evidenced by the results M_e attained.

The eigenvalues of A_0^{-1} and $\tilde{A}_0^{-1}$ for two R_e and N_e combinations presented in Table 4.2 are plotted in Figure 4.5. The plots in subfigures 4.5(a) and 4.5(b) relate to the cases of $R_e = 12$ with $N_e = (0, 8, 16, 32)$ and $R_e = 24$ with $N_e = (6, 12, 24, 48)$, respectively. The first eighty eigenvalues of A_0^{-1} are plotted in each case using blue circles. The remaining eigenvalues of A_0^{-1} are close to one and have been omitted. The first eighty eigenvalues of $\tilde{A}_0^{-1}$ where the standard version of ARPACK has been applied

		D_e		M_e	
R_e	N_e	Standard	Modified	Standard	Modified
8	$(0, 0, 16, 32)$	$2.7997e-1$	$3.0768e-1$	62	50
12	$(0, 8, 16, 32)$	$2.0623e-1$	$2.3901e-1$	116	59
16	$(4, 8, 16, 32)$	$1.7247e-1$	$2.2543e-1$	159	64
24	$(6, 12, 24, 48)$	$1.2217e-1$	$1.7726e-1$	262	94

Table 4.2: The values D_e and M_e obtained for four combinations of R_e and N_e, respectively, with the standard or proposed modified version of ARPACK applied.



(a) $R_e = 12$, $N_e = (0, 8, 16, 32)$ (b) $R_e = 24$, $N_e = (6, 12, 24, 48)$

Figure 4.5: Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles), $\tilde{A}_0^{-1}$ where the standard version of ARPACK has been applied (red circles), and $\tilde{A}_0^{-1}$ where the proposed modified version of ARPACK has been applied (black circles). Two combinations of R_e and N_e presented in Table 4.2 are highlighted.

are plotted using red circles. The first eighty eigenvalues of $\tilde{A}_0^{-1}$ where the proposed modified version of ARPACK has been applied are plotted using black circles. The eigenvalues of $\tilde{A}_0^{-1}$ in the plots in subfigures 4.5(a) and 4.5(b) with the two versions of ARPACK applied have similar distributions, which corresponds to the small differences in D_e shown in Table 4.2.

The results presented demonstrate that applying the proposed modified version of ARPACK produces a less accurate approximation to A_0^{-1} than achievable using the standard version of ARPACK. However, constructing an approximation to A_0^{-1} with the proposed modified version of ARPACK applied has been shown to be less computationally expensive in all cases studied, as evidenced by the results M_e attained.

This indicates that applying the proposed modified version of ARPACK is a viable strategy for reducing the computational cost. However, as we are interested in obtaining more accurate approximations, the standard version of ARPACK is applied in the numerical studies presented in the remainder of this thesis.

4.12 Comparing approximations to the inverse Hessian

The numerical study presented here focuses on the choice of N_e in (4.25), that is, how many eigenpairs to calculate at each grid level in the framework of the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1. Specifically, various cases of $\tilde{A}_0^{-1}$ are considered. The accuracy of $\tilde{A}_0^{-1}$ is evaluated in each case using D_e in (4.39), with M_e in (4.40) also being calculated to provide an indication of computational cost.

We first suppose that there are no memory restrictions in place. The maximum allowable case of R_e in (4.31) given the model problem setup is $R_e = 400$, so we can set $N_e = (400, 0, 0, 0)$. That is, all eigenvalues (excluding one) of $\tilde{Q}_0(A_0)$ in (4.22) are computed on the finest grid at grid level $k = 0$ (note that this is the maximum number of eigenvalues computable at grid level $k = 0$ using ARPACK). The output $\tilde{A}_0^{-1}$ in this case is therefore the best possible approximation to A_0^{-1} achievable using the multilevel eigenvalue decomposition algorithm. With this choice of N_e, $D_e = 5.5185e - 13$ and $M_e = 401$. The eigenvalues of A_0^{-1} and $\tilde{A}_0^{-1}$ are plotted in Figure 4.6. The first eighty eigenvalues of $\tilde{A}_0^{-1}$ are plotted using red circles. The remaining eigenvalues of $\tilde{A}_0^{-1}$ are close to one and have been omitted. The first eighty eigenvalues of A_0^{-1} are also plotted using blue circles. However, these eigenvalues are indistinguishable from the eigenvalues of $\tilde{A}_0^{-1}$ in the plot in Figure 4.6, as $\tilde{A}_0^{-1}$ is an excellent approximation to A_0^{-1}. This is supported by the very small value of D_e attained. The result M_e obtained is indicative of a high computational cost as expected.

Suppose now instead that the maximum allowable case of R_e is $R_e = 24$. We now consider every combination of R_e and N_e within this fixed memory framework, and calculate the resulting values of D_e. Figure 4.7 shows D_e plotted against R_e. The dashed blue line highlights the true minimum D_e attained. The solid blue line represents the average of D_e calculated with respect to the 5% of cases of N_e that resulted in the

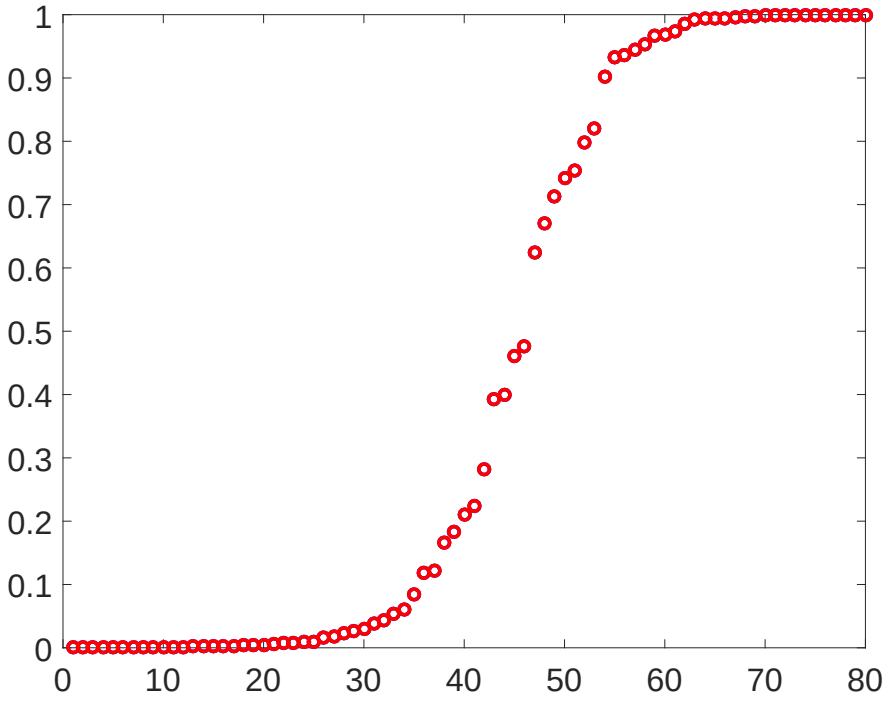


Figure 4.6: Comparison plot showing the first eighty eigenvalues of $\tilde{A}_0^{-1}$ (red circles). The first eighty eigenvalues of A_0^{-1} plotted using blue circles are indistinguishable from the eigenvalues of $\tilde{A}_0^{-1}$. The combination implemented is $R_e = 400$ and $N_e = (400, 0, 0, 0)$. The associated measures of accuracy and computational cost are $D_e = 5.5185e - 13$ and $M_e = 401$, respectively.

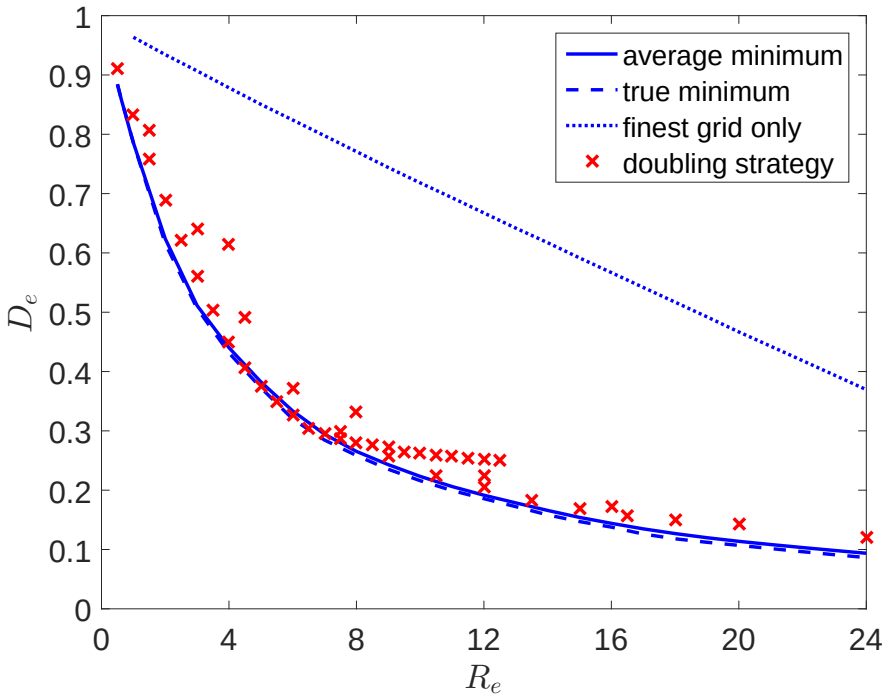


Figure 4.7: Comparison plot showing D_e plotted against R_e where the maximum allowable memory ratio is $R_e = 24$. The average of D_e calculated with respect to the 5% of cases of N_e that resulted in the lowest D_e (solid blue line). The true minimum D_e attained (dashed blue line). The result D_e obtained where the case of N_e implemented involved only the finest grid at grid level $k = 0$ (dotted blue line). The value D_e obtained where the case of N_e implemented resulted from applying the doubling strategy (red crosses).

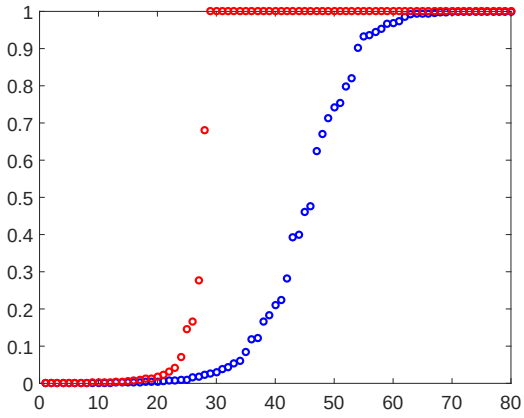
lowest D_e. This is included in order to show that the true minimum is not an outlier. The dotted blue line shows D_e where N_e involved only the finest grid at grid level $k = 0$, that is, $N_e = (R_e, 0, 0, 0)$. The red crosses represent D_e where $N_e = (n_0, 2n_0, 4n_0, 8n_0)$, for some n_0. We refer to this approach for specifying N_e as a doubling strategy. For our small example, the particular case of N_e that results in the minimum D_e for a given choice of R_e was determined by trying all possible combinations. However, this approach is obviously not feasible in practice. The doubling strategy introduced is proposed as an alternative method for specifying N_e in general.

A number of key observations can be made based on the plot in Figure 4.7. The first observation is that D_e decreased as R_e increased in almost all instances. That is, as R_e increased, the accuracy of $\tilde{A}_0^{-1}$ also increased. It is therefore expected that $D_e \to 0$ as $R_e \to \infty$. The second observation is that involving only the finest grid

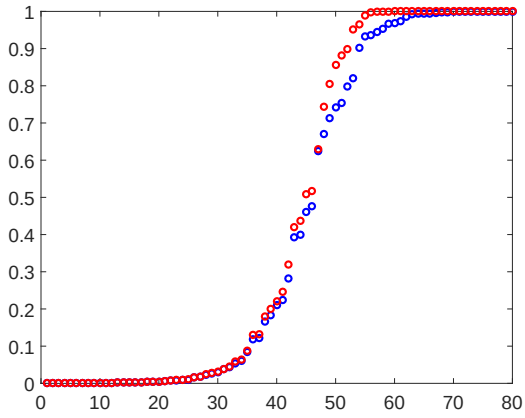
in N_e resulted in the largest D_e, namely, the least accurate approximation to A_0^{-1}, in every case of R_e studied. Of course, given unlimited memory, using only the fine grid eigenvalues would give the most accurate approximation, as discussed at the beginning of this section. The final observation is that implementing the doubling strategy failed to produce the true minimum D_e, that is, the most accurate approximation to A_0^{-1}, in every case of R_e considered. However, applying this method resulted in at least one reasonably accurate approximation to A_0^{-1} in all cases, as substantiated by the values D_e obtained.

The eigenvalues of A_0^{-1} and $\tilde{A}_0^{-1}$ for various cases of R_e included in the plot in Figure 4.7 are plotted in Figures 4.8 and 4.9. The specific case of N_e implemented in each instance is highlighted. Note that the particular combinations of R_e and N_e considered are selected from Tables 4.4, 4.5, and 4.6. The value of D_e obtained for each R_e and N_e combination is presented in order to demonstrate the accuracy of $\tilde{A}_0^{-1}$. The approach adopted is the same as that employed in the plot in Figure 4.6. That is, the first eighty eigenvalues of A_0^{-1} and $\tilde{A}_0^{-1}$ are plotted using blue and red circles, respectively. The plots in subfigures 4.8(a), 4.8(c), and 4.8(e) correspond to the case of $R_e = 4$. The plots in subfigures 4.8(b), 4.8(d), and 4.8(f) relate to the case of $R_e = 24$. The cases of N_e pertaining to the plots in subfigures 4.8(a) and 4.8(b) resulted in the true minimum D_e. The cases of N_e corresponding to the plots in subfigures 4.8(c) and 4.8(d) resulted from applying the doubling strategy. The cases of N_e relating to the plots in subfigures 4.8(e) and 4.8(f) involve only the finest grid at grid level $k = 0$. The first observation is that the eigenvalues of $\tilde{A}_0^{-1}$ are closest to the eigenvalues of A_0^{-1} in the plots in subfigures 4.8(a) and 4.8(b), as expected, with the smallest values of D_e obtained in each case. Note that the eigenvalues of $\tilde{A}_0^{-1}$ in the plots in subfigures 4.8(a) and 4.8(c) have similar distributions. This is also evident from the plots in subfigures 4.8(b) and 4.8(d). However, the eigenvalues of $\tilde{A}_0^{-1}$ are closer to the eigenvalues of A_0^{-1} in the plot in subfigure 4.8(b) than in the plot in subfigure 4.8(d). The eigenvalues of $\tilde{A}_0^{-1}$ are furthest from the eigenvalues of A_0^{-1} in the plots in subfigures 4.8(e) and 4.8(f). This is again substantiated by the result D_e obtained in each case.

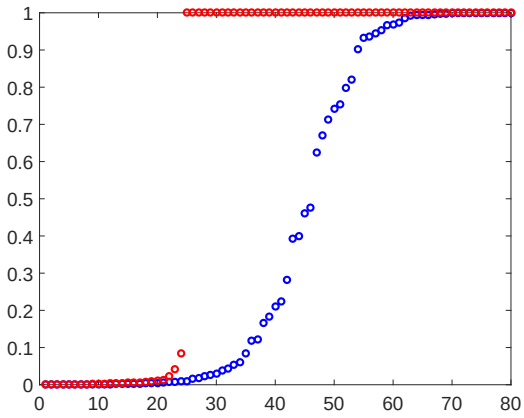
The plots in subfigures 4.9(a) and 4.9(b) correspond to the case of $R_e = 16$. The



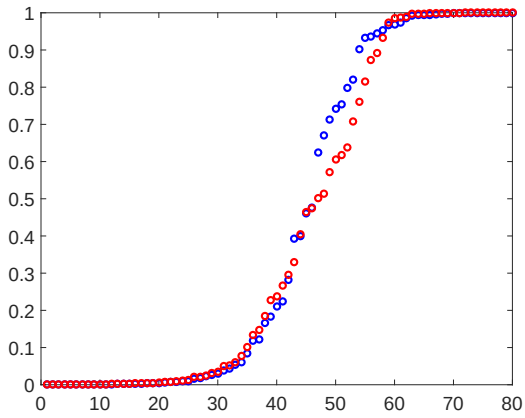
(a) $R_e = 4$, $N_e = (0, 0, 4, 24)$, $D_e = 4.3365e - 1$



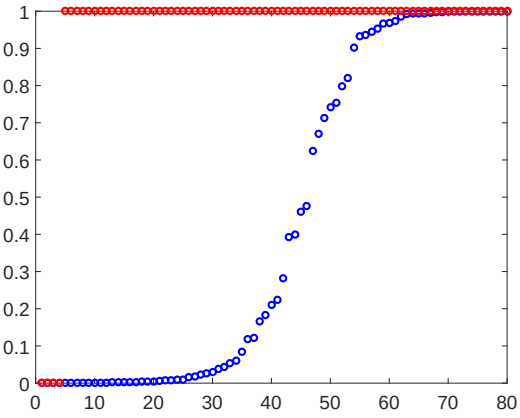
(b) $R_e = 24$, $N_e = (3, 36, 0, 24)$, $D_e = 8.8838e - 2$



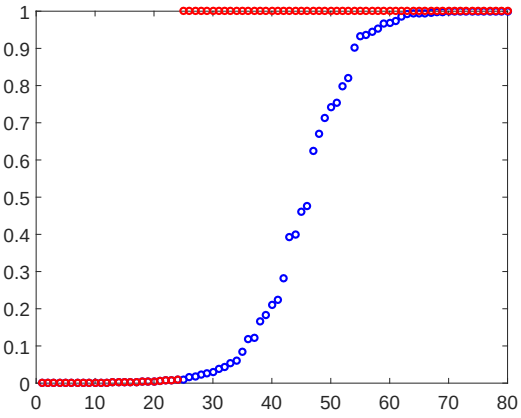
(c) $R_e = 4$, $N_e = (0, 0, 8, 16)$, $D_e = 4.4996e - 1$



(d) $R_e = 24$, $N_e = (6, 12, 24, 48)$, $D_e = 1.2217e-1$

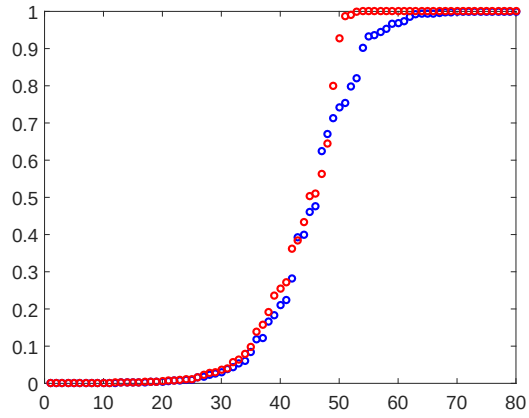


(e) $R_e = 4$, $N_e = (4, 0, 0, 0)$, $D_e = 8.7840e - 1$

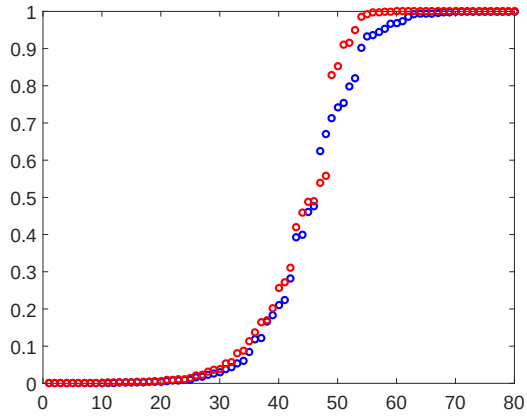


(f) $R_e = 24$, $N_e = (24, 0, 0, 0)$, $D_e = 3.6951e - 1$

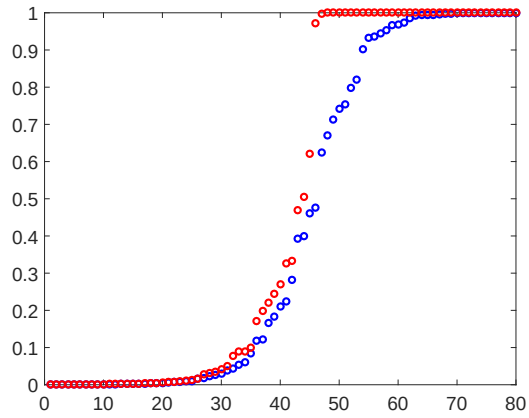
Figure 4.8: Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles) and $\tilde{A}_0^{-1}$ (red circles). Six combinations of R_e and N_e are highlighted. The results D_e obtained are presented.



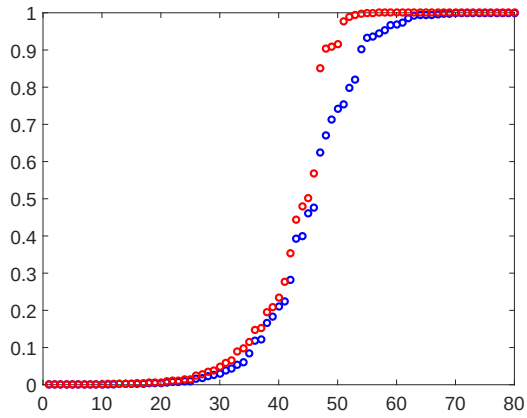
(a) $R_e = 16$, $N_e = (0, 14, 31, 10)$, $D_e = 1.3759e{-}1$



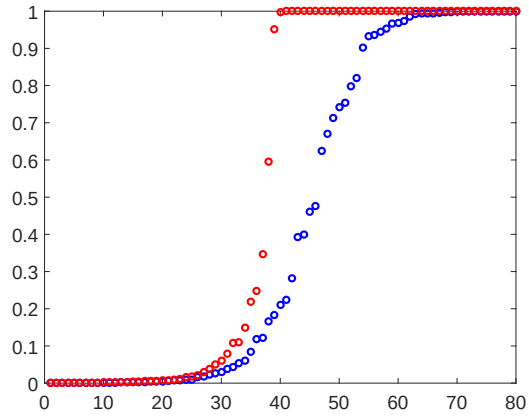
(b) $R_e = 16$, $N_e = (4, 8, 16, 32)$, $D_e = 1.7247e - 1$



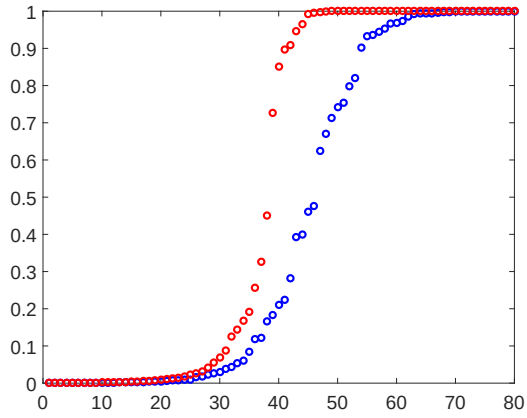
(c) $R_e = 12$, $N_e = (0, 7, 28, 12)$, $D_e = 1.8564e - 1$



(d) $R_e = 12$, $N_e = (0, 8, 16, 32)$, $D_e = 2.0623e - 1$



(e) $R_e = 8$, $N_e = (0, 1, 21, 18)$, $D_e = 2.5832e - 1$



(f) $R_e = 8$, $N_e = (0, 0, 16, 32)$, $D_e = 2.7997e - 1$

Figure 4.9: Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles) and $\tilde{A}_0^{-1}$ (red circles). Six combinations of R_e and N_e are highlighted. The results D_e obtained are presented.

R_e	D_e^a	M_e^a
4	$4.4026e-1$	57
8	$2.6536e-1$	86
12	$1.9155e-1$	131
16	$1.4411e-1$	186
20	$1.1377e-1$	169
24	$9.3671e-2$	205

Table 4.3: The average values D_e^a and M_e^a calculated with respect to the 5% of cases of N_e that resulted in the lowest D_e for six instances of R_e included in the plot in Figure 4.7.

R_e	N_e	D_e	M_e
4	$(0, 0, 4, 24)$	$4.3365e-1$	56
8	$(0, 1, 21, 18)$	$2.5832e-1$	87
12	$(0, 7, 28, 12)$	$1.8564e-1$	126
16	$(0, 14, 31, 10)$	$1.3759e-1$	325
20	$(1, 31, 0, 28)$	$1.0697e-1$	112
24	$(3, 36, 0, 24)$	$8.8838e-2$	156

Table 4.4: The values D_e and M_e obtained for six cases of R_e included in the plot in Figure 4.7 where N_e resulted in the true minimum D_e.

plots in subfigures 4.9(c) and 4.9(d) relate to the case of $R_e = 12$. The plots in subfigures 4.9(e) and 4.9(f) pertain to the case of $R_e = 8$. The cases of N_e corresponding to the plots in subfigures 4.9(a), 4.9(c), and 4.9(e) resulted in the true minimum D_e. The cases of N_e relating to the plots in subfigures 4.9(b), 4.9(d), and 4.9(f) resulted from applying the doubling strategy. The key observation is that the eigenvalues of $\tilde{A}_0^{-1}$ in the plots in subfigures 4.9(a), 4.9(c), and 4.9(e) have similar distributions to those in the corresponding plots in subfigures 4.9(b), 4.9(d), and 4.9(f), respectively. This indicates that the doubling strategy is a viable method for specifying N_e. The effect of transitioning from involving the two coarsest grids at grid levels $k = 2$ and $k = 3$, to incorporating all four grids, in N_e on the eigenvalue distribution of $\tilde{A}_0^{-1}$ is illustrated in the plots in subfigures 4.9(b), 4.9(d), and 4.9(f), respectively.

