THÈSE

présentée en vue de l'obtention du titre de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE

Spécialité: Informatique et Télécommunications

par

Marc BABOULIN

CERFACS

Résolution de problèmes de moindres carrés linéaires denses de grande taille sur des calculateurs parallèles distribués. Application au calcul de champ de gravité terrestre.

Solving large dense linear least squares problems on parallel distributed computers. Application to the Earth's gravity field computation.

Thèse présentée le 21 Mars 2006 à Toulouse devant le jury composé de:

Jack DONGARRA	Professeur, Université du Tennessee	USA	*Rapporteur*
Nicholas J. HIGHAM	Professeur, Université de Manchester	Royaume–Uni	*Rapporteur*
Georges BALMINO	Directeur de Recherche, cnes–cnrs	France	*Examinateur*
Iain S. DUFF	Directeur de Recherche, cerfacs et ral	Royaume–Uni	*Examinateur*
Luc GIRAUD	Professeur, enseeiht	France	*Directeur de thèse*
Serge GRATTON	Chercheur Senior, cerfacs	France	*Examinateur*
Joseph NOAILLES	Professeur Emérite, enseeiht	France	*Examinateur*

Résumé

Dans cette thèse, nous présentons le résultat de nos recherches dans le domaine du calcul scientifique haute performance pour les moindres carrés linéaires. En particulier, nous nous intéressons au développement de logiciels parallèles efficaces permettant de traiter des problèmes de moindres carrés denses de très grande taille. Nous fournissons également des outils numériques permettant d'étudier la qualité de la solution. Cette thèse est aussi une contribution au projet GOCE[1] dont l'objectif est de fournir un modèle très précis du champ de gravité terrestre. Le lancement de ce satellite est prévu pour 2007 et à cet égard, notre travail constitue une étape dans la définition d'algorithmes pour ce projet.

Nous présentons d'abord les stratégies numériques susceptibles d'être utilisées pour mettre à jour la solution en prenant en compte des nouvelles observations fournies par GOCE. Puis nous décrivons un solveur parallèle distribué que nous avons développé afin d'être intégré dans le logiciel du CNES[2] chargé de la détermination d'orbite et du calcul de champ de gravité. Les performances de notre solveur sont compétitives par rapport à celles des librairies parallèles standards ScaLAPACK et PLAPACK sur les machines opérationnelles utilisées dans l'industrie spatiale, tout en nécessitant un stockage mémoire deux fois moindre grâce à la prise en compte des symétries du problème.

Afin d'améliorer le passage à l'échelle et la portabilité de notre solveur, nous définissons un format "packed" distribué qui repose sur des noyaux ScaLAPACK. Cette approche constitue une amélioration significative car il n'existe pas à ce jour de format "packed" distribué pour les matrices symétriques et triangulaires denses. Nous présentons les exemples pour la factorisation de Cholesky et la mise à jour d'une factorisation QR. Ce format peut être aisément étendu à d'autres opérations d'algèbre linéaire.

Cette thèse propose enfin des résultats nouveaux dans le domaine de l'analyse de sensibilité des moindres carrés linéaires résultant de problèmes d'estimation de paramètres. Nous proposons notamment une formule exacte, des bornes précises et des estimateurs statistiques pour évaluer le conditionnement d'une fonction linéaire de la solution d'un problème de moindres carrés. Le choix entre ces différentes formules dépendra de la taille du problème et du niveau de précision souhaité.

Mots clés: Calcul haute performance, moindres carrés linéaires, algorithmes parallèles distribués, format de stockage "packed", factorisation de Cholesky, factorisation QR et mise à jour, conditionnement "normwise", estimateur statistique de conditionnement, estimation de paramètres.

---

[1]Gravity field and steady-state Ocean Circulation Explorer - Agence Spatiale Européenne
[2]Centre National d'Etudes Spatiales - Toulouse, France

Abstract

In this thesis, we present our research in high performance scientific computing for linear least squares. More precisely we are concerned with developing efficient parallel software that can solve very large dense linear least squares problems and with providing numerical tools that can assess the quality of the solution. This thesis is also a contribution to the GOCE[3] mission that strives for a very accurate model of the Earth's gravity field. This satellite is scheduled for launch in 2007 and in this respect, our work represents a step in the definition of algorithms for the project.

We present an overview of the numerical strategies that can be used for updating the solution with new observations coming from GOCE mesurements. Then we describe a parallel distributed solver that we implemented in order to be used in the CNES[4] software package for orbit determination and gravity field computation. This solver compares well in terms of performance with the standard parallel libraries ScaLAPACK and PLA-PACK on the operational platforms used in the space industry while saving about half the memory, thanks to taking into account the symmetry of the problem.

In order to improve the scalability and the portability of our solver, we define a packed distributed format that is based on ScaLAPACK kernel routines. This approach is a significant improvement since there is no packed distributed format available for symmetric or triangular matrices in the existing dense parallel libraries. Examples are given for the Cholesky factorization and for the updating of a QR factorization. This format can easily be extended to other linear algebra calculations.

This thesis also contains new results in the area of sensitivity analysis for linear least squares resulting from parameter estimation problems. Specifically we provide a closed formula, bounds of correct order of magnitude and also statistical estimates that enable us to evaluate the condition number of linear functionals of least squares solution. The choice between the different expressions will depend on the problem size and on the desired level of accuracy.

Keywords: High performance computing, linear least squares, parallel distributed algorithms, packed storage format, Cholesky factorization, QR factorization and updating, normwise condition number, statistical condition estimate, parameter estimation.

---

[3]Gravity field and steady-state Ocean Circulation Explorer - European Space Agency
[4]Centre National d'Etudes Spatiales - Toulouse, France

Acknowledgements

First of all, I would like to thank Luc Giraud and Serge Gratton for being advisors of my thesis. I really appreciated their competence and availability.

I would also like to thank my team leader at CERFACS Iain Duff for his guidance and Georges Balmino for his precious help with the geodesy application.

I wish to thank the two referees of my thesis Jack Dongarra and Nick Higham for their interest in my work and for valuable comments on the manuscript.

Thank you also to Joseph Noailles who gave me the motivation for returning to science after working in the software industry.

Finally, I want to express my gratitude to all the persons with whom I had fruitful discussion and who, at a moment or another, enabled me to progress in my work: Mario Arioli, Isabelle d'Ast, M'Barek Fares, Julien Langou, Daniel Loghin, Jean-Charles Marty, Masha Sosonkina, Robert van de Geijn, Christof Vömel.

Contents

Introduction

This thesis was financed by CERFACS and the Centre National d'Etudes Spatiales (French Space Agency) in the framework of the GOCE[5] mission from the European Space Agency. After the satellite missions CHAMP [6] and GRACE[7], the GOCE satellite is scheduled for launch in 2007 and will provide a model of the Earth's gravity field and of the geoid with an unprecedented accuracy. It will have applications in many scientific areas such as solid-Earth physics, geodesy, oceanography, glaciology and climate change. GOCE will provide about $90,000$ model parameters of the Earth's gravity field. These parameters are estimated via an incremental linear least squares problem that involves a huge quantity of data (several millions of observations).

Our work in this thesis consists in proposing efficient and accurate methods that can tackle these very large least squares problems. To comply with the requirements of the GOCE mission, we had to deal with two bottlenecks that are the memory storage and the computational time.

For the sake of obtaining high performance and thanks to the increasing possibilities of parallel computers, we studied in-core solvers (where data are kept in the core memory of the computer) and we did not investigate the out-of-core approach (where data are stored on disk) that was also used in the GRACE mission. The parallel distributed solver that we developed provides good Gflops performance and compares competitively with standard parallel libraries on target platform for the GOCE mission while saving about half the memory. After a slight modification, the same code was also successfully used to solve the linear systems arising from boundary integral equations in electromagnetism.

We also define in this thesis a storage format that enables us to store compactly symmetric or triangular matrices in a parallel distributed environment. This format can be generalized to many dense linear algebra calculations.

The incremental QR factorization described in this thesis is appropriate for the gravity field computations. The good computational time obtained combined with the well-known better accuracy of orthogonal transformations are very encouraging.

Also because the accuracy is of major concern for the GOCE mission, we propose in this thesis several methods to estimate the condition number of linear combinations of the linear least squares solution components.

Even if the physical data for simulating GOCE computations were not available when this thesis was written, all tests have been made on similarly dimensioned problems and we assessed the accuracy of the solution with physical data coming from GRACE observations.

---

[5] Gravity field and steady-state Ocean Circulation Explorer, ESA
[6] CHAllenging Minisatellite Payload for Geophysical Research an Application, GFZ, launched July 2000
[7] Gravity Recovery and Climate Experiment, NASA, launched March 2002

This manuscript is structured as follows.

Part I is divided into two chapters. In Chapter I we present the objectives of the GOCE mission, the corresponding physical problem and the GINS [11] software currently used at the French Space Agency for computing satellite orbits and gravity field parameters inter alia. We also describe the direct methods for solving incremental least squares problems and the existing tools that can be used for their parallel implementation.
In Chapter II we describe the parallel distributed solver that we developed to compute the solution of a large dense linear least squares problem by the normal equations approach including performance analysis. Finally, we give in this chapter the gravity field results that we obtained after integrating our solver in the GINS software.

In Part II, we define a distributed storage format that exploits the symmetry or the triangular structure of a matrix when this matrix is kept in-core. This work has been motivated by the fact that, contrary to LAPACK [5], there is no routine that handles packed matrices in the standard parallel library ScaLAPACK [23]. The main characteristic of our approach is the use of ScaLAPACK computational kernels that enables good load balance even for high processor counts. We also present performance results for the Cholesky factorization and for the updating of the R factor in the QR factorization.

In Part III, we consider the linear least squares problem $\min_{y \in \mathbb{R}^n} \|Ay - b\|_2$ where $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$ is a matrix of full column rank n and we denote its solution by x. We assume that both A and b can be perturbed and that these perturbations are measured using the Frobenius or the spectral norm for A and the Euclidean norm for b. We are concerned with the condition number of a linear function of x ($L^T x$ where $L \in \mathbb{R}^{n \times k}$, $k \leq n$) for which we provide a sharp estimate that lies within a factor $\sqrt{3}$ of the true condition number. If the triangular R factor of A from $A^T A = R^T R$ is available, this estimate can be computed in $2kn^2$ flops. We also propose a statistical method that estimates the partial condition number by using the exact condition numbers in random orthogonal directions. If R is available, this statistical approach enables us to obtain a condition estimate at a lower computational cost. In the case of the Frobenius norm, we derive a closed formula for the partial condition number that is based on the singular values and the right singular vectors of the matrix A.

In the Appendix, we present an extension of the solver described in Chapter 2 to solve the large linear complex symmetric non Hermitian systems encountered in electromagnetism.

Finally, we give some conclusions and possible tracks that deserve further research.

I

Chapter 1

Numerical strategies for solving linear least squares problems arising in gravity field computations

1.1 Physical problem

1.1.1 Objectives of the GOCE mission

The GOCE [9, 10, 92] mission strives for a very accurate model of the Earth's gravity field and of the geoid, the geoid being defined as the equipotential surface of the Earth's gravity field which best fits, in a least squares sense, the global mean sea level averaged over a given time period.

According to the Newton's law of gravitation, the gravitational field due to an elementary mass m at the point P located at distance r from m is expressed by $\frac{Gm}{r^2}$ and it derives from the potential $\frac{Gm}{r}$ (Figure 1.1) where G is the universal gravitational constant. When we sum the gravitational field over the whole planet and when we take into account the Earth's rotation, we obtain what is referred to as the gravity field (expressed in m/s^2). As shown in Figure 1.2, the gravity field has two components:

- the gravitational force whose variations are due to the distance OP and to the planet's structure,
- the centrifugal force due to the rotation of the Earth that varies with the latitude (this force is very exaggerated in Figure 1.2).

The distance between the surface of the Earth and the centre is not identical everywhere, since the geometrical shape of the Earth can be approximated by an ellipsoid. Furthermore, the composition of the layers of the Earth's crust is not homogeneous and the depth of the layers varies from place to place.

The result is that the gravitational acceleration g and consequently the gravity field is far from being spatially constant and is not equal to the value 9.8 that is often considered in academic examples. Indeed g varies from $\sim 9.78 m/s^2$ at the equator to $\sim 9.83 m/s^2$
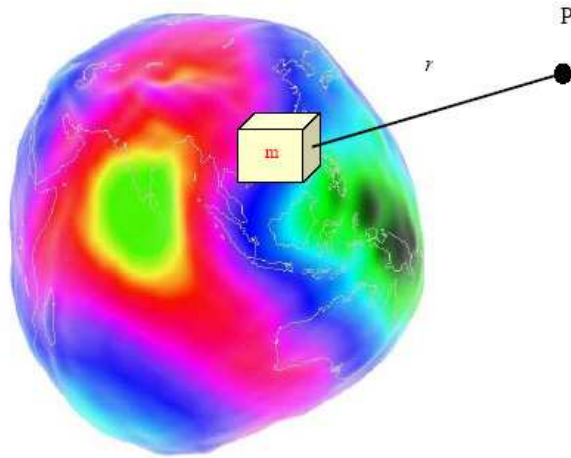
Figure 1.1: Earth's gravitational field.
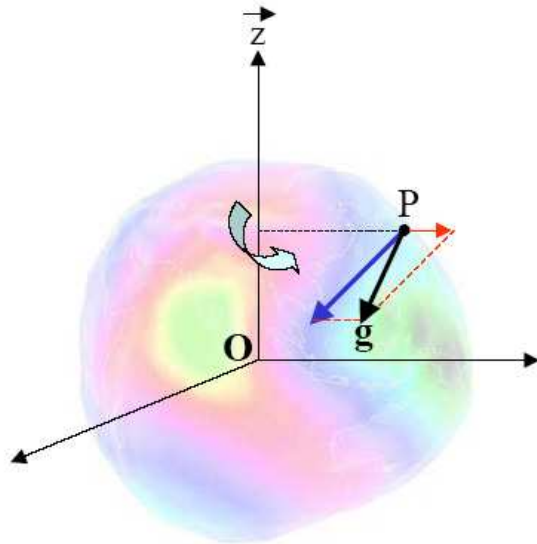
Figure 1.2: Gravity field components.

at the poles. A precise model of the gravity field will provide an advanced knowledge of the physics and dynamics of the Earth's interior.

The geoid is depicted in Figure 1.3 and corresponds to a particular surface of equal gravitational potential of a hypothetical ocean at rest. Note that the gravity field is not constant on the geoid but only its potential is constant.

The geoid is used notably
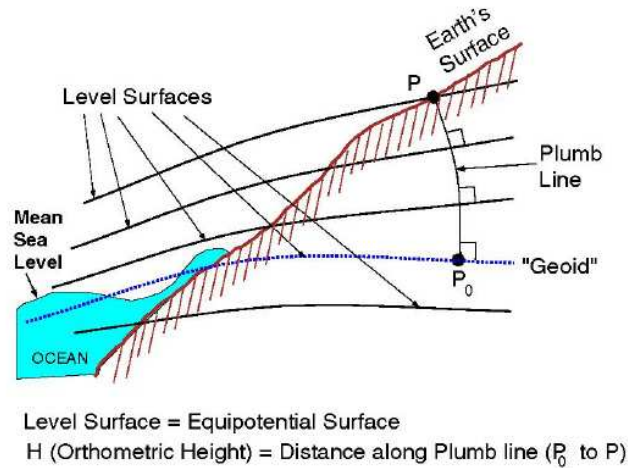
Figure 1.3: Geoid representation.

- to define physical altitudes,
- to define and forecast the water circulation that enables us to study ocean circulation, ice motion, sea-level change.

Due to the wind excitation, to the difference of density, and to the gravity waves resulting from the ocean tides, the geoid differs from the mean surface of the sea. The distance between these two surfaces is measured in order to obtain information on oceanic circulation.

In terms of accuracy, the GOCE mission objectives are

- to determine the gravity-field anomalies with an accuracy of 1 mGal (where $1 mGal = 10^{-5} m/s^2$),
- to determine the geoid with an accuracy of 1-2 cm,
- to achieve the above at a spatial resolution better than 100 km.

The model of the gravitational potential will be represented by about $90,000$ spherical harmonic coefficients up to a degree of 300.

The gravity gradients are measured by satellite gravity gradiometry, combined with satellite-to-satellite tracking using GPS (SST-hl) (Figure 1.4). The estimation of the Earth's gravity field using GOCE observations is a numerical and computational challenge. The numerical difficulty comes from the observation noise inherent in instruments used for measurement (gradiometers). Numerical instability may also result from missing observations at the poles, due to the non-polar orbit of the satellite.
The computational task is quite challenging because of the huge quantity of daily accumulated data (about $90,000$ parameters and several million observations) and because of the coupling of the parameters resulting in completely dense matrices.
Following [12], the Earth's gravitational potential V is expressed in spherical coordinates
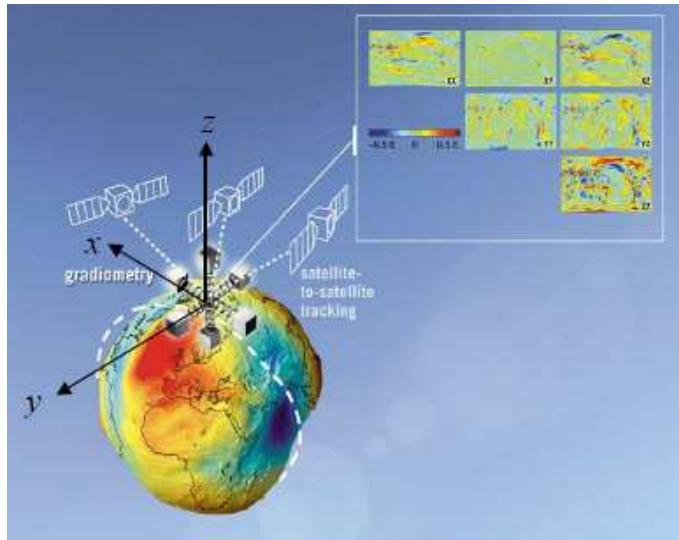
Figure 1.4: Gradiometry (measure of gravity gradients).

(r, θ, λ) by:

$$V(r, \theta, \lambda) = \frac{GM}{R} \sum_{l=0}^{l_{max}} \left(\frac{R}{r} \right)^{l+1} \sum_{m=0}^{l} \overline{P}_{lm}(\cos \theta) \left[\overline{C}_{lm} \cos m\lambda + \overline{S}_{lm} \sin m\lambda \right] \qquad (1.1)$$

where G is the gravitational constant, M is the Earth's mass, R is the Earth's reference radius, the $\overline{P}_{lm}$ represent the fully normalized Legendre functions of degree l and order m and $\overline{C}_{lm}, \overline{S}_{lm}$ are the corresponding normalized harmonic coefficients. In the above expression, we have $|m| \leq l \leq l_{max}$ with $l_{max} \simeq 300$ (about $90,000$ unknowns). For the previous missions CHAMP [86] and GRACE [71], we had respectively $l_{max} \simeq 120$ (about $15,000$ unknowns) and $l_{max} \simeq 150$ (about $23,000$ unknowns). We point out that the number of unknown parameters is expressed by

$$n = (l_{max} + 1)^2.$$

We have to compute the harmonic coefficients $\overline{C}_{lm}$ and $\overline{S}_{lm}$ as accurately as possible.

1.1.2 Application of orbit determination to gravity field computation

The principles of orbit determination can be applied to compute gravity field parameters as described below.

An orbit of the satellite is computed using orbit determination software (e.g GINS [11] for CNES). Measurements which are a function of a satellite position and/or velocity are taken into account. These observations are obtained via ground stations (Laser, Doppler) or other satellites (GPS) [26]. Then we aim to minimize the difference between the measurements and the corresponding quantities evaluated from the computed orbit by adjusting given parameters (here the gravity field coefficients). Figure 1.5 shows an example
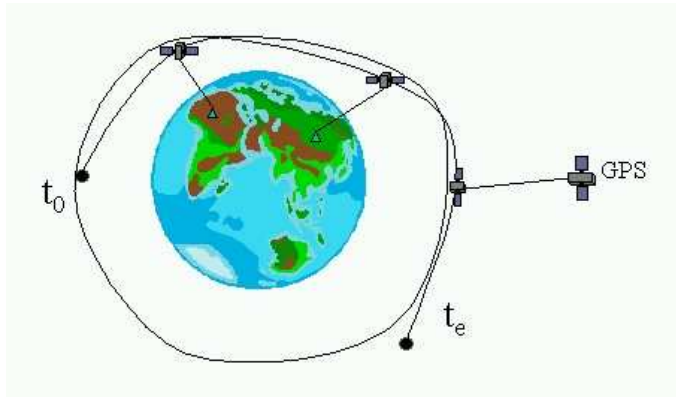
Figure 1.5: Example of computed and measured orbits.

of reference orbit and some positions of the satellite measured using ground stations and GPS.

The general orbit determination problem is a nonlinear estimation problem [13] that is expressed by two nonlinear formulations.

The first one is the differential equation related to the dynamics of the system. In the GOCE application the dynamics can be modelled for $t \geq t_0$ by:

$$\ddot{r} = f(r, \dot{r}, \gamma, t), \qquad r(t_0) = r_0, \qquad \dot{r}(t_0) = r_0', \tag{1.2}$$

where the vector $\gamma = \begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_n \end{pmatrix}$ contains the gravity field coefficients $\overline{C}_{lm}$ and $\overline{S}_{lm}$ mentioned in Equation (1.1). The vectors $r(t_0)$ and $\dot{r}(t_0)$ represent respectively the position and the velocity at time t_0.

The second part in the formulation of an orbit determination problem is related to the measurements involving the observed quantities (e.g the gravity gradients):

$$Q_j = h(r, \dot{r}, \gamma, t_j) + \varepsilon_j, \tag{1.3}$$

where the Q_j, $j = 1 \cdots, m$ are the observations at time t_j and the errors in the observations are represented by ε_j.

Let $\widetilde{Q_j}$ be a sample of the random variable Q_j and $h_j(\gamma) = h(r, \dot{r}, \gamma, t_j)$. Then we estimate γ in the least squares sense by solving

$$\min_{\gamma} \sum_{j=1}^{m} \|\widetilde{Q_j} - h_j(\gamma)\|_{W_j}^2 \tag{1.4}$$

in which the norm $\|x\|_{W_j}^2$ is defined as being $x^T W_j x$. The matrix W_j represents the weights of measurements and takes into account the orbit errors, the measurement errors, the errors coming from ground stations, and the errors due to instruments at reception.

The nonlinear least squares problem (1.4) is solved via an algorithm that is referred to as the Gauss-Newton algorithm in [64] as described below:

9

0.   choose $\gamma^{(0)}$
1. $\overline{\Delta\gamma} = arg\min_{\Delta\gamma} \sum_{j=1}^{m} \|h_j(\gamma^{(k)}) - \widetilde{Q_j} + h'_j(\gamma^{(k)}).\Delta\gamma\|_{W_j}^2$
2. $\gamma^{(k)} \leftarrow \gamma^{(k)} + \overline{\Delta\gamma}$; $k \leftarrow k+1$
3.   goto 1

In the GINS computations, the Jacobian matrices h'_j will be considered as constant over the iterations (inexact Gauss-Newton) i.e $\forall k$, $h'_j(\gamma^{(k)}) = h'_j(\gamma^{(0)})$.

We now describe how these Jacobian matrices are approximated in the framework of the GOCE calculations. The GOCE observations Q_j measured by the gradiometers are the gravity gradients i.e the second order spatial derivatives of the gravitational potential V expressed in (1.1). If we consider a coordinate frame (X_1, X_2, X_3) related to the gradiometer (along track, across track, radial), then the gravity gradients are expressed by

$$V''_{kl} = \frac{\partial^2 V}{\partial X_k \partial X_l}.$$

The symmetric matrix $\nabla^2 U = (V''_{kl})$ is referred to as the gravity tensor. Note that the V''_{kl} are not measured with the same accuracy and that only the diagonal elements $V''_{11}, V''_{22}, V''_{33}$ are accurately measured and this influences the choice of the matrix W_j .
With our notations, the $\widetilde{Q_j}$ represent the measured values of the gravity gradient V''_{kl} and we have

$$h'_j(\gamma).\Delta\gamma = \sum_{i=1}^{n} \frac{\partial Q_j}{\partial\gamma_i}\Delta\gamma_i.$$

For the m observations $Q_1, \cdots, Q_m$ we get

$$h'(\gamma).\Delta\gamma = \begin{pmatrix} \frac{\partial Q_1}{\partial\gamma_1} & \cdots & \frac{\partial Q_1}{\partial\gamma_n} \\ \vdots & & \vdots \\ \frac{\partial Q_m}{\partial\gamma_1} & \cdots & \frac{\partial Q_m}{\partial\gamma_n} \end{pmatrix} \begin{pmatrix} \Delta\gamma_1 \\ \vdots \\ \Delta\gamma_n \end{pmatrix}.$$

The coefficients $\frac{\partial Q_j}{\partial\gamma_i}$ can be obtained via the equation of the dynamics (1.2) since we have

$$\frac{\partial Q_j}{\partial\gamma_i} = \frac{\partial Q_j}{\partial r} \cdot \frac{\partial r}{\partial\gamma_i} + \frac{\partial Q_j}{\partial\dot{r}} \cdot \frac{\partial\dot{r}}{\partial\gamma_i}. \tag{1.5}$$

The quantities $\frac{\partial Q_j}{\partial r}$ and $\frac{\partial Q_j}{\partial\dot{r}}$ are computed analytically and the quantities $\frac{\partial r}{\partial\gamma_i}$ and $\frac{\partial\dot{r}}{\partial\gamma_i}$ are computed numerically by deriving Equation (1.2) with respect to γ_i as follows:

$$\frac{\partial\ddot{r}}{\partial\gamma_i} = \frac{\partial F}{\partial r} \cdot \frac{\partial r}{\partial\gamma_i} + \frac{\partial F}{\partial\dot{r}} \cdot \frac{\partial\dot{r}}{\partial\gamma_i} + \frac{\partial F}{\partial\gamma_i},$$

where F is the gravity force.
We have here n differential equations where the derivatives of F are analytically computed and the unknowns are $\frac{\partial r}{\partial\gamma_i}$ and $\frac{\partial\dot{r}}{\partial\gamma_i}$. These equations are integrated for several time steps (the lower the orbit, the shorter the integration time) at which r , $\dot{r}$, $\frac{\partial r}{\partial\gamma_i}$ and $\frac{\partial\dot{r}}{\partial\gamma_i}$ are computed. For the GOCE orbit, this step size will be between 1 and 5 seconds.

10

After interpolation at the measurement times t_j (that are different from the times at which the positions are computed), we obtain the coefficients $\frac{\partial Q_j}{\partial \gamma_i}$ using Equation (1.5). This enables us to obtain the data matrix

$$A = \begin{pmatrix} \frac{\partial Q_1}{\partial \gamma_1} & \cdots & \frac{\partial Q_1}{\partial \gamma_n} \\ \vdots & & \vdots \\ \frac{\partial Q_m}{\partial \gamma_1} & \cdots & \frac{\partial Q_m}{\partial \gamma_n} \end{pmatrix}.$$

Using linear algebra notations we set $x = \Delta\gamma$ and $b = \begin{pmatrix} \widetilde{Q_1} - h_1(\gamma) \\ \vdots \\ \widetilde{Q_m} - h_m(\gamma) \end{pmatrix}$ and we have

to minimize $Ax - b$ in the least squares sense that can be written as $\min_{x \in \mathbb{R}^n} \|Ax - b\|_W$ where W is a block diagonal matrix containing the W_j. Note that, in practice, the vector γ contains some coefficients that do not depend on the dynamics of the satellite (e.g measurement biases).

In the rest of this thesis we assume that W is already integrated into the observations, which is the case for the GINS software.

1.2 Numerical methods

As shown in Section 1.1.2, the computation of the Earth's gravitational potential is a non-linear least squares problem that can be solved by successive linear least squares problems (LLSP) where we have to solve

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2 \tag{1.6}$$

where $b \in \mathbb{R}^m$ is the observation vector and $A \in \mathbb{R}^{m \times n}$ is the data matrix (Jacobian of a nonlinear least squares problem). In the GOCE application, the number of unknowns n is about $90,000$ and the number of observations m is about 10^6.

After accumulating a sufficient number of observations and/or by using regularization techniques, A is a full column rank matrix. In that case (1.6) has a unique solution [50, p. 237]. For GRACE, a gravity field up to degree 150 can be obtained after one month of measurement (but 10 days can be sufficient to get a solution up to degree 100). For GOCE, a solution can be computed after 6 months of measurement but recent simulations have shown that with 20 days of measures we get a gravity field up to degree 120 [11].

It is possible to solve (1.6) via iterative methods based on conjugate gradient or FFT techniques [82]. Fast multipole methods can also be used [47]. Provided that the conditions for convergence are satisfied, these iterative methods give fast computation times but still need improvement regarding the accuracy of the solution. Recently new promising methods based on spherical wavelets have been brought into the Earth's gravity field calculations [43]. Although orthogonal transformations have been recently introduced for GRACE computations (e.g out-of-core QR factorization [56]), the geodesists involved in gravity field computations traditionally use, as many statisticians [62] do, the normal equations method that solves the system

$$A^T Ax = A^T b.$$

Experiments performed in [82] showed that this method should provide a solution with better accuracy than some iterative methods mentioned above though this gain in accuracy could be far below the observation noise level. Another advantage of this method is that it enables us to obtain the covariance matrix $(A^T A)^{-1}$ which is sometimes needed by the physicists.

The downside of this method is that it demands high computing capabilities to assemble the normal equations and also large memory capacities to store the n-by-n symmetric matrix in the core memory of the computer. But the increasing possibilities of current parallel computers encourage us to implement this method for GOCE.

In the sequel of this paragraph we recall some major numerical results regarding the way of solving linear least squares problems that will be taken into account in GOCE computations.

1.2.1 Selected methods for linear least squares

Many methods are available for solving LLSP which are detailed in [21, 61]. Giving a survey of all these methods and of their numerical stability is out of the scope of this thesis. That is why, in the rest of this chapter, we focus on the two methods that are likely to be used by the physicists from the geodesy community especially for the GOCE project where the matrices are dense, large, and accurate solutions are wanted. These methods can be based either on the normal equations approach or on the QR factorization.

In the normal equations approach, we solve the linear system of equations

$$A^T A x = A^T b.$$

If the rank of A is n then the n-by-n matrix $A^T A$ is symmetric positive definite and can be decomposed using a Cholesky factorization as $A^T A = U^T U$, where U is an upper triangular matrix.

Then the normal equations become $U^T U x = A^T b$, and the unknown vector x can be computed by solving $U^T y = A^T b$ (forward substitution phase) followed by $U x = y$ (backward substitution phase).

The cost in arithmetic operations can be split as follows:

1. In the construction of the symmetric matrix $A^T A$, we only compute and store the upper triangular part of $A^T A$. Hence the cost in operations will be $\mathcal{O}(mn^2)$.
2. The Cholesky factorization algorithm involves $n^3/3$ operations.
3. In the final step we solve two triangular systems. Hence it involves $2n^2$ operations.

If we neglect the terms in $\mathcal{O}(n^2)$, then the cost in floating-point operations of the normal equations method is $n^2(m + n/3)$ [50, p. 238].

A more reliable way of solving linear least squares consists in using orthogonal transformations. The commonly used QR factorization can be performed by using orthogonal transformations called Householders reflections [50, p. 208]. The QR factorization of A is given by

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$$

where Q is an m-by-m orthogonal matrix and R is an n-by-n upper triangular matrix. Since A has full column rank, then R is nonsingular. If we denote $Q = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}$ where Q_1 and Q_2 correspond respectively to the n first columns and the $m - n$ remaining columns of Q, then we have $A = Q_1 R$.

From $\|Ax - b\|_2 = \|Q^T b - Q^T A x\|_2 = \left\| \begin{matrix} Q_1{}^T b - Rx \\ Q_2{}^T b \end{matrix} \right\|_2$ it follows that x can be computed by solving the triangular system $Rx = Q_1{}^T b$.

In the QR method, the matrix R may overwrite the upper triangular part of A while the Householder vectors are stored in the lower trapezoidal part of A.

Note that Q_1 is not required explicitly and we just need to apply Q_1^T to a vector. The computation of $Q_1{}^T b$ can be performed by applying the Householder transformation used to compute R to the vector b. This is achieved by appending b to A and factorizing $\begin{pmatrix} A & b \end{pmatrix}$.

This factorization overwrites b by $\tilde{b}$ and we solve $Rx = \tilde{b}$ using the first n elements of $\tilde{b}$.

If we neglect the cost of the triangular solve, then the computational cost of the least squares solution using a Householder QR factorization is $2n^2(m - n/3)$ [50, p. 225].

We have seen that the solution is computed by solving $Rx = Q_1{}^T b$. It is sometimes necessary to solve this system with several right-hand sides that are not available when the QR factorization is performed. In that case we need to store Q_1. Even if A is large and sparse, this may not be true for Q_1 and storing Q_1 can be expensive. If A can be saved, it can be useful to use the semi-normal equations (SNE) method where we solve the system

$$R^T R x = A^T b,$$

a straightforward reformulation of the normal equations. In the SNE method, x is computed without using Q_1. This makes it possible to treat additional right-hand sides that are not necessarily available when the QR factorization is performed. This corresponds to the situation that occurs for GOCE calculations since the matrix $R^T R$ will be fixed (cf Section 1.1.2) whereas the different right-hand sides will be available in sequence within the Gauss-Newton algorithm.

1.2.2 Normal equations vs QR factorization

When choosing between the normal equations method and QR factorization, we have to consider two criteria that are the computational cost and the numerical stability.

If we compare the floating-point operations involved in each method, we observe that when $m \gg n$ the normal equations approach requires about half the flop count of the Householder QR factorization (mn^2 vs $2mn^2$). This explains why the normal equations method is often favoured by users in geodesy.

In the following, we compare the normal equations method and the QR factorization in terms of accuracy.

The sensitivity of an LLSP to perturbations can be notably measured by bounding the difference between the exact solution and the solution of the perturbed problem. Let

$K(A) = \|A\|_2 \|A^\dagger\|_2 = \frac{\sigma_1(A)}{\sigma_{rank(A)}}$ be the condition number of A where $A^\dagger$ denotes the Moore-Penrose pseudo inverse of A and $\sigma_1(A) \geq \cdots \geq \sigma_{rank(A)} > 0$ are the nonzero singular values of A. The following theorem established by Wedin [101] is also given in [61, p. 382] and [21, p. 30].

