# Determining Rotational Elements of Planetary Bodies

## Method and Implementation of an Inertial Frame Bundle Block Adjustment

vorgelegt von
Dipl.-Math.
Steffi Burmeister
geb. in Schwerin

von der Fakultät VI – Planen Bauen Umwelt
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktorin der Naturwissenschaften
– Dr.-rer.-nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Frank Neitzel
Gutachter: Prof. Dr. Jürgen Oberst
Gutachter: Prof. Dr. Jürgen Kusche

Tag der wissenschaftlichen Aussprache: 27. Januar 2017

Berlin 2017

**Acknowledgement**

I'd like to thank Professor J. Oberst for the opportunity to write a PhD thesis and do research on a fascinating and inspiring topic, for the gained insight in the field of planetary geodesy and the knowledge that I was able to gather during the past four years of work under his supervision.
I am also grateful to my colleagues Dr. K. Willner, A. Pasewaldt and to F. Preusker for various discussions and especially for providing the data sets of Phobos and Vesta used for this work.
Finally I thank Simon Fear for his clear opinion about tables, and my family and friends for their curiosity, encouragements and kindness.

# Abstract

The rotational elements of a planetary body state its orientation in space with respect to the International Celestial Reference Frame (ICRF). They are used for computing control point networks (CPNs) and for creating e.g. maps or terrain models of the body. This work describes the *inertial frame bundle block adjustment*. It is a method to determine rotational elements of planetary bodies in a direct and analytical way from image point measurements. Simultaneously with the rotational parameters, a corresponding CPN is computed and the external orientation data of the camera are improved. In contrary to the classical photogrammetric bundle block adjustment, here the interdependence of the body's rotational model and the camera's external orientation is resolved. The method was tested on simulated data and applied to two real science cases. The amplitude of forced libration, which contributes to the orientation of the prime meridian, was determined for the Martian moon Phobos. Using combined images of the missions Viking and Mars Express, an amplitude of $1.13^\circ \pm 0.13$ was computed together with a CPN of 680 points and an uncertainty of $\varnothing$ 55 m. The amplitude is in agreement with previous results, derived by indirect methods (e.g. Willner et al., 2010; Oberst et al., 2014). Secondly, the pole axis orientation of Vesta was confirmed using images of the current Dawn mission. The computed CPN of 82 806 points is compared with the body-fixed solution of Preusker et al. (2012), showing that both networks are plausible within the average intersection error of 10 m. The software which was implemented is focusing on the application to large data sets. It could be achieved that the costs of memory and running time depend only on the number of images. In the example of Vesta with 5440 images, 164 hours (out of 168) are saved with respect to a reference adjustment software. The method is suitable to study and refine rotational models of bodies with solid surfaces.

adjustment, bundle block adjustment ICRF, forced libration amplitude, Phobos, Vesta, parallel inversion

# Zusammenfassung

Die Rotationselemente eines planetaren Körpers geben seine Orientierung im Raum in Bezug auf den internationalen Himmelsreferenzrahmen (engl. ICRF) an. Sie sind notwendig für die Berechnung von Kontrollpunktnetzen (KPN) und für die Erstellung von z.B. Karten oder Geländemodellen des Körpers. Diese Arbeit beschreibt den *Bündelblockausgleich im inertialen Referenzrahmen*. Das ist eine Methode, um Rotationselemente planetarer Körper direkt und analytisch anhand photogrammetrischer Messungen zu bestimmen. Simultan wird ein KPN berechnet und die externen Orientierungen der Kamera werden verbessert. Im Gegensatz zum klassischen körperfesten Bündelblockausgleich ist hier die bestehende Abhängigkeit zwischen dem Rotationsmodell des Körpers und der Kameraorientierung aufgelöst. Die Methode wurde anhand einer Simulation getestet und auf zwei reale Datensätze angewandt. Die Amplitude der erzwungenen Libration, die sich auf die Orientierung des Hauptmeridians auswirkt, wurde für den Marsmond Phobos bestimmt. Mittels Bilddaten aus den Missionen Viking und Mars Express ist eine Amplitude von $1,13°$ ($\pm 0,13$) berechnet worden, das zugehörige KPN mit 680 Punkten hat eine Unsicherheit von $\varnothing$ 55 m. Die Amplitude stimmt mit den Ergebnissen von Willner et al. (2010) und Oberst et al. (2014), die durch indirekte Methoden erhalten wurden, überein. Die Polachsenorientierung von Vesta wurde mittels Bilddaten aus der aktuellen Dawn Mission bestätigt. Das berechnete KPN mit 82 806 Punkten wurde verglichen mit dem Ergebnis von Preusker et al. (2012) aus einem körperfesten Bündelblockausgleich; beide Lösungen sind innerhalb des mittleren Schnittfehlers von 10 m plausibel. Die Software wurde mit Schwerpunkt auf große Datensätze implementiert. Dabei konnte erreicht werden, dass die Kosten für Speicher und Laufzeit nur von der Anzahl der Bilder abhängig sind. Beim Vesta-Beispiel mit 5440 Bildern werden im Vergleich zu einer Referenz-Software 164 von 168 Stunden eingespart. Die Methode ist geeignet, um Rotationsmodelle von planetaren Körpern mit fester Oberfläche zu studieren und zu verbessern.

Ausgleichsrechnung, Bündelblockausgleich ICRF, Librationsamplitude, Phobos, Vesta, parallele Inversion

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation and content

Since the final decades of the 20th century, spacecraft have been launched to study objects in the solar system. In many cases the spacecraft carries an imaging system to gather optical information about the target. For this thesis the focus is set on planetary bodies with solid surfaces - that means planets, dwarf planets, moons, asteroids or comets which are suitable for a photogrammetric survey. In the field of planetary geodesy images are used to study properties of planetary bodies, e.g. topography, shape and rotation. Major objectives of these studies are the definition of reference systems, the determination of rotational parameters as well as mapping and modelling of planetary surfaces. This includes in particular the computation of control point networks (CPNs) which are a first-order realisation of the body's reference system and define the *body-fixed* reference frame by three-dimensional coordinates. While a body moves in space, it changes its orientation e.g. by self-rotation, the precession of the pole axis or librations. These changes in orientation are described by the rotational model of the body in terms of functions of time, formulated with respect to an inertial reference system. These functions indicate the direction of the pole axis as well as the orientation of the body's prime meridian. Any error in the rotational model contributes to a misalignment between optical measurements and the corresponding ground coordinates. Hence, in order to create stable maps of high quality, a precise knowledge of the rotational parameters is desired.

In the past century spin poles, rates and sense of rotation of the large asteroids were mostly determined by earth-based measurements of lightcurves and an analysis which included a theoretical shape model. Magnusson (1986) gives a good overview about the applied methods and the literature of that era. He consolidated some of the common methods and proposed a procedure which is here referred to as *(parameter) range scan method*. There, a series of analyses with trial values for the parameters to be determined is performed, and the data are statistically evaluated such that a best-fit yields the final result. This approach, gradually improved e.g. by De Angelis (1993), was widely used. With the available image data of the Hubble Space Telescope Camera and the spacecraft missions, the trial values are typically evaluated with respect to a computed CPN or the related improvements of image observations (Thomas et al., 1995). A special effect, that has been verified for a few moons in the solar system, is the forced libration of satellites. This small variation in the mean self rotation rate depends on the satellite's moments of inertia and the gravitational torque that a primary body exerts on its satellite. The amplitude of the forced libration contributes to the orientation of the prime meridian and is therefore an important rotational parameter. An angular error in the model about the prime rotation axis will lead to a positional error of the control points; this error is proportional to the size of the body. Furthermore, since the magnitude of this libration is related to the magnitude of the gravitational torque, conclusions about the internal structure of a body can be drawn (Murray and Dermott, 1999). By applying the range scan method with a bundle block adjustment or similar type of CPN analysis, amplitudes of forced libration have been obtained for the Martian moon Phobos (Duxbury and Callahan, 1989; Willner et al., 2010, a.o.) as well as for the Saturnian moons Janus, Epimetheus (Tiscareno et al., 2009) and Mimas (Tajeddine et al., 2013). An alternative method has been introduced by Oberst et al. (2014) who used a rotation model of Phobos without longitudinal libration and evaluated the spacecraft position residuals over the anomalistic period[1] after a bundle block adjustment. With respect to Phobos also a theoretical approach has been used, where the equations of motion are solved by numerical integration and the magnitude of forced libration could be determined as one of the

[1] pericenter to pericenter

orbit parameters (Jacobson, 2010; Rambaux et al., 2012).

It is important to pay attention to the interdependency of rotational elements and the computed CPN of a planetary body. The initial result of the CPN, that is derived from the image measurements, reflects errors in the measurements themselves and in the camera's external orientation (i.e. the position and pointing data of the camera). In order to obtain a reliable network, the camera orientation parameters need to be improved. The classic approach to minimise such errors is the mentioned bundle block adjustment, an iterative procedure that finds a Least-Squares Regression solution for the underlying mathematical model. This is a powerful and well-proven tool in terrestrial geodesy. In planetary geodesy the orientation of the target is generally not well-known, esp. not fixed with respect to the observer. The orientation of the camera reference frame, e.g. obtained by star tracking, is given with respect to an inertial reference frame. Also the positions are given in inertial coordinates; they are obtained by earth-based tracking methods (e.g. range and Doppler measurements) and eventually by further numerical integration of the spacecraft's trajectory. Based on the assumed rotational model of the target body, the position vectors and pointing angles are converted into orientation data with respect to the body-fixed frame and this way, they are linked to the rotational elements. The classical photogrammetric bundle block adjustment, designed for geodetic measurements on Earth, cannot resolve this fundamental dependency since the underlying mathematical model is formulated in the body-fixed reference frame only. Errors in the rotational model lead to large position offsets and falsified pointing data in the body-fixed frame. I.e. the (body-fixed) orientation uncertainties include the uncertainties of rotational elements of the observed body, too. Therefore, a series of CPNs is computed together with a set of statistic information which yield the basis of decision with respect to the most reliable solution.

The methods to determine rotational elements today are indirect methods. The range scan method requires a large number of adjustments, one for each assumed value of the rotational parameter in question. Hence, it is only suitable for small data sets of a few hundred images. As Thomas et al. (1997) could determine the spin pole of Vesta with very few images, the precession movement, recently indicated by theoretical considerations of Konopliv et al.

(2014), could not be detected by this approach. In order to cover long-time periods of rotational behaviour as well as the surface of larger planetary bodies, larger data sets need to be formed and processed. When this is done, the non-linear increasing computational effort stresses the memory capacity, running time and hardware expenses in such a way, that even for a single adjustment an effective resource management is required. The range scan method in these cases is no longer practicable.

It is an aim of this work to present a method that allows the direct determination of rotational elements based on an analysis of image data. This method will be referred to as inertial frame bundle block adjustment, since the underlying mathematical equations are formulated with respect to the inertial frame. In order to avoid a possible misunderstanding the expression *direct computation* shall be clarified. In contrary to the previously mentioned techniques which yield their results indirectly and by interpretation, within the inertial frame bundle block adjustment the (time-independent) rotational parameters become directly adjustable and it takes only the iterations of a single adjustment to obtain the result. To exemplify the new method, two planetary objects have been chosen: Phobos, the inner moon of Mars and the protoplanet Vesta, which is located in the main asteroid belt. The priority here is the forced libration amplitude of Phobos and Vesta's mean pole axis orientation. The second objective of this work is the performance optimisation of the implemented software with respect to the computation of large CPNs.
In order to describe planetary bodies and their orientation in space, the concepts of reference frames and rotational elements have been developed and continuously improved. The current chapter provides an introduction to the topic and its application to the reconstruction of a body's surface from image data. It also includes an overview about the numerical tasks, related existing solutions and the involved missions and image data. More technical information and concrete formula will be given in chapter 2 after the statement of the mathematical notation and foundations. There also the classical method of a photogrammetric bundle block adjustment is described. This method will then be extended to an inertial context in chapter 3, where the own work contribution starts. Subject of chapter 4 are all aspects of implementation, hardware

and software considerations focusing the challenge of large data sets. The new concepts and methods are then applied to the science cases Phobos and Vesta in chapter 5. Chapter 6 includes the summary and an overview about future goals and further software development.

## 1.2 Reference frames

The motion and orientation of bodies in the solar system, being natural or artificial, and specific locations on a certain body are formulated in various different reference systems. By convention, in planetary science these are right-handed orthogonal coordinate systems. It proved to be practical to work with 3D cartesian coordinates for locations and with angular expressions to describe orientations at a given time. Hence, reference systems are defined by specifying three principle axes (in right-handed order) and the origin. Coordinate systems of moving and rotating objects are associated with a reference epoch. The current standard epoch is J2000.0 (1st January 2000, 12:00:00 Barycentric Dynamical Time) and time is measured in SI seconds with respect to that date. It is furthermore distinguished between the abstract level of reference systems and the concrete level of reference frames. While a reference system is given by definition, the respective reference frame is its concrete realisation by coordinate values. The coordinate system which is given by the frame respects the definitions of the reference system as close as possible and is decisive for the description of the object in question. The following subsections will describe all coordinate systems which are relevant for this work and how they are related to each other. Figure 1.2 shows an exemplary scheme of a spacecraft with a mounted camera and a target body which are moving in space. The explicit mathematical definition of the frame conversions follows in section 2.2.

### 1.2.1 International Celestial Reference Frame (ICRF)

The inertial reference system which is taken into account for this work is the International Celestial Reference System (ICRS). Its origin is the solar system barycentre and the three principle axes are not rotating. In 1998 with the introduction of the International Celestial Reference Frame (ICRF) this idealised system was realised first-time. Using radio interferometry with very long base-

lines (VLBI), the ICRF is defined through the positions of extragalactic radio sources (mainly quasars) that move only marginal, relative to our solar system (Ma et al., 1998). Out of the 608 listed sources, grouped according to different criteria of quality, 212 *defining sources* were chosen to specify the frame (see figure 1.1). According to Ma et al., the alignment uncertainty of the principle axes is less than 0.02 milliarcseconds (compared with the ICRS axes). As a

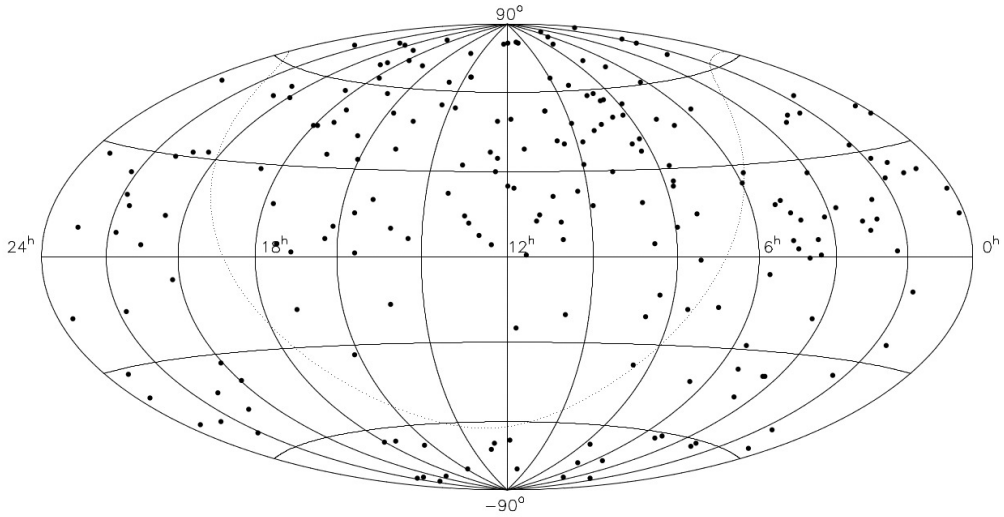



Figure 1.1: Distribution of the ICRF defining sources on the celestial sphere (Ma et al., 1998, fig. 10)

consequence the ICRF itself is epochless. All other types of reference frames can be related directly or in subsequent order to the ICRF. The inertial frame is used to describe not only the motion of an object within the frame, but also its orientation relative to the (stable) principle axes. At a given time, any reference frame can be described with respect to the ICRF by specifying the orientation of both frames relative to each other. The International Earth Rotation Service (IERS) who monitors the radio sources released an update of the frame (Ma et al., 2009) which was adopted by the International Astronomical Union (IAU) as of 1st January 2010. The update is officially referred to as ICRF2 and the original frame as ICRF1. Since the update is without any consequence to the rotational models, the distinction is dropped here.

### 1.2.2 Hipparcos Celestial Reference Frame (HCRF)

The inertial reference system ICRS is also realised at optical wavelengths by the Hipparcos Celestial Reference Frame (HCRF), consisting of 85% of the stars in the Hipparcos catalog. According to Kaplan (2005) the position errors of these stars in ICRF coordinates with respect to J2000.0 range between 5 and 10 milliarcseconds (mas). The orientation of the spacecraft and camera reference frame with respect to the ICRF is typically obtained by star tracking, i.e. it is based on optical observation of stars. Hence, this frame is used by services which provide the orientation data of spacecraft cameras, e.g. NASA's Navigation and Ancillary Information Facility (NAIF).

### 1.2.3 Spacecraft and camera reference frame

**Hierarchy of reference frames with ICRF as fundamental frame**

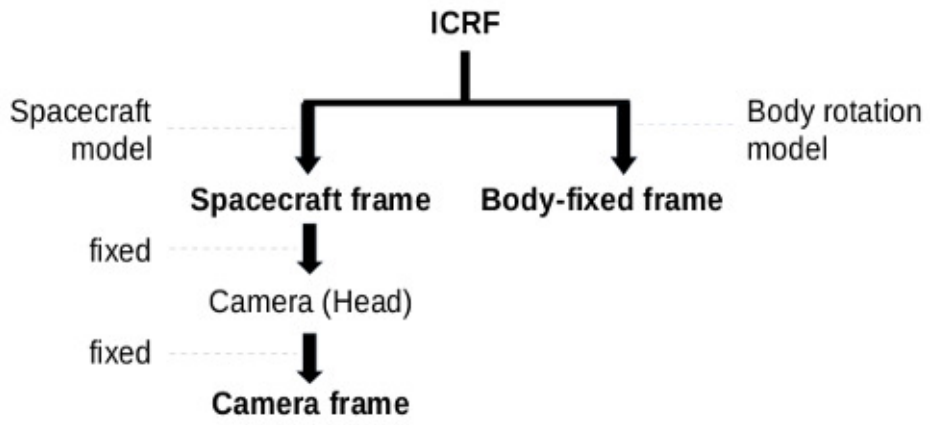



Figure 1.2: Orientation and motion of spacecraft and body are formulated within the ICRF. The conversion between the frames is accomplished by the (possibly successive) application of rotation matrices. In this example, the transformation between spacecraft and camera frame is constant.

The spacecraft motion is described in inertial coordinates, but it has an attached local frame which is pre-defined by the mission team. The origin in ICRF coordinates is equal to the position of the probe. Navigation and stabilisation of the flight system's orientation is achieved by the attitude con-

trol subsystems onboard (Doody, 2011). In the frame hierarchy, the camera reference frame is subordinate to the spacecraft frame (see figure 1.2). There is a special platform (called instrument head) on which the camera system is mounted. It has a fixed orientation with respect to the coordinate system used for the spacecraft. The camera itself has a certain alignment with respect to the platform and hence to the spacecraft. This alignment is either fix or depends on pre-defined rsp. commanded moves of the camera. In praxis, the mounting is often neglected, i.e. the position of camera and spacecraft is taken as identical.

Here only CCD frame cameras are considered and the object which needs to be described is the image that the camera has captured at a certain time. The reference frame is defined by the coordinates of the CCD sensors (pixel) which lay in a plane behind the camera lens. The origin of the camera frame is the focal point. The direction from the images' principle point to the lens is the bore side direction and the distance between them is the (nominal) *focal length*. The information about an instrument on board of a spacecraft is collected in the *instrument kernel* (IK). The IK contains all required information to transform pixel coordinates into metric coordinates (focal length, the number of valid pixels in line and sample direction, the coordinates of the principle point, the pixel size etc.), including the definition of the three primary axes. The assignment of the axes varies for different instruments rsp. missions, but typically one axis contains the bore side vector and the other axes are parallel to the rectangular CCD boundaries.
Example: $+Z$ points in bore side direction, then the $X$-$Y$-plane is parallel to the CCD – all pixels have the same $Z$-coordinate which agrees with the negative focal length. If $+X$ points from left to right (watching from $+Z$), $+Y$ points from top to bottom.
The measurement of the image point observations is accomplished within the coordinate system of the image. In order to compute three-dimensional (3D) surface coordinates of a target body, they are transformed into 3D centred and metric coordinates of the camera reference frame. Furthermore the orientation of the frame including its position needs to be known. All camera orientation data used in this work are either directly obtained by the SPICE kernels of NAIF (Acton, 1996) or have been originally provided by them and

where modified later on.

### 1.2.4 Body-fixed reference frame

The description of the location of a crater on a body, constantly moving and changing its orientation with respect to the inertial frame, requires a coordinate frame that is tied to the body. Content of this subsection are the recommendations and conventions given by the IAU *Working Group on Cartographic Coordinates and Rotational Elements* (Archinal et al., 2011).
An object-centred reference system is defined with respect to its mean axis of rotation and an additional choice of the zero-longitude that defines the direction of the X-axis. The corresponding reference frame is called *body-fixed* reference frame and its main purpose is to facilitate mapping. The frame is defined by control points with the origin commonly pinned to the center of mass.
For planets and moons, the $Z$-axis is identified with the mean rotation axis containing the north pole of rotation on the positive side.[2] If there is no other information the axis is assumed to be normal to the mean orbital plane. For the catagory of small bodies the positive pole of rotation is defined by the spinning direction of the body: looking onto +Z the body appears to rotate anti-clockwise. The term *north pole* is not used here, since there are cases where the spin pole moves in a short time between north and south of the invariable plane. As a consequence only planets and moons can have retrograde revolutions; Pluto being formerly a retrograde rotating planet is considered a prograde rotating dwarf planet with its positive pole south of the invariable plane. According to the recommendations for small irregular bodies, the initial definition of a body-fixed coordinate system should be, if possible, based on a shape model and estimate of the moments of inertia (Archinal et al., 2011).
The prime meridian can be chosen in principle as arbitrary as on Earth, but the IAU recommends to align it with the longest axis or the minimum moment of inertia. Often a landmark is used, but the prime meridian can be defined by reference to another celestial object, too. On images topographic features, typically small craters, are identified and a candidate is chosen that defines a

[2]The pole that lays north of the invariable solar plane, as it is for Earth

specific longitude. E.g. on Mercury the prime is defined with respect to Hun Kal which marks the meridian 20° W (Robinson et al., 1999).
The body-fixed coordinates are considered as independent of the self-rotation and the body's orientation in space. A reorientation of the spin axis in terms of polar motion, also known as true polar wander (Matsuyama et al., 2006; Harada, 2012), is not considered in this context. Furthermore, possible changes of the topography, e.g. by tidal deformation, are not taken into account since they are expected small enough to be neglected.