R_e	N_e	D_e	M_e
4	$(4, 0, 0, 0)$	$8.7840e-1$	27
8	$(8, 0, 0, 0)$	$7.7086e-1$	30
12	$(12, 0, 0, 0)$	$6.6738e-1$	41
16	$(16, 0, 0, 0)$	$5.6683e-1$	40
20	$(20, 0, 0, 0)$	$4.6678e-1$	38
24	$(24, 0, 0, 0)$	$3.6951e-1$	38

Table 4.5: The values D_e and M_e obtained for six cases of R_e included in the plot in Figure 4.7 where N_e involved only the finest grid at grid level $k = 0$.

R_e	N_e	D_e	M_e
4	$(0, 0, 8, 16)$	$4.4996e-1$	62
8	$(0, 0, 16, 32)$	$2.7997e-1$	62
12	$(0, 8, 16, 32)$	$2.0623e-1$	116
16	$(4, 8, 16, 32)$	$1.7247e-1$	159
20	$(5, 10, 20, 40)$	$1.4219e-1$	285
24	$(6, 12, 24, 48)$	$1.2217e-1$	262

Table 4.6: The values D_e and M_e obtained for six cases of R_e included in the plot in Figure 4.7 where N_e followed the doubling strategy.

In Table 4.3, the average values of D_e and M_e calculated with respect to the 5% of cases of N_e that resulted in the lowest D_e are tabulated for six instances of R_e included in the plot in Figure 4.7. These figures are denoted by D_e^a and M_e^a. The key observation based on the results presented in Table 4.3 is that M_e^a increased as R_e increased, and D_e^a decreased, in almost all instances. Note that it is difficult to predict M_e in advance since this depends on the performance of the ARPACK software used to solve the eigenvalue problems in the framework of the multilevel eigenvalue decomposition algorithm. The results D_e and M_e tabulated in Tables 4.4, 4.5, and 4.6 are for the six cases of R_e presented in Table 4.3. Specifically, the cases of N_e presented in Table 4.4 resulted in the true minimum D_e. The cases of N_e presented in Table 4.5 involve only the finest grid at grid level $k = 0$. The cases of N_e presented in Table 4.6 resulted from applying the doubling strategy. Note that more than one case of N_e resulted from applying the doubling strategy in the instances of $R_e = 4, 8, 12$. The case of N_e that resulted in the minimum D_e has been selected in these instances. The first observation based on the results presented in Table 4.4 is that every case of N_e implemented, excluding the instance of $N_e = (0, 0, 4, 24)$ pertaining to $R_e = 4$, involved three grid levels. The second observation is that all cases of N_e implemented involved the coarsest grid at grid level $k = 3$. The cases of N_e implemented that involved three grid levels included the grid at grid level $k = 1$. The cases of $N_e = (1, 31, 0, 28)$ and $N_e = (3, 36, 0, 24)$ relating to $R_e = 20$ and $R_e = 24$, respectively, involved the finest grid at grid level $k = 0$, but not the grid at grid level $k = 2$. However, the values of n_0 implemented in N_e in these cases was small. An additional observation based on the results presented in Tables 4.4, 4.5, and 4.6 is that involving only the finest grid at grid level $k = 0$ in N_e required computing the smallest number of Hessian vector products at the finest grid level $k = 0$ in all instances, as shown by the values of M_e. In addition, the result M_e attained in the cases of $R_e = 8, 12, 16$ where the instance of N_e implemented resulted from applying the doubling strategy is smaller than obtained in the corresponding true minimum case.

Overall, the results presented demonstrate that the accuracy of $\tilde{A}_0^{-1}$ depends on R_e, that is, the memory allocated to the multilevel eigenvalue decomposition algorithm. These results also show that, when R_e is restricted, then specifying N_e based on a

multilevel approach produces a more accurate approximation to A_0^{-1} than achievable by employing a single-level procedure. The results presented indicate that doubling the values in N_e successively from the finest grid, to the coarsest grid, is a viable strategy for generating reasonably accurate approximations to A_0^{-1}. However, constructing an approximation to A_0^{-1} using the multilevel eigenvalue decomposition algorithm has been shown to be computationally expensive, as evidenced by the values of M_e obtained.

4.13 Conclusion

This chapter introduced a multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of any given symmetric positive definite matrix $A \in \mathbb{R}^{N \times N}$ with eigenvalues clustered around unity. It was assumed that A could be approximated using a specified number of its leading eigenvalues, and corresponding eigenvectors, by means of a limited-memory approximation of the form defined in (4.1).

The numerical studies presented in this chapter involved approximating the inverse Hessian for the specific model problem outlined in Section 4.6. The approach adopted focused on fixing the memory ratio R_e in (4.31) allocated to the multilevel eigenvalue decomposition algorithm and specifying N_e in (4.25) accordingly. This included evaluating the accuracy of approximations to the inverse Hessian by calculating D_e in (4.39). The total number of Hessian vector products computed on the finest grid at grid level $k = 0$ was evaluated by calculating M_e in (4.40). This provided an indication of computational cost.

The choice of interpolation method used to implement the interpolation operator in the multilevel eigenvalue decomposition algorithm was considered in Section 4.10. The results presented demonstrated that implementing the interpolation operator P_k^{k-i} in (4.2) using linear interpolation produced a less accurate approximation to the inverse Hessian than achieved by employing cubic spline interpolation. However, constructing an approximation to the inverse Hessian with P_k^{k-i} implemented using linear interpolation was shown to be less computationally expensive in almost all cases studied, as substantiated by the results M_e obtained in these instances. The results presented in

Section 4.10 indicated that implementing P_k^{k-i} using cubic spline interpolation was the best strategy in terms of accuracy.

A modified version of the software package ARPACK that permits only one shift to be executed was considered in Section 4.11. The results presented demonstrated that applying the proposed modified version of ARPACK in order to solve the eigenvalue problems in the framework of the multilevel eigenvalue decomposition algorithm produced a less accurate approximation to the inverse Hessian than achieved using the standard version of ARPACK. However, constructing an approximation to the inverse Hessian with the proposed modified version of ARPACK applied was shown to be less computationally expensive in all cases studied, as substantiated by the results M_e obtained. The results presented in Section 4.11 indicated that applying the proposed modified version of ARPACK was a viable strategy for reducing the computational cost. However, as we were interested in obtaining more accurate approximations, this was not used in practice.

The numerical study presented in Section 4.12 focused on comparing various approximations to the inverse Hessian constructed using the multilevel eigenvalue decomposition algorithm. The results presented demonstrated that the accuracy of the approximation to the inverse Hessian constructed using the multilevel eigenvalue decomposition algorithm depended on the memory allocated to this algorithm, as measured by R_e. These results also showed that, when R_e was restricted, then specifying N_e based on a multilevel approach produced a more accurate approximation to the inverse Hessian than achieved by employing a single-level procedure. The results presented in Section 4.12 indicated that doubling the values in N_e successively from the finest grid at grid level $k = 0$, to the coarsest grid at grid level $k = 3$, was a viable strategy for generating reasonably accurate approximations to the inverse Hessian. However, constructing an approximation to the inverse Hessian using the multilevel eigenvalue decomposition algorithm was shown to be computationally expensive, as substantiated by the results M_e obtained. Given the computational expense of constructing the multilevel approximations considered so far, the remainder of this thesis is devoted to modifications to the algorithm presented in this chapter which reduce implementation costs.

A novel decomposition of the Hessian C as the sum of a set of local Hessians is introduced in Chapter 5, which leads to two practical algorithms for constructing limited-memory approximations to the inverse Hessian (and inverse square root Hessian). These algorithms are based on the multilevel eigenvalue decomposition algorithm presented in this chapter.

Chapter 5

Practical algorithms for approximating the inverse Hessian

In this chapter, two practical algorithms for constructing a limited-memory approximation to the inverse (and inverse square root) of the Hessian C in (2.50) are presented. Firstly, a modified version of the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 that involves generating a representation of the matrix A at the next finest grid level in the multilevel construction is considered in Section 5.1. The numerical study presented focuses on approximating the inverse Hessian for the specific model problem outlined in Section 4.6. A novel decomposition of the Hessian as the sum of a set of local Hessians is then derived in Section 5.2.

The Hessian decomposition algorithm presented in Section 5.3 is the first practical algorithm introduced in this chapter. The premise is to reduce the computational cost of constructing an approximation to the inverse Hessian, or inverse square root Hessian, compared to applying the multilevel eigenvalue decomposition algorithm. An outline of the Hessian decomposition algorithm is presented in Section 5.3.1. This algorithm is described in detail in Section 5.3.2, and summarised in Section 5.3.3. The Hessian decomposition algorithm is applied to the Hessian associated with the specific model problem outlined in Section 4.6 in order to construct an approximation to the inverse Hessian (see Section 5.4). Additional implementation details pertaining to the Hessian

decomposition algorithm are discussed in Section 5.5. The numerical study presented in Section 5.6 focuses on comparing various approximations to the inverse Hessian constructed using the Hessian decomposition algorithm.

The reduced memory Hessian decomposition algorithm presented in Section 5.7 is the second practical algorithm introduced in this chapter. The premise is to reduce the memory requirements of the Hessian decomposition algorithm outlined in Section 5.3. An outline of the reduced memory Hessian decomposition algorithm is presented in Section 5.7.1. This algorithm is described in detail in Section 5.7.2, and summarised in Section 5.7.3. The reduced memory Hessian decomposition algorithm is also applied to the Hessian associated with the specific model problem outlined in Section 4.6 in order to construct an approximation to the inverse Hessian (see Section 5.8). The numerical study presented in Section 5.9 focuses on comparing various approximations to the inverse Hessian constructed using the reduced memory Hessian decomposition algorithm.

5.1 Modified version of the multilevel eigenvalue decomposition algorithm

A modified version of the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 that involves generating a representation of the matrix A at the next finest grid level $k-i$ $(0 < i \leq k)$ in the multilevel construction is considered. The premise is to modify the first step of this algorithm by replacing $Q_k(A_0)$ in (4.16) with $Q_k(A_{k-i})$, where $A_{k-i} \in \mathbb{R}^{m_{k-i} \times m_{k-i}}$ denotes a representation of the matrix A at grid level $k-i$. Specifically, the first step of the proposed modified version of the multilevel eigenvalue decomposition algorithm is to represent A_{k-i} on the grid at grid level k using the grid transfer operator $Q_k(.)$ in (4.15) as follows:

$$Q_k(A_{k-i}) = (P_k^{k-i})^T (A_{k-i} - I_{k-i}) P_k^{k-i} + I_k \quad (0 < i \leq k). \tag{5.1}$$

The next step is to precondition $Q_k(A_{k-i})$ in (5.1) analogously to $Q_k(A_0)$ in (4.17) to obtain

$$\tilde{Q}_k(A_{k-i}) = (Z_{k+1}^k)^T Q_k(A_{k-i}) Z_{k+1}^k \quad (0 < i \leq k). \tag{5.2}$$

The Lanczos method is then used to compute estimates of a specified number n_k of the largest eigenvalues, and corresponding eigenvectors, of $\tilde{Q}_k(A_{k-i})$ in (5.2). The resulting n_k eigenpairs $\{\lambda_k^i, \mathbf{u}_k^i\}$ $(i = 1, \ldots, n_k)$ are required in order to construct a limited-memory approximation to $\tilde{Q}_k^{-1}(A_{k-i})$ or $\tilde{Q}_k^{-1/2}(A_{k-i})$ of the form defined in (4.18) or (4.19).

The particular case of the matrix A considered in this thesis is the Hessian C in (2.50). Suppose that C is represented on the finest grid at grid level $k = 0$ in the multilevel construction. Let $C_k \in \mathbb{R}^{m_k \times m_k}$ denote a representation of the matrix C at grid level k. It is assumed that $C_0\mathbf{w}_0$ is computable for any given vector $\mathbf{w}_0 \in \mathbb{R}^{m_0}$ defined at grid level $k = 0$. Let $B_0^{1/2} \in \mathbb{R}^{m_0 \times m_0}$, $\hat{H}_0 \in \mathbb{R}^{p \times m_0}$, $\hat{H}_0^T \in \mathbb{R}^{m_0 \times p}$, and $\hat{R}_0^{-1} \in \mathbb{R}^{p \times p}$ denote representations of the matrices $B^{1/2}$, $\hat{H}$, $\hat{H}^T$, and $\hat{R}^{-1}$ in (2.50), respectively, at grid level $k = 0$ where

$$p = \sum_{i=0}^{n} p_i \tag{5.3}$$

with p_i the number of available observations at time t_i (see Section 2.1). It follows that

$$C_0 = I_0 + B_0^{1/2} \hat{H}_0^T \hat{R}_0^{-1} \hat{H}_0 B_0^{1/2} \tag{5.4}$$

where I_0 is the $m_0 \times m_0$ identity matrix associated with grid level $k = 0$.

Suppose that $\tilde{M}_k \in \mathbb{R}^{m_k \times m_k}$ and $\tilde{M}_k^T \in \mathbb{R}^{m_k \times m_k}$ are representations of the tangent linear model (TLM) $\tilde{M}_{i,i+1}$ in (2.36) and the adjoint model $\tilde{M}_{i,i+1}^T$, respectively, at grid level k. These can be used to define $\hat{H}_k \in \mathbb{R}^{p \times m_k}$ and $\hat{H}_k^T \in \mathbb{R}^{m_k \times p}$ which are representations of the matrices $\hat{H}$ and $\hat{H}^T$ in (2.50), respectively, at grid level k. Then C_k can be generated at grid level k using the grid transfer operator $Q_k(.)$ in (4.15) as follows:

$$C_k = I_k + Q_k(B_0^{1/2}) \hat{H}_k^T Q_k(\hat{R}_0^{-1}) \hat{H}_k Q_k(B_0^{1/2}) \tag{5.5}$$

where I_k is the $m_k \times m_k$ identity matrix associated with grid level k. Note that this definition of C_k is used in the Hessian decomposition and reduced memory Hessian decomposition algorithms presented in Sections 5.3 and 5.7, respectively.

The numerical study presented here involves approximating the inverse Hessian

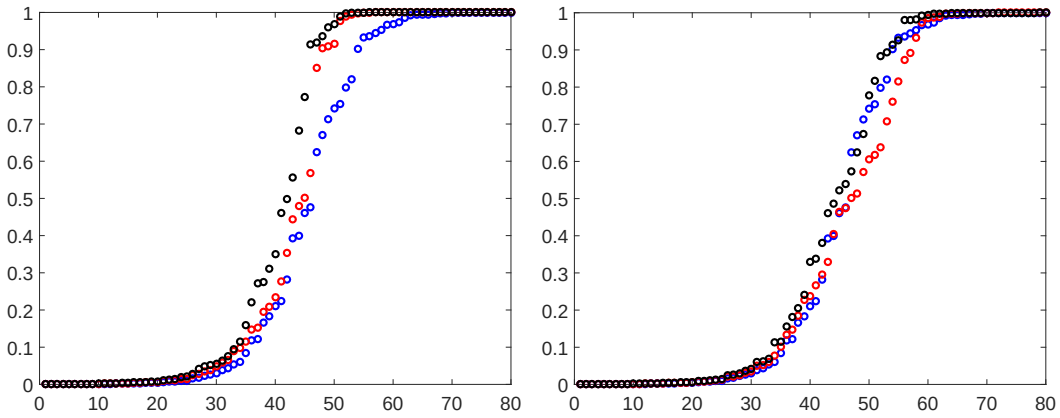
		D_e		M_e	
R_e	N_e	Standard	Modified	Standard	Modified
8	$(0, 0, 16, 32)$	$2.7997e-1$	$3.0169e-1$	62	33
12	$(0, 8, 16, 32)$	$2.0623e-1$	$2.2598e-1$	116	63
16	$(4, 8, 16, 32)$	$1.7247e-1$	$1.9786e-1$	159	106
24	$(6, 12, 24, 48)$	$1.2217e-1$	$1.4722e-1$	262	191

Table 5.1: The values D_e and M_e obtained for four combinations of R_e and N_e, respectively, using the standard or proposed modified version of the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1.

for the specific model problem outlined in Section 4.6. The notation introduced in Section 4.8 is employed. Let $A_k \in \mathbb{R}^{401\times401}$ denote a representation of the matrix A at grid level k (equivalent to C_k in (5.5)). The two versions of the multilevel eigenvalue decomposition algorithm are compared by evaluating the accuracy of $\tilde{A}_0^{-1}$ in each case using D_e in (4.39), with M_e in (4.40) also being calculated to provide an indication of computational cost.

The results obtained for four combinations of R_e in (4.31) and N_e in (4.25) previously used in Table 4.6 are tabulated in Table 5.1. The key observation based on the results presented in Table 5.1 is that applying the proposed modified version of the multilevel eigenvalue decomposition algorithm resulted in larger values of D_e, that is, less accurate approximations to A_0^{-1}, in all instances. However, implementing the proposed modified version of the multilevel eigenvalue decomposition algorithm involved computing a smaller number of Hessian vector products at the finest grid level $k = 0$ in all cases, as shown by the values of M_e tabulated.

The eigenvalues of A_0^{-1} and $\tilde{A}_0^{-1}$ for two combinations of R_e and N_e presented in Table 5.1 are plotted in Figure 5.1. The plots in subfigures 5.1(a) and 5.1(b) relate to the cases of $R_e = 12$ with $N_e = (0, 8, 16, 32)$ and $R_e = 24$ with $N_e = (6, 12, 24, 48)$, respectively. The first eighty eigenvalues of A_0^{-1} are plotted in each case using blue circles. The remaining eigenvalues of A_0^{-1} are close to one and have been omitted. The first eighty eigenvalues of $\tilde{A}_0^{-1}$ where the standard version of the multilevel eigenvalue decomposition algorithm has been applied are plotted using red circles. The first eighty eigenvalues of $\tilde{A}_0^{-1}$ where the proposed modified version of the multilevel eigen-



(a) $R_e = 12,\ N_e = (0, 8, 16, 32)$ (b) $R_e = 24,\ N_e = (6, 12, 24, 48)$

Figure 5.1: Comparison plots showing the first eighty eigenvalues of: A_0^{-1} (blue circles), $\tilde{A}_0^{-1}$ where the standard version of the multilevel eigenvalue decomposition algorithm has been applied (red circles), and $\tilde{A}_0^{-1}$ where the proposed modified version of the multilevel eigenvalue decomposition algorithm has been applied (black circles). Two combinations of R_e and N_e presented in Table 5.1 are highlighted.

value decomposition algorithm has been applied are plotted using black circles. The eigenvalues of $\tilde{A}_0$ in the plots in subfigures 5.1(a) and 5.1(b) with the two versions of the multilevel eigenvalue decomposition algorithm applied have similar distributions, which corresponds to the small differences in D_e shown in Table 5.1.

The results presented demonstrate that applying the proposed modified version of the multilevel eigenvalue decomposition algorithm produces a less accurate approximation to A_0^{-1} than achievable using the standard version of this algorithm. However, constructing an approximation to A_0^{-1} using the proposed modified version of the multilevel eigenvalue decomposition algorithm has been shown to be less computationally expensive in all cases studied, as evidenced by the results M_e attained. This indicates that applying the proposed modified version of the multilevel eigenvalue decomposition algorithm is a viable strategy for reducing the computational cost.

5.2 Decomposition of the Hessian

In this section, a novel decomposition of the Hessian C in (2.50) as the sum of a set of local Hessians is derived. The idea is based on considering the local impact of observational data on C. Recall that the observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ in (2.2)

are included in the 4D-Var cost function $\mathcal{J}(\mathbf{x}_0)$ in (2.4), with the uncertainty in $\mathbf{y}_i$ $(i = 0, \ldots, n)$ represented in $\mathcal{J}(\mathbf{x}_0)$ using the observation error covariance matrix R_i. The block diagonal matrix $\hat{R}$ in (2.38) contains the matrices R_i $(i = 0, \ldots, n)$ along its main diagonal, so the observational component of 4D-Var is captured by C through the matrix $\hat{R}^{-1}$.

The proposed decomposition of C is derived by factorising the matrix $\hat{R}^{-1}$ in (2.50) as follows:

$$\hat{R}^{-1} = \hat{R}^{-1/2} I_p \hat{R}^{-1/2} \tag{5.6}$$

where I_p is the $p \times p$ identity matrix and p has been defined in (5.3). Substituting the expression for $\hat{R}^{-1}$ in (5.6) into (2.50) gives

$$C = I_N + B^{1/2} \hat{H}^T \hat{R}^{-1/2} I_p \hat{R}^{-1/2} \hat{H} B^{1/2}. \tag{5.7}$$

Let $\tilde{I}$ denote a set containing the indices of the diagonal entries of I_p. Partition $\tilde{I}$ into L disjoint subsets $\tilde{I}^l$ $(l = 1, \ldots, L)$. Assuming that $\hat{I}^l \in \mathbb{R}^{p \times p}$ $(l = 1, \ldots, L)$ is a diagonal matrix defined such that

$$\hat{I}^l_{i,i} = \begin{cases} 1, & i \in \tilde{I}^l \\ 0, & i \notin \tilde{I}^l \end{cases}, \; i = 1, \ldots, p,$$

then I_p is expressible in the form

$$I_p = \sum_{l=1}^{L} \hat{I}^l.$$

It follows from (5.7) that

$$\begin{aligned} C &= I_N + \sum_{l=1}^{L} B^{1/2} \hat{H}^T \hat{R}^{-1/2} \hat{I}^l \hat{R}^{-1/2} \hat{H} B^{1/2} \\ &= I_N + \sum_{l=1}^{L} (C^l - I_N) \end{aligned} \tag{5.8}$$

where

$$C^l = I_N + B^{1/2} \hat{H}^T \hat{R}^{-1/2} \hat{I}^l \hat{R}^{-1/2} \hat{H} B^{1/2}. \tag{5.9}$$

We refer to C^l in (5.9) as a local Hessian.

To use this decomposition of C, a strategy for partitioning the observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ in (2.2) to be associated with C^l in (5.9) is required. The observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ are typically provided by sensors. The approach employed to define a practical partition of $\tilde{I}$ uses the fact that the impact of observations provided by a sensor located within any given spatial subdomain is also spatially localised. Suppose that S sensors are deployed. Assuming that each sensor j $(j = 1, \ldots, S)$ provides $\hat{p}$ observations at time t_i $(i = 0, \ldots, n)$, then $p_i = S\hat{p}$ in (5.3) and $p = (n+1)S\hat{p}$. A stationary sensor has a fixed position in space and provides observational data captured at this point. On the other hand, a moving sensor traverses a spatial domain providing observational data captured within this domain. Let $x_j(t_i)$ $(j = 1, \ldots, S)$ denote the position of sensor j in the spatial domain Ω at time t_i $(i = 0, \ldots, n)$. If sensor j is stationary, then $x_j(t_i) = x_j(t_0)$ for all $i \in [0, n]$. It is assumed that $\hat{R}$ in (2.50) has the form defined in (2.38), where $R_i \in \mathbb{R}^{S\hat{p} \times S\hat{p}}$ $(i = 0, \ldots, n)$. The premise is to partition Ω into L disjoint subdomains Ω_l $(l = 1, \ldots, L)$ and divide the observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ into sets based on the position of sensor j $(j = 1, \ldots, S)$ at time t_i by defining $\tilde{I}^l$ as follows:

$$\tilde{I}^l = \{u \in \mathbb{N}, u = (j-1)(n+1)\hat{p} + i\hat{p} + v : x_j(t_i) \in \Omega_l, v = 1, \ldots, \hat{p}\}. \tag{5.10}$$

The standard approach is to order the observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ first by time t_i, and then by sensor j $(j = 1, \ldots, S)$. However, this novel approach requires the observations to be reordered first by sensor, and then by time. Specifically, $\tilde{I}^l$ in (5.10) defines the relationship between the indices of the observations provided by sensor j, and associated with the subdomain Ω_l, in the novel and standard formulations. Provided that Ω_l $(l = 1, \ldots, L)$ cover Ω and do not overlap, then the decomposition of C in (5.8)-(5.9) is defined for this partition. Note that this approach is employed in the numerical studies presented in Sections 5.6, 5.9, and 6.7.

Using (5.8)-(5.9), we can decompose C_k in (5.5) at grid level k in the multilevel construction as

$$C_k = I_k + \sum_{l=1}^{L} (C_k^l - I_k) \tag{5.11}$$

where

$$C_k^l = I_k + Q_k(B_0^{1/2})\hat{H}_k^T Q_k(\hat{R}_0^{-1/2}\hat{I}^l\hat{R}_0^{-1/2})\hat{H}_k Q_k(B_0^{1/2}) \tag{5.12}$$

is a local Hessian defined at grid level k. Note that C_k^l in (5.12) can be approximated using a specified number of its leading eigenvalues, and corresponding eigenvectors, by means of a limited-memory approximation of the form defined in (2.51). That is, for a specific partition of $\tilde{I}$, and any $\gamma \in \mathbb{R}$,

$$(C_k^l)^\gamma \simeq I_k + \sum_{i=1}^{r_k^l} ((\lambda_{k,i}^l)^\gamma - 1)\mathbf{u}_{k,i}^l(\mathbf{u}_{k,i}^l)^T \tag{5.13}$$

where $\{\lambda_{k,i}^l, \mathbf{u}_{k,i}^l\}$ $(i = 1, \ldots, r_k^l)$ denote r_k^l eigenpairs of C_k^l, with $r_k^l \ll m_k$. The limited-memory decomposition of $(C_k^l)^\gamma$ in (5.13) is relevant to the Hessian decomposition and reduced memory Hessian decomposition algorithms presented in Sections 5.3 and 5.7, respectively.