Theorem 1. *Let $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$ and $A + \Delta A$ both be of full rank and let*

$$x = \arg\min_{x \in \mathbb{R}^n} \|Ax - b\|_2, \qquad r = b - Ax,$$
$$\tilde{x} = \arg\min_{x \in \mathbb{R}^n} \|(A + \Delta A)x - (b + \Delta b)\|_2, \quad \tilde{r} = b + \Delta b - (A + \Delta A)\tilde{x},$$
$$\|\Delta A\|_2 \leq \epsilon \|A\|_2, \qquad \|\Delta b\|_2 \leq \epsilon \|b\|_2.$$

Then, if $K(A)\epsilon < 1$, we have

$$\frac{\|x - \tilde{x}\|_2}{\|x\|_2} \leq \frac{K(A)\epsilon}{1 - K(A)\epsilon} \left(2 + (K(A) + 1)\frac{\|r\|_2}{\|A\|_2 \|x\|_2}\right),$$

and

$$\frac{\|r - \tilde{r}\|_2}{\|b\|_2} \leq (1 + 2K(A))\epsilon.$$

Theorem 1 shows that the sensitivity of an LLSP is measured by $K(A)$ when the residual $\|b - Ax\|_2$ is small (or zero) compared with the norms of A and x; otherwise, it is measured by $K(A)^2$.

In the following, we assume that the computations are performed in floating-point arithmetic. We consider the standard model defined by $fl(x \ op \ y) = (x \ op \ y)(1 + \epsilon)$ where fl denote the computed value of an expression, $op = +, -, *, /$, ϵ is such that $|\epsilon| \leq u$ with $u = $ unit roundoff defined in [50, 61, 89], and x and y are already in finite precision. We shall quote available results explaining the behaviour of the methods described above in floating-point arithmetic.

From [74, p. 90] or [50, p. 240], we deduce the following theorem that summarizes the advantage of the Householder QR factorization algorithm:

Theorem 2. *Let $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$ have full rank and suppose the least squares problem $\min_{x \in \mathbb{R}^n} \|Ax - b\|_2$ is solved using the Householder QR factorization method. The computed solution $\tilde{x}$ is the exact solution to*

$$\min_{x \in \mathbb{R}^n} \|(A + \Delta A)x - (b + \Delta b)\|_2,$$

with

$$\|\Delta A\|_F \leq c_{m,n}u \|A\|_F + \mathcal{O}(u^2)$$

and

$$\|\Delta b\|_2 \leq c_{m,n}u \|b\|_2 + \mathcal{O}(u^2),$$

where $c_{m,n}$ is a $\mathcal{O}(mn)$ constant.

14

This desirable property, where the computed solution is the exact solution of a slightly perturbed problem with a relative perturbation on data bounded by a not-too-large constant multiplied by u, is called normwise backward stability. It makes the Householder QR factorization method a reliable way for solving an LLSP. This implies in particular that the relative forward error $\delta = \frac{\|x - \tilde{x}\|}{\|x\|}$ satisfies the inequality mentioned in Theorem 1 when $\tilde{x}$ is computed via a QR factorization.

The numerical stability properties of the normal equations method are less satisfactory for the two following reasons:

A first reason is related to the assembly of $A^T A$ and/or the formation of $A^T b$ that may lead to a loss of accuracy and in some cases the computed matrix $A^T A$ can be singular or indefinite (see the Läuchli [73] matrix examples in [21, p. 44] or [61, p. 386]).

A second downside of the normal equations is that it produces a forward error bound that may be larger than the one obtained for a backward stable method like QR. To show this, we first state the following theorem related to the backward stability of the Cholesky factorization. This result was established by Wilkinson [105] and mentioned in [21, p. 49].

Theorem 3. *Let $C \in \mathbb{R}^{n \times n}$ be a symmetric positive definite matrix. Provided that*

$$2n^{\frac{3}{2}}uK(C) < 0.1$$

the Cholesky factor of C can be computed without breakdown, and the computed $\widetilde{U}$ will satisfy

$$\widetilde{U}^T \widetilde{U} = C + E \ , \ \|E\|_2 < 2.5n^{\frac{3}{2}}u \, \|U\|_2^2 \, .$$

Hence, $\widetilde{U}$ is the exact Cholesky factor of a matrix close to C.

We now consider the ideal situation where there is no rounding error coming from the formation of $A^T A$ and $A^T b$ or from the triangular solves. Then the LLSP solution computed via the normal equations approach satisfies the equation

$$(A^T A + E)\tilde{x} = A^T b$$

with $\|E\|_2 < 2.5n^{\frac{3}{2}}u \, \|U\|_2^2$. Then, from [21, p. 49], the resulting relative forward error δ is such that

$$\delta \leq 2.5n^{\frac{3}{2}}uK(A)^2.$$

We have seen that, when the residual is small, the forward error resulting from a QR factorization is bounded by an expression that is proportional to $K(A)$. Since the normal equations method produces a forward error that can be proportional to $K(A)^2$, it is potentially less accurate than the QR factorization.

The error analysis of the SNE method has been studied in [79] for the minimum norm problem and in [20] for the LLSP. It is shown in [20] that, similarly to the normal equations method, the forward error bound involves a factor $K(A)^2$, even if we use a R-factor that is of better quality than the Cholesky factor because it has been computed

15

via a backward stable algorithm. As explained in [21, p. 126 and p. 250], the accuracy of the SNE method can be improved by using the corrected semi-normal equations method (CSNE) that consists in adding one step of fixed precision iterative refinement to the SNE as follows:

1. Let $\tilde{x}$ solve $R^T R x = A^T b$
2. compute $\tilde{r} = b - A\tilde{x}$
3. solve $R^T R w = A^T \tilde{r}$
4. corrected solution $y = \tilde{x} + w$

The forward error bound for the CSNE method is such that [61, p. 392]

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq c_{m,n} \left(K(A)u \cdot K(A)^2 u \left(1 + \frac{\|b\|_2}{\|A\|_2 \|x\|_2} \right) + \frac{K(A)^2 u \|r\|_2}{\|A\|_2 \|x\|_2} \right), \qquad (1.7)$$

where $c_{m,n}$ is a small constant. If $K(A)^2 u \leq 1$, then the error bound given in (1.7) is similar to that of Theorem 1 for a backward stable method (and even smaller when r is small). In that case, the CSNE method is more satisfactory than the SNE method but this is not true for all A. We also mention the approach that consists in considering the augmented system [61, p. 383]

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

and applying iterative refinement to it [17, 18]. This method can provide an accurate solution especially for sparse LLSP [7, 19] but in our physical application where the matrix is considered as dense, the resulting linear system of $m + n$ equations would be very expensive to store and to solve (we recall that the order of m is 10^6).

When the previous missions CHAMP and GRACE were launched, the normal equations method was chosen because of its cheaper computational cost and because of the capabilities of the current computers. Will it still be the case for the GOCE measurements?
We have seen above that the condition number of A may influence the choice of the method for solving our least squares problem. This is why we implemented a condition number estimate for $A^T A = R^T R$ (note that $K(A^T A) = K(A)^2$) in Chapter 3. Furthermore, since all solution components are not of interest for the physicists, results related to the condition number of a linear function of the least squares solution are given in Chapter 4. Since the QR factorization involves twice the floating-point operations needed for the normal equations method (for $m \gg n$, as is the case for GOCE), we provide in Chapter 3 an efficient algorithm that makes the QR factorization affordable for GOCE computations.

1.2.3   Incremental approach for linear least squares

As shown in Figure 1.6, the GOCE observations are obtained during a six months measurement phase. In this figure, the thick black line corresponds to the altitude of the satellite on the orbit. The blue regions correspond to eclipse periods and the right axis show their

16

duration in minutes per revolution. Due to the fact that electricity comes mainly from the solar panels, measurement is not possible when the eclipse duration per revolution is greater than 15 minutes. This explains the measurement interruption phase between $T0+9$ and $T0+14$. The measurements are accumulated on a daily basis and, if the matrix $A^T A$ is nonsingular, a solution can be computed. Regularization techniques are possible that make the solution computation more stable. The commonly used Kaula [25, 68] regularization consists of adding to $A^T A$ a diagonal matrix $D = diag(0, \cdots, 0, \alpha, \cdots, \alpha)$ where α is a constant that is proportional to $\frac{10^{-5}}{l_{max}^2}$ and l_{max} is the quantity defined in Section 1.1.1. The nonzero terms in D correspond to the variables that need to be regularized. In that case, in the normal equations approach, we have to invert the matrix $A^T A + D$. This regularization technique is a special case of the Tikhonov [60, 95] regularization method in the general form

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2 + \lambda^2 \|Lx\|_2^2 \text{ where } L \in \mathbb{R}^{n \times n} \text{ (here } \lambda^2 L^T L = \alpha D).$$
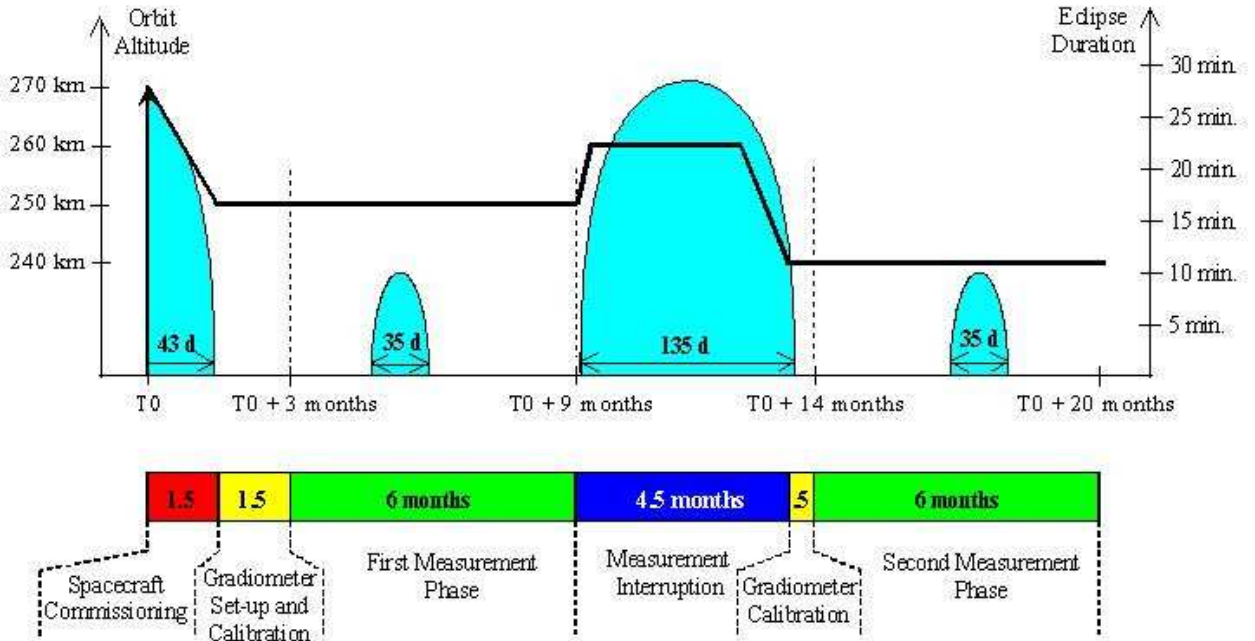

Figure 1.6: GOCE mission profile (early 2007 - end 2008).

We now describe an algorithm that aims to solve the nonlinear least squares problem resulting from Equations (1.2) and (1.3). At each step of this algorithm, we solve a linear least squares problem.

Let $\mathcal{A}_N$ and $\mathcal{B}_N$ be respectively the cumulated parameter matrix and observation vector up to date N.

Let A_N and b_N be respectively the data matrix and observation vector that has been collected at date N.

Then we have:

$$\mathcal{A}_N = \begin{pmatrix} A_1 \\ \vdots \\ A_N \end{pmatrix} \text{ and } \mathcal{B}_N = \begin{pmatrix} b_1 \\ \vdots \\ b_N \end{pmatrix}$$

The least squares problem to solve at date N can be stated as: $\min_{x \in \mathbb{R}^n} \|\mathcal{A}_N x - \mathcal{B}_N\|_2$.

We assume that we have solved the least squares problem at date N.

Then, at date $N+1$, we have:

$$\mathcal{A}_{N+1} = \begin{pmatrix} \mathcal{A}_N \\ A_{N+1} \end{pmatrix} \text{ and } \mathcal{B}_{N+1} = \begin{pmatrix} \mathcal{B}_N \\ b_{N+1} \end{pmatrix}$$

If we use the normal equations method, then the matrix of normal equations is expressed by

$$\mathcal{A}_{N+1}^T \mathcal{A}_{N+1} = \mathcal{A}_N^T \mathcal{A}_N + A_{N+1}^T A_{N+1}$$

and the right hand-side is given by

$$\mathcal{A}_{N+1}^T \mathcal{B}_{N+1} = \mathcal{A}_N^T \mathcal{B}_N + A_{N+1}^T b_{N+1}$$

where $\mathcal{A}_N^T \mathcal{A}_N$ and $\mathcal{A}_N^T \mathcal{B}_N$ have been previously computed at date N. Then, in an incremental process, only the normal equations matrix (and the right hand side $\mathcal{A}_N^T \mathcal{B}_N$) has to be stored for further computations. The updating of the normal equations at date $N+1$ is performed by computing $A_{N+1}^T A_{N+1}$ and $A_{N+1}^T b_{N+1}$ and then adding them respectively to the stored data $\mathcal{A}_N^T \mathcal{A}_N$ and $\mathcal{A}_N^T \mathcal{B}_N$.

Let consider the case where we use a QR approach that utilizes Householder transformations. If we denote by R_N the R-factor obtained at date N, then R_{N+1} corresponds to the R-factor in the QR factorization of $\mathcal{A}_{N+1}$. But we observe that the QR factorization of $\mathcal{A}_{N+1} = \begin{pmatrix} \mathcal{A}_N \\ A_{N+1} \end{pmatrix}$ produces the same upper triangular factor as does the factorization of $\begin{pmatrix} R_N \\ A_{N+1} \end{pmatrix}$ i.e R_{N+1}.

Furthermore, the storage of the Householder vectors can be avoided by appending the observation vector b_N to the matrix to be factorized and overwriting this vector with the $(n+1)$-th column of the so-obtained triangular factor.

The result is that the updating of the R-factor at date $N+1$ is done by performing the QR factorization of $\begin{pmatrix} R_N & \tilde{\mathcal{B}}_N \\ A_{N+1} & b_{N+1} \end{pmatrix}$ where $\tilde{\mathcal{B}}_N$ contains the updated values of $\mathcal{B}_K$ ($K \leq N$) resulting from the N previous QR factorizations.

This enables us to obtain the upper triangular matrix $\begin{pmatrix} R_{N+1} & \tilde{\mathcal{B}}_{N+1} \end{pmatrix}$ and the solution x_{N+1} is computed by solving $R_{N+1} x_{N+1} = \mathcal{Z}_{N+1}$ where $\mathcal{Z}_{N+1}$ contains the first n elements of $\tilde{\mathcal{B}}_{N+1}$. The algorithm that performs the QR factorization of $\begin{pmatrix} R_N & \tilde{\mathcal{B}}_N \\ A_{N+1} & b_{N+1} \end{pmatrix}$

will be described in Chapter 3.

Similar algorithms for incremental Cholesky and QR factorizations are described in [21, p. 224] in the framework of block angular least squares problems. Note that alternative algorithms referred to as covariance algorithms update $(\mathcal{A}^T\mathcal{A})^{-1}$ or a factor of it [65].

1.2.4 Parallel libraries for dense computations

This thesis strives to study and implement methods that can tackle the least squares problem arising from the GOCE observations and that also satisfy the operational constraints of the project in terms of accuracy and computing time.

If we consider the fact that the GOCE calculations involve about $90,000$ variables, then storing the whole matrix $A^T A$ (or R) would require about 65 Gbytes of memory. Furthermore, supposing that a matrix-matrix product can be performed at the rate of 4.2 Gflops, then the assembly time for the normal equations would be about 22 days. This shows that, if we want to keep $A^T A$ (or R) in memory and if computations must be achieved within a day, then we need to exploit parallelism.

The existing libraries for dense in-core calculations are ScaLAPACK [23] and PLAPACK [98].

The widely used parallel library ScaLAPACK [23] (Scalable Linear Algebra PACKage) is a parallel extension of the sequential library LAPACK [5]. The ScaLAPACK project is a collaborative effort involving several institutions: Oak Ridge National Laboratory, Rice University, University of California at Berkeley, University of California at Los Angeles, University of Illinois at Urbana-Champaign, University of Tennessee at Knoxville. It has been designed to perform dense linear algebra calculations on distributed-memory message-passing computers. The first release (version 1.0) was in February 1995 and the latest one in August 2001 (version 1.7).

ScaLAPACK includes routines for

- solving linear systems and linear least squares problems,
- solving eigenvalue and singular value problems,
- factorizing matrices and estimating condition numbers.

Similarly to LAPACK, the ScaLAPACK routines minimize data movement between the different levels of the memory hierarchy (registers, caches, processor memory, other processors memory) by using blocked algorithms. Moreover they aim to obtain a good partition of work (also called load balance) between processors by partitioning and distributing matrices and vectors among the processors. This good load balance is obtained by distributing data using a two-dimensional block-cyclic distribution [23, p. 58].

The software components of ScaLAPACK are the BLAS [1, 39] and the BLACS [41] libraries that perform respectively the common linear algebra operations and the message-passing communication between processors. These two libraries perform the low-level tasks of ScaLAPACK and are not referenced explicitly when developing ScaLAPACK-based implementations. ScaLAPACK is fully portable provided that the BLAS and BLACS libraries are supported by the parallel platform being used.

The PBLAS (Parallel BLAS) [32] library provides routines implementing in parallel the basic linear algebra calculations performed in sequential by the BLAS library. The PBLAS interface has been designed to look as similar as possible to the BLAS. Since the PBLAS

routines handle distributed data, they are also commonly used in ScaLAPACK implementations.

The PLAPACK library was developed in 1997 at the University of Texas. It implements linear algebra functionalities similar to those proposed in ScaLAPACK. PLAPACK provides an object-based approach to programming and a simplified way for distributing matrices [98, p. 3]. It also introduces the possibility of using algorithmic blocking for matrix-matrix multiply that corresponds to a level of blocking above the one used for distributing data among processors.

The two parallel libraries provide different computational routines that may be compared on some particular examples but this study is out of the scope of this thesis.

To handle larger problems, out-of-core implementations of the QR, Cholesky or LU factorization routines were created. These implementations were described in [38] for ScaLA-PACK and [57, 58] for PLAPACK.

Chapter 2

An operational parallel solver for GOCE gravity field calculations

2.1 Motivations

The solution of large dense linear systems appears in many engineering and scientific applications of computational sciences. This is for instance the case in geodesy where the calculation of the gravity field requires the solution of large linear least-squares problems that are often solved using the normal equations approach. Such dense linear systems also arise in electromagnetics applications when boundary elements are used. In these fields, the recent developments of the Fast Multipole Method (FMM) enable us to perform dense matrix-vector products efficiently. This opens the possibility of using iterative Krylov solvers for solving these large systems. In particular, in electromagnetic computations, the FMM are established methods providing reliable solution of linear systems up to a few tens of million unknowns on parallel computers [93] for industrial calculations. In gravity field computations, these techniques are studied in research codes, but further investigations, mainly related to the accuracy of the solution, are needed to bring them into use in operational codes. For these latter calculations, dense factorizations are still the methods of choice and out-of-core implementations might be an alternative when the matrix does not fit into the memory [38, 57]. With the advent of distributed memory parallel platforms, in-core codes are affordable and the parallel libraries ScaLAPACK [23] and PLAPACK [98] are often selected to perform this linear algebra calculation. In particular these libraries implement the Cholesky factorization for symmetric positive definite linear systems but they do not exploit the symmetry for the storage, and the complete matrix should be allocated while the factorization only accesses half of it. This "waste" of memory cannot be afforded for operational industrial calculations since the memory of moderate size parallel computers is often the main bottleneck. This is the main reason why we develop a parallel distributed dense Cholesky factorization based on MPI [44] that exploits the symmetry and stores only half of the matrix.

We choose a factorization algorithm and a data distribution that are different from the libraries ScaLAPACK and PLAPACK. The resulting implementation will compare competitively with these libraries for up to 32 processors. For higher processor counts, these choices should be reconsidered.

Our algorithm implements:

- a j-variant [6, 78] block Cholesky factorization algorithm,
- a block-cyclic column data distribution [23, p. 60] where only the upper part of the matrix is stored, this storage being implemented in a row-wise format,
- message passing performed by non-blocking asynchronous MPI routines,
- level 3 BLAS [39] routines in order to account for the memory hierarchy on each processor.

The target application is here the gravity field computation in the framework of the GOCE mission. As pointed out in Section 1.2, the estimation process of the gravity field parameters is incremental and is currently performed at CNES (Centre National d'Etudes Spatiales) by the normal equations method. Note that the physicists do not compute a solution a each step of the incremental process. We also point out that there is no updating of the Cholesky factor as described in [16, p. 44] and that, when a solution is computed, the users perform a complete Cholesky factorization of the normal equations matrix. This does not penalize the global performance since, as we will see below, the computational cost of the Cholesky factorization is negligible compared with that of the normal equations assembly. In this chapter, we focus on one step of the incremental process described in Section 1.2.3 that consists of solving the least squares problem $\min_{x \in \mathbb{R}^n} \|Ax - b\|_2$ where $b \in \mathbb{R}^m$ is the observation vector and $A \in \mathbb{R}^{m \times n}$ is a full column rank matrix. In order to be integrated into the GINS existing software we first implemented a parallel distributed solver based on the normal equations approach that consists in forming and then solving the normal equations:

$$A^T A x = A^T b.$$

In this application, m is about 10^6 and n is slightly less than 10^5. The three phases are:

1. the assembly of $A^T A$ (cost $\mathcal{O}(mn^2)$),
2. the Cholesky factorization of $A^T A$ (cost $n^3/3$),
3. two triangular system solves (cost $2n^2$).

We point out that steps 1 and 2 are the most time-consuming tasks and need an efficient parallel distributed implementation. The construction and storage of $A^T A$ has been implemented using MPI and the obtained performance is close to the peak performance of the computers on a matrix-matrix multiply [84, Section 6.3]. The triangular solve, which is less critical in terms of arithmetic operations, has been also implemented using MPI.

The numerical simulations we are interested in are performed in a daily production mode at CNES for the geodesy application. For this project, the target parallel platforms are moderately parallel computers with up to 32 processors and less than 2Gbytes memory per processor.

The parallel solver described in this chapter is now part of the GINS software at CNES. GINS computes an accurate orbit around a body in the solar sytem and determines some geophysical parameters such as:

- the gravity field coefficients,

- the coefficients of the oceanical tide model,
- the mean ocean surface,
- some coefficients of the atmosphere model (temperature, several gaz densities),
- the poles coordinates and the universal time,
- the position and speed of stations,
- · · ·

In particular, GINS performs the classical orbit estimation process described in Section 1.1.2. GINS is also interfaced with software used in other geodesy institutes in Europe.

This chapter is organized as follows. In Section 2.2, we compare the features of the Cholesky factorization algorithm as it is implemented respectively in our solver and in ScaLAPACK or PLAPACK. For both implementations we successively describe the block algorithms in Section 2.2.1, the data distributions in Section 2.2.2, while the expected performance based on theoretical models is discussed in Section 2.2.3. The parallel implementation of our solver is detailed in Section 2.2.4. Then in Section 2.3.2, we give some numerical results obtained on the target operational parallel distributed platforms in order to evaluate the performance of our parallel implementation. These results are compared with those obtained with ScaLAPACK and PLAPACK. Finally, some concluding remarks are given in Section 2.4.

2.2 Parallel implementation

2.2.1 Block Cholesky factorization

There are two main variants used for the block Cholesky factorization. The variant referred

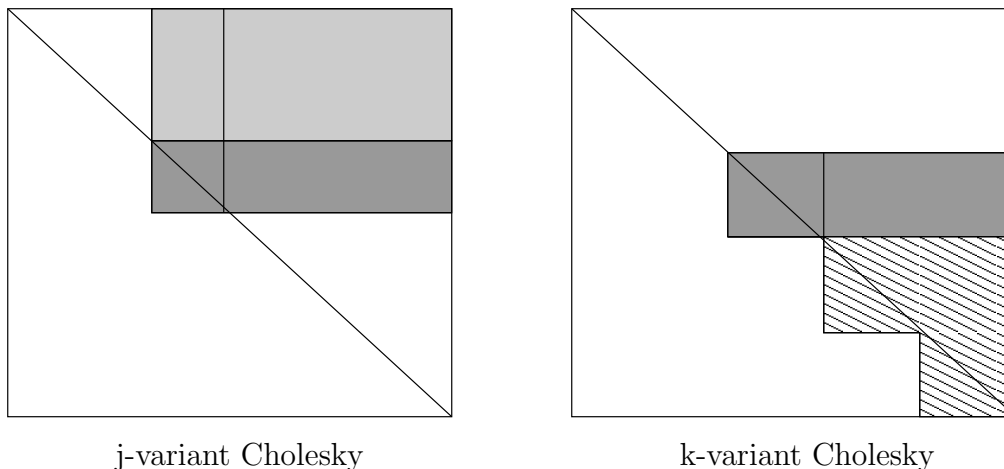j-variant Cholesky   k-variant Cholesky

Figure 2.1: Memory access patterns for two variants of the Cholesky factorization.

to as j-variant in [6, 78] or left-looking algorithm in [57] computes one block row of U at a time, using previously computed lines that are located at the top of the block row being updated. The second variant is referred to as k-variant in [6] or right-looking algorithm

23

in [57]. This variant performs the Cholesky decomposition $B = U^T U$ by computing a block row at each step and using it to update the trailing sub-matrix. The k-variant algorithm is implemented in the ScaLAPACK routine PDPOTRF [33] and the PLAPACK routine PLA_Chol [3]. The j-variant and the k-variant are described in Figure 2.1, where the shaded part refers to matrix elements being accessed, the dark shaded part represents the block row being computed (current row), and the hatched part corresponds to the data being updated.

In order to describe the difference between these two variants, let us consider the following symmetric block matrix:

$$
\begin{pmatrix}
B_{11} & B_{12} & B_{13} \\
{B_{12}}^T & \mathbf{B_{22}} & B_{23} \\
{B_{13}}^T & {B_{23}}^T & B_{33}
\end{pmatrix},
$$

where B_{22} is the diagonal block to be factorized, assuming that the first block row has been computed and that we want to obtain the second block row.

The j-variant algorithm allows us to advance the factorization as described below:

$$
\begin{pmatrix}
U_{11} & U_{12} & U_{13} \\
 & B_{22} & B_{23} \\
 & & B_{33}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
U_{11} & U_{12} & U_{13} \\
0 & U_{22} & U_{23} \\
0 & 0 & B_{33}
\end{pmatrix},
$$

where the second block row is computed with the following steps:

1. $U_{22} \leftarrow Chol(B_{22} - U_{12}^T U_{12})$,
2. $U_{23} \leftarrow U_{22}^{-T}(B_{23} - U_{12}^T U_{13})$.

The block B_{22} is first updated by using a matrix-matrix product and then factorized. Each block belonging to the rest of the block row is updated by a matrix-matrix multiply followed by a triangular solve with multiple right-hand sides.

According to the k-variant algorithm, we advance the factorization as follows:

$$
\begin{pmatrix}
U_{11} & U_{12} & U_{13} \\
 & \tilde{B}_{22} & \tilde{B}_{23} \\
 & & \tilde{B}_{33}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
U_{11} & U_{12} & U_{13} \\
0 & U_{22} & U_{23} \\
0 & 0 & \tilde{B}_{33}
\end{pmatrix},
$$

where the second block row is computed with the following steps:

1. $U_{22} \leftarrow Chol(\tilde{B}_{22})$,
2. $U_{23} \leftarrow U_{22}^{-T}\tilde{B}_{23}$,
3. $\tilde{B}_{33} \leftarrow \tilde{B}_{33} - U_{23}^T U_{23}$.

The block $\tilde{B}_{22}$ is first factorized then U_{23} is computed by using a triangular solve with multiple right-hand sides. Finally, the trailing sub-matrix $\tilde{B}_{33}$ is updated.

We notice that in both algorithms, the computational work is contained in three routines: the matrix-matrix multiply, the triangular solve with multiple right-hand sides and the Cholesky factorization. These numerical kernels can be respectively implemented using the Level 3 BLAS and LAPACK [5] routines: DGEMM, DTRSM and DPOTRF. As explained in [6], similar performance can be expected from both algorithms when the

three dominant routines are implemented equally well. Table 2.1 shows for two sample matrices that the operations involved in the DGEMM routine represent the major part of the operations for matrices involved in our applications and that this percentage increases with the size of the matrix (b corresponds to the block size chosen in our algorithm).

Table 2.1: Breakdown of floating-point operations for block Cholesky algorithm.

routine	$n = 500$ and $b = 64$	$n = 40,000$ and $b = 128$
DGEMM	84.6	99.5
DTRSM	14.1	0.5
DPOTRF	1.3	0.001

We point out that our j-variant Cholesky algorithm that is sometimes called left-looking has to be distinguished from the algorithm referred to as left-looking LU in [40, p. 83] which performs more DTRSM triangular solves than the two other LU variants considered in that book.

2.2.2 Data distribution

The data layout used in our solver is the one-dimensional block-cyclic column distribution. According to this layout, we choose a block size s and we divide the columns into groups of size s. Then we distribute these groups among processors in a cyclic manner column by column. Figure 2.2 shows an example of such a distribution for a 8-by-8 block matrix when we have 4 processors numbered 0,1,2 and 3. In this figure, each block is labeled with the number of the processor that stores it. The k-variant Cholesky algorithm implemented in ScaLAPACK or PLAPACK is based on a 2-D block-cyclic data distribution. In this type of distribution, the processors are arranged in a p-by-q rectangular array of processors. According to this choice of grid, the matrix blocks are assigned in a cyclic manner to different processors. Figure 2.2 shows an example of such a distribution for a 8-by-8 block matrix if we have 4 processors with $p = 2$ and $q = 2$. Note that a 1-D column distribution is a particular choice of 2-D distribution if $q = 1$. We refer to [23] for a more detailed description of these two types of data distribution. An important issue when choosing a data distribution is the load-balancing [40, p. 43] that involves to split the work among the processors throughout the algorithm to ensure the most efficient use of resources. This issue motivated the choice of the 2-D distribution in ScaLAPACK and PLAPACK. We refer to [23] for a more detailed description of the data distribution properties.

2.2.3 Performance prediction

As we have chosen an algorithm and a data distribution that are different from the standard libraries, we may see the consequence of this choice on the expected factorization times. We model the theoretical performance by evaluating the elapsed computational time and assuming that the communication is perfectly overlapped by the computation. The model parameters are the problem size n, the block size s, the number of processors $p \times q$, and the peak performance γ of a matrix-matrix product DGEMM of size s on the target machine. The parameter s is usually tuned in order to obtain the best value for γ. We

0	1	2	3	0	1	2	3
	1	2	3	0	1	2	3
		2	3	0	1	2	3
			3	0	1	2	3
				0	1	2	3
					1	2	3
						2	3
							3

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

1-D block cyclic column   2-D block cyclic $p = q = 2$

Figure 2.2: Block-cyclic data distributions.

consider a peak performance $\gamma \sim 3.3$ Gflops obtained with a block size $s = 128$, this choice being consistent with the performance of current parallel platforms. We denote n_b the number of columns of the block matrix.

As shown in Table 2.1, the major part of computation for both algorithms consists in performing the matrix-matrix multiply DGEMM. Let f denote the total number of arithmetic operations involved in the Cholesky factorization, then the elapsed factorization time can be accurately approximated by $\frac{f}{\gamma}$.

For the j-variant algorithm using a 1-D block column distribution, we have:

$$f = n_b \frac{s^3}{3} + \sum_{i=1}^{n_b} n_c((i-1)2s^3 + s^3),$$

where n_c is the maximum number of block columns treated per processor at step i.

For the k-variant algorithm and a 2-D block cyclic distribution for a $p \times q$ processors grid, we have:

$$f = n_b \frac{s^3}{3} + \sum_{i=1}^{n_b} (n_s s^3 + n_u 2s^3),$$

where n_s and n_u are the maximum number of blocks treated per processor involved respectively in the triangular solve and in the update of the trailing sub-matrix. We noticed experimentally that the choice of grid corresponding to $\frac{1}{2} \leq \frac{p}{q} \leq 1$ ensures the best time (or close to).

The efficiency of the algorithm can be evaluated by measuring how the performance degrades as the number of processors p increases while each processor uses the same amount of memory. This measures what we define as isomemory scalability. This type of performance measurement is suitable for our target applications since we aim to use most of the memory available for each processor of the parallel machine. Using the above formula of s, we get a theoretical factorization time and the resulting plots evaluate what we define as theoretical isomemory scalability of the algorithm. The problem size n_p is such that

each processor uses about the same amount of memory σ. In particular σ is the memory size required to solve a problem of order n_1 on one processor. We first choose a value of n_1 that is compatible with the minimal memory capacity of current parallel machines. We take $n_1 = 10,000$ which corresponds to a $\sigma = 800$ Mbytes for solvers storing the whole matrix and 400 Mbytes memory storage for our symmetric solver. Then we estimate the factorization time for a problem of size n_p that has to be solved on p processors. The invariance of storage per processor can be written $\frac{n_1(n_1+1)}{2} = \frac{n_p(n_p+1)}{2p}$, and then approximated by $n_p \simeq n_1\sqrt{p}$. In Figure 2.3 (a), we obtain the theoretical factorization times when the number of processors increases from 1 to 256 and the corresponding matrix size increases from 10,000 to 160,000. We see in Figure 2.3 (a) that a discrepancy between the two curves occurs for 32 processors (corresponding to a problem size of 56,569), due to difference of load-balance resulting from the choice of data distribution. When more than 64 processors are used, figures given by the model confirm the well-known result that a 2-D block cyclic distribution is better than a 1-D block cyclic column distribution. However, as we can see in Figure 2.3 (b), the scalability properties of both layouts are very similar for processor counts lower than 32.

(a) # processors varying from 1 to 256   (b) # processors varying from 1 to 32
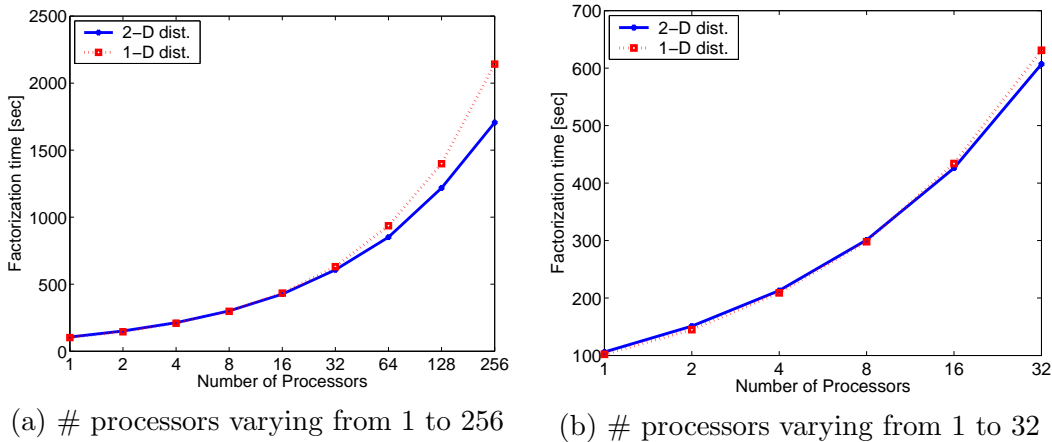
Figure 2.3: Theoretical isomemory scalability (peak = 3.3 Gflops).