## 1.3 Rotational elements

The rotational elements of a planetary body are used to describe the orientation of its body-fixed reference frame with respect to the ICRF at a given time. The specific directions of the principle axes are recorded with respect to the standard epoch.
The celestial coordinates $\alpha$ (right ascension) and $\delta$ (declination) describe the direction of the pole axis, from the center to the north pole (rsp. the positive pole). This vector is given in ICRF polar coordinates and is referred to as pole axis orientation. In addition the angle $W$ denotes the position of the body's prime meridian, relative to the intersection of the body's equatorial plane and the ICRF equator (see figure 1.3). The node $Q = (\alpha + 90°, 0°)$ coincides with $(W, 0°)$ in the body-fixed frame (measured as positive west) and $W$ is referred to as prime meridian orientation.
The angles $\alpha, \delta, W$ are the time-depending functions of the body's *rotational model* and include further models of effects like precession, acceleration or forced libration. Three subsequent rotations align the ICRF with the reference frame of the body. A detailed description of this conversion is given in section 2.2 (page 27). It is worth to point out, that in praxis only the rotational elements are used to describe the principle axes of the body-fixed coordinate system although there might be a defining CPN. The rotational model implies the coordinates of the feature that defines the zero-longitude. Hence, when rotational elements are improved, the original definition of the prime meridian should be respected (Archinal et al., 2011). In the following, the two rotational models of Phobos and Vesta will be given. It is $t = s/86\,400$ the magnitude of

**The orientation of a body with respect to the ICRF**

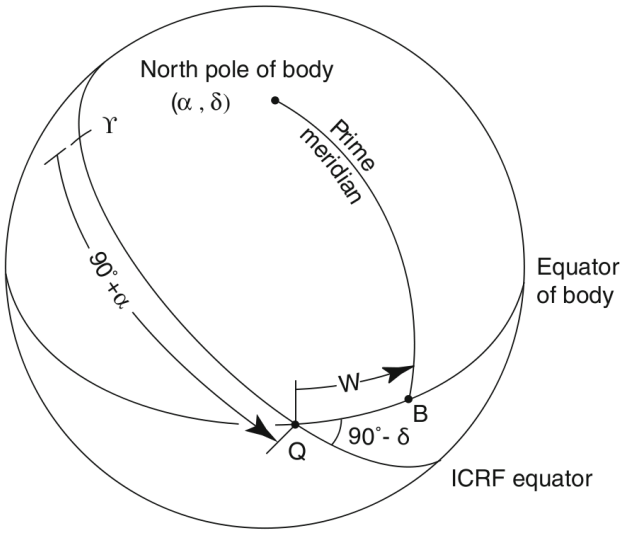



Figure 1.3: The pole axis orientation $(\alpha, \delta)$ states right ascension and declination of the body's Z-axis; $W$ is the prime meridian orientation w.r.t. the node $Q$. The longitude $L$ of $Q$ in both reference frames is: $L_{ICRF} = 90° + \alpha$ (pos. east) and $L_{BF} = W$ (pos. west). Figure, according to Archinal et al. (2011), is modified to match current notation.

days since J2000.0 (without unit) and $T = 36\,525\,t$ equals one Julian century; dates before the standard epoch result in negative values.

**Phobos** (Archinal et al., 2011):

$$\begin{aligned} \alpha(t) &= \alpha_0 - 0.108^\circ\,T + 1.79^\circ \sin(M_1) \\ \delta(t) &= \delta_0 - 0.061^\circ\,T - 1.08^\circ \cos(M_1) \\ W(t) &= 35.06^\circ + 1128.844585^\circ\,t + 8.864^\circ\,T^2 - 1.42^\circ \sin(M_1) - \lambda_0 \sin(M_2) \end{aligned} \tag{1.1}$$

with the time-independent parameters

$$\alpha_0 = 317.68^\circ, \qquad \delta_0 = 52.90^\circ, \qquad \lambda_0 = 0.78^\circ\ .$$

The function $M_1 = M_1(t) = 169.51^\circ + 0.435764^\circ\,t$ models the precession motion of 826 days and $M_2 = M_2(t) = 192.93^\circ + 1128.40967^\circ\,t + 8.864^\circ\,T^2$ the orbital motion. The similarity of $M_2$ and $W$ reflects that Phobos is in synchronous orbit around Mars. The second term in the definition of $W$ is the spin rate in degrees per day (one revolution takes about 7 h 39 min) and the + sign indicates the direct sense of rotation. The third term models the spin acceleration which is caused by Phobos' gradual approach to Mars. The coordinates ($\alpha_0$, $\delta_0$) describe the axis of precession and $\lambda_0$ is the amplitude of forced libration. In section 1.4 alternative results of $\lambda_0$ are given.

**Vesta** (Archinal et al., 2013; Li and Mafi, 2013):

$$\begin{aligned} \alpha(t) &= 309.031^\circ \pm 0.01 \\ \delta(t) &= 42.235^\circ \pm 0.01 \\ W(t) &= W_0 + 1617.3329428^\circ\,t\ , \quad W_0 = 285.39^\circ\ . \end{aligned} \tag{1.2}$$

The model above contains a fixed pole axis orientation and the rotation rate of 5 h 20 m 31.662 s. The coordinate system with $W_0 = 285.39^\circ$ is called *Claudia Double-Prime* system, it assigns a longitude of 146° W to the crater named Claudia such that the prime meridian passes through the feature named *Olbers Regio* as defined by Thomas et al. (1997). There is an alternative coordinate system with $W_0 = 75.39^\circ$ used by the Dawn team before. It is known as *Dawn-Claudia* system and assigns a longitude of 356° W to Claudia. Hence both systems are rotated by 210 degree with respect to each other. Table 1.1 shows a selection of earlier solutions of Vesta's rotational elements.

**Earlier rotational models of Vesta**

| publication | $\alpha$ | $\delta$ | rotation rate | |
|---|---|---|---|---|
| Gehrels (1967) | 191.1° | 75.1° | $5h\,20m\,31.665s$ | *proposed* |
| | 304.4° | 48.7° | $5h\,20m\,31.171s$ | *dismissed* |
| Magnusson (1986) | 307.3° | 38.5° | $5h\,20m\,31.646s$ | |
| Thomas et al. (1997) | 301.0° | 41.0° | $5h\,20m\,31.664s$ | |
| Li et al. (2010) | 305.8° | 41.4° | – | |

Table 1.1: Selected solutions of the decades prior to Dawn. The pole coordinates ($\alpha$, $\delta$) are w.r.t. J2000 (values before 1997 are converted). Gehrels dismissed the 2nd solution because of a larger error in the rotation rate.

## 1.4 Control point network analysis

The introduced concepts of reference frames and rotational elements are required to obtain the 3D coordinates of surface points, the so called control point network (CPN) of a body. As mentioned before, a CPN is the coordinate-based realisation of the body-fixed reference system and corresponds to a unique set of rotational parameters. In fact, when the range scan method is applied with a series of bundle block adjustments, both – the rotational elements and the CPN – are obtained simultaneously. In this section an overview about CPNs and the related analysis is given. This includes a brief introduction in how they can be computed and statistically evaluated. To avoid confusion, it is helpful to think of first-order CPNs, defining the body-fixed reference frame, and second-order CPNs which are obtained by using this reference frame. Those can be considerable larger than and also be completely different from the defining network.

The coordinates of a control point (in the body-fixed reference frame) are given by the intersection of multiple rays which connect the point with its image representation (in the camera reference frame). The underlying *co-linearity equations* involve a rotation matrix which converts the camera reference frame to the body-fixed reference frame at image time. Hence, the orientation of both reference frames with respect to the ICRF need to be known at the specific time (see frame hierarchy in figure 1.2). Merged to a single rotation matrix,

also the uncertainties of both frames merge and possible errors affect the initial result of the intersection points. By the procedure of the bundle block adjustment, where the equations are solved for all control points simultaneously, the intersection errors and the camera orientation errors can be reduced. The knowledge of the uncertainties is very important, because of its major influence on the result. It is taken into account for the calculation in terms of a stochastic model which is, briefly stated, a weighting rule for the observations. A different stochastic model will yield a different result. Hence, in order to reproduce the results, the uncertainties of all involved observations must be specified.

An important outcome of the analysis is the stochastic model aposteori, which states the uncertainties of a) the computed control points, b) the improved orientation data of the camera and c) the image point observations. Based on the weighted residuals of all observations an overall system error is computed. All of these quantities can be taken into account for decision when a parameter range scanning is applied. E.g. Tiscareno et al. (2009) analysed the difference between the original image point measurements and their predicted positions by the functional model after the adjustment and found a forced libration amplitude ($\lambda_0$) of $0.3° \pm 0.9$ for Janus and $5.9° \pm 1.2$ for Epimetheus. Tajeddine et al. (2013) used a similar evaluation method for Mimas and determined $\lambda_0 = 0.805° \pm 0.022$. They applied different rotation models for the backward projection into the images, but here the camera positions were corrected with the limb fit method and the pointing angles were kept fixed. Concerning Phobos Willner et al. (2010) obtained the forced libration amplitude in correspondence with the minimal mean error of the CPN; table 1.2 shows their solution as well as those of other selected authors.

The following two examples of CPN usage are related to a fixed object-centred reference frame. In the process of creating a digital terrain model (DTM), consisting of millions of points, also a bundle block adjustment is deployed, but to a much smaller subset of control points. Here, the improved camera orientation and their computed uncertainties are the most relevant results of the analysis and the CPN is only a by-product. The selected control points work as inner constrains of the network to adjust the orientation data of the camera. The resulted positions and pointing angles are kept fix when the final number of

**The forced libration amplitude of Phobos**

| $\lambda_0$ | | Publication | Method |
|---|---|---|---|
| 0.80° | ± 0.20 | Duxbury and Callahan (1989) | CPN |
| 1.24° | ± 0.15 | Willner et al. (2010) | CPN |
| 1.09° | ± 0.10 | Oberst et al. (2014) | CPN |
| 1.03° | ± 0.22 | Jacobson (2010) | O |
| 1.10° | – | Rambaux et al. (2012) | O |

Table 1.2: Selected results of Phobos' forced libration amplitude $\lambda_0$, obtained either by CPN analysis or by analysing the orbital equation of motion (O)

points is computed subsequently. The coordinates of the CPN can also be used to compute the coefficients of surface spherical harmonics, functions which approximate the whole surface and the principle axes of the frame. Apart from the resulting uncertainty, this alternative representation does not depend on the original measurements any more, but yields a global shape model. Large networks can also directly be triangulated and rendered to model the surface, examples of the science cases Phobos and Vesta are given in chapter 5.

## 1.5 Implementation

Deploying a bundle block adjustment to large networks requires a lot of computation power and the resources running time and memory need to be considered. The memory restriction of a computer is a hard limit and decides whether or not a certain program can be executed. The running time also contributes to qualitative aspects. Considering common block adjustments with one fixed rotational model, computation times can take several weeks and months including pre- and post-processing (private communication with F. Preusker). Since time itself is a critical resource in scientific projects, data sets might be reduced in order to publish results in time. Here, matrix products and matrix inversion are the most time expensive parts of the program. Compression techniques and an intelligent implementation design are keys to an efficient management of both resources. In chapter 4 the topic of sparse matrix compression and the advantage of symmetric matrices is discussed in

detail. This section gives an overview about the numerical tasks which need to be performed, related available solutions and the aspect of computation time. The reader shall get an idea of the concepts and elements which will be used later on.

Technically for a symmetric matrix $N$ of dimension $n$ and a given vector $b$, the equation $Nx = b$ is to solve and $n$ agrees with the number of parameters that are to adjust. The straight forward approach is, to invert $N$ and build the product with $b$ subsequently. It is known that the computation of $x = N^{-1}b$ can be realised much more efficiently (Dahmen and Reusken (2008), chapter 13), but the bundle block adjustment also includes an evaluation phase that requires $N^{-1}$. The handling of measurement errors and the stochastic model aposteori depend on the inverted matrix. Hence, a solution is sought that reduces running time and maintains the statistical information. There are plenty approaches to matrix inversion, and all involve some transformation into triangular shape and the inversion of the obtained triangular matrix. One of the common methods to transform a symmetric matrix $M$ into triangular form, is the Cholesky decomposition. The algorithm computes a lower triangular $L$ with $LL^T = M$ or in a variant a lower triangular $L$ and a diagonal $D$ with $LDL^T = M$ (Dahmen and Reusken (2008), chapter 3). The inverse is then given by the product $(L^T)^{-1}D^{-1}L^{-1}$.

The actual running time, i.e. the time which it takes for a computer to execute an algorithm (or program), depends highly on the specifications of the machine. In order to analyse the effectiveness of an algorithm, a measure is required that is independent of the hardware and its performance specifications. Such a measure is the number of required floating points operations (flops), e.g. multiplications or additions which are applied to floating point numbers. The number of flops usually depends on variables of the algorithm which determine the size of the problem, for instance the dimension parameters of a matrix. In modern books of computer science the term *time complexity* has been introduced for this performance-independent consideration of the execution times, while in classic numerical literature the term *running time* is used (assuming that the context of a theoretical consideration is clear). The symbol $\mathcal{O}(n^3)$ denotes a running time of cubic order, stating that the number of flops is bound by $kn^3$ for some real number $k$. The value for $n^3$ is clearly dominant,

but the larger $n$ grows, the more the actual value of $k$ needs to be considered. A standard matrix product of two square matrices with dimension $n$ takes $\mathcal{O}(n^3)$ flops, so does a standard inversion algorithm.
Considering six orientation parameters per image and three coordinates per control point as unknown parameters, $n$ equals 30 000 if the data set consists of 1000 images and 8000 control points. It is $n^3 = 27\,000 \cdot 10^9$ flops, which is equal to 27 000 Giga flops (1 Gflop = $10^9$ flops). To estimate the machine-depending computation time, a value for $x = kn^3$ and the average core performance $y$ in Gflops/s must be specified. The time in seconds is then approximated by

$$t = \frac{x\ flops}{y\ Gflops/s} = \frac{x}{y}\,10^{-9} s\ .$$

A core performance of 3.0 Gflops/s and a value of $x = \frac{1}{3}n^3$ yields $t = 3000\,s$ rsp. 50 minutes. The structure of the matrix $N$ allows the application of a well-known technique (Niemeier, 2008) such that, in this case, the computation time shrinks to about six minutes. The adjustment problem is split into two partial problems of size $n_1 = 6000,\ n_2 = 24\,000$ where $n_1$ only depends on the number of images and $n_2$ only on the number of control points. The technique is explained in detail in chapter 4 (section 4.2.1). It basically reduces the inversion time to $\mathcal{O}(n_1^3)$ with the additional costs for a product of about $\mathcal{O}(n_1^2 n_2)$. It is owed to the special structure of $N$, esp. the sparse distribution of non-zero elements, that the use of the splitting method is of advantage. The resulting performance improvement is even for small data sets significant, but the challenge remains if the number of images grows large. For the Vesta data set, $n_1$ equals 32 640 and it takes with the specifications above 1.5 hours to solve the partial problem and more than 35 hours for the remaining product. Using a reference software at DLR Berlin these are realistic numbers; the total adjustment takes four iterations and about 168 hours. Furthermore the demand on memory depends on the number of control points and exceeds 250 GB (Preusker; private communication).
While implementing the inertial bundle block adjustment, tools of an effective resource management, e.g. sparse matrix compression and parallel computation, have been taken into account. All aspects of implementation, hardware and software considerations focusing the challenge of large data sets are subject of chapter 4, including the comparison of running times. It will be shown that

by a combination of splitting technique and matrix compression the product, which takes about 95 % of the computation time, can be eliminated such that the requirements for running time can be reduced to $\mathcal{O}(n_1^3)$ and for memory to $\mathcal{O}(n_1^2)$. Hence, the number of involved images becomes the most relevant factor with respect to resource limitations. At a second level of optimisation an algorithm has been developed to benefit from multi-core architecture, either CPU (compute processing unit) or GPU (graphics processing unit) using parallel computation for the inversion task. This approach combines the introduced splitting technique and the Cholesky decomposition. Because of the parallel design for the the single steps in the splitting technique, the procedure allows a large amount of parallel computation in contrary to other available inversion techniques (section 4.2.4).
Since the performance of a program depends on different hardware aspects and the environment of the operational system, it was tested on three different architectures, all equipped with Linux RedHat Enterprises 6. In addition to the work-station at TU Berlin which has been used for software development (4 cores, 16 GB RAM), also computing facilities of DLR Berlin (32 cores, 256 GB RAM) and the North-German Supercomputing Alliance HLRN (24 cores, 64 GB RAM) were used. All core specifications are related to Intel Xeon CPU chips, differing mainly in clock cycle and Gflops/s. The performance value of each machine was obtained based on the Whetstone benshmark test (Curnow and Wichmann, 1976).

## 1.6 Missions and data

There are three different space missions contributing to this work, two of them are still on-going. NASA's Viking mission to Mars and ESA's Mars Express mission provided the image data of Phobos. The image data of Vesta is obtained by the Dawn mission of NASA. The next subsections provide brief information about these missions and describes the selected data of the involved framing cameras.

## Viking

The Viking mission of NASA towards Mars was represented by two identical spacecraft, Viking 1 and Viking 2, each consisting of an orbiter and a lander. The primary mission objectives were to obtain high resolution images of the Martian surface, characterise the structure and composition of the atmosphere and surface, and search for evidence of life. During the whole operation period, about 500 images of the Martian moons were taken during flybys. Viking Orbiter 1 (VO1) arrived at Mars on 19th June 1976 and concluded its mission after 1489 orbits of Mars on 7th August 1980. From 12th February to 11th March 1977, VO1 approached Phobos and maintained an orbit that was synchronised with the flyby period of the moon. During this close encounter phase of about 30 martian revolutions, Phobos could be pictured from distances between 90 to 650 km above the surface. Duxbury and Callahan (1988) provide a detailed description of astrometric Viking observations that cover the complete operation time, focusing on the possibility to improve the ephemeris data of the satellites. Their study includes 166 images of Phobos and states the uncertainties of spacecraft positions and camera pointing angles, shown in table 1.3. In this study only images of VO1 are considered and the majority of them is taken during the early encounter phase, at an average distances of 460 km; only two of the images are captured in late 1977 at 1840 km and 3100 km distance. The Viking Imaging System (VIS) consisted of two identical cameras, VIS A and VIS B. There is one image of VIS B and 18 images of VIS A included.

**Viking orientation uncertainties**

| | | |
|---|---|---|
| Position | $1.0 - 8.0$ km | errors in the early mission phase are smaller (related to the periapsis passage) |
| Pointing | $0.03 - 0.3$ km | with star tracking method |
| | up to 3.0 km | when no star tracking was available |

Table 1.3: Uncertainties ($1\sigma$) of position and pointing data of the Viking Imaging System VIS, the true values are expected in $3\sigma$ range (Duxbury and Callahan, 1988).

## Mars Express

Mars Express (MEX) is Europe's first mission to Mars. The spacecraft entered an orbit around Mars on 25th December 2003 carrying a lander and seven instruments, including a High Resolution Stereo Camera (HRSC). The global investigation of Mars and its two moons, Phobos and Deimos, is still on-going. MEX performed several manoeuvres to capture Phobos during flybys. The HRSC pointed to a predefined inertial position where Phobos was expected to enter the field of view and the activated Super Resolution Channel (SRC) took a series of images. A detailed report about the imaging performance of the SRC on Mars Express is given by Oberst et al. (2008). For this study 53 frame images of the SRC were selected, taken at an average distance of 1887 km with respect to the moon's centre of mass. Pasewaldt et al. (2015) evaluated the image data with respect to the position and pointing uncertainties; they state that the positions have an uncertainty of ±224 m (according to ESOC) and the uncertainty for pointing angles is given with 0.01 degree for most of the images. For some images, shifts of up to 160 pixel in sample direction and 90 pixel in line direction are recorded (Pasewaldt et al., 2015, fig. 2), and the maximum shift corresponds to a pointing error of 0.084 degree. The SRC image data and stochastic model used here is the same as in Willner et al. (2010) and has been attached in appendix A (table A.1).

**Image data of Phobos**

| | **Viking (VIS)** | **MEX (SRC)** |
|---|---|---|
| Pixel Size | 0.01176 mm | 0.009 mm |
| Focal Length | 474.610 mm | 984.76 mm |
| Distance | ∅ 674 km (226 – 3078 km) | ∅ 1877 km (579 – 5265 km) |
| Resolution | ∅ 16.68 m (5.6 – 76.3 m) | ∅ 17.12 m ( 5.3 – 48.4 m) |
| Images | 19 | 53 |

Table 1.4: The nominal ground distance (km), assuming a radius of 11 km and the resulting ground resolution (m) of Viking and MEX images. The average resolution of all 72 images together is 17 m.

## Dawn

The launch date of the Dawn mission was 27th September 2007 and has been recently extended after the end of its prime mission on 30th June 2016. The top level question of the mission is to understand, how size and water determined the evolution of the planets. Therefore two massive protoplanets, Vesta and Ceres, were chosen as mission targets. Dawn is the first spacecraft which went in orbit around an object of the main asteroid belt. It took about 69 000 images and collected more than 132 GB of scientific data, and it continues to orbit its second mission target, the dwarf planet Ceres. The giant asteroid Vesta was explored from July 2011 to August 2012. There were two mission phases in a High-Altitude Mapping Orbit (HAMO): 28th September 2011 to 2nd November 2011 (HAMO-1) and 5th June 2012 to 25th July 2012 (HAMO-2). The data set considered here, consists of 5440 images, combined from HAMO-1 phase and HAMO-2 phase. Unlike the Phobos data, here all images are taken by the same spacecraft at distances which are close to the average distance of 944.5 km. Minimum and maximum are within 35 km range of the mean, implying a homogeneous ground resolution of about 88.1 meter (see table 1.5). The distance is measured with respect to the centre of mass. For all images the position uncertainty is given with 35 m and the pointing uncertainty with 0.006 gon.

**Image data of Vesta**

| | | |
|---|---|---|
| Pixel Size | 0.0140 mm | |
| Focal Length | 150.07 mm | |
| Distance | ∅ 944.5 km | (915.2 – 979.1 km) |
| Resolution | ∅ 88.1 m | (85.40 – 91.3 m) |
| Images | 5440 | |

Table 1.5: The 5440 images, combined from both of Dawn's High-Altitude Mapping Orbits, have an average nominal ground resolution of 88.1 metres.

# Chapter 2

# Definitions and Foundations

## 2.1 Mathematical foundations and notation

The methods which will be described and developed in this work, e.g. the bundle block adjustment (section 2.3), involve theorems and subsequent results of Linear Algebra. In this section the notation for mathematical objects will be given first. Secondly, a collection of fundamental algebraic results, proven and presented in detail by Fischer (2012), is stated here and taken for granted in all further considerations of matrix operations and related algorithms.

- $\mathbb{R}$ denotes the real numbers and $\mathbb{N}$ the natural numbers including zero.
- $M \in \mathcal{M}_{n,m}$ is any matrix with $n$ rows, $m$ columns and coefficients in $\mathbb{R}$.
- $\mathcal{M}_n = \mathcal{M}_{n,n}$ is the subset of square matrices, i.e. rows equal columns; the restriction to only one dimension parameter will imply that the matrix in question is square.
- $I_n$ is the unit matrix in $\mathcal{M}_n$, i.e. a diagonal matrix with $n$ ones along the diagonal; the dimension parameter is omitted if the context is clear.
- $\mathcal{O}(n^x)$ is an asymptotic bounding condition of a numerical series that depends on the variable $n$, $x \in \mathbb{R}$ arbitrary. The definition is:

$$S = \mathcal{O}(n^x) \iff \lim_{n\to\infty} \frac{S}{n^x} = 0 \iff \frac{S}{n^x} < k \text{ for some constant } k.$$

This notation is called *big-O notation* (reading $S$ is of order $n^x$). If $x = 1$, $S$ is of linear order, for $x = 2$ (rsp. $x = 3$) $S$ is of quadratic (rsp. cubic) order.

• *Definition 1 (block diagonal matrix):* A square matrix $A$ with dimension $n$ is called a *block diagonal matrix* with block dimension $s$ if the $s \times s$ blocks along the diagonal

$$B_k = \{a_{ij} \in A : (k-1)s <= i, j < ks\}, \quad k = 1, \ldots, \frac{n}{s}$$

contain the only possible non-zero entries of $A$. That is

$$|a_{ij}| > 0 \implies a_{ij} \in \bigcup_{k=1}^{n/s} B_k \, .$$

Like the block diagonal matrix is an extension of the diagonal matrix, the concept of a triangular matrix is extended in an analog way.