The introduction of local Hessians leads to a number of potential computational savings. These are outlined as follows:

- Each local Hessian C_k^l in (5.12) differs from I_k only within a set region around the subdomain Ω_l referred to as the area of influence, whose size depends on the specific processes incorporated in the dynamical model $\mathcal{M}(t_{i+1}, t_i, .)$ in (2.1). The computation of an individual local Hessian C_k^l is therefore not as computationally expensive as computing the global Hessian C_k in (5.5).

- The computational cost of calculating each local Hessian C_k^l in (5.12) can be reduced by setting $k > 0$, however, accuracy may be lost.

- A significant reduction in computational cost is achievable by calculating the local Hessians in parallel. In this case, the computational cost of calculating C_k in (5.11)-(5.12) will be the highest cost to calculate an individual local Hessian C_k^l in (5.12). Note that parallel computing is not considered in this thesis.

- If observations from a particular sensor s $(s = 1, \ldots, S)$ are of less importance, then it may not be necessary to compute C_k^l in (5.12) for every subdomain Ω_l. This concept is not considered in this thesis.

5.3 The Hessian decomposition algorithm

5.3.1 Outline

Suppose that C_0 is available on the finest grid at grid level $k = 0$ in the multilevel construction described in Section 4.1 in the form of $C_0\mathbf{w}_0$, where $\mathbf{w}_0 \in \mathbb{R}^{m_0}$ denotes any given vector defined on the finest grid. Full details of the Hessian decomposition algorithm for constructing a limited-memory approximation to $C_0^{-1}\mathbf{w}_0$ and $C_0^{-1/2}\mathbf{w}_0$ are described in Section 5.3.2, but first an outline of this algorithm is presented here. The premise is to reduce the computational cost of constructing an approximation to C_0^{-1} or $C_0^{-1/2}$ compared to applying the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1. The steps involved in the Hessian decomposition algorithm are as follows:

1. Define $\hat{I}^l$ $(l = 1, \ldots, L)$ in (5.12) by partitioning the observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ in (2.2).

2. Calculate each local Hessian C_k^l in (5.12) at a specified grid level k in the multilevel setup. This involves restricting the matrices $B_0^{1/2}$ and $\hat{R}_0^{-1}$ in (5.4) to grid level k.

3. Compute estimates of a specified number (n_k^l, say) of the largest eigenvalues, and corresponding eigenvectors, of C_k^l using the Lanczos method and store in memory.

4. Construct a limited-memory approximation to C_k^l of the form defined in (5.13) to be used to obtain a limited-memory approximation to C_k analogously to (5.11)-(5.12).

5. Prolong the limited-memory approximation to C_k to the finest grid at grid level $k = 0$ using the grid transfer operator $S_{k-i}(.)$ in (4.14).

6. Apply the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 to the prolonged limited-memory approximation to C_k.

An approximation to $C_0^{-1}\mathbf{w}_0$ or $C_0^{-1/2}\mathbf{w}_0$ can be computed on termination of the Hessian decomposition algorithm using the n_k eigenpairs calculated at each grid level k $(k = 0, \ldots, k_c)$ in the framework of the multilevel eigenvalue decomposition algorithm.

5.3.2 Details

Following step 3 of the Hessian decomposition algorithm, the resulting n_k^l eigenpairs $\{\lambda_{k,i}^l, \mathbf{u}_{k,i}^l\}$ $(i = 1, \ldots, n_k^l)$ are used to construct a limited-memory approximation to C_k^l of the form defined in (5.13). Specifically,

$$\tilde{C}_k^l = I_k + \sum_{i=1}^{n_k^l} (\lambda_{k,i}^l - 1)\mathbf{u}_{k,i}^l(\mathbf{u}_{k,i}^l)^T \quad (l = 1, \ldots, L). \tag{5.14}$$

A limited-memory approximation to C_k is then constructed using $\tilde{C}_k^l$ in (5.14) as follows:

$$\tilde{C}_k = I_k + \sum_{l=1}^{L} (\tilde{C}_k^l - I_k). \tag{5.15}$$

The definition of $\tilde{C}_k$ in (5.15) is based on the decomposition of C_k in (5.11)-(5.12). The grid transfer operator $S_{k-i}(.)$ in (4.14) is used to prolong $\tilde{C}_k$ to the finest grid at grid level $k = 0$. The multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 is applied to $S_0(\tilde{C}_k)$ at this stage.

In what follows, it is useful to represent the parameters n_k^l $(l = 1, \ldots, L)$ in vector form by setting

$$N_d = (n_k^1, n_k^2, \ldots, n_k^L), \quad \hat{N}_d = \sum_{l=1}^{L} n_k^l \tag{5.16}$$

where $\hat{N}_d$ is the sum of the entries in N_d. The accuracy of an approximation to $C_0^{-1}\mathbf{w}_0$ or $C_0^{-1/2}\mathbf{w}_0$ constructed using the Hessian decomposition algorithm is dependent on the choices of N_d in (5.16) and N_e in (4.25). That is, the values n_k^l in N_d and n_k in N_e are key in determining the quality of the approximation obtained. Various combinations of k, N_d, and N_e are considered in the numerical study presented in Section 5.6.

5.3.3 Summary

The Hessian decomposition algorithm described in Section 5.3.2 is outlined in Figure 5.2. Note that the matrices involved in this procedure are not explicitly constructed in practice. Specifically, only matrix vector products are computed. The inputs to the Hessian decomposition algorithm are C_0 in the form of $C_0\mathbf{w}_0$, where $\mathbf{w}_0 \in \mathbb{R}^{m_0}$ denotes

1: $[\Lambda_0, U_0] = HD(C_0, k, N_d, N_e)$
2: Define $\hat{I}^l$ $(l = 1, \ldots, L)$ in (5.12)
3: **for** $l = 1, \ldots, L$ **do**
4: Calculate C_k^l in (5.12)
5: Compute $\{\lambda_{k,i}^l, \mathbf{u}_{k,i}^l\}$ $(i = 1, \ldots, n_k^l)$ by applying the Lanczos method to C_k^l and store in memory
6: **end for**
7: Construct $\tilde{C}_k$ in (5.15) using $\tilde{C}_k^l$ in (5.14)
8: Prolong $\tilde{C}_k$ to grid level $k = 0$ using $S_{k-i}(.)$ in (4.14)
9: Compute Λ_0 in (4.29) and U_0 in (4.30) by applying the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 to $S_0(\tilde{C}_k)$

Figure 5.2: Outline of the Hessian decomposition algorithm.

any given vector defined at grid level $k = 0$, a specified grid level k $(0 \leq k \leq k_c)$, N_d in (5.16), and N_e in (4.25). The outputs are the vectors Λ_0 in (4.29) and U_0 in (4.30) as before. An approximation to $C_0^{-1}\mathbf{w}_0$ or $C_0^{-1/2}\mathbf{w}_0$ can be computed on termination of the Hessian decomposition algorithm by means of (4.23) or (4.24) using Λ_0 and U_0.

Recall that the memory ratio for the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 is R_e in (4.31). The Hessian decomposition algorithm requires additional memory since a further $\hat{N}_d$ eigenpairs must be stored, where $\hat{N}_d$ is defined in (5.16). The finest grid at grid level $k = 0$ is composed of $m_0 = m + 1$ grid points. Given that the grid at grid level k contains $m_k = m/2^k + 1$ grid points, an estimate of the ratio of memory required for the Hessian decomposition algorithm in terms of m is obtained by calculating

$$R_d = \frac{1}{2^k}\hat{N}_d + R_e. \tag{5.17}$$

5.4 Approximating the inverse Hessian using the Hessian decomposition algorithm

The Hessian decomposition algorithm outlined in Figure 5.2 is proposed as a suitable method for constructing a limited-memory approximation to the inverse (and inverse square root) of the Hessian C in (2.50). The numerical study presented in Section 5.6 involves approximating the inverse Hessian for the specific model problem outlined in Section 4.6. This numerical study focuses on comparing various approximations

to the inverse Hessian constructed using the Hessian decomposition algorithm. Let $C_k \in \mathbb{R}^{m_k \times m_k}$ denote a representation of the matrix $C \in \mathbb{R}^{401\times 401}$ at grid level k ($k = 0, 1, 2, 3$). This is defined analogously to C_k in (5.5). Note that C_0 is available in the form of $C_0\mathbf{w}_0$, where $\mathbf{w}_0 \in \mathbb{R}^{401}$ denotes any given vector defined at the finest grid level $k = 0$. For this small example, C_0^{-1} is also available by direct computation. Let $\tilde{C}_0^{-1} \in \mathbb{R}^{401\times 401}$ denote an approximation to C_0^{-1} constructed using the Hessian decomposition algorithm. Additional implementation details relevant to the Hessian decomposition algorithm are discussed in Section 5.5.

5.5 Additional implementation details

The sensor configuration scheme S_1 discussed in Section 4.6 comprises of seven stationary sensors. That is, $L = 7$ in the framework of the Hessian decomposition algorithm outlined in Figure 5.2. The observational data generated is partitioned using the approach described in Section 5.2 in order to define $\hat{I}^l$ ($l = 1, \ldots, 7$) and decompose C_k at grid level k ($0 \leq k \leq 3$) analogously to C_k in (5.11)-(5.12). Let $C_k^l \in \mathbb{R}^{m_k \times m_k}$ denote C_k^l in (5.12) at grid level k for the particular model problem studied. This corresponds to a stationary sensor located at the point $x = \tilde{x}_l$ in the spatial domain $\Omega = [0, 1]$. Specifically, $\tilde{x}_1 = 0.3$, $\tilde{x}_2 = 0.4$, $\tilde{x}_3 = 0.45$, $\tilde{x}_4 = 0.5$, $\tilde{x}_5 = 0.55$, $\tilde{x}_6 = 0.6$, and $\tilde{x}_7 = 0.7$ in this case.

The eigenvalue problems involving C_k^l at grid level k are solved using the standard version of ARPACK; see [78]. That is, estimates of the n_k^l largest, in terms of magnitude, eigenvalues of C_k^l, and the corresponding eigenvectors, are computed using ARPACK. The default tolerance used in the convergence criterion in (3.41) in ARPACK is $t_e = 10^{-2}$. The eigenvalue computations must be conducted in double precision. However, it is acceptable to store the resulting eigenpairs in single precision, as has been discussed in Section 4.7.

5.6 Comparing approximations to the inverse Hessian

The numerical study presented here focuses on the choices of k and N_d in (5.16) in the framework of the Hessian decomposition algorithm outlined in Figure 5.2. Specifically,

various cases of $\tilde{C}_0^{-1}$ are considered. Note that these cases are symmetric positive definite. The accuracy of $\tilde{C}_0^{-1}$ is evaluated in each case using D_e in (4.39). The ARPACK procedure involves computing Hessian vector products of the form $C_k^l \mathbf{w}_k$ at grid level k, where $\mathbf{w}_k \in \mathbb{R}^{m_k \times m_k}$ denotes any given vector defined at grid level k. Recall that the finest grid at grid level $k = 0$ is composed of $m = 401$ grid points, and the grid at grid level k ($k = 1, 2, 3$) contains $m_k = m/2^k + 1$ grid points. An estimate of the number of Hessian vector products computed in terms of the cost of one Hessian vector product at the finest grid level $k = 0$ is obtained by calculating

$$\hat{M}_e = \max_{1 \leq l \leq 7} \hat{m}_l \tag{5.18}$$

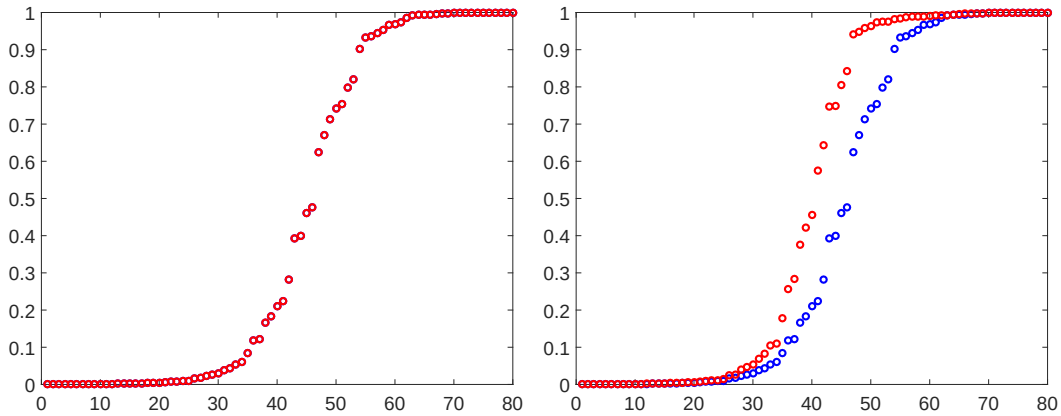
where $\hat{m}_l = \tilde{m}_l/2^k$ and $\tilde{m}_l$ denotes the number of Hessian vector products computed at grid level k in order to solve the eigenvalue problem involving C_k^l at this grid level. Note that $\hat{M}_e$ in (5.18) is calculated to provide an indication of the computational cost if the eigenvalue problems involving C_k^l at grid level k were solved in parallel.

We first suppose that there are no memory restrictions in place. The maximum allowable case of R_d in (5.17) given the model problem setup is $R_d = 3200$. This corresponds to calculating each local Hessian C_k^l at the finest grid level $k = 0$ in the multilevel construction, computing the maximum number of eigenpairs of C_k^l, that is, $n_k^l = 400$ in N_d in (5.16), and setting $N_e = (400, 0, 0, 0)$ in (4.25). Note that this choice of N_e has been considered in the numerical study presented in Section 4.12. The output $\tilde{C}_0^{-1}$ in this instance is the best approximation to C_0^{-1} achievable using the Hessian decomposition algorithm. The results D_e in (4.39) and $\hat{M}_e$ in (5.18) obtained are tabulated in Table 5.2. The eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ are plotted in subfigure 5.3(a). The first eighty eigenvalues of $\tilde{C}_0^{-1}$ are plotted using red circles. The remaining eigenvalues of $\tilde{C}_0^{-1}$ are close to one and have been omitted. The first eighty eigenvalues of C_0^{-1} are plotted using blue circles. However, these eigenvalues are indistinguishable from the eigenvalues of $\tilde{C}_0^{-1}$ in the plot in subfigure 5.3(a). The result $\hat{M}_e$ obtained is indicative of a high computational cost as expected.

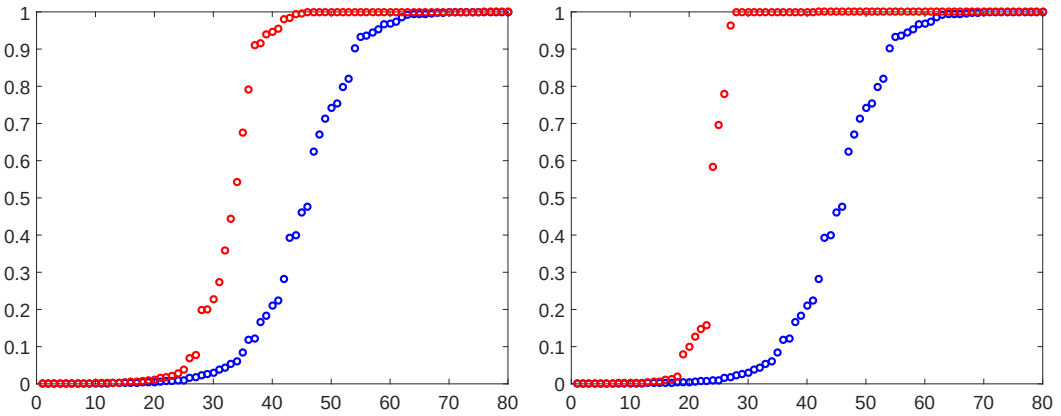
Suppose now instead that memory is restricted. A strategy for reducing R_d in (5.17) is to specify k such that $k > 0$, that is, compute the local Hessians at a coarser

R_d	k	n_k^l	D_e	$\hat{M}_e$
3200	0	400	$4.2496e-12$	401
1100	1	200	$1.6716e-1$	101
575	2	100	$3.4664e-1$	26
444	3	50	$5.3251e-1$	7

Table 5.2: The values D_e and $\hat{M}_e$ obtained for four combinations of k and n_k^l in N_d where $N_e = (400, 0, 0, 0)$. The memory ratio R_d is also tabulated.



(a) $k = 0$, $n_k^l = 400$, $D_e = 4.2496e-12$ (b) $k = 1$, $n_k^l = 200$, $D_e = 1.6716e-1$



(c) $k = 2$, $n_k^l = 100$, $D_e = 3.4664e-1$ (d) $k = 3$, $n_k^l = 50$, $D_e = 5.3251e-1$

Figure 5.3: Comparison plots showing the first eighty eigenvalues of: C_0^{-1} (blue circles) and $\tilde{C}_0^{-1}$ (red circles). The eigenvalues of C_0^{-1} are indistinguishable from the eigenvalues of $\tilde{C}_0^{-1}$ in the plot in subfigure 5.3(a). The four combinations of k and n_k^l presented in Table 5.2 are highlighted. The results D_e obtained are presented.

grid level. We consider the three applicable cases of k for our model problem, namely, $k = 1, 2, 3$. The maximum number of eigenpairs is computed in each case. Specifically, $n_k^l = 200$ for $k = 1$, $n_k^l = 100$ for $k = 2$, and $n_k^l = 50$ for $k = 3$. Again, the particular case of N_e implemented is $N_e = (400, 0, 0, 0)$. The output $\tilde{C}_0^{-1}$ in each of the three cases studied is therefore the best approximation to C_0^{-1} achievable on a specific grid level. The associated values of D_e, $\hat{M}_e$, and R_d are tabulated in Table 5.2. The key observations based on the results presented in Table 5.2 are that D_e increased, that is, the accuracy of $\tilde{C}_0^{-1}$ decreased, but R_d and $\hat{M}_e$ decreased, as k varied from $k = 0$ to $k = 3$. This shows that specifying k such that $k > 0$ is a viable strategy for reducing the memory required and the computational cost. The eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ for the three combinations of k and n_k^l considered are plotted in subfigures 5.3(b)-5.3(d). The plots in subfigures 5.3(b)-5.3(d) relate to the cases of $k = 1$, $k = 2$, and $k = 3$. The first eighty eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ are plotted in each case using blue and red circles, respectively.

We now fix $k = 1$, again with $n_k^l = 200$, and tabulate D_e, $\hat{M}_e$, and R_d in Table 5.3 for the six cases of N_e previously used in Table 4.6. The output $\tilde{C}_0^{-1}$ in each of the six cases studied is the best approximation to C_0^{-1} achievable given k and N_e. Suppose now that n_k^l is varied depending on the particular case of N_e implemented. Specifically, we use the minimum n_k^l required to produce approximations $\tilde{C}_0^{-1}$ of similar accuracy in terms of D_e. The results are tabulated in Table 5.4. The results presented in Tables 5.3 and 5.4 show that using a smaller value of n_k^l than the maximum allowable case of $n_k^l = 200$ is a viable method for reducing the memory required and the computational cost. The eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ for four combinations of n_k^l and N_e presented in Table 5.4 are plotted in Figure 5.4. The plots in subfigures 5.4(a)-5.4(d) relate to the cases of $N_e = (6, 12, 24, 48)$ with $n_k^l = 9$, $N_e = (0, 8, 16, 32)$ with $n_k^l = 9$, $N_e = (0, 0, 16, 32)$ with $n_k^l = 7$, and $N_e = (0, 0, 8, 16)$ with $n_k^l = 5$. The first eighty eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ are plotted in each case using blue and red circles, respectively.

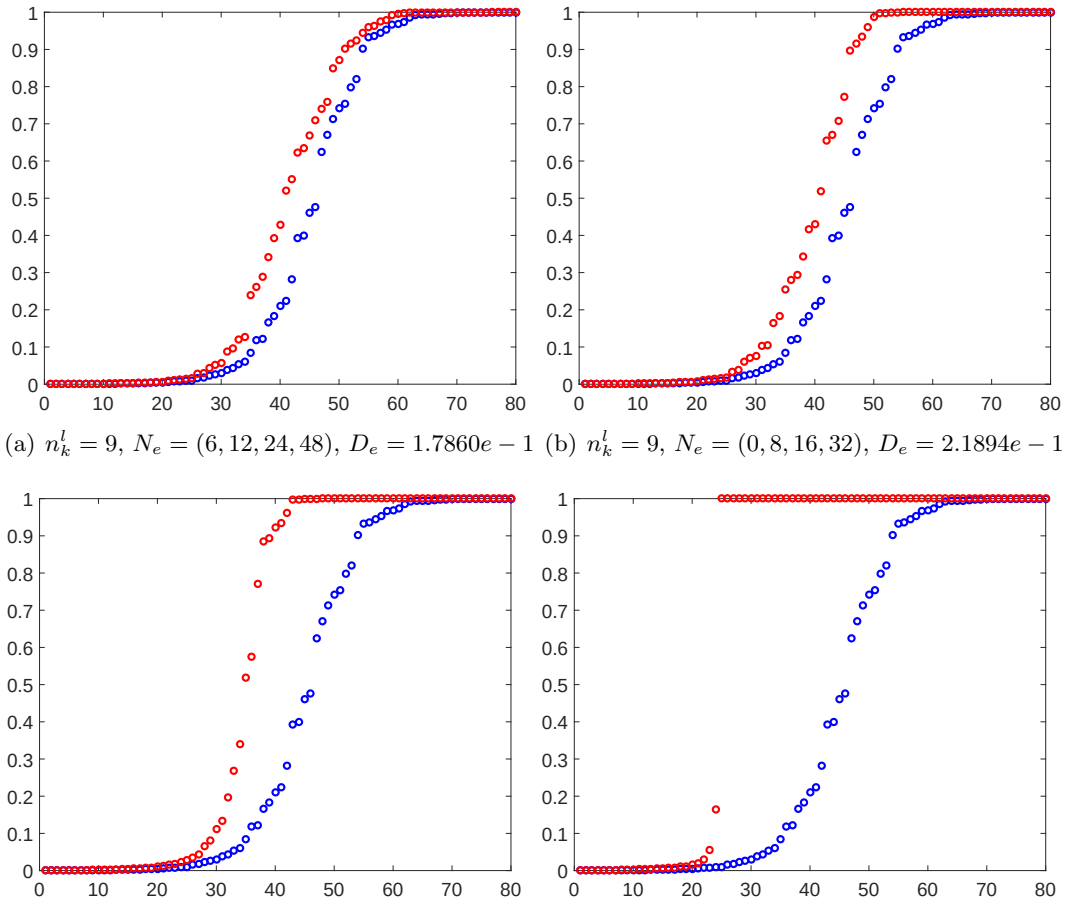
The results presented demonstrate that, for a particular choice of N_e, applying the Hessian decomposition algorithm produces a less accurate approximation to C_0^{-1} than achievable using the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1. However, constructing an approximation to C_0^{-1} using the Hessian de-

R_d	N_e	D_e	$\hat{M}_e$
704	$(0, 0, 8, 16)$	$4.5095e-1$	101
708	$(0, 0, 16, 32)$	$2.8444e-1$	101
712	$(0, 8, 16, 32)$	$2.1870e-1$	101
716	$(4, 8, 16, 32)$	$2.0397e-1$	101
720	$(5, 10, 20, 40)$	$1.8334e-1$	101
724	$(6, 12, 24, 48)$	$1.7746e-1$	101

Table 5.3: The values D_e and $\hat{M}_e$ obtained for the six cases of N_e presented in Table 4.6 where $k = 1$ and $n_k^l = 200$ in N_d. The memory ratio R_d is also tabulated.

R_d	n_k^l	N_e	D_e	$\hat{M}_e$
22	5	$(0, 0, 8, 16)$	$4.5541e-1$	5
33	7	$(0, 0, 16, 32)$	$2.8981e-1$	5
44	9	$(0, 8, 16, 32)$	$2.1894e-1$	6
44	8	$(4, 8, 16, 32)$	$2.0628e-1$	6
48	8	$(5, 10, 20, 40)$	$1.8667e-1$	6
56	9	$(6, 12, 24, 48)$	$1.7860e-1$	6

Table 5.4: The values D_e and $\hat{M}_e$ obtained for the six cases of N_e presented in Table 5.3 where $k = 1$ and n_k^l in N_d is specific to the particular case of N_e implemented. The memory ratio R_d is also tabulated.



(a) $n_k^l = 9$, $N_e = (6, 12, 24, 48)$, $D_e = 1.7860e-1$ (b) $n_k^l = 9$, $N_e = (0, 8, 16, 32)$, $D_e = 2.1894e-1$

(c) $n_k^l = 7$, $N_e = (0, 0, 16, 32)$, $D_e = 2.8981e-1$ (d) $n_k^l = 5$, $N_e = (0, 0, 8, 16)$, $D_e = 4.5541e-1$

Figure 5.4: Comparison plots showing the first eighty eigenvalues of: C_0^{-1} (blue circles) and $\tilde{C}_0^{-1}$ (red circles). Four combinations of n_k^l and N_e presented in Table 5.4 are highlighted. The results D_e obtained are presented.

composition algorithm has been shown to be less computationally expensive in some cases studied where $k > 0$, since $\hat{M}_e < M_e$ in these instances, where M_e is defined in (4.40). The results presented also show that, for a specific choice of N_e, the Hessian decomposition algorithm requires more memory than the multilevel eigenvalue decomposition algorithm, that is, $R_d > R_e$, where R_e is defined in (4.31). These results also indicate that computing estimates of a small number of the largest eigenvalues, and corresponding eigenvectors, of each local Hessian C_k^l at grid level k $(0 < k \leq k_c)$ is a viable strategy for reducing the memory required and the computational cost.