2.2.4 Parallel implementation of the j-variant Cholesky algorithm

2.2.4.1 Choice of a data structure

As explained in Section 2.1, one of the main objectives of our implementation is to exploit the symmetry of the matrix by storing only about half of it. This implies that the memory of each processor will only contain the blocks assigned to this processor that belong to the upper triangular part of the symmetric matrix B. An appropriate choice for the structure containing these data must comply with the memory hierarchy constraints (level 1, 2 or 3 cache or TLB). More precisely, the fact that blocks are stored row-wise or column-wise in a local array might have a significant influence on performance in terms of Mflops. This will be illustrated in this paragraph by some experiments performed on IBM pSeries 690 (using the essl scientific library).

We saw in Section 2.2.1 that the calls to the BLAS 3 routine DGEMM and among them, the outer product between two different block columns represent the major part of the operations performed in a Cholesky factorization. If we denote (B_{ij}) the block triangular array corresponding to B, then this outer product consists in computing the operation on blocks $B_{ij} \leftarrow B_{ij} - \sum_{k=1}^{i-1} B_{ki}^T B_{kj}$ where i is the current row and $B_{ij}(j > i)$ is the element of column j to update. If n_{max} is the maximum number of blocks owned by a given processor and s is the block size defined in Section 2.2.3 then data blocks can be arranged in a Fortran array according to either a block-row storage or a block-column storage by using respectively an array of leading dimension s and $n_{max} \times s$ columns or an array of leading dimension $n_{max} \times s$ and s columns. As an example, the matrix sample given in Figure 2.2 leads for processor 0 to a local array that can be represented using a block-row storage by:

B_{11}	B_{15}	B_{25}	B_{35}	B_{45}	B_{55}

whereas it will be represented using a block-column storage by:

B_{11}
B_{15}
B_{25}
B_{35}
B_{45}
B_{55}

.

In order to evaluate the memory effect generated by each data structure, we plot in Figure 2.4 the performance obtained on the IBM pSeries 690 for the instruction $B_{ij} \leftarrow B_{ij} - \sum_{k=1}^{n_{max}} B_{ki}^T B_{kj}$ where the block columns $(B_{ki})_{k=1,n_{max}}$ and $(B_{kj})_{k=1,n_{max}}$ are stored either row-wise or column-wise in a Fortran array. In our experiments we set $s = 128$ and $n_{max} = 50$ but we obtained similar curves for $n_{max} > 50$. Figure 2.4 (a) is related to a

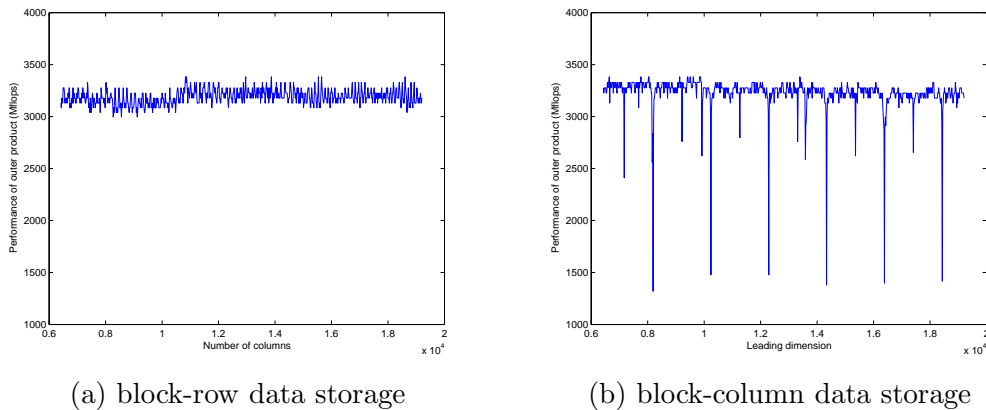(a) block-row data storage   (b) block-column data storage

Figure 2.4: Cache misses on IBM pSeries 690 for variable size arrays.

block-row data storage and the number of columns varies from $n_{max} \times s$ to $3 \times n_{max} \times s$.

Figure 2.4 (b) is related to a block-column data storage and the leading dimension varies from $n_{max} \times s$ to $3 \times n_{max} \times s$. Since the IBM pSeries 690 has a L1 cache 2-way associative of 32 KB, we obtain the worst performance when we successively access to data that are distant from a multiple of 16 KB (because they are stored in the same set of the cache memory). In a Fortran array, the distance between two consecutive data of the same line is the leading dimension of the array. Hence cache misses are expected when we access to two consecutive double-precision real in a line of an array whose leading dimension is a multiple of 2048 ($2048 = \frac{16K}{8}$). We notice in Figure 2.4 (b) that these spikes appear exactly with the same period when performing the outer product instruction and varying the leading dimension. We also observe the secondary spikes appearing at fraction of 2048 [48]. On the contrary, in a block-row structure, the distance between two consecutive entries in a line is s and performance obtained by computing the outer product is more stable with much less cache misses. Furthermore, if we use a block-row storage, the blocks belonging to the same block column will be contiguous in memory and then will map better to the highest levels of cache. This data contiguity is also an advantage when performing MPI operations. Taking these results into consideration, we chose the row-wise way of storing data blocks in the local memory of each processor.

2.2.4.2 Parallel algorithms

The performance of the normal equations assembly is not an issue in this chapter since we obtained a Gflops rate that is close to the sustained peak performance of a matrix-matrix product.
However we give here the parallel algorithm that was implemented in order to construct $B = A^T A$. This algorithm is such that:

1. All processors read the matrix A but a given processor computes only the contribution to blocks that are stored in its local memory.
   Rows of A are read and stored in a temporary array $BUFF$ containing r lines and $n+1$ columns (including thus the right-hand side vector b).

2. Blocks are then transmitted to a master processor to be written on disk.

3. The program uses both MPI and Open MP [28].

Algorithm 1. : $A^T A$ assembling

   while *not end-of-file* **do**

      for *Read A and b from file and store in BUFF*

      for *each local block* A_{ij} **OpenMP parallel do**

         Update A_{ij} *using BUFF (BLAS 3 DGEMM)*

      end *parallel do*

      for *each local block* b_i **OpenMP parallel do**

         Update b_i *using BUFF (BLAS 2 DGEMV)*

      end *parallel do*

   end *(while-loop)*

The parallel algorithm for the Cholesky factorization is described as follows. Half of the symmetric matrix B is loaded into memory distributed in a block upper triangular array $(B_{ij})_{j>i}$ according to the layout described in Section 2.2.2 and stored into memory according to the data storage described in Section 2.2.4.1. We recall that n_b is the number of columns of the block matrix (B_{ij}) and we denote by p the *id* of the current processor, $(proc(i))_{i=1,n_b}$ the array containing the processor *id* for each block column i, and *nprocs* the total number of processors involved in the parallel calculation. Then the following parallel block algorithm is executed simultaneously by all processors:

Algorithm 2. : Parallel block Cholesky

for $i = 1 : n_b$

    if $p = proc(i)$ **then**

        1. $proc(i)$ sends $(B_{ki}, \ k \leq i - 1)$ to other processors (**Isend1**)

        2. $B_{ii} \leftarrow B_{ii} - \sum_{k=1}^{i-1} B_{ki}^T B_{ki}$

        3. $B_{ii} \leftarrow Chol(B_{ii})$

        4. $proc(i)$ sends B_{ii} to other processors (**Isend2**)

        5. Completion of Isend1 (**Wait1**)

        6. for each block column $j > i$ assigned to proc p :

            $B_{ij} \leftarrow B_{ij} - \sum_{k=1}^{i-1} B_{ki}^T B_{kj}$

        7. Completion of Isend2 (**Wait2**)

        8. for each block column j assigned to proc p

            $B_{ij} \leftarrow B_{ii}^{-T} B_{ij}$

    else

        9. receive $(B_{ki}, k \leq i - 1)$ from $proc(i)$ (**Irecv1**)

        10. completion of Irecv1 (**Wait1**)

        11. receive B_{ii} from $proc(i)$ (**Irecv2**)

        12. for each block column $j > i$, $B_{ij} \leftarrow B_{ij} - \sum_{k=1}^{i-1} B_{ki}^T B_{kj}$

        13. completion of Irecv2 (**Wait2**)

        14. $B_{ij} \leftarrow B_{ii}^{-T} B_{ij}$

    endif

  end *(i-loop)*

As it can be seen in instructions 1 and 4 of Algorithm 2, data are sent as soon as they are available by a non blocking instruction MPI_Isend. This allows the overlapping of the communications by the computations. The update of the rest of the block row can be performed by all the processors while the communications are still ongoing. When these data are necessary for further use by the sending processor at step 6 and 8, the communication must be completed by using MPI_Wait instruction (respectively in 5 and 7). In the same manner, messages are received as soon as possible using the non blocking

instruction MPI_Irecv in 9 and 11 and then completed by MPI_Wait just before the received data are used in the computations involved in instructions 12 and 14. These non-blocking message exchanges save about 10% of the factorization times on a matrix of dimension 24,000 on 16 processors of the IBM platform, compared with an implementation using blocking instructions MPI_Send and MPI_Recv.

2.3 Experiments

Experiments were conducted on the following platforms:

1. one node of an IBM pSeries 690 (32 processors Power-4/1.3 GHz and 64 Gbytes memory per node),

2. a HP-COMPAQ Alpha Server (10 SMP nodes with 4 processors EV68 1 GHz and 4 Gbytes memory per node) installed at CERFACS.

Computations with ScaLAPACK were performed using the pessl library on the IBM pSeries 690 and the COMPAQ ScaLAPACK V1.0.0 library on the HP-COMPAQ Alpha Server. On both machines, we used PLAPACK release 3.0.

2.3.1 Tuning the normal equations formation

First, we obtained the peak performance of the matrix-matrix product by running DGEMM on the machine for increasing size of matrix. The maximum value was 3.4 Gflops. Then the program parameters r (read block size) and s (block size) have to be tuned in order to find the optimal values. As expected, we observed here a BLAS 3 effect: the peak performance is achieved for block size larger than a minimum value related to memory characteristics such as sizes of the various cache levels.
As shown in Figure 2.5, we have good results for a buffer read block size greater than 256. In Figure 2.6 we notice that the optimal block size for size matrix 20,000 is greater than 128.

2.3.2 Performance analysis of the Cholesky factorization

We compare performance obtained by our j-variant Cholesky algorithm and that of the Cholesky factorization routines PDPOTRF and PLA_Chol on the parallel machines described above. In Figure 2.7, we plot the run times obtained for several problem sizes such that the memory allocation per processor is constant (i.e in the condition of isomemory scalability as defined in Section 2.2.3). On this curve, the problem size varies from 10,000 (for 1 processor and corresponding to a memory storage per processor of about 800 Mbytes compatible with our target platforms) to 56,569 (for 32 processors). Figure 2.7 shows that, for up to 16 processors, our solver provides factorization times that are close to those obtained by ScaLAPACK and PLAPACK. For 32 processors, performance of the libraries using a 2-D block cyclic distribution are, as expected, better and this is consistent with the theoretical model described in Section 2.2.3.

Table 2.2 measures the scalability in floating-point operations of both algorithms, that we name isoflop scalability. It shows how the performance per processor (in Gflops) of
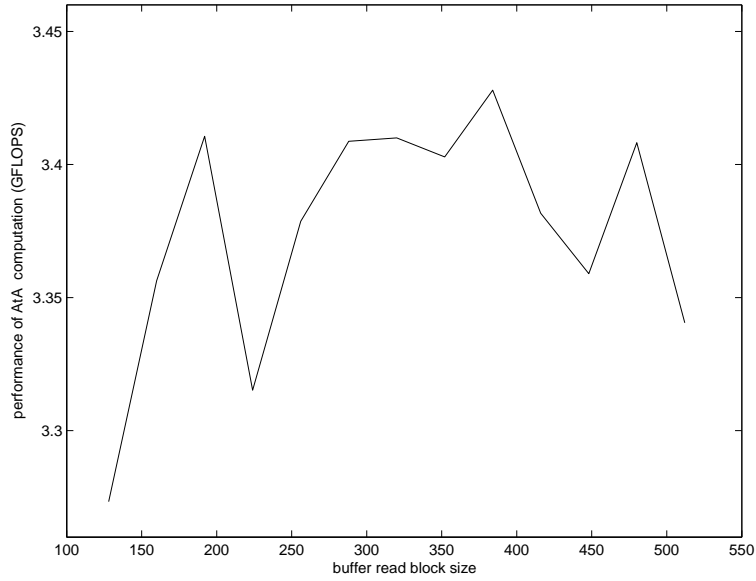
Figure 2.5: variation of read block size for matrix size 20,000, block size 128, 2 processors, 4 threads
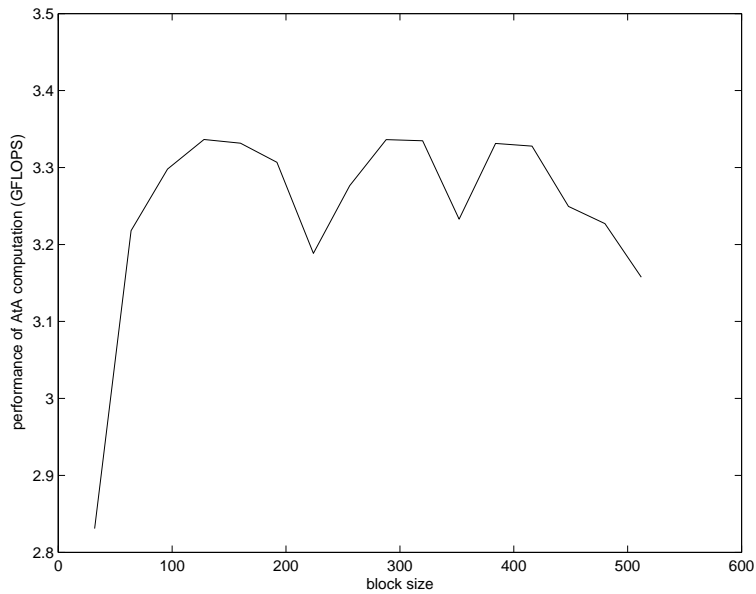

Figure 2.6: variation of block size for matrix size 20,000, read block size 256, 2 processors, 4 threads

the algorithms behaves when the number of processors increases and while each processor performs the same number of floating-point operations. Measuring a performance *per processor* enables us to compare easily with the peak performance of the machine. In the

33
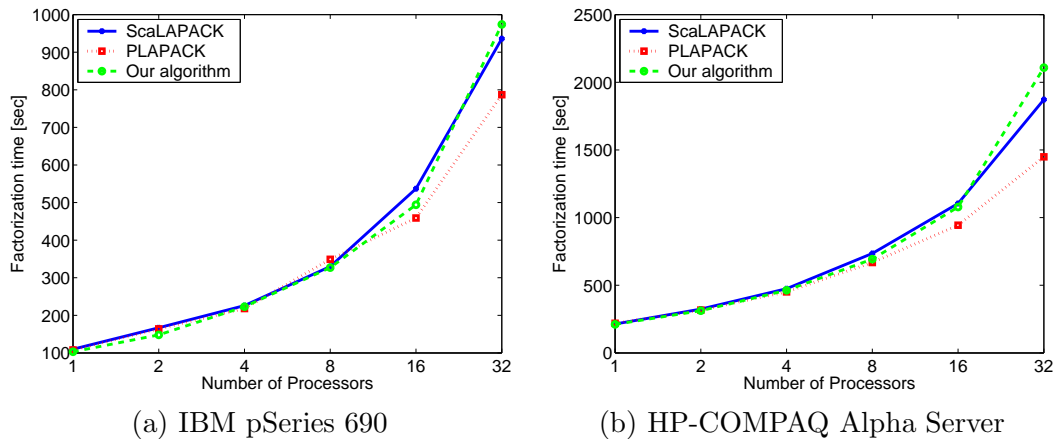
Figure 2.7: Isomemory scalability of the Cholesky factorization.

ideal case, i.e when the communication is perfectly overlapped by the computation, we expect this performance being constant. This implies that, with the same notations as in Section 2.2.3, $n_p^3 = pn_1^3$ i.e $n_p = n_1 \sqrt[3]{p}$. In our experiments, we chose $n_1 = 9,449$ and this corresponds to a memory storage per processor of about 700 Mbytes for ScaLAPACK and PLAPACK (350 Mbytes for our solver) and a fixed number of about $2.8\ 10^{11}$ floating-point operations per processor. It can be seen in Table 2.2 that the performance of the three routines is similar up to 16 processors and that, for 32 processors, our solver is slightly less efficient than ScaLAPACK and PLAPACK. For all routines, we notice that performance decreases significantly with the number of processors, due to the cost of the communications.

Table 2.2: Isoflop scalability for the Cholesky factorization (Gflops).

nb procs	size	IBM pSeries 690			HP-COMPAQ Alpha Server		
		our solver	PDPOTRF	PLA_Chol	our solver	PDPOTRF	PLA_Chol
1	9449	3.3	2.9	2.9	1.5	1.5	1.5
2	11906	3.2	2.9	2.9	1.5	1.4	1.5
4	15000	2.9	2.9	2.9	1.4	1.3	1.4
8	18899	2.7	2.7	2.7	1.3	1.2	1.3
16	23811	2.5	2.5	2.5	1.1	1.1	1.2
32	30000	2	2.2	2.2	0.7	0.8	1

Now if we compare factorization times obtained on each machine for a given problem size, we notice that the IBM pSeries 690 is about twice faster than the HP-COMPAQ Alpha, as it was expected since the Power 4 and the EV68 processors have a peak computation rate of respectively 5.2 Gflops and 2 Gflops.
Our solver gives performance results that are similar to ScaLAPACK and PLAPACK on moderately parallel platforms (up to 32 processors). Nevertheless, it is less scalable than these libraries on higher processor counts because it uses a one-dimensional block cyclic

distribution. However our solver uses about half the memory required by ScaLAPACK and PLAPACK. Thus choosing between our solver and the standard parallel libraries will involve a trade-off between storage and computation time.

2.3.3 Gravity field computation

The spherical harmonic coefficients $\overline{C}_{lm}$ and $\overline{S}_{lm}$ that represent the gravity field are computed by GINS via the method described in Section 1.1.2.

In the following example we consider 10 days of observations using GRACE measurements. The total number of observations is $165,960$. The number of gravity field parameters (problem size n) was initially $22,801$ but the problem was not solved for degrees larger that 99 ($l_{max} = 99$). Then the number of parameters effectively computed is $(l_{max}+1)^2 = 10,000$.

The solution computed by GINS is validated by the physicists using the spectrum depicted in Figure 2.8. This figure represents the geoid height error between the computed model and a reference model. The geoid height error is expressed as a function of the degree l and using a logarithmic scale for the y-axis.

The computed solution and the reference solution are represented respectively by the red and the black curves. We notice that both curves coincide because the order of the difference is of several centimeters. However, a slight discrepancy between the two curves occurs for the largest values of l.

For a more accurate analysis, the physicists plot the difference between the computed and the reference solution for each degree (blue curve). We notice that the slope of this curve increases with the degree. As mentioned in [11], the low degrees coefficients are not accurately computed. This can be verified here because there is a spike at the beginning of the blue curve ($l \leq 4$).

The yellow curve represents for each degree l the same difference as the blue curve but cumulated (in quadratic sum) for the degrees lower than l. This curve indicates to the physicists the global quality of the solution. The computed solution is very satisfactory for $l \leq 100$. For values of l larger than 100, we notice that this curve goes up, showing then that the difference between the reference and the computed gravity field increases.

Figure 2.9 represents a geoid map (which is partial because of the truncation) resulting from the gravity field computed above. Note that the main mountains (red) and the submarine pits (blue) appear clearly on this map.
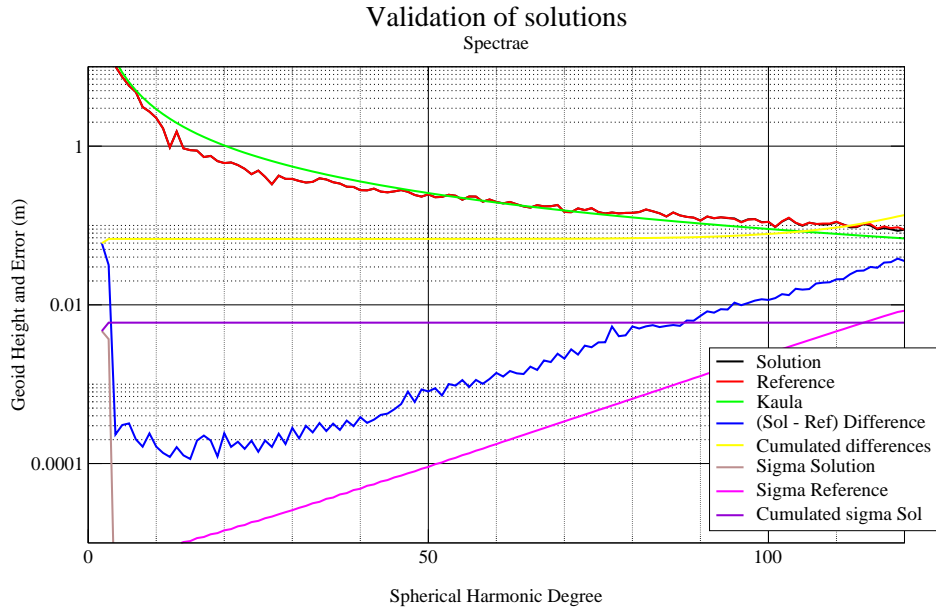
Figure 2.8: Gravity field computation for $m = 165,960$ and $n = 22,801$.
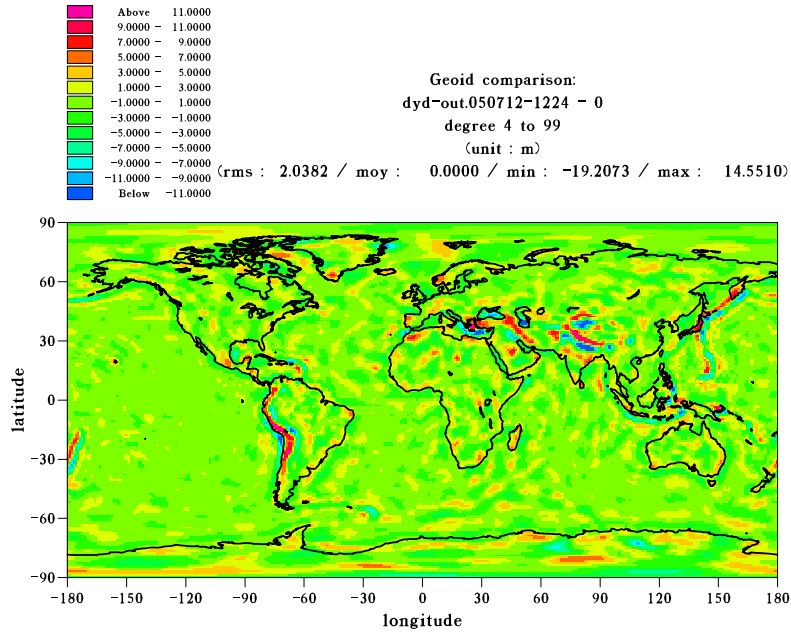
Figure 2.9: Geoid map ($4 \leq l \leq 99$).

37

2.4   Conclusion of Chapter 2

In this chapter, we described an implementation of a parallel symmetric solver involving algorithmic and distribution choices that are different from that of ScaLAPACK and PLA-PACK. The behaviour that one can expect from the theoretical model has been confirmed by experiments up to 32 processors.

The solution obtained complies with the operational constraints and we can solve problems of size corresponding to the target application. For instance, problems of size 100,000 were solved on an IBM pSeries 690 in 1 hr 20 mins using 32 processors. We also solved problems of size 100,000 on HP-COMPAQ Alpha in 2 hr 40 mins using 32 processors on nodes that have 2 Gbytes memory per processor.

After integrating this solver into the GINS software that computes the gravity field coefficients, we obtained a solution that satisfies the requirements of the physicist in terms of performance and accuracy.
Other approaches to design parallel linear solvers might consist in using high-level parallel routines in PLAPACK or PBLAS/ScaLAPACK. They deserve to be investigated and are the topic of Chapter 3 where their benefit in terms of load-balance will be studied.
From a numerical point of view, some additional work can be developed. In this respect, a condition number estimate is studied in Chapter 4 that will be added to our implementation to make the normal equations approach more robust.

II

Chapter 3

A distributed packed storage for scalable large parallel calculations

3.1 Introduction to distributed packed storage

Even if the current parallel platforms provide increasing memory capacity, they are also used to solve ever larger linear systems. This is the case for instance in geosciences or electromagnetic computations where the usual problem size is several tens of thousands. It is now possible to perform these calculations since the distributed memory parallel machines available today offer several Gbytes memory per processor. But when the dense matrices involved in these computations are symmetric, Hermitian or triangular, it could be worth to take profit of the structure by storing only half the matrix.

The ScaLAPACK [23] library has been designed to perform linear algebra parallel calculations on dense matrices. Contrary to the serial library LAPACK [5], ScaLAPACK does not currently support packed format for symmetric, Hermitian or triangular matrices [37]. A parallel solver has been studied in Chapter 2 (and in [8]) that solves linear least squares problems encountered in gravity field calculations using the normal equations method. This solver also handles large symmetric linear systems in complex arithmetic resulting from BEM modelling of electromagnetic scattering. This solver uses about half the memory required by ScaLAPACK and gives performance results similar to ScaLAPACK on moderately parallel platforms (up to 32 processors). Nevertheless, it is less scalable than ScaLAPACK on higher processor counts because it uses a one-dimensional block cyclic distribution [23, p. 58]. The distributed packed storage format proposed in this chapter exploits the good load balancing of the two-dimensional block cyclic distribution [23, p. 58] as it is implemented in ScaLAPACK. The calls to PBLAS [32] or ScaLAPACK routines in the applications that exploit this format will also ensure the portability of software built upon them since these libraries are supported by all current parallel platforms. We shall see that, thanks to the set of routines we provide, building applications using matrices in distributed packed form is very easy and gives good performance while saving a significant amount of memory compared with the full storage of the matrix.

This chapter is organized as follows. In Section 3.2, we give an overview of the existing packed formats. The purpose of Section 3.3 is to describe the implementation of the distributed packed storage that we intend to use in parallel algorithms based on PBLAS

or ScaLAPACK kernel routines. In Section 3.4, we explain how the Cholesky factorization can be implemented using the distributed packed format, this includes the descriptions of algorithms and a performance analysis on the IBM pSeries 690 and the CRAY XD1 cluster. Then in Section 3.5, we present an implementation of the updating of the QR factorization that keeps the R factor in memory using the distributed packed format and we give performance results on the IBM pSeries 690. Finally, some concluding comments are given in Section 3.6.

3.2   Generalities on packed storage formats

The sequential libraries LAPACK [5] or BLAS [39] provide a packed storage for symmetric, hermitian or triangular matrices. This format allows us to store half the matrix by addressing only the lower-triangular or upper-triangular part of the matrix, this part being held by columns.

For instance, the upper triangle of $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ * & a_{22} & a_{23} \\ * & * & a_{33} \end{pmatrix}$ will be stored compactly in the matrix AP (A Packed) such that $AP = \begin{bmatrix} a_{11}, a_{12}, a_{22}, a_{13}, a_{23}, a_{33} \end{bmatrix}$.

For symmetric matrices, either the lower triangle or the upper triangle can be stored. In both cases, the triangle is packed by columns but one may notice that this is the same as storing the opposite triangle by rows. This packed storage format has been implemented in several routines of the Level-2 BLAS and LAPACK for:

- solving symmetric indefinite or symmetric/Hermitian positive definite linear systems,
- computing eigenvalues and eigenvectors for symmetric or symmetric-definite generalized eigenproblems (with condition number estimation and error bounds on the solution),
- multiplying symmetric/Hermitian matrices and solving triangular systems.

Unfortunately, this format gives poor performance results when used in dense linear algebra computations since the algorithms are not able to use optimally the memory hierarchy. Blocked operations cannot be performed which means no Level-3 BLAS and a dramatic loss of efficiency compared to the full storage (see e.g [4]).

An answer to this problem consists in using blocking techniques and storing the lower-triangular or upper-triangular part of the blocked matrix. For instance, the blocked matrix $A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ * & A_{22} & A_{23} \\ * & * & A_{33} \end{pmatrix}$ can be stored in the blocked packed matrix

$$\begin{bmatrix} A_{11}, A_{12}, A_{22}, A_{13}, A_{23}, A_{33} \end{bmatrix}$$

or

$$\begin{bmatrix} A_{11}, A_{12}, A_{13}, A_{22}, A_{23}, A_{33} \end{bmatrix}.$$

For serial implementations, the authors of [4] define a so-called Upper (resp. Lower) Blocked Hybrid Format. In both formats, the blocks are ordered by columns to permit efficient operations on blocks using Level-3 BLAS (e.g blocked Cholesky algorithm in [4])

and the diagonal blocks are stored in packed format so that exactly half of the matrix is stored.

Regarding parallel implementations, there is presently no satisfying packed storage available for dense matrices. A packed storage for symmetric matrices distributed in a 1-D block cyclic column distribution is used in [8] for a least squares solver based on the normal equations approach. In this implementation the blocks are ordered by columns and stored in a block-row array, giving an extra-storage due to the diagonal blocks that are fully stored. All blocks are manipulated using Level-3 BLAS or LAPACK blocked routines but communication is performed by MPI primitives whereas the distributed packed format that we propose in this paper relies on the ScaLAPACK communication layer BLACS [41].

A preliminary study on a packed storage extension for ScaLAPACK has been carried in [36]. In this format only the lower (or upper) part of each block column of the matrix is stored into a panel considered as a separate ScaLAPACK matrix. This packed storage stores also the entire diagonal blocks. We can find in [36] experiments on the Cholesky factorization and symmetric eigensolvers. Our approach is an extension of this format and we will see its limitation.

3.3 Distributed packed format

3.3.1 Definitions

ScaLAPACK proposes a data layout based on a two-dimensional block cyclic distribution. In this type of distribution, a matrix of size n is divided into blocks of size s that are assigned to processors in cyclic manner according to a $p \times q$ process grid. We refer to [23] for more details about this data layout. The blocks of size s that are spread among processors are called elementary blocks and the blocks of size $p.s \times q.s$ corresponding to the $p \times q$ process grid are called grid blocks.

In order to be stored in a distributed packed format, a matrix is first partitioned into larger square blocks of size b such that b is proportional to $l.s$ where l is the least common multiple of p and q ($b \geq l.s$). We define these blocks as "distributed blocks". In the rest of this chapter, the algorithms will be expressed in terms of distributed blocks that will be simply called "blocks". Note that the distributed block performs naturally what is defined in [98] as algorithmic blocking.

The following figure summarizes the hierarchy between the elementary block (hosted by one processor), the grid block (corresponding to the process grid), and the distributed block (square block consisting of grid blocks). It shows the three kinds of blocks that we get when we consider a 2×3 process grid, $s = 1$, $b = 6$ and each block is labeled with the number of processor that stores it.

43
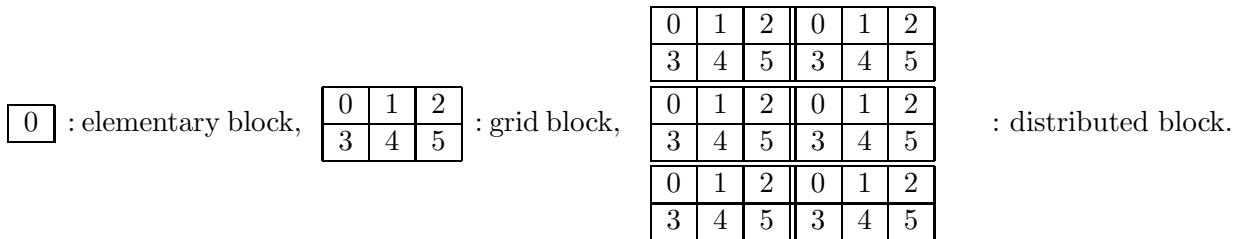
We consider here a matrix A partitioned into distributed blocks A_{ij} and A can be either symmetric or upper triangular or lower triangular. We propose to store A compactly in a distributed packed format that consists in storing the blocks belonging to the upper or the lower triangle of A in a ScaLAPACK matrix ADP (A Distributed Packed).

The blocks of A will be stored horizontally in ADP so that the entries in the elementary, grid and distributed blocks are contiguous in memory and then will map better to the highest levels of cache.

Let us consider the following symmetric matrix A described using distributed blocks, that is

$$A = \begin{pmatrix} A_{11} & A_{21}^T & A_{31}^T \\ A_{21} & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

We provide two ways of storing A using our distributed packed format. In the Lower distributed packed format, the lower triangle of A is packed **by columns** in ADP i.e:

$$ADP = \begin{bmatrix} A_{11} & A_{21} & A_{31} & A_{22} & A_{32} & A_{33} \end{bmatrix}.$$

In the Upper distributed packed format, the upper triangle of A is packed **by rows** in ADP i.e:

$$ADP = \begin{bmatrix} A_{11} & A_{21}^T & A_{31}^T & A_{22} & A_{32}^T & A_{33} \end{bmatrix}.$$

The distributed packed storage for upper and lower triangular matrices follows from that of a symmetric matrix since the upper triangular blocked matrix $A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ 0 & A_{22} & A_{23} \\ 0 & 0 & A_{33} \end{pmatrix}$ is stored in a packed distributed format as

$$ADP = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{22} & A_{23} & A_{33} \end{bmatrix}$$

and the lower triangular blocked matrix $A = \begin{pmatrix} A_{11} & 0 & 0 \\ A_{21} & A_{22} & 0 \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$ will be stored as

$$ADP = \begin{bmatrix} A_{11} & A_{21} & A_{31} & A_{22} & A_{32} & A_{33} \end{bmatrix}.$$

Notice that contrary to LAPACK where upper and lower triangular matrices are both packed by columns, our distributed packed format is different for upper and lower triangular matrices since they are respectively packed by rows and columns. Note also that the diagonal blocks are full blocks and then do not exploit the triangular or symmetric structure.