• *Definition 2 (block triangular matrix):* $A$ is called a *block triangular matrix* if there exist a block diagonal matrix $B$ and a triangular matrix $T$ such that $A = B + T$.
More precisely: Let $B_k \in \mathcal{M}_{s,s}$ be defined as above and $s$ a factor of $n$ s.th. $n/s = n_s \in \mathbb{N}$. A matrix $A \in \mathcal{M}_n$ with the structure

$$A = \begin{pmatrix} B_1 & & & \\ & B_2 & \text{U} & \\ & & \cdots & \\ & \text{L} & \cdots & \\ & & & B_{n_s} \end{pmatrix}$$

is called *upper block triangular matrix* (with block size $s$) if $a_{ij} = 0 \; \forall a_{ij} \in L$ and *lower block triangular matrix* if $a_{ij} = 0 \; \forall a_{ij} \in U$. It is

$$L = \bigcup_{k=2}^{n_s} \{a_{ij} \in A : (k-1)s <= i < ks; j < (k-1)s\} \text{ (lower part)}$$

$$U = \bigcup_{k=1}^{n_s-1} \{a_{ij} \in A : (k-1)s <= i < ks; j >= ks\} \quad \text{(upper part)}$$

The structure of block triangular matrices will be relevant for the implementation of the algorithm, described in 4.2.4.

• The determinant of a matrix is a function $det : A \mapsto \mathbb{R}$ that assigns a real number to the matrix A. For all $A \in \mathcal{M}_n$ the following holds:

$det(A) = 0 \iff$ there is no inverse of A. Consequently, if $det(A)$ differs from zero, the inverse $A^{-1}$ exists and satisfies $A\,A^{-1} = I_n$.

• For all $A, B \in \mathcal{M}_n$ with $|det(A)| > 0$ and $|det(B)| > 0$ it is $|det(AB)| > 0$ and $(AB)^{-1} = B^{-1}A^{-1}$.

• A $3 \times 3$ matrix $M$ with real coefficients corresponds unique to set of 3 functions that are defined on $\mathbb{R}^3$. $M$ then operates on $v \in \mathbb{R}^3$ by multiplication

$$Mv = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} a_{11}X & + & a_{12}Y & + & a_{13}Z \\ a_{21}X & + & a_{22}Y & + & a_{23}Z \\ a_{31}X & + & a_{32}Y & + & a_{33}Z \end{pmatrix}.$$

The vector $v$ is mapped to another vector, where each component is a linear combination of the original vector components. If $M$ has a determinant equal to $+1$, the matrix acts as a rotation. The columns are orthonormal vectors, which are the coordinates of the three base vectors $e_1, e_2, e_3$ after the rotation.

• The transposed of a matrix $A \in \mathcal{M}_{n,m}$, denoted with $A^T$, is obtained by exchanging rows with columns, hence $A^T \in \mathcal{M}_{m,n}$. The products $AA^T \in \mathcal{M}_n$ and $A^TA \in \mathcal{M}_m$ are always symmetric. Further is $(AB)^T = B^TA^T$.

• If all column vectors of a matrix are linear independent, it is said to have *full rank*. Any square matrix with full rank has a determinant different from zero, hence its inverse exists. If $A \in \mathcal{M}_{n,m}$ with full rank, then $N = A^TA \in \mathcal{M}_m$ has full rank, too. Hence $N^{-1}$ exists.

Note: It follows, that the definition of $N$ can be extended to $N = A^TPA$ if $P$ is a diagonal matrix with strictly positive diagonal entries. To see this, $P$ is written as product $P = P_{sqrt}P_{sqrt}$ with $P_{sqrt}(ii) = \sqrt{P(ii)}$ and $A$ is replaced with $P_{sqrt}A$.

• The trigonometric functions $\sin(x)$, $\cos(x)$ and $\tan(x)$ are taken as unit sensitive, i.e. units are transformed into radiant automatically by

$$x \mapsto \frac{\pi\,x}{t} \qquad \text{with} \quad t = \begin{cases} 180 & x \text{ in degree} \\ 200 & x \text{ in gon} \\ \pi & x \text{ is radiant} \end{cases}.$$

## 2.2 Conversion between reference frames

As stated in the introduction, the conversion of one reference frame into another is accomplished by rotation, i.e. the application of a rotation matrix $M$

to the frame's base vectors. If a vector is given with respect to frame $A$ and shall be transformed into a vector of frame $B$, $A$ is called the *local frame* and $B$ the *target frame*.

There are several equivalent options to realise such a rotation, which are all presented by $M$. One possibility is to split $M$ in different primary rotations, that are rotations around a main axis of the coordinate system. They are defined at first, then two ways of rotation sequences, that matter for the implementation of the bundle block adjustment, are distinguished.

**Primary Rotations**

Considering a right hand cartesian coordinate system, the three primary rotations $R_1 = R_{(1,0,0)}$ around the $X$-axis, $R_2 = R_{(0,1,0)}$ around the $Y$-axis and $R_3 = R_{(0,0,1)}$ around the $Z$-axis are defined such that for $\theta = 90°$ the following transformation of base vectors is realised:

$$\begin{aligned}
R_1(\theta) &: (0,1,0) \longmapsto (0,0,1) \quad , \quad Y^+ \mapsto Z^+ \quad \text{anti-clockwise} \\
R_2(\theta) &: (1,0,0) \longmapsto (0,0,1) \quad , \quad X^+ \mapsto Z^+ \quad \text{clockwise} \\
R_3(\theta) &: (1,0,0) \longmapsto (0,1,0) \quad , \quad X^+ \mapsto Y^+ \quad \text{anti-clockwise}
\end{aligned}$$

Given this, the single matrices that rotate a vector by $\theta°$ accordingly have the form

$$\begin{aligned}
R_1(\theta) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{pmatrix}, \; R_2(\theta) = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \\
R_3(\theta) &= \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} .
\end{aligned}$$

**Note:** Since $\sin(-x) = -\sin(x)$ and $\cos(-x) = \cos(x)$ the matrix $R_i^T(\theta) = R_i(-\theta)$ realises the inverse rotation, that is the rotation in the reversed direction. Hence $R_i^T(\theta)\, R_i(\theta) = I$ is for all $\theta$ and for all $i \in \{1,2,3\}$ the unit matrix. In general any product of two rotational matrices is a rotation matrix again and, hence, any sequence of products, too.

### 2-1-3 Rotation

$M = M_{2-1-3}(\phi, \omega, \kappa) := R_3(\kappa)R_1(\omega)R_2(\phi)$ will be called shortly a 2-1-3 rotation. The term specifies the order of the rotation sequence:

- 1st rotation: $\phi^\circ$ around $Y$-axis, 2nd rotation: $\omega^\circ$ around $X$-axis, 3rd rotation: $\kappa^\circ$ around $Z$-axis

The 2-1-3 rotation is used e.g. to rotate a camera reference frame into a body-fixed reference frame. The angles $\phi$ and $\omega$ reflect the relative orientation of the pole axis to the $Z$-axis of the camera, so the first two rotations will align both axes with each other. Then, $\kappa$ is the offset between the remaining axes. The order of the rotation sequence is relevant for the computation of the pointing angles' derivatives, $M$ is defined by the entries

$$\begin{aligned}
M_{11} &= \cos(\phi)\cos(\kappa) + \sin(\omega)\sin(\phi)\sin(\kappa);\\
M_{12} &= \cos(\omega)\sin(\kappa);\\
M_{13} &= -\sin(\phi)\cos(\kappa) + \sin(\omega)\cos(\phi)\sin(\kappa);\\
M_{21} &= -\cos(\phi)\sin(\kappa) + \sin(\omega)\sin(\phi)\cos(\kappa);\\
M_{22} &= \cos(\omega)\cos(\kappa);\\
M_{23} &= \sin(\phi)\sin(\kappa) + \cos(\kappa)\sin(\omega)\cos(\phi);\\
M_{31} &= \sin(\phi)\cos(\omega);\\
M_{32} &= -\sin(\omega);\\
M_{33} &= \cos(\omega)\cos(\phi);
\end{aligned} \tag{2.1}$$

and the angles can be received by the following relationship

$$\phi = \tan\left(\frac{M_{31}}{M_{33}}\right), \qquad \omega = \sin(M_{32}), \qquad \kappa = \tan\left(\frac{M_{21}}{M_{22}}\right) \quad . \tag{2.2}$$

The vector $(M_{31}, M_{32}, M_{33})$ is the pole axis vector $\boldsymbol{e_3}$ in camera coordinates.

### 3-1-3 Rotation

The second important rotation sequence is the 3-1-3 rotation which is related to the rotational model of a planetary body $B$. The ICRF is the local reference frame here and the body-fixed reference frame is the target frame. The rotation matrix, defined by the angles $(\alpha, \delta, W)$

$$R_B = R_{\alpha,\delta,W} := R_3(W)R_1(\frac{\pi}{2} - \delta)R_3(\frac{\pi}{2} + \alpha) \tag{2.3}$$

converts local ICRF coordinates into body-fixed coordinates. This sequence of rotation matches the IAU convention to describe the orientation of the body-fixed reference frame with respect to the ICRF by means of the body's rotational elements.
The inverse rotation $R^T$ converts body-fixed coordinates into ICRF coordinates. From there they can be further converted into coordinates with respect to another reference frame (e.g. camera or primary body) by application of the corresponding rotation matrix (see eq. 2.4).

**Implicit Rotation**
Let $C$ and $B$ two reference frames and $R_C$, $R_B$ the rotation matrix that rotates the ICRF into $C$ rsp. $B$. then the product

$$R_{CB} = R_B R_C^T \tag{2.4}$$

defines the transformation from $C$ to the target frame $B$. The local frame $C$ is rotated back into the ICRF via $R_C^T$ and from there rotated into the target frame $B$. Using (2.2) for $R_{CB}$, the relative frame orientation can be received. On the other hand, if $R_B$ and $R_{CB}$ are known, the matrix that converts ICRF to C-centered coordinates is obtained by

$$R_C = R_{CB}^T R_B \ . \tag{2.5}$$

## 2.3 Body-fixed bundle block adjustment

Here, the classical method of a photogrammetric bundle block adjustment which is formulated in the body-fixed reference frame will be described, as used in planetary geodesy by e.g. Willner et al. (2010) or Preusker et al. (2012). As stated before, by applying the method, the orientation parameter of the camera can be significantly improved. The resulting CPN is cleaned from errors in image point measurements and the overall intersection error is reduced. The functional model is based on the following context: A surface point that was captured multiple times and identified in $n \geq 2$ images, can be expressed as the intersection point of $n$ lines. Knowing the position of the camera and the orientation of the reference frames at the time the picture was taken, the 3D coordinates of the point can be calculated.

In table 2.1 are listed the parameters which are estimated by the classical (body-fixed) bundle block adjustment. The first two columns show the type of parameter and the corresponding symbol used for it. The third columns shows the number of parameters for each group, there are three coordinates for each control point and of each camera position as well as three pointing angles. Columns four and five hold the counting variables for each type rsp. for the total number of parameters. For each of these parameters a starting value is required that will be updated during the process. Table 2.2 lists the observation parameters and is structured in the same way. Parameters listed in both tables are *additional observations*, the image point observations are kept constant.

**Table of Unknowns**

| **parameter** | **symbol** | **dim** | **count** | **total** |
|---|---|---|---|---|
| Control Point | $(X, Y, Z)$ | 3 | nCP | 3nCP |
| Camera Position | $(X_0, Y_0, Z_0)$ | 3 | nIm | 3nIm |
| Camera Pointing | $(\omega, \phi, \kappa)$ | 3 | nIm | 3nIm |

Table 2.1: Parameters which are estimated in the bundle block adjustment, their symbols and counting variables

**Table of Observations**

| **parameter** | **symbol** | **dim** | **count** | **total** |
|---|---|---|---|---|
| Image Point | $(\xi, \eta)$ | 2 | nIP | 2nIP |
| Camera Position | $(X_0, Y_0, Z_0)$ | 3 | nIm | 3nIm |
| Camera Pointing | $(\omega, \phi, \kappa)$ | 3 | nIm | 3nIm |

Table 2.2: Parameters which function as observations in the bundle block adjustment. Since the camera parameters also function as unknowns (table 2.1) they are called additional observations.

### 2.3.1 The functional model

The functional model for a photogrammetric bundle block adjustment is derived from the mathematical equation of a line in the three dimensional space $\mathbb{R}^3$. A line $L$ is defined by two points $(P_0, P_1)$ or a point and a direction vector $P$, the direction can be obtained by $P = P_1 - P_0$.

$$L = P_0 + m\,P = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + m \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad m \in \mathbb{R} \tag{2.6}$$

Like $P_0 = (x_0, y_0, z_0)$ corresponds with $m = 0$ and $P_1$ with $m = 1$, any point of this line belongs to a specific value of $m$.
Considering the example for a camera reference frame given in subsection 1.2.3 and a planetary body that is pictured by this camera, $P_0$ is the point that represents the camera lens - the origin of the camera's coordinate system. Equation 2.6 describes a line that connects a point $P$ on the CCD with a surface point of the pictured object (for some unknown but yet existing $m$).
To express this point in coordinates that are centred within the target body, the rotation matrix $R_{CTB} = R_{2-1-3}(\phi, \omega, \kappa)$, which depends on the pointing information of the camera, is applied as discussed in 2.2 (p. 27).
The surface point of the target body $TB$ is then given by the functional relation

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{TB} = m\,R_{CTB} \begin{pmatrix} \xi \\ \eta \\ z_f \end{pmatrix}_{Camera} + \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix}_{TB} \tag{2.7}$$

for some real number $m$ where $(\xi, \eta, z_f)$ is the image point vector in camera coordinates and $(X_0, Y_0, Z_0)_{TB}$ is $P_0$ in $TB$-coordinates. Solving for the image point vector leads to

$$R_{CTB}^T \begin{pmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{pmatrix}_{TB} = m \begin{pmatrix} \xi \\ \eta \\ z_f \end{pmatrix}_{Camera} . \tag{2.8}$$

The nine parameters on the left side are referred to as *unknowns*, where as the image point vector is called *observation*. Here, $(\xi, \eta)$ are the coordinates in the image plane and $z_f$ is the (negative) focal length. Obviously this parameter is

constant in all images of the same camera. The parameter $z_f$ is used to further transform the equation and find a form, that will no longer involve $m$.
Equation 2.8 is a system of three single linear equations, therefore sometimes referred to as co-linearity equation. Denoting the left hand side of 2.8

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix} = R_{CTB}^T \begin{pmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{pmatrix}_{TB}$$

the three lines can be written shortly as

$$X' = m\xi, \quad Y' = m\eta, \quad Z' = mz_f.$$

Because each line $i \in \{1, 2, 3\}$ depends on the same set of parameters $u = (\omega, \phi, \kappa, X_0, Y_0, Z_0, X, Y, Z)$ resulting from the product of row $i$ of $R_{CTB}^T$ with the difference vector, one can set

$$\begin{array}{rcccl} X' & = & F_1(u) & = & m\xi \\ Y' & = & F_2(u) & = & m\eta \\ Z' & = & F_3(u) & = & mz_f \end{array} .$$

Dividing each line by $m = Z'/z_f$ the representation of the image point observation does not depend on $m$ any more and

$$\xi = z_f \frac{X'}{Z'} = z_f \frac{F_1(u)}{F_3(u)} =: F_\xi(u) \tag{2.9}$$

$$\eta = z_f \frac{Y'}{Z'} = z_f \frac{F_2(u)}{F_3(u)} =: F_\eta(u) \tag{2.10}$$

can be written as functions of unknowns.
Applying the Taylor development to first order gives

$$\xi = F_\xi(u_0) + F'_\xi(u_0)(u - u_0) + w_\xi \tag{2.11}$$

$$\eta = F_\eta(u_0) + F'_\eta(u_0)(u - u_0) + w_\eta. \tag{2.12}$$

with $F'_\xi(u_0) = (p'_1, \ldots, p'_9)$ the vector of partial derivatives, that are obtained by using the chain and quotient rule:

$$p'_k = z_f \frac{F'_1(u_0)_k \, F_3(u_0)_k - F'_3(u_0)_k \, F_1(u_0)_k}{(F_3(u_0)_k)^2} .$$

By replacing $F_1$ with $F_2$ the partial derivatives $F'_\eta(u_0) = (p'_{10}, \ldots, p'_{18})$ are obtained. The equation above corresponds to exactly one image point observation. The variable $u_0$ will be different for any other observation, since it either corresponds with a different control point or the same point in a different image. Therefore, a subscribt $i$ is added to all variables indicating that they correspond to observation $L_i = (\xi_i, \eta_i)$. In order to respect the standard notation in common literature about adjustment theory (Niemeier, 2008, chapter 4), let

$$A_i = \begin{pmatrix} p'_{i1} & \cdots & p'_{i9} \\ p'_{i10} & \cdots & p'_{i18} \end{pmatrix}, \quad v_i = -\begin{pmatrix} w_{\xi_i} \\ w_{\eta_i} \end{pmatrix}, \quad \hat{x}_i = u_i - u_{i0} \tag{2.13}$$

and

$$L_i^0 = \hat{L}_i = \begin{pmatrix} F_{\xi_i}(u_{i0}) \\ F_{\eta_i}(u_{i0}) \end{pmatrix}, \quad l_i = L_i - \hat{L}_i\,.$$

Equations 2.11 and 2.12 can then be rewritten in matrix form

$$l_i + v_i = A_i \hat{x}_i, \tag{2.14}$$

which yields the linearised functional model for image point observation $L_i$. Considering all image point observations and combining them into one vector, a system of $2nIP$ linear equations is built analogue to (2.14), yielding

$$l + v = A\hat{x}\ .$$

Vector $l$ holds the differences of all the actual observations and their approximations according to equations 2.9 and 2.10, $v$ is built analogue to 2.13. Further $\hat{x}$ is the difference of all the unknown parameters and their starting approximations – $nU = 6nIm + 3nCP$ in total (compare table 2.1). $A$ is the enlarged Jacobi matrix with $nU$ columns, where the entries for uninvolved parameters are set to zero. For the observed unknowns (additional observations) of each image $j$, the functional model is already linear.

$$(L_{j1}, \ldots, L_{j6}) = (\omega, \phi, \kappa, X_0, Y_0, Z_0)_j$$

denotes the a priori values of the observed image parameters, which are kept fixed during the adjustment. $\hat{L}_{jk}$ $(k = 1, \ldots, 6)$ denotes the estimated or

improved value, which changes during the course. Hence $l_{jk} = L_{jk} - \hat{L}_{jk}$ leads to the functional model

$$l_{jk} + v_{jk} = L_{jk} - \hat{L}_{jk} = 1 \cdot \hat{x}_{jk}$$

for the observed unknowns. Therefore, in addition to the $2nIP$ rows of $A$, for each of the additional observations (see table 2.2) a line is added that has an entry of $+1$ in the corresponding column and zero elsewhere. Consequently the vectors $l$ and $v$ are extended by the corresponding elements of $l_{jk}, v_{jk}$. By design the columns of $A$ are related to the unknown parameters in the order given in table 2.3.

**Column order of unknown parameters**

| Image 1 | Image 2 | ... | CP 1 | CP 2 | ... |
|---|---|---|---|---|---|
| $(\omega, \phi, \kappa, X_0, Y_0, Z_0)_1$ | $(\omega, \phi, \kappa, X_0, Y_0, Z_0)_2$ | ... | $(X, Y, Z)_1$ | ... | ... |
| Image parameter columns $1 : 6\,nIm$ | | | CP col. $6\,nIm + 1 : nU$ | | |

Table 2.3: Correspondence of unknown parameters and the columns of the design matrix $A$; the total number of parameters is $nU = 6\,nIm + 3\,nCP$ where $nIm$ is the count of images and $nCP$ the count of control points.

The system $A\hat{x} = l + v$ has infinitely many solutions. If it is solved such that $v$ is minimal in length, the functional representation of the observations is optimal. A Least Squares Method will lead to this result.
For the moment let $N = A^T A$, then

$$N\hat{x} = A^T l + A^T v \tag{2.15}$$

is called the *normal equation*. If $\hat{x}$ solves $N\hat{x} = b$ obviously $A^T v = 0$. The Least Squares algorithm solves $\hat{x} = N^{-1} A^T l$ and updates $u_0 = u_0 + \hat{x}$.
It terminates if $\hat{x} \to 0$ which is equivalent to $u = u_0$ or, more likely, $v^T v \to \epsilon > 0$. This ensures $v^T v$ is minimal at the end and the vector $u_0$ contains the improved parameters from table 2.1.
This, so far, explains the functional model and the mathematical background. It is possible to further control the outcome of the adjustment, if a stochastic

model is introduced in addition. That means, the use of statistical information, that are available for the set of observations (table 2.2). If no information are available, the statistic quantities (*weights*) can be calculated.

### 2.3.2 The stochastic model

In praxis, observations come with a measure of uncertainty. For image points this value can be derived from the pixel size, pointing and position information are assigned with an assumed deviation ($\sigma$) that might depend on the method of tracking.
The stochastic model is given by the covariance matrix $\Sigma_{ll}$. If $\sigma_i$ is belongs to observation i and $n$ is the total number of observations, a $n \times n$ matrix $\Sigma_{ll}$ is built that satisfies $\Sigma_{ll}(i,i) = \sigma_i^2$. For large systems, covariances are often neglected and a diagonal matrix is used instead. This praxis is followed in this work, too. From there the weight matrix $P$ is constructed as

$$P = \sigma_0^2 \Sigma_{ll}^{-1}$$

for some reasonable numerical constant $\sigma_0 > 0$. The value ensures numerical stability and can be set to the mean $\frac{1}{n}\sum \sigma_i$ or simply to the value that corresponds to the uncertainty of an image point observations.
Redefining $N$ as $N = A^T P A$ changes the normal equation to

$$N\hat{x} = A^T P l + A^T P v =: b \, . \tag{2.16}$$

Of course, it is preferable to maintain the statistical information during the process. If the improvements shall be evaluated after each iteration step, technically the inverse of the normal matrix $N$, denoted with $N^{-1}$, is required. In the literature also $Q_{xx}$ is used to denote the inverse. The diagonal of this matrix contains at the end the improved weights for the unknown parameters. To evaluate the image observations the weights are calculated from the *HAT*-matrix

$$H = AN^{-1}A^T \, .$$

The dimension of $H$ agrees with the total number of observations $n = |l|$. Hence, if $n$ is large $H$ requires for the $n^2$ entries a large amount of memory. Typically only the diagonal entries of $H$ are used to compute the weights for

observations $l$ and improvements $v$. Setting $s_0^2 = v^T Pv/(n - nU)$ the new weight for observation $i$ is obtained by

$$s_0 \sqrt{H(i,i)} = \sigma_i \ . \tag{2.17}$$

Since the additional observations are contained here, the new weights of the camera orientation parameters are computed implicitly in this step. These values are not be updated during the iterations of the program. By

$$s_0 \sqrt{P^{-1} - H(i,i)} = \sigma_{v_i}$$

the standard deviation of the improvement $v_i$ is obtained. The quality variable sigma a posteriori $s_0$ estimates the system error. Since $\sigma_0^2$ is replaced with $s_0^2$ in the next iteration, the factor $\sigma_0/s_0$ converges to $+1$.
The vector obtained by $\bar{v}_i = v_i/\sigma_{v_i}$ holds the normed improvements. A specified threshold $\sigma_{max}$ is used to evaluate the magnitude of the improvements. Observations for which $|\bar{v}_i| > \sigma_{max}$ are eliminated from the data set. See Niemeier (2008) for further information.
An effective way to calculate the HAT-diagonal without building $H$ is given in chapter 4 (see page 65).

# Chapter 3

# Inertial Frame Bundle Block Adjustment

In this chapter the classic bundle block adjustment, described in section 2.3, will be extended and formulated within an inertial reference frame. This reference frame is the ICRF. Figure 3.1 shows a scheme of the extended method. Compared with the body-fixed context, almost all aspects here are the same. The major difference is, that the rotational model of the body is taken into account in form of a second rotation matrix. As a consequence the orientation of the camera is no longer given with respect to the body-centred reference frame but with respect to the ICRF. This affects also the computation of the derivatives of the classical parameters which are given in section 3.3.