5.7 The reduced memory Hessian decomposition algorithm

5.7.1 Outline

The premise of the reduced memory Hessian decomposition algorithm is to reduce the memory requirements of the Hessian decomposition algorithm outlined in Figure 5.2. The steps involved in the reduced memory Hessian decomposition algorithm are as follows:

1. Define $\hat{I}^l$ $(l = 1, \ldots, L)$ in (5.12) by partitioning the observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ in (2.2).

2. Calculate each local Hessian C_k^l in (5.12) at a specified grid level k in the multilevel setup. This involves restricting the matrices $B_0^{1/2}$ and $\hat{R}_0^{-1}$ in (5.4) to grid level k.

3. Compute estimates of a specified number (n_k^l, say) of the largest eigenvalues, and corresponding eigenvectors, of C_k^l using the Lanczos method and store in memory.

4. Construct a limited-memory approximation to $(C_k^l)^{-1}$ of the form defined in (5.13).

5. Apply the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 to the limited-memory approximation to $(C_k^l)^{-1}$ in order to obtain a limited-memory approximation to C_k^l.

6. Construct a limited-memory approximation to C_k analogously to (5.11)-(5.12) using the limited-memory approximation to C_k^l.

7. Prolong the limited-memory approximation to C_k to the finest grid at grid level $k = 0$ using the grid transfer operator $S_{k-i}(.)$ in (4.14).

8. Apply the multilevel eigenvalue decomposition algorithm to the prolonged limited-memory approximation to C_k.

An approximation to $C_0^{-1}\mathbf{w}_0$ or $C_0^{-1/2}\mathbf{w}_0$ can be computed on termination of the reduced memory Hessian decomposition algorithm using the n_k eigenpairs calculated at each grid level k $(k = 0, \ldots, k_c)$ in the framework of the multilevel eigenvalue decomposition algorithm.

5.7.2 Details

The first steps are exactly as before: define $\hat{I}^l$ $(l = 1, \ldots, L)$ in (5.12) by partitioning the observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$ in (2.2); calculate each local Hessian C_k^l in (5.12); use the Lanczos method to compute estimates of a specified number n_k^l of the largest eigenvalues, and corresponding eigenvectors, of C_k^l. Instead of being used to construct a limited-memory approximation to C_k^l, however, here the resulting n_k^l eigenpairs $\{\lambda_{k,i}^l, \mathbf{u}_{k,i}^l\}$ $(i = 1, \ldots, n_k^l)$ are required in order to construct a limited-memory approximation to the local inverse Hessian $(C_k^l)^{-1}$. Specifically, we obtain

$$(\tilde{C}_k^l)^{-1} = I_k + \sum_{i=1}^{n_k^l} ((\lambda_{k,i}^l)^{-1} - 1)\mathbf{u}_{k,i}^l(\mathbf{u}_{k,i}^l)^T \quad (l = 1, \ldots, L). \tag{5.19}$$

The multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 is then applied to each $(\tilde{C}_k^l)^{-1}$ in (5.19) to obtain a limited memory approximation to $\tilde{C}_k^l$. This is denoted by $\hat{C}_k^l$ $(l = 1, \ldots, L)$. Note that

$$\hat{C}_k^l = Q_k^{-1}((\tilde{C}_k^l)^{-1}) \quad (l = 1, \ldots, L) \tag{5.20}$$

where $Q_k^{-1}(.)$ is defined in (4.20). A limited-memory approximation to C_k is now constructed using $\hat{C}_k^l$ in (5.20) as follows:

$$\hat{C}_k = I_k + \sum_{l=1}^{L} (\hat{C}_k^l - I_k). \tag{5.21}$$

The definition of $\hat{C}_k$ in (5.21) is based on the decomposition of C_k in (5.11)-(5.12). Finally, the grid transfer operator $S_{k-i}(.)$ in (4.14) is used to prolong $\hat{C}_k$ to the finest grid at grid level $k = 0$ as before, and the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 is applied to $S_0(\hat{C}_k)$ at this stage.

The parameters n_k^l $(l = 1, \ldots, L)$ can again be represented in vector form using N_d in (5.16). Note, however, that the additional parameters $\hat{n}_{k'}^l$ $(k' = k, \ldots, k_c)$ must be defined in order to apply the multilevel eigenvalue decomposition algorithm to $(\tilde{C}_k^l)^{-1}$ in (5.19). In what follows, it is useful to represent these parameters in vector form by setting

$$N_k^l = (\hat{n}_k^l, \hat{n}_{k+1}^l, \ldots, \hat{n}_{k_c}^l), \quad \hat{N}_k^l = \sum_{k'=k}^{k_c} \hat{n}_{k'}^l \quad (l = 1, \ldots, L) \tag{5.22}$$

where $\hat{N}_k^l$ is the sum of the entries in N_k^l. The idea is to define N_k^l in (5.22) such that

$$\sum_{k'=k}^{k_c} \frac{\hat{n}_{k'}^l}{2^{k'}} < \frac{n_k^l}{2^k}. \tag{5.23}$$

The accuracy of an approximation to $C_0^{-1}\mathbf{w}_0$ or $C_0^{-1/2}\mathbf{w}_0$ constructed using the reduced memory Hessian decomposition algorithm is dependent on the choices of N_d in (5.16), N_k^l in (5.22), and N_e in (4.25). That is, the values n_k^l in N_d, $\hat{n}_k^l$ in N_k^l, and n_k in N_e are key in determining the quality of the approximation obtained. Various cases of N_k^l are considered in the numerical study presented in Section 5.9 for specific combinations of k, N_d, and N_e.

5.7.3 Summary

The reduced memory Hessian decomposition algorithm described in Section 5.7.2 is outlined in Figure 5.5. Note that the matrices involved in this procedure are not explicitly constructed in practice. Specifically, only matrix vector products are computed

1: $[\Lambda_0, U_0] = RMHD(C_0, k, N_d, N_k^l, N_e)$
2: Define $\hat{I}^l$ $(l = 1, \ldots, L)$ in (5.12)
3: **for** $l = 1, \ldots, L$ **do**
4: Calculate C_k^l in (5.12)
5: Compute $\{\lambda_{k,i}^l, \mathbf{u}_{k,i}^l\}$ $(i = 1, \ldots, n_k^l)$ by applying the Lanczos method to C_k^l and store in memory
6: Construct $(\tilde{C}_k^l)^{-1}$ in (5.19)
7: Compute Λ_k^l in (5.24) and U_k^l in (5.25) by applying the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 to $(\tilde{C}_k^l)^{-1}$
8: **end for**
9: Construct $\hat{C}_k$ in (5.21) using $\hat{C}_k^l$ in (5.20) based on Λ_k^l and U_k^l
10: Prolong $\hat{C}_k$ to grid level $k = 0$ using $S_{k-i}(.)$ in (4.14)
11: Compute Λ_0 in (4.29) and U_0 in (4.30) by applying the multilevel eigenvalue decomposition algorithm to $S_0(\hat{C}_k)$

Figure 5.5: Outline of the reduced memory Hessian decomposition algorithm.

as before. The inputs to the reduced memory Hessian decomposition algorithm are C_0 in the form of $C_0\mathbf{w}_0$, where $\mathbf{w}_0 \in \mathbb{R}^{m_0}$ denotes any given vector defined at grid level $k = 0$, a specified grid level k, N_d in (5.16), N_k^l in (5.22), and N_e in (4.25). The outputs are the vectors Λ_0 in (4.29) and U_0 in (4.30) as before. An approximation to $C_0^{-1}\mathbf{w}_0$ or $C_0^{-1/2}\mathbf{w}_0$ can be computed on termination of the reduced memory Hessian decomposition algorithm by means of (4.23) or (4.24) using Λ_0 and U_0.

The two vectors generated at step 7 of the reduced memory Hessian decomposition algorithm are defined as follows:

$$\Lambda_k^l = [(\lambda_{k_c}^1)^l, \ldots, (\lambda_{k_c}^{\hat{n}_{k_c}})^l, (\lambda_{k_c-1}^1)^l, \ldots, (\lambda_{k_c-1}^{\hat{n}_{k_c-1}})^l, \ldots, (\lambda_k^1)^l, \ldots, (\lambda_k^{\hat{n}_k})^l], \tag{5.24}$$

$$U_k^l = [(\mathbf{u}_{k_c}^1)^l, \ldots, (\mathbf{u}_{k_c}^{\hat{n}_{k_c}})^l, (\mathbf{u}_{k_c-1}^1)^l, \ldots, (\mathbf{u}_{k_c-1}^{\hat{n}_{k_c-1}})^l, \ldots, (\mathbf{u}_k^1)^l, \ldots, (\mathbf{u}_k^{\hat{n}_k})^l]. \tag{5.25}$$

The entries of Λ_k^l in (5.24) are the eigenvalue estimates $(\lambda_k^i)^l$ $(i = 1, \ldots, \hat{n}_k)$ of $\tilde{Q}_k((\tilde{C}_k^l)^{-1})$, where $\tilde{Q}_k(.)$ is defined in (4.17), computed at grid levels $k' = k, \ldots, k_c$. The corresponding eigenvectors $(\mathbf{u}_k^i)^l$ $(i = 1, \ldots, \hat{n}_k)$ are contained in U_k^l in (5.25). The vector Λ_k^l contains a total of $\hat{N}_k^l$ entries, where $\hat{N}_k^l$ is defined in (5.22). The total number of entries in U_k^l is therefore

$$\sum_{k'=k}^{k_c} \hat{n}_{k'}^l m_{k'} = \left(\sum_{k'=k}^{k_c} \frac{\hat{n}_{k'}^l}{2^{k'}} \right) m + \hat{N}_k^l$$

since $m_k = m/2^k + 1$.

Finally, note that the eigenpairs $\{\lambda_{k,i}^l, \mathbf{u}_{k,i}^l\}$ $(i = 1, \ldots, n_k^l)$ computed at step 5 of the reduced memory Hessian decomposition algorithm can be discarded following step 7. The memory ratio for the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 is R_e in (4.31). Given that the finest grid at grid level $k = 0$ is composed of $m_0 = m + 1$ grid points, an estimate of the ratio of memory required for the reduced memory Hessian decomposition algorithm in terms of m is obtained by calculating

$$\hat{R}_d = \overline{N}_d + R_e \tag{5.26}$$

where

$$\overline{N}_d = \sum_{l=1}^{L} \sum_{k'=k}^{k_c} \frac{\hat{n}_{k'}^l}{2^{k'}}.$$

5.8 Approximating the inverse Hessian using the reduced memory Hessian decomposition algorithm

The reduced memory Hessian decomposition algorithm outlined in Figure 5.5 is proposed as a suitable method for constructing a limited-memory approximation to the inverse (and inverse square root) of the Hessian C in (2.50). The numerical study presented in Section 5.9 involves approximating the inverse Hessian for the specific model problem outlined in Section 4.6. This numerical study focuses on comparing various approximations to the inverse Hessian constructed using the reduced memory Hessian decomposition algorithm. The notation introduced in Section 5.4 is employed. Let $\tilde{C}_0^{-1} \in \mathbb{R}^{401 \times 401}$ denote an approximation to C_0^{-1} constructed using the reduced memory Hessian decomposition algorithm here. The additional implementation details discussed in Section 5.5 are also relevant to the reduced memory Hessian decomposition algorithm.

5.9 Comparing approximations to the inverse Hessian

The numerical study presented here focuses on the choice of N_k^l in (5.22) in the framework of the reduced memory Hessian decomposition algorithm outlined in Figure 5.5.

$\hat{R}_d$	N_k^l	D_e	$\hat{M}_e$
31	$(0, 0, 0, 8)$	$5.7919e-1$	6
38	$(0, 0, 8, 0)$	$3.3861e-1$	6
52	$(0, 8, 0, 0)$	$1.8164e-1$	6
68	$(0, 0, 0, 50)$	$5.7844e-1$	6
199	$(0, 0, 100, 0)$	$3.3436e-1$	6
724	$(0, 200, 0, 0)$	$1.7937e-1$	6

Table 5.5: The values D_e and $\hat{M}_e$ obtained for six cases of N_k^l where $k = 1$, $n_k^l = 9$ in N_d, and $N_e = (6, 12, 24, 48)$. The memory ratio $\hat{R}_d$ is also tabulated.

Specifically, various cases of $\tilde{C}_0^{-1}$ are considered. Note that these cases are symmetric positive definite. The accuracy of $\tilde{C}_0^{-1}$ is evaluated in each case using D_e in (4.39), with $\hat{M}_e$ in (5.18) also being calculated to provide an indication of computational cost.

We first suppose that there are no memory restrictions in place. The maximum allowable memory ratio $\hat{R}_d$ in (5.26) given the model problem setup is $\hat{R}_d = 3200$. The combination of $k = 0$, $n_k^l = 400$ in N_d in (5.16), and $N_e = (400, 0, 0, 0)$ in (4.25) used in the numerical study presented in Section 5.6 is considered. The case of N_k^l in (5.22) implemented is $N_k^l = (400, 0, 0, 0)$. The output $\tilde{C}_0^{-1}$ in this instance is the best approximation to C_0^{-1} achievable using the reduced memory Hessian decomposition algorithm. With these choices of n_k^l, N_e, and N_k^l, $D_e = 4.3068e-12$ and $\hat{M}_e = 401$, respectively. The eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ are plotted in Figure 5.6. The first eighty eigenvalues of $\tilde{C}_0^{-1}$ are plotted using red circles. The remaining eigenvalues of $\tilde{C}_0^{-1}$ are close to one and have been omitted. The first eighty eigenvalues of C_0^{-1} are plotted using blue circles. However, these eigenvalues are indistinguishable from the eigenvalues of $\tilde{C}_0^{-1}$ in the plot in Figure 5.6. The result $\hat{M}_e$ obtained is indicative of a high computational cost as expected.

Suppose now instead that $k = 1$. A particular combination of n_k^l and N_e previously used in Table 5.4 is considered. Specifically, we set $n_k^l = 9$ and $N_e = (6, 12, 24, 48)$. The associated values of D_e, $\hat{M}_e$, and $\hat{R}_d$ obtained for six cases of N_k^l are tabulated in Table 5.5. Note that the cases of N_k^l presented in Table 5.5 involve only one grid at grid level k $(0 < k \leq 3)$. The maximum number of eigenpairs is computed in the

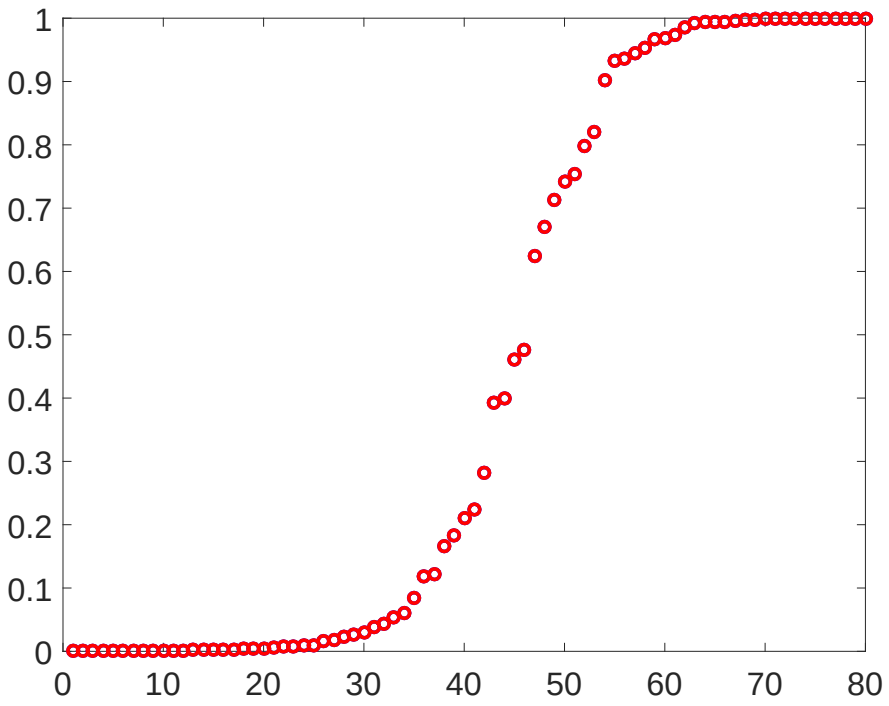
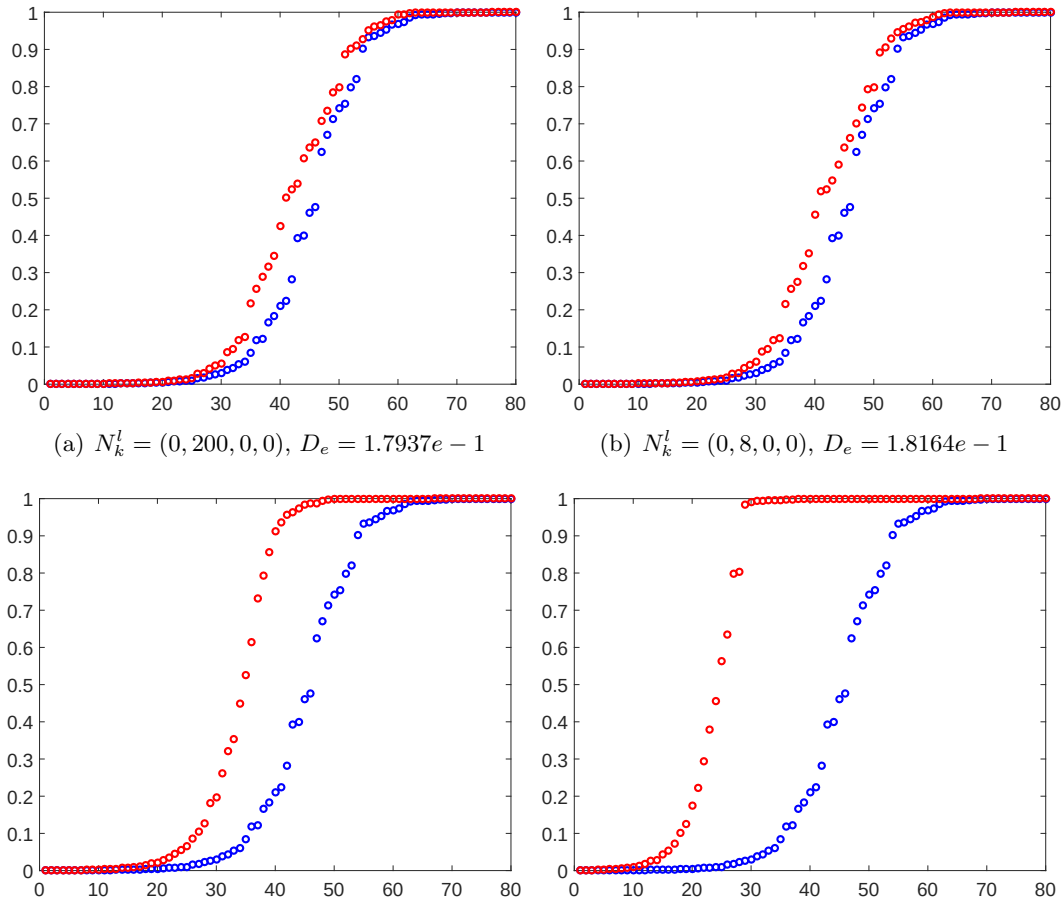


Figure 5.6: Comparison plot showing the first eighty eigenvalues of $\tilde{C}_0^{-1}$ (red circles). The first eighty eigenvalues of C_0^{-1} plotted using blue circles are indistinguishable from the eigenvalues of $\tilde{C}_0^{-1}$. The combination implemented is $k = 0$ where $n_k^l = 400$, $N_k^l = (400, 0, 0, 0)$, and $N_e = (400, 0, 0, 0)$. The associated measures of accuracy and computational cost are $D_e = 4.3068e - 12$ and $\hat{M}_e = 401$, respectively.



(a) $N_k^l = (0, 200, 0, 0)$, $D_e = 1.7937e-1$ (b) $N_k^l = (0, 8, 0, 0)$, $D_e = 1.8164e-1$

(c) $N_k^l = (0, 0, 8, 0)$, $D_e = 3.3861e-1$ (d) $N_k^l = (0, 0, 0, 8)$, $D_e = 5.7919e-1$

Figure 5.7: Comparison plots showing the first eighty eigenvalues of: C_0^{-1} (blue circles) and $\tilde{C}_0^{-1}$ (red circles). Four cases of N_k^l presented in Table 5.5 are highlighted. The results D_e obtained are presented.

cases of $N_k^l = (0, 200, 0, 0)$, $N_k^l = (0, 0, 100, 0)$, and $N_k^l = (0, 0, 0, 50)$. The output $\tilde{C}_0^{-1}$ in the case of $N_k^l = (0, 200, 0, 0)$ is the best approximation to C_0^{-1} achievable given k, n_k^l, and N_e. The first observation based on the results presented in Table 5.5 is that D_e increased, that is, the accuracy of $\tilde{C}_0^{-1}$ decreased, but $\hat{R}_d$ decreased, as the grid involved in N_k^l varied from the grid at grid level $k = 1$, to the grid at grid level $k = 3$. An additional observation is that using smaller values of $\hat{n}_k^l$ than the maximum allowable cases of $\hat{n}_k^l = 200$ for $k = 1$, $\hat{n}_k^l = 100$ for $k = 2$, and $\hat{n}_k^l = 50$ for $k = 3$ resulted in approximations $\tilde{C}_0^{-1}$ of similar accuracies in terms of D_e. The results presented in Table 5.5 demonstrate that using a smaller value of $\hat{n}_k$ than the maximum allowable case on each grid level k is a viable strategy for reducing the memory required. However, this approach has been shown to have no effect on the computational cost, as evidenced by the values $\hat{M}_e$ tabulated. The computational cost incurred by implementing the reduced memory Hessian decomposition algorithm, measured in terms of $\hat{M}_e$, results from computing the n_k^l eigenpair estimates of each local Hessian, as in the case of the Hessian decomposition algorithm outlined in Figure 5.2. The figures $\hat{M}_e$ tabulated in Table 5.5 are the same since $n_k^l = 9$ in all cases studied. The eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ for four cases of N_k^l presented in Table 5.5 are plotted in Figure 5.7. The plots in subfigures 5.7(a)-5.7(d) relate to the cases of $N_k^l = (0, 200, 0, 0)$, $N_k^l = (0, 8, 0, 0)$, $N_k^l = (0, 0, 8, 0)$, and $N_k^l = (0, 0, 0, 8)$. The first eighty eigenvalues of C_0^{-1} and $\tilde{C}_0^{-1}$ are plotted in each case using blue and red circles, respectively. The eigenvalues of $\tilde{C}_0^{-1}$ in the plots in subfigures 5.7(a) and 5.7(b) have similar distributions. This is confirmed by the results D_e attained.

The results presented demonstrate that, for a particular combination of k, N_d, and N_e, applying the reduced memory Hessian decomposition algorithm produces a less accurate approximation to C_0^{-1} than achievable using the Hessian decomposition algorithm. However, constructing an approximation to C_0^{-1} using the reduced memory Hessian decomposition algorithm has been shown to be as computationally expensive in all cases studied, as substantiated by the results $\hat{M}_e$ obtained. The results presented also show that, if N_k^l is chosen carefully, then $\hat{R}_d < R_d$, where R_d is defined in (5.17). That is, in this case, the reduced memory Hessian decomposition algorithm requires less memory than the Hessian decomposition algorithm.

5.10 Conclusion

This chapter introduced two practical algorithms for constructing a limited-memory approximation to the inverse (and inverse square root) of the Hessian C in (2.50). A modified version of the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 that involves generating a representation of the matrix A at the next finest grid level in the multilevel construction was considered in Section 5.1. A novel decomposition of the Hessian as the sum of a set of local Hessians was then derived in Section 5.2. The Hessian decomposition algorithm outlined in Figure 5.2 was the first practical algorithm introduced in this chapter. The premise was to reduce the computational cost of constructing an approximation to the inverse Hessian, or inverse square root Hessian, compared to applying the multilevel eigenvalue decomposition algorithm. The reduced memory Hessian decomposition algorithm outlined in Figure 5.5 was the second practical algorithm introduced in this chapter. The premise was to reduce the memory requirements of the Hessian decomposition algorithm.

The numerical studies presented in Sections 5.1, 5.6, and 5.9 involved approximating the inverse Hessian for the specific model problem outlined in Section 4.6. The approaches adopted included evaluating the accuracy of approximations to the inverse Hessian by calculating D_e in (4.39). The total number of Hessian vector products computed on the finest grid at grid level $k = 0$ was evaluated in the numerical study presented in Section 5.1 by calculating M_e in (4.40). An estimate of the number of Hessian vector products computed at the finest grid level $k = 0$ was obtained in the numerical studies presented in Sections 5.6 and 5.9 by calculating $\hat{M}_e$ in (5.18). The results $\hat{M}_e$ and M_e attained in these cases provided indications of computational cost.