Throughout this chapter we will use the following designations and notations. The distributed packed format will be simply referred to as the packed format and an implementation using this format as a packed implementation. We denote by $N = \frac{n}{b}$ the number

of block rows in A. The packed structure ADP will be described using the A_{ij} as previously or will be denoted as the blocked matrix $\begin{bmatrix} B_1 & B_2 & B_3 & B_4 & B_5 & B_6 \end{bmatrix}$. A block $B_k = A_{ij}$ in ADP will be addressed through the indirect addressing INDGET that maps (i, j) to k.

3.3.2 Tuning parameters

In order to obtain the best Gflops performance as possible, one usually determines the dominant operation(s) involved in the computation in terms of floating-point operation count (e.g the matrix-matrix multiplication performed by the Level-3 routine DGEMM in the sequential Cholesky factorization). Then we try to optimize the efficiency of this dominant operation by tuning the program parameters. In parallel implementations, these parameters are often the size s of an elementary block size and the values of p and q in the process grid.

If the DGEMM routine is the dominant operation, then s is generally determined as being the value that enables us to obtain the best sustained performance for a matrix-matrix product of size s on the target machine. This parameter is closely related to the machine characteristics and to the memory hierarchy constraints (level 1,2,3 cache or TLB).

The optimal values for the physical parameters p and q generally depend on the type of algorithm that is implemented. In [33] optimal values for the ratio $\frac{p}{q}$ are proposed for the LU, Cholesky and QR factorizations performed by ScaLAPACK.

Performance tuning can sometimes become more complicated when the dominant operation (i.e the operation that must be optimized) changes with the distribution parameters or when the dominant operation in terms of flops is not the dominant operation in terms of time. In that case, a heuristic needs to be found that will often lead to a compromise (not degrading the most efficient routine while improving the less efficient one).

Finally, a parameter that influences the performance of a packed implementation is the size of b. As seen in Section 3.3.1, we have $b \geq l.s$. b may be chosen significantly larger than $l.s$ but in that case it would demand more memory storage because the diagonal blocks would be bigger. The ratio between the memory storage required by a block size b and that required by a block size $l.s$ is given by

$$\alpha = \left(\frac{n}{b}(\frac{n}{b} + 1)b^2 \right) / \left(\frac{n}{l.s}(\frac{n}{l.s} + 1)(l.s)^2 \right) = \frac{n+b}{n+l.s}.$$

Then the increase (in percentage) of the memory storage when using a blocking of size b instead of using a blocking of size $l.s$ is expressed by $\alpha - 1$ i.e $\frac{b-l.s}{n+l.s}$. In the rest of this chapter, this quantity is referred to as extra-storage.

Remark 1. We did not compare the required storage using a blocking size b with the "ideal" storage corresponding to the LAPACK packed storage described in Section 3.2 since we store here the entire diagonal blocks and our parallel implementation is based on distributed blocks whose minimum size is $l.s \times l.s$ for easy use of ScaLAPACK routines. Indeed, in our packed distributed storage, the choice $b = l.s$ is optimal from a memory point of view but in the absolute, an optimal packed storage would store $n(n+1)/2$ entries of the matrix.

Let ρ be the maximum extra-storage that we are ready to accept. Then the maximum value of b will be such that $b \leq \rho(n + l.s) + l.s$. We shall see that the choice of b will represent a trade-off between the performance (if large b improve the dominant operations) and the memory storage (b must be consistent with the maximum extra-storage that we accept).

3.4 Application to the Cholesky factorization

Based on the distributed packed storage defined in Section 3.3.1, we describe in this section how a packed distributed Cholesky factorization can be designed on top of some PBLAS and ScaLAPACK kernels.

3.4.1 Description of the algorithms

The packed implementation of the Cholesky factorization is based on the Level-3 PBLAS routines PDGEMM (matrix-matrix product), PDSYRK (rank-k update), PDTRSM (solving triangular systems with multiple right-hand-sides) and on the ScaLAPACK routine PDPOTRF(Cholesky factorization). We present in this section the packed implementations of the right-looking and left-looking variants of the Cholesky factorization that are given respectively in Algorithms 3 and 4. The symmetric positive definite matrix partitioned into distributed blocks $\begin{pmatrix} B_1 & B_2^T & B_3^T \\ B_2 & B_4 & B_5^T \\ B_3 & B_5 & B_6 \end{pmatrix}$ is stored in a lower distributed packed format as $\begin{bmatrix} B_1 & B_2 & B_3 & B_4 & B_5 & B_6 \end{bmatrix}$ that we also denote by $B_{1:6}$.

We notice in both algorithms that the PDGEMM and the PDTRSM instructions involve rectangular arrays. In the implementations, these instructions are performed using a loop that performs the multiplication of the $b \times b$ blocks one by one in order to reduce cache misses.

We also point out that when b is minimum ($b = l.s$) then N is maximum and thus the number of synchronizations involved in Algorithm 3 is maximum, if we assume that any call to a ScaLAPACK routine results in a synchronization. This explains why, even if taking $b = l.s$ requires less memory storage, this value of b will be rarely chosen.

46

Algorithm 3. *:* Packed right-looking Cholesky

for $i = 1 : N$

$\quad j = INDGET(i, i)$

$\quad B_j \leftarrow chol(B_j)$ **(PDPOTRF**)

$\quad B_{j+1:j+N-i} \leftarrow B_{j+1:j+N-i} B_j^{-T}$ **(PDTRSM**)

$\quad$ for $ii = i + 1 : N$

$\quad\quad k = INDGET(ii, ii)$

$\quad\quad B_k \leftarrow B_k - B_{j+ii-i} B_{j+ii-i}^T$ **(PDSYRK** *rank-b update*)

$\quad\quad B_{k+1:k+N-ii} \quad\quad \leftarrow \quad\quad B_{k+1:k+N-ii} \quad - $
$B_{j+1+ii-i:j+N-i} B_{j+ii-i}^T$ **(PDGEMM**)

$\quad$ end *(ii-loop)*

end *(i-loop)*

Algorithm 4. *:* Packed left-looking Cholesky

for $i = 1 : N$

$\quad j = INDGET(i, i)$

$\quad$ for $ii = 1 : i - 1$

$\quad\quad k = INDGET(ii, i)$

$\quad\quad B_j \leftarrow B_j - B_k B_k^T$ **(PDSYRK** *rank-b update*)

$\quad\quad B_{j+1:j+N-ii}^T \leftarrow B_{j+1:j+N-ii}^T - B_{k+1:k+N-i}^T B_k^T$ **(PDGEMM**)

$\quad$ end *(ii-loop)*

$\quad B_j \leftarrow chol(B_j)$ **(PDPOTRF**)

$\quad B_{j+1:j+N-i}^T \leftarrow B_{j+1:j+N-i}^T B_j^{-T}$ **(PDTRSM**)

end *(i-loop)*

3.4.2 Tuning

Both algorithms were implemented on an IBM pSeries 690 (2 nodes of 32 processors Power-4/1.7 GHz and 64 Gbytes memory per node) installed at CINES[1]. There were linked with

[1]Centre Informatique National de l'Enseignement Supérieur, Montpellier, France

the PBLAS and ScaLAPACK libraries provided by the vendor (in particular the Pessl library).

As in a sequential blocked Cholesky algorithm, the matrix-matrix multiply performed by the routines PDGEMM or PDSYRK represents the major part of the computation. Both routines call essentially the Level-3 BLAS routine DGEMM that gives good performance for $s \geq 128$ for our platform. The value of $s = 128$ will be taken in all following experiments.

3.4.2.1 Influence of the distributed block

Let now see the influence of b on the performance on the kernel routines used in our packed implementation.
Figure 3.1 represents the unitary performance of each routine on a matrix of size b using 16 processors. The curves show that the performance increases with b. However we notice that a spike occurs for $b = 8192$ and a smaller one for $b = 4096$. Since $b = 8192$ corresponds to local array per processor that has a leading dimension of 2048, the main spike is explained by cache misses occurring when we access to two consecutive double-precision real distant of 2048 due the size of the IBM pSeries 690 L1 cache. The spike observed for $b = 4096$ corresponds to secondary cache misses. Details on this phenomenon have been given in Section 2.2.4.1.
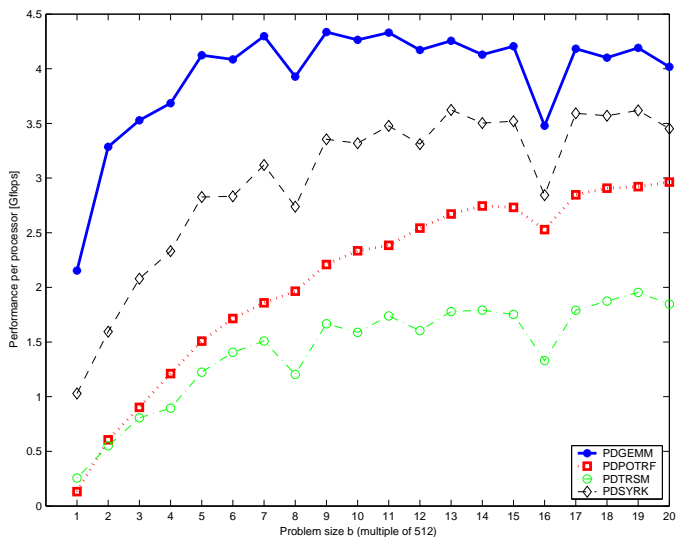

Figure 3.1: Performance of PBLAS routines involved in the Cholesky factorization (16 processors).

Regarding Algorithm 3, the repartition of floating-point operations among the different routines depends on the problem size n and on the block size b. But the performance of each routine can also depend on b and on the number of processors involved in the computation.

48

Table 3.1 gives for a particular example the number of floating-point operations and the time spent in each routine for our packed implementation and for the full storage ScaLA-PACK factorization. We notice that the PDTRSM routine performs 8.1% of the operations and takes 41% of the time. This shows that, as observed in Figure 3.1, PDTRSM is far less efficient than PDGEMM and PDSYRK. We note that the number of floating-point operations corresponding to the Cholesky factorization of the diagonal blocks is negligible, as mentioned in Section 2.2.1. These operations are performed by PDPOTRF in the packed implementation and by PDPOTF2 in ScaLAPACK. They are not mentioned in Table 3.1 because they do not represent a significant time in the global factorization. Then

Table 3.1: Breakdown of operations and time for right-looking Cholesky ($n = 81920$, 64 processors).

	packed solver ($b = 5120$)		ScaLAPACK ($s = 128$)	
PBLAS routine	% operations	% time	% operations	% time
PDGEMM and PDSYRK	91.9	58	99.8	92
PDTRSM	8.1	41	0.2	0.7

a heuristic for tuning the parameter b may consist in choosing an "acceptable" ratio r of operations performed by the PDTRSM routine. The floating-point operations performed by PDTRSM are:

$$\sum_{i=1}^{N-1} ib^3 = b^3 \frac{N(N-1)}{2}$$

Hence, since the Cholesky factorization involves $\frac{n^3}{3} = \frac{(Nb)^3}{3}$ operations, we have

$$\frac{3(N-1)}{2N^2} \leq r.$$

Thus we get $N = \frac{n}{b} \geq \frac{3+\sqrt{9-24r}}{4r}$ and the maximum value for b is:

$$b_{max} = \frac{4rn}{3 + \sqrt{9 - 24r}}.$$

As seen in Section 3.3.2, we have $b \leq \rho(n + l.s) + l.s$ and then the chosen value for b will be $\min(b_{max}, \rho(n + l.s) + l.s)$.

3.4.2.2 Influence of the process grid

The $p \times q$ process grid can also have an influence on the performance of the code. In a packed implementation, the classical choice of a roughly squared grid ($\frac{1}{2} \leq \frac{p}{q} \leq 1$) given by [33] for the ScaLAPACK Cholesky factorization is not necessarily the best one because the operation that slows down the global performance of the program is the PDTRSM routine. Figure 3.2 shows for a 16 processors grid that the PDTRSM routine applied to one block of size b is more efficient when using a rectangular grid such that $\frac{p}{q} > 1$ whatever the value of b is. Table 3.2 summarizes the tuning parameters and the heuristics for the packed Cholesky implementation on the IBM pSeries 690.
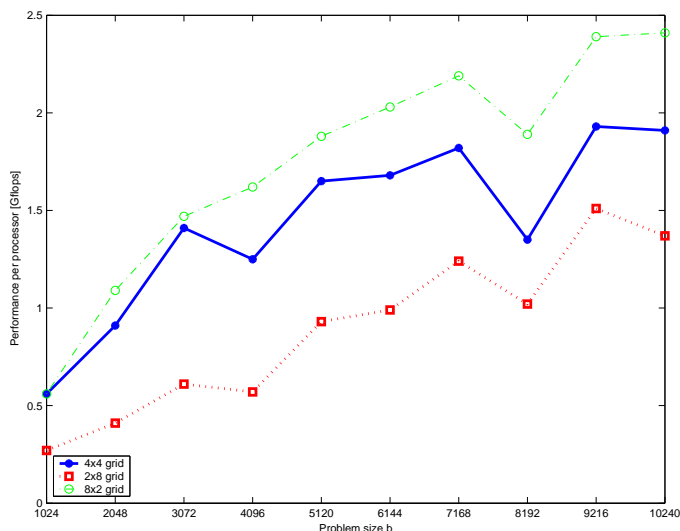
Figure 3.2: Performance of PDTRSM for different process grids (16 processors).

Table 3.2: Tuning parameters/heuristics for packed Cholesky implementation (IBM pSeries 690).

parameter	heuristic	suggested value (IBM pSeries 690)
s	sustained performance of DGEMM	128
p, q	$\frac{p}{q} > 1$	depends on processor count
b	accepted extra-storage ρ	$b \leq \min(\frac{4rn}{3+\sqrt{9-24r}}, \rho(n+l.s)+l.s)$
	accepted PDTRSM operations r	

3.4.3 Performance results

In the following results, the processor count varies from 1 to 64 and the corresponding problem size n has been determined so that each processor uses about the same amount of memory (with $n = 10240$ for 1 processor, which corresponds to a storage of about 840 Mbytes per processor for ScaLAPACK). For each problem size, the size b of the distributed block has been determined using the heuristic given in Section 3.4.2. We first compute for each problem size the maximum value of b that can be obtained by accepting a maximum of 10% of memory extra-storage ρ and 15% of operations performed by the routine PDTRSM. Then b is adjusted to the nearest number that is lower to the maximum value, proportional to $l.s$ and a submultiple of n. The resulting values of b are given in Table 3.3. The actual extra-storage corresponding to the chosen value of b is the quantity $\alpha - 1 = \frac{b-l.s}{n+l.s}$ that has been defined in Section 3.3.2.

We present below performance results obtained by the right-looking implementation (Algorithm 3 described in Section 3.4.1) rather than the left-looking one (Algorithm 4) since it gives in our experiments factorization times that are slightly better for high processor count. The selected values of b are those displayed in Table 3.3. In accordance

Table 3.3: Tuned block size for $\rho = 0.1$ and $r = 0.15$.

problem size n	10240	14336	20480	28672	40960	61440	81920
$p \times q$ grid	1×1	2×1	4×1	4×2	8×2	8×4	16×4
block size b	1024	1024	2048	2048	4096	6144	10240
extra-storage	8.6%	7%	7.3%	5.3%	7.3%	8.2%	9.7%

with Section 3.4.2, the grid parameters are such that $\frac{p}{q} > 1$ and more precisely $2 \leq \frac{p}{q} \leq 4$ since it provides experimentally better results. In that table, t_{packed} is the resulting factorization time.

In Table 3.4 the performance of the packed solver is compared with that of a ScaLAPACK Cholesky factorization storing the whole matrix but performing the same number of operations (routine PDPOTRF). The corresponding factorization time t_{scal} is obtained using a $p \times q$ process grid in accordance with [33] i.e such that $\frac{1}{2} \leq \frac{p}{q} \leq 1$. The difference of performance is then measured by computing the overhead $\frac{t_{packed} - t_{scal}}{t_{scal}}$.

Table 3.4: Cholesky factorization time (sec) for packed solver and ScaLAPACK.

n	10240	14336	20480	28672	40960	61440	81920
# procs	1	2	4	8	16	32	64
t_{packed}	102	127	194	290	474	912	1298
$p \times q$	1	2×1	4×1	4×2	8×2	8×4	16×4
t_{scal}	106	153	219	321	471	890	1178
$p \times q$	1	1×2	2×2	2×4	4×4	4×8	8×8
overhead	-3.8%	-17%	-11.4%	-9.7%	0.6%	2.5%	10%

We notice in Table 3.4 that the factorization times are better than ScaLAPACK for less than 32 processors and similar to ScaLAPACK for 32 processors. For 64 processors, there is an overhead of 10%. This overhead can be diminished by considering larger blocks. Table 3.5 shows that the performance increases with b but that it also requires more memory.

Table 3.5: Performance vs extra-storage ($n = 81920, 16 \times 4$ procs).

block size b	10240	20480
factorization time (sec)	1298	1234
overhead with ScalaPACK	10%	5%
extra-storage	9.7%	22%

In order to evaluate the scalability of the packed solver and of the ScaLAPACK Cholesky, we plot in Figure 3.3 the Gflops performance of both algorithms. Since each algorithm maintains a constant memory use per processor, these curves measure what is referred to as isoefficiency or isogranularity in [23, p. 96]. We notice that the packed solver is more efficient for small processor count. Performance degrades for both algorithms when

51

the number of processors increases but the ScaLAPACK routine is slightly faster for 32 and 64 processors.
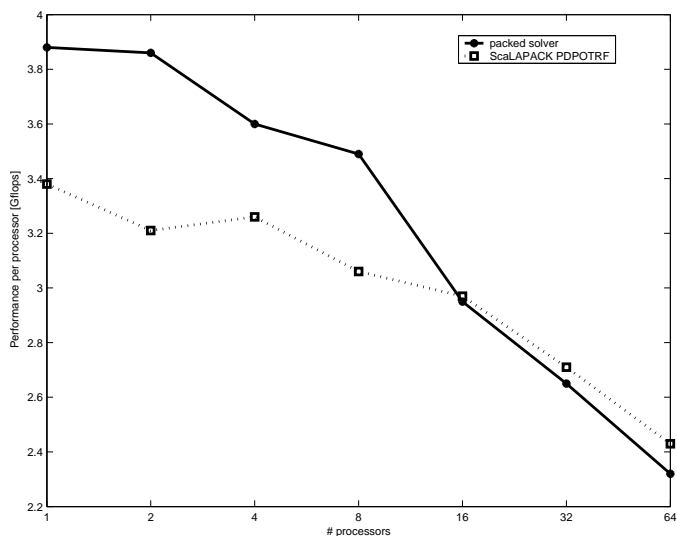
Figure 3.3: Isogranularity of packed Cholesky solver and ScaLAPACK on IBM pSeries 690.

Remark 2. Since the memory required by the packed implementation depends on the chosen value for b, it is interesting to compare in details in Table 3.6 the memory per processor in Mbytes required by the packed solver and by the ScaLAPACK routine PDPOTRF for the experiments summarized previously in Table 3.4 and Figure 3.3. The quantities mentioned here do not include the work arrays used by ScaLAPACK computations. Note that an optimal storage that stores exactly $n(n+1)/2$ entries would require about 419 Mbytes.

The experiments described in this paragraph have been performed on nodes of IBM pSeries 690 offering 2 Gbytes memory per processor. Due to the memory required by PDPOTRF (more than 850 Mbytes), we could not achieve these comparisons on nodes having 1 Gbytes memory per processor (because part of the memory is also used by the system). This confirms again the limitation due to the full storage for symmetric matrices.

Table 3.6: Memory required per processor by the packed solver and ScaLAPACK (Mbytes).

n	10240	14336	20480	28672	40960	61440	81920
# procs	1	2	4	8	16	32	64
block size b	1024	1024	2048	2048	4096	6144	10240
Mbytes for packed solver	461	440	461	440	461	519	472
Mbytes for PDPOTRF	839	822	839	822	839	943	839
saved memory	45%	46%	45%	46%	45%	45%	44%

Remark 3. By using 4 nodes of the IBM pSeries 690 (32-way SMP each) available at the CINES, we could evaluate how the performance of the packed solver degrades when running on 128 processors and with the same memory per processor as previously. We present in Table 3.7 the performance obtained for $n = 114688$ and 16×8 processors. The best factorization time corresponds to about twice the time obtained using 64 processors. We notice that there is here no interest in choosing a block size larger than 16384 since the performance degrades and it requires more storage.

Table 3.7: Performance of packed implementation using 128 (16×8) processors ($n = 114688$).

block size b	16384	28672
factorization time (sec)	2741	3043
Gflops per proc.	1.43	1.29
extra-storage	12.3%	22.8%

Regarding the Cholesky factorization, some other experiments were performed. They deserve to be mentioned here because they have influenced some choices made for the implementation described previously.

The first one was about the influence of the structure of the ScaLAPACK array for the packed structure. As mentioned in Section 3.3 the distributed blocks are stored row-wise in the ScaLAPACK array ADP. This choice is justified by the fact that the operations in Algorithm 3 are performed by column and thus the blocks are contiguous in memory. In Figure 3.4, we use 16 processors on a same node of IBM and plot the Gflops performance of a packed implementation of the Cholesky using either a row-wise storage or a column-wise for the distributed blocks. It confirms that a block-row storage provides better performance, thanks to a better data locality.

Some experiments were also performed in order to compare the right-looking and the left-looking variants of the packed implementation (using an horizontal structure for ADP).
In the left-looking variant, the matrix A is stored compactly using the Upper distributed packed format. This enables us to have a memory contiguity of the blocks belonging to a same block-row in Algorithm 4. Table 3.8 contains the factorization times obtained for both algorithms and shows that the left-looking implementation is slightly better when using less than 16 processors and that the right-looking implementation provides better results when using more than 32 processors.

3.4.4   Experiments on clusters

Several experiments were performed on the CRAY XD1 cluster at CERFACS (120 AMD Opteron 2.4 GHz, 240 Gbytes memory).
We consider problem sizes similar to those defined in Section 3.4.3 for the IBM pSeries 690 i.e by considering a constant memory per processor.
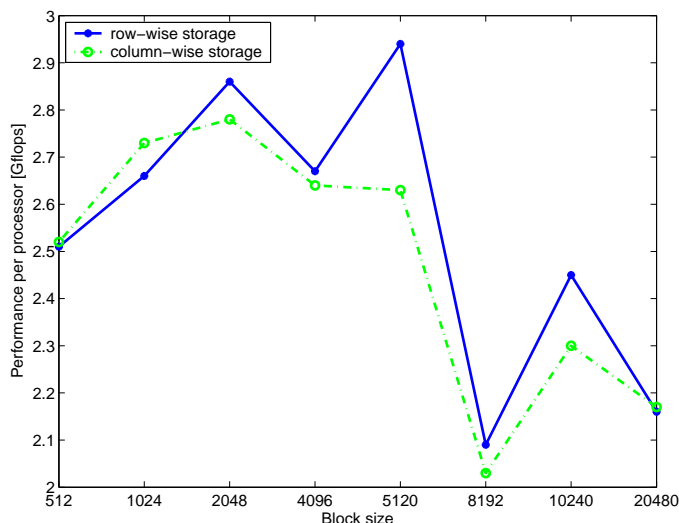
Figure 3.4: row-wise storage vs column-wise storage in the distributed packed format.

Table 3.8: Factorization time (sec) for left-looking and right-looking variants of the packed distributed Cholesky.

n	10240	14336	20480	28672	40960	61440	81920
# procs	1	2	4	8	16	32	64
block size b	256	256	512	1024	1024	5120	5120
left-looking	87	123	194	283	465	966	1578
right-looking	90	146	222	306	548	1092	1442

As shown in Figure 3.5, the sustained peak rate of a serial matrix-matrix product DGEMM on this machine is about 4 Gflops. Taking $s = 128$ is here again satisfactory because it provides a DGEMM rate of 3.8 Gflops.

Since the PBLAS routine PDGEMM represents the major part of the computation in a parallel Cholesky factorization, we also tune the parameters of this routine for several problem sizes. Figure 3.6 shows that, contrary to the IBM pSeries 690, the performance of PDGEMM degrades significantly when the problem size increases for a given number of processors (here 4). This can be explained by the slower communication system in a cluster architecture. This encourages us to choose smaller block sizes than in Section 3.4.3 for the packed implementation.

We also notice that a $p \times q$ rectangular grid such that $p > q$ gives better results than a square grid. This has a consequence on the choice of grid for the Cholesky factorization. Table 3.9 contains the factorization times obtained for PDPOTRF using $p \times q$ process grids with $p = q$ and $p > q$. We observe that a rectangular grid provides a performance that is twice that of a square grid when we consider 4 or 16 processors. That leads us to consider the same grids for ScaLAPACK and the packed solver.

54
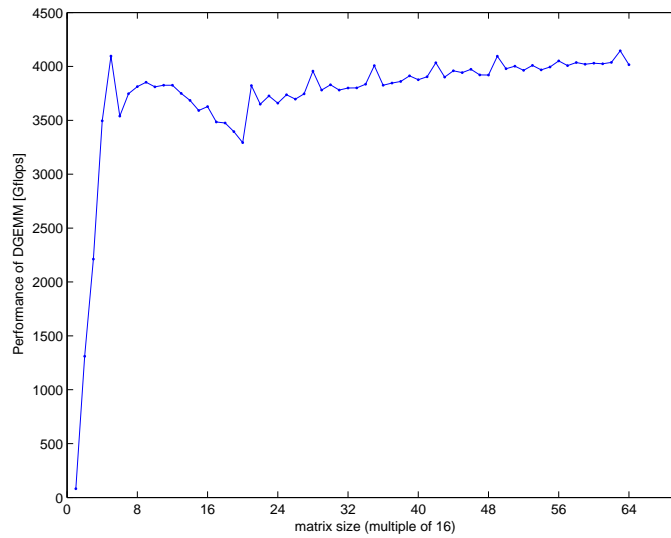
Figure 3.5: Performance of serial matrix-matrix multiply DGEMM on one processor of the CRAY Cluster XD1.
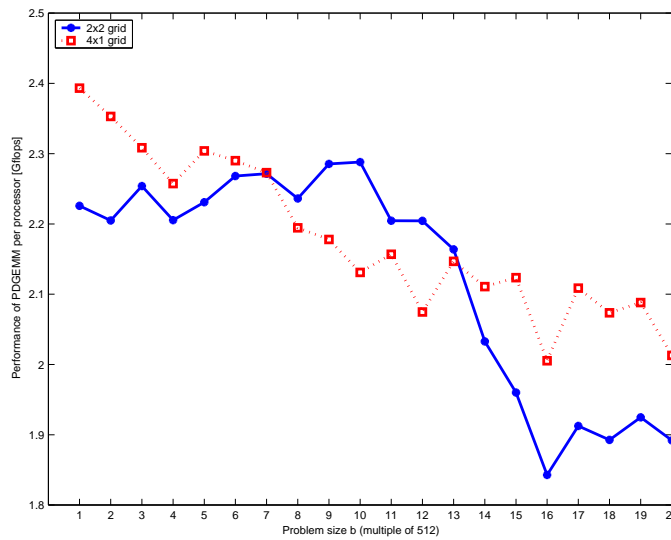
Figure 3.6: Performance of PDGEMM routine for 2 grid shapes on the CRAY Cluster XD1.

After tuning the grid shapes for ScaLAPACK, we now compare the performance of the packed implementation and the ScaLAPACK routine PDPOTRF. Table 3.10 shows that ScaLAPACK performance is slightly better than that of the packed solver for more than 8 processors. The overhead $\frac{t_{packed}-t_{scal}}{t_{scal}}$ is always lower than 14 % and can considered as acceptable. There is no collapse for processor count larger than 32.

The isogranularity of each algorithm measured in Gflops per second is depicted in

Table 3.9: Influence of the process grid on ScaLAPACK PDPOTRF performance (CRAY XD1).

n	20480	40960	81920
$p \times q$	2×2	4×4	8×8
Factorization time (sec)	574	1073	2292
$p \times q$	4×1	8×2	16×4
Factorization time (sec)	255	561	1622

Table 3.10: Cholesky factorization time (sec) for packed solver and ScaLAPACK (CRAY XD1).

n	10240	14336	20480	28672	40960	61440	81920	107520
procs	1	2	4	8	16	32	64	112
$p \times q$	1	2x1	4x1	4x2	8x2	8x4	16x4	28x4
b	128	256	512	512	1024	1024	2048	3584
t_{packed}	113	168	270	419	616	1127	1473	2022
t_{scal}	153	184	278	370	561	1005	1399	1776
overhead	-26%	-9%	-3%	13%	10%	12%	5%	14%

Figure 3.7. Similarly to the IBM pSeries 690, the packed solver is more efficient for small processor count and the ScaLAPACK Cholesky provides better Gflops rates when using more than 8 processors while having similar behaviour when the number of processors increases.

Figure 3.7: Isogranularity of packed Cholesky solver and ScaLAPACK on CRAY XD1 Cluster.

To summarize these experiments on the cluster, we note that the tuning of the distributed block is simplified because of the monotonic decrease in performance of the PDGEMM routine (see Figure 3.6). This implies that we should use small blocks for the packed implementation. This has the advantage of requiring minimal memory.
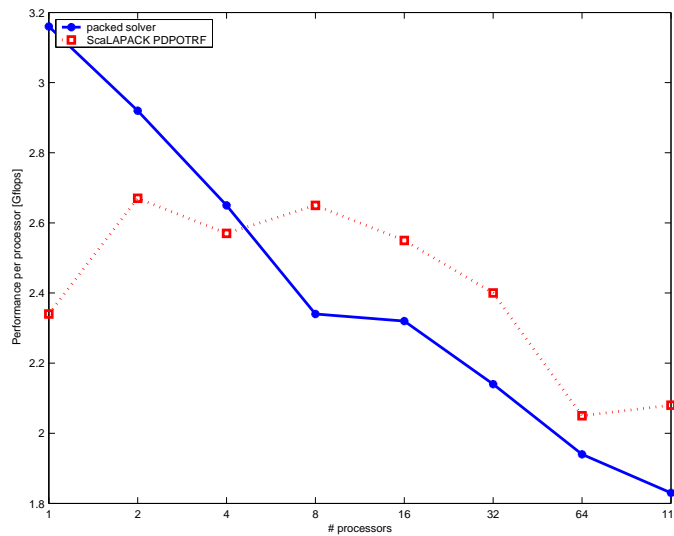On the other hand, the performance of the ScaLAPACK routine PDPOTRF is improved by considering rectangular grids that are not common for the Cholesky factorization.
Then PDPOTRF gives slightly better performance than the packed solver when using more than 8 processors (up to 14% for 112 processors).

3.5   Application to the updating of a QR factorization

3.5.1   Description of the algorithm

Many parameter estimation problems lead to linear least squares problems of the type

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|_2 \tag{3.1}$$

where each row of A and b corresponds to one observation, these observations being collected periodically. If (3.1) is solved via a QR approach, then it is appropriate to update the previous QR factorization or at least the R factor with the newly collected data rather than computing a whole QR factorization involving original data combined with new data. Such an incremental algorithm is more efficient in terms of computational cost than performing the QR factorization on the whole matrix.
A possible out-of-core algorithm that updates a QR factorization is described in [58]. In that case, the R factor and the right-hand side are both updated and a new solution is computed that takes into account the new observations. For faster computation of a partial solution or of the covariance, we may want to keep the R factor in-core.

A packed implementation of the QR factorization updating can be based on the ScaLA-PACK routines PDGEQRF (QR factorization) and PDORMQR (multiplication by Q^T).
We suppose that the R factor is partitioned into distributed blocks $\begin{pmatrix} B_1 & B_2 & B_3 \\ 0 & B_4 & B_5 \\ 0 & 0 & B_6 \end{pmatrix}$
and stored in a distributed packed format as $\begin{bmatrix} B_1 & B_2 & B_3 & B_4 & B_5 & B_6 \end{bmatrix}$ that we also denote by $B_{1:6}$.
The new observations are stored in a block matrix $L_{1:N}$ that contains $N.b$ columns and we first assume that L contains b rows. The updating of R is obtained by successively performing the QR factorization of each block row of R with L, as described below. At the first step, we factor:

$$\boxed{\begin{array}{c} B_{1:3} \\ \hline L_{1:3} \end{array}} \longrightarrow \boxed{\begin{array}{c} \widetilde{B}_{1:3} \\ \hline \widetilde{L}_{1:3} \end{array}}$$

and we advance the updating of the R factor as follows:

and so on until completion.

Let now consider the work array $C = \begin{bmatrix} B_{j:j+N-i} \\ \widetilde{L}_{i:N} \end{bmatrix}$ where $j = INDGET(i,i)$, i.e C contains $2b$ rows and $(N-i+1)b$ columns.

The different ways for implementing the i-th stage in the R updating are described in Figure 3.8, where the shaded part refers to the part of C that is factored by the routine PDGEQRF and the dark shaded part represents the part of C to which we apply the Householder transformations using the routine PDORMQR.

In Figure 3.8 (a), we perform the QR factorization of the whole matrix C. In that case,

(a) PDGEQRF on C

(b) PDGEQRF + PDORMQR

(c) PDGEQRF partial
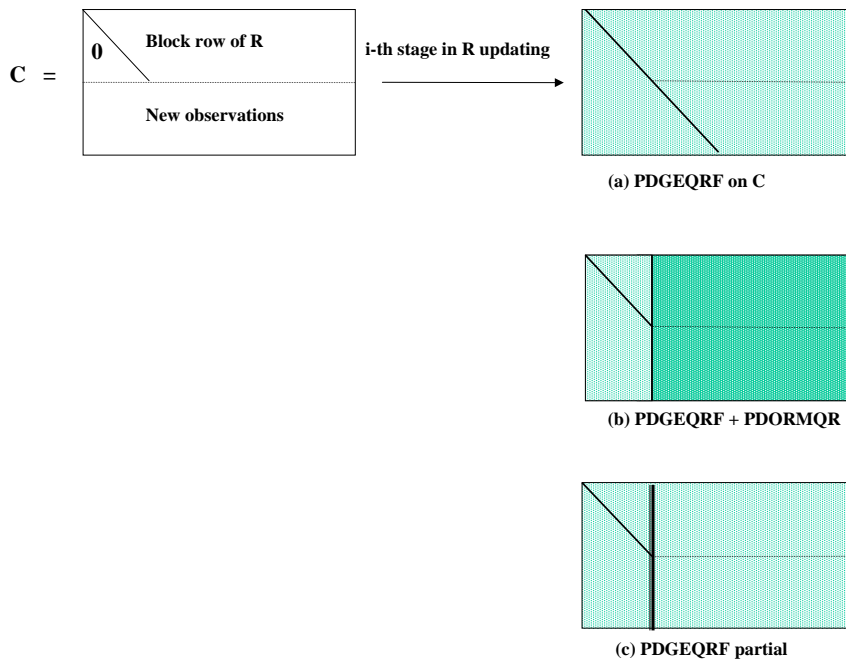
Figure 3.8: Different possibilities for the QR factorization of C.

using the flop counts given in [50, p. 213 and 225], the R updating algorithm involves about $4bn^2$ operations (if $n \gg b$).