## 3.1 The extended functional model

The matrix $R_{CTB}$ in equation (2.7) on page 30 transforms the image coordinates directly into body-fixed coordinates. The principle of the bundle block adjustment in the inertial frame is the split of the rotation matrix $R_{CTB}$ into two single rotations which involve the inertial reference frame (ICRF). From the analytical rotational model the angles $\alpha$, $\delta$ and $W$ are calculated for the time associated with the image and the 3-1-3 rotation matrix $R_{TB}$ is built according to equation (2.3) on page 27. If a light time correction was applied to obtain the position vector of the camera, the corrected time is used here, too. The orientation of the camera with respect to the ICRF is given by the matrix

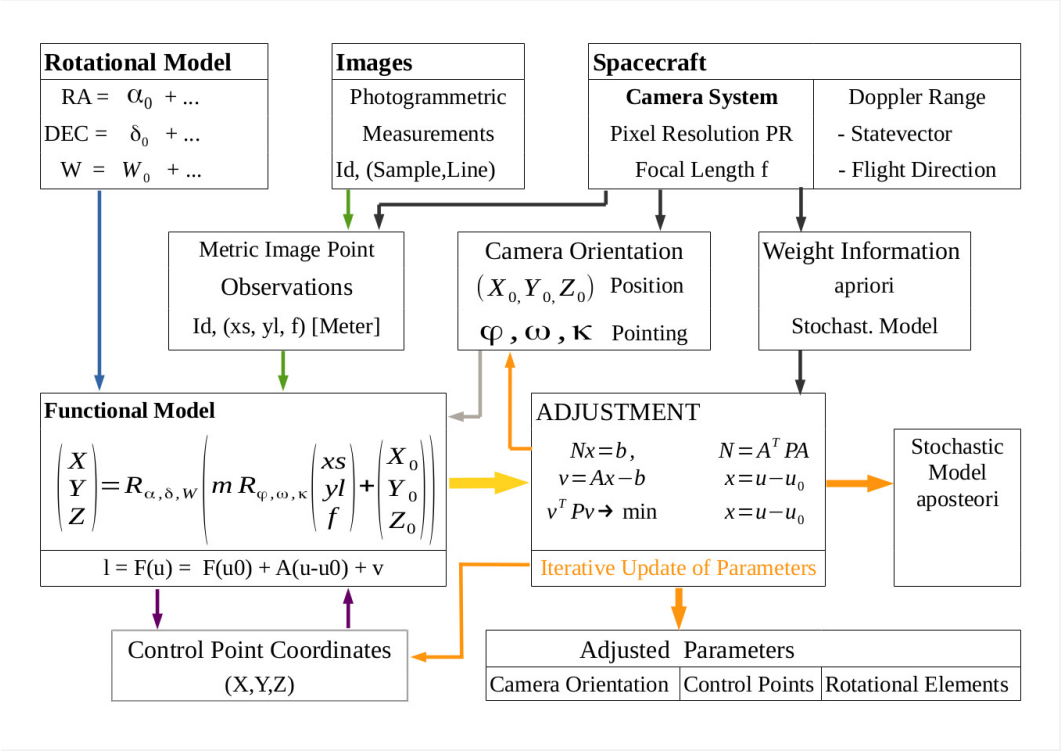



Figure 3.1: Overview of the inertial frame bundle block adjustment

$R_C$ such that $R_C$ converts ICRF coordinates in camera coordinates and $R_C^T$ vice versa. The conversion between camera reference frame and body-fixed reference frame can be written as $R_{CTB} = R_{TB}R_C^T$. Hence, co-linearity equation (2.7) on page 30, is equivalent to

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{TB} = m\, R_{TB} R_C^T \begin{pmatrix} \xi \\ \eta \\ z_f \end{pmatrix}_{Camera} + R_{TB} \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix}_{ICRF} \tag{3.1}$$

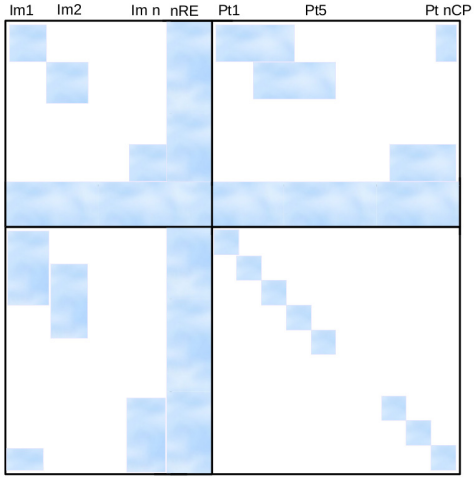
The position vector of the camera is now given with respect to the ICRF. The solving for the image point vector leads to

$$m \begin{pmatrix} \xi \\ \eta \\ z_f \end{pmatrix}_{Camera} = R_C \left( R_{TB}^T \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{TB} - \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix}_{ICRF} \right) \tag{3.2}$$

where the right hand side can be shorten again with $(X', Y', Z')$ and the representation

$$\xi = z_f \frac{X'}{Z'} \quad , \quad \eta = z_f \frac{Y'}{Z'}$$

of the image point observation remains the same as before. Since the equations for each entry in $R_{TB}$ are known, it is possible to build the partial derivatives for specified rotational elements and include them in the bundle block adjustment. As a consequence, the vector of unknowns is extended and likewise the Jacobi matrix $A$ which holds now in addition the derivatives with respect to these new parameters. Let $nRE$ the number of rotational elements which are to determine. Since the new unknowns are time independent, they simply appear once in the unknown vector and result in only $nRE$ additional columns of $A$ (see figure 3.2).




Pattern of the normal matrix for
$n$ images ($Im$),
$nCP$ control points ($Pt$),
$nRE$ rotational elements.
The blank areas are zero entries. The columns for the rotational elements are dense, all other columns contain mostly zeros.

Figure 3.2: Distribution pattern of non-zero elements of normal matrix $N$

Two advantages follow: First, this set up leads to a larger coherence of the whole network - via the rotational parameters each observation is now connected with any other. Secondly, the computational effort will not significantly increase. Nevertheless the new approach requires a completely new implementation of the adjustment algorithm.

## 3.2 Derivatives of rotational elements

The possible parameters, that are to include in the functional model, depend on the complexity of the functions, that define the rotational elements. A general representation is given by

$$\alpha(t) = \alpha_0 + a(t), \quad \delta(t) = \delta_0 + d(t), \quad W(t) = W_0 + w(t)$$

where $a$, $d$, $w$ are time-dependent functions and the other variables are the initial coordinates for the three angles with respect to the reference epoch. In case that $a$, $d$ and $w$ are zero for $t = 0$, they give exactly the orientation of the body at J2000.0 . Since the observations are functions of $\alpha, \delta$ and $W$ and furthermore the angles are arguments of trigonometric functions, it becomes necessary to apply the chain rule for differentiating composite functions $t \mapsto f(g(t))$, which is

$$f(g(t))' = g'(t)f'(g(t)) \quad .$$

For the body Vesta it is $a = d = 0$ and $\lambda_0 = 0$. In this simple case, only the inertial values are included as rotational elements. Since

$$\frac{\partial \alpha}{\partial \alpha_0} = \frac{\partial \delta}{\partial \delta_0} = \frac{\partial W}{\partial W_0} = 1\,,$$

the chain rule gives

$$\frac{\partial f(g)}{\partial g_0} = \frac{\partial g}{\partial g_0}\frac{\partial f(g)}{\partial g} = 1 \cdot \frac{\partial f}{\partial g} = f'(g)\,, \quad g \in \{\alpha, \delta, W\}\,. \tag{3.3}$$

For a body with precession and a forced libration like Phobos the functions $a, d, w$ can be simplified written as

$$a(t) = p_\alpha \sin P_t, \quad d(t) = p_\delta \cos P_t, \quad w(t) = rt + p_W \sin P_t + \lambda_0 \sin O_t$$

where $P_t$ depends on the precession, $O_t$ on the orbit position and $r$ is the mean spin rate. In this case $\lambda_0$ can be included as a unknown parameter. Because of $\partial W/\partial \lambda_0 = \sin O_t$ the chain rule leads to

$$\frac{\partial f(W)}{\partial \lambda_0} = \frac{\partial W}{\partial \lambda_0}\frac{\partial f(W)}{\partial W} = \sin O_t f'(W)\,.$$

These examples are sufficient to demonstrate the principle and to show that the derivatives are specific for a chosen target body.

There is a procedure which is independent of the choice of rotational elements, the required building of the derivatives $f'(\alpha), f'(\delta), f'(W)$ for the set of functions $f$ that define the entries of the rotation matrix $R_{TB}$. To do this, equation 3.2 is rewritten as

$$m \begin{pmatrix} \xi \\ \eta \\ z_f \end{pmatrix} = R_C \begin{pmatrix} r_{11}X & +r_{21}Y & +r_{31}Z \\ r_{12}X & +r_{22}Y & +r_{32}Z \\ r_{13}X & +r_{23}Y & +r_{33}Z \end{pmatrix} - R_C \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} \tag{3.4}$$

with $r_{ij}$ entries in $R_{TB} = R_{3-1-3}(\alpha + 90, 90 - \delta, W)$ as defined in 2.3. The entries $r_{ij}$ are linear combinations of trigonometric functions:

$$\begin{pmatrix} r_{11} \\ r_{21} \\ r_{31} \end{pmatrix} = \begin{pmatrix} -\cos(W)\sin(\alpha) - \sin(W)\sin(\delta)\cos(\alpha) \\ \sin(W)\sin(\alpha) - \cos(W)\sin(\delta)\cos(\alpha) \\ \cos(\delta)\cos(\alpha) \end{pmatrix}$$

$$\begin{pmatrix} r_{12} \\ r_{22} \\ r_{32} \end{pmatrix} = \begin{pmatrix} \cos(W)\cos(\alpha) - \sin(W)\sin(\delta)\sin(\alpha) \\ -\sin(W)\cos(\alpha) - \cos(W)\sin(\delta)\sin(\alpha) \\ \cos(\delta)\sin(\alpha) \end{pmatrix}$$

$$\begin{pmatrix} r_{13} \\ r_{23} \\ r_{33} \end{pmatrix} = \begin{pmatrix} \sin(W)\cos(\delta) \\ \cos(W)\cos(\delta) \\ \sin(\delta) \end{pmatrix} .$$

Since $\sin'(x) = \cos(x)$ and $\cos'(x) = -\sin(x)$, it is $\forall a \in \mathbb{R}$

$$\frac{\partial\,(a\sin(x))}{\partial x} = a\cos(x)\,, \quad \frac{\partial\,(a\cos(x))}{\partial x} = -a\sin(x)\,. \tag{3.5}$$

For the sake of space saving let $c$ denote the cosine and $s$ the sine function in the following three matrices, obtained by building the derivatives for each entry of $R_{TB}$:

$$R'_\alpha = \begin{pmatrix} -c(W)c(\alpha) + s(W)s(\delta)s(\alpha) & -c(W)s(\alpha) - s(W)s(\delta)c(\alpha) & 0 \\ s(W)c(\alpha) + c(W)s(\delta)s(\alpha) & s(W)s(\alpha) - c(W)s(\delta)c(\alpha) & 0 \\ -c(\delta)s(\alpha) & c(\delta)c(\alpha) & 0 \end{pmatrix}$$

$$R'_\delta = \begin{pmatrix} -s(W)c(\delta)c(\alpha) & -s(W)c(\delta)s(\alpha) & -s(W)s(\delta) \\ -c(W)c(\delta)c(\alpha) & -c(W)c(\delta)s(\alpha) & -c(W)s(\delta) \\ -s(\delta)c(\alpha) & -s(\delta)s(\alpha) & c(\delta) \end{pmatrix}$$

$$R'_W =$$

$$\begin{pmatrix} s(W)s(\alpha) - c(W)s(\delta)c(\alpha) & -s(W)c(\alpha) - c(W)s(\delta)s(\alpha) & c(W)c(\delta) \\ c(W)s(\alpha) + s(W)s(\delta)c(\alpha) & -c(W)c(\alpha) + s(W)s(\delta)s(\alpha) & -s(W)c(\delta) \\ 0 & 0 & 0 \end{pmatrix} .$$

$R'_\alpha$ and $R'_W$ can both be built from columns (rsp. rows) of matrix $R_{TB}$ itself:

$$\partial\, r_{ij}/\partial\alpha \;:\quad r'_{i1} = -r_{i2},\quad r'_{i2} = r_{i1},\quad r'_{i3} = 0;\quad i = 1,2,3$$

$$\partial\, r_{ij}/\partial W \;:\quad r'_{1i} = r_{2i},\quad r'_{2i} = -r_{1i},\quad r'_{3i} = 0;\quad i = 1,2,3$$

Let

$$u(t) := (\omega, \phi, \kappa, X_0, Y_0, Z_0, \alpha, \delta, W)_t$$

the subset of the unknown vector that corresponds with the image time $t$, then

$$m \begin{pmatrix} \xi \\ \eta \\ z_f \end{pmatrix} = F(u(t), X, Y, Z)$$

holds for the observation of point $(X, Y, Z)$ at time $t$. To get the partial derivatives with respect to $g \in \{\alpha, \delta, W\}$, the entries $r_{ij}$ in equation (3.4) are replaced with the corresponding entries of the derivative matrix; the right term is ignored, since it is independent of the rotational elements. This leads to the general formula

$$\frac{\partial F(u(t), X, Y, Z)}{\partial g} = R_C R'_g \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \quad g \in \{\alpha, \delta, W\}\,. \tag{3.6}$$

For a particular rotational element $p$, equation (3.6) becomes

$$\frac{\partial F(u(t), X, Y, Z)}{\partial p} = \frac{\partial g}{\partial p} R_C R'_g \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}, \quad g \in \{\alpha, \delta, W\}\,.$$

## 3.3 Derivatives of classical parameters

**Positions:** The position values are obtained as in the body-fixed case, but here $R^T_{CTB}$ is replaced with $R_C$. Hence, if $col_i$ is column $i \in \{1,2,3\}$ of $R_C$, then

$$\frac{\partial F}{\partial X_0} = col_1\,, \quad \frac{\partial F}{\partial Y_0} = col_2\,, \quad \frac{\partial F}{\partial Z_0} = col_3$$

with $F$ as before. It is clearly to see, that the determination of the position vector no longer depends on the rotational model.

**Control Points:** In the body-fixed case, the values for the control points are simply the negative position derivatives. Here, the derivatives are derived from

$$R_C R_{TB}^T \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = R_C \begin{pmatrix} r_{11}X & +r_{21}Y & +r_{31}Z \\ r_{12}X & +r_{22}Y & +r_{32}Z \\ r_{13}X & +r_{23}Y & +r_{33}Z \end{pmatrix}$$

as it was the case for the rotational elements. It follows for the $X$-coordinate

$$\frac{\partial F}{\partial X} = \begin{pmatrix} rc_{11}r_{11} + rc_{12}r_{12} + rc_{13}r_{13} \\ rc_{21}r_{11} + rc_{22}r_{12} + rc_{23}r_{13} \\ rc_{31}r_{11} + rc_{32}r_{12} + rc_{33}r_{13} \end{pmatrix} ,$$

where $rc_{ij}$ represent the entries of $R_C$. Equally the derivatives for $Y$ are obtained by replacing $r_{1i}$ with $r_{2i}$ rsp. for $Z$ by replacing with $r_{3i}$, $i \in \{1, 2, 3\}$.

**Pointing angles:** It is convenient to build the derivative matrix of $R_C$ with respect to each angle, denoted with $R'_\omega$, $R'_\phi$ and $R'_\kappa$. Let

$$P = R_{TB}^T \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{TB} - \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix}_{ICRF}$$

the difference of the control point and the position in ICRF coordinates. then

$$\frac{\partial F}{\partial \omega} = R'_\omega P$$

is the derivative vector with respect to $\omega$ and equally $R'_\phi P$ rsp. $R'_\kappa P$ for the remaining angles. The 2-1-3 rotation was defined on page 27 and the derivatives can be obtained with rule (3.5) on page 41 as in the classical case.

## 3.4 Test case - a simulation of Phobos

The principle was first tested on a simulation, because this procedure allows maximal control and evaluation of an algorithm. Another consequence is a better understanding of the behaviour with respect to the stochastic model. It is important to find out, how well the observations need to be known in order to

reconstruct the given topography successfully. It is furthermore of advantage to model preferably realistic errors. Here, a realistic model of the Martian Moon Phobos was created, based on an existing data set of 680 control points which are distributed in 73 images with given frame parameters (position and pointing of the camera).

**Creating the synthetic data**

With the given orientation data the initial coordinates of each control point are obtained indepently by a simple adjustment minimising the forward intersection error. Now these values are kept fix as the defined topography of the simulated object. The next step is, to change the coordinates of the image measurements in such a way, that they exactly match the defined points. This can be done by applying equation 2.8 to each point $(X, Y, Z)$. The vector from the origin of the camera's reference frame to the point on the surface is obtained by

$$\begin{pmatrix} X' \\ Y' \\ Z' \end{pmatrix}_{Camera} = m \begin{pmatrix} \xi \\ \eta \\ z_f \end{pmatrix}_{Camera} .$$

Then $\xi = z_f\, X'/Z'$, $\eta = z_f\, Y'/Z'$ are calculated to eliminate the scaling factor $m$. If the unit of focal length $z_f$ is millimetres, the unit of the image coordinates will be millimetres, too. In this way image point coordinates are simulated that perfectly match the defined control points. Furthermore, the simulation respects the physical properties and rotational elements given in the IAU report (Archinal et al., 2011) as they are stated on page 12.

For later reference it is defined

$$p_\alpha = 1.79, \qquad p_\delta = 1.08, \qquad p_W = 1.42, \qquad \lambda_0 = 0.78 \tag{3.7}$$

and emphasised that $\lambda_0$ is the forced libration amplitude which is focused here while the other three parameter are related to the precession cone. Furthermore are $\alpha_0 = 317.68°$ and $\delta_0 = 52.90°$ the time-independent coordinates of the mean pole axis orientation.

**Falsification of the rotational model**

To test functional model and software, in a first step only parameters of the rotational model above were changed and all other parameters kept unchanged. Rotational parameters like $\lambda_0$, $p_\alpha$ but also $\alpha_0$ and $\delta_0$ were set to

different starting values; the results of the adjustment were then compared to the original input values: the difference of both is the reconstruction error, shown in table 3.1. To give an example, $\lambda_0 \in \{-1, 0, 5\}$ (degree) means, that three different adjustments were performed and the maximal deviation of the final value from 0.78° lies under 0.00012 degree. All other lines in the table represent a single adjustment for a given pair of parameters, $\epsilon = 10^{-4}$ denotes the magnitude of the reconstruction error. The results are obtained with the

**Reconstruction of rotational parameters**

| **parameter** | **start value** | **true value** | **error** | **iterations** |
|---|---|---|---|---|
| $\lambda_0$ | $\{-1, 0, 5\}$ | 0.78 | $< 1.2\,\epsilon$ | $2-4$ |
| $p_\alpha, \lambda_0$ | $(0, 0)$ | $(1.79, 0.78)$ | $(5.4\,\epsilon, 1\,\epsilon)$ | 22 |
| | $(1, -0.1)$ | | $(3.0\,\epsilon, 1\,\epsilon)$ | 17 |
| $\alpha_0, \delta_0$ | $(316.8, 51.9)$ | $(317.68, 52.9)$ | $< 2.4\,\epsilon$ | 4 |
| | $(315, 55)$ | | $< 1.0\,\epsilon$ | 5 |
| | $(300, 40)$ | | $(0.5\,\epsilon, 3\,\epsilon)$ | 9 |

Table 3.1: Different adjustment cases for a simulation with falsified rotational parameters: forced libration amplitude $\lambda_0$, precession cone parameter $p_\alpha$ and mean pole axis orientation $(\alpha_0, \delta_0)$. Columns two to four show the false start values, the true simulated values and the reconstruction errors; $\epsilon = 10^{-4}$, values are in degree. Column five shows the number of iterations that the algorithm needed to perform.

corresponding stochastic model, i.e. negligible errors for camera orientation. Other stochastic models, assuming higher pointing and position errors induce slightly higher reconstruction errors and up to four more iterations. The maximum error of $26\,\epsilon$ corresponds to a model that falsely assumes position errors of 30 m.

**Falsification of the orientation data**

In a next step, errors were also added to the orientation data of the camera. The image point observations themself were not changed.

A pseudo random number $0 \leq r \leq mag$ and a random sign $s = \pm 1$ were

obtained in C/C++ language by setting

$$\begin{aligned} r &= drand48() * mag \\ s &= drand48() < 0.5? -1 : 1\,. \end{aligned}$$

So $sr$ is either positive or negative within magnitude $mag$. The pseudo random generator above creates uniformly distributed numbers. This way, a total of 24 different data sets with falsified initial values were created:

- 4 different data sets with pointing errors (gon) :
  $mag \in \{0.01, 0.05, 0.1, 0.8\}$
- 4 different data sets with position errors (m) :
  $mag \in \{1, 15, 100, 300\}$
- 16 different combinations of the pointing and position error magnitudes above, e.g. (0.01 gon, 1 m); (0.05 gon, 1 m); (0.1 gon, 1 m); (0.8 gon, 1 m) etc.

Now for this falsified data sets, the rotational elements were changed as described before. The adjustment for all eight cases (listed in table 3.1) was repeated for all the 24 data sets with different stochastic models. Because this complexity lead to far more than 300 test runs, the results are summarised rather then having all of them listed:

- In all constructed data sets, the forced libration amplitude could be reconstructed within two to five iterations at an average accuracy level of $4.5\,\epsilon$.
- Pointing data could be reconstructed up to an offset of 0.1 gon with accuracy of $3.5\,\epsilon$ gon.
- Position offsets could not be successfully reconstructed, since the algorithm is more sensitive to pointing corrections. The resulting CPN is shifted in the magnitude of the mean position error.

Summarising it is to say, that the successful reconstruction of the rotational elements did not depend on the position or pointing accuracy. It also did not depend on the successful reconstruction of the topography. The latter

is unexpected and could result from the uniform distribution of pseudo random numbers, so that this independency might not hold in a real data case. Furthermore, the position accuracy must be smaller than the pixel resolution error, in order to successfully reconstruct the CPN. For the data sets which satisfy this condition, the original CPN could be reconstructed within 15 m. The algorithm itself is functional and reasonabe results for data sets within the determined error bounds can be expected.

## Functional dependency of the precession amplitudes

There is a functional dependency of the precession amplitudes $p_\alpha = 1.79$, $p_\delta = 1.08$, $p_W = 1.42$ and the declination of the precession axis $\delta_0 = 52.9$ which shall be discussed briefly. The line that connects the ICRF-Origin with the point $(\alpha_0, \delta_0)$ is the precession axis of the pole, $\alpha_0 = 317.69$ in the current example. The plane which is orthogonal to it contains the complementary precession axes of the other two frame axes. The pole axis describes a circle of radius $p_\delta$ around the precession axis, such that the declination $\delta$ ranges within $[\delta_0 + p_\delta, \delta_0 - p_\delta]$. Simultaneously, the $X$-axis describes a circle of radius $p_\delta$ around the complementary precession axis which affects the value of $W$. For $\delta_0 = 0$ the trajectory of the pole in ICRF coordinates would be circular, but if $|\delta_0| > 0$ it has the shape of an ellipse. Because of the declination above the ICRF equator the pole axis passes through a larger range of longitudes and the change in right ascension is larger than $p_\delta$. The radius of the sphere at latitude $\delta_0$ is given by $\cos(\delta_0)$. The induced trajectory of $+X$ around the complementary precession axis has an elliptic shape, too. This axis has a declination of $\delta_0 - 90$ and the sphere radius at this level is $\cos(\delta_0 - 90) = \sin(\delta_0)$. This leads to the following relationship between the precession constants:

$$p_\alpha = \frac{p_\delta}{\cos(\delta_0)} \quad \text{and} \quad p_W = p_\alpha \sin(\delta_0) = p_\delta \tan(\delta_0), \tag{3.8}$$

where $\delta_0$ is the time independent declination of the precession axis over the ICRF equator and $p_\delta$ the radius of the precession cone. This dependency needs to be considered e.g. when the partial derivatives are built with respect to $\delta_0$.