The numerical study presented in Section 5.1 involved approximating the inverse Hessian using a generalised version of the multilevel eigenvalue decomposition algorithm. The results presented demonstrated that applying the proposed modified version of the multilevel eigenvalue decomposition algorithm produced a less accurate approximation to the inverse Hessian than could be achieved using the standard version of this algorithm. However, constructing an approximation to the inverse Hessian using the proposed modified version of the multilevel eigenvalue decomposition algorithm was

shown to be less computationally expensive in all cases studied, as substantiated by the results M_e obtained. This indicated that applying the proposed modified version of the multilevel eigenvalue decomposition algorithm was a viable strategy for reducing the computational cost.

The numerical study presented in Section 5.6 focused on comparing various approximations to the inverse Hessian constructed using the Hessian decomposition algorithm. The results presented demonstrated that, for a particular choice of N_e in (4.25), applying the Hessian decomposition algorithm produced a less accurate approximation to the inverse Hessian than could be achieved using the multilevel eigenvalue decomposition algorithm. However, constructing an approximation to the inverse Hessian using the Hessian decomposition algorithm was shown to be less computationally expensive in some cases studied where $k > 0$, since $\hat{M}_e < M_e$ in these instances. The results presented in Section 5.6 also showed that, for a specific choice of N_e, the Hessian decomposition algorithm required more memory than the multilevel eigenvalue decomposition algorithm, that is, $R_d > R_e$, where R_e and R_d are defined in (4.31) and (5.17), respectively. These results also indicated that computing estimates of a small number of the largest eigenvalues, and corresponding eigenvectors, of each local Hessian at grid level k $(0 < k \leq k_c)$ was a viable strategy for reducing the memory required and the computational cost, respectively.

The numerical study presented in Section 5.9 focused on comparing various approximations to the inverse Hessian constructed using the reduced memory Hessian decomposition algorithm. The results presented demonstrated that, for a particular combination of k, N_d in (5.16), and N_e, applying the reduced memory Hessian decomposition algorithm produced a less accurate approximation to the inverse Hessian than could be achieved using the Hessian decomposition algorithm. However, constructing an approximation to the inverse Hessian using the reduced memory Hessian decomposition algorithm was shown to be as computationally expensive in all cases studied, as substantiated by the results $\hat{M}_e$ obtained. The results presented in Section 5.9 also showed that, when N_k^l in (5.22) was chosen carefully, then $\hat{R}_d < R_d$, where $\hat{R}_d$ is defined in (5.26). That is, in this case, the reduced memory Hessian decomposition algorithm required less memory than the Hessian decomposition algorithm.

In the next chapter, the performances of the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms will be compared in the context of preconditioning the system of linear equations in the inner step of a Gauss-Newton procedure within the framework of incremental 4D-Var.

Chapter 6

Preconditioning in a Gauss-Newton procedure using approximations to the inverse Hessian

In this chapter, the effectiveness of approximations to the inverse Hessian generated using the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms outlined in Figures 4.1, 5.2, and 5.5, respectively, will be compared in a practical application, namely, as preconditioners in a preconditioned conjugate gradient (PCG) method (see Section 3.5) within an incremental 4D-Var procedure (see Section 2.6). As outlined in Section 2.5, the Gauss-Newton method can be applied in the framework of 4D-Var. Assuming that the control variable transform discussed in Section 2.9 is implemented in incremental 4D-Var, then applying the Gauss-Newton method involves solving the system of linear equations in (2.49) in an inner loop. The premise here is therefore to solve

$$C\delta\mathbf{z}_0^{(j)} = \mathbf{g} \tag{6.1}$$

where C is the Hessian defined in (2.50) and

$$\mathbf{g} = \mathbf{z}_0^b - \mathbf{z}_0^{(j)} + B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{\mathbf{d}}. \tag{6.2}$$

An appropriate iterative method is typically used to solve (6.1) in practice. The conjugate gradient (CG) method (see Section 3.4) is applicable for solving (6.1) since C is symmetric positive definite. It is usually necessary to precondition (6.1) to accelerate the convergence of the iterative scheme, as discussed in Section 3.5. An approximation to C^{-1}, if inexpensive to compute, can be used to precondition (6.1).

The application considered in this chapter focuses on constructing approximations $\tilde{C}^{-1} \in \mathbb{R}^{N\times N}$ to C^{-1} using the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms outlined in Figures 4.1, 5.2, and 5.5, respectively. Note that the cases of $\tilde{C}^{-1}$ considered are symmetric positive definite. The procedure employed involves constructing $\tilde{C}^{-1}$ once per Newton step and solving the resulting preconditioned system of linear equations by applying the preconditioned conjugate gradient (PCG) method. This differs from the approach considered in [31] where the multigrid method used to precondition the system of linear equations in (6.1) was applied on each preconditioned conjugate gradient iteration. The model problem outlined in Section 4.6 is implemented in the numerical studies presented in Sections 6.5, 6.6, and 6.7. A further three model problems are also implemented in the numerical study presented in Section 6.7: these are described in Section 6.1. The implementation details pertaining to the four model problems studied in this chapter are discussed in Section 6.2. The numerical measures used to evaluate preconditioners constructed using the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms are defined in Section 6.3. The choices of preconditioners applied in the numerical studies presented in Sections 6.5, 6.6, and 6.7 are discussed in Section 6.4. The effect of introducing preconditioning at different stages of the Gauss-Newton method is considered in Section 6.5. The numerical studies presented in Sections 6.6 and 6.7 focus on comparing the various preconditioners constructed using the multilevel eigenvalue decomposition algorithm and the Hessian decomposition, or reduced memory Hessian decomposition, algorithm.

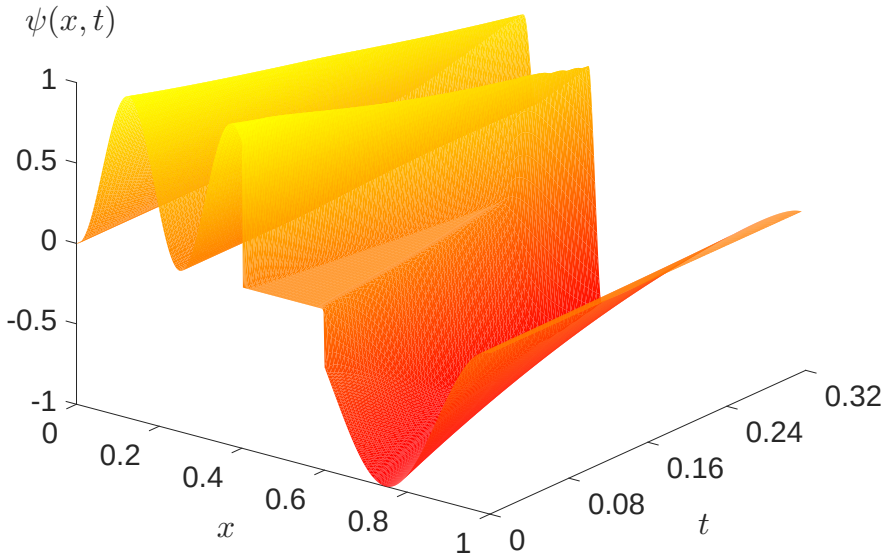


Figure 6.1: Flow evolution of $\varphi(x,t)$ satisfying (4.32)-(4.34) for initial condition $\varphi_2(x,0)$ in (6.3).

6.1 Additional model problems

As well as the model problem outlined in Section 4.6, a further three model problems are also used in the numerical studies presented in this chapter. These additional model problems result from varying the initial condition ($\varphi_1(x,0)$ in (4.35)), or the sensor configuration scheme (S_1, discussed in Section 4.6), or both.

The alternative initial condition for (4.32)-(4.34) is defined as follows:

$$\varphi_2(x,0) = \begin{cases} 0.5\left[1-\cos\left(8\pi x\right)\right], & 0 < x \leq 0.4 \\ 0.5\left[\cos\left(4\pi\left(x-1\right)\right)-1\right], & 0.6 \leq x < 1 \\ 0 & \text{otherwise} \end{cases} \cdot \tag{6.3}$$

Note that the initial conditions $\varphi_1(x,0)$ in (4.35) and $\varphi_2(x,0)$ in (6.3) differ in terms of the magnitude of non-linear effects. The flow evolution of $\varphi(x,t)$ satisfying (4.32)-(4.34) where $\varphi(x,0) = \varphi_2(x,0)$ is plotted in Figure 6.1.

The alternative sensor configuration scheme S_2 comprises of one moving sensor that traverses the spatial domain $\Omega = [0,1]$ twice during the observation time. This sensor

Model problem	Sensor scheme	Initial condition
MP1	S_1	$\varphi_1(x,0)$
MP2	S_1	$\varphi_2(x,0)$
MP3	S_2	$\varphi_1(x,0)$
MP4	S_2	$\varphi_2(x,0)$

Table 6.1: Summary of the four model problems studied.

configuration scheme is included to imitate satellite observations.

The four model problems studied in this chapter are summarised in Table 6.1. Note that model problem MP1 is used in the numerical studies presented in Sections 6.5, 6.6, and 6.7. However, the three model problems MP2, MP3, and MP4 are only used in the numerical study presented in Section 6.7.

6.2 Implementation details

The implementation details discussed in Section 4.7 apply to all four model problems in Table 6.1. Incremental 4D-Var (see Section 2.6) is implemented in each case. Specifically, the 4D-Var problem presented in Section 2.1 is solved twenty-five times with a perturbed background $\mathbf{x}_0^b$ and perturbed observations $\mathbf{y}_i$ $(i = 0, \ldots, n)$; see [44, §5.2] for details. The Gauss-Newton method is used to solve the 4D-Var problem as in Section 2.5, and twenty-five outer loop iterations are performed.

The preconditioned conjugate gradient (PCG) method outlined in Section 3.5 is used to solve the inner system of linear equations (with the Hessian as coefficient matrix) within the framework of the Gauss-Newton method. The tolerance employed in the PCG stopping criterion is $t_c = 10^{-2}$. The number of PCG iterations permitted is restricted, as this is usually the case in NWP applications. Two cases are considered in this instance, namely, $N_c = 5$ and $N_c = 10$, respectively, where N_c denotes the maximum allowable number of PCG iterations per Gauss-Newton step.

The notation introduced in Section 5.4 is employed for the four model problems summarised in Table 6.1. Let $\tilde{C}_0^{-1} \in \mathbb{R}^{401\times401}$ denote an approximation to C_0^{-1} constructed using the multilevel eigenvalue decomposition, Hessian decomposition, or reduced mem-

ory Hessian decomposition algorithm outlined in Figures 4.1, 5.2, and 5.5 in each case. The Hessian decomposition and reduced memory Hessian decomposition algorithms are applied in the numerical study presented in Section 6.7. The implementation details discussed in Section 5.5 apply to these algorithms in the cases of model problems MP1 and MP2. However, the alternative sensor configuration scheme S_2 is implemented in the cases of model problems MP3 and MP4. In these instances, the spatial domain $\Omega = [0, 1]$ is divided into six subdomains Ω_l $(l = 1, \ldots, 6)$, so $L = 6$. Specifically, $\Omega_1 = [0, 0.2)$, $\Omega_2 = [0.2, 0.4)$, $\Omega_3 = [0.4, 0.5)$, $\Omega_4 = [0.5, 0.6)$, $\Omega_5 = [0.6, 0.8)$, and $\Omega_6 = [0.8, 1]$ in this case. The observational data generated is partitioned using the approach described in Section 5.2 in order to define $\hat{I}^l$ and decompose C_k at grid level k analogously to C_k in (5.11)-(5.12). Note that the observations associated with the subdomain Ω_l are captured by C_k^l in (5.12).

6.3 Evaluating preconditioner performance

The performance of preconditioners will be evaluated by calculating two important convergence measures. The first convergence measure employed is the deviation norm. Specifically,

$$\epsilon_d = \|\mathbf{z}_0^{(j)} - \hat{\mathbf{z}}_0\|_2 \quad (j = 1, \ldots, 25) \tag{6.4}$$

where $\mathbf{z}_0^{(j)}$ is the iterate computed at iteration number j of the Gauss-Newton method, $\hat{\mathbf{z}}_0$ denotes the final estimate of $\mathbf{z}_0$, and $\|.\|_2$ represents the ℓ_2-norm. The second convergence measure used is the gradient norm. That is,

$$\epsilon_g = \|\mathbf{g}\|_2 = \|\mathbf{z}_0^b - \mathbf{z}_0^{(j)} + B^{1/2}\hat{H}^T\hat{R}^{-1}\hat{\mathbf{d}}\|_2 \quad (j = 1, \ldots, 25) \tag{6.5}$$

recalling the definition of $\mathbf{g}$ in (6.2).

Computational cost is evaluated in the numerical studies which follow using two units. The first unit, N_g, denotes the number of cost function and gradient evaluations conducted on the finest grid at grid level $k = 0$ at iteration number j of the Gauss-Newton method. The second unit, N_v, denotes the number of Hessian vector products computed on the finest grid at grid level $k = 0$ at iteration number j of the Gauss-

Newton method. The Hessian vector products counted in N_v are those associated with constructing $\tilde{C}_0^{-1}$ and applying the preconditioned conjugate gradient (PCG) method. The total number of Hessian vector products computed at the finest grid level $k = 0$ in order to construct $\tilde{C}_0^{-1}$ using the multilevel eigenvalue decomposition algorithm is evaluated by calculating M_e in (4.40). An estimate of the number of Hessian vector products computed at the finest grid level $k = 0$ to construct $\tilde{C}_0^{-1}$ using the Hessian decomposition, or reduced memory Hessian decomposition, algorithm is obtained by calculating $\hat{M}_e$ in (5.18). Note that applying the preconditioned conjugate gradient (PCG) method involves computing N_c Hessian vector products at the finest grid level $k = 0$, where N_c denotes the maximum allowable number of iterations (which is either $N_c = 5$ or $N_c = 10$ here).

Ensemble data generated for the model problems outlined in Table 6.1 is presented in Sections 6.5, 6.6, and 6.7. Specifically, the ensemble averages of ϵ_d in (6.4), ϵ_g in (6.5), N_g, and N_v are calculated; see [44, §5.2] for details. Let ϵ_d^i, ϵ_g^i, N_g^i, and N_v^i $(i = 1, \ldots, 25)$ denote ϵ_d, ϵ_g, N_g, and N_v, respectively, computed at minimisation number i of the incremental 4D-Var procedure implemented. The ensemble averages of ϵ_d, ϵ_g, N_g, and N_v are calculated as follows:

$$\hat{\epsilon}_d = \frac{1}{25}\sum_{i=1}^{25}\epsilon_d^i, \quad \hat{\epsilon}_g = \frac{1}{25}\sum_{i=1}^{25}\epsilon_g^i, \tag{6.6}$$

$$\hat{N}_g = \frac{1}{25}\sum_{i=1}^{25}N_g^i, \quad \hat{N}_v = \frac{1}{25}\sum_{i=1}^{25}N_v^i. \tag{6.7}$$

Note that ensemble average data generated for model problem MP1 summarised in Table 6.1 is presented in Sections 6.5, 6.6, and 6.7. Ensemble average data generated for the three additional model problems MP2, MP3, and MP4 is also presented in Section 6.7.

6.4 Choices of preconditioners

Various preconditioners $\tilde{C}_0^{-1}$ are applied in the numerical studies presented in this chapter. Four cases of $\tilde{C}_0^{-1}$ are constructed using the multilevel eigenvalue decomposition

Preconditioner	R_e	N_e
P1A	4	$(0, 0, 8, 16)$
P1B	8	$(0, 0, 16, 32)$
P1C	12	$(0, 8, 16, 32)$
P1D	16	$(4, 8, 16, 32)$

Table 6.2: Details of the four preconditioners generated using the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1.

Preconditioner	Algorithm	n_k^l	N_e	N_k^l
P2A	HD	5	$(0, 0, 8, 16)$	-
P2B	HD	7	$(0, 0, 16, 32)$	-
P2C	HD	9	$(0, 8, 16, 32)$	-
P2D	HD	9	$(6, 12, 24, 48)$	-
P3A	RMHD	9	$(6, 12, 24, 48)$	$(0, 0, 8, 0)$
P3B	RMHD	9	$(6, 12, 24, 48)$	$(0, 8, 0, 0)$

Table 6.3: Details of the preconditioners generated using the Hessian decomposition ('HD') and reduced memory Hessian decomposition ('RMHD') algorithms outlined in Figures 5.2 and 5.5, respectively.

algorithm outlined in Figure 4.1, namely, those corresponding to four combinations of R_e in (4.31) and N_e in (4.25) presented in Table 4.6. These parameters are summarised in Table 6.2. Preconditioner P1B is used in the numerical study presented in Section 6.5, while all four preconditioners P1A, P1B, P1C, and P1D are used in the numerical study presented in Section 6.6.

A further four cases of $\tilde{C}_0^{-1}$ are constructed using the Hessian decomposition algorithm outlined in Figure 5.2, using four combinations of n_k^l in N_d and N_e presented in Table 5.4 (note that $k = 1$ in all cases). Also, $L = 7$ in model problems MP1 and MP2, but $L = 6$ for MP3 and MP4, respectively. The parameters used are summarised in Table 6.3 (Algorithm 'HD'). The corresponding memory ratios R_d in (5.17) for model problems MP1/MP2 and MP3/MP4 are given in Table 6.4. The four preconditioners P2A, P2B, P2C, and P2D are used in the numerical study presented in Section 6.7.

An additional two cases of $\tilde{C}_0^{-1}$ are constructed using the reduced memory Hessian

		Model problem			
		MP1/MP2		MP3/MP4	
Preconditioner	Algorithm	R_d	$\hat{R}_d$	R_d	$\hat{R}_d$
P2A	HD	22	-	19	-
P2B	HD	33	-	29	-
P2C	HD	44	-	39	-
P2D	HD	56	-	51	-
P3A	RMHD	-	38	-	36
P3B	RMHD	-	52	-	48

Table 6.4: The memory ratios R_d and $\hat{R}_d$ corresponding to the preconditioners presented in Table 6.3 for the four model problems summarised in Table 6.1.

decomposition algorithm outlined in Figure 5.5, using two instances of N_k^l from Table 5.5. Note that $k = 1$, $n_k^l = 9$ in N_d in (5.16), and $N_e = (6, 12, 24, 48)$ in (4.25) in these cases. The two preconditioners generated using the reduced memory Hessian decomposition algorithm are summarised in Table 6.3 (Algorithm 'RMHD'). The corresponding memory ratios $\hat{R}_d$ in (5.26) for model problems MP1/MP2 and MP3/MP4 are included in Table 6.4. The two preconditioners P3A and P3B are used in the numerical study presented in Section 6.7.

6.5 Introducing preconditioning at different stages of the Gauss-Newton method

In this section, the effect of introducing preconditioning at different stages of the Gauss-Newton method is considered. Specifically, preconditioning is introduced at various outer iteration numbers of the Gauss-Newton method, applied to model problem MP1. Five preconditioned conjugate gradient (PCG) iterations are used at each Gauss-Newton outer iteration, that is, $N_c = 5$. The preconditioner applied is P1B (see Table 6.2).

The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ in (6.6)-(6.7) calculated with preconditioning introduced at outer iteration numbers $j = 1, 2, 3, 6$ of the Gauss-Newton

method are plotted in the convergence diagram in Figure 6.2. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. The plots in subfigures 6.2(a) and 6.2(b) show $\log_{10}\hat{\epsilon}_d$ plotted against $\hat{N}_g$ and $\hat{N}_v$, respectively. Subfigures 6.2(c) and 6.2(d) show analogous plots for $\log_{10}\hat{\epsilon}_g$. The cases of preconditioning introduced at outer iteration numbers $j = 1, 2, 3, 6$ of the Gauss-Newton method are represented in the plots in subfigures 6.2(a)-6.2(d) using solid red, blue, green, and pink lines. The first observation based on the plots in subfigures 6.2(a)-6.2(d) is that $\hat{\epsilon}_d$ decreased as $\hat{N}_g$ and $\hat{N}_v$ increased in all instances, excluding in the case of preconditioning introduced at outer iteration number $j = 1$ of the Gauss-Newton method. Specifically, introducing preconditioning at outer iteration number $j = 1$ of the Gauss-Newton method resulted in an initial increase in $\hat{\epsilon}_d$. However, $\hat{\epsilon}_d$ decreased from outer iteration number $j = 2$ in this instance. A further observation is that $\hat{\epsilon}_g$ decreased as $\hat{N}_g$ and $\hat{N}_v$ increased in all instances. The key observation is that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ decreased faster initially as the outer iteration number of the Gauss-Newton method with preconditioning introduced varied from $j = 1$ to $j = 6$.

In Table 6.5, the ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with preconditioning introduced at outer iteration numbers $j = 1, 2, 3, 6$ are tabulated. Note that $\log_{10}\hat{\epsilon}_d$ and $\log_{10}\hat{\epsilon}_g$ are presented. The key observation based on the results tabulated in Table 6.5 is that the final value of $\hat{N}_v$ decreased as the outer iteration number of the Gauss-Newton method with preconditioning introduced varied from $j = 1$ to $j = 6$. However, the final results $\hat{N}_g$ obtained in the cases of preconditioning introduced at outer iteration numbers $j = 2, 3, 6$ of the Gauss-Newton method are the same. An additional observation is that introducing preconditioning at outer iteration number $j = 2$ of the Gauss-Newton method resulted in the smallest final values of $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$, but a larger final value of $\hat{N}_v$ than attained with preconditioning introduced at outer iteration numbers $j = 3, 6$. The computational cost, measured in terms of $\hat{N}_v$, associated with introducing preconditioning at outer iteration number $j = 2$ of the Gauss-Newton method outweighed the small gain in accuracy achieved compared to introducing preconditioning at outer iteration number $j = 6$.

The results presented demonstrate that introducing preconditioning at outer iter-

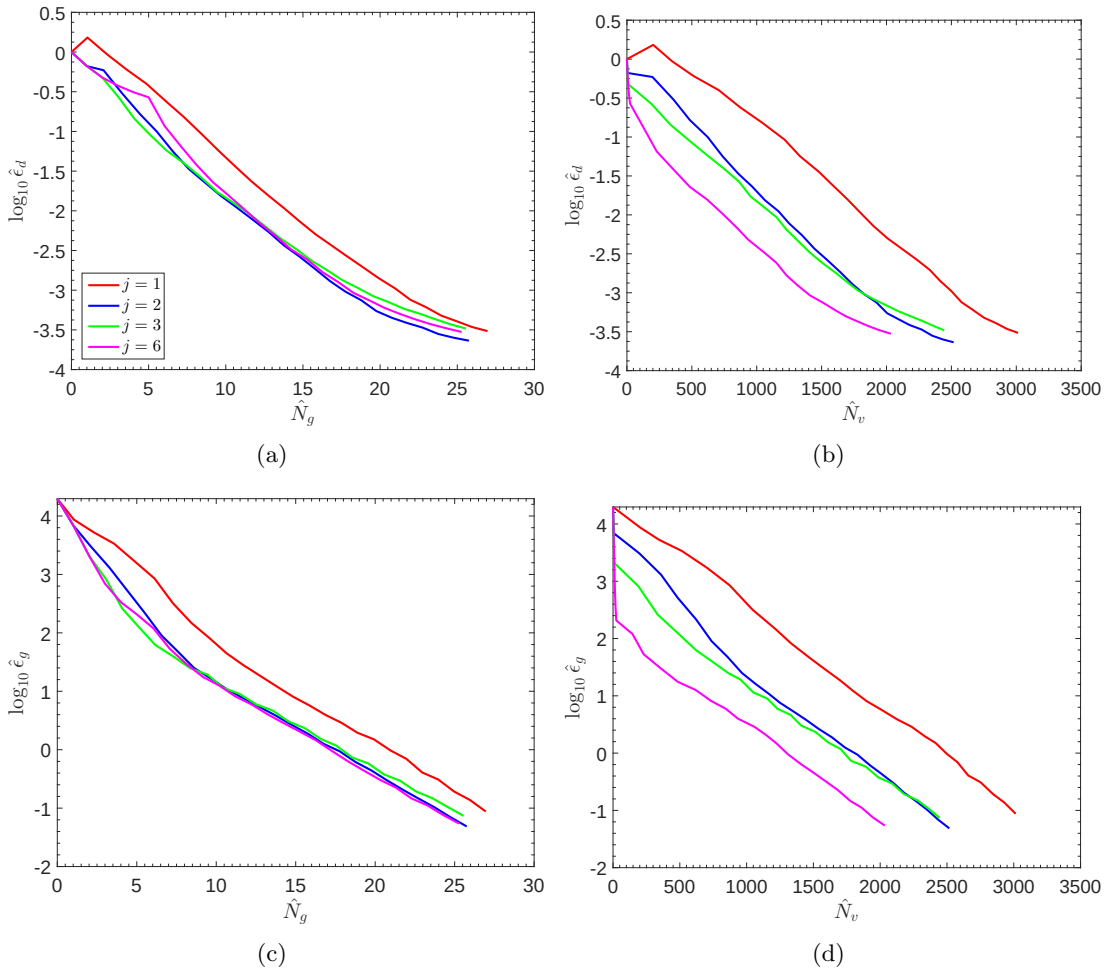


Figure 6.2: Convergence diagram for model problem MP1 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioner P1B introduced at outer iteration number j of the Gauss-Newton method: $j = 1$ (solid red line), $j = 2$ (solid blue line), $j = 3$ (solid green line), and $j = 6$ (solid pink line).