This flop count can be reduced by performing a QR factorization of the first b columns of C subsequently followed by the updating of the remaining columns by the Householder transformations (Figure 3.8 (b)). From [50], the computational cost becomes about $3bn^2$ (still if $n \gg b$).

Figure 3.9 compares the Gflops rate of a QR factorization of a $b \times 2b$ matrix performed either using PDGEQRF on the whole matrix or using PDGEQRF on the first b columns then PDORMQR on the remaining b columns. One may notice that the combination of

PDGEQRF and PDORMQR is much less efficient due to the extra-cost in communication (using subsequently two routines involves one more synchronization and also PDORMQR exchanges data that was already available while executing PDGEQRF). Thus step i in the
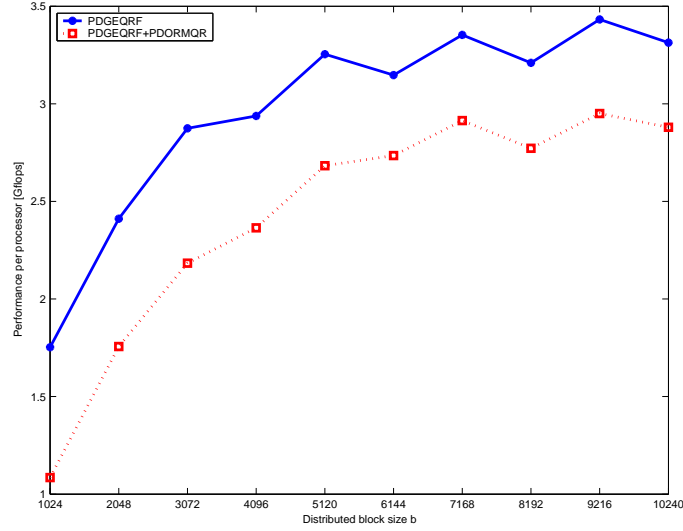
Figure 3.9: QR factorization of a $b \times 2b$ matrix (4 processors of IBM pSeries 690).

R updating was implemented as a call to the PDGEQRF routine applied to the whole matrix C that stops the factorization after the first b columns. The global updating involves $3bn^2$ operations that are performed efficiently. For that reason, we modified the ScaLAPACK routine PDGEQRF by stopping the QR factorization of an m-by-n matrix after the $\frac{m}{2}$ first columns. The so-modified routine is named PDGEQRF_partial. Algorithm 5 represents the updating of the R factor using PDGEQRF_partial.

Algorithm 5. : Updating the R factor in a QR factorization

    read new data in $L_{1:N}$; $\widetilde{L}_{1:N} \leftarrow L_{1:N}$

    for $i = 1 : N$

        $j = INDGET(i,i)$

        $C \leftarrow \begin{bmatrix} B_{j:j+N-i} \\ \widetilde{L}_{i:N} \end{bmatrix}$

        $\widetilde{C} = qr(C)$ *stopped after the b first columns have been factored*
        $\rightarrow \widetilde{C} = \begin{bmatrix} \widetilde{B}_{j:j+N-i} \\ * & \overline{L}_{i+2:N} \end{bmatrix}$ (**PDGEQRF_partial***)*

        $B_{j:j+N-i} \leftarrow \widetilde{B}_{j:j+N-i}$

    end *(i-loop)*

One may notice that this algorithm does not take into account the upper triangular structure of $B_{j:j}$. As it will be confirmed on experiments, this can be compensated by storing more data into L and thus applying PDGEQRF_partial to a block matrix C containing more that $2b$ rows. As a result, the number of floating-point operations will also decrease. The initialization of the R factor can be implemented by starting with $R = 0$ and then by successively updating the previous rows by a new one until we processed the N block rows of R. This allows us to compare the Gflops performance of Algorithm 5 with that of PDGEQRF applied to an n-by-n matrix (notice that the factorization time cannot be compared since the initialization of R by successive updates involves 1.5 more operations).

We point out that the ScaLAPACK or PBLAS routines do not have to be modified to support the new packed storage format. The PDGEQRF routine has been modified only for sake of performance.

The algorithm described above is appropriate for GOCE calculations where we consider A as dense. Note that in the case where A has a given block structure, it is possible to reduce the computational effort by exploiting the structure of A and R [49].

Remark 4. A condition number estimate for GOCE:

We consider a right-hand side $X = \begin{pmatrix} X_1 \\ \vdots \\ X_N \end{pmatrix}$ that is partitioned into distributed row-blocks of size b and mapped onto the same process grid as the R factor. Then a simple implementation of the triangular solve in packed format can be described in Algorithm 6.

Algorithm 6. *: Packed triangular solve*

    for $i = 1 : N$

        $j = INDGET(i, i)$

        for $k = i + 1 : N$

            $X_i \leftarrow X_i - B_{j+k-i}X_k$ **(PDGEMV***)*

        end *(k-loop)*

        $X_i \leftarrow B_j^{-1}X_i$ **(PDTRSV***)*

    end *(i-loop)*

This routine performs the product of R^{-1} by a vector X. Using the same kind of implementation, we obtain routines that perform the products RX, $R^T X$ and $R^{-T} X$ using the packed format. Then it becomes easy to get a packed implementation of the condition number of $A^T A = R^T R$ based for instance on the Power method or on the Lanczos method [50].

Remark 5. Kaula regularization for GOCE:

We observe that the Kaula regularization mentioned in Section 1.2.3 can be performed by

computing the upper triangular factor in the QR factorization of

$$\begin{pmatrix} R \\ D \end{pmatrix},$$

with $D = diag(0, \cdots, 0, \alpha, \cdots, \alpha)$. This regularization is performed via the updating algorithm described above.

3.5.2 Performance results

All the following experiments have been performed on the IBM pSeries 690 described in Section 3.3.2. Contrary to the Cholesky factorization, there is here only one ScaLAPACK routine that must be considered when tuning our program parameters s, b and the $p \times q$ process grid. Here again s will be chosen equal to 128 since it provides good performance of the PDGEQRF routine on the chosen platform. The $p \times q$ process grid is determined in accordance with [33] i.e such that $\frac{1}{4} \leq \frac{p}{q} \leq \frac{1}{2}$. The value of b cannot be too large since it also influences the size of the matrix C and then the required storage for the calculation. We take $b = l.s$ in our experiments.

Table 3.11 compares the performance of the initialization of R by successive updates with the performance of a QR factorization of an n-by-n matrix using PDGEQRF. To see the effect of the communication on the performance, we choose values of n such that each processor uses roughly the same amount of memory. We notice that the obtained Gflops rates are similar to ScaLAPACK for all processor counts considered in this study.

Table 3.11: Initialization of R by updates vs ScaLAPACK QR (Gflops).

problem size n	10240	14336	20480	28672	40960	61440	81920
$p \times q$ process grid	1	1×2	1×4	2×4	2×8	4×8	4×16
Initialization of R	2.47	3.02	3.30	2.87	2.89	2.80	2.37
ScaLAPACK PDGEQRF	3.50	3.36	3.20	3.25	2.93	2.83	2.63

Let n_L be the number of rows in the matrix L that contains the new observations for updating the QR factorization. In Table 3.12, we update a 25600×25600 matrix R by 51200 new observations and n_L varies from 512 to 25600. As expected at the end of Section 3.5.1, the number of operations decreases as n_L increases. This gain in operations is evaluated by computing the ratio between the operations involved in the updating of R and the operations required in a QR factorization of the 76800×25600 matrix containing the original data and the new observations. Then if n_L increases, the factorization time decreases but the performance is stable (close to the peak performance of the ScaLAPACK routine PDGEQRF). Here again, choosing the best size for L corresponds to find a compromise between performance (in time) and storage since large L demands more storage.

Table 3.12: Updating of a 25600×25600 R factor by 51200 new observations (1×4 procs).

Number of rows in L	512	1024	2048	5120	10240	12800	25600
Storage (Gbytes)	0.72	0.75	0.80	0.96	1.22	1.35	2.00
Flops overhead (vs ScaLAPACK)	1.50	1.31	1.22	1.16	1.14	1.14	1.13
Factorization time (sec)	7577	5824	5255	5077	5001	4894	4981
Performance (Gflops)	3.33	3.61	3.59	3.47	3.44	3.50	3.40

3.6   Conclusion of Chapter 3

The distributed packed storage defined in this chapter allows us to handle symmetric and triangular matrices in parallel distributed environments using ScaLAPACK and Level-3 PBLAS routines. This format was implemented for the Cholesky factorization and for the updating of the R factor in a QR factorization. The example of the Cholesky factorization shows that choosing the optimal distributed block size leads to a trade off between performance and memory. Some heuristics have been proposed that provide performance similar to ScaLAPACK while requiring much less memory. The QR updating is another framework that can benefit from the packed distributed storage. The associated algorithm gives very good performance due to the efficient implementation of the ScaLAPACK kernel routine PDGEQRF. In general, the performance of our packed implementations relies on the performance of the underlying ScaLAPACK kernel routines. The good results that we obtained encourage us to extend this packed storage to other linear algebra calculations involving symmetric and triangular matrices. An improvement might consist in extending to blocked parallel implementations the Rectangular Full Packed storage that was recently defined in [59] for serial implementations. Such a format could enable us to minimize the storage required by the diagonal blocks but it will be necessary to evaluate the performance of the ScaLAPACK kernel routines on this format. This will be the topic of further studies.

The codes using the distributed packed storage are research codes that have not been tried yet in the GOCE application. The Cholesky code provides better performance than the operational solver described in Chapter 2 when using more than 32 processors. This can be explained by the better load-balancing of the two-dimensional block cyclic distribution supported in ScaLAPACK. The QR code that we presented in this chapter should, as discussed in Section 1.2.2, improve the accuracy of GOCE computations when it will be integrated in the GINS software. Another advantage of solvers proposed in Chapter 3 is that all communications are performed by standard routines from the PBLAS or ScaLAPACK libraries. This will ensure a better portability of the codes implementing the distributed packed storage.

III

Chapter 4

Partial condition number for linear least squares problems

4.1 Sensitivity of least squares problems

4.1.1 Introduction

Alan Turing [96] introduced the sensitivity of a numerical problem solution to changes in its data as a way to measure the difficulty of solving the problem accurately. Condition numbers are now considered as fundamental to sensitivity analysis. They have been applied to many problems of linear algebra such as linear systems, linear least squares, or eigenvalue problems [21, 34, 42, 61, 90]. They are interesting quantities in themselves, as they can be interpreted as a measure of the mathematical difficulty of the problem. Indeed, by definition, unless exact arithmetic is used, it is expected to be difficult to find accurately the solution of an ill-conditioned problem.

Traditional error analysis, which aims at taking into account various error sources such as discretization, truncature and round-off error is also expressed in terms of backward errors [104]. Associated with an approximated solution, they measure the distance between the data of the original problem and the nearby problem for which the approximate solution is an exact one. Since backward errors measure perturbation on the data, they can be compared with errors such as discretization errors or instrumental errors. Then they can be used to assess the quality of a solution obtained by a direct method or stop the iteration of an iterative solver.

A particularly interesting situation occurs when both the condition numbers of a problem and the corresponding backward error associated with an approximate solution are available, thanks to closed formulae or estimates. In this case, a common "rule of thumb" is that the product of those two quantities provides an estimation of the true forward error associated with the computed solution [61]. More precisely, since the condition number is a measure of the sensitivity at first order of the solution, the estimate of the forward error will be accurate if the first order assumption is reasonable. The joint use of condition number and backward error to analyze the mathematical difficulty, the stability and the accuracy of a computed solution is often termed "backward error analysis".

In this paragraph, we give some basic material for the estimation of condition numbers in linear algebra and we present some recent results.

4.1.2 Condition numbers

The condition number is a measure of the sensitivity of a mapping to perturbations. Among the most general definitions of condition numbers, the seminal paper [87] considers mapping defined on *normed manifolds*, where it is assumed that the metrics used to measure the perturbations are differentiable. We introduce here a summarized view on condition numbers which enables us to handle most of the functions involved in linear algebra.

We assume that the data space $\mathbb{R}^m$ and the solution space $\mathbb{R}^n$ are equipped respectively with the norms $\|.\|_{\mathcal{D}}$ and $\|.\|_{\mathcal{S}}$. We suppose that the solution x corresponding to the data y is defined explicitly by $x = g(y)$, where g maps a neighbourhood of $y_0 \in \mathbb{R}^m$ to $\mathbb{R}^n$.

Definition 1. *The absolute condition number of g at $y_0 \in \mathbb{R}^m$ is the quantity $K(y_0)$ defined by*

$$K(y_0) = \lim_{\delta \to 0} \sup_{0 < \|y_0 - y\|_{\mathcal{D}} \leq \delta} \frac{\|g(y_0) - g(y)\|_{\mathcal{S}}}{\|y_0 - y\|_{\mathcal{D}}}, \tag{4.1}$$

if $\|g(y_0)\|_{\mathcal{S}}$ is nonzero, the relative condition number of g at $y_0 \in \mathbb{R}^m$ is

$$K^{(rel)}(y_0) = K(y_0)\|y_0\|_{\mathcal{D}}/\|g(y_0)\|_{\mathcal{S}}.$$

This definition shows that $K(y_0)$ measures an asymptotic sensitivity and that this quantity depends on the chosen norms for the data and solution spaces. Note that the condition number of a function is either infinite or a real positive number. If g is Lipschitz-continuous in a neighbourhood of y_0, with Lipschitz constant $c \geq 0$, then $K(y_0) \leq c$. It turns out that if g is Fréchet-differentiable in a neighbourhood of y_0, the condition number has a simpler expression. We denote by $|||.|||$ the operator norm induced by the norms $\|.\|_{\mathcal{D}}$ and $\|.\|_{\mathcal{S}}$. For a linear operator $\mathcal{L}$ mapping $\mathbb{R}^m$ to $\mathbb{R}^n$:

$$|||\mathcal{L}||| = \max_{y \neq 0} \frac{\|\mathcal{L}(y)\|_{\mathcal{S}}}{\|y\|_{\mathcal{D}}}.$$

Proposition 1. *[87] If the mapping g is Fréchet-differentiable in a neighbourhood of y_0, the condition number is the norm of the Fréchet derivative : $K(y_0) = |||g'(y_0)|||$.*

The above proposition can be easily generalized to cover the case where the dependence of x in y is expressed using an implicit form $F(x, y) = 0$ with $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$.

Proposition 2. *Suppose that F is continuous in a neighbourhood of (x_0, y_0) such that $F(x_0, y_0) = 0$. Assume that $\partial_x F$ exists in a neighbourhood of (x_0, y_0) and is continuous at (x_0, y_0), and that $\partial_x F(x_0, y_0)$ is a nonsingular linear operator. Then there exist neighbourhoods V_1 and V_2 of x_0 and y_0 respectively, such that, for any $y \in S_2$, the equation $F(x, y)$ has a unique solution $x = g(y)$. If $\partial_y F$ exists at (x_0, y_0), g is Fréchet différentiable at y_0 and $g'(y_0) = -\partial_x F(x_0, y_0)^{-1} \partial_y F(x_0, y_0)$. The condition number of the problem $F(x, y) = 0$ is then*

$$K(y_0) = |||\partial_x F(x_0, y_0)^{-1} \partial_y F(x_0, y_0)|||.$$

In this presentation of the condition number, we measure the perturbation using norms of $\mathbb{R}^m$ and $\mathbb{R}^n$, and the condition numbers obtained in this approach are called *normwise* condition numbers. It is also possible to define condition numbers where the norms are replaced by other functions, called metrics, and that take positive values. For instance, in the *componentwise* metric, the quantity $\|y_0 - y\|_{\mathcal{D}}$ is replaced by

$$d(y_0, y) = \min\{\omega, \ s.t. \ |y_0 - y| \le \omega|y_0|\},$$

and similarly, $\|g(y_0) - g(y)\|_{\mathcal{S}}$ is replaced by

$$d(g(y_0), g(y)) = \min\{\omega, \ s.t. \ |g(y_0) - g(y)| \le \omega|g(y_0)|\},$$

in Definition 1. The resulting quantity is called *componentwise* condition number. We denote by D_g (resp. D_y) the diagonal matrix whose diagonal entries are the entries of $g(y_0)$ (resp. y_0), and if we identify $g'(y_0)$ with the matrix representing this operator in the canonical basis. If all the entries of $g(y_0)$ are nonzero, it can be easily proved that the *componentwise* condition number is $\|D_g^{-1}g'(y_0)D_y\|_\infty$.
We refer to [27] for more details concerning the *componentwise* stability.

4.1.3 Adjoint condition numbers

In a recent paper [53], a technique is presented to compute or estimate condition numbers using adjoint formulae. The results are presented in Banach spaces, and make use of the corresponding duality results. The aim of the present section is to prove that the same results become easy and are helpful when presented in the framework of Euclidean spaces.

Consider a linear mapping $J : E \to G$ where E and G are equipped respectively with norms $\|.\|_E$, $\|.\|_G$ and scalar products $< .,. >_E$ and $< .,. >_G$. We define $J^* : G \to E$ by

$$< y, Jx >_G = < J^*y, x >_E,$$

where $(x, y) \in E \times G$. We define the dual norm $\|.\|_{E*}$ of $\|.\|_E$ by

$$\|x\|_{E*} = \max_{u \ne 0} \frac{< x, u >_E}{\|u\|_E},$$

and define similarly the dual norm $\|.\|_{G*}$. For the linear applications mapping E to G, we denote by $\|.\|_{E,G}$ the operator norm induced by the norms $\|.\|_E$ and $\|.\|_G$. Likewise, the norm $\|.\|_{G*,E*}$ is the operator norm for linear applications mapping G to E and induced by the dual norms $\|.\|_{G*}$ and $\|.\|_{E*}$.
Then

$$
\begin{aligned}
\|J\|_{E,G} &= \max_{x \in E} \frac{\|Jx\|_G}{\|x\|_E} \\
&= \max_{x \in E, u \in G} \frac{< Jx, u >_G}{\|u\|_{G*}\|x\|_E} \quad \text{we use the "duality theorem" [63, p. 287]} \\
&= \max_{u \in G} \frac{1}{\|u\|_{G*}} \max_{x \in E} \frac{< x, J^*u >_E}{\|x\|_E} \\
&= \max_{u \in G} \frac{\|J^*u\|_{E*}}{\|u\|_{G*}} \\
&= \|J^*\|_{G*,E*},
\end{aligned}
$$

which proves the following theorem.

Theorem 4.
$$\|J\|_{E,G} = \|J^*\|_{G*,E*}$$

Let consider the application to the full rank least squares problem where the solution x is expressed by
$$x = g(A,b) = (A^T A)^{-1} A^T b.$$

Using the formula of $g'(A,b)$ given in [51, p. 525], we are interested in the linear function $J = g'(A,b)$ expressed by

$$
\begin{array}{rcl}
J \; : \; \mathbb{R}^{m \times n} \times \mathbb{R}^m & \longrightarrow & \mathbb{R}^n \\
(B,c) & \longmapsto & (A^T A)^{-1} B^T r - A^\dagger B x + A^\dagger c \; , \\
& & = J_1(B) + J_2(c)
\end{array}
\tag{4.2}
$$

where $A^\dagger$ denotes the pseudo inverse of A and $r = b - Ax$ is the residual vector.

We consider the following norms and scalar products :

- on $\mathbb{R}^n$, $<x,y> = x^T y$, and we use the norm $\|x\| = (x^T x)^{1/2}$. Then $\|x\|_* = \|x\|$.

- on $\mathbb{R}^{m \times n} \times \mathbb{R}^m$, we use the scalar product $<(A,b),(B,c)> = \text{trace}(A^T B) + b^T c$, and the product norm $\|(A,b)\|_\nu = \nu(\|A\|_2, \|b\|)$ where ν is an absolute norm on $\mathbb{R}^2$ (i.e such that $\nu(|x|) = \nu(x) \; \forall x \in \mathbb{R}^2$, [72, p. 367]).

Let ν_* be the dual of ν with respect to the canonical inner-product of $\mathbb{R}^2$. Let consider now the dual $\|.\|_{\nu*}$ of the product norm $\|.\|_\nu$ with respect to the scalar product of $\mathbb{R}^{m \times n} \times \mathbb{R}^m$. The following theorem shows how $\|.\|_{\nu*}$ can be expressed as a function of the dual norms on matrices and vectors.

Theorem 5. *The dual of the product norm* $(A,b) \mapsto \|(A,b)\|_\nu$ *is the norm* $(A,b) \mapsto \nu_*(\|A\|_{2*}, \|b\|)$.

This result can be justified as follows. By definition we have

$$\|(A,b)\|_{\nu*} = \max_{\|(B,c)\|_\nu = 1} \text{trace}(A^T B) + b^T c.$$

From $\|A\|_{2*} = \max_{B \neq 0} \frac{\text{trace}(A^T B)}{\|B\|_2}$ it follows that $\forall B \in \mathbb{R}^{m \times n} \; \text{trace}(A^T B) \leq \|A\|_{2*} \|B\|_2$. Since we also have $b^T c \leq \|b\|\|c\|$ then we get

$$\|(A,b)\|_{\nu*} \leq \max_{\|(B,c)\|_\nu=1} \|B\|_2 \|A\|_{2*} + \|c\|\|b\| = \max_{\nu(\|B\|_2,\|c\|)=1} \begin{pmatrix} \|A\|_{2*} \\ \|b\| \end{pmatrix}^T \begin{pmatrix} \|B\|_2 \\ \|c\| \end{pmatrix} = \nu_*(\|A\|_{2*}, \|b\|).$$

This establishes that $\nu_*(\|A\|_{2*}, \|b\|)$ is an upper-bound for the dual of the product norm. Let now consider B_A and c_b such that $\text{trace}(A^T B_A) = \|B_A\|_2 \|A\|_{2*}$ and $b^T c_b = \|c_b\|\|b\|$.

Then $\nu_*(\|A\|_{2*}, \|b\|) = \max_{\nu(\alpha\|B_A\|_2, \beta\|c_b\|)=1} \begin{pmatrix} \|A\|_{2*} \\ \|b\| \end{pmatrix}^T \begin{pmatrix} \alpha\|B_A\|_2 \\ \beta\|c_b\| \end{pmatrix}$ is attained for a particular value (α', β') for which we have

$$
\begin{array}{rcl}
\nu_*(\|A\|_{2*}, \|b\|) & = & \alpha'\|A\|_{2*}\|B_A\|_2 + \beta'\|b\|\|c_b\| \\
& = & \alpha' \, \text{trace}(A^T B_A) + \beta' \, b^T c_b.
\end{array}
$$

Since ν is an absolute norm, we have

$$\|(\alpha' B_A, \beta' c_b)\|_\nu = \nu(|\alpha'| \|B_A\|_2, |\beta'| \|c_b\|) = \nu(\alpha' \|B_A\|_2, \beta' \|c_b\|) = 1$$

and then we get

$$\|(A,b)\|_{\nu*} = \max_{\|(B,c)\|_\nu = 1} \operatorname{trace}(A^T B) + b^T c \geq \operatorname{trace}(A^T \alpha' B_A) + b^T \beta' c_b.$$

Thus $\|(A,b)\|_{\nu*} \geq \nu_*(\|A\|_{2*}, \|b\|)$ and finally we obtain $\|(A,b)\|_{\nu*} = \nu_*(\|A\|_{2*}, \|b\|)$.

Remark 6. Generalization:
Theorem 5 can be easily generalized to a product norm involving p spaces $\mathbb{R}^{m_i \times n_i}$, $i = 1, \cdots, p$. On the product space we consider the scalar product

$$< (A_1, \cdots, A_p), (B_1, \cdots, B_p) >= \operatorname{trace}(A_1{}^T B_1) + \cdots + \operatorname{trace}(A_p{}^T B_p),$$

and the product norm

$$\|(A_1, \cdots, A_p)\|_\nu = \nu(\|A_1\|_2, \cdots, \|A_p\|_2)$$

where ν is an absolute norm on $\mathbb{R}^p$. Then we have

$$\|(A_1, \cdots, A_p)\|_{\nu*} = \nu_*(\|A_1\|_{2*}, \cdots, \|A_p\|_{2*}).$$

Lemma 3.5 of [90, p. 78] shows exactly that $\|A\|_{2*} = \|\sigma(A)\|_1$, and since $\operatorname{trace}(A^T A) = \|A\|_F^2$, $\|A\|_{F*} = \|A\|_F$.
Using (4.7), we obtain for the first part of the adjoint of the derivative J,

$$
\begin{aligned}
\forall u \in \mathbb{R}^n < u, J_1 B > \;&=\; u^T((A^T A)^{-1} B^T r - A^\dagger B x) \\
&=\; \operatorname{trace}((A^T A)^{-1} B^T r u^T) - \operatorname{trace}(A^\dagger B x u^T) \\
&=\; \operatorname{trace}(r u^T (A^T A)^{-1} B^T) - \operatorname{trace}(x u^T A^\dagger B) \\
&=\; \operatorname{trace}((r u^T (A^T A)^{-1})^T B) - \operatorname{trace}(x u^T A^\dagger B) \\
&=\; \operatorname{trace}(((r u^T (A^T A)^{-1})^T - x u^T A^\dagger) B) \\
&=\; < r u^T (A^T A)^{-1} - A^{\dagger T} u x^T, B > \\
&=\; < J_1{}^* u, B > .
\end{aligned}
$$

For the second part of the adjoint of the derivative J, we have

$$
\begin{aligned}
\forall u \in \mathbb{R}^n < u, J_2 c > \;&=\; u^T A^\dagger c \\
&=\; < A^{\dagger T} u, c > \\
&=\; < J_2{}^* u, c > .
\end{aligned}
$$

We therefore have the following results :

Theorem 6. *The adjoint of J, Fréchet derivative of the full rank least squares solution,*

$$
\begin{array}{rlc}
J \;:\; \mathbb{R}^{m \times n} \times \mathbb{R}^m & \longrightarrow & \mathbb{R}^n \\
(B, c) & \longmapsto & (A^T A)^{-1} B^T r - A^\dagger B x + A^\dagger c
\end{array}
\tag{4.3}
$$

is

$$
\begin{array}{rlc}
J^* \;:\; \mathbb{R}^n & \longrightarrow & \mathbb{R}^{m \times n} \times \mathbb{R}^m \\
u & \longmapsto & (r u^T (A^T A)^{-1} - A^{\dagger T} u x^T, A^{\dagger T} u)
\end{array}
\cdot
\tag{4.4}
$$

As mentioned in [53] the main advantage of working with the adjoint J^* is that the operator norm computation, involved in the condition number definition, implies a maximization over a vector space of dimension n, instead of a maximization over a vector space of dimension $mn + m$ for J. Indeed we have, with the notation of Section 4.1.2

$$K(A, b) = \max_{\|(B,c)\|_\nu = 1} \|J(B, c)\| = \max_{\|u\| = 1} \|J^*(u)\|_{\nu*}.$$

Note that when we consider the Frobenius norm on matrices i.e when the product norm is $\nu(\|A\|_F, \|b\|)$, we get

$$\|(A, b)\|_{\nu*} = \nu_*(\|A\|_F, \|b\|).$$

We will see in Section 4.4 that this idea will be especially attractive in the case where the operator norm is directly computed using statistical methods based on sampling.

4.1.4 Backward error

Let us consider, as in Section 4.1.2, a mathematical problem where the dependence between the solution x and the data y can be expressed using an implicit form $F(x, y) = 0$. Assume we have an approximate solution $\tilde{x}$, a backward error aims to measure the distance between the data y of the initial problem and the set of all data for which $\tilde{x}$ is the solution of the perturbed problem, i.e $\{\tilde{y} : F(\tilde{x}, \tilde{y}) = 0\}$.

Definition 2. *The (normwise) backward error in the meaning of the norm $\|.\|$ is expressed by*

$$\eta(\tilde{x}) = \inf\{\|\Delta y\| : F(\tilde{x}, y + \Delta y) = 0\}.$$

Note that the quantity $\eta(\tilde{x})$ is sometimes called *optimal* backward error in the literature. We may notice that in this definition there is no reference made to the exact solution for the original data y.

Different situations may occur. If there does not exist a perturbation Δy such that $F(\tilde{x}, y + \Delta y) = 0$ then we set by convention $\eta(\tilde{x}) = +\infty$. If there exists Δy_0 such that $F(\tilde{x}, y + \Delta y_0) = 0$ and the partial function $F(\tilde{x}, .)$ is continuous, then the set $\{\Delta y : F(\tilde{x}, y + \Delta y) = 0$ and $\|\Delta y\| \leq \|\Delta y_0\|\}$ is a compact set. Then the existence of $\eta(\tilde{x}) =$ is guaranteed and a Δy that achieves $\eta(\tilde{x})$ is called an optimal perturbation.

$\eta(\tilde{x})$ is useful for measuring the quality of the solution in the sense that if it is smaller than the errors in the definition of the problem (e.g discretization or measurement errors), then $\tilde{x}$ can be considered as solving the problem. Conversely if $\eta(\tilde{x})$ is greater than these uncertainties, then $\tilde{x}$ must be rejected.

We give here some fundamental results for the backward error analysis. The first one is related to linear systems and was established by [88]:

Theorem 7. *If $\|.\|$ denotes any vector norm and the corresponding subordinate matrix norm, then the (relative) backward error*

$$\eta(\tilde{x}) = \min\{\epsilon \; : \; (A + \Delta A)\tilde{x} = b + \Delta b, \; \|\Delta A\| \leq \epsilon\|A\|, \; \|\Delta b\| \leq \epsilon\|b\|\}$$

is given by

$$\eta(\tilde{x}) = \frac{\|r\|}{\|A\|\|\tilde{x}\| + \|\Delta b\|}$$

70

where $r = b - A\tilde{x}$. Optimal perturbations are

$$\Delta A = \frac{\|A\|\|\tilde{x}\|}{\|A\|\|\tilde{x}\| + \|b\|} r z^T \ , \ \Delta b = -\frac{\|b\|}{\|A\|\|\tilde{x}\| + \|b\|} r$$

where z is a vector such that $z^T \tilde{x} = \|\tilde{x}\| \|z\|_* = 1$ (i.e z is the dual vector of $\tilde{x}$).

Note that in the case of linear systems, the backward error is inexpensive to compute when the norms are easily computed.
Concerning the LLSP, an exact value for the backward error with respect to the Frobenius norm was established by [100]:

Theorem 8. *Let* $A \in \mathbb{R}^{m \times n}$ ($m \geq n$) *and* $b \in \mathbb{R}^m$ *and* $r = b - A\tilde{x}$. *The backward error*

$$\eta(\tilde{x}) = \min\{\|[\Delta A \ \ \theta \Delta b]\|_F \ : \ \|(A + \Delta A)\tilde{x} - (b + \Delta b)\|_2 = \min\}$$

is given by

$$\eta(\tilde{x}) = \left(\frac{\|r\|_2^2}{\|\tilde{x}\|_2} \mu + \min\{0, \lambda_*\} \right)^{\frac{1}{2}}, \tag{4.5}$$

where $\lambda_* = \lambda_{min}\{AA^T - \mu \frac{rr^T}{\|\tilde{x}\|_2^2}\}$ *and* $\mu = \frac{\theta^2 \|\tilde{x}\|_2^2}{1 + \theta^2 \|\tilde{x}\|_2^2}$.

As mentioned in [35], the parameter θ enables us to monitor perturbations on A and b. For instance taking $\theta = \infty$ (resp. $\theta = 0$) allows us to perturb A (resp. b) only. Also $\theta = \frac{\|A\|_F}{\|b\|_2}$ produces a relative backward error.
Formula (4.5) involves the difference of two potentially very small quantities when $\lambda_* < 0$. In this respect its numerical evaluation may be unstable. Using the fact that, for the case $\lambda_* < 0$, we have

$$\left(\frac{\|r\|_2^2}{\|\tilde{x}\|_2} \mu + \lambda_* \right)^{\frac{1}{2}} = \sigma_{min}([A \ R]), \ R = \sqrt{\mu} \frac{\|r\|_2}{\|\tilde{x}\|_2} (I - rr^\dagger)$$

where σ_{min} denotes the smallest singular value and the pseudo-inverse of a nonzero vector x corresponds to $x^\dagger = x^T/(x^T x)$, [100] gives another expression for the backward error that is more suitable for calculation:

Corollary 1.
$$\eta(\tilde{x}) = \min\{\phi, \sigma_{min}([A \ \phi(I - rr^\dagger)])\}, \ \phi = \sqrt{\mu} \frac{\|r\|_2}{\|\tilde{x}\|_2}.$$

This modified formula requires the SVD of an $m \times (n + m)$ matrix, which can be very expensive in practice. This is why an estimate has been proposed in [52, 54, 67] for $\eta(\tilde{x})$ when only A is perturbed. This estimate can be written

$$\tilde{\eta}(\tilde{x}) = \left\| \left(\|\tilde{x}\|_2^2 A^T A + \|r\|_2^2 I \right)^{-1/2} A^T r \right\|_F. \tag{4.6}$$

71

We can find in [91] a synthesis of how this estimate has been studied.
First it comes from [67] that

$$\frac{\tilde{\eta}(\tilde{x})}{\eta(\tilde{x})} \leq \frac{2 + \sqrt{2}}{2} \approx 1.707.$$

Then it has been shown in [54] that

$$\frac{\|r_*\|_2}{\|r\|_2} \leq \frac{\tilde{\eta}(\tilde{x})}{\eta(\tilde{x})} \leq \frac{1 + \sqrt{5}}{2} \approx 1.618,$$

where $r_* = b - Ax$ is the true residual of the LLSP and assumed that A has full column rank. Finally, [52] established that $\tilde{\eta}(\tilde{x})$ asymptotically equals $\eta(\tilde{x})$ i.e that

$$\lim_{\tilde{x} \to x} \frac{\tilde{\eta}(\tilde{x})}{\eta(\tilde{x})} = 1,$$

where x is any solution of the LLSP and assumed that A, r_* and x are not zero.
We point out that an inequality of the type $\tilde{\eta}(\tilde{x}) \leq 2\eta(\tilde{x})$ is ideal for detecting instability if $\tilde{\eta}(\tilde{x})$ is too large. But if $\tilde{\eta}(\tilde{x})$ is small, it cannot be generally inferred that $\eta(\tilde{x})$ is small.
The computation of $\tilde{\eta}(\tilde{x})$ requires less computational effort than the computation of $\eta(\tilde{x})$ using Corollary 1. As mentioned in [91], the computational cost of $\tilde{\eta}(\tilde{x})$ using (4.6) is roughly the same number of floating-point operations as a SVD of A i.e $\mathcal{O}(mn^2)$. Also, [76] has proposed an algorithm based on the Lanczos bidiagonalization that approximates the backward perturbation bound for large sparse LLSP and requires $\mathcal{O}((m+n)l)$ operations with $l \ll \min(m, n)$.