## 3.5 Numerical stability

It has been noticed that the determinant of the normal matrix equals zero in many cases. This is caused by the large numerical range of the entries, i.e. the magnitude for pointing parameters differs greatly from the magnitude for position parameters. Common implementations of inversion algorithms either refuse the matrix in this case or compute a pseudo-inverse based on rounded values. Using Matlab there will be an error message, that the result is a pseudo-inverse. To avoid an involved pseudo-inverse, a scaling method has been developed that smooths the numerical range and ensures, that the diagonal entries are dominant in magnitude. The application of the scaling increases the numerical stability of the inversion procedure.
Let $N$ the normal matrix and denote with $N_{ii}$ the diagonal entries, $1 \leq i \leq n$. The matrix $D$, defined by

$$D_{ij} = \begin{cases} \sqrt{N_{ii}}^{-1} & i = j\,, \\ 0 & \text{else} \end{cases}$$

is the scaling matrix for $N$. $D$ is applied by multiplication from left and right to $N$, such that

$$\mathcal{N} = D\,N\,D \tag{3.9}$$

is the scaled normal matrix. The multiplication from left scales the rows of $N$ and the multiplication from right the columns. Hence, equation (3.9) is equivalent to the definition

$$\mathcal{N}_{ij} = N_{ij}\,D_{ii}\,D_{jj}\ .$$

If $N$ is in compressed sparse format with $nnz$ non-zero elements, the scaling can be quickly realised in $\mathcal{O}(nnz)$ (in chapter 4, page 53, the compression is explained). When tested with Matlab for a defect $N$, there was no error message for the scaled matrix.
The inverted matrix $\mathcal{N}^{-1}$ needs to be rescaled for the statistical evaluation. It is

$$\mathcal{N}^{-1} = D^{-1}N^{-1}D^{-1}$$

and a repeated application of the transformation (3.9) gives $N^{-1}$. To only calculate the solution $x = N^{-1}b$, the matrix does not need to be rescaled. It is

convenient to scale the vector $b$, too and solve instead of $Nx = b$ the equation

$$\mathcal{N}y = Db \; .$$

The solution is given by

$$\begin{aligned} y &= \mathcal{N}^{-1}Db \\ &= D^{-1}N^{-1}b \\ &= D^{-1}x \end{aligned}$$

and hence, it follows

$$x = Dy \; .$$

# Chapter 4

# Application to Large Data Sets

The data set for the test case in the previous chapter is so small, that considerations about memory and running time requirements were not necessary so far. If it comes to large data sets, the computational effort increases significantly. Therefor the demands on memory and computation time will be analysed in this chapter. Starting with the general structure of the adjustment problem, three different levels of resource optimisation will be discussed. The reference cost level is given by a software and a computing machine of the DLR Berlin for which the resource requirements are available (Preusker; private communication). The data set of Vesta with 5440 images and $\approx$ 83000 control points is used to exemplify the saving in resources by applying the methods of this chapter. The reference machine in all corresponding figures is the *cluster* listed in table 4.1.

The costs with respect to memory and computing time using the reference facilities depend on the number of images as well as on the number of control points. The method, described in section 4.2.2, reduces the main cost factor to the number of images. The method rsp. algorithm, described in section 4.2.4, can be implemented by taking parallel programming into account. The parallel design is discussed in section 4.3 and further reduces the running time by using appropriate parallel computation architecture. Since the performance of the parallel algorithm is machine-depending, additional computing facilities were used for performance tests. The operating system of all machines is Linux Redhat 6, the individual configuration is shown in table 4.1.

In chapter 2, page 34, the adjustment problem is formulated in terms of the

**Configuration of the testing machines**

| | HLRN | Cluster | Work-station |
|---|---|---|---|
| Chip (Intel Xeon) | E5-2670 0 | X 7560 | E5-1607 0 |
| | 2.60 GHz | 2.27 GHz | 3.0 GHz |
| Gflops/s (per core) | 2.99 | 2.14 | 3.04 |
| Cores | 24 | 32 | 4 |
| RAM | 64 GB | 128 GB | 16 GB |

Table 4.1: The values for the processor performance in Gflops/s have been obtained based on the Whetstone benshmark test Curnow and Wichmann (1976).

normal equation $N\hat{x} = b$. Recall, that $N$ is built from the Jacobi matrix $A$ of the functional model and the weight matrix $P$,derived from the stochastic model, by $N = A^T PA$. The solution consists of two components,

1) functional solution: computation of $\hat{x}$

2) stochastic solution: computation of the diagonal elements of $N^{-1}$ and the diagonal of $AN^{-1}A^T$ .

It shall be emphasised again that the stochastic solution is necessary for obtaining the stochastic model aposteori and to evaluate the computed improvements of the functional model.

The following general assumption on data sets is made: The data set to which the bundle block adjustment is applied, consisting of $nIm$ images, $nCP$ control points, and $nObs$ image point observations, respects the relations

$$\frac{nObs}{nCP} \ll nIm \ll nCP \; . \tag{4.1}$$

The assumption states that the average observation rate of the control points is small against the number of images and the latter is small against the number of control points. Typically (4.1) is satisfied, but there might be exceptions which are to exclude from further consideration.
In addition to the notation above, throughout the chapter, $N$ will denote the normal matrix and $dim$ its dimension.

## 4.1 Sparse matrix compression

By construction, the Jacobi matrix $A$ contains mainly zeros in each line and $N$ has only non-zero entries for correlated parameters. A matrix that has a majority of elements which are zero is called a *sparse matrix*, because of its sparse distribution of non-zero elements ($nnz$). Sparse matrices are compressed and used in compressed form since the beginning of programming. The best example is a diagonal matrix $D$ with dimension $n$. In this case the location of all $nnz$ is precisely known, hence the only thing that needs to stored is a vector of length $n$. This is a very efficient compression of $D$. Now for any $m \times n$ matrix $M$ it is true that the more elements are zero the more memory would be wasted if the space for all $mn$ matrix elements would be allocated. Hence a representation of a matrix that saves this memory space is of great advantage. The sparsity $\mathfrak{S}$ of a matrix $M \in \mathcal{M}_{n,m}$ is defined as

$$\mathfrak{S}(M) := \frac{size}{n\,m}\,, \qquad size = \text{number of nnz elements}\,. \tag{4.2}$$

Mathematically $\mathfrak{S}$ is the density of the $nnz$, it will be zero for the zero-matrix and one for a dense matrix. Hence, the smaller $\mathfrak{S}(M)$ is, the less memory space is required to store $M$ and the more meaningful a compression becomes. Since the structure of a matrix is often known beforehand, the density can help a programmer to decide, whether or not a compression is useful.

*Description of a typical compression for a general matrix:* Technically the only information required is where in the matrix the $nnz$ are located. A vector $I_1$ stores the index of the row of each $nnz$ element and another vector $I_2$ keeps the information where in this vector a new column starts.

Example:

$$M = \begin{pmatrix} 1 & 0 & 4 \\ 2 & 3 & 0 \\ 0 & 0 & 5 \end{pmatrix} \qquad \begin{aligned} nnz &= \{1,2,3,4,5\} & n &= 3 \\ I_1 &= \{0,1,1,0,2\} & m &= 3 \\ I_2 &= \{0,2,3,5\} & size &= 5 \end{aligned}$$

Rows and columns are indexed with 0, 1, 2, such that $I_1$ reflects the information that "1" is in row 0, "2, 3" are in row 1, "4" is in row 0 and "5" is in row 2. This vector has always the same number of elements as the $nnz$ vector. The vector $I_2$ has always one elements more than the number of columns, here $m+1 = 4$. The numbers "1", "3" and "4" are the first nnz of column j=0,1,2

| // **A csm in C-code** | |
|---|---|
| 1 | typedef struct csm = { |
| 2 | double *nnz; |
| 3 | int *$I1$, *$I2$; |
| 4 | int n,m,size; |
| 5 | bool ccf; |
| 6 | }; |

Table 4.2: The data type *csm*, a compressed sparse matrix, consists of a pointer to the non-zero elements (2), two pointers to index elements (3), variables to store dimension parameter and size (4), the format specification (5); defined in C-language.

and their index inside the $nnz$ vector is given by $I_2[j]$. Therefore if $j_1 = I_2[j]$ and $j_2 = I_2[j+1]$

$$nnz[j_1] \leq x < nnz[j_2] \ , \qquad I_1[j_1] \leq i < I_1[j_2]$$

yields all non-zero elements $x$ of column $j$ with the corresponding row index $i$. Together with the number of rows $n$ and columns $m$ and the variable $size$ that keeps the length of the $nnz$-vector a compressed sparse matrix (csm) system that represents $M$ is given. The entry at $I_2[m] = size$ ensures a save access of elements on the implementation level. Table 4.2 shows an example of the structure in C-code. The length of $I_1$ is the same as that of $nnz$, but its elements are integer values. It takes 4 Byte to store an integer value and 8 Byte to store a double value. Neglecting the minor requirements of $\mathcal{O}(n)$ for $I_2$ and the data type itself, it takes $\frac{3}{2}size \cdot 8$ Byte to store the csm. Since $8nm$ Byte are required to store the whole matrix and

$$\frac{3}{2}\frac{8size}{8nm} = \frac{3}{2}\mathfrak{S}(M) \ ,$$

memory is only saved, if $\mathfrak{S}(M) < \frac{2}{3}$.

*Formats:* In the given example the compression format is the *column compressed format* (ccf) representing a column major order of $M$. If the column numbers are stored in $I_1$ and the starts of the next row in $I_2$, this results in a row compressed format corresponding with a row major order of $M$. Both

| // | **Example for a simple transposition method** | |
|---|---|---|
| 1 | csmTranspose (csm* M) { | |
| 2 | int $swap = M \rightarrow m$; | |
| 3 | $M \rightarrow m = M \rightarrow n$; | |
| 4 | $M \rightarrow n \ \ = swap$; | // n,m are swapped |
| 5 | $M \rightarrow ccf = !ccf$; | // format is flipped; |
| 6 | }; | |

Table 4.3: C/C++ code example for a transposition of a compressed sparse matrix $M$; lines 2 to 4 swap the variables for rows $n$ and columns $m$, in line 5 the boolean variable $ccf$ is negated which changes $true$ to $false$ and vice versa.

formats are useful, but the most important aspect is that one format is the transposed version of the other. Transpositions are used to a large extent within the adjustment program. Therefore the boolean variable $ccf$ was added to the csm system, such that the order of the representation can be changed quickly. Setting the variable $ccf$ of a column compressed csm from $true$ to $false$ changes $M$ to $M^T$ (which is now row compressed). Table 4.3 shows the code example. If $\mathfrak{S} \geq \frac{2}{3}$, no memory space is saved any more, but there is another important aspect of csm-representation, that still does carry weight – the running time acceleration during matrix multiplication.

The product $AB = C$ of two square matrices $A$ and $B$ can be realised by:

```
1 int i,j,k;
2 for (i = 0, i < n, i++)
3  for (j = 0, j < n, j++)
4  { double a_ij = A[i][j];
5    for (k = 0, k < n, k++)
6     C[i][k] += a_ij B[j][k];
7  } .
```

Now for any $a_{ij} = 0$ in line 4 the following two lines will have no mathematical effect, they only consume runtime of $\mathcal{O}(n)$. If $A$ is a csm, $a_{ij}$ runs only through the nnz-vector performing $size \cdot n$ multiplications. Hence the gain in speed is already $(n^2 - size)n$ flops and the number of operations required

shrinks further if $B$ is also compressed. Therefore the multiplication speeds up considerably. In the extreme case, where $A, B$ and hence $C$ are diagonal, the product is done in just $n$ flops instead of $\mathcal{O}(n^3)$.
Fast csm-products are possible between the following compress format combinations: $[ccf, ccf]$, $[crf, crf]$ and $[ccf, crf]$. If both matrices have the same format, the product itself can be built as a csm in this format, too. Since here, the columns (rsp. rows) are built one after another, both versions are also suitable for parallel computing. For the remaining $[crf, ccf]$ combination the product can be realised too, but not with the same speed up factor. A benefit of this version is, that the output compression format can be chosen, which improves the flexibility in the algorithm's design. Code examples for csm-product routines are given in appendix B.

A square matrix with dimension $dim$ requires $8\,dim^2$ Bytes to be allocated in double point precision. This means the memory need increases, depending on the number of parameters to adjust, in quadratic order $\mathcal{O}(dim^2)$. Because the normal matrix $N$ is symmetric, the allocation can be restricted to elements of the upper triangle of the matrix without loosing information. This reduces the memory space to $8\,dim(dim+1)/2$ Bytes, which is still of quadratic order but saves almost half of the memory. This reduction will be referred to as *triangular allocation*. Figure 3.2 on page 39 shows the distribution pattern of the $nnz$ in $N$. The matrix can be split into four parts

$$\begin{bmatrix} N_{11} & N_{12} \\ N_{12}^T & N_{22} \end{bmatrix} \text{ with } N_{11} \in \mathcal{M}_{dim_1}, N_{22} \in \mathcal{M}_{dim_2} \text{ and } N_{12} \in \mathcal{M}_{dim_1, dim_2},$$

and all required parts can be stored in seperated memory blocks, as shown on figure 4.1. The size of $dim_2$ is given by the number of control points, i.e. $dim_2 = 3nCP$, and $dim_1 = dim - dim_2$ is the number of remaining parameters which agree with the additional observations (compare section 2.3).
In the following is shown how much memory is required for the single parts, depending on $dim_1, dim_2$ and the observation rate of each control point. Based on the concrete values of the Vesta data set, figure 4.2 shows the need for the triangular allocation in total (level 0) and the reduced need if $N_{22}$ is compressed (level 1) respective if $N_{22}$ and $N_{12}$ are compressed (level 2). The data set consists of nIm = 5440 images and nCP = 82 829 control points with an average rate of 9.3 observations per control point, hence nObs = 770 310 observations.

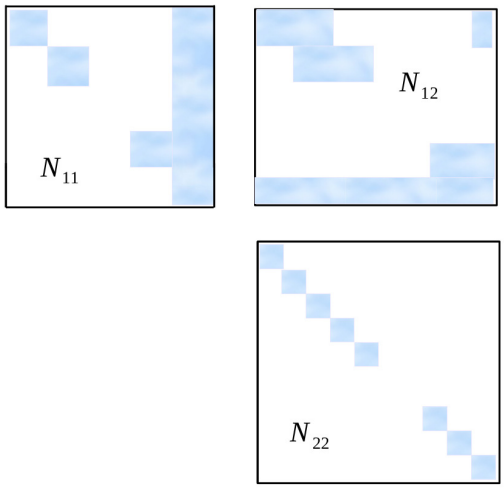



Figure 4.1: The splitting of normal matrix $N$

Out of the 294 GB RAM for the triangular allocation, about 290 GB can be saved, since:

$N_{11}$ : only depends on camera parameters: $dim_1 = 6 \times$ nIm
memory: 3.97 GB

$N_{22}$ : only depends on control points: $dim_2 = 3 \times$ nCP
6 non-zero entries per control point with triangular allocation
memory: 5.69 MB

$N_{12}$ : the correlation between the two groups of unknown parameters
$6 \times 3 \times$ nObs non-zero entries, memory : 105.8 MB

$N_{12}^T$ : the transposed $N_{12}$ matrix does not need to be saved .

The example corresponds to a bundle block adjustment in the body-fixed formulation, described in section 2.3. In this case a csm-representation reduces the memory cost from 290.4 GB to just 111.5 MB (Mega Byte = $1024^2$ Byte). In the inertial frame context $N_{12}$ also contains the correlations of the rotational parameters and the control points. In this case, the number of nnz increases for each rotational parameter by $dim_2$ for $N_{12}$ and by $dim_1$ for $N_{11}$. Since the rotational parameters are very small in number and each only yields a very low contribution to the resource demands (about 2 MB in the example above),

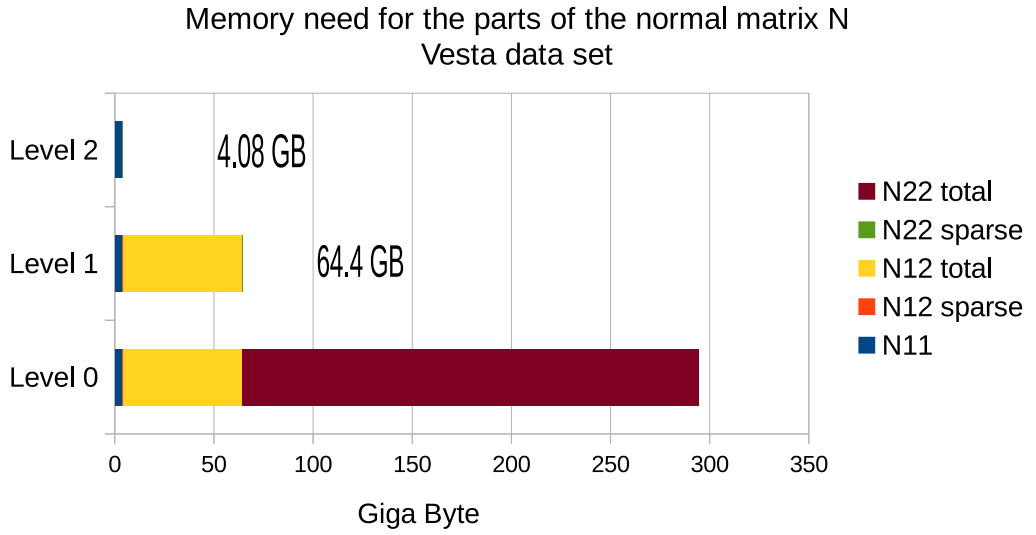



Figure 4.2: Memory need for the single parts of normal matrix $N$ considering a triangular representation – total (level 0), with compressed $N_{22}$ part (level 1), with compressed $N_{22}$ and $N_{12}$ parts (level 2). The need for the compressed matrices ($< 0.2GB$) is below eye resolution.

they will not be considered any further in this context. It is because of the special structure of the adjustment problem, that $N_{22}$ contains only $nnz$ on a $3 \times 3$ block diagonal. The assumption, that the relations (4.1) hold, ensures that $\mathfrak{S}(N_{12})$ is very small, too. The sparsity, defined in (4.2), for both matrices is given by

$$\mathfrak{S}(N_{22}) = \frac{2}{3}\frac{1}{nCP} \quad , \quad \mathfrak{S}(N_{12}) = \frac{1}{nCP}\frac{nObs}{nIm} \, .$$

The two tables on page 59 show how the memory requirements increase with growing data sets. There are three real cases (Phobos, Mercury and Vesta) and one general example listed. In table 4.4 the memory need for a general triangular allocation is compared with the need for the $N_{11}$ part of the matrix; table 4.5 shows the memory savings, if the triangular allocation method and sparse matrix compression are combined, including the requirements for the non-zero elements of $N_{12}$ and $N_{22}$ for different data sets.
Remark: Considering 30000 images one would exceed 120 GB of RAM, so there will be a natural hardware limit at some point. There is some extra

**Memory requirements to store normal matrix $N$**

| | **Images** | **Complete $N$** | | **Part $N_{11}$ of $N$** | |
|---|---|---|---|---|---|
| **Data Set** | | dim | memory | $dim_1$ | memory |
| Phobos | 72 | 2496 | 23.7 MB | 432 | 0.71 MB |
| Mercury | 3446 | 48192 | 8.65 GB | 20676 | 1.59 GB |
| Vesta | 5440 | 281127 | 294.42 GB | 32640 | 3.97 GB |
| Example | 10000 | 1532640 | 8750.6 GB | 60000 | 13.4 GB |

Table 4.4: Memory need to store the complete normal matrix $N$ (triangular allocation) and the $N_{11}$ part of $N$ (see figure 4.1), in four different cases. The requirements increase with the number of images in quadratic order.

**Advantage of using the splitting technique and compression**

| | **Control** | **Part $N_{12}$ and $N_{22}$** | | **Total Memory** | |
|---|---|---|---|---|---|
| **Data Set** | **Points** | no csm | csm | needed | saved |
| Phobos | 688 | 23 MB | 1 MB | 1.78 MB | 21.92 MB |
| Mercury | 9172 | 7 GB | 13 MB | 1.60 GB | 7.05 GB |
| Vesta | 82829 | 290 GB | 112 MB | 4.08 GB | 290.34 GB |
| Example | 500000 | 8737 GB | 641 MB | 14.02 GB | 8736.6 GB |

Table 4.5: Accumulated memory requirements for $N_{12}$ and $N_{22}$: without compression (*no csm*) and with sparse matrix compression (*csm*) and the total memory to store $N$, if the splitting technique is used together with compression (incl. $N_{11}$ part, see table 4.4 above). In the case Example a point observation ratio of 9:1 is assumed, values are rounded.

space required for the data structure of the compressed matrix. Of course, also $N_{11}$ could be compressed, but in order to solve the adjustment problem, there is space required for matrix operations. The method which is going to be described now, uses the memory region of $N_{11}$ to perform the required transformations. Hence, with respect to the memory resource, the number of involved images is the most significant factor.

## 4.2 Optimisation of running time

Having completed the considerations of the memory resource, the same evaluation is done with respect to running time. As described in chapter 1, it is measured in flops and the concrete times depend on the specifications of the computing machine.

### 4.2.1 The splitting technique

Using the subdivision scheme for $N$ as before and assuming non-zero determinants of $N, N_{11}$ and $N_{22}$, the inverse of $N$ is according to Bronstein and Semendjajew (2005) given by:

$$N^{-1} = \begin{bmatrix} N'_{11} & -N'_{11}N_{12}N_{22}^{-1} \\ -N_{22}^{-1}N_{12}^{T}N'_{11} & N'_{22} \end{bmatrix} \tag{4.3}$$

with

$$\begin{aligned} N'_{11} &= (N_{11} - N_{12}N_{22}^{-1}N_{12}^{T})^{-1} \in \mathcal{M}_{dim_1}\,, \\ N'_{22} &= N_{22}^{-1}N_{12}^{T}N'_{11}N_{12}N_{22}^{-1} + N_{22}^{-1} \in \mathcal{M}_{dim_2}\,. \end{aligned}$$

In Niemeier (2008) the principle is applied within a body-fixed bundle adjustment; since both sources use it without proof, a proof is attached in the appendix (see A.1). The computation of $N^{-1}$ according to this pattern, requires two inversions and four products[1]. The inversions and the first two products form a task sequence that will be called *forward operation*. The single tasks are listed in table 4.6 with their magnitude in flops, the final column shows the reduction in costs if sparse compression is used. The sequence of the remaining two products will be called *backward operation*.