Preconditioning introduced	$\log_{10}\hat{\epsilon}_d$	$\log_{10}\hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
$j = 1$	-3.5151	-1.0585	27	3014
$j = 2$	-3.6360	-1.3136	26	2518
$j = 3$	-3.4829	-1.1311	26	2446
$j = 6$	-3.5262	-1.2644	26	2035

Table 6.5: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with preconditioning introduced at outer iteration numbers $j = 1, 2, 3, 6$. Note that $\log_{10}\hat{\epsilon}_d$ and $\log_{10}\hat{\epsilon}_g$ are presented. The model problem is MP1 where $N_c = 5$. The preconditioner applied is P1B.

ation number $j = 1$ of the Gauss-Newton method gives an intermediate solution that initially diverges. However, these results also show that no initial divergence in the intermediate solution occurs with preconditioning introduced at outer iteration numbers $j = 2, 3, 6$, suggesting a possible improvement in the stability of the method. This is the case since the Burgers' equation implemented in the four model problems summarised in Table 6.1 includes strongly non-linear constraints; see, for example, [88]. The results presented indicate that performing a number of unpreconditioned Gauss-Newton outer iterations initially, before introducing preconditioning, is a viable stratgy for reducing $\hat{N}_v$, namely, the computational cost. As introducing preconditioning at outer iteration number $j = 6$ of the Gauss-Newton method resulted in the smallest final value of $\hat{N}_v$, this approach is adopted in the subsequent numerical studies.

6.6 Comparing preconditioners constructed using the multilevel eigenvalue decomposition algorithm

In this numerical study, various preconditioners $\tilde{C}_0^{-1}$ constructed using the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 are applied to model problem MP1. Specifically, the four preconditioners P1A, P1B, P1C, and P1D are applied (see Table 6.2). Five preconditioned conjugate gradient (PCG) iterations are used at each Gauss-Newton outer iteration, that is, $N_c = 5$. Preconditioning is introduced at outer iteration number $j = 6$ of the Gauss-Newton method, as discussed in Section 6.5.

The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ in (6.6)-(6.7) calculated for the four preconditioners considered are plotted in the convergence diagram in Figure 6.3. The ensemble average data generated in the unpreconditioned case denoted by 'NP' is also plotted in the convergence diagram in Figure 6.3. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. The plots in subfigures 6.3(a) and 6.3(b) show $\log_{10} \hat{\epsilon}_d$ plotted against $\hat{N}_g$ and $\hat{N}_v$, respectively. Subfigures 6.3(c) and 6.3(d) show equivalent plots for $\log_{10} \hat{\epsilon}_g$. The preconditioners P1A, P1B, P1C, and P1D are represented in the plots in subfigures 6.3(a)-6.3(d) using solid red, blue, green, and pink lines, respectively. The unpreconditioned case is represented in the plots in subfigures 6.3(a)-6.3(d) using a solid black line. The key observation based on the plots in subfigures 6.3(a)-6.3(d) is that

Preconditioner	$\log_{10}\hat{\epsilon}_d$	$\log_{10}\hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-1.1178	0.9963	25	125
P1A	-2.1845	0.7290	26	1375
P1B	-3.5262	-1.2644	26	2035
P1C	-3.8476	-4.9869	31	4950
P1D	-3.8476	-5.9865	37	7239

Table 6.6: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P1A, P1B, P1C, and P1D, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10}\hat{\epsilon}_d$ and $\log_{10}\hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP1 where $N_c = 5$.

$\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ decreased faster, in terms of computational costs, in the four preconditioned cases studied than in the unpreconditioned case. Specifically, $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ decreased as the preconditioner applied varied from P1A to P1D.

The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method for the four preconditioners considered and the unpreconditioned case denoted by 'NP' are tabulated in Table 6.6. Note that $\log_{10}\hat{\epsilon}_d$ and $\log_{10}\hat{\epsilon}_g$ are presented. The key observation based on the results tabulated in Table 6.6 is that the final values of $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ obtained in the four preconditioned cases studied are smaller than attained in the unpreconditioned case. An additional observation is that the final values of $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ decreased in almost all instances as the preconditioner applied varied from P1A to P1D.

The results presented demonstrate that, when the memory allocated to the multilevel eigenvalue decomposition algorithm, namely, R_e in (4.31), is fixed, then applying the preconditioners generated using this algorithm increases the accuracy of the solution obtained by means of the Gauss-Newton method compared to the unpreconditioned case. These results also show that the performance of the preconditioners generated by applying the multilevel eigenvalue decomposition algorithm depends primarily on R_e. However, constructing the preconditioners using the multilevel eigenvalue decomposition algorithm has been shown to be computationally expensive in all cases studied, as substantiated by the results $\hat{N}_v$ obtained.

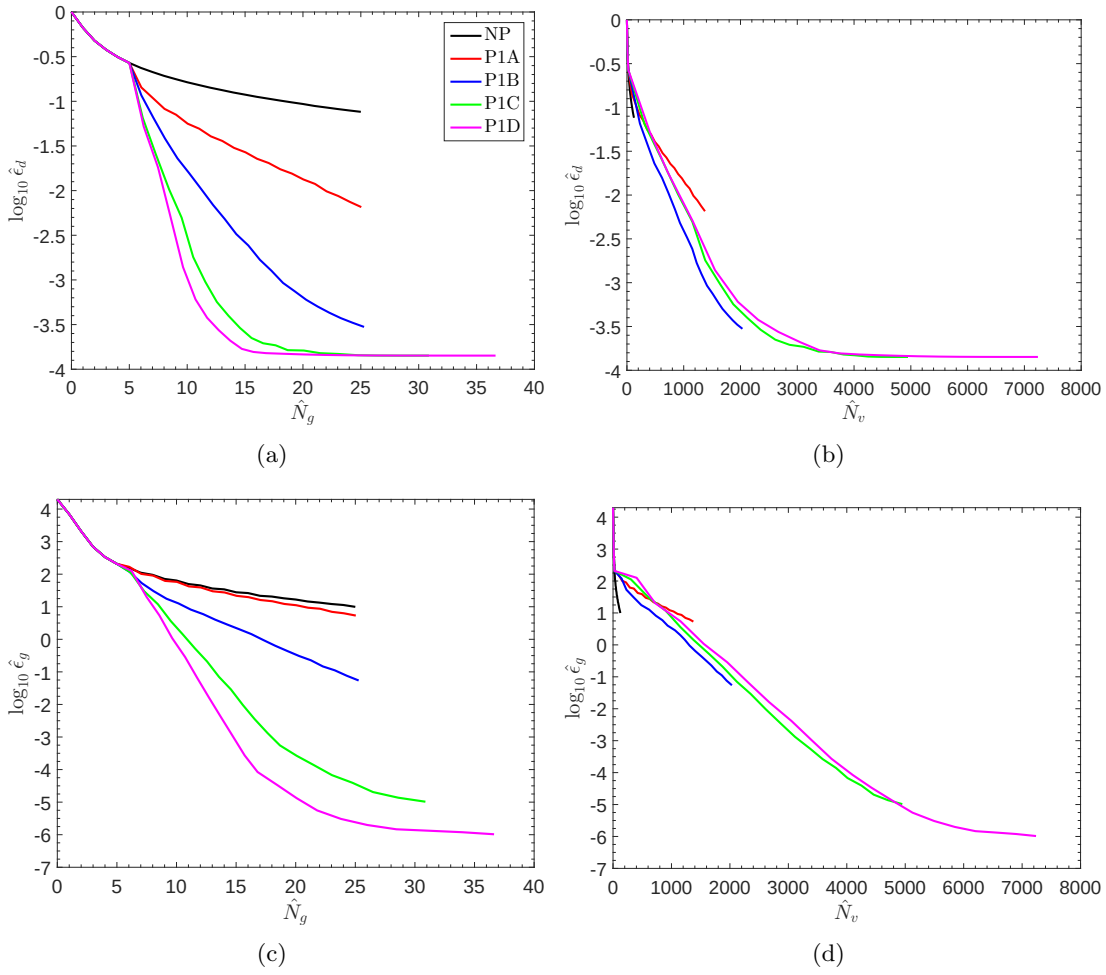


Figure 6.3: Convergence diagram for model problem MP1 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P1A (solid red line), P1B (solid blue line), P1C (solid green line), and P1D (solid pink line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

6.7 Comparing preconditioners constructed using the Hessian decomposition and reduced memory Hessian decomposition algorithms

Now various preconditioners $\tilde{C}_0^{-1}$ constructed using the Hessian decomposition and reduced memory Hessian decomposition algorithms outlined in Figures 5.2 and 5.5, respectively, are applied to the four model problems studied. Specifically, the six preconditioners P2A, P2B, P2C, P2D, P3A, and P3B are applied (see Table 6.3). Five or ten preconditioned conjugate gradient (PCG) iterations are used at each Gauss-Newton outer iteration, that is, $N_c = 5$ or $N_c = 10$. Preconditioning is introduced at outer iteration number $j = 6$ of the Gauss-Newton method as before.

The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ in (6.6)-(6.7) calculated for the six preconditioners considered are plotted in the convergence diagrams in Figures 6.4-6.11. The ensemble average data generated in the unpreconditioned cases denoted by 'NP' are also plotted in the convergence diagrams in Figures 6.4-6.11. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. The plots in Figures 6.4-6.5, 6.6-6.7, 6.8-6.9, and 6.10-6.11 pertain to model problems MP1, MP2, MP3, and MP4 outlined in Table 6.1, respectively. The plots in Figures 6.4, 6.6, 6.8, and 6.10 relate to $N_c = 5$. The plots in Figures 6.5, 6.7, 6.9, and 6.11 correspond to $N_c = 10$. The plots in subfigures 6.4(a)-6.11(a) and 6.4(b)-6.11(b) show $\log_{10} \hat{\epsilon}_d$ plotted against $\hat{N}_g$ and $\hat{N}_v$, respectively. The plots in subfigures 6.4(c)-6.11(c) and 6.4(d)-6.11(d) show analogous plots for $\log_{10} \hat{\epsilon}_g$. The preconditioners P2A, P2B, P2C, P2D, P3A, and P3B are represented in the plots in subfigures 6.4(a)-6.11(d) using solid red, dashed red, dotted red, dash-dot red, solid blue, and dashed blue lines, respectively. The unpreconditioned cases are represented in the plots in subfigures 6.4(a)-6.11(d) using solid black lines.

The key observation based on the plots in subfigures 6.4(a)-6.11(d) is that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ decreased faster, in terms of computational costs, in the six preconditioned cases studied than in the unpreconditioned case in all instances. Specifically, $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ decreased faster in most cases as N_c, namely, the number of of PCG iterations conducted, varied from $N_c = 5$ to $N_c = 10$. An additional observation is that the performances of the preconditioners considered improved in the cases of model problems MP3 and MP4. This

is due to the differences between the sensor configuration schemes S_1 and S_2 implemented in these cases. That is, the differences in observational impact with stationary or moving sensors deployed. With a stationary sensor, observational information is introduced at one specific point and propagated over the spatial domain ($\Omega = [0, 1]$ in this case). On the other hand, with a moving sensor, observational information is distributed throughout the spatial domain.

The preconditioner P2A performed worst in all cases considered. The preconditioners P2C, P2D, and P3B performed best in the case of model problem MP1 with $N_c = 5$. However, the preconditioners P2C, P2D, P3A, and P3B performed similarly in the instance where $N_c = 10$. The preconditioner P3A performed best in the cases of model problem MP2 with $N_c = 5$ and $N_c = 10$. The preconditioners P2B and P2C performed similarly in the instance where $N_c = 5$. The preconditioners P2D and P3B also performed similarly in the case of model problem MP2 with $N_c = 5$. The preconditioners P2B, P2C, P2D, and P3B performed similarly in the case of model problem MP2 with $N_c = 10$. The preconditioners P2C and P2D performed best in the case of model problem MP3 with $N_c = 5$. However, the preconditioners P2C, P2D, P3A, and P3B performed similarly in the instance where $N_c = 10$. The preconditioner P3A performed best in the case of model problem MP4 with $N_c = 5$. The preconditioners P2C, P2D, P3A, and P3B performed similarly in the instance where $N_c = 10$.

The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method for the six preconditioners considered and the unpreconditioned cases denoted by 'NP' are tabulated in Tables 6.7-6.14. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The key observation based on the results tabulated in Tables 6.7-6.14 is that the final values of $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ obtained in the six preconditioned cases are smaller than attained in the unpreconditioned case in all instances. A further observation is that the final values of $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ decreased in most cases as N_c varied from $N_c = 5$ to $N_c = 10$.

The results presented demonstrate that, when the memory allocated to the Hessian decomposition and reduced memory Hessian decomposition algorithms, namely, R_d in (5.17) and $\hat{R}_d$ in (5.26), respectively, is fixed, then applying the preconditioners generated using these algorithms increases the accuracy of the solution obtained by means

Preconditioner	$\log_{10} \hat{\epsilon}_d$	$\log_{10} \hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-1.1178	0.9963	25	125
P2A	-2.4133	0.5944	25	226
P2B	-3.7585	-1.8821	26	233
P2C	-3.8446	-5.3939	32	246
P2D	-3.8474	-5.7701	36	245
P3A	-3.8221	-4.1745	26	236
P3B	-3.8469	-5.7241	36	235

Table 6.7: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP1 where $N_c = 5$.

of the Gauss-Newton method compared to the unpreconditioned case. Furthermore, constructing the preconditioners using the Hessian decomposition and reduced memory Hessian decomposition algorithms has been shown to be less computationally expensive than applying the multilevel eigenvalue decomposition algorithm outlined in Figure 4.1 in all cases studied, as substantiated by the results $\hat{N}_v$ obtained. The results presented also show that the performance of the preconditioners generated by applying the Hessian decomposition and reduced memory Hessian decomposition algorithms depends on the particular sensor configuration scheme implemented, as well as the specific model problem studied.

6.8 Conclusion

This chapter focused on comparing the effectiveness of approximations to the inverse Hessian generated using the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms outlined in Figures 4.1, 5.2, and 5.5, respectively, in a practical application, namely, as preconditioners in a preconditioned conjugate gradient (PCG) method within a Gauss-Newton procedure in the framework of incremental 4D-Var. The numerical studies presented in this chap-

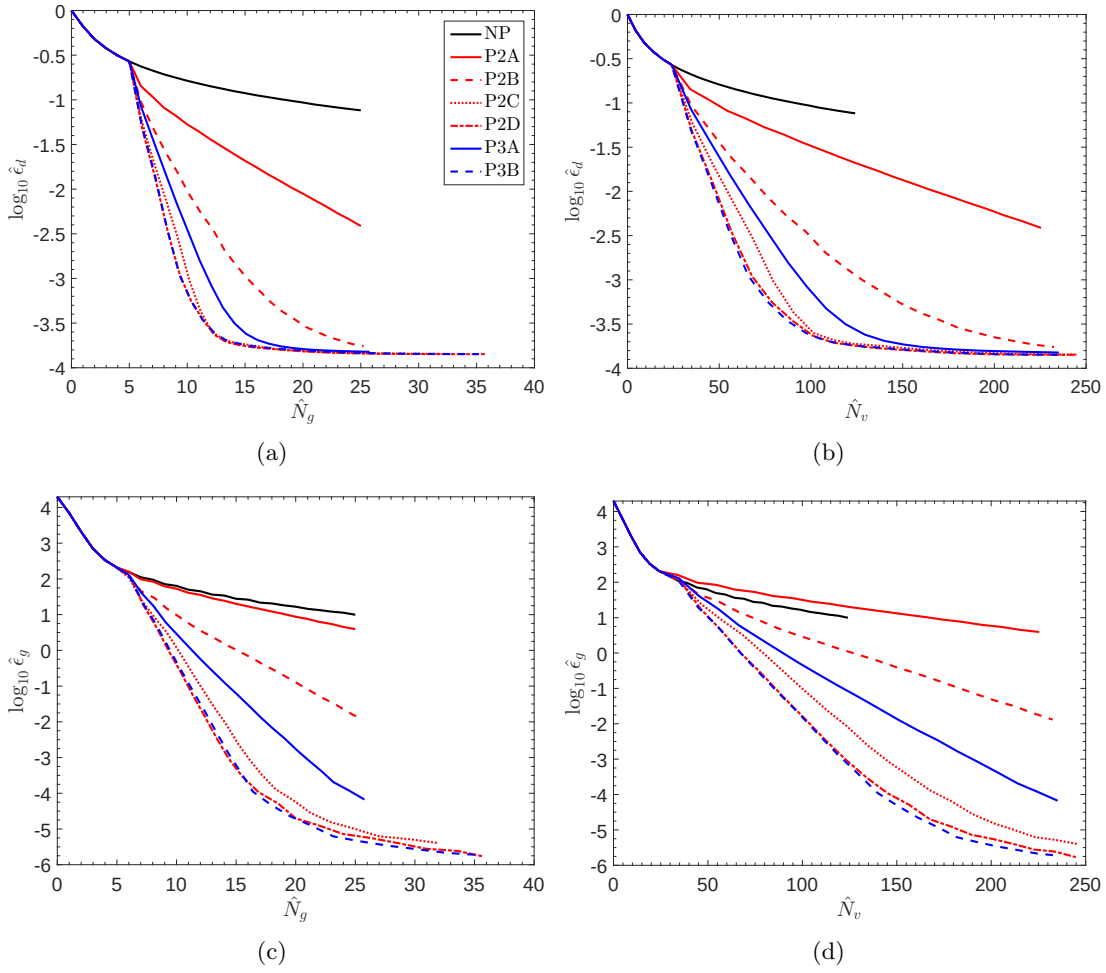


Figure 6.4: Convergence diagram for model problem MP1 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

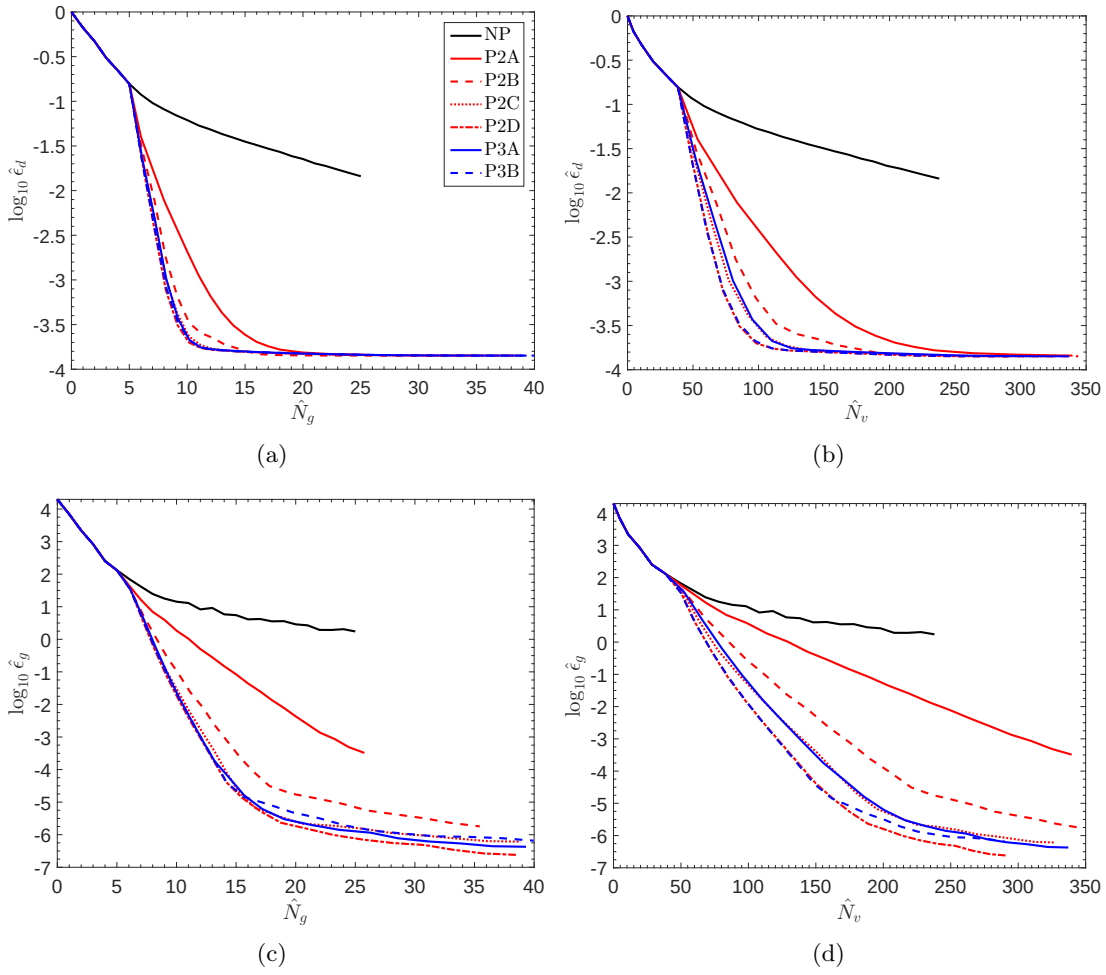


Figure 6.5: Convergence diagram for model problem MP1 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

Preconditioner	$\log_{10} \hat{\epsilon}_d$	$\log_{10} \hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-1.8393	0.2412	25	238
P2A	-3.8398	-3.4894	26	340
P2B	-3.8476	-5.7452	35	345
P2C	-3.8475	-6.2132	39	327
P2D	-3.8475	-6.6200	39	292
P3A	-3.8475	-6.3705	40	338
P3B	-3.8475	-6.1836	40	290

Table 6.8: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP1 where $N_c = 10$.

Preconditioner	$\log_{10} \hat{\epsilon}_d$	$\log_{10} \hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-0.6460	1.6306	25	124
P2A	-1.1691	1.2659	26	227
P2B	-2.1728	0.4209	26	234
P2C	-2.1349	0.5443	26	247
P2D	-1.9506	0.7940	26	247
P3A	-3.2887	-1.0045	27	240
P3B	-1.9682	0.7468	26	240

Table 6.9: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP2 where $N_c = 5$.

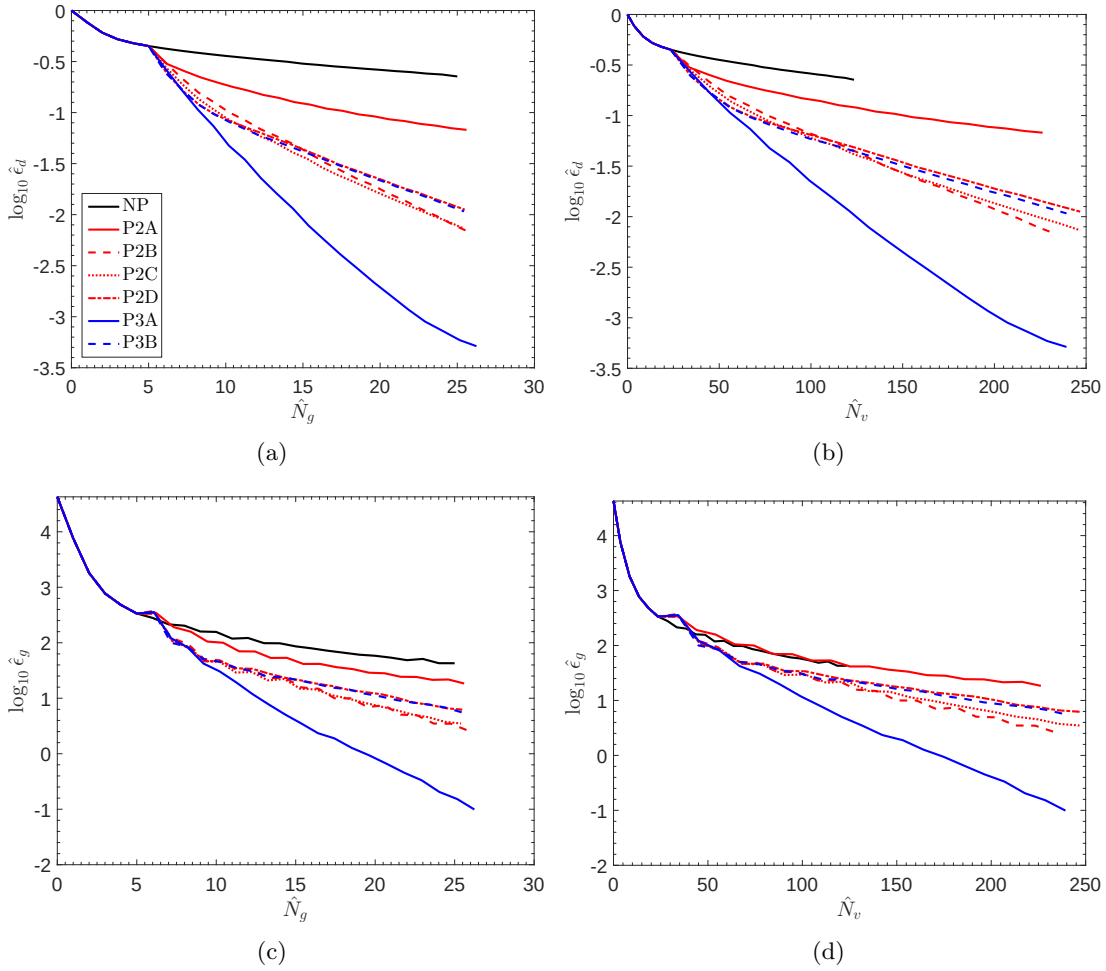


Figure 6.6: Convergence diagram for model problem MP2 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

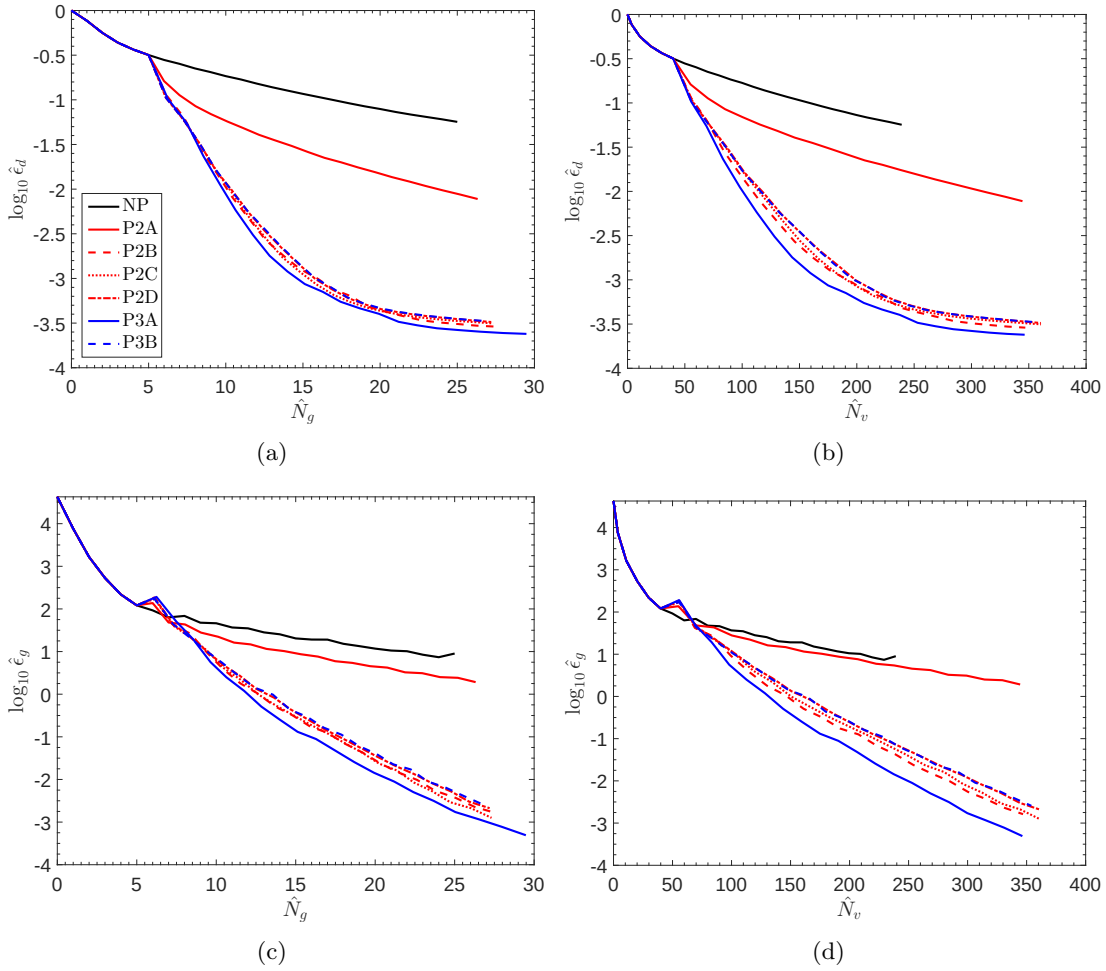


Figure 6.7: Convergence diagram for model problem MP2 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

Preconditioner	$\log_{10} \hat{\epsilon}_d$	$\log_{10} \hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-1.2467	0.9540	25	240
P2A	-2.1105	0.2832	27	345
P2B	-3.5404	-2.7891	28	347
P2C	-3.5034	-2.9003	28	361
P2D	-3.4880	-2.6738	28	361
P3A	-3.6206	-3.3085	29	347
P3B	-3.4764	-2.5911	27	354

Table 6.10: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP2 where $N_c = 10$.