4.2 Partial condition number

4.2.1 Motivation

In the rest of this chapter we consider the problem of calculating the quantity $L^T x$, where x is the solution of the LLSP $\min_{x \in \mathbb{R}^n} \|Ax - b\|_2$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$ is a matrix of full column rank n and $L \in \mathbb{R}^{n \times k}$ $(k \leq n)$. This estimation is a fundamental problem of parameter estimation in the framework of the Gauss-Markov Model [85, p. 137]. More precisely, we focus here on the evaluation of the sensitivity of $L^T x$ to small perturbations of the matrix A and/or the right-hand side b.
The interest for this question stems for instance from parameter estimation where the parameters of the model can often be divided into two parts : the variables of physical significance and a set of ancillary variables involved in the models. For example, this situation occurs in the determination of positions using the GPS system, where the 3-D coordinates are the quantities of interest but the statistical model involves other parameters such as clock drift and GPS ambiguities [66] that are generally estimated during the solution process. It is then crucial to ensure that the solution components of interest can be computed with satisfactory accuracy. The main goal of this chapter is to formalize this problem in terms of a condition number and to describe practical methods to compute or estimate this quantity. Note that as far as the sensitivity of a subset of the solution

components is concerned, the matrix L is a projection whose columns consist of vectors of the canonical basis of $\mathbb{R}^n$.

In the special case where L is the identity matrix, we can summarize (similarly to [53]) the formulae that evaluate the condition number of an LLSP in Table 4.1 where $\sigma_{\min} = 1/\left\|A^\dagger\right\|_2$ is the smallest singular value of A. We notice that these expressions are closed (in Frobenius norm) or of correct order of magnitude (in spectral norm). Note also the dependence in $\sigma_{\min}^2$ when $\|r\|_2 = \|b - Ax\|_2$ is large.

Table 4.1: Condition number expressions for the full rank LLSP.

source	data	solution	formula	status
Björck 96	$\frac{\|\delta A\|_2}{\|A\|_2}$	$\frac{\|\delta x\|_2}{\|x\|_2}$	$\frac{\|A\|_2\|r\|_2}{\sigma_{\min}^2\|x\|_2} + \frac{\|A\|_2}{\sigma_{\min}}$	estimate
Geurts 82	$\frac{\|\delta A\|_F}{\|A\|_F}$	$\frac{\|\delta x\|_2}{\|x\|_2}$	$\frac{\|A\|_F}{\sigma_{\min}\|x\|_2}\sqrt{\frac{\|r\|_2^2}{\sigma_{\min}^2} + \|x\|_2^2}$	exact
Gratton 96	$\sqrt{\frac{\|\delta A\|_F^2}{\|A\|_F^2} + \frac{\|\delta b\|_F^2}{\|b\|_F^2}}$	$\frac{\|\delta x\|_2}{\|x\|_2}$	$\frac{1}{\sigma_{\min}\|x\|_2}\sqrt{\frac{\|A\|_F^2\|r\|_2^2}{\sigma_{\min}^2} + \|A\|_F^2\,\|x\|_2^2 + \|b\|_2^2}$	exact
Grcar 04	$\max\left\{\frac{\|\delta A\|_{F \text{ or } 2}}{\|A\|_{F \text{ or } 2}}, \frac{\|\delta b\|_2}{\|b\|_2}\right\}$	$\frac{\|\delta x\|_2}{\|x\|_2}$	$\frac{\|A\|_{F \text{ or } 2}\|r\|_2}{\sigma_{\min}^2\|x\|_2} + \frac{\|A\|_{F \text{ or } 2}}{\sigma_{\min}} + \frac{\|b\|_2}{\sigma_{\min}\|x\|_2}$	estimate

If we consider the norms $\|(A, b)\|_F = \sqrt{\|A\|_F^2 + \|b\|_2^2}$ for the data space and $\|x\|_2$ for the solution space, then [51] gives an explicit formula for the relative condition number $K^{(rel)}(A, b)$:

$$K^{(rel)}(A, b) = \left\|A^\dagger\right\|_2 \left(\left\|A^\dagger\right\|_2^2 \|r\|_2^2 + \|x\|_2^2 + 1\right)^{\frac{1}{2}} \frac{\|(A, b)\|_F}{\|x\|_2}.$$

But does the value of $K^{(rel)}(A, b)$ give us useful information about the sensitivity of $L^T x$? Can it in some cases overestimate the error in components or on the contrary be too optimistic?
Let us consider the following example.

$$A = \begin{pmatrix} 1 & 1 & \epsilon^2 \\ \epsilon & 0 & \epsilon^2 \\ 0 & \epsilon & \epsilon^2 \\ \epsilon^2 & \epsilon^2 & 2 \end{pmatrix}, \quad x = \begin{pmatrix} \epsilon \\ \epsilon \\ \frac{1}{\epsilon} \end{pmatrix} \text{ and } b = \begin{pmatrix} 3\epsilon \\ \epsilon^2 + \epsilon \\ \epsilon^2 + \epsilon \\ 2\epsilon^3 + \frac{2}{\epsilon} \end{pmatrix},$$

x is here the exact solution of the LLSP $\min_{x \in \mathbb{R}^3} \|Ax - b\|_2$. If we take $\epsilon = 10^{-8}$ then we have $x = (10^{-8}, 10^{-8}, 10^8)^T$ and the solution computed in Matlab [94] using a machine precision $2.22 \cdot 10^{-16}$ is $\tilde{x} = (1.5 \cdot 10^{-8}, 1.5 \cdot 10^{-8}, 10^8)^T$. The LLSP condition number is $K^{(rel)}(A, b) = 2.4 \cdot 10^8$ and the relative errors on the components of x are

$$\frac{|x_1 - \tilde{x}_1|}{|x_1|} = \frac{|x_2 - \tilde{x}_2|}{|x_2|} = 0.5 \text{ and } \frac{|x_3 - \tilde{x}_3|}{|x_3|} = 0.$$

Then, if $L = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$, we expect a large value for the condition number of $L^T x$ because there is a 50% relative error on x_1 and x_2. If now $L = (0, 0, 1)^T$, then we expect that

the condition number of $L^T x$ would be close to 1 because $\tilde{x}_3 = x_3$. For these two values of L, the LLSP condition number is far from giving a good idea of the sensitivity of $L^T x$. Note in this case that the perturbations are due to roundoff errors.

Let us now consider a simple example in the framework of parameter estimation where in addition to roundoff errors, random errors are involved. Let $b = \{b_i\}_{i=1,\cdots,10}$ be a series of observed values depending on data $s = \{s_i\}$ where $s_i = 10 + i, i = 1, \cdots, 10$. We determine a 3-degree polynomial that approximates b in the least squares sense, and we suppose that the following relationship holds

$$b = x_1 + x_2 \frac{1}{s} + x_3 \frac{1}{s^2} + x_4 \frac{1}{s^3} \text{ with } x_1 = x_2 = x_3 = x_4 = 1.$$

We assume that the perturbation on each b_i is 10^{-8} multiplied by a normally distributed random number and denote by $\tilde{b} = \{\tilde{b}_i\}_{i=1,\cdots,10}$ the perturbed quantity. This corresponds to the LLSP $\min_{x \in \mathbb{R}^4} \|Ax - \tilde{b}\|_2$ where A is the Vandermonde matrix defined by $A_{ij} = \frac{1}{s_i^{j-1}}$. Let $\tilde{x}$ and $\tilde{y}$ be the computed solutions corresponding to two perturbed right-hand sides. Then we obtain the following relative errors on each component:

$$\frac{|\tilde{x}_1 - \tilde{y}_1|}{|\tilde{x}_1|} = 2 \cdot 10^{-7}, \frac{|\tilde{x}_2 - \tilde{y}_2|}{|\tilde{x}_2|} = 6 \cdot 10^{-6}, \frac{|\tilde{x}_3 - \tilde{y}_3|}{|\tilde{x}_3|} = 6 \cdot 10^{-5}, \text{ and } \frac{|\tilde{x}_4 - \tilde{y}_4|}{|\tilde{x}_4|} = 10^{-4}.$$

We have $K^{(rel)}(A, b) = 3.1 \cdot 10^5$. Regarding the disparity between the sensitivity of each component, we need a quantity that evaluates more precisely the sensitivity of each solution component of the LLSP.

The idea of analyzing the accuracy of some solution components in linear algebra is by no means new. For linear systems $Ax = b$, $A \in \mathbb{R}^n$ and for LLSP, [29] defines so called componentwise condition numbers that correspond to amplification factors of the relative errors in solution components due to perturbations of data A or b and explains how to estimate them. In our formalism, these quantities are upper bounds of the condition number of $L^T x$ where L is a column of the identity matrix. We also emphasize that the term "componentwise" refers here to the solution components and must be distinguished from the metric used for matrices and for which [103] provides a condition number for generalized inversion and linear least squares.

For LLSP, [70] provides a statistical estimate for componentwise condition numbers due to either relative or structured perturbations. In the case of linear systems, [24] proposes a statistical approach, based on [69] that enables us to compute the condition number of $L^T x$ in $\mathcal{O}(n^2)$.

Our approach differs from the previous studies in the following aspects:

1. we are interested in the condition of $L^T x$ where L is a general matrix and not only a canonical vector of $\mathbb{R}^n$,
2. we are looking for a condition number based on the Fréchet-derivative, and not only for an upper bound of this quantity.

We present in this chapter three ways to obtain information on the condition of $L^T x$. The first one uses an explicit formula based on the singular value decomposition of A. The second is at the same time an upper bound of this condition number and an estimate of correct order of magnitude. The third method supplies a statistical estimate. The choice

between these three methods will depend on the size of the problem (computational cost) and on the accuracy desired for this quantity.

The sequel of this chapter is organized as follows. In Section 4.2.2, we define the notion of a partial condition number. Then, when perturbations on A are measured using a Frobenius norm, we give a closed formula for this condition number in the general case where $L \in \mathbb{R}^{n \times k}$ and in the particular case when $L \in \mathbb{R}^n$. In Section 4.3, we establish bounds of the partial condition number in Frobenius as well as in spectral norm, and we show that these bounds can be considered as estimates of it. In Section 4.4, we describe a statistical method that enables us to estimate the partial condition number. In Section 4.5, we present numerical results in order to compare the statistical estimate and the exact condition number on sample matrices A and L. In Section 4.6, we give a summary comparing the three ways to compute the condition of $L^T x$ as well as a numerical illustration. Finally, some concluding remarks are given in Section 4.7.

We use the following notations. The matrix I is the identity matrix and e_i is the i-th canonical vector. We also denote by $\text{Im}(A)$ the space spanned by the columns of A and by $\text{Ker}(A)$ the null space of A.

4.2.2   A closed formula for the partial condition number of an LLSP

Let L be an $n \times k$ matrix, with $k \leq n$. We consider the function

$$
\begin{array}{rccc}
g : & \mathbb{R}^{m \times n} \times \mathbb{R}^m & \longrightarrow & \mathbb{R}^k \\
& A, b & \longmapsto & g(A, b) = L^T x(A, b) = L^T (A^T A)^{-1} A^T b.
\end{array}
\tag{4.7}
$$

Since A has full rank n, g is continuously F-differentiable in a neighbourhood of (A, b) and we denote by g' its F-derivative. Let α and β be two positive real numbers. In the present chapter we consider the Euclidean norm for the solution space $\mathbb{R}^k$. For the data space $\mathbb{R}^{m \times n} \times \mathbb{R}^m$, we use the product norms defined by

$$
\|(A, b)\|_F = \sqrt{\alpha^2 \|A\|_F^2 + \beta^2 \|b\|_2^2}, \ \alpha, \beta > 0
$$

and

$$
\|(A, b)\|_2 = \sqrt{\alpha^2 \|A\|_2^2 + \beta^2 \|b\|_2^2}, \ \alpha, \beta > 0.
$$

These norms are very flexible since they allow us to monitor the perturbations on A and b. For instance, large values of α (resp. β) enable us to obtain condition number problems where mainly b (resp. A) are perturbed. A more general weighted Frobenius norm $\|(AT, \beta b)\|_F$, where T is a positive diagonal matrix is sometimes chosen. This is for instance the case in [102] who give an explicit expression for the condition number of rank deficient linear least squares using this norm.

According to Geurts [46], the absolute condition numbers of g at the point (A, b) using the two product norms defined above is given by:

$$
\kappa_{g,F}(A, b) = \max_{(\Delta A, \Delta b)} \frac{\|g'(A, b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_F}
$$

and

$$
\kappa_{g,2}(A, b) = \max_{(\Delta A, \Delta b)} \frac{\|g'(A, b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_2}.
$$

The corresponding relative condition numbers of g at (A, b) are expressed by

$$\kappa_{g,F}^{(rel)}(A, b) = \frac{\kappa_{g,F}(A, b) \; \|(A, b)\|_F}{\|g(A, b)\|_2}$$

and

$$\kappa_{g,2}^{(rel)}(A, b) = \frac{\kappa_{g,2}(A, b) \; \|(A, b)\|_2}{\|g(A, b)\|_2}.$$

We call the condition numbers related to $L^T x(A, b)$ *partial condition numbers* of the LLSP with respect to the linear operator L. The partial condition number defined using the product norm $\|(.,.)\|_F$ is given by the following theorem.

Theorem 9. *Let $A = U\Sigma V^T$ be the thin singular value decomposition of A defined in [50] with $\Sigma = diag(\sigma_i)$ and $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_n > 0$. The absolute condition number of $g(A, b) = L^T x(A, b)$ is given by*

$$\kappa_{g,F}(A, b) = \left\|SV^T L\right\|_2$$

where $S \in \mathbb{R}^{n \times n}$ is the diagonal matrix with diagonal elements $S_{ii} = \sigma_i^{-1}\sqrt{\frac{\sigma_i^{-2}\|r\|_2^2 + \|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}}$.

Proof. The demonstration is divided into three parts. In Part 1, we establish an explicit formula of $g'(A, b).(\Delta A, \Delta b)$. In Part 2, we derive an upper bound for $\frac{\|g'(A,b).(\Delta A,\Delta b)\|_2}{\|(\Delta A,\Delta b)\|_F}$. In Part 3, we show that this bound is reached for a particular $(\Delta A, \Delta b)$.

Part 1:

Let $\Delta A \in \mathbb{R}^{m \times n}$ and $\Delta b \in \mathbb{R}^m$. Using the chain rules of composition of derivatives, we get

$$g'(A, b).(\Delta A, \Delta b) =$$
$$L^T (A^T A)^{-1} \Delta A^T (b - A(A^T A)^{-1} A^T b) - L^T (A^T A)^{-1} A^T \Delta A (A^T A)^{-1} A^T b + L^T A^\dagger \Delta b$$

i.e

$$g'(A, b).(\Delta A, \Delta b) = L^T (A^T A)^{-1} \Delta A^T r - L^T A^\dagger \Delta A x + L^T A^\dagger \Delta b. \qquad (4.8)$$

We write $\Delta A = \Delta A_1 + \Delta A_2$ by defining $\Delta A_1 = AA^\dagger \Delta A$ (projection of ΔA on Im(A)) and $\Delta A_2 = (I - AA^\dagger)\Delta A$ (projection of ΔA on Im $(A)^\perp$). We have $\Delta A_1^T r = 0$ (because $r \in \text{Im}(A)^\perp$) and $A^\dagger \Delta A_2 = 0$. Then we obtain

$$g'(A, b).(\Delta A, \Delta b) = L^T (A^T A)^{-1} \Delta A_2^T r - L^T A^\dagger \Delta A_1 x + L^T A^\dagger \Delta b. \qquad (4.9)$$

Part 2:

We now prove that $\kappa_{g,F}(A, b) \leq \left\|SV^T L\right\|_2$. Let u_i and v_i be the i-th column of respectively U and V.

From $A^\dagger = V\Sigma^{-1}U^T$, we get $AA^\dagger = UU^T = \sum_{i=1}^n u_i u_i^T$ and since $\sum_{i=1}^n v_i v_i^T = I$, we have $\Delta A_1 = \sum_{i=1}^n u_i u_i^T \Delta A$ and $\Delta A_2 = (I - AA^\dagger)\Delta A \sum_{i=1}^n v_i v_i^T$. Moreover, still using the thin SVD of A and $A^\dagger$, it follows that

$$(A^T A)^{-1} v_i = \frac{v_i}{\sigma_i^2}, \quad A^\dagger u_i = \frac{v_i}{\sigma_i} \text{ and } A^\dagger \Delta b = \sum_{i=1}^n v_i u_i^T \frac{\Delta b}{\sigma_i}. \qquad (4.10)$$

76

Thus (4.9) becomes

$$g'(A,b).(\Delta A, \Delta b) = \sum_{i=1}^{n} L^T v_i \left[v_i^T \Delta A^T (I - AA^\dagger)\frac{r}{\sigma_i^2} - u_i^T \Delta A \frac{x}{\sigma_i} + u_i^T \frac{\Delta b}{\sigma_i} \right]$$

$$= L^T \sum_{i=1}^{n} v_i y_i,$$

where we set $y_i = v_i^T \Delta A^T (I - AA^\dagger)\frac{r}{\sigma_i^2} - u_i^T \Delta A \frac{x}{\sigma_i} + u_i^T \frac{\Delta b}{\sigma_i} \in \mathbb{R}$.

Thus if $Y = (y_1, y_2, \cdots, y_n)^T$, we get $\|g'(A,b).(\Delta A, \Delta b)\|_2 = \|L^T V Y\|_2$ and then

$$\|g'(A,b).(\Delta A, \Delta b)\|_2 = \|L^T V S S^{-1} Y\|_2 \leq \|S V^T L\|_2 \|S^{-1} Y\|_2.$$

We denote by $w_i = \frac{v_i^T \Delta A^T (I - AA^\dagger)r}{S_{ii}\sigma_i^2} - \frac{u_i^T \Delta A x}{S_{ii}\sigma_i} + \frac{u_i^T \Delta b}{S_{ii}\sigma_i}$ the i-th component of $S^{-1}Y$. Then we have

$$|w_i| \leq \alpha \left\| v_i^T \Delta A^T (I - AA^\dagger)^T \right\|_2 \frac{\|r\|_2}{\alpha S_{ii}\sigma_i^2} + \alpha \left\| u_i^T \Delta A \right\|_2 \frac{\|x\|_2}{\alpha S_{ii}\sigma_i} + \beta |u_i^T \Delta b| \frac{1}{\beta S_{ii}\sigma_i}$$

$$\leq \left(\frac{\|r\|_2^2}{\alpha^2 S_{ii}^2 \sigma_i^4} + \frac{\|x\|_2^2}{\alpha^2 S_{ii}^2 \sigma_i^2} + \frac{1}{\beta^2 S_{ii}^2 \sigma_i^2} \right)^{\frac{1}{2}} \left(\alpha^2 \left\| (I - AA^\dagger)\Delta A v_i \right\|_2^2 + \alpha^2 \left\| u_i^T \Delta A \right\|_2^2 + \beta^2 |u_i^T \Delta b|^2 \right)^{\frac{1}{2}}$$

$$= \frac{S_{ii}}{S_{ii}} \left(\alpha^2 \left\| (I - AA^\dagger)\Delta A v_i \right\|_2^2 + \alpha^2 \left\| u_i^T \Delta A \right\|_2^2 + \beta^2 |u_i^T \Delta b|^2 \right)^{\frac{1}{2}}.$$

Hence

$$\|S^{-1}Y\|_2^2 \leq \sum_{i=1}^{n} \alpha^2 \left\| (I - AA^\dagger)\Delta A v_i \right\|_2^2 + \alpha^2 \left\| u_i^T \Delta A \right\|_2^2 + \beta^2 |u_i^T \Delta b|^2$$

$$= \alpha^2 \left\| (I - AA^\dagger)\Delta A V \right\|_F^2 + \alpha^2 \|U^T \Delta A\|_F^2 + \beta^2 \|U^T \Delta b\|_2^2$$

$$= \alpha^2 \left\| (I - AA^\dagger)\Delta A \right\|_F^2 + \alpha^2 \|U^T \Delta A\|_F^2 + \beta^2 \|U^T \Delta b\|_2^2.$$

Since $\|U^T \Delta A\|_F = \|UU^T \Delta A\|_F = \|AA^\dagger \Delta A\|_F$ and $\|U^T \Delta b\|_2 = \|UU^T \Delta b\|_2 \leq \|\Delta b\|_2$, we get

$$\|S^{-1}Y\|_2^2 \leq \alpha^2 \|\Delta A_1\|_F^2 + \alpha^2 \|\Delta A_2\|_F^2 + \beta^2 \|\Delta b\|_2^2.$$

From $\|\Delta A\|_F^2 = \|\Delta A_1\|_F^2 + \|\Delta A_2\|_F^2$, we get $\|S^{-1}Y\|_2^2 \leq \|(\Delta A, \Delta b)\|_F^2$ and thus

$$\|g'(A,b).(\Delta A, \Delta b)\|_2 \leq \|S V^T L\|_2 \|(\Delta A, \Delta b)\|_F.$$

So we have shown that $\|S V^T L\|_2$ is an upper bound for $\kappa_{g,F}(A,b)$.

Part 3:

We now prove that this upper bound can be reached i.e that $\|S V^T L\|_2 = \frac{\|g'(A,b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_F}$ holds for some $(\Delta A, \Delta b) \in \mathbb{R}^{m \times n} \times \mathbb{R}^m$.

Let consider the particular choice of $(\Delta A, \Delta b)$ defined by

$$(\Delta A, \Delta b) = (\Delta A_2 + \Delta A_1, \Delta b) = \left(\sum_{i=1}^{n} \frac{\alpha_i}{\alpha} \frac{r}{\|r\|_2} v_i^T + \sum_{i=1}^{n} \frac{\beta_i}{\alpha} u_i \frac{x^T}{\|x\|_2}, \sum_{i=1}^{n} \frac{\gamma_i}{\beta} u_i \right)$$

77

where α_i, β_i, γ_i are real constants to be chosen in order to achieve the upper bound obtained in Part 2.

Since $\Delta A_1^T r = 0$ and $A^\dagger \Delta A_2 = 0$, it follows from (4.9) and (4.10) that

$$
\begin{aligned}
g'(A,b).(\Delta A, \Delta b) &= L^T (A^T A)^{-1} \sum_{i=1}^n \frac{\alpha_i}{\alpha} \|r\|_2 v_i^T - L^T A^\dagger \sum_{i=1}^n \frac{\beta_i}{\alpha} u_i \|x\|_2 + L^T A^\dagger \sum_{i=1}^n \frac{\gamma_i}{\beta} u_i \\
&= L^T \sum_{i=1}^n \frac{\alpha_i}{\alpha \sigma_i^2} v_i \|r\|_2 - L^T \sum_{i=1}^n \frac{\beta_i}{\alpha \sigma_i} v_i \|x\|_2 + L^T \sum_{i=1}^n \frac{\gamma_i}{\beta \sigma_i} v_i \\
&= \sum_{i=1}^n L^T v_i \left(\frac{\alpha_i}{\alpha \sigma_i^2} \|r\|_2 - \frac{\beta_i}{\alpha \sigma_i} \|x\|_2 + \frac{\gamma_i}{\beta \sigma_i} \right).
\end{aligned}
$$

Thus by denoting $\xi_i = [L^T v_i \frac{\|r\|_2}{\alpha \sigma_i^2}, -L^T v_i \frac{\|x\|_2}{\alpha \sigma_i}, \frac{L^T v_i}{\beta \sigma_i}] \in \mathbb{R}^{k \times 3}$ and $\Gamma = [\xi_1, \ldots, \xi_n] \in \mathbb{R}^{k \times 3n}$, and $X = (\alpha_1, \beta_1, \gamma_1, \cdots, \alpha_n, \beta_n, \gamma_n)^T \in \mathbb{R}^{3n \times 1}$ we get

$$
g'(A,b).(\Delta A, \Delta b) = \Gamma X. \tag{4.11}
$$

Since $\forall i,j$ $\text{trace}\left((\frac{r}{\|r\|_2} v_i^T)^T (\frac{r}{\|r\|_2} v_i^T) \right) = \text{trace}\left((u_i \frac{x^T}{\|x\|_2})^T (u_i \frac{x^T}{\|x\|_2}) \right) = \delta_{ij}$ where δ_{ij} is the Kronecker symbol and $\text{trace}\left((\frac{r}{\|r\|_2} v_i^T)^T (u_i \frac{x^T}{\|x\|_2}) \right) = 0$, then $\{ \frac{r}{\|r\|_2} v_i^T \}_{i=1,\cdots,n}$ and $\{ u_i \frac{x^T}{\|x\|_2} \}_{i=1,\cdots,n}$ form an orthonormal set of matrices for the Frobenius norm and we get $\|\Delta A\|_F = \sum_{i=1}^n (\alpha_i^2 + \beta_i^2)$. It follows that

$$
\|(\Delta A, \Delta b)\|_F^2 = \sum_{i=1}^n \alpha_i^2 + \sum_{i=1}^n \beta_i^2 + \sum_{i=1}^n \gamma_i^2 = \|X\|_2^2,
$$

and Equation (4.11) yields

$$
\frac{\|g'(A,b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_F} = \frac{\|\Gamma X\|_2}{\|X\|_2}.
$$

We know that $\|\Gamma\|_2 = \max_X \frac{\|\Gamma X\|_2}{\|X\|_2}$ is reached for some $X = (\alpha_1, \beta_1, \gamma_1, \cdots, \alpha_n, \beta_n, \gamma_n)^T$. Then for the $(\Delta A, \Delta b)$ corresponding to this X, we have $\frac{\|g'(A,b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_F} = \|\Gamma\|_2$. Furthermore we have

$$
\begin{aligned}
\Gamma \Gamma^T &= L^T v_1 \left(\frac{\|r\|_2^2}{\alpha^2 \sigma_1^4} + \frac{\|x\|_2^2}{\alpha^2 \sigma_1^2} + \frac{1}{\beta^2 \sigma_1^2} \right) v_1^T L + \cdots + L^T v_n \left(\frac{\|r\|_2^2}{\alpha^2 \sigma_n^4} + \frac{\|x\|_2^2}{\alpha^2 \sigma_n^2} + \frac{1}{\beta^2 \sigma_n^2} \right) v_n^T L \\
&= L^T v_1 S_{11}^2 v_1^T L + \cdots + L^T v_n S_{nn}^2 v_n^T L \\
&= (L^T V S)(S V^T L).
\end{aligned}
$$

Hence

$$
\|\Gamma\|_2 = \sqrt{\|\Gamma \Gamma^T\|_2} = \|S V^T L\|_2
$$

and $\alpha_1, \beta_1, \gamma_1, \cdots, \alpha_n, \beta_n, \gamma_n$ are such that $\frac{\|g'(A,b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_F} = \|S V^T L\|_2$.

Thus $\|S V^T L\|_2 \leq \kappa_{g,F}(A,b)$, which concludes the proof. $\qquad \square$

78

Remark 7. Let l_j be the j-th column of L, $j = 1, \cdots, k$. From

$$SV^T L = \begin{pmatrix} S_{11}v_1{}^T \\ \vdots \\ S_{nn}v_n{}^T \end{pmatrix} (l_1, \cdots, l_k) = \begin{pmatrix} S_{11}v_1{}^T l_1 & \cdots & S_{11}v_1{}^T l_k \\ \vdots & & \vdots \\ S_{nn}v_n{}^T l_1 & \cdots & S_{nn}v_n{}^T l_k \end{pmatrix},$$

it follows that $\left\| SV^T L \right\|_2$ is large when there exists at least one large S_{ii} and a l_j such that $v_i{}^T l_j \neq 0$. In particular, the condition number of $L^T x(A, b)$ is large when A has small singular values and L has components in the corresponding right singular vectors or when $\|r\|_2$ is large.

Remark 8. In the general case where L is an $n \times k$ matrix, the computation of $\kappa_{g,F}(A, b)$ via the exact formula given in Theorem 9 requires the computation of the singular values and the right singular vectors of A, which might be expensive in practice since it involves $2mn^2$ operations if we use a R-SVD algorithm and if $m \gg n$ (see [50, p. 254]). If the LLSP is solved using a direct method, the R factor of the QR decomposition of A (or equivalently in exact arithmetic, the Cholesky factor of $A^T A$) might be available. Since the right singular vectors of A are also those of R, the condition number can be computed in about $12n^3$ flops (using the Golub-Reinsch SVD, [50, p. 254]).

4.2.3 Special cases and GSVD

In this section, we analyze some special cases of practical relevance. Moreover, we relate the formula given in Theorem 9 for

$$\kappa_{g,F}(A, b)$$

to the Generalized Singular Value Decomposition (**GSVD**) ([21, p. 157], [50, p. 466], and [80, 99]). Using the GSVD of A and L^T, there exist $U_A \in \mathbb{R}^{m \times m}, U_L \in \mathbb{R}^{k \times k}$ orthogonal matrices and $Z \in \mathbb{R}^{n \times n}$ invertible such that:

$$U_A^T A = \begin{pmatrix} D_A \\ 0 \end{pmatrix} Z \quad \text{and} \quad U_L^T L^T = \begin{pmatrix} D_L & 0 \end{pmatrix} Z$$

with

$$D_A = diag(\alpha_1, \cdots, \alpha_n), \qquad D_L = diag(\beta_1, \cdots, \beta_k),$$
$$\alpha_i^2 + \beta_i^2 = 1 \quad i = 1, \cdots, k, \qquad \alpha_i = 1, \quad i = k+1, \cdots, n.$$

The diagonal matrix S can be decomposed in the product of two diagonal matrices

$$S = \Sigma^{-1} D$$

with

$$D_{ii} = \sqrt{\frac{\sigma_i{}^{-2} \|r\|_2^2 + \|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}}.$$

Then, taking into account the following relations

$$\left\| SV^T L \right\|_2 = \left\| L^T V S \right\|_2 = \left\| L^T V \Sigma^{-1} U^T U D \right\|_2 = \left\| L^T A^\dagger U D \right\|_2,$$
$$L^T A^\dagger = U_L \begin{pmatrix} D_L & 0 \end{pmatrix} Z Z^{-1} \begin{pmatrix} D_A^{-1} & 0 \end{pmatrix} U_A^T,$$

79

we can represent $\kappa_{g,F}(A,b)$ as

$$\kappa_{g,F}(A,b) = \left\| T\widetilde{H}D \right\|_2$$

where $T \in \mathbb{R}^{k \times k}$ is a diagonal matrix with $T_{ii} = \beta_i/\alpha_i, \quad i = 1, \cdots, k$ and $\widetilde{H} \in \mathbb{R}^{k \times n}$ is

$$\widetilde{H} = \left(\begin{array}{cc} I & 0 \end{array} \right) U_A^T U.$$

Note that $\left\| L^T A^\dagger \right\|_2 = \|T\|_2$.

We also point out that the diagonal entries of T are the nonzero generalized eigenvalues of

$$\lambda A^T A z = LL^T z.$$

There are two interesting special cases where the expression of $\kappa_{g,F}(A,b)$ is simpler. First, when $r = 0$, i.e. the LLSP problem is consistent, we have

$$D = \sqrt{\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}} \;\; I$$

and

$$\kappa_{g,F}(A,b) = \left\| T\widetilde{H} \right\|_2 \sqrt{\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}}.$$

Second, if we allow only perturbations on b and if we use the expression (4.8) of the derivative of $g(A,b)$, we get

$$\kappa_{g,F}(A,b) = \frac{\left\| L^T A^\dagger \right\|_2}{\beta} = \frac{\|T\|_2}{\beta}$$

(see Remark 11 in Section 4.3).

Other relevant cases where the expression for $\kappa_{g,F}(A,b)$ has a special interest are $L = I$ and L is a column vector.

In the special case where $L = I$, the formula given by Theorem 9 becomes

$$\kappa_{g,F}(A,b) = \left\| SV^T L \right\|_2 = \|S\|_2 = \max_i S_{ii} = \sigma_n^{-1} \sqrt{\frac{\sigma_n^{-2} \|r\|_2^2 + \|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}}.$$

Since $\left\| A^\dagger \right\|_2 = \sigma_n^{-1}$, we obtain that

$$\kappa_{g,F}(A,b) = \left\| A^\dagger \right\|_2 \sqrt{\frac{\|A^\dagger\|_2^2 \|r\|_2^2 + \|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}}.$$

This corresponds to the result known from [51] and also to a generalization of the formula of the condition number in Frobenius norm given in [46, p. 92] (where only A was perturbed).

Finally, let us study the particular case where L is a column vector i.e when g is a scalar derived function.

80

Corollary 2. *In the particular case when L is a vector ($L \in \mathbb{R}^n$), the absolute condition number of $g(A,b) = L^T x(A,b)$ is given by*

$$\kappa_{g,F}(A,b) = \left(\left\| L^T(A^TA)^{-1} \right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\| L^T A^\dagger \right\|_2^2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right) \right)^{\frac{1}{2}}.$$

Proof. By replacing $(A^TA)^{-1} = V\Sigma^{-2}V^T$ and $A^\dagger = V\Sigma^{-1}U^T$ in the expression of $K = (\left\| L^T(A^TA)^{-1} \right\|_2^2 \|r\|_2^2 + \left\| L^T A^\dagger \right\|_2^2 (\|x\|_2^2 + 1))^{\frac{1}{2}}$ we get

$$
\begin{aligned}
K^2 &= \left\| L^T V\Sigma^{-2}V^T \right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\| L^T V\Sigma^{-1}U^T \right\|_2^2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right) \\
&= \left\| L^T V\Sigma^{-2} \right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\| L^T V\Sigma^{-1} \right\|_2^2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right) \\
&= \left\| \Sigma^{-2}V^T L \right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\| \Sigma^{-1}V^T L \right\|_2^2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right).
\end{aligned}
$$

By writing $(z_1, \cdots, z_n)^T$ the vector $V^T L \in \mathbb{R}^n$ we obtain

$$
\begin{aligned}
K^2 &= \sum_{i=1}^n \frac{z_i^2}{\sigma_i^4} \frac{\|r\|_2^2}{\alpha^2} + \sum_{i=1}^n \frac{z_i^2}{\sigma_i^2} \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right) \\
&= \sum_{i=1}^n \frac{z_i^2}{\sigma_i^2} \left(\frac{\sigma_i^{-2}\|r\|_2^2 + \|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right) \\
&= \sum_{i=1}^n S_{ii}^2 z_i^2 \\
&= \left\| SV^T L \right\|_2^2,
\end{aligned}
$$

and Theorem 9 gives the result. $\qquad\qquad\square$

Remark 9. If the R factor in the QR decomposition of A is available, then we have

$$\left\| L^T A^+ \right\|_2 = \left\| R^{-T}L \right\|_2 \text{ and } \left\| L^T(A^TA)^{-1} \right\|_2 = \left\| R^{-1}(R^{-T}L) \right\|_2. \qquad (4.12)$$

It follows that, when $L \in \mathbb{R}^n$, the computation of $\kappa_{g,F}(A,b)$ can be performed by solving two successive n-by-n triangular systems which involve about $2n^2$ flops.