The direct inversion of $N$ requires $\mathcal{O}(dim^3)$ flops and for $dim = dim_1 + dim_2$

[1] Additions are neglected here.

| Task | | without Compression | with Compression | Gain Factor |
|---|---|---|---|---|
| $N_{22}^{-1}$ | inv. | $\mathcal{O}(dim_2^3)$ | $\mathcal{O}(dim_2)$ | $1/dim_2^2$ |
| $N_{12}N_{22}^{-1}$ | prod. | $\mathcal{O}(dim_1 dim_2^2)$ | $\mathcal{O}(nObs)$ | $\mathfrak{S}(N_{12})\mathfrak{S}(N_{22})$ |
| $N_{12}N_{22}^{-1}N_{12}^T$ | prod. | $\mathcal{O}(dim_1^2 dim_2)$ | $\mathcal{O}(dim_1^2)$ | $\mathfrak{S}(N_{22})$ |
| $N'_{11}$ | inv. | $\mathcal{O}(dim_1^3)$ | $\mathcal{O}(dim_1^3)$ | 1 |

Table 4.6: The task sequence of the forward operation to compute $N^{-1}$ in split mode (see eq. 4.3). Columns two and three show the magnitude of required flops without and with sparse matrix compression, column four the factor of flop cost reduction using compression.

the identity

$$dim^3 = (dim_1 + dim_2)^3 = dim_1^3 + dim_2^3 + 3dim_1^2 dim_2 + 3dim_2^2 dim_1$$

holds, such that the cost gain is not achieved by the splitting technique itself, but by the sparsity of $N_{12}$ and $N_{22}$.
Since $N_{22}$ is a block matrix, its inverse is obtained fast in $\mathcal{O}(nCP)$ and can be even done in parallel. Using $N_{22}$ in compressed form and optimising the forward products reduces the costs for the whole forward operation to $\mathcal{O}(dim_1^3)$ flops. It remains the expensive backward operation of $\mathcal{O}(dim_1^2 dim_2)$. The application of the splitting technique with a partial compression is implemented within the reference software and reduces a running time of 40 days to just 1.5 days, but this optimisation can be improved further.

### 4.2.2 The inverse decomposition method

To a symmetric matrix $M \in \mathcal{M}_n$ a pair $(L, D)$ of matrices, where D is diagonal and L lower triangular, that satisfies $M = LDL^T$ is called the *Cholesky LDLT decomposition* of $M$. It is named after the founder of the widely implemented and used algorithm that transforms $M$ into $L$ (Dahmen and Reusken, 2008, chapter 3 p. 87). Since $L$ is triangular, it can be inverted quickly with only $n(n+1)/2$ flops.
**Claim:** For a given cholesky decomposition $(L, D)$ of $M$, it is

$$M^{-1} = (L^T)^{-1} D^{-1} L^{-1} = (L^{-1})^T D^{-1} L^{-1} . \tag{4.4}$$

**Proof**: It is by definition $M^{-1}M = I$, which is equivalent to

$$\begin{aligned} & M^{-1} \quad LDL^T &&= I \\ \iff & M^{-1} \quad LD &&= (L^T)^{-1} \\ \iff & M^{-1} \quad L &&= (L^T)^{-1}D^{-1} \\ \iff & M^{-1} &&= (L^T)^{-1}D^{-1}L^{-1}\,. \end{aligned}$$

This is the first equality, to see the second we use $I = I^T$ and replace

$$\begin{aligned} I = (L\,L^{-1})^T &= (L^{-1})^T\,L^T \quad | \times (L^T)^{-1} \\ \iff (L^T)^{-1} &= (L^{-1})^T \qquad \square \end{aligned}$$

This principle will now be combined with the split technique. Let

$$\bar{N}_{11} := N_{11} - N_{12}N_{22}^{-1}N_{12}^T\,,$$

the matrices which need to be inverted can be written as

$$L_1 D_1 L_1^T = \bar{N}_{11} \tag{4.5}$$

$$L_2 D_2 L_2^T = N_{22} \tag{4.6}$$

using their Cholesky decompositions. Consequently

$$\bar{N}_{11}^{-1} = (L_1^{-1})^T D_1^{-1} L_1^{-1} \tag{4.7}$$

$$N_{22}^{-1} = (L_2^{-1})^T D_2^{-1} L_2^{-1}\,. \tag{4.8}$$

Replacing $N'_{11} = \bar{N}_{11}^{-1}$ in equation (4.3) leads to

$$N^{-1} = \begin{bmatrix} (L_1^{-1})^T D_1^{-1} L_1^{-1} & -(L_1^{-1})^T D_1^{-1} L_1^{-1} N_{12} N_{22}^{-1} \\ -N_{22}^{-1} N_{12}^T (L_1^{-1})^T D_1^{-1} L_1^{-1} & N_{22}^{-1} + N_{22}^{-1} N_{12}^T (L_1^{-1})^T D_1^{-1} L_1^{-1} N_{12} N_{22}^{-1} \end{bmatrix}$$

which can be split into the product

$$N^{-1} = \begin{bmatrix} (L_1^{-1})^T D_1^{-1} & 0 \\ -N_{22}^{-1} N_{12}^T (L_1^{-1})^T D_1^{-1} & (L_2^{-1})^T D_2^{-1} \end{bmatrix} \begin{bmatrix} L_1^{-1} & -L_1^{-1} N_{12} N_{22}^{-1} \\ 0 & L_2^{-1} \end{bmatrix}\,.$$

By defining

$$S := \begin{bmatrix} L_1^{-1} & -L_1^{-1} N_{12} N_{22}^{-1} \\ 0 & L_2^{-1} \end{bmatrix}\,,\; D := \begin{bmatrix} D_1^{-1} & 0 \\ 0 & D_2^{-1} \end{bmatrix}$$

a decomposition of $N^{-1}$ is found, that satisfies

$$N^{-1} = S^T D S \; . \tag{4.9}$$

This is of great advantage, since the decomposition is sufficient to solve the adjustment problem. The proof of sufficiency will be given in the next subsection. It can be seen in the definition of $S$, that $N'_{11}$ does not need to be computed fully and the large matrix $N'_{22}$ does not need to be computed at all. The product of the triangular matrix $L^{-1}$ with $N_{12}N_{22}^{-1}$ can be realised in $\mathcal{O}(18nObs \cdot dim_1)$ flops, if both matrices are given as csm, but additional memory is required here. It will be shown, that these additional costs can be saved, too, by writing $S$ as

$$S = \begin{bmatrix} L_1^{-1} & 0 \\ 0 & I_{dim_2} \end{bmatrix} \begin{bmatrix} I_{dim_1} & -N_{12}N_{22}^{-1} \\ 0 & L_2^{-1} \end{bmatrix} . \tag{4.10}$$

In fact, there are no backward products required with this method. The tasks which are to perform are

- decomposing $N_{22}$ and getting $L_2^{-1}, D_2^{-1}$
- computing $-N_{12}N_{22}^{-1}$ and $\bar{N}_{11}$
- decomposing $\bar{N}_{11}$ and getting $L_1^{-1}, D_1^{-1}$ .

Figure 4.3 shows how effective this method is; for the large Vesta data set the method here (level 2) is compared with the previous level of optimisation (level 1). About 36 hours, 96 % of running time in this example, are saved.

### 4.2.3 Sufficiency of the inverse decomposition

Given the matrix scheme of the inverse normal matrix $N^{-1} = \left[N'_{11}, N'_{12}; (N'_{12})^T, N'_{22}\right]$ and its decomposition $(S, D)$ as before, it is to show that the functional and the stochastic solution (see page 52) of the adjustment problem can be obtained. To solve the normal equation for $\hat{x}$, the vectors $\hat{x}$ and $b$ are split into two parts of $[dim_1, dim_2]$ accordingly to the split of $N$. then $\hat{x} = N^{-1}b$ becomes

$$\begin{pmatrix} \hat{x}_1 \\ \hat{x}_2 \end{pmatrix} = \begin{pmatrix} N'_{11} & N'_{12} \\ (N'_{12})^T & N'_{22} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

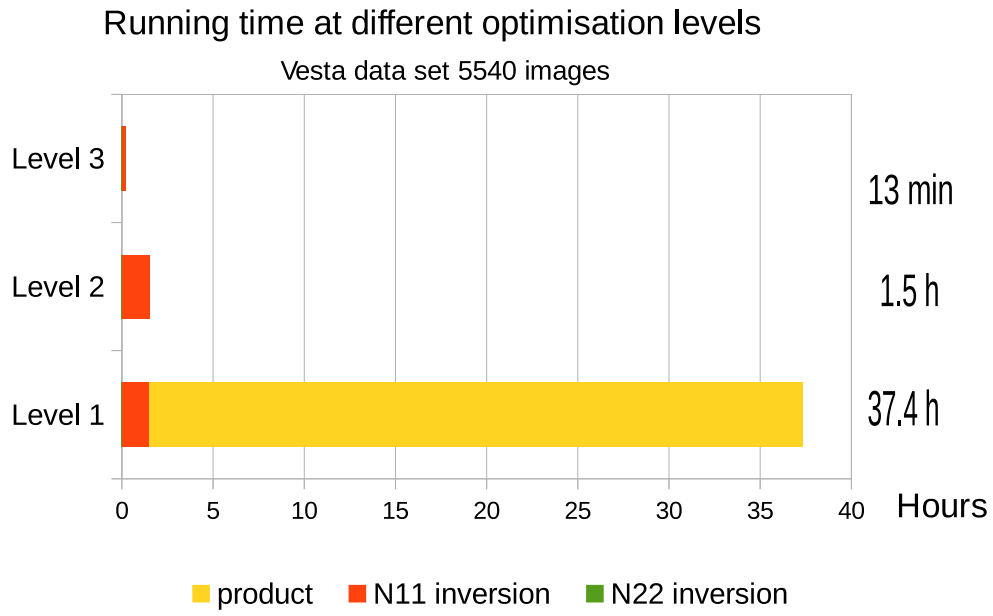



Figure 4.3: Optimisation of inverting $N$ with split technique – level 1: reference software with partial compression, level 2: method described in subsection 4.2.2, level 3: method described in subsection 4.2.4 using parallel computation. The inversion of $N_{22}$ takes only a few seconds.

and hence by definition of $N^{-1}$

$$\begin{aligned}\hat{x}_1 &= N'_{11}b_1 - N'_{11}N_{12}N_{22}^{-1}b_2 \\ \hat{x}_2 &= -(N_{12}N_{22}^{-1})^T\mathbf{N'_{11}b_1} + (N_{12}N_{22}^{-1})^T\mathbf{N'_{11}N_{12}N_{22}^{-1}b_2} + N_{22}^{-1}b_2\,.\end{aligned}$$

The variables in $\hat{x}_2$ (printed in bold) equal $-\hat{x}_1$, such that the equations above can be simplified as follows

$$\hat{x}_1 = N'_{11}(b_1 - N_{12}N_{22}^{-1}b_2) \tag{4.11}$$

$$\hat{x}_2 = N_{22}^{-1}b_2 - (N_{12}N_{22}^{-1})^T\hat{x}_1 \tag{4.12}$$

From the equations above it can be seen, that the computation of $N'_{12}$ is not necessary, further on equation (4.11) can be realised in two steps

$$\begin{aligned}y_1 &= D_1^{-1}L_1^{-1}(b_1 - N_{12}N_{22}^{-1}b_2) \\ \hat{x}_1 &= (L_1^{-1})^T y_1\,.\end{aligned}$$

Hence, the decomposition of $N'_{11}$ is sufficient. The same holds for $N_{22}^{-1}$, since the parts of its decomposition can be applied to $b_2$ sequentially in the same way. The product with vectors is obtained quickly, esp. if the matrix on the left hand side is sparse and compressed. Also recall, that in this case the involved transpositions only require zero-runtime[2] (compare section 4.1 on page 55).

It remains to show that the diagonal of the HAT-matrix $H := AN^{-1}A^T$ can be computed. By associative law it is

$$H = A(S^T\,D\,S)A^T = (AS^T)\,D\,(SA^T)\,.$$

Since $(SA^T)^T = AS^T$, a similar decomposition is found for $H$. Setting $L_H := SA^T$ yields to $H = L_H^T D L_H$. Now let $h_i$ be column $i$ of $L_H$, then

$$H_{ii} = h_i^T D h_i = \sum_j D_{jj} \cdot h_i(j)^2$$

is the $i$-th diagonal entry of $H$.
Hence, the calculation of $L_H$ is sufficient. Denoting with $n = |l|$ the length of

[2] i.e. less than 1 second

vector $l$, the matrix $L_H$ has $dim \times n$ elements. If $L_H$ is computed columnwise, only the space for a vector of length $dim$ needs to be allocated, which can be reduced even further.
As mentioned earlier, the computation of $L_1^{-1}\,(-N_{12}N_{22}^{-1})$ can be avoided. Using equation (4.10) leads to

$$L_H = \begin{pmatrix} L_1^{-1} & 0 \\ 0 & I_{dim_2} \end{pmatrix} \left[ \begin{pmatrix} I_{dim_1} & -N_{12}N_{22}^{-1} \\ 0 & L_2^{-1} \end{pmatrix} A^T \right]$$

and the right hand product with $A^T$ involves only very sparse matrices. Once again, each column $a(i)$ of $A^T$ is split into $[a_1(i), a_2(i)]$ of dimension $dim_1$ and $dim_2$; the first part corresponds with additional observation parameters only and the second contains the remaining control point derivatives. The equation above becomes

$$\begin{aligned} h_i &= \begin{pmatrix} L_1^{-1} & 0 \\ 0 & I_{dim_2} \end{pmatrix} \begin{pmatrix} a_1(i) - N_{12}N_{22}^{-1}a_2(i) \\ L_2^{-1}a_2(i) \end{pmatrix} \\ &= \begin{pmatrix} L_1^{-1}(a_1(i) - N_{12}N_{22}^{-1}a_2(i)) \\ L_2^{-1}a_2(i) \end{pmatrix} \end{aligned}$$

and the lower part of the right hand vector $L_{22}a_2(i)$ consists of only 3 non-zero elements for all $i$. Since the upper part is bound by $dim_1$, $h_i$ can be stored in compressed form of maximal $dim_1 + 3$ elements. Hence, again memory as well as running time is saved by taking into account the sparse nature of $A$.

### 4.2.4 An iterative inverse decomposition algorithm

The decomposition of the partial matrix $\bar{N}_{11}$ consumes still running time of cubic order, although the dimension of the problem has been reduced to $dim_1$. For large data sets this is a relevant cost factor, esp. since a bundle block adjustment consists of more than just one iteration. Table 4.7 shows the running times on all testing machines with respect to the inversion of $\bar{N}_{11}$ for the previously considered data sets of Phobos, Mercury and Vesta (table 4.4 on page 59); the values are based on $n^3/3$ flops with dimension parameter $n = 6nIm$. It can be seen, that for 10 000 images just four iterations take 1–2 days of computation time. Therefore an algorithm was developed which makes use of the multi-core architecture of modern computing machines and high performance

**Running time for inversion of partial matrix $\bar{N}_{11}$**

| Data Set | Images | HLRN | cluster | work-station |
|---|---|---|---|---|
| Phobos | 72 | 00:00:00 | 00:00:00 | 00:00:00 |
| Mercury | 3446 | 00:16:25 | 00:22:57 | 00:16:09 |
| Vesta | 5440 | 01:04:37 | 01:30:17 | 01:03:33 |
| Example | 10000 | 06:41:20 | 09:20:45 | 06:34:44 |

Table 4.7: Running times (hh:mm:ss) for different data sets based on $n^3/3$ flops for partial matrix $\bar{N}_{11}$, $n$ is sixfold number of images. See table 4.1 on page 52 for description of the three machines.

computers of the North-German Supercomputing Alliance. Depending on the hardware that is used a saving factor of $\backsim$ 1:7 (cluster) rsp. 1:8 (HLRN) could be achieved for this part of the adjustment program. The testing machines are described on page 52.
The algorithm transforms a symmetric matrix $M$ iteratively into a matrix $S$ and computes simultaneously a diagonal matrix $D$, such that $S^T D S = M^{-1}$; $S$ is a block triangular matrix and the dimension of each block can be bound by an arbitrary given block size.

Compared with the definition of a block triangular matrix on page 2.1, the condition that all blocks need to have the same dimension is relaxed here in such a way, that the block structure is bound by $s$. That means, the dimension of all blocks is less or equal to $s$. The generalisation is owing to the fact, that the dimension of an arbitrary matrix is not necessarily a multiple of an arbitrary given block size $s$. But if for a matrix $M \in \mathcal{M}_{dim}$ the dimensions of the diagonal blocks are bound by $s$, this matrix can be naturally embedded into a larger matrix $M'$, that has the following properties:

$$dim = n\,s + r,\ r < s \implies M' \in \mathcal{M}_{(n+1)s} \text{ with } M' = \begin{pmatrix} M & 0 \\ 0 & I_{s-r} \end{pmatrix}.$$

Since $I_{s-r}$ is a unit matrix, it follows

$$(M')^{-1} = \begin{pmatrix} M^{-1} & 0 \\ 0 & I_{s-r} \end{pmatrix}.$$

Both, $M'$ and $(M')^{-1}$, are block triangular with block size $s$. Such an embedding can always be done and hence, the block size can be chosen freely. Although the algorithm will be applied especially to $\bar{N}_{11}$, the proof here is for a general symmetric $M$ for which $M^{-1}$ exists. In case the inverse does not exist, the algorithm stops with an error message.
Following the steps of a complete induction, the algorithm is described for an arbitrary block size $s \leq dim$ with $n = dim/s$ number of diagonal blocks. W.l.o.g. $dim/s \in \mathbb{N}$ holds.

**Algorithm for $n = 1$ and $s = dim$:**
This is simply equation 4.4 with $S = L^{-1}$ and $D = D^{-1}$.

**Algorithm for $n = 2$ and $s = dim_1 = dim_2 = dim/2$:**
This is the split technique and requires only one iteration. The algorithm performs the following steps as a **forward operation**:

0: extract $M_{22}$ block and compute its LDLT form: $M_{22} = L_2 D_2 L_2^T$
invert the UTM $L_2^T$ and replace $M_{22}$ with the transposed $L_2^{-1}$

1: compute $B_2 = -M_{12}M_{22}^{-1}$ and replace $M_{12}$ with $B_2$ ([3])

2: compute $\bar{M}_{11} = M_{11} - M_{12}M_{22}^{-1}M_{12}^T$ (updating $M_{11}$)

3: repeat step 0 on $\bar{M}_{11}$ : $(L_1^{-1})^T D_1^{-1} L_1^{-1} = \bar{M}_{11}^{-1} = M'_{11}$
$\Rightarrow$ replace $\bar{M}_{11}$ with $L_1^{-1}$

After this steps the **backward operation** is to perform

- replace $B_2$ with $B'_2 = L_1^{-1} B_2$

The obtained matrices look like this:

$$S = \begin{pmatrix} L_1^{-1} & B'_2 \\ 0 & L_2^{-1} \end{pmatrix}, \; D = \begin{pmatrix} D_1^{-1} & 0 \\ 0 & D_2^{-1} \end{pmatrix}.$$

[3] Assuming that a copy of $M_{12}$ exists; this assumption will be droped later on.

Keeping the definition of $B_2, B_2'$ in mind, it follows that

$$\begin{aligned} S^T D S &= \begin{pmatrix} (L_1^T)^{-1} D_1^{-1} L_1^{-1} & (L_1^T)^{-1} D_1^{-1} B_2' \\ (B_2')^T D_1^{-1} L_1^{-1} & (B_2')^T D_1^{-1} B_2' + (L_2^{-1})^T D_2^{-1} L_2^{-1} \end{pmatrix} \\ &= \begin{pmatrix} M_{11}' & M_{11}' B_2 \\ B_2^T M_{11}' & B_2^T M_{11}' B_2 + M_{22}^{-1} \end{pmatrix} \\ &= M^{-1} \end{aligned} \tag{4.13}$$

**Algorithm for arbitrary $n$ and $s = dim/n$:**
Setting the original matrix $M = M_n$, a *brief description* is given by:
Start with calculation of $L_n^{-1}, D_n^{-1}$. Complete the forward operation and set $\bar{M}_{11} = M_{n-1}$. Repeat the forward procedure until $M_1$, the upper left $16 \times 16$ block, is reached and compute the last elements $L_1^{-1}, D_1^{-1}$.

*Detailled description*: For general $n \in \mathbb{N}$ the forward steps are

0: $(n-1)$ times - extract diagonal block and compute $L_k^{-1}, D_k^{-1}, n \geq k \geq 2$

1: $(n-1)$ times - compute $B_k = -(M_{12} M_{22}^{-1})^{(k)}, n \geq k \geq 2$

2: $(n-1)$ times - building $\bar{M}_{11} =: M_{k-1},\ n \geq k \geq 2$

3: repeat step 0 on $M_1$

After the forward operation terminates, $M$ is

$$\begin{pmatrix} L_1^{-1} & B_2 & \dots & * & * \\ 0 & L_2^{-1} & \dots & B_{n-1} & * \\ & & \dots & * & B_n \\ & 0 & & L_{n-1}^{-1} & * \\ & & & 0 & L_n^{-1} \end{pmatrix} \tag{4.14}$$

and $d := (D_1^{-1}, D_2^{-1}, \dots, D_n^{-1})$ is the connected diagonal.

For the $n-1$ steps of the backward operation, write $D^{(k)}$ for the diagonal matrix with diagonal $d_k = (D_1^{-1}, \dots, D_k^{-1})$ and set again $S_1 = L_1^{-1}$. Starting with the square in the upper left corner of the matrix

- replace $B_2$ with $B_2' = S_1^{-1}B_2$ and set $S_2 = \begin{pmatrix} S_1^{-1} & B_2' \\ 0 & L_2^{-1} \end{pmatrix}$

For the case $n = 2$ it was showed that $S_2^T D^{(2)} S_2 = M_2^{-1}$. Analog

- replace $B_k$ with $B_k' = S_{k-1}B_k$ and set $S_k = \begin{pmatrix} S_{k-1} & B_k' \\ 0 & L_k^{-1} \end{pmatrix}$

for the remaining $k \leq n$. This way, the matrix (4.14) changes stepwise to

$$\begin{pmatrix} S_2 & B_3' & \dots & * \\ 0 & L_3^{-1} & \dots & B_n \\ & & \dots & * \\ & & 0 & L_n^{-1} \end{pmatrix}, \dots, \begin{pmatrix} S_{n-2} & B_{n-1}' & * \\ 0 & L_{n-1} & B_n \\ 0 & 0 & L_n^{-1} \end{pmatrix}, \begin{pmatrix} S_{n-1} & B_n' \\ 0 & L_n^{-1} \end{pmatrix} \tag{4.15}$$

and $S_k^T D^{(k)} S_k = M_k^{-1}$ holds for each $k$. The final step $n-1 \mapsto n$ is evaluated:

$$S := S_n = \begin{pmatrix} S_{n-1} & B_n' \\ 0 & L_n^{-1} \end{pmatrix},\ D := D^{(n)}$$

and it follows

$$\begin{aligned} S^T D S &= \begin{pmatrix} S_{n-1}^T D^{(n-1)} S_{n-1} & S_{n-1}^T D^{(n-1)} B_n' \\ (B_n')^T D^{(n-1)} S_{n-1} & (B_n')^T D^{(n-1)} B_2' + (L_n^{-1})^T D_n^{-1} L_n^{-1} \end{pmatrix} \\ &= \begin{pmatrix} M_{n-1}^{-1} & M_{n-1}^{-1} B_n \\ B_n^T M_{n-1}^{-1} & B_n^T M_{n-1}^{-1} B_n + (L_n^{-1})^T D_n^{-1} L_n^{-1} \end{pmatrix} \\ &= \begin{pmatrix} M_{11}' & -M_{11}' M_{12} M_{22}^{-1} \\ -M_{22}^{-1} M_{12}^T M_{11}' & M_{22}^{-1} M_{12}^T M_{11} M_{12} M_{22}^{-1} + M_{22}^{-1} \end{pmatrix} \\ &= M^{-1}. \end{aligned} \tag{4.16}$$

The equality of line 2 and line 3 is obtained after renaming the variables according to the case of the single split scenario $n = 2$ which holds for arbitrary dimensions $dim_2 > 0, dim_1 = dim - dim_2$. □

Because of the iterative structure, the backward operation can not be omitted here, but matrix products are suitable for parallel computation. The decomposition and inversion of the diagonal blocks can be achieved quickly if the block

size is small enough. A size of $s = 16$ was chosen which will be referred to as the *tile size*. Because of the specification regarding the memory transfer on a computing unit, this is currently the optimal tile size in parallel programming (Hsu and Kremer, 2004).
Applying the decomposition algorithm to $\bar{N}_{11}$, the running time reduces further if the number of parallel processes is sufficiently large. On figure 4.3 the time for level 3 corresponds with this optimisation, the 1.5 hours are reduced to 13 minutes of computation time. Subsection 4.2.5 is dedicated to the topic of parallel computation in general and section 4.3 explains the implementation design of the algorithm in detail.