Preconditioner	$\log_{10} \hat{\epsilon}_d$	$\log_{10} \hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-0.9627	0.8836	25	125
P2A	-2.9614	-0.8450	27	293
P2B	-3.6398	-2.5796	28	281
P2C	-3.8542	-3.7241	29	275
P2D	-3.8474	-3.8658	29	274
P3A	-3.8348	-3.5336	29	238
P3B	-3.8476	-3.6716	29	237

Table 6.11: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP3 where $N_c = 5$.

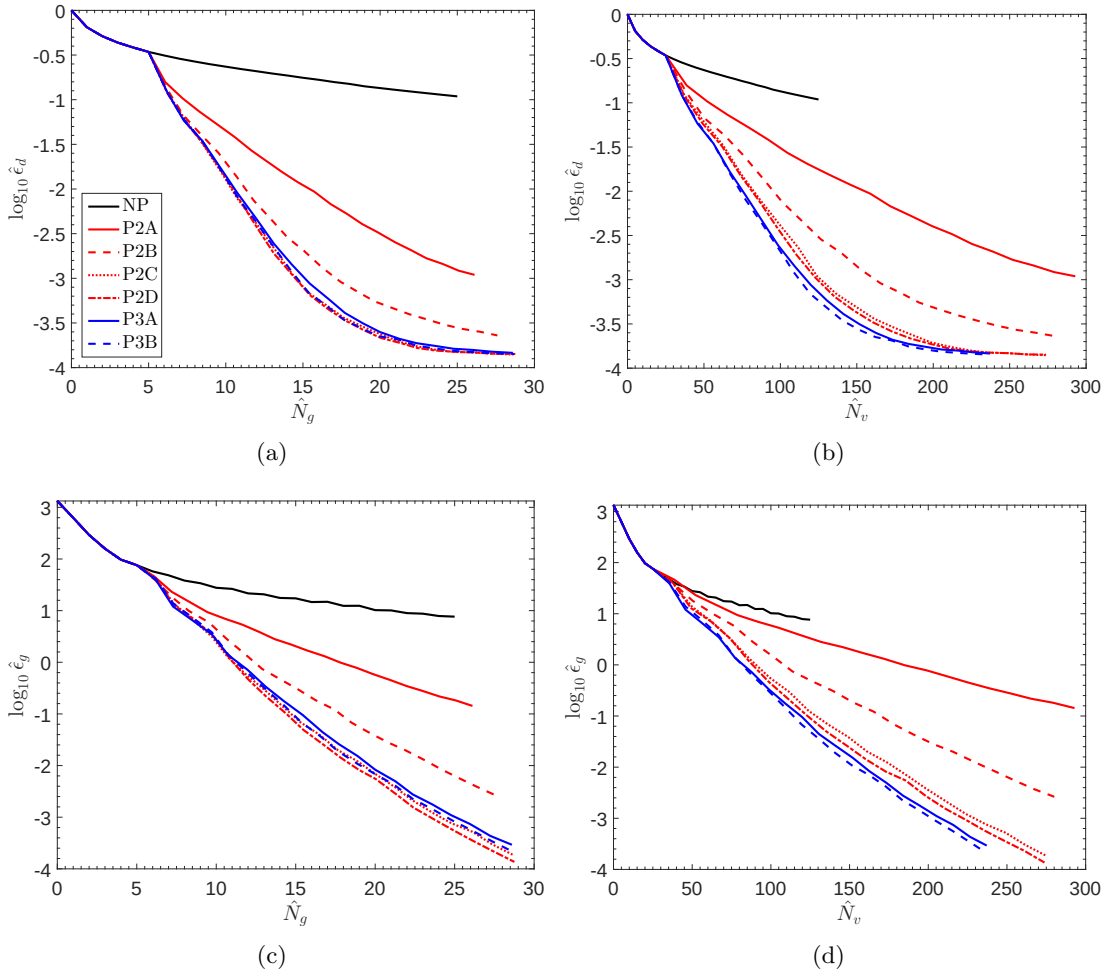


Figure 6.8: Convergence diagram for model problem MP3 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

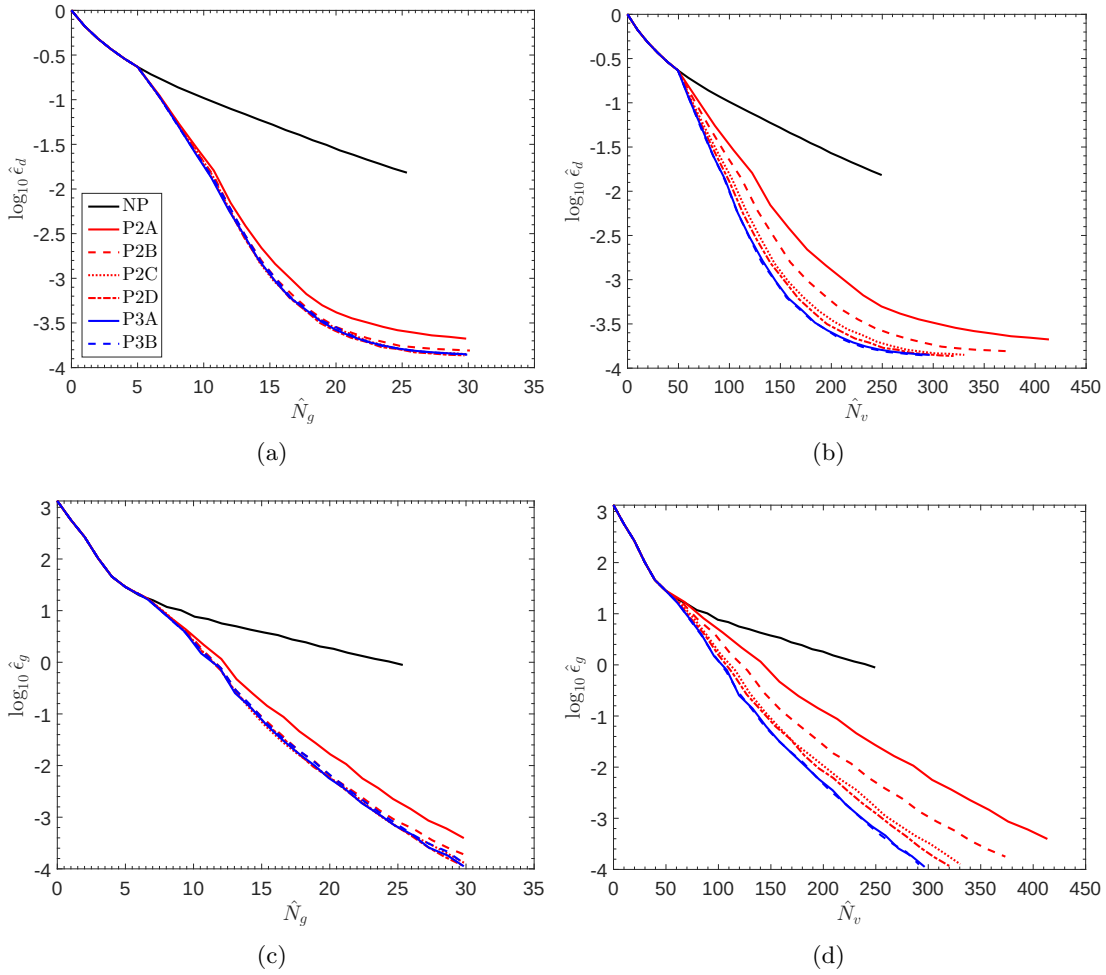


Figure 6.9: Convergence diagram for model problem MP3 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

Preconditioner	$\log_{10} \hat{\epsilon}_d$	$\log_{10} \hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-1.8171	-0.0517	26	250
P2A	-3.6757	-3.4044	30	414
P2B	-3.8088	-3.7500	31	374
P2C	-3.8498	-3.8977	30	331
P2D	-3.8625	-3.9310	30	320
P3A	-3.8506	-3.9507	30	297
P3B	-3.8552	-3.9042	30	292

Table 6.12: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP3 where $N_c = 10$.

Preconditioner	$\log_{10} \hat{\epsilon}_d$	$\log_{10} \hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-1.0165	0.7764	25	125
P2A	-2.7710	-0.1609	28	265
P2B	-3.2487	-1.4309	28	291
P2C	-3.6738	-3.5101	29	260
P2D	-3.7785	-4.0002	29	259
P3A	-3.8039	-4.1054	29	251
P3B	-3.7779	-3.8877	29	250

Table 6.13: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10} \hat{\epsilon}_d$ and $\log_{10} \hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP4 where $N_c = 5$.

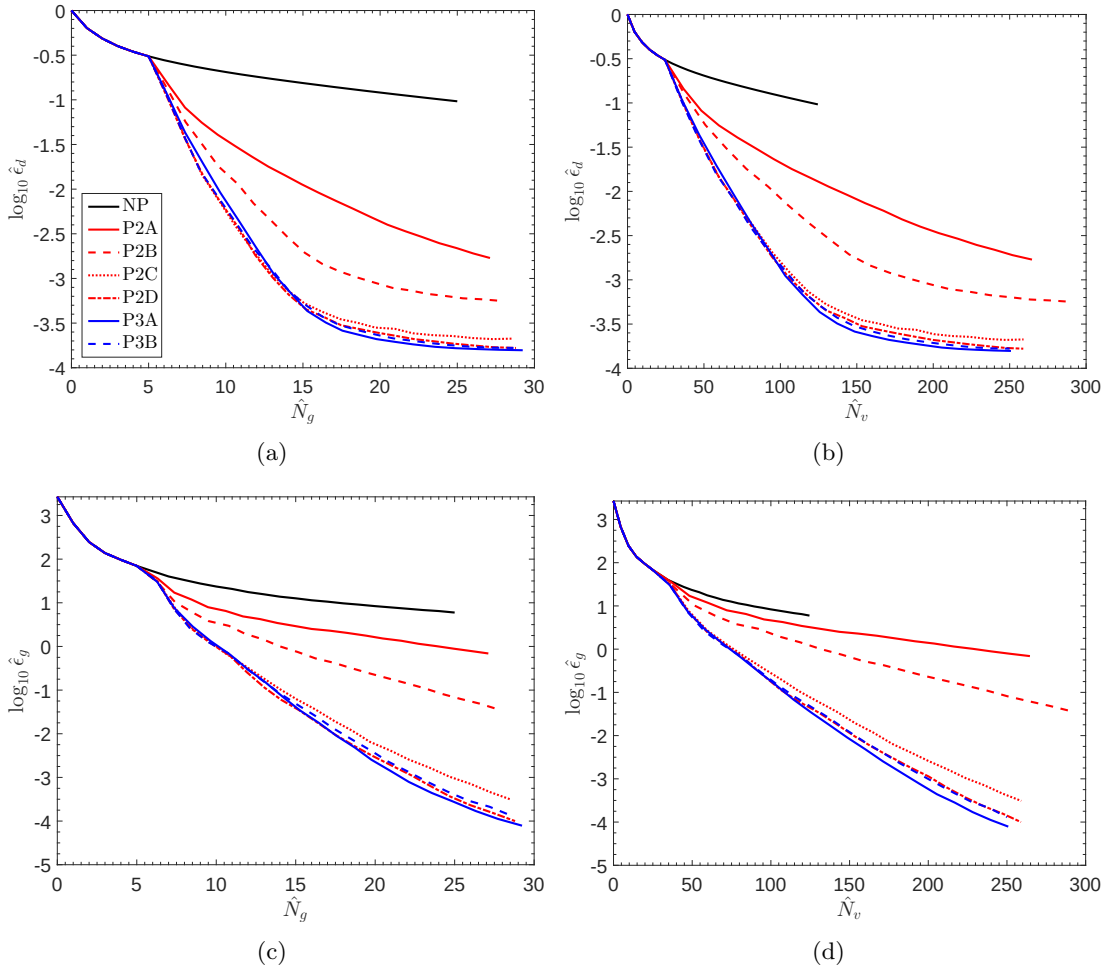


Figure 6.10: Convergence diagram for model problem MP4 where $N_c = 5$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

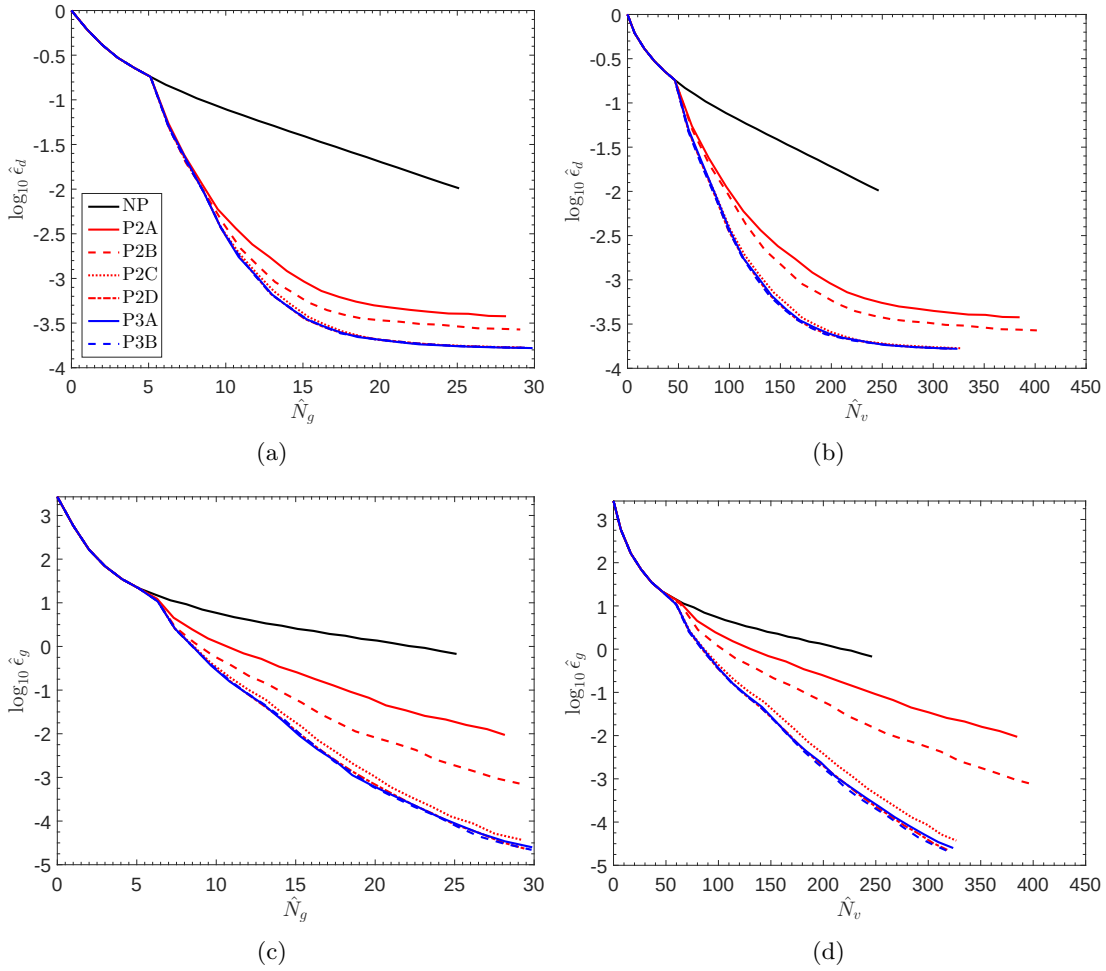


Figure 6.11: Convergence diagram for model problem MP4 where $N_c = 10$. The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ are plotted. Note that $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ are plotted on a $\log_{10}$-scale. Preconditioners P2A (solid red line), P2B (dashed red line), P2C (dotted red line), P2D (dash-dot red line), P3A (solid blue line), and P3B (dashed blue line) introduced at outer iteration number $j = 6$ of the Gauss-Newton method. The solid black line represents the unpreconditioned case denoted by 'NP'.

Preconditioner	$\log_{10}\hat{\epsilon}_d$	$\log_{10}\hat{\epsilon}_g$	$\hat{N}_g$	$\hat{N}_v$
NP	-1.9908	-0.1718	25	247
P2A	-3.4232	-2.0279	28	385
P2B	-3.5716	-3.1404	29	402
P2C	-3.7732	-4.4277	29	327
P2D	-3.7775	-4.6431	30	319
P3A	-3.7797	-4.6009	30	324
P3B	-3.7773	-4.6583	30	318

Table 6.14: The ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ calculated at outer iteration number $j = 25$ of the Gauss-Newton method with the preconditioners P2A, P2B, P2C, P2D, P3A, and P3B, respectively, introduced at outer iteration number $j = 6$. Note that $\log_{10}\hat{\epsilon}_d$ and $\log_{10}\hat{\epsilon}_g$ are presented. The results obtained in the unpreconditioned case denoted by 'NP' are also tabulated. The model problem is MP4 where $N_c = 10$.

ter involved calculating the ensemble average values $\hat{\epsilon}_d$, $\hat{\epsilon}_g$, $\hat{N}_g$, and $\hat{N}_v$ in (6.6)-(6.7) for the specific model problem outlined in Section 4.6. The numerical study presented in Section 6.7 also included ensemble average data generated for the three additional model problems outlined in Section 6.1. The values $\hat{\epsilon}_d$ and $\hat{\epsilon}_g$ obtained were used as convergence measures. The values $\hat{N}_g$ and $\hat{N}_v$ attained provided indications of computational cost.

The effect of introducing preconditioning at different stages of the Gauss-Newton method was considered in Section 6.5. The results presented demonstrated that introducing preconditioning at outer iteration number $j = 1$ of the Gauss-Newton method gave an intermediate solution that initially diverged. However, these results also showed that no initial divergence in the intermediate solution occurred with preconditioning introduced at outer iteration numbers $j = 2, 3, 6$, suggesting a possible improvement in the stability of the method. The results presented in Section 6.5 indicated that performing a number of unpreconditioned Gauss-Newton outer iterations initially, before introducing preconditioning, was a viable strategy for reducing $\hat{N}_v$, that is, the computational cost. These results showed that introducing preconditioning at outer iteration number $j = 6$ was the best strategy in terms of reducing $\hat{N}_v$.

The numerical study presented in Section 6.6 focused on comparing the various

preconditioners constructed using the multilevel eigenvalue decomposition algorithm. The results presented demonstrated that, when the memory allocated to the multilevel eigenvalue decomposition algorithm, namely, R_e in (4.31), was fixed, then applying the preconditioners generated using this algorithm increased the accuracy of the solution obtained by means of the Gauss-Newton method compared to the unpreconditioned case. These results also showed that the performance of the preconditioners generated by applying the multilevel eigenvalue decomposition algorithm depended primarily on R_e. However, constructing the preconditioners using the multilevel eigenvalue decomposition algorithm was shown to be computationally expensive in all cases studied, as substantiated by the results $\hat{N}_v$ obtained.

The numerical study presented in Section 6.7 focused on comparing the various preconditioners constructed using the Hessian decomposition and reduced memory Hessian decomposition algorithms. The results presented demonstrated that, when the memory allocated to the Hessian decomposition and reduced memory Hessian decomposition algorithms, namely, R_d in (5.17) and $\hat{R}_d$ in (5.26), respectively, was fixed, then applying the preconditioners generated using these algorithms increased the accuracy of the solution obtained by means of the Gauss-Newton method compared to the unpreconditioned case. Specifically, constructing the preconditioners using the Hessian decomposition and reduced memory Hessian decomposition algorithms was shown to be less computationally expensive than applying the multilevel eigenvalue decomposition algorithm in all cases studied, as substantiated by the results $\hat{N}_v$ obtained. The results presented in Section 6.7 also showed that the performance of the preconditioners generated by applying the Hessian decomposition and reduced memory Hessian decomposition algorithms depended on the particular sensor configuration scheme implemented, as well as the specific model problem studied.

Chapter 7

Conclusions

A multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of any given symmetric positive definite matrix with eigenvalues clustered around unity was introduced in Chapter 4. This algorithm was applied to the Hessian in the framework of incremental four-dimensional variational data assimilation (4D-Var), with the standard control variable transform implemented, in order to construct an approximation to the inverse Hessian. The accuracy of approximations to the inverse Hessian generated using the multilevel eigenvalue decomposition algorithm was investigated in Chapter 4. The key results from the numerical studies presented in Chapter 4 are as follows:

- The accuracy of the approximation to the inverse Hessian generated using the multilevel eigenvalue decomposition algorithm depended on the memory allocated to this algorithm.
- Adopting a multilevel approach in the multilevel eigenvalue decomposition algorithm produced a more accurate approximation to the inverse Hessian than could be achieved by employing a single-level procedure when the memory allocated to this algorithm was restricted.
- Applying the multilevel eigenvalue decomposition algorithm to the Hessian was computationally expensive since this involved computing Hessian vector products at the finest grid level in the multilevel setup.

The multilevel eigenvalue decomposition algorithm can be applied to other symmetric

positive definite matrices with the appropriate form. This algorithm could potentially be implemented in different application areas.

A novel decomposition of the Hessian as the sum of a set of local Hessians was introduced in the setting of incremental 4D-Var in Chapter 5. Two practical variants of the multilevel eigenvalue decomposition algorithm for constructing a limited-memory approximation to the inverse (and inverse square root) of this Hessian were presented in Chapter 5. The accuracy of approximations to the inverse Hessian generated using the two algorithms proposed was also investigated. The key results from the numerical studies presented in Chapter 5 are as follows:

- Applying the Hessian decomposition and reduced memory Hessian decomposition algorithms produced less accurate approximations to the inverse Hessian than could be achieved using the multilevel eigenvalue decomposition algorithm.
- Implementing the reduced memory Hessian decomposition algorithm produced a less accurate approximation to the inverse Hessian than could be attained using the Hessian decomposition algorithm.
- The reduced memory Hessian decomposition algorithm required less memory than the Hessian decomposition algorithm when parameters were chosen carefully.
- Generating an approximation to the inverse Hessian by applying the Hessian decomposition algorithm was less computationally expensive than implementing the multilevel eigenvalue decomposition algorithm when the local Hessians were defined at a coarser grid level than the finest grid level and a small number of eigenpairs of each local Hessian was computed.
- Constructing an approximation to the inverse Hessian using the reduced memory Hessian decomposition algorithm was as computationally expensive as applying the Hessian decomposition algorithm.