4.3   Estimate of the partial condition number in Frobenius and spectral norms

In many cases, obtaining a lower and/or an upper bound of $\kappa_{g,F}(A,b)$ is satisfactory when these bounds are tight enough and significantly cheaper to compute than the exact formula. Moreover, many applications use condition numbers expressed in the spectral norm. In the following theorem, we give bounds that have a correct order of magnitude for the partial condition numbers in the Frobenius and spectral norms.

Theorem 10. *The absolute condition numbers of* $g(A,b) = L^T x(A,b)$ $(L \in \mathbb{R}^{n \times k})$ *in the Frobenius and spectral norms can be respectively bounded as follows*

$$\frac{f(A,b)}{\sqrt{3}} \leq \kappa_{g,F}(A,b) \leq f(A,b)$$

$$\frac{f(A,b)}{\sqrt{3}} \leq \kappa_{g,2}(A,b) \leq \sqrt{2} f(A,b)$$

where

$$f(A,b) = \left(\left\| L^T (A^T A)^{-1} \right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\| L^T A^\dagger \right\|_2^2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right) \right)^{\frac{1}{2}}.$$

Proof. **Part 1:**
We start by establishing the lower bounds. Let w_1 and w_1' (resp. a_1 and a_1') be right (resp. the left) singular vectors corresponding to the largest singular values of respectively $L^T (A^T A)^{-1}$ and $L^T A^\dagger$. We use a particular perturbation $(\Delta A, \Delta b)$ expressed as

$$(\Delta A, \Delta b) = \left(\frac{r}{\alpha \|r\|_2} w_1^T + \epsilon w_1' \frac{x^T}{\alpha \|x\|_2}, -\epsilon \frac{w_1'}{\beta} \right),$$

where $\epsilon = \pm 1$.
By replacing this value of $(\Delta A, \Delta b)$ in (4.8) we get

$$\begin{aligned}
g'(A,b).(\Delta A, \Delta b) &= \frac{\|r\|_2}{\alpha} L^T (A^T A)^{-1} w_1 + \frac{\epsilon}{\alpha \|x\|_2} L^T (A^T A)^{-1} x w_1'^T r \\
&\quad - L^T A^\dagger r \frac{w_1^T x}{\alpha \|r\|_2} - \frac{\epsilon \|x\|_2}{\alpha} L^T A^\dagger w_1' - \frac{\epsilon}{\beta} L^T A^\dagger w_1'.
\end{aligned}$$

Since $r \in \operatorname{Im}(A)^\perp$ we have $A^\dagger r = 0$. Moreover, we have $w_1' \in \operatorname{Ker}(L^T A^\dagger)^\perp$ and thus $w_1' \in \operatorname{Im}(A^{+T} L)$ and can be written $w_1' = A^{+T} L \delta$ for some $\delta \in \mathbb{R}^k$. Then $w_1'^T r = \delta^T L^T A^\dagger r = 0$. It follows that

$$g'(A,b).(\Delta A, \Delta b) = \frac{\|r\|_2}{\alpha} L^T (A^T A)^{-1} w_1 - \frac{\epsilon \|x\|_2}{\alpha} L^T A^\dagger w_1' - \frac{\epsilon}{\beta} L^T A^\dagger w_1'.$$

From $L^T (A^T A)^{-1} w_1 = \left\| L^T (A^T A)^{-1} \right\|_2 a_1$ and $L^T A^\dagger w_1' = \left\| L^T A^\dagger \right\|_2 a_1'$, we obtain

$$g'(A,b).(\Delta A, \Delta b) = \left\| L^T (A^T A)^{-1} \right\|_2 \frac{\|r\|_2}{\alpha} a_1 - \epsilon \left(\frac{\|x\|_2}{\alpha} + \frac{1}{\beta} \right) \left\| L^T A^\dagger \right\|_2 a_1'.$$

Since a_1 and a_1' are unit vectors, $\|g'(A,b).(\Delta A, \Delta b)\|_2$ can be developed as

$$\begin{aligned}
\left\| g'(A,b).(\Delta A, \Delta b) \right\|_2^2 &= \left\| L^T (A^T A)^{-1} \right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\| L^T A^\dagger \right\|_2^2 \left(\frac{\|x\|_2}{\alpha} + \frac{1}{\beta} \right)^2 \\
&\quad - 2\epsilon \left\| L^T (A^T A)^{-1} \right\|_2 \frac{\|r\|_2}{\alpha} \left(\frac{\|x\|_2}{\alpha} + \frac{1}{\beta} \right) \left\| L^T A^\dagger \right\|_2 \cos(a_1, a_1').
\end{aligned}$$

82

By choosing $\epsilon = -sign(\cos(a_1, a_1'))$ the third term of the above expression becomes positive. Furthermore we have $(\frac{\|x\|_2}{\alpha} + \frac{1}{\beta})^2 \geq \frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}$. Then we obtain

$$\left\|g'(A,b).(\Delta A, \Delta b)\right\|_2 \geq \left(\left\|L^T(A^TA)^{-1}\right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\|L^TA^\dagger\right\|_2^2 (\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2})\right)^{\frac{1}{2}}$$

i.e

$$\left\|g'(A,b).(\Delta A, \Delta b)\right\|_2 \geq f(A,b).$$

On the other hand, we have

$$\|\Delta A\|_F^2 = \left\|\frac{r}{\alpha \|r\|_2}w_1^T\right\|_F^2 + \left\|w_1'\frac{x^T}{\alpha \|x\|_2}\right\|_F^2 + 2\,\epsilon\,\text{trace}\left((\frac{r}{\alpha \|r\|_2}w_1^T)^T(w_1'\frac{x^T}{\alpha \|x\|_2})\right) \text{ and } \left\|\frac{w_1'}{\beta}\right\|_2^2 = \frac{1}{\beta^2}$$

with

$$\left\|\frac{r}{\alpha \|r\|_2}w_1^T\right\|_F^2 = \left\|w_1'\frac{x^T}{\alpha \|x\|_2}\right\|_F^2 = \frac{1}{\alpha^2} \text{ and } \text{trace}\left((\frac{r}{\alpha \|r\|_2}w_1^T)^T(w_1'\frac{x^T}{\alpha \|x\|_2})\right) = 0.$$

Then $\|(\Delta A, \Delta b)\|_F = \sqrt{3}$ and thus we have $\frac{\|g'(A,b).(\Delta A,\Delta b)\|_2}{\|(\Delta A,\Delta b)\|_F} \geq \frac{f(A,b)}{\sqrt{3}}$ for a particular value of $(\Delta A, \Delta b)$. Furthermore, from $\|(\Delta A, \Delta b)\|_2 \leq \|(\Delta A, \Delta b)\|_F$ we get $\frac{\|g'(A,b).(\Delta A,\Delta b)\|_2}{\|(\Delta A,\Delta b)\|_2} \geq \frac{f(A,b)}{\sqrt{3}}$ (for the same particular value of $(\Delta A, \Delta b)$).
Then we obtain $\kappa_{g,F}(A,b) \geq \frac{f(A,b)}{\sqrt{3}}$ and $\kappa_{g,2}(A,b) \geq \frac{f(A,b)}{\sqrt{3}}$.

Part 2:
Let us now establish the upper bound for $\kappa_{g,F}(A,b)$ and $\kappa_{g,2}(A,b)$.
If $\Delta A_1 = AA^\dagger \Delta A$ and $\Delta A_2 = (I - AA^\dagger)\Delta A$, then it comes from (4.9) that $\forall(\Delta A, \Delta b) \in \mathbb{R}^{m\times n} \times \mathbb{R}^m$

$$\begin{aligned}\left\|g'(A,b).(\Delta A, \Delta b)\right\|_2 &\leq \left\|L^T(A^TA)^{-1}\right\|_2 \|\Delta A_2\|_2 \|r\|_2 + \left\|L^TA^\dagger\right\|_2 \|\Delta A_1\|_2 \|x\|_2 + \left\|L^TA^\dagger\right\|_2 \|\Delta b\|_2 \\ &= YX,\end{aligned}$$

where

$$Y = \left(\frac{\left\|L^T(A^TA)^{-1}\right\|_2 \|r\|_2}{\alpha}, \frac{\left\|L^TA^\dagger\right\|_2 \|x\|_2}{\alpha}, \frac{\left\|L^TA^\dagger\right\|_2}{\beta}\right)$$

and

$$X = (\alpha \|\Delta A_2\|_2, \alpha \|\Delta A_1\|_2, \beta \|\Delta b\|_2)^T.$$

Hence, from the Cauchy-Schwarz inequality we get

$$\left\|g'(A,b).(\Delta A, \Delta b)\right\|_2 \leq \|Y\|_2 \|X\|_2, \tag{4.13}$$

with

$$\|X\|_2^2 = \alpha^2 \|\Delta A_1\|_2^2 + \alpha^2 \|\Delta A_2\|_2^2 + \beta^2 \|\Delta b\|_2^2 \leq \alpha^2 \|\Delta A_1\|_F^2 + \alpha^2 \|\Delta A_2\|_F^2 + \beta^2 \|\Delta b\|_2^2$$

83

and
$$\|Y\|_2 = f(A, b).$$

Then, since $\|\Delta A\|_F^2 = \|\Delta A_1\|_F^2 + \|\Delta A_2\|_F^2$, we have $\|X\|_2 \le \|(\Delta A, \Delta b)\|_F$ and (4.13) yields
$$\left\|g'(A, b).(\Delta A, \Delta b)\right\|_2 \le \|(\Delta A, \Delta b)\|_F \|Y\|_2$$

which implies that
$$\kappa_{g,F}(A, b) \le f(A, b).$$

An upper bound of $\kappa_{g,2}(A, b)$ can be computed in a similar manner: we get from (4.8) that

$$
\begin{aligned}
\left\|g'(A, b).(\Delta A, \Delta b)\right\|_2 &\le \; (\left\|L^T(A^T A)^{-1}\right\|_2 \|r\|_2 + \left\|L^T A^\dagger\right\|_2 \|x\|_2) \|\Delta A\|_2 + \left\|L^T A^\dagger\right\|_2 \|\Delta b\|_2 \\
&= \; Y'X',
\end{aligned}
$$

where $Y' = \left(\frac{\left\|L^T(A^T A)^{-1}\right\|_2 \|r\|_2 + \left\|L^T A^\dagger\right\|_2 \|x\|_2}{\alpha}, \frac{\left\|L^T A^\dagger\right\|_2}{\beta}\right)$ and $X' = (\alpha \|\Delta A\|_2, \beta \|\Delta b\|_2)^T$.
Since $\|X'\|_2 = \|(\Delta A, \Delta b)\|_2$ we have $\kappa_{g,2}(A, b) \le \|Y'\|_2$.
Using then the inequality

$$
\left(\left\|L^T(A^T A)^{-1}\right\|_2 \|r\|_2 + \left\|L^T A^\dagger\right\|_2 \|x\|_2\right)^2 \le 2\left(\left\|L^T(A^T A)^{-1}\right\|_2^2 \|r\|_2^2 + \left\|L^T A^\dagger\right\|_2^2 \|x\|_2^2\right)
$$

we get $\|Y'\|_2 \le \sqrt{2}\,\|Y\|_2$ and finally obtain $\kappa_{g,2}(A, b) \le \sqrt{2} f(A, b)$ which concludes the proof. $\qquad\square$

Theorem 10 shows that $f(A, b)$ can be considered as an estimate of the partial condition number that lies within a factor $\sqrt{3}$ of $\kappa_{g,F}(A, b)$ or $\kappa_{g,2}(A, b)$.
Another observation is that we have

$$\frac{1}{\sqrt{6}} \le \frac{\kappa_{g,F}(A, b)}{\kappa_{g,2}(A, b)} \le \sqrt{3}.$$

Thus even if the Frobenius and spectral norms of a given matrix can be very different (for $X \in \mathbb{R}^{m \times n}$, we have $\|X\|_2 \le \|X\|_F \le \sqrt{n}\,\|X\|_2$), the condition numbers expressed in both norms are of same order. The result is that a good estimate of $\kappa_{g,F}(A, b)$ is also a good estimate of $\kappa_{g,2}(A, b)$.
Moreover, (4.12) shows that if the R factor of A is available, $f(A, b)$ can be computed by solving two n-by-n triangular systems with k right-hand sides and thus the computational cost is $2kn^2$.

Remark 10. We can check on the following example that $\kappa_{g,F}(A, b)$ is not equal to $f(A, b)$. Let us consider

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad L = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \text{ and } b = \begin{pmatrix} 2/\sqrt{2} \\ 1/\sqrt{2} \\ 1 \end{pmatrix}.$$

84

We have

$$x = (1/\sqrt{2}, 1/\sqrt{2})^T \text{ and } \|x\|_2 = \|r\|_2 = 1,$$

and we get

$$\kappa_{g,F}(A, b) = \frac{\sqrt{45}}{4} < f(A, b) = \frac{\sqrt{13}}{2}.$$

Remark 11. Using the definition of the condition number and of the product norms, tight estimates for the partial condition number for perturbations of A only (resp. b only) can be obtained by taking $\alpha > 0$ and $\beta = +\infty$ (resp. $\beta > 0$ and $\alpha = +\infty$) in Theorem 10. In particular, when we perturb only b we have, with the notations of Section 4.2.3,

$$f(A, b) = \frac{\left\|L^T A^\dagger\right\|_2}{\beta} = \frac{\|T\|_2}{\beta} = \kappa_{g,F}(A, b).$$

Moreover, when $r = 0$ we have

$$f(A, b) = \left\|L^T A^\dagger\right\|_2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}\right)^{\frac{1}{2}} = \|T\|_2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}\right)^{\frac{1}{2}}.$$

Remark 12. In the special case where $L = I$, we have

$$f(A, b) = \left(\left\|(A^T A)^{-1}\right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\|A^\dagger\right\|_2^2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}\right)\right)^{\frac{1}{2}}.$$

Since $\left\|(A^T A)^{-1}\right\|_2 = \left\|A^\dagger\right\|_2^2$ we obtain that

$$f(A, b) = \left\|A^\dagger\right\|_2 \sqrt{\frac{\|A^\dagger\|_2^2 \|r\|_2^2 + \|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2}}.$$

In that case $\kappa_{g,F}(A, b)$ is exactly equal to $f(A, b)$ due to [51].
Regarding the condition number in spectral norm, since we have $\|(\Delta A, \Delta b)\|_2 \leq \|(\Delta A, \Delta b)\|_F$ we get $\kappa_{g,2}(A, b) \geq f(A, b)$. This lower bound is similar to that obtained in [46] (where only A is perturbed). As mentioned in [46], an upper bound of $\kappa_{g,2}(A)$ is $\kappa_{g,2}^u(A) = \left\|A^\dagger\right\|_2^2 \|r\|_2 + \left\|A^\dagger\right\|_2 \|x\|_2$. If we take $\alpha = 1$ and $\beta = +\infty$, we notice that $f(A, b) \leq \kappa_{g,2}^u(A) \leq \sqrt{2} f(A, b)$ showing thus that our upper bound and $\kappa_{g,2}^u(A)$ are essentially the same.

Remark 13. Generalization to other product norms:
Other product norms may have been used for the data space $\mathbb{R}^{m \times n} \times \mathbb{R}^m$.
If we consider a norm ν on $\mathbb{R}^2$ such that $c_1 \nu(x, y) \leq \sqrt{x^2 + y^2} \leq c_2 \nu(x, y)$ then we can define a product norm $\|(A, b)\|_{F,\nu} = \nu(\alpha \|\Delta A\|_F, \beta \|\Delta b\|_2)$. For instance in [53], ν corresponds to $\|.\|_\infty$. Note that the product norm $\|(.,.)\|_F$ used throughout this chapter corresponds to $\nu = \|.\|_2$ and that with the above notation we have $\|(A, b)\|_{F,2} = \|(A, b)\|_F$. Then the following inequality holds

$$c_1 \|(\Delta A, \Delta b)\|_{F,\nu} \leq \|(\Delta A, \Delta b)\|_F \leq c_2 \|(\Delta A, \Delta b)\|_{F,\nu}.$$

If we denote $\kappa_{g,F,\nu}(A,b) = \max_{(\Delta A, \Delta b)} \frac{\|g'(A,b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_{F,\nu}}$ we obtain

$$\frac{\kappa_{g,F,\nu}(A,b)}{c_2} \leq \kappa_{g,F}(A,b) \leq \frac{\kappa_{g,F,\nu}(A,b)}{c_1}.$$

Using the bounds for $\kappa_{g,F}$ given in Theorem 10 we can obtain tight bounds for the partial condition number expressed using the product norm based on ν and when the perturbations on matrices are measured with the Frobenius norm:

$$\frac{c_1}{\sqrt{3}} f(A,b) \leq \kappa_{g,F,\nu}(A,b) \leq c_2 f(A,b).$$

Similarly, if the perturbations on matrices are measured with the spectral norm, we get

$$\frac{c_1}{\sqrt{3}} f(A,b) \leq \kappa_{g,2,\nu}(A,b) \leq c_2 \sqrt{2} f(A,b).$$

The bounds obtained for three possible product norms ($\nu = \|.\|_\infty$, $\nu = \|.\|_2$ and $\nu = \|.\|_1$) are given in Table 4.2 when using the Frobenius norm for matrices and in Table 4.3 when using the spectral norm for matrices.

Table 4.2: Bounds for partial condition number (Frobenius norm on matrices).

product norm	ν, c_1, c_2	lower bound (factor of $f(A,b)$)	upper bound (factor of $f(A,b)$)
$\max\{\alpha \|\Delta A\|_F, \beta \|\Delta b\|_2\}$	$\|.\|_\infty$, $\frac{1}{\sqrt{2}}$, 1	$\frac{1}{\sqrt{6}}$	1
$\sqrt{\alpha^2 \|\Delta A\|_F^2 + \beta^2 \|\Delta b\|_2^2}$	$\|.\|_2$, 1, ,1	$\frac{1}{\sqrt{3}}$	1
$\alpha \|\Delta A\|_F + \beta \|\Delta b\|_2$	$\|.\|_1$, 1, $\sqrt{2}$	$\frac{1}{\sqrt{3}}$	$\sqrt{2}$

Table 4.3: Bounds for partial condition number (spectral norm on matrices).

product norm	ν, c_1, c_2	lower bound (factor of $f(A,b)$)	upper bound (factor of $f(A,b)$)
$\max\{\alpha \|\Delta A\|_2, \beta \|\Delta b\|_2\}$	$\|.\|_\infty$, $\frac{1}{\sqrt{2}}$, 1	$\frac{1}{\sqrt{6}}$	$\sqrt{2}$
$\sqrt{\alpha^2 \|\Delta A\|_2^2 + \beta^2 \|\Delta b\|_2^2}$	$\|.\|_2$, 1, 1	$\frac{1}{\sqrt{3}}$	$\sqrt{2}$
$\alpha \|\Delta A\|_2 + \beta \|\Delta b\|_2$	$\|.\|_1$, 1, $\sqrt{2}$	$\frac{1}{\sqrt{3}}$	2

4.4   Statistical estimation of the partial condition number

In this section we compute a statistical estimate of the partial condition number. We have seen in Section 4.3 that using the Frobenius or the spectral norm for the matrices gives condition numbers that are of the same order of magnitude. We compute here a statistical estimate of $\kappa_{g,F}(A,b)$.

Let $(z_1, z_2, \cdots, z_q)$ be an orthonormal basis for a subspace of dimension q ($q \leq k$) that has been randomly and uniformly selected from the space of all q-dimensional subspaces of $\mathbb{R}^k$ (this can be done by choosing q random vectors and then orthogonalizing). Let us denote $g_i(A, b) = (Lz_i)^T x(A, b)$.

Since $Lz_i \in \mathbb{R}^n$, the absolute condition number of g_i can be computed via the exact formula given in Corollary 2 i.e

$$\kappa_{g_i, F}(A, b) = \left(\left\| (Lz_i)^T (A^T A)^{-1} \right\|_2^2 \frac{\|r\|_2^2}{\alpha^2} + \left\| (Lz_i)^T A^\dagger \right\|_2^2 \left(\frac{\|x\|_2^2}{\alpha^2} + \frac{1}{\beta^2} \right) \right)^{\frac{1}{2}}. \qquad (4.14)$$

We define the random variable $\phi(q)$ by

$$\phi(q) = (\frac{k}{q} \sum_{i=1}^{q} \kappa_{g_i, F}(A, b)^2)^{\frac{1}{2}}.$$

Let the operator $E(.)$ denote the expected value. The following proposition shows that the root mean squared of $\phi(q)$, defined by $R(\phi(q)) = \sqrt{E(\phi(q)^2)}$ can be considered as an estimate for the condition number of $g(A, b) = L^T x(A, b)$.

Proposition 3. *The absolute condition number can be bounded as follows:*

$$\frac{R(\phi(q))}{\sqrt{k}} \leq \kappa_{g, F}(A, b) \leq R(\phi(q)). \qquad (4.15)$$

Proof. Let vec be the operator that stacks the columns of a matrix into a long vector and M be the k-by-$m(n+1)$ matrix such that $vec(g'(A, b).(\Delta A, \Delta b)) = M \begin{pmatrix} vec(\alpha \Delta A) \\ vec(\beta \Delta b) \end{pmatrix}$.

Note that M depends on A, b, L and not on the z_i.
Then we have:

$$\kappa_{g, F}(A, b) = \max_{(\Delta A, \Delta b)} \frac{\|g'(A, b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|_F} = \max_{(\Delta A, \Delta b)} \frac{\|vec(g'(A, b).(\Delta A, \Delta b))\|_2}{\left\| \begin{pmatrix} vec(\alpha \Delta A) \\ vec(\beta \Delta b) \end{pmatrix} \right\|_2}$$

$$= \max_{z \in \mathbb{R}^{m(n+1)}, z \neq 0} \frac{\|M z\|_2}{\|z\|_2} = \|M\|_2 = \|M^T\|_2.$$

Let $Z = [z_1, z_2, \cdots, z_q]$ be the k-by-q random matrix with orthonormal columns z_i. From [55] it follows that $\frac{k}{q} \|M^T Z\|_F^2$ is an unbiased estimator of the Frobenius norm of the $m(n+1)$-by-k matrix M^T i.e we have $E(\frac{k}{q} \|M^T Z\|_F^2) = \|M^T\|_F^2$.
From

$$\|M^T Z\|_F^2 = \|Z^T M\|_F^2$$

$$= \left\| \begin{pmatrix} z_1^T M \\ \vdots \\ z_q^T M \end{pmatrix} \right\|_F^2$$

we get, since $z_i^T M$ is a row vector,

$$\left\|M^T Z\right\|_F^2 \;=\; \sum_{i=1}^q \left\|z_i^T M\right\|_2^2 .$$

We notice that for all vector $u \in \mathbb{R}^k$, if we consider the function $g_u(A,b) = u^T g(A,b)$, then we have $\left\|u^T M\right\|_F = \|g_u'(A,b)\| = \kappa_{g_u,F}(A,b)$ and therefore

$$\left\|z_i^T M\right\|_F = \kappa_{g_i,F}(A,b).$$

Eventually we obtain

$$\left\|M^T\right\|_F^2 = E\big(\frac{k}{q}\sum_{i=1}^q \kappa_{g_i,F}(A,b)^2\big) = E(\phi(q)^2).$$

Moreover, considering that $M^T \in \mathbb{R}^{m(n+1)\times k}$ and using the well-known inequality

$$\frac{\left\|M^T\right\|_F}{\sqrt{k}} \le \left\|M^T\right\|_2 \le \left\|M^T\right\|_F ,$$

we get the result (4.15). Then we will consider $\phi(q)\frac{\|(A,b)\|_F}{\|L^T \tilde{x}\|_2}$ as an estimator of $\kappa_{g,F}^{(rel)}(A,b)$. $\qquad\square$

The root mean squared of $\phi(q)$ is an upper bound of $\kappa_g(A,b)$, and estimates $\kappa_{g,F}(A,b)$ within a factor $\sqrt{k}$. Proposition 3 involves the computation of the condition number of each $g_i(A,b), i = 1, \cdots, q$. From Remark 9, it follows that the computational cost of each $\kappa_{g_i,F}(A,b)$ is $2n^2$ (if the R factor of the QR decomposition of A is available). Hence, for a given sample of vectors z_i, $i = 1, \ldots, q$, computing $\phi(q)$ requires about $2qn^2$ flops. However, Proposition 3 is mostly of theoretical interest, since it relies on the computation of the root mean squared of a random variable, without providing a practical method to obtain it. In the next proposition, the use of the small sample estimate theory developed by Kenney and Laub [55] gives a first answer to this question by showing that the evaluation of $\phi(q)$ using only one sample of q vectors $z_1, z_2, \cdots, z_q$ in the unit sphere may provide an acceptable estimate.

Proposition 4. *For any $\alpha > 10$,*

$$Pr\left(\frac{\phi(q)}{\alpha\sqrt{k}} \le \kappa_{g,F}(A,b) \le \alpha\phi(q)\right) \ge 1 - \alpha^{-q}.$$

This probability approaches 1 very fast as q increases. For $\alpha = 11$ and $q = 3$ the probability for $\phi(q)$ to estimate $\kappa_{g,F}(A,b)$ within a factor $11\sqrt{k}$ is 99.9%.

Proof. We define as in the proof of Proposition 3 the matrix M as the matrix related to the *vec* operation representing the linear operator $g'(A,b)$. From [55, (4) p. 781 and (9) p. 783] we get

$$Pr\left(\frac{\left\|M^T\right\|_F}{\alpha} \le \phi(q) \le \alpha\left\|M^T\right\|_F\right) \ge 1 - \alpha^{-q}. \tag{4.16}$$

88

We have seen in the proof of Proposition 3 that $\kappa_{g,F}(A,b) = \left\|M^T\right\|_2$. Then we have

$$\kappa_{g,F}(A,b) \leq \left\|M^T\right\|_F \leq \kappa_{g,F}(A,b) \ \sqrt{k}.$$

It follows that, for the random variable $\phi(q)$, we have

$$Pr\left(\frac{\kappa_{g,F}(A,b)}{\alpha} \leq \phi(q) \leq \alpha\kappa_{g,F}(A,b) \ \sqrt{k}\right) \geq Pr\left(\frac{\left\|M^T\right\|_F}{\alpha} \leq \phi(q) \leq \alpha\left\|M^T\right\|_F\right).$$

Then we obtain the result from

$$Pr\left(\frac{\kappa_{g,F}(A,b)}{\alpha} \leq \phi(q) \leq \alpha\kappa_{g,F}(A,b) \ \sqrt{k}\right) = Pr\left(\frac{\phi(q)}{\alpha\sqrt{k}} \leq \kappa_{g,F}(A,b) \leq \alpha\phi(q)\right).$$

$\square$

We see from this proposition that it may not be necessary to estimate the root mean squared of $\phi(q)$ using sophisticated algorithms. Indeed only one sample of $\phi(q)$ obtained for $q = 3$ provides an estimate of $\kappa_{g,F}(A,b)$ within a factor $\alpha\sqrt{k}$.

Remark 14. If $k = 1$ then $Z = 1$ and the problem is reduced to computing $\kappa_{g_1}(A,b)$ In this case, $\phi(1)$ is exactly the partial condition number of $L^T x(A,b)$.

Remark 15. Concerning the computation of the statistical estimate in the presence of roundoff errors, the numerical reliability of the statistical estimate relies on an accurate computation of the $\kappa_{g_i,F}(A,b)$ for a given z_i. Let A be a 17-by-13 Vandermonde matrix, b a random vector and $L \in \mathbb{R}^n$ the right singular vector v_n
Using the Mathematica software [106] that computes in exact arithmetic, we obtained $\kappa_{g,F}^{(rel)}(A,b) \approx 5 \cdot 10^8$. If the triangular factor R form $A^T A = R^T R$ is obtained by the QR decomposition of A, we get $\kappa_{g,F}^{(rel)}(A,b) \approx 5 \cdot 10^8$. If R is computed via a classical Cholesky factorization, we get $\kappa_{g,F}(A,b)^{(rel)} \approx 10^{10}$.
Corollary 2 and Remark 9 show that the computation of $\kappa_{g,F}(A,b)^{(rel)}$ involves linear systems of the type $A^T A x = d$, which differs from the usual normal equation for least squares in their right-hand side. Our observation that for this kind of ill-conditioned systems, a QR factorization is more accurate than a Cholesky factorization is in agreement with [45].

Another approach is proposed in [24] and [70] to obtain partial condition statistical estimates for LLSP and linear systems. Using this method gives the following proposition.

Proposition 5. *The partial condition number of* $g(A) = p^T x(A,b)$ *can be approximated by*

$$\psi(q) = \frac{E_q(\sum_{i=1}^{q} \kappa_{g_i,F}(A,b)^2)^{\frac{1}{2}}\|(A,b)\|}{E_k \left\|p^T\tilde{x}\right\|_2}$$

where $z_1, z_2, \cdots, z_q$ *are* q *orthonormal vectors uniformly and randomly selected in* S_{k-1}, E_q *and* E_k *being the Wallis factors. Moreover, we have the inequality*

$$\psi(q) \leq \sqrt{q}\frac{E_q}{E_k}\kappa_{g,F}^{(rel)}(A,b)$$

89

Proof. It comes from (4.1) that for estimating the condition number of $g(A, b) = p^T x(A, b)$, we are concerned with the relative error $\delta = \frac{\|p^T x - p^T \tilde{x}\|_2}{\|p^T x\|_2}$. If we approximate x by $\tilde{x}$, then we obtain

$$\delta = \frac{\|p^T (x - \tilde{x})\|_2}{\|p^T \tilde{x}\|_2} \tag{4.17}$$

We will estimate δ by using a method that is referred to as small-sample statistical method [69]. For any $v \in \mathbb{R}^k$, if $z_1, \cdots, z_q$ are random orthogonal vectors from S_{k-1}, then we have

$$E(\sqrt{|v^T z_1|^2 + \cdots + |v^T z_q|^2}) = \frac{E_k}{E_q} \|v\|_2 \tag{4.18}$$

where $E_1 = 1$, and for $k > 1$

$$E_k = \frac{1 \cdot 3 \cdot 5 \cdots (k-2)}{2 \cdot 4 \cdot 6 \cdots (k-1)} \text{ for } k \text{ odd}$$

$$E_k = \frac{2}{\pi} \frac{2 \cdot 4 \cdot 6 \cdots (k-2)}{1 \cdot 3 \cdot 5 \cdots (k-1)} \text{ for } k \text{ even.}$$

E_k is called the Wallis factor and can be approximated by $\sqrt{\frac{2}{\pi(k-\frac{1}{2})}}$.
That means that we can estimate $\|v\|_2$ by

$$\zeta(q) = \frac{E_q}{E_k} E(\sqrt{|v^T z_1|^2 + \cdots + |v^T z_q|^2}).$$

For instance for $q = 3$ the probability that $\zeta(q)$ lies within a factor α of $\|v\|_2$ is

$$Pr(\frac{\|v\|_2}{\alpha} \leq \zeta(q) \leq \alpha \|v\|_2) \approx 1 - \frac{32}{3\pi^2 \alpha^3}. \tag{4.19}$$

For $\alpha = 10$, we obtain a probability of 99.9%.
Now we are going to apply the above method to estimate $v = p^T (x - \tilde{x})$. We consider q random orthogonal vectors in S_{k-1}, these vectors being obtained for instance via a QR factorization of a random matrix $Z \in \mathbb{R}^{k \times q}$. We denote

$$\varepsilon_q = \frac{E_q (\sum_{i=1}^{q} ((pz_i)^T (x - \tilde{x}))^2)^{\frac{1}{2}}}{E_k \|p^T x\|_2}.$$

Then for each $i \in \{1, \cdots, q\}$, we have the first-order bound

$$|(pz_i)^T (x - \tilde{x})| \leq \|g_i'(A, b).(\Delta A, \Delta b)\|_2. \tag{4.20}$$

Then using 4.1 we get

$$|(pz_i)^T (x - \tilde{x})| \leq \|g_i'(A, b)\| \|(\Delta A, \Delta b)\|.$$

Hence

$$\varepsilon_q \leq \frac{E_q (\sum_{i=1}^{q} \|g_i'(A, b)\|^2)^{\frac{1}{2}} \|(A, b)\|}{E_k \|p^T \tilde{x}\|_2} \frac{\|(\Delta A, \Delta b)\|}{\|(A, b)\|}.$$

90

Since, with the notation of Section 4.2.2, we have

$$\forall (\Delta A, \Delta b) \in \mathbb{R}^{m \times n} \times \mathbb{R}^m, \frac{\|p^T x - p^T \tilde{x}\|_2}{\|p^T x\|_2} \leq \kappa_{g,F}^{(rel)}(A, b) \frac{\|(\Delta A, \Delta b)\|}{\|(A, b)\|}$$

then we will consider that

$$\psi(q) = \frac{E_q (\sum_{i=1}^q \kappa_{g_i,F}(A, b)^2)^{\frac{1}{2}} \|(A, b)\|}{E_k \|p^T \tilde{x}\|_2} \tag{4.21}$$

is an estimate of $\kappa_{g,F}^{(rel)}(A, b)$, each $\kappa_{g_i,F}(A, b)$ being computed using (4.14). From $g_i'(A, b).(\Delta A, \Delta b) = z_i^T g'(A, b).(\Delta A, \Delta b)$ we get

$$\|g_i'(A, b)\| = \max_{(\Delta A, \Delta b)} \frac{|g_i'(A, b).(\Delta A, \Delta b)|}{\|(\Delta A, \Delta b)\|} = \max_{(\Delta A, \Delta b)} \frac{|z_i^T g'(A, b).(\Delta A, \Delta b)|}{\|(\Delta A, \Delta b)\|}$$

and Cauchy-Schwarz yields

$$\|g_i'(A, b)\| \leq \|z_i^T\|_2 \max_{(\Delta A, \Delta b)} \frac{\|g'(A, b).(\Delta A, \Delta b)\|_2}{\|(\Delta A, \Delta b)\|} = \kappa_{g,F}(A, b).$$

Then from (4.21) it is straightforward that

$$\psi(q) \leq \sqrt{q} \frac{E_q}{E_k} \kappa_{g,F}^{(rel)}(A, b).$$

$\square$

Contrary to the result of Proposition 3, the expected value of $\psi(q)$ may not be an upper bound of $\kappa_{g,F}^{(rel)}(A, b)$ because of the first-order approximation (4.20). This is the reason why we think that $\phi(q)$ should be preferred.