### 4.2.5 Parallel computation

On hardware that is equipped with multiple core processors, several tasks can be performed in parallel at the same time. A problem is suitable for parallel computation, if it can be broken in tasks which are independent of each other (task parallel) and if the memory parts that need to be changed by the tasks are disjoint (data parallel); it might be required to synchronise certain tasks which need to be performed on the same memory region successively (Munshi and Gaster et al., 2012).
The responsibility to avoid conflicts and to synchronise overlapping tasks lies with the Software designer. The programming language and the compiler need to be capable of managing parallel programming threads. A simple way of using parallel computing in C language, is the inclusion of the *pthread* library. For more complex tasks that require a deeper control of thread-administration, there are the application programming interfaces OpenMP and OpenCL. Both are capable of task parallelism and data parallelism, but with OpenCL one can execute code on Graphics Processing Units (GPU), too.
For the inversion of triangular matrices parallel computation with *pthread* is used within the implementation of the bundle block adjustment. Here, the columns can be computed independently of each other. The part of the program where the $3 \times 3$ block matrix $N_{22}$ is inverted also involves *pthread*. Since each block can be inverted independently, the task is very suitable for parallel computing, too. The speed-up factor matches the number of available parallel threads $nThreads$, if the problem size is an (integral) multiple of $nThreads$.

Another field, for which parallel computation is used within the software, is matrix multiplication: To compute the product $AB$ for $A \in \mathcal{M}_{n,m}, B \in \mathcal{M}_m$ one needs $m^2$ operations for each line, $nm^2$ operations in total. Since each line can be computed independently, one can divide the task in different groups which act parallel. In the case, were the problem size $n$ agrees with the number of available threads, all lines can be computed at the same time. More typical is a realisation where the memory region of the matrices is structured in multiple blocks of a certain dimension, the so called *tile size*, and each block is computed in parallel. Not all matrix products here are done in parallel, but the principle is used within a parallel version of the iterative decomposition algorithm (4.2.4). The implementation was realised in OpenCL and is discussed in the next section. This section closes with a comparison of the running times for the Vesta using the introduces decomposition methods. The running time of one complete iteration of the bundle adjustment, including the set up of the sparse system and the statistic evaluation is about 34 minutes on the supercomputer (HLRN). On the multi-core cluster it takes about 60 minutes and on the work-station about 90 minutes. Figure 4.4 shows the running times of one iteration with the method of subsection 4.2.2 alone (right bar) and with the parallel version of the iterative decomposition algorithm of subsection 4.2.4 in addition (left bar). All other aspects are the same. The speed up factor due to the parallel computation shrinks to 1:3 for HLRN and to 1:2 for the cluster. The acceleration factor in the previous chapter is given with respect to the part of the program, which formerly consumed most of the running time, the computation of $N^{-1}$ rsp. its decomposition. Considering the whole iteration on the HLRN, the evaluation takes about half of the running time now and the decomposition only a quarter. The adjustments for Vesta took four iterations with a total running time of 2h:16min on HLRN, about four hours on the cluster and about six hours on the work-station. In this example the factor of memory reduction is 1:16, with respect to running time about 1:20 at optimisation level 2 and about 1:40 at level 3 (using the cluster).

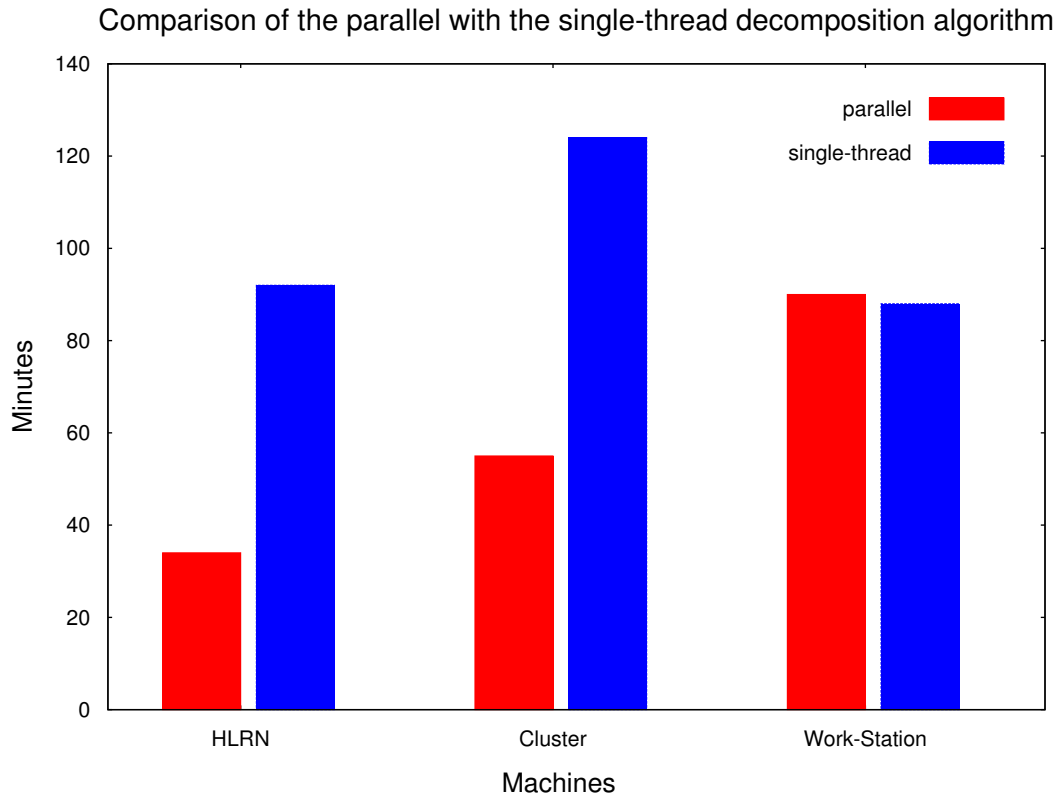



Figure 4.4: Running times for one iteration of the inertial frame bundle block adjustment, processing the Vesta data set on different hardware – using the parallel decomposition algorithm (left bar) and the single-thread decomposition (right bar).

## 4.3 Implementation of the decomposition algorithm in OpenCL

The algorithm in 4.2.4 requires the use of both, multiplication and inversions of triangular matrices, on a large scale. Special efforts were taken to ensure a balanced workload on the computation device which is recommended in Munshi and Gaster et al. (2012). It could be achieved, that the forward and the backward operation can be performed at the same time (task parallelism) – after the point of synchronisation for both operations has been passed for each iteration.

### 4.3.1 Memory design

The algorithm is applied to a symmetric matrix $M \in \mathcal{M}_{dim}$ transforming it into a block triangular matrix $S$. The dimension of the blocks of $S$ respectively the tile size is denoted with $T$. For all $dim \in \mathbb{N}$ there exist natural numbers $n$ and $r \leq T$ such that $dim = (n-1)T + r$. As stated on page 67, $M$ can be embedded into a larger $M'$ and $M' \in \mathcal{M}_{nT}$. Hence, if necessary $M$ is enlarged to dimension $nT$ such that it consists of $n^2$ blocks of size $T \times T$. In a worse case scenario this would lead to an overallocation of $(n+T-1)*T-n = (T+n)(T-1)$ elements which is of $\mathcal{O}(n)$ and can be neglected.
A continuous memory region, that can hold the non-zero elements of a block triangular matrix with block size $T$ and dimension $nT$, is allocated. This corresponds with $\frac{1}{2}n(n+1)T^2$ elements. The consequence of saving the memory of the zero-blocks is, that it must be thought of a memory design in order to access each element of $M$. The $T \times T$ sub-blocks are sequentially numbered from 0 to $\frac{1}{2}n(n+1) - 1$ and a coordinate system, comparable to that of a matrix, is used to access them. There are ordered in $n$ column-groups $cg$ and $n$ row-groups $rg$, as it is shown on figure 4.5.

The numbers of the first blocks in each column-group $j$, form a numerical sequence 0,1,3,6,... given by the rule

$$id_j = \sum_{i=1}^{j} i = \frac{1}{2}j(j+1), \quad j = 0, 1, \ldots, n-1$$

and the final block for each group is given by the number

$$id_j + j, \quad j = 0, 1, \ldots, n-1 \ .$$

Hence, $M[id_j T^2]$ will access the first entry of cg $j$, while $M\ [\ (\ id_j\ +\ j\ )\ T^2\ ]$ refers to the first entry of the diagonal block. This allows to navigate through the matrix which is partitioned into equal sized small $T \times T$ blocks, optimised for parallel programming. The values of $M$ are transferred to the memory block according to this pattern.

### 4.3.2 Tile size

Meanwhile a lot of publications describe the topic of the tile size in detail. Considering the publication Hsu and Kremer (2004), which examines the influence

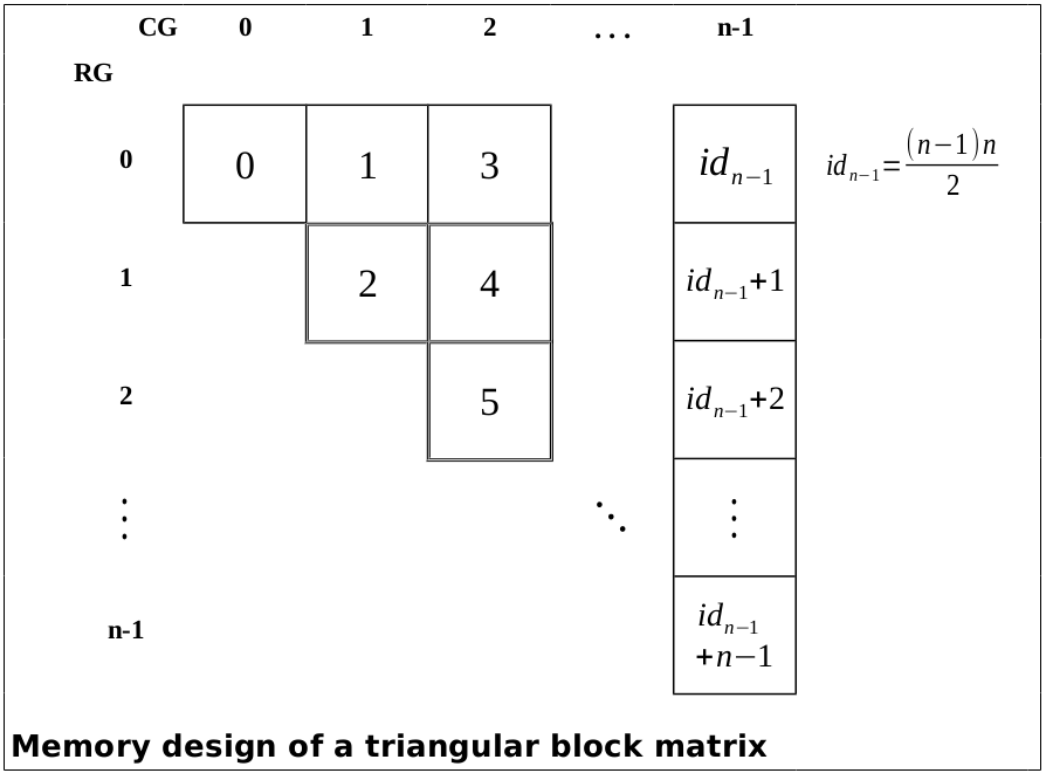



Figure 4.5: Memory design of a block triangular matrix. The sub-blocks are numbered from 0 to $\frac{1}{2}n(n+1)-1$ and ordered in a matrix-like structure. The coordinates $CG$ and $RG$ are used to identify each id via $CG(CG-1)/2+RG$.

of the tile size on the performance of parallel multiplication, the term *tile* refers to the dimension of the sub blocks of the product which are computed by a single thread. It can be translated into *block* size within the current context. According to the result there, 16 is an optimal tile size, based on some relevant hardware properties of the computing device. A performance analysis of the algorithm for different values for T, revealed that for $T = 6$ the performance is worse and confirmed that $T = 16$ is an optimal choice. Despite the fact, that $T = 6$ would be an intuitive choice, the tile size was therefore set to 16.

### 4.3.3 Work-groups and work-items

The functions that are executed in parallel on the OpenCL device are called *kernels*. Each instance of a kernel is called *work-item* and has a unique global ID. The number of requested work-items can be larger than the number of

factual parallel cores on the device; in this case the surplus of requests is queued until an item has finished its work and a new instance can be called. Hence, a work-item corresponds with a *thread* of the *pthread* libraries. The work- items are organised in *work-groups* which ensure that all items of the group execute on the same compute unit. In the simplest case the number of items in each work-group is just one. For the CPU-version of the algorithm, which is focused and described here, this is the recommended configuration. According to the memory design (figure 4.5) each work-group is, depending on the task which is to perform, assigned to a column-group $cg$ or a row-group $rg$ and operates on the specified memory region. The number of work-groups $nWG$ is set to

$$nWG = n - 1$$

and corresponds with the number of images via

$$6nIm = (n-1)T + r \Longrightarrow nWG = \frac{6}{T} nIm - 1 \ .$$

This implies that there are less work-groups required than images are given. Since the number of work-groups is limited to 8892 (for Intel) about 23 700 images can be processed without increasing the tile size $T$. If $T$ must be increased, it should be doubled to 32.

### 4.3.4 The parallel decomposition algorithm

The number of required iterations is $n - 2$. For each iteration $k$ consider the forward steps according to *detailed description* on page 69. The following tasks within each step can be separated in independent (data parallel) groups:

**0:** inverting the triangular $[16, 16]$ matrix $L_k$
groups: maximal 16, each inverting one column

**1:** computing $B_k$: $[16, (k-1)] \times [16, 16]$
groups: $k - 1$, each computing one block $(b_{ki})$ of $B_k$

**2:** computing the $\frac{1}{2}(k-1)k$ blocks of $M_{12}M_{22}^{-1}M_{12}^T$: $[16, (k-1)] \times [16, (k-1)]$
groups: $k - 1$, each group computes $k/2$ (resp. $(k-1)/2$) blocks
design: group $i = 0, \ldots, k-2$ operates on cg $i$ (and $k - 2 - i$, if $i < \frac{k}{2}$)

The algorithm starts at $k = n - 1$ and stops at $k = 1$, step 0 is then repeated separately on the remaining block $L_0$. Hence the number of the required work-groups decreases in each iteration.
*Note (1:)* If step 1 is performed at once, one needs to store $B_k$ temporarily beforehand. The work-item of work-group $i$ computes the product in two steps:

$$\textbf{1a)} \quad \tilde{b}_{ki} = (\text{block } i \text{ of } M_{12}) \times (L_k^{-1})^T \tag{4.17}$$

$$\textbf{1b)} \quad b_{ki} = \tilde{b}_{ki} \times (D_k^{-1}\, L_k^{-1}) \; . \tag{4.18}$$

There is no lack of performance, if (4.18) is done after step 2 has finished. The storage of $B_k$ becomes unnecessary, the product in step 2 simplifies to $\tilde{B}_k D_k \tilde{B}_k^T$ and the completion of step 1b) completes the forward operation.

*Note (2:)* There is only one column-group $j = \frac{k}{2} - 1$ that has exactly $\frac{k}{2}$ blocks.[4] The column-groups left of it have less and the column-groups to the right contain more blocks. For each $cg < \frac{k}{2}$ there is a partner $cg' = k - 2 - cg$, such that their number of block adds up to $k$. Hence, both can share the workload easily.

Backward operation: The transformation of the backward products, see (4.15) on page 70, requires $n - 1$ iteration steps. As described before, they would need to be performed after all steps of the forward operation are completed. Recall that $B'_k = S_{k-1} B_k$ has to be computed in each iteration for $k = 1, \ldots, n$. Since with growing $k$ the dimensions of $S_{k-1}$ and $B_k$ increase, the workload increases for each iteration. It is true that by definition each of the $S_k$ is a factor of all sequentially following $S_j$, $j > k$. Indeed

$$\begin{aligned}
S_1 &= \begin{bmatrix} S_0, & 0 \\ 0, & I \end{bmatrix} \begin{bmatrix} I, & B_1 \\ 0, & L_1^{-1} \end{bmatrix} = \begin{bmatrix} S_0, & S_0 B_1 \\ 0, & L_1^{-1} \end{bmatrix} \\
S_2 &= \begin{bmatrix} S_1, & 0 \\ 0, & I \end{bmatrix} \begin{bmatrix} I, & B_2 \\ 0, & L_2^{-1} \end{bmatrix} = \begin{bmatrix} S_1, & S_1 B_2 \\ 0, & L_2^{-1} \end{bmatrix} \\
&\ldots \\
S_n &= \begin{bmatrix} S_n, & 0 \\ 0, & I \end{bmatrix} \begin{bmatrix} I, & B_n \\ 0, & L_n^{-1} \end{bmatrix} = \begin{bmatrix} S_{n-1}, & S_{n-1} B_n \\ 0, & L_n^{-1} \end{bmatrix} .
\end{aligned} \tag{4.19}$$

Now each $B_k$ consists of k blocks $b_{k0}, \ldots, b_{k(k-1)}$, placed above the diagonal block of column-group $k$ which holds $L_k^{-1}$. In particular it is $B_1 = b_{10}$, $B_2 =$

[4] It needs to distinguish between odd and even $k$.

$(b_{20}, b_{21})^T$ and $S_0 = L_0^{-1}$. If the $S_k$ are sequentially replaced by their definition, multiplying everything out leads to

$$
\begin{aligned}
S_1 &= \begin{bmatrix} L_0^{-1}, & L_0^{-1} b_{10} \\ 0, & L_1^{-1} \end{bmatrix} \\
S_2 &= \begin{bmatrix} L_0^{-1}, & L_0^{-1} B_1, & L_0^{-1}(b_{20} + b_{10}\, b_{21}) \\ & L_1^{-1}, & L_1^{-1} b_{21} \\ 0 & & L_2^{-1} \end{bmatrix} \\
S_3 &= \begin{bmatrix} L_0^{-1}, & L_0^{-1} b_{10}, & L_0^{-1} b'_{20}, & L_0^{-1}(b'_{30} + b_{10} b'_{31}) \\ & L_1^{-1}, & L_1^{-1} b_{21}, & b'_{31} \\ 0 & & L_2^{-1}, & L_2^{-1} b_{32} \\ & & & L_3^{-1} \end{bmatrix}.
\end{aligned}
$$

This identity can is used for an alternative approach to realise the backward operation. To explain the idea, consider $S_3$ above: after the transformation is done the first time for the final column, the memory block $b_{32}$ is no longer required and can be replaced with $L_2^{-1} b_{32}$. Similar in the next iteration, the blocks $b_{21}$ and $b'_{31}$ are no longer involved in further calculations; the can be replaced with $L_1^{-1} b_{21}$ and $L_1^{-1} b'_{31}$. This way the matrix $S$ is built from bottom to top and both operations can be done in parallel. As a consequence, the decomposition is obtained faster if enough parallel cores are available. The transformation is explained in greater detail within the appendix (page 106). Hence, the backward operation is obtained by

$$b_{ji} \mathrel{+}= b_{ki} b_{jk} \text{ for all column-groups } j > k, \ \forall i = 0, \ldots, k$$

For each $j > k$ a work-group can be set up and launched parallel to the forward operation after $B_k$ is computed (step 1*b* on page 77). If $k = n-1$ the number of backward work-groups is zero, if $k = n-2$ the number is one and so on. Since the number of forward work-groups decreases from $n-2$ to one simultaneously, there is a constant number of $n-2$ work-groups throughout the time. This way an optimal balance of the workload is ensured for parallel computation. Figure 4.6 illustrates the specific design that realises backward computation on column-groups $j \geq k$ and the next iteration of forward calculation on column-groups $j < k$ (see figure 4.6). Hence, this is an example of task parallelism and step 1*b* works as a synchronisation point between both.

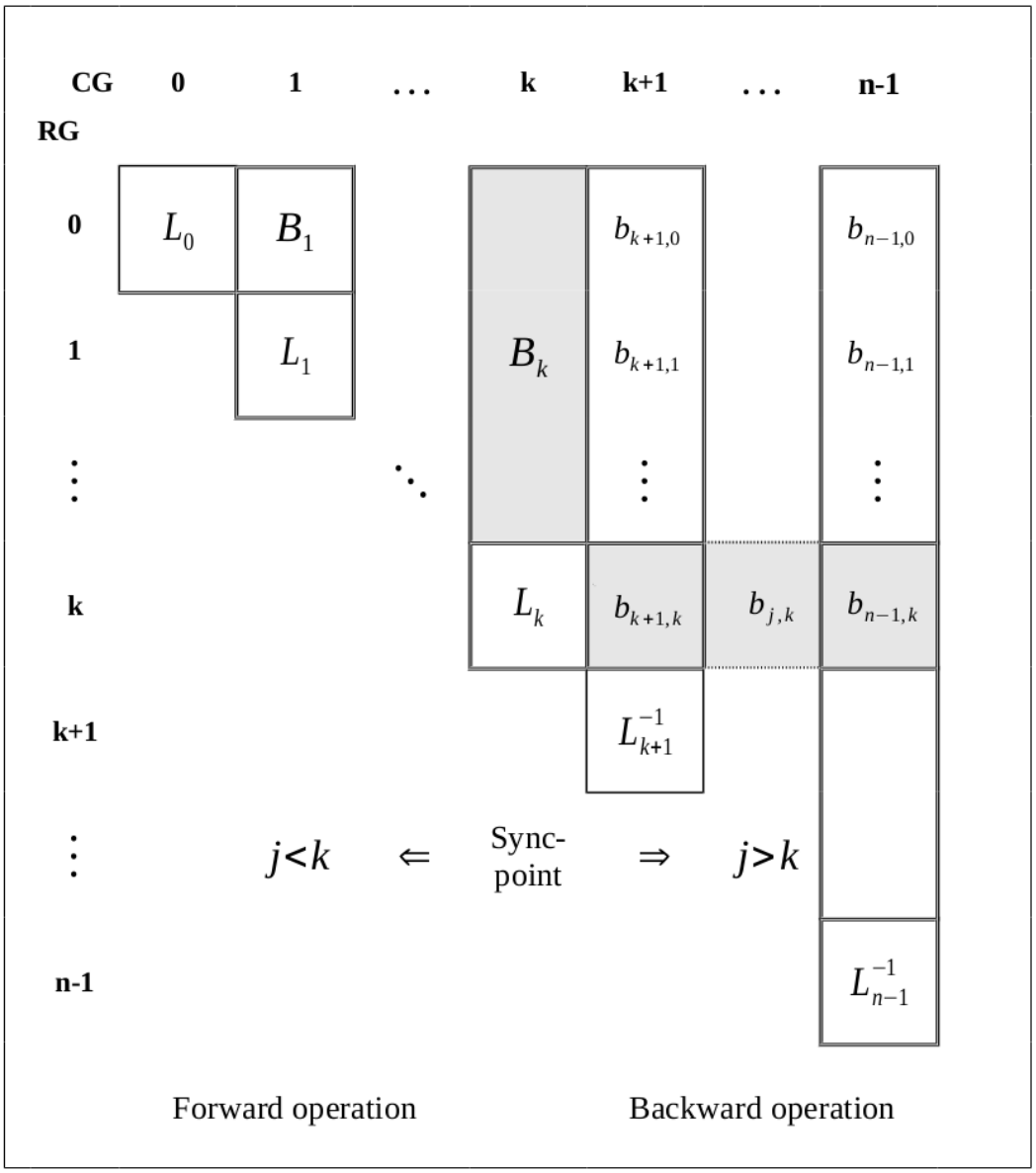



Figure 4.6: Backward and forward operation in task-parallelism. The backward operation is performed on the column-groups $j > k$ and the forward operation on $j < k$. The point of synchronisation between both tasks is the completion of the transformations on column-group $CG = k$.