The Hessian decomposition and reduced memory Hessian decomposition algorithms may be applicable in operational implementations of incremental 4D-Var. The option of computing the local Hessians in parallel would be a useful resource in such cases.

The application considered in Chapter 6 focused on preconditioning the system of linear equations in the inner step of a Gauss-Newton procedure in incremental 4D-Var with an approximation to the inverse Hessian generated using the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms. The preconditioned conjugate gradient (PCG) method was used to solve this system of linear equations. The key results from the numerical studies presented in Chapter 6 are as follows:

- The solution obtained by means of the Gauss-Newton method was more accurate with the preconditioners generated using the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms applied than in the unpreconditioned cases.

- The performance of preconditioners generated using the multilevel eigenvalue decomposition algorithm depended primarily on the memory allocated to this algorithm.

- Generating preconditioners by applying the multilevel eigenvalue decomposition algorithm was computationally expensive since this involved computing Hessian vector products at the finest grid level in the multilevel setup.

- The performance of preconditioners generated using the Hessian decomposition and reduced memory Hessian decomposition algorithms depended primarily on the particular sensor configuration scheme implemented, as well as the specific model problem studied.

- Generating preconditioners by applying the Hessian decomposition and reduced memory Hessian decomposition algorithms was less computationally expensive than implementing the multilevel eigenvalue decomposition algorithm.

The Hessian decomposition and reduced memory Hessian decomposition algorithms may be useful for constructing preconditioners in the framework of a minimisation procedure in operational implementations of incremental 4D-Var. The option of computing the local Hessians in parallel would be particularly relevant in this context.

The novel concepts introduced in this thesis and published in [15] could be studied further by considering higher dimensional problems. Alternative sensor configuration schemes could also be considered. The multilevel eigenvalue decomposition algorithm presented in Chapter 4 could potentially be applied to other symmetric positive definite matrices with the appropriate form in different application areas.

The model problems considered in this thesis were small-scale and based on a one-dimensional Burgers' equation. An obvious next step would be to study a two-dimensional model problem on a larger scale. This could potentially facilitate the application of the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms in a practical 4D-Var setting.

In all of the numerical studies presented in this thesis, the outputs of the multilevel eigenvalue decomposition, Hessian decomposition, and reduced memory Hessian decomposition algorithms were symmetric positive definite. However, there were no theoretical proofs of this given for the general case. Further study focusing on the theoretical aspects of the three algorithms proposed in this thesis would also be a potential next step.

Appendix A

Tangent linear problem for Burgers' equation

The tangent linear problem associated with the one-dimensional Burgers' equation in (4.32)-(4.34) is

$$\frac{\partial v}{\partial t}+\frac{\partial}{\partial x}(\varphi v) = \frac{\partial}{\partial x}\left(\mu(\varphi)\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial x}\left(\mu'(\varphi)v\frac{\partial \varphi}{\partial x}\right), \quad v = v(x,t), \quad x \in [0,1], \quad t \in [0,T] \tag{A.1}$$

subject to the boundary conditions

$$\left.\frac{\partial v}{\partial x}\right|_{x=0} = 0, \quad \left.\frac{\partial v}{\partial x}\right|_{x=1} = 0 \tag{A.2}$$

where $v(x,0) = \hat{v}(x)$ and

$$\mu'(\varphi) = \frac{\partial}{\partial x}\left(\mu(\varphi)\right) = 2\rho_1\left(\frac{\partial \varphi}{\partial x}\right)\frac{\partial^2 \varphi}{\partial x^2}.$$

The tangent linear model defined by (A.1)-(A.2), and the associated adjoint model, are generated in practice using the automatic differentiation engine TAPENADE; see [58].

Bibliography

[1] E. Andersson, J. Haseler, P. Undén, P. Courtier, G. Kelly, D. Vasiljevic, C. Brankovic, C. Gaffard, A. Hollingsworth, C. Jakob, P. Janssen, E. Klinker, A. Lanzinger, M. Miller, F. Rabier, A. Simmons, B. Strauss, P. Viterbo, C. Cardinali, and J.-N. Thépaut. The ECMWF implementation of three-dimensional variational assimilation (3D-Var). III: Experimental results. *Quarterly Journal of the Royal Meteorological Society*, 124:1831–1860, 1998.

[2] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9:17–29, 1951.

[3] M. Asch, M. Bocquet, and M. Nodet. *Data Assimilation: Methods, Algorithms, and Applications*. SIAM, 2016.

[4] O. Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics*, 25:165–187, 1985.

[5] O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, 1994.

[6] J. Baglama, D. Calvetti, and L. Reichel. Iterative methods for the computation of a few eigenvalues of a large symmetric matrix. *BIT Numerical Mathematics*, 36:400–421, 1996.

[7] R. N. Bannister. A review of forecast error covariance statistics in atmospheric variational data assimilation. I: Characteristics and measurements of forecast error covariances. *Quarterly Journal of the Royal Meteorological Society*, 134:1951–1970, 2008.

[8] R. N. Bannister. A review of forecast error covariance statistics in atmospheric variational data assimilation. II: Modelling the forecast error covariance statistics. *Quarterly Journal of the Royal Meteorological Society*, 134:1971–1996, 2008.

[9] L. Bengtsson, M. Ghil, and E. Källén, editors. *Dynamic Meteorology: Data Assimilation Methods*. Springer-Verlag New York, 1981.

[10] M. Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, 182:418–477, 2002.

[11] A. Borzi and V. Schulz. Multigrid methods for PDE optimization. *SIAM Review*, 51:361–395, 2009.

[12] F. Bouttier and P. Courtier. Data assimilation concepts and methods. *Meteorological Training Course Lecture Series*, 2002.

[13] A. M. Brauset. *A Survey of Preconditioned Iterative Methods*. CRC Press, 1995.

[14] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, second edition, 2000.

[15] K. L. Brown, I. Gejadze, and A. Ramage. A multilevel approach for computing the limited-memory Hessian and its inverse in variational data assimilation. *SIAM Journal on Scientific Computing*, 38:A2934–A2963, 2016.

[16] M. Buehner, J. Morneau, and C. Charette. Four-dimensional ensemble-variational data assimilation for global deterministic weather prediction. *Nonlinear Processes in Geophysics*, 20:669–682, 2013.

[17] D. Calvetti, L. Reichel, and D.C. Sorensen. An implicitly restarted Lanczos method for large symmetric eigenvalue problems. *Electronic Transactions on Numerical Analysis*, 2:1–21, 1994.

[18] F. Chatelin. *Eigenvalues of Matrices: Revised Edition*. SIAM, 2012.

[19] Y. Chen and D. S. Oliver. Ensemble randomized maximum likelihood method as an iterative ensemble smoother. *Mathematical Geosciences*, 44:1–26, 2012.

[20] A. M. Clayton, A. C. Lorenc, and D. M. Barker. Operational implementation of a hybrid ensemble/4D-Var global data assimilation system at the Met Office. *Quarterly Journal of the Royal Meteorological Society*, 139:1445–1461, 2013.

[21] S. Costiner and S. Ta'asan. Adaptive multigrid techniques for large-scale eigenvalue problems: solutions of the Schrödinger problem in two and three dimensions. *Physical Review E*, 51:3704, 1995.

[22] P. Courtier. Variational methods. *Journal of the Meteorological Society of Japan. Ser. II*, 75:211–218, 1997.

[23] P. Courtier, E. Andersson, W. Heckley, D. Vasiljevic, M. Hamrud, A. Hollingsworth, F. Rabier, M. Fisher, and J. Pailleux. The ECMWF implementation of three-dimensional variational assimilation (3D-Var). I: Formulation. *Quarterly Journal of the Royal Meteorological Society*, 124:1783–1807, 1998.

[24] P. Courtier, J.-N. Thépaut, and A. Hollingsworth. A strategy for operational implementation of 4D-Var, using an incremental approach. *Quarterly Journal of the Royal Meteorological Society*, 120:1367–1387, 1994.

[25] J. Cullum. The simulataneous computation of a few of the algebraically largest and smallest eigenvalues of a large, sparse, symmetric matrix. *BIT Numerical Mathematics*, 18:265–275, 1978.

[26] J. Cullum and W. E. Donath. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, real symmetric matrices. In *1974 IEEE Conference on Decision and Control including the 13th Symposium on Adaptive Processes*, pages 505–509. IEEE, 1974.

[27] J. Cullum and R. A. Willoughby. Computing eigenvalues of very large symmetric matrices—an implementation of a Lanczos algorithm with no reorthogonalization. *Journal of Computational Physics*, 44:329–358, 1981.

[28] J. Cullum and R. A. Willoughby. A survey of Lanczos procedures for very large real 'symmetric' eigenvalue problems. *Journal of Computational and Applied Mathematics*, 12:37–60, 1985.

[29] J. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Vol. I: Theory.* SIAM, 2002.

[30] R. Daley. *Atmospheric Data Analysis.* Cambridge University Press, 1991.

[31] L. Debreu, E. Neveu, E. Simon, F.-X. Le Dimet, and A. Vidard. Multigrid solvers and multigrid preconditioners for the solution of variational data assimilation problems. *Quarterly Journal of the Royal Meteorological Society*, 142:515–528, 2016.

[32] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* SIAM, 1996.

[33] M. Ehrendorfer and J. J. Tribbia. Optimal prediction of forecast error covariances through singular vectors. *Journal of the Atmospheric Sciences*, 54:286–313, 1997.

[34] E. S. Epstein. The role of initial uncertainties in prediction. *Journal of Applied Meteorology*, 8:190–198, 1969.

[35] M. Fisher. Minimization algorithms for variational data assimilation. In *Seminar on Recent Developments in Numerical Methods for Atmospheric Modelling, 7-11 September 1998*, pages 364–385. ECMWF, 1998.

[36] M. Fisher, J. Nocedal, Y. Trémolet, and S. J. Wright. Data assimilation in weather forecasting: a case study in PDE-constrained optimization. *Optimization and Engineering*, 10:409–426, 2009.

[37] D. A. Flanders and G. Shortley. Numerical determination of fundamental modes. *Journal of Applied Physics*, 21:1326–1332, 1950.

[38] S. J. Fletcher. *Data Assimilation for the Geosciences: From Theory to Application.* Elsevier, 2017.

[39] A. M. Fowler and A. S. Lawless. An idealized study of coupled atmosphere-ocean 4D-Var in the presence of model error. *Monthly Weather Review*, 144:4007–4030, 2016.

[40] J. G. F. Francis. The QR transformation a unitary analogue to the LR transformation - Part 1. *The Computer Journal*, 4:265–271, 1961.

[41] J. G. F. Francis. The QR transformation - Part 2. *The Computer Journal*, 4:332–345, 1962.

[42] D. Furbish, M. Y. Hussaini, F.-X. Le Dimet, P. Ngnepieba, and Y. Wu. On discretization error and its control in variational data assimilation. *Tellus A*, 60:979–991, 2008.

[43] P. Gauthier, C. Charette, L. Fillion, P. Koclas, and S. Laroche. Implementation of a 3D variational data assimilation system at the Canadian Meteorological Centre. Part I: The global analysis. *Atmosphere-Ocean*, 37:103–156, 1999.

[44] I. Yu. Gejadze, F.-X. Le Dimet, and V. Shutyaev. On analysis error covariances in variational data assimilation. *SIAM Journal on Scientific Computing*, 30:1847–1874, 2008.

[45] I. Yu. Gejadze, F.-X. Le Dimet, and V. Shutyaev. On optimal solution error covariances in variational data assimilation problems. *Journal of Computational Physics*, 229:2159–2178, 2010.

[46] M. Ghil and P. Malanotte-Rizzoli. Data assimilation in meteorology and oceanography. *Advances in Geophysics*, 33:141–266, 1991.

[47] R. Giering and T. Kaminski. Recipes for adjoint code construction. *ACM Transactions on Mathematical Software*, 24:437–474, 1998.

[48] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[49] G. H. Golub and R. Underwood. The block Lanczos method for computing eigenvalues. In J. R. Rice, editor, *Mathematical Software*, pages 361–377. Academic Press, 1977.

[50] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, third edition, 1996.

[51] S. Gratton, A. S. Lawless, and N. K. Nichols. Approximate Gauss-Newton methods for nonlinear least squares problems. *SIAM Journal on Optimization*, 18:106–132, 2007.

[52] A. Greenbaum and Z. Strakos. Predicting the behaviour of finite precision Lanczos and conjugate gradient computations. *SIAM Journal on Matrix Analysis and Applications*, 13:121–137, 1992.

[53] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, second edition, 2008.

[54] S. A. Haben, A. S. Lawless, and N. K. Nichols. Conditioning and preconditioning of the variational data assimilation problem. *Computers & Fluids*, 46:252–256, 2011.

[55] W. Hackbusch. On the computation of approximate eigenvalues and eigenfunctions of elliptic operators by means of a multi-grid method. *SIAM Journal on Numerical Analysis*, 16:201–215, 1979.

[56] W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer-Verlag New York, first edition, 1994.

[57] W. Hackbush. *Multi-Grid Methods and Applications*. Springer-Verlag Berlin Heidelberg, 1985.

[58] L. Hascoët and V. Pascual. TAPENADE 2.1 user's guide. Technical report, INRIA, Sophia Antipolis, 2004.

[59] M. R. Hestenes. *Conjugate Direction Methods in Optimization*. Springer-Verlag New York, 1980.

[60] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[61] A. Jennings. Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method. *IMA Journal of Applied Mathematics*, 20:61–72, 1977.

[62] E. Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, 2003.

[63] S. Kaniel. Estimates for some computational techniques in linear algebra. *Mathematics of Computation*, 20:369–378, 1966.

[64] E. Klinker, F. Rabier, G. Kelly, and J.-F. Mahfouf. The ECMWF operational implementation of four-dimensional variational assimilation. III: Experimental results and diagnostics with operational configuration. *Quarterly Journal of the Royal Meteorological Society*, 126:1191–1215, 2000.

[65] A. V. Knyazev and K. Neymeyr. Efficient solution of symmetric eigenvalue problems using multigrid preconditioners in the locally optimal block conjugate gradient method. *Electronic Transactions on Numerical Analysis*, 15:38–55, 2003.

[66] V. N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Computational Mathematics and Mathematical Physics*, 1:637–657, 1962.

[67] W. Lahoz, B. Khattatov, and R. Menard, editors. *Data Assimilation: Making Sense of Observations*. Springer-Verlag Berlin Heidelberg, 2010.

[68] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45:255–282, 1950.

[69] A. S. Lawless. Variational data assimilation for very large environmental problems. In M. Cullen, M. A. Freitag, S. Kindermann, and R. Scheichl, editors, *Large Scale Inverse Problems: Computational Methods and Applications in the Earth Sciences*, pages 55–90. De Gruyter, 2013.

[70] A. S. Lawless, S. Gratton, and N. K. Nichols. An investigation of incremental 4D-Var using non-tangent linear models. *Quarterly Journal of the Royal Meteorological Society*, 131:459–476, 2005.

[71] A. S. Lawless and N. K. Nichols. Inner-loop stopping criteria for incremental four-dimensional variational data assimilation. *Monthly Weather Review*, 134:3425–3435, 2006.

[72] A. S. Lawless, N. K. Nichols, C. Boess, and A. Bunse-Gerstner. Using model reduction methods within incremental four-dimensional variational data assimilation. *Monthly Weather Review*, 136:1511–1522, 2008.

[73] F.-X. Le Dimet, I. M. Navon, and R. Ştefănescu. Variational data assimilation: optimization and optimal control. In S. K. Park and L. Xu, editors, *Data Assimilation for Atmospheric, Oceanic, and Hydrologic Applications (Vol. III)*, pages 1–53. Springer, Cham, 2017.

[74] F.-X. Le Dimet, I. M. Navon, and D. N. Daescu. Second-order information in data assimilation. *Monthly Weather Review*, 130:629–648, 2002.

[75] F.-X. Le Dimet and O. Talagrand. Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects. *Tellus A*, 38:97–110, 1986.

[76] R. B. Lehoucq. *Analysis and implementation of an implicitly restarted Arnoldi iteration.* PhD thesis, Rice University, Houston, Texas, 1995.

[77] R. B. Lehoucq and D. C. Sorensen. Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM Journal on Matrix Analysis and Applications*, 17:789–821, 1996.

[78] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK users' guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.* SIAM, 1998.

[79] M. Leutbecher and T. N. Palmer. Ensemble forecasting. *Journal of Computational Physics*, 227:3515–3539, 2008.

[80] C. Liu, Q. Xiao, and B. Wang. An ensemble-based four-dimensional variational data assimilation scheme. Part I: Technical formulation and preliminary test. *Monthly Weather Review*, 136:3363–3373, 2008.

[81] A. C. Lorenc. Analysis methods for numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, 112:1177–1194, 1986.

[82] A. C. Lorenc. Optimal nonlinear objective analysis. *Quarterly Journal of the Royal Meteorological Society*, 114:205–240, 1988.

[83] A. C. Lorenc. Development of an operational variational assimilation scheme. *Journal of the Meteorological Society of Japan*, 75:339–346, 1997.

[84] A. C. Lorenc, S. P. Ballard, R. S. Bell, N. B. Ingleby, P. L. F. Andrews, D. M. Barker, J. R. Bray, A. M. Clayton, T. Dalby, D. Li, T. J. Payne, and F. W. Saunders. The Met. Office global three-dimensional variational data assimilation scheme. *Quarterly Journal of the Royal Meteorological Society*, 126:2991–3012, 2000.

[85] J. F. Mahfouf and F. Rabier. The ECMWF operational implementation of four-dimensional variational assimilation. II: Experimental results with improved physics. *Quarterly Journal of the Royal Meteorological Society*, 126:1171–1190, 2000.

[86] M. Moakher. A differential geometric approach to the geometric mean of symmetric positive-definite matrices. *SIAM Journal on Matrix Analysis and Applications*, 26:735–747, 2005.

[87] I. M. Navon. Data assimilation for numerical weather prediction: a review. In S. K. Park and L. Xu, editors, *Data Assimilation for Atmospheric, Oceanic and Hydrological Applications*, pages 21–65. Springer-Verlag Berlin Heidelberg, 2009.

[88] L. Nazareth. Some recent approaches to solving large residual nonlinear least squares problems. *SIAM Review*, 22:1–11, 1980.

[89] N. K. Nichols. Data assimilation: Aims and basic concepts. In R. Swinbank, V. Shutyaev, and W. A. Lahoz, editors, *Data Assimilation for the Earth System*, pages 9–20. Springer, Dordrecht, 2003.

[90] N. K. Nichols. Mathematical concepts of data assimilation. In W. Lahoz, B. Khattatov, and R. Menard, editors, *Data Assimilation: Making Sense of Observations*, pages 13–39. Springer, Berlin, Heidelberg, 2010.

[91] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag New York, second edition, 2006.

[92] C. C. Paige. Practical use of the symmetric Lanczos process with reorthogonalization. *BIT Numerical Mathematics*, 10:183–195, 1970.

[93] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices*. PhD thesis, University of London, 1971.

[94] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10:373–381, 1972.

[95] C. C. Paige. Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *IMA Journal of Applied Mathematics*, 18:341–349, 1976.

[96] C. C. Paige. Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Algebra and its Applications*, 34:235–258, 1980.

[97] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, 1998.

[98] B. N. Parlett and D. S. Scott. The Lanczos algorithm with selective orthogonalization. *Mathematics of Computation*, 33:217–238, 1979.

[99] S. Patankar. *Numerical Heat Transfer and Fluid Flow*. CRC Press, 1980.

[100] F. Rabier. Overview of global data assimilation developments in numerical weather-prediction centres. *Quarterly Journal of the Royal Meteorological Society*, 131:3215–3233, 2005.

[101] F. Rabier and P. Courtier. Four-dimensional assimilation in the presence of baroclinic instability. *Quarterly Journal of the Royal Meteorological Society*, 118:649–672, 1992.

[102] F. Rabier, H. Järvinen, E. Klinker, J.-F. Mahfouf, and A. Simmons. The ECMWF operational implementation of four-dimensional variational assimilation. I: Experimental results with simplified physics. *Quarterly Journal of the Royal Meteorological Society*, 126:1143–1170, 2000.

[103] F. Rabier, A. McNally, E. Andersson, P. Courtier, P. Undén, J. Eyre, A. Hollingsworth, and F. Bouttier. The ECMWF implementation of three-dimensional variational assimilation (3D-Var). II: Structure functions. *Quarterly Journal of the Royal Meteorological Society*, 124:1809–1829, 1998.

[104] F. Rawlins, S. P. Ballard, K. J. Bovis, A. M. Clayton, D. Li, G. W. Inverarity, A. C. Lorenc, and T. J. Payne. The Met Office global four-dimensional variational data assimilation scheme. *Quarterly Journal of the Royal Meteorological Society*, 133:347–362, 2007.

[105] Y. Saad. On the rates of convergence of the Lanczos and the block-Lanczos methods. *SIAM Journal on Numerical Analysis*, 17:687–706, 1980.

[106] Y. Saad. *Iterative Methods for Sparse Linear Systems.* SIAM, second edition, 2003.

[107] Y. Saad. *Numerical Methods for Large Eigenvalue Problems: Revised Edition.* SIAM, 2011.

[108] M. Sharan and J. P. Issartel, editors. *Data Assimilation and its Applications.* Birkhäuser Basel, 2012.

[109] H. D. Simon. Analysis of the symmetric Lanczos algorithm with reorthogonalization methods. *Linear Algebra and its Applications*, 61:101–131, 1984.

[110] P. J. Smith, A. M. Fowler, and A. S. Lawless. Exploring strategies for coupled 4D-Var data assimilation using an idealised atmosphere-ocean model. *Tellus A: Dynamic Meteorology and Oceanography*, 67:27025, 2015.

[111] D. C. Sorensen. Implicit application of polynomial filters in a k-step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications*, 13:357–385, 1992.

[112] D. C. Sorensen. Numerical methods for large eigenvalue problems. *Acta Numerica*, 11:519–584, 2002.

[113] G. W. Stewart. The convergence of the method of conjugate gradients at isolated extreme points in the spectrum. *Numerische Mathematik*, 24:85–93, 1975.

[114] G. W. Stewart. *Matrix Algorithms Volume II: Eigensystems*. SIAM, 2001.

[115] R. Swinbank, V. Shutyaev, and W. A. Lahoz, editors. *Data Assimilation for the Earth System*. Springer Netherlands, 2003.

[116] J. M. Tabeart, S. L. Dance, S. A. Haben, A. S. Lawless, N. K. Nichols, and J. A. Waller. The conditioning of least-squares problems in variational data assimilation. *Numerical Linear Algebra with Applications*, 2018.

[117] O. Talagrand. Assimilation of observations, an introduction. *Journal of the Meteorological Society of Japan*, 75:191–209, 1997.

[118] W. C. Thacker. The role of the Hessian matrix in fitting models to measurements. *Journal of Geophysical Research*, 94:6177–6196, 1989.

[119] J.-N. Thépaut and P. Courtier. Four-dimensional variational data assimilation using the adjoint of a multilevel primitive-equation model. *Quarterly Journal of the Royal Meteorological Society*, 117:1225–1254, 1991.

[120] Z. Toth and E. Kalnay. Ensemble forecasting at NMC: The generation of perturbations. *Bulletin of the American Meteorological Society*, 74:2317–2330, 1993.

[121] Y. Trémolet. Diagnostics of linear and incremental approximations in 4D-Var. *Quarterly Journal of the Royal Meteorological Society*, 130:2233–2251, 2004.

[122] Y. Trémolet. Accounting for an imperfect model in 4D-Var. *Quarterly Journal of the Royal Meteorological Society*, 132:2483–2504, 2006.

[123] Y. Trémolet. Incremental 4D-Var convergence study. *Tellus A*, 59:706–718, 2007.

[124] Y. Trémolet. Model-error estimation in 4D-Var. *Quarterly Journal of the Royal Meteorological Society*, 133:1267–1280, 2007.

[125] J. Tshimanga, S. Gratton, A. T. Weaver, and A. Sartenaer. Limited-memory preconditioners, with application to incremental four-dimensional variational data assimilation. *Quarterly Journal of the Royal Meteorological Society*, 134:751–769, 2008.

[126] R. R. Underwood. *An iterative block Lanczos method for the solution of large sparse symmetric eigenproblems*. PhD thesis, Stanford University, 1975.

[127] A. van der Sluis and H. A. van der Vorst. The rate of convergence of conjugate gradients. *Numerische Mathematik*, 48:543–560, 1986.

[128] D. S. Watkins. Understanding the QR algorithm. *SIAM Review*, 24:427–440, 1982.

[129] D. S. Watkins. *The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods*. SIAM, 2007.

[130] D. S. Watkins. The QR algorithm revisited. *SIAM Review*, 50:133–145, 2008.

[131] P. Wesseling. *An Introduction to Multigrid Methods*. John Wiley & Sons, 1992.

[132] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, 1965.

[133] H. Wozniakowski. Roundoff-error analysis of a new class of conjugate-gradient algorithms. *Linear Algebra and its Applications*, 29:507–529, 1980.

[134] T. Yang. Theoretical error bounds on the convergence of the Lanczos and block-Lanczos methods. *Computers & Mathematics with Applications*, 38:19–38, 1999.