4.5   Numerical experiments

All experiments were performed in Matlab 6.5 using a machine precision $2.22 \cdot 10^{-16}$.

4.5.1   Examples

For the examples of Section 4.2, we compute the partial condition number using the formula given in Theorem 9.
In the first example we have

$$A = \begin{pmatrix} 1 & 1 & \epsilon^2 \\ \epsilon & 0 & \epsilon^2 \\ 0 & \epsilon & \epsilon^2 \\ \epsilon^2 & \epsilon^2 & 2 \end{pmatrix}$$

and we assume that only A is perturbed. If we consider the values for L that are $\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$ and $L = (0, 0, 1)^T$ then we obtain partial condition numbers $\kappa_{g,F}^{(rel)}(A)$ that

are respectively 10^{24} and 1.22, as expected since there is 50% relative error on x_1 and x_2 and there is no error on x_3.

In the second example where A is the $10 - by - 4$ Vandermonde matrix defined by $A_{ij} = \frac{1}{(10+i)^{j-1}}$ and only b is perturbed, the partial condition numbers $\kappa_{g,F}^{(rel)}(b)$ with respect to each component x_1, x_2, x_3, x_4 are respectively $4.5 \cdot 10^2, 2 \cdot 10^4, 3 \cdot 10^5, 1.4 \cdot 10^6$ which is consistent with the error variation given in Section 4.2 for each component.

4.5.2 Average behaviour of the statistical estimate

We compare here the statistical estimate $\phi(q)$ described in the previous section with the partial condition number obtained via the exact formula given in Theorem 9. We suppose that only A is perturbed and then the partial condition number can be expressed as $\kappa_{g,F}^{(rel)}(A)$. We use the method described in [81], also used in [22], in order to construct test problems $[A, x, r, b] = P(m, n, n_r, l)$ with

$$A = Y \begin{pmatrix} D \\ 0 \end{pmatrix} Z^T \in \mathbb{R}^{m \times n}, Y = I - 2yy^T, Z = I - 2zz^T,$$

where $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^n$ are random unit vectors and $D = n^{-l} diag(n^l, (n-1)^l, \cdots, 1)$. $x = (1, 2^2, \cdots, n^2)^T$ is given and $r = Y \begin{pmatrix} 0 \\ c \end{pmatrix} \in \mathbb{R}^m$ is computed with $c \in \mathbb{R}^{m-n}$ random vector of norm n_r. The right-hand side is $b = Y \begin{pmatrix} DZx \\ c \end{pmatrix}$. By construction, the condition number of A and D is n^l.

In our experiments, we consider the matrices

$$A = \begin{pmatrix} A_1 & E' \\ E & A_2 \end{pmatrix} \text{ and } L = \begin{pmatrix} I \\ 0 \end{pmatrix},$$

where $A_1 \in \mathbb{R}^{m_1 \times n_1}$, $A_2 \in \mathbb{R}^{m_2 \times n_2}$, $L \in \mathbb{R}^{n \times n_1}$, $m_1 + m_2 = m$, $n_1 + n_2 = n$, and E and E' contain the same element e_p which defines the coupling between A_1 and A_2. The matrices A_1 and A_2 are randomly generated using respectively $P(m_1, n_1, n_{r_1}, l_1)$ and $P(m_2, n_2, n_{r_2}, l_2)$.

For each sample matrix, we compute in Matlab:

1. the partial condition number $\kappa_{g,F}^{(rel)}(A)$ using the exact formula given in Theorem 9 and based on the singular value decomposition of A,

2. the statistical estimate $\phi(3)$ using three random orthogonal vectors and computing each $\kappa_{g_i,F}(A, b)$, $i = 1, 2, 3$ with the R factor of the QR decomposition of A.

These data are then compared by computing the ratio

$$\gamma = \frac{\phi(3)}{\kappa_{g,F}^{(rel)}(A)}.$$

Table 4.4 contains the mean $\overline{\gamma}$ and the standard deviation s of γ obtained on 1000 random matrices with $m_1 = 12, n_1 = 10, m_2 = 17, n_2 = 13$ by varying the condition

92

numbers $n_1{}^{l_1}$ and $n_2{}^{l_2}$ of respectively A_1 and A_2 and the coupling coefficient e_p. The residual norms are set to $n_{r_1} = n_{r_2} = 1$. In all cases, $\overline{\gamma}$ is close to 1 and s is about 0.3. The statistical estimate $\phi(3)$ lies within a factor 1.22 of $\kappa_{g,F}^{(rel)}(A)$ which is very accurate in condition number estimation. We notice that in two cases, $\phi(3)$ is lower than 1. This is possible because Proposition 3 shows that $E(\phi(3)^2)$ is an upper bound of $\kappa_{g,F}(A)^2$ but not necessarily $\phi(3)^2$.

Table 4.4: Mean and standard deviation of the ratio between statistical and exact condition number of $L^T x$.

condition		$e_p = 10^{-5}$		$e_p = 1$		$e_p = 10^5$	
l_1	l_2	$\overline{\gamma}$	s	$\overline{\gamma}$	s	$\overline{\gamma}$	s
1	1	1.22	$2.28 \cdot 10^{-1}$	1.15	$2.99 \cdot 10^{-1}$	1.07	$3.60 \cdot 10^{-1}$
1	8	1.02	$3.19 \cdot 10^{-1}$	1.22	$3.05 \cdot 10^{-1}$	1.21	$3.35 \cdot 10^{-1}$
8	1	$9 \cdot 10^{-1}$	$3 \cdot 10^{-1}$	1.13	$3 \cdot 10^{-1}$	1.06	$3.45 \cdot 10^{-1}$
8	8	$9.23 \cdot 10^{-1}$	$2.89 \cdot 10^{-1}$	1.22	$2.95 \cdot 10^{-1}$	1.18	$3.33 \cdot 10^{-1}$

4.6 Estimates vs exact formula

We assume that the R factor of the QR decomposition of A is known. We gather in Table 4.5 the results obtained in this chapter in terms of accuracy and flops counts for the estimation of the partial condition number for the LLSP.

Table 4.5: Comparison between exact formula and estimates for $\kappa_{g,F}(A, b)$.

$\kappa_{g,F}(A, b)$	flops	accuracy
exact formula $n \ll m$	$12n^3$	exact
estimate $f(A, b)$ $k \ll n$	$2kn^2$	$\frac{f(A,b)}{\sqrt{3}} \leq \kappa_{g,F}(A, b) \leq f(A, b)$
stat. estimate $\phi(q)$ $q \ll k$	$2qn^2$	$\frac{\phi(q)}{\alpha\sqrt{k}} \leq \kappa_{g,F}(A, b) \leq \alpha\phi(q)$ $Pr \geq 1 - \alpha^{-q}$ for $\alpha > 10$

Table 4.6 gives the exact value and the estimates of the partial condition number (with their computational cost) in the particular situation where

$$m = 1500, \ n = 1000, \ k = 50,$$

$$A_1 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, \ L_1 = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix},$$

$$A = \begin{pmatrix} A_1 & 0 \\ 0 & I_{n-2} \\ 0 & 0 \end{pmatrix} \text{ and } b = \frac{1}{\sqrt{2}}(2, 1, \cdots, 1)^T, \ L = \begin{pmatrix} L_1 & 0 \\ 0 & I_{k-2} \\ 0 & 0 \end{pmatrix}.$$

We see here that the statistical estimates may provide information on the condition number using a very small amount of floating-point operations compared with the two other methods.

Table 4.6: Exact value and estimates of the partial condition number for a particular example.

Computed quantity	$\kappa_{g,F}^{(rel)}(A,b)$	$f(A,b)\frac{\|(A,b)\|_F}{\|L^T\tilde{x}\|_2}$	$\phi(q)\frac{\|(A,b)\|_F}{\|L^T\tilde{x}\|_2}$
Obtained value	$2.09 \cdot 10^2$	$2.18 \cdot 10^2$	$11.44 \cdot 10^2$
Flop count $(\times 10^6)$	12,000	100	6

4.7   Conclusion of Chapter 4

We have shown the relevance of the partial condition number for test cases from parameter estimation. This partial condition number evaluates the sensitivity of $L^T x$ where x is the solution of an LLSP when A and/or b are perturbed. It can be computed via a closed formula, an estimate of correct order of magnitude or a statistical estimate. The choice will depend on the size of the LLSP and on the needed accuracy. The closed formula requires $\mathcal{O}(n^3)$ flops and is affordable for small problems only. The estimates will be preferred for larger problems especially if $k \ll n$ since their computational cost is in $\mathcal{O}(n^2)$.

The partial condition number defined in this chapter can be very useful for the GOCE application. On the one hand, the computed solution sometimes includes internal variables that are inherent in instruments or in the mission. These variables do not correspond to gravity field coefficients and must not be considered when estimating the condition number of the LLSP solution. On the other hand, the polar gap problem due to the non-polar orbit implies an observability problem at the poles. It is observed that this effect may cause numerical instabilities that affect more particularly some gravity field parameters [83]. These coefficients may be computed with much less accuracy than others and it could be worth to compute different partial condition numbers in that case. The partial condition number has not been implemented yet in the parallel solver. It will be tested in a further step, first in a serial version, when the GOCE data will be available.

Conclusion

This thesis proposes computational and numerical tools that help in solving the large linear least squares problems encountered in gravity field computations.

Our work contributes to the research in two areas which are in dense parallel computing and in linear least squares error analysis. The distributed packed storage format defined in this thesis enables us to save about half the memory for in-core calculations because it takes advantage of the symmetric or triangular structure of the matrices. This possibility is not yet available in the standard parallel libraries for dense linear algebra. This distributed packed storage can be easily extended to other linear algebra parallel computations. Moreover, we have established closed formulae and have proposed sharp and statistical estimates in order to compute the condition number of a linear function of a least squares solution.

This thesis is also of practical interest for computing efficiently and accurately the gravity field parameters coming from GOCE observations. An operational parallel solver based on the normal equations approach was integrated into the GINS solver from the French Space Agency. Also, a parallel distributed QR solver using our distributed packed storage scheme was implemented in a research version. When the GOCE data will be available, the users will have the choice between the two methods, depending on the required accuracy. This choice will involve a trade off between computational cost and accuracy (normal equations vs QR factorization) and also between memory and performance for large processor counts (packed implementation vs existing libraries). Because of the partial condition number proposed in Chapter 4, physicists will have more information about the sensitivity of the physical problem.

Subsequent to this thesis there are some research directions that deserve to be investigated:

- The performance tuning performed throughout this thesis was often based on empirical models. The selection of the algorithm and the choice of the parameters strongly depend on the platform and on the implementation of the parallel libraries on these platforms. Future work could consist in studying models and automatic tools for algorithm tuning and selection in a parallel distributed environment.
- In order to also handle the huge complex linear systems encountered in electromagnetics (see Appendix), the packed storage described in Chapter 3 could be generalized to out-of-core calculations.

- Some further tracks related to the results of Chapter 4 would be interesting to follow. We could investigate extending our results to componentwise condition numbers. We could also perform a study on backward error analysis for the partial solution of an LLSP. When applied to linear systems, a first application of this "partial" backward error could be the validation of the averaging process used in electromagnetics with the Radar Cross Section (see Appendix) where the computed quantities are of the form $p^T A^{-1} p$.
- Regarding the application to GOCE, a next step would be the use of the QR parallel solver described in Chapter 3 in an operational mode for GOCE calculations. It could be also useful to realize a parallel implementation of the partial condition number studied in Chapter 4. Regularization techniques for ill-posed problems are only mentioned in this thesis. They deserve a more detailed study for solutions obtained via the QR approach (e.g L-curves [60]).

Appendix: application to electromagnetism

Motivation

In recent years, there has been a significant amount of work on the simulation of electromagnetic wave propagation phenomena, addressing various topics ranging from radar cross section to electromagnetic compatibility, to absorbing materials, and antenna design. To address these problems the Maxwell equations are often solved in the frequency domain [31, 97]. The discretization by the Boundary Element Method (BEM) results in linear systems with dense complex symmetric matrices [30, 77]. With the advent of parallel processing, solving these equations via direct methods has become viable for large problems and the typical problem size in the electromagnetics industry is on the increase. Nowadays, the usual problem size is a few tens of thousands. We may notice that there is no efficient parallel solver based on compact storage for symmetric dense complex matrices suited for the modelling of electromagnetic scattering and running on moderate processor configurations (less than 32 processors).

The numerical simulations we are interested in are performed in a daily production mode by the Electro-Magnetism and Control (EMC) Project at CERFACS.

We propose to use the parallel distributed Cholesky factorization described in Chapter 2 in order to solve the large, dense and symmetric linear systems in complex arithmetic resulting from boundary-element formulations for the solution of the 3D-Maxwell's equations. Many of these applications are still exploiting direct methods like LU (or sometimes LDL^T) factorization [2, 14].

By extending the factorization technique that we developed in real arithmetic to complex arithmetic, we save 50% of the storage compared with standard software considered in [2, 14]. Moreover, the computational cost of a $U^T U$ factorization is half that of a LU factorization ($\frac{n^3}{3}$ instead of $\frac{2n^3}{3}$ and with **complex** operations). Note that a LDL^T factorization with Bunch-Kauffman diagonal pivoting [50, p. 169] could have also be used but, as shown in [15], if there is no need for pivoting, a $U^T U$ factorization is as accurate and faster. A particular class of complex matrices of the form $A = B + iC$ with B and C both symmetric positive definite is mentioned in [61, p. 209] as having a normwise backward stable LU factorization without pivoting. Although the computations performed by the EMC project never required any pivoting, we could not justify the fact that the matrices resulting from BEM modelling are of the type described above. The properties of these matrices will need further investigations. However, in order to improve

the accuracy and stability of our solver, we implemented, similarly to [75], an iterative refinement functionality [61, p. 232]. In pratice, we did not need to use this possibility in our experiments.

Our solver enables us to switch the data type easily and to replace the LAPACK routine DPOTRF (that factors diagonal blocks) by a routine performing unblocked complex $U^T U$ factorization. Even if this routine is less efficient than the corresponding LAPACK routine ZPOTRF (which cannot be used here because it provides a $U^H U$ factorization), this has no significant effect on the factorization time since the number of operations involved in factorizing the diagonal blocks represent a very small part of total operations.

Numerical results

Experiments were conducted on eight processors of the HP-COMPAQ Alpha Server from CERFACS. The selected geometry is an aircraft with a mesh size of $18,264$ represented in Figure 4.1. The observed elapsed times were $762s$ for our $U^T U$ factorization and $1,310s$ for the ScaLAPACK LU factorization used in the code CESC [14] from the CERFACS EMC Project. We computed the corresponding scaled residual $\frac{\|A\tilde{x}-b\|}{\|b\|}$ and we obtained $2.75 \cdot 10^{-14}$ (for 64-bit double precision calculation). Since the physicists require a scaled residual of order 10^{-6}, iterative refinement was not activated.
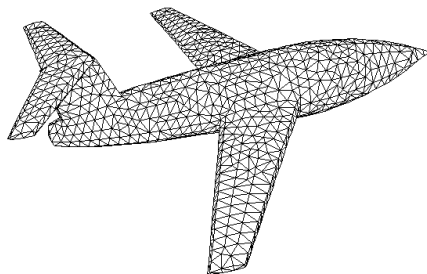

Figure 4.1: Mesh of aircraft test example for electromagnetism calculation.

Another validation performed by physicists consists in plotting the Radar Cross Section curve that represents the response of the object to the wave. Figure 4.2 represents the RCS curve for the aircraft depicted in Figure 4.1. We notice on the RCS that both curves coincide and this complies with the requirements of the physicists. However, the RCS can be viewed as an averaging process on the components and it does not fully guarantee the reliability of the solution. Thus it cannot replace the computation of the scaled residual $\frac{\|A\tilde{x}-b\|}{\|b\|}$ that is needed to warn (or confirm) the numerical quality of the computed solution.

Even if our algorithm involves half the number of operations required for an LU fac-
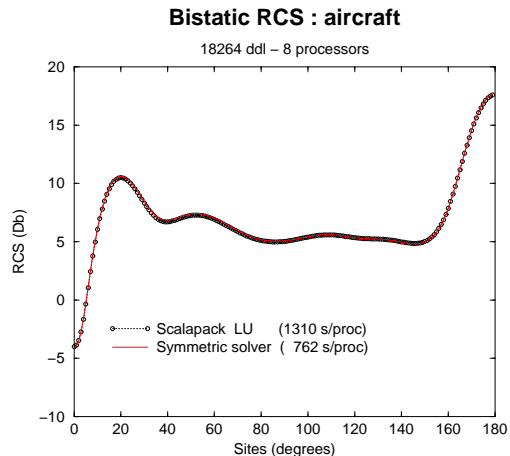

Bistatic RCS : aircraft

Figure 4.2: Radar Cross Section.

torization, we do not obtain a factorization time that is exactly half that of a LU. This relatively poor performance of a Cholesky factorization compared to a LU factorization can be explained by the fact that the ScaLAPACK LU factorization does not involve twice the number of message exchanges as our Cholesky factorization. In fact, at each step of the algorithm, the Cholesky factorization performed either by ScaLAPACK or our symmetric solver involves two communications as opposed to three for the ScaLAPACK LU. To illustrate this, we show in Table 4.7 the ratio between factorization times obtained by ScaLAPACK LU and our solver. These results have been obtained in complex arithmetic and the matrix size depends on the number of processors with the same rule of isogranularity detailed in Section 2.2.3 but taking here $n_1 = 6,600$ (corresponding to a constant storage of 700 Mbytes per processor for ScaLAPACK LU and 350 Mbytes for our left-looking solver). We notice that the performance ratio decreases when the number of processors grows and thus when there is a bigger impact of the communication on the global performance. We obtain similar ratios when we compare performance of ScaLA-PACK LU with that of ScaLAPACK Cholesky. Similar behaviour was observed by [33].

Table 4.7: Performance ratio between LU and $U^T U$ factorization (HP-COMPAQ Alpha).

nb procs	size	LU ScaLAPACK/Our solver
1	6600	1.95
2	9334	1.91
4	13200	1.81
8	18668	1.72
16	26400	1.62

Summary

The parallel distributed solver presented in Chapter 2 has been extended to solve efficiently complex linear systems encountered in electromagnetism. It enables us to save about half the memory required by the existing software. The solutions obtained meet the precision needed by the physicists and we can solve problems of a size corresponding to the target applications.

Bibliography

[1] *Basic Linear Algebra Subprograms Technical Forum Standard*, Int. J. of High Performance Computing Applications **16** (2002), no. 1.

[2] G. Alléon, S. Amram, N. Durante, P. Homsi, D. Pogarieloff, and C. Farhat, *Massively parallel processing boosts the solution of industrial electromagnetic problems: high-performance out-of-core solution of complex dense systems*, (1997), SIAM Conference on Parallel Processing for Scientific Computing.

[3] P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, , J. Overfelt, R. van de Geijn, and Y. Wu, *PLAPACK: Parallel Linear Algebra Libraries Design Overview*, (1997), Proceedings of Super Computing Conference 97.

[4] B. Andersen, J. Gunnels, F. Gustavson, J. Reid, and J. Waśniewski, *A fully portable high performance minimal storage hybrid Cholesky algorithm*, ACM Trans. Math. Softw. **31** (2005), no. 2, 201–207.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK user's guide*, SIAM, 1999, Third edition.

[6] E. Anderson and J. Dongarra, *Evaluating block algorithm variants in LAPACK*, Tech. report, 1990, LAPACK Working Note 19.

[7] M. Arioli, I. S. Duff, and P. P. M de Rijk, *On the augmented system approach to sparse least squares problems*, Numerische Mathematik **55** (1989), 667–684.

[8] M. Baboulin, L. Giraud, and S. Gratton, *A parallel distributed solver for large dense symmetric systems: applications to geodesy and electromagnetism problems*, Int. J. of High Performance Computing Applications **19** (2005), no. 4, 353–363.

[9] G. Balmino, *Le champ de gravité de la Terre et la mission GOCE*, (2004), Conférence invitée Alcatel-Space, Cannes.

[10] G. Balmino, *The European GOCE Gravity Consortium (EGG-C)*, (April 2001), 7–12, Proceedings of the International GOCE User Workshop.

[11] G. Balmino, S. Bruinsma, and J-C. Marty, *Numerical simulation of the gravity field recovery from GOCE mission data*, (8-10 March 2004), Proceedings of the Second International GOCE User Workshop "GOCE, The Geoid and Oceanography", ESA-ESRIN, Frascati, Italy.

[12] G. Balmino, A. Cazenave, A. Comolet-Tirman, J. C. Husson, and M. Lefebvre, *Cours de géodésie dynamique et spatiale*, ENSTA, 1982.

[13] B. Barotto, *Introduction de paramètres stochastiques pour améliorer l'estimation des trajectoires d'un système dynamique par une méthode de moindres carrés. Application à la détermination de l'orbite d'un satellite avec une précision centimétrique*, Ph.D. thesis, 1995, Université Paul Sabatier.

[14] A. Bendali and M'B. Fares, *CERFACS electromagnetics solver code*, Technical Report TR/EMC/99/52, CERFACS, Toulouse, France, 1999.

[15] N. Béreux, *A Cholesky algorithm for some complex symmetric systems*, Technical Report 515, Ecole Polytechnique, Centre de Mathématiques Appliquées, Palaiseau, France, 2003.

[16] G. J. Bierman, *Factorization methods for discrete sequential estimation*, Academic Press, 1977, Volume 128 in Mathematics in Science and Engineering.

[17] Å. Björck, *Iterative refinement of linear least squares solutions I*, BIT **7** (1967), 257–278.

[18] ——, *Iterative refinement of linear least squares solutions II*, BIT **8** (1968), 8–30.

[19] ——, *Methods for sparse linear least squares problems*, in Sparse Matrix Computations, J .Bunch and D. J. Rose, Academic Press (1976), 177–199.

[20] ——, *Stability analysis of the method of semi-normal equations for least squares problems*, Linear Algebra and its Applications **88/89** (1987), 31–48.

[21] ——, *Numerical methods for least squares problems*, SIAM, 1996.

[22] Å. Björck, T. Elfving, and Z. Strako š , *Stability of conjugate gradient and LANCZOS methods for linear least squares problems*, SIAM J. Matrix Analysis and Applications **19** (1998), no. 3, 720–736.

[23] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK user's guide*, SIAM, 1997.

[24] Y. Cao and L. Petzold, *A subspace error estimate for linear systems*, SIAM J. Matrix Analysis and Applications **24** (2003), 787–801.

[25] J.P. Carrou, *Mécanique spatiale - tome 1*, Cépaduès Editions, 1995.

[26] ——, *Mécanique spatiale - tome 2*, Cépaduès Editions, 1995.

[27] F. Chaitin-Chatelin and V. Frayssé, *Lectures on finite precision computations*, SIAM, Philadelphia, 1996.

[28] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan, and J. McDonald, *Parallel programming in openmp*, Morgan Kaufmann Publishers, 2000.

[29] S. Chandrasekaran and I. C. F Ipsen, *On the sensitivity of solution components in linear systems of equations*, Numerical Linear Algebra with Applications **2** (1995), 271–286.

[30] G. Chen and J. Zhou, *Boundary Element Methods*, Academic Press, New-York, 1992.

[31] W.C. Chew, J.-M. Jin, E. Michielssen, and J.M. Song, *Fast and efficient algorithms in computational electromagnetics*, Artech House, 2001.

[32] J. Choi, J. Dongarra, L. Ostrouchov, A. Petitet, D. Walker, and R. Whaley, *A proposal for a set of parallel basic linear algebra subprograms*, Tech. report, 1995, LAPACK Working Note 100.

[33] _____, *The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines*, Scientific Programming **5** (1996), 173–184.

[34] A. J. Cox and N. J. Higham, *Accuracy and stability of the null space method for solving the equality constrained least squares problem*, BIT **39** (1999), no. 1, 34–50.

[35] _____, *Backward error bounds for constrained least squares problems*, BIT **39** (1999), no. 2, 210–227.

[36] E. D'Azevedo and J. Dongarra, *Packed storage extension for ScaLAPACK*, Tech. report, 1998, LAPACK Working Note 135.

[37] J. Demmel and J. Dongarra, *Reliable and scalable software for linear algebra computations on high end computers*, (2004), Proposal for Software and Tools for High-End Computing (ST-HEC).

[38] J. Dongarra and E. D'Azevedo, *The design and implementation of the parallel out-of-core ScaLAPACK LU, QR, and Cholesky factorization routines*, Tech. report, 1997, LAPACK Working Note 118.

[39] J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Softw. **16** (1990), 1–17.

[40] J. Dongarra, I. Duff, D. Sorensen, and H. van der Vorst, *Numerical linear algebra for high-performance computers*, SIAM, 1998.

[41] J. Dongarra and R. Whaley, *A user's guide to the BLACS v1.1*, Tech. report, 1997, LAPACK Working Note 94.

[42] L. Eldén, *Perturbation Theory for the Least Squares Problem with Linear Equality Constraints*, SIAM J. Numerical Analysis **17** (1980), 338–350.

[43] M. Fengler, W. Freeden, and M. Gutting, *Multiscale modeling from EIGEN-1S, EIGEN-2, EIGEN-GRACE01S, UCPH2002_0.5, EGM96*, (2005), 145–150, Earth observation with CHAMP, Results from three years in orbit.

[44] Message Passing Interface Forum, *MPI : A message-passing interface standard*, Int. J. Supercomputer Applications and High Performance Computing, 1994.

[45] V. Frayssé, S. Gratton, and V. Toumazou, *Structured backward error and condition number for linear systems of the type* $A^*Ax = b$, BIT **40** (2000), 74–83.

[46] A. J. Geurts, *A contribution to the theory of condition*, Numerische Mathematik **39** (1982), 85–96.

[47] O. Glockner, *Gravitational field modelling from champ-ephemerides by harmonic splines and fast multipole technniques*, Tech. report, 2002, Contribution to first CHAMP Science Meeting.

[48] S. Goedecker and A. Hoisie, *Performance optimization of numerically intensive codes*, SIAM, 2001.

[49] G. H. Golub, P. Manneback, and P. L. Toint, *A comparison between some direct and iterative methods for certain large scale geodetic least squares problems*, SIAM J. Scientific Computing **7** (1986), no. 3, 799–816.

[50] G. H. Golub and C. F. van Loan, *Matrix computations*, The Johns Hopkins University Press, 1996, Third edition.

[51] S. Gratton, *On the condition number of linear least squares problems in a weighted Frobenius norm*, BIT **36** (1996), 523–530.

[52] J. F. Grcar, *Optimal sensitivity analysis of linear least squares*, Technical Report LBNL-52434, Lawrence Berkeley National Laboratory, 2003.

[53] ———, *Adjoint formulas for condition numbers applied to linear and indefinite least squares*, Technical Report LBNL-55221, Lawrence Berkeley National Laboratory, 2004.

[54] M. Gu, *Backward perturbation bounds for linear least squares problems*, SIAM J. Matrix Analysis and Applications **20** (1998), no. 2, 363–372.

[55] T. Gudmundsson, C. S. Kenney, and A. J. Laub, *Small-sample statistical estimates for matrix norms*, SIAM Matrix Analysis and Applications **16** (1995), 776–792.

[56] B. Gunter, E. Quintana-Ortí, R. van de Geijn, and T. Joffrain, *Parallel out-of-core LU and QR factorization*, (2004), ScicomP 10, Texas Advanced Computing Center, Austin.

[57] B. Gunter, W. Reiley, and R. van de Geijn, *Implementation of out-of-core Cholesky and QR factorizations with POOCLAPACK*, Tech. report, 2000, PLAPACK Working Note 12.

[58] B. Gunter and R. van de Geijn, *Parallel out-of-core computation and updating of the QR factorization*, ACM Trans. Math. Softw. **31** (2005), no. 1, 60–78.

[59] F. G. Gustavson, *New generalized data structures for matrices lead to a variety of high performance dense linear algebra algorithms*, (June 20-23, 2004), 11–20, Proceedings of PARA'04, Workshop on state-of-the art in scientific computing.

[60] P. C. Hansen, *Rank-deficient and discrete ill-posed problems*, SIAM, 1998.

[61] N. J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, 2002, Second edition.

[62] N. J. Higham and G. W. Stewart, *Numerical linear algebra in statistical computing*, The State of the Art in Numerical Analysis (1987), 41–57, A. Iserles and M. J. D. Powell, editors, Oxford University Press.

[63] R. A. Horn and C. R. Johnson, *Matrix analysis*, Cambridge University Press, 1985.

[64] J. E. Dennis Jr. and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*, SIAM, 1996.

[65] T. Kailath, A .H. Sayed, and B. Hassibi, *Linear estimation*, Prentice-Hall, 2000.

[66] E. D. Kaplan, *Understanding GPS : Principles and applications*, Artech House Publishers, Boston, 1996.

[67] R. Karlson and B. Waldén, *Estimation of optimal backward perturbation bounds for the linear least squares problem*, BIT **37** (1997), no. 4, 862–869.

[68] W. M. Kaula, *Theory of satellite geodesy*, Blaisdell Press, Waltham, Mass., 1966.

[69] C. S. Kenney and A. J. Laub, *Small-sample statistical condition estimates for general matrix functions*, SIAM J. Sci. Comput. **15** (1994), 36–61.

[70] C. S. Kenney, A. J. Laub, and M. S. Reese, *Statistical condition estimation for linear least squares*, SIAM Matrix Analysis and Applications **19** (1998), 906–923.

[71] Jet Propulsion Laboratory, *GRACE: Gravity Recovery and Climate Experiment - Science and Mission Requirements Document, revision A, JPLD-15928*, (1998), 1–84, NASA's Earth System Pathfinder Program.

[72] P. Lancaster and M. Tismenetsky, *The theory of matrices*, Academic Press Inc., 1985, Second edition with applications.

[73] P. Läuchli, *Jordan-elimination und Ausgleichung nach kleinsten Quadraten*, Numerische Mathematik **3** (1961), 226–240.

[74] C. L. Lawson and R. J. Hanson, *Solving least squares problems*, SIAM, 1974.

[75] X. Li and J. Demmel, *SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Math. Softw. **29** (2003), 110–140.

[76] A. Malyshev and M. Sadkane, *Computation of optimal backward perturbation bounds for large sparse linear least squares problems*, BIT **41** (2002), no. 4, 739–747.

[77] J.-C. Nédélec, *Acoustic and electromagnetic equations. Integral representations for harmonic problems*, vol. 144, Springer-Verlag, New-York, 2001.

[78] J. M. Ortega and C. H. Romine, *The ijk forms of factorization methods II. Parallel systems*, Parallel Computing **7** (1988), 149–162.

[79] C. C. Paige, *An error analysis of a method for solving matrix equations*, Mathematics of Computation **27** (1973), 355–359.

[80] C. C. Paige and M. A. Saunders, *Toward a generalized singular value decomposition*, SIAM J. Numerical Analysis **18** (1981), 398–405.

[81] ⸺, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Softw. **8** (1982), 43–71.

[82] R. Pail and G. Plank, *Assessment of three numerical solution strategies for gravity field recovery from GOCE satellite gravity gradiometry implemented on a parallel platform*, J. Geodesy 76:462-474 (2002).

[83] R. Pail, G. Plank, and G. Schuh, *Spatially restricted data distribution on the sphere: the method of orthonormalized functions and applications*, J. Geodesy 75:44-56 (2001).

[84] The Parallel Algorithms Project, *Scientific Report for 2002*, Technical Report TR/PA/02/124, CERFACS, Toulouse, France, 2002, http://www.cerfacs.fr/algor/reports.

[85] C. R. Rao and S. K. Mitra, *Generalized inverse of matrices and its applications*, Wiley, New York, 1971.

[86] Ch. Reigber, R. Bock, Ch. Forste, L. Grunwaldt, N. Jakowski, H. Lühr, P. Schwintzer, and C. Tilgner, *CHAMP: Phase B Executive Summary*, (1996), 1–37, G.F.Z., STR96/13.

[87] J. Rice, *A theory of condition*, SIAM J. Numerical Analysis **3** (1966), 287–310.

[88] J. L. Rigal and J. Gaches, *On the compatibility of a given solution with the data of a linear system*, J. Assoc. Comput. Mach. **14:3** (1967), 543–548.

[89] G. W. Stewart, *Afternotes on numerical analysis*, SIAM, 1996.

[90] G. W. Stewart and Jiguang Sun, *Matrix perturbation theory*, Academic Press, New York, 1991.

[91] Zheng Su, *Computational methods for least squares problems and clinical trials*, Ph.D. thesis, 2005, Stanford University.

[92] H. Sünkel, *From Eötvös to milligal+, Final Report*, ESA/ESTEC Contract No. 13392/98/NL/GD, Graz University of Technology, 2000.

[93] G. Sylvand, *La méthode multipôle rapide en électromagnétisme : performances, parallélisation, applications*, Ph.D. thesis, 2002, Ecole Nationale des Ponts et Chaussées.

[94] The MathWorks, *Matlab reference guide*, The MathWorks Inc., Natick, MA, 1992.

[95] A. N. Tikhonov, *Regularization of incorrectly posed problems*, Soviet Math. **4** (1963), 1624–1627.

[96] A. Turing, *Rounding-off errors in matrix processes*, Quart. J. Mech. and Applied Math. **1** (1948), 287–308.

[97] J. Van Bladel, *Electromagnetic fields*, revised printing ed., Hemisphere Publishing Corporation, New-York, 1985.

[98] R. van de Geijn, *Using PLAPACK*, The MIT Press, 1997.

[99] C. F. van Loan, *Generalizing the singular value decomposition*, SIAM J. Numerical Analysis **13** (1976), 76–83.

[100] B. Waldén, R. Karlson, and J. Sun, *Optimal backward perturbation bounds for the linear least squares problem*, Numerical Linear Algebra with Applications **2** (1995), no. 3, 271–286.

[101] P.-Å. Wedin, *Perturbation theory for pseudo-inverses*, BIT **13** (1973), 217–232.

[102] Yimin Wei, Huaian Diao, and Sanzheng Qiao, *Condition number for weighted linear least squares problem and its condition number*, Technical Report CAS 04-02-SQ, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, 2004.

[103] Yimin Wei, Wei Xu, Sanzheng Qiao, and Huaian Diao, *Componentwise condition numbers for generalized matrix inversion and linear least squares*, Technical Report CAS 03-12-SQ, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, 2003.

[104] J. H. Wilkinson, *Rounding errors in algebraic processes*, vol. 32, Her Majesty's Stationery Office, London, 1963.

[105] ———, *A priori error analysis of algebraic processes*, (1968), 629–639, in Proceedings International Congress Math., Izdat. Mir, Moscow.

[106] S. Wolfram, *Mathematica, a system for doing mathematics by computer*, Addison-Wesley Publishing Company, 1988.