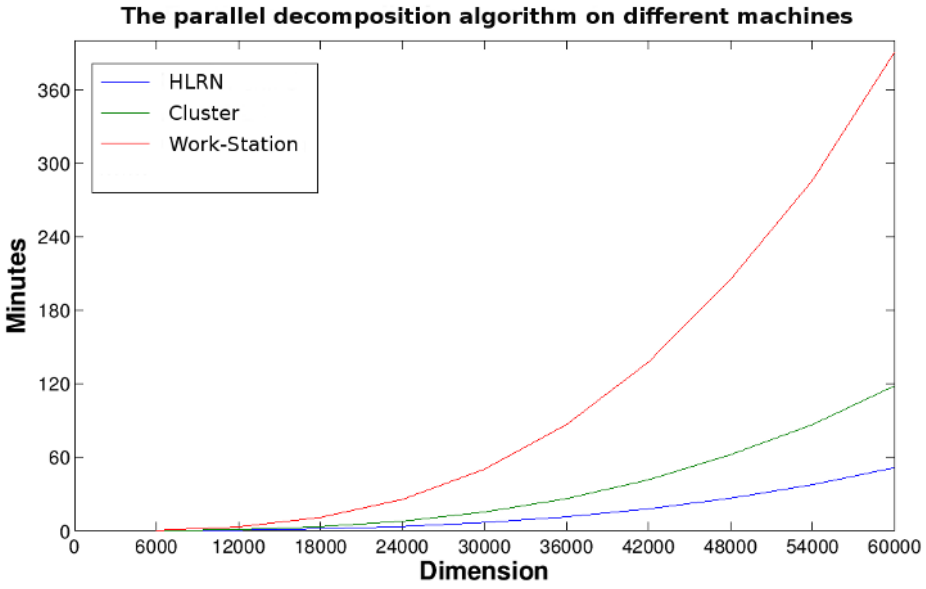



Figure 4.7: Parallel decomposition algorithm on different computing machines (specifications on page 52); the horizontal axis shows the dimension of the quadratic matrix.

Figure 4.7 shows the running times of the OpenCL algorithm on the three machines for matrices with dimensions from 6000 to 60 000. This corresponds with data sets of 1000 to 10 000 images. For the quad-core PC no acceleration could be noticed - the times are almost identical to the non-iterative decomposition approach described in 4.2.2. Here the number of cores is not sufficient to benefit from the iterative and parallel algorithm. The processor performance of the HLRN with about 3 Gflops/s per core and the 24 physical cores lead to a speed up factor of approximately 1:8. The DLR cluster is equipped with more cores but the processor performance is only 2.14 Gflops/s per core; it takes almost twice the time of the HLRN and the local speed factor of 1:7 is lower. The comparison shows that both quantities, the number of cores and the performance factor need to be considered.

# Chapter 5

# Application to the science cases Phobos and Vesta

Credit: The data sets described and evaluated here, were produced by Dr. Konrad Willner (Phobos) and Frank Preusker (Vesta) who have carried out the photogrammetric measurements as well as the tie point matching across the image bundle. A data set is a list of image objects as shown in table 5.1, the position and pointing data of the involved cameras were obtained from NASA and ESOC.

| **Image observation** | | | **Camera Orientation** | | | |
|---|---|---|---|---|---|---|
| Image number | Focal length | Image time | Position | X | Y | Z |
| *Control point* | *Image point* | | | | | |
| ID | Sample | Line | Pointing | $\phi$ | $\omega$ | $\kappa$ |
| ... | ... | ... | | | | |

Table 5.1: Template of an image object within a data set. Image number, time and focal length as well as position and pointing data of the camera are assigned as meta-data to each object; the list of image point observations in camera coordinates together with a unique ID of the projected control point completes the format.

## 5.1 Phobos

The Phobos data set consists of 689 control points and 8010 image point measurements, distributed over 19 Viking Orbiter and 53 Mars Express images. Willner et al. (2010) computed a control point network of 665 points with an average accuracy of 40 m and a forced libration amplitude of 1.24°. The image data has been described in the introduction.
The analysis of the heterogeneous data is challenging due to the huge differences between the uncertainties of the cameras' orientations.

### Weighting model and pre-adjustment

**Weights:** Since the MEX positions are far more accurate than the Viking positions, the stochastic model is not homogeneous. Furthermore the quality of SRC images is much higher, esp. with respect to the contrast values. To compensate the latter the uncertainty for MEX observations was set to one pixel (0.009 mm) and the uncertainty for Viking observations to the twofold pixel size (0.02352 mm).
The weighting model for the SRC orientation data is taken from Willner et al. (2010), see A.1 on page 108. The uncertainties range from 1 m to 1000 m for positions and from 0.01 gon to 0.5 gon for the pointing angles. For the Viking images, the position weights were initially set to 5 km for close range images and to 24 km for the two images at far distance from late 1977. For the pointing uncertainty a constant value of 0.4 gon was adopted. This completes the initial weighting model.
**Pre-adjustment:** Firstly, the Viking orientation were adjusted in a preparation step. For the pre-adjustment the parameters $\alpha_0$ and $\delta_0$ were included with a uncertainty of 0.2° and the libration amplitude $\lambda_0$ with a uncertainty of 1° in a joint adjustment. By this means of control, the initial values for the rotational elements did not change. The Viking orientation were replaced with the new values and the resulting weights were replaced in the stochastic model for the Viking frames only. This way an updated data set with an updated weighting model was obtained; the appendix contains both as table A.2 (weights) and table A.3 (orientation).

### 5.1.1 Forced libration amplitude

Recalling the rotation model of Phobos (as given on page 12):

$$\begin{aligned} \alpha(t) &= \alpha_0 - 0.108T + 1.79\sin(M1) \\ \delta(t) &= \delta_0 - 0.061T - 1.08\cos(M1) \\ W(t) &= 35.06^\circ + 1128.8445850t + 8.864T^2 - 1.42\sin(M1) - \lambda_0\sin(M2) \end{aligned}$$

with the time independent parameters

$$\alpha_0 = 317.68^\circ, \qquad \delta_0 = 52.90^\circ, \qquad \lambda_0 = 0.78^\circ \ .$$

After the preparation three different types of adjustments were performed, with the weighting model resulted from the pre-adjustment. In the first case, only the forced libration amplitude $\lambda_0$ was included in the functional model: the given start value of 0.78° was adjusted to 1.13° (±0.013).
In order to test the independence of this result from other parameters, two additional cases have been studied. In a second case, only the constant pole coordinates ($\alpha_0,\ \delta_0$) were resolved: a change in right ascension of 0.07° and in declination of only 0.003° was noticed. In the third case all three parameters have been adjusted jointly. Here $\hat{\lambda}_0$ was computed with 1.132° (±0.0133), confirming the result of the first case. The results of these three adjustments are listed in table 5.2. In all three cases the rotational elements are included as complete unknown parameters like the control points and unlike the orientation parameters which undergo a weight control.
The result for $\hat{\lambda}_0 = 1.13^\circ$ is in agreement with $1.09^\circ \pm 0.1^\circ$ (Oberst et al., 2014) and $1.24^\circ \pm 0.15^\circ$ (Willner et al., 2010), and very close to 1.1°, the result of the theoretical consideration of Rambaux et al. (2012).

### 5.1.2 Control point network for Phobos

The computed CPN of the adjusted data set consists of 680 points. Considering the first adjustment case, the average norm of the weight vector $(\sigma_X, \sigma_Y, \sigma_Z)$ is about 55 m and the forward intersection error of the control points is about 13 m (table 5.3 shows the CPN statistics for all three cases).
Figure 5.2 shows a three dimensional model of Phobos, which was created from 679 control points (Burmeister, 2016a). The point cloud is triangulated and

**Results for rotational elements** (pre-adjusted Viking)

| case | param. | start val. | end value | diff. | $\sigma$ |
|---|---|---|---|---|---|
| Libration | $\lambda_0$ | 0.78° | 1.13° | +0.35° | 0.013° |
| Pole axis | $\alpha_0$ | 317.68° | 317.75° | +0.07° | 0.018° |
| | $\delta_0$ | 52.9° | 52.897° | −0.003° | 0.01° |
| Joint adj. | $\lambda_0$ | 0.78° | 1.132° | +0.352° | 0.0133° |
| | $\alpha_0$ | 317.68° | 317.654° | −0.026° | 0.0130° |
| | $\delta_0$ | 52.9° | 52.885° | −0.015° | 0.01° |

Table 5.2: Results for three separate adjustments (case) of rotational elements (param.) with initial values (start value) are shown in col. 4 (end value); col. 5 shows the difference and col. 6 the uncertainty ($\sigma$). In the first case only libration amplitude $\lambda_0$ is adjusted, in the second case only the mean pole axis orientataion ($\alpha_0,\ \delta_0$), case 3 is a joint adjustment of all the three parameters. The joint adjustment affects the changes in ($\alpha_0,\ \delta_0$) but not the change in $\lambda_0$.

**CPN Statistics** (for results in table 5.2)

| case | $s_0$ | mean error | fwi error |
|---|---|---|---|
| Libration | $5.09 \cdot 10^{-11}$ | 55.65 m | 13.35 m |
| Pole axis | $5.50 \cdot 10^{-11}$ | 60.39 m | 14.10 m |
| Joint adj. | $5.06 \cdot 10^{-11}$ | 55.55 m | 13.35 m |

Table 5.3: Control point network statistic: Col. 1 as in table 5.2, col. 2 holds the overall system error $s_0$, the CPN mean error in col. 3 is computed from $||(\sigma_X, \sigma_Y, \sigma_Z)||$ of the adjusted points, col. 4 shows the forward intersection error. In all three cases, the network consists of 680 points.

rendered with IDL; shading and light source simulation which are needed to make topographic features visible were done with CloudCompare Viewer. The large Stickney crater on the western hemisphere is clearly to see in the upper left image (leading side) and partly to see in the lower left image (northern hemisphere).

The 3D control points have been converted to polar coordinates and their distribution is shown in figure 5.1. The figure also shows the different ranges

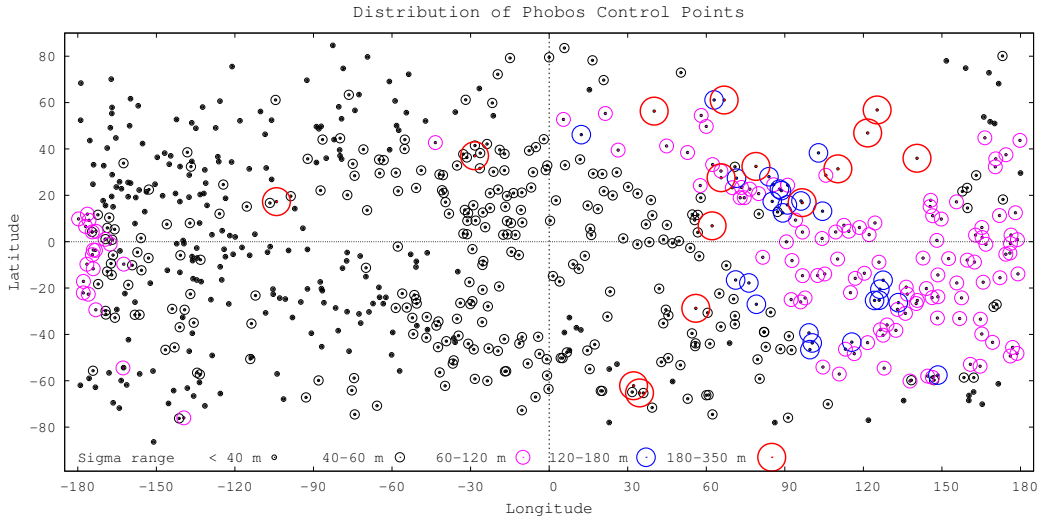
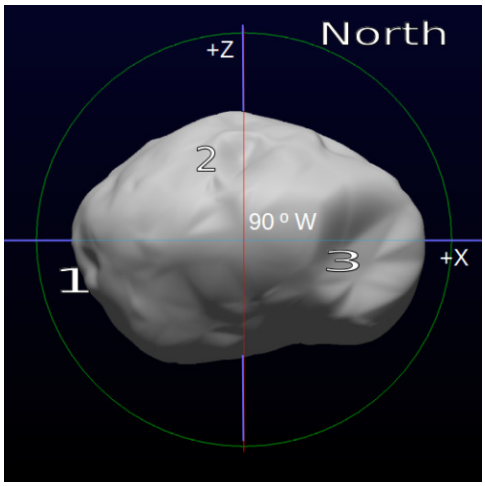
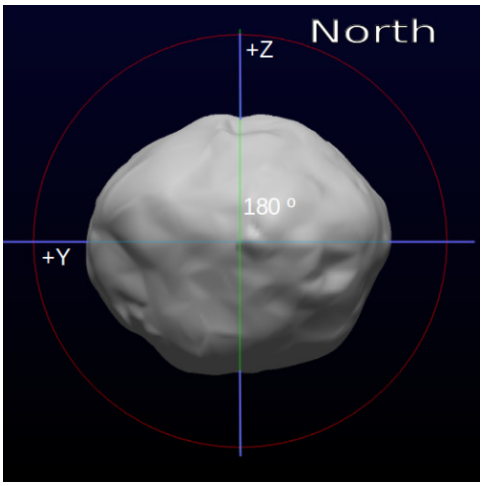



Figure 5.1: Distribution of control points on Phobos (unit sphere projection) with sigma values from 30 m to 345 m. The majority of points has values under 60 m, the CPN mean error is 55.65 m; in the region between 60° and 100° East on the northern hemisphere Viking and MEX images overlap.

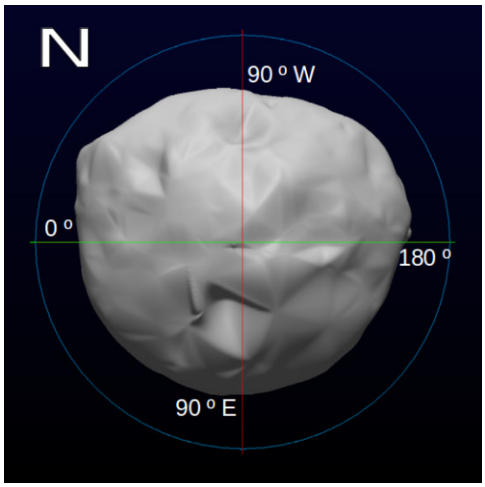
of $\sigma = ||(\sigma_X, \sigma_Y, \sigma_Z)||$ for each control point separated in four classes: $\sigma_0 = 60$ m and smaller, $2\sigma_0$, $3\sigma_0$ and above. There are 15 points over 180 m, most of them located in the north-eastern quadrant. The larger sigma values occur in the region where Viking and MEX images overlap and for the CPs in images close to that. The CPN solution is consistent with the previously obtained by Willner et al. (2010). The difference in the libration amplitude $\lambda_0$ of 0.11 degree leads to a maximal positional offset of about 100 m for the MEX images, measured in the body-fixed reference frame. The average effect on the positions is about 30 m, if all images are considered. As far as comparable, both CPNs differ in parts up to 260 m. The accumulated uncertainty of the points is about 100 m; further deviations can be explained with a possibly different stochastic model for the Viking images (which is not available for comparison). The solution obtained here with the inertial frame method is closer to the result of a body-fixed adjustment that only involves MEX SRC images.



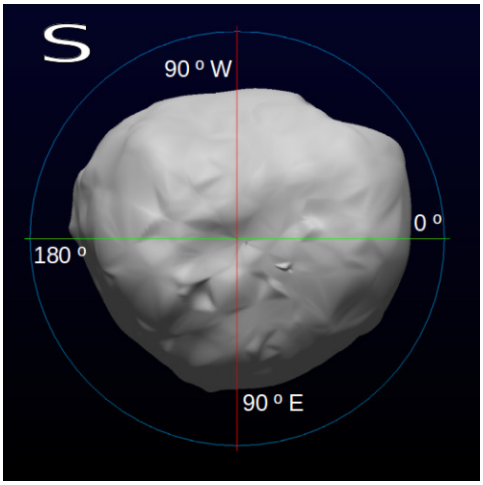

Western hemisphere showing the craters 1) Todd, 2) Drunlo and 3) Stickney




View centred at 180° (vertical) and equator (horizontal), with Todd on the right half below the equator




Northern hemisphere with four longitudes $(0°, 90°E, 180°E, 90°W)$




Southern hemisphere with four longitudes $(0°, 90°W, 180°E, 90°E)$

Figure 5.2: Reconstructed shape of Phobos from 679 control points, triangulated and rendered with simulated light source. The four different perspectives are projections into fundamental planes with the view centred at the third axis. +X intersects the prime meridian, +Y intersects 90° E and +Z contains the northern pole.

## 5.2 Vesta

There are 82 806 control points distributed over 5440 images and the average observation rate is 9.3 per point. Unlike the Phobos data, here all images are taken by the same spacecraft at distances which are close to the average distance of 944.5 km (table 1.5 on page 21). The distance is measured with respect to the centre of mass. The weighting model is provided by Preusker (private data contribution). For all images the position uncertainty is 35 metres and the pointing uncertainty is 0.006 gon. The images do not completely cover Vesta around the northern pole region.

### 5.2.1 Pole axis orientation

The current rotational model for Vesta is simple, the only dynamical term is the self rotation period of about 5 h 20 min. Parameters of interest are here the coordinates $(\alpha, \delta)$ of the pole axis orientation. Two different models will be considered in this subsection. The first agrees with the Dawn-Claudia system stated on page 12. The uncertainty of the pole axis orientation is given with 0.01° (Archinal et al., 2013). The second model, proposed by Preusker (private communication), was applied during the preprocessing of the data set which is used here:

$$\begin{aligned} \text{(Preusker)} \quad \alpha(t) &= 309.03312^\circ \\ \delta(t) &= 42.22623^\circ \\ W(t) &= W_0 + 1617.3331237^\circ\, t\,, \quad W_0 = 74.6625^\circ \quad . \end{aligned}$$

The inertial frame bundle block adjustment was performed for both models. The uncertainty for $\alpha$ and $\delta$ was set to 0.03 degree ($3\sigma$). The rotation rate and the value for $W_0$ remained constant.

Table 5.4 lists the results and the difference with respect to the starting values for both adjustments . For the first model, the changes for both, $\alpha$ and $\delta$ are less than 0.01°. For the second model, the change in right ascension is slightly above 0.01° but closer to the IAU value for $\alpha$. Both results confirm the current IAU pole axis orientation of (309.031°, 42.235°) within the given range of 0.01°. Compared with each other, both results differ by 0.0019° (alpha) and 0.0074° (delta), which is within the computed error of ±0.008°. The overall system error $\sigma_{apost}$ of both inertial frame solutions is about $1.4 \cdot 10^{-6}$ .

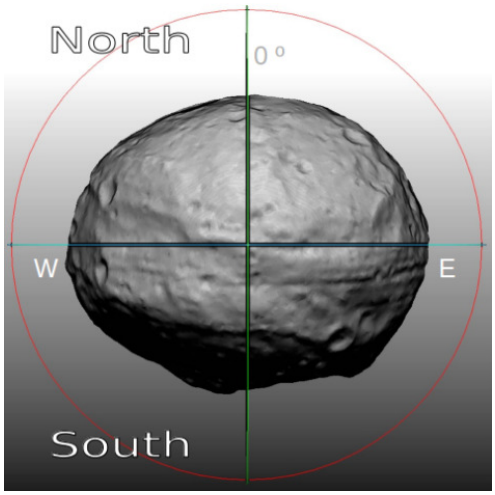
**Adjusted pole axis orientation for Vesta**

| **init. model** | Dawn-Claudia | | Preusker | |
|---|---|---|---|---|
| **parameter** | **result** | **change** | **result** | **change** |
| $\alpha$ | 309.0246° | −0.0064° | 309.0227° | −0.0104° |
| $\delta$ | 42.2296° | −0.0054° | 42.2222° | −0.0040° |
| **difference of both results**: | $\Delta\alpha = 0.0019°$ , | | $\Delta\delta = 0.0074°$ | $< 0.008°$ |

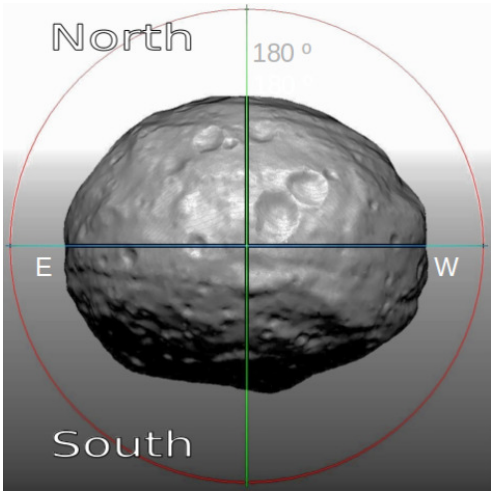
Table 5.4: Adjusted pole axis orientation of Vesta (*result*) ±0.008° for two different initial models, *change* indicates the difference to the initial values. Both results are, compared with the IAU model (309.031°, 42.235°), within the uncertainty range of 0.01° and, compared with each other, within 0.008°.

### 5.2.2 Comparison with previous CPN solution

Since here, all input and output data of the previous bundle block adjustment in the body-fixed reference frame are available, the solution can be compared in detail with the adjustment solution formulated in the inertial frame. Hence, Preusker's body-fixed solution and the results, obtained here by the inertial frame method, are compared with respect to the camera orientation data and the coordinates of the control points. The difference in right ascension and declination of both solutions is $\Delta\alpha = 0.0104°$, $\Delta\delta = 0.004°$. The effect of this difference on the positions of the body-fixed solution is in average 76 m and about 0.009 gon on the pointing angles. Hence, the difference of these parameters to the parameters of the inertial frame solution should not be significantly larger. The adjusted positions differ by ∅ 42 m and the pointing angles by ∅ 0.0042 gon. Finally, the coordinates of the control points only differ by 15 m in average, respecting the given error range of ∅10 m for both CPNs. This shows, that both networks are plausible. It has to be emphasised that the assumed uncertainty values of (30 m, 0.006 gon) for the camera orientation data can not compensate a difference of 0.01° in the pole axis orientation. Taking the uncertainty of the rotational model into account, the body-fixed orientation uncertainties would need to be increased.
Figure 5.3 shows three perspectives of a 3D model that has been created from the adjusted CPN (Burmeister, 2016b). The views are projections into stan-

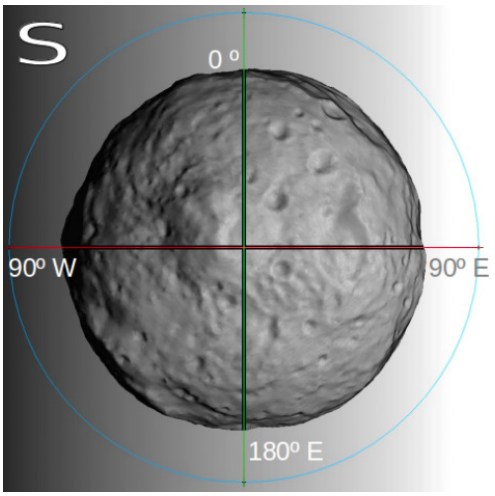



View centred at the prime meridian (vertical) and equator (horizontal)




Left view rotated by 180° (vertical axis), showing the "snow man"

The 3D model was created from 82 806 control points, obtained by the forward intersection method after the orientation parameters of the camera have been adjusted. The point cloud is triangulated and rendered with a simulated light source, in order to make the topographic features visible.




Southern hemisphere

Figure 5.3: Reconstruction of Vesta from 82 806 control points - the perspectives above are projections into Y-Z plane, the far side view is 180° rotated around the Z-axes (w.r.t leading side). The perspective below shows the X-Y plane viewed from above the south pole. See also the reference DTM from Preusker (Figure 5.4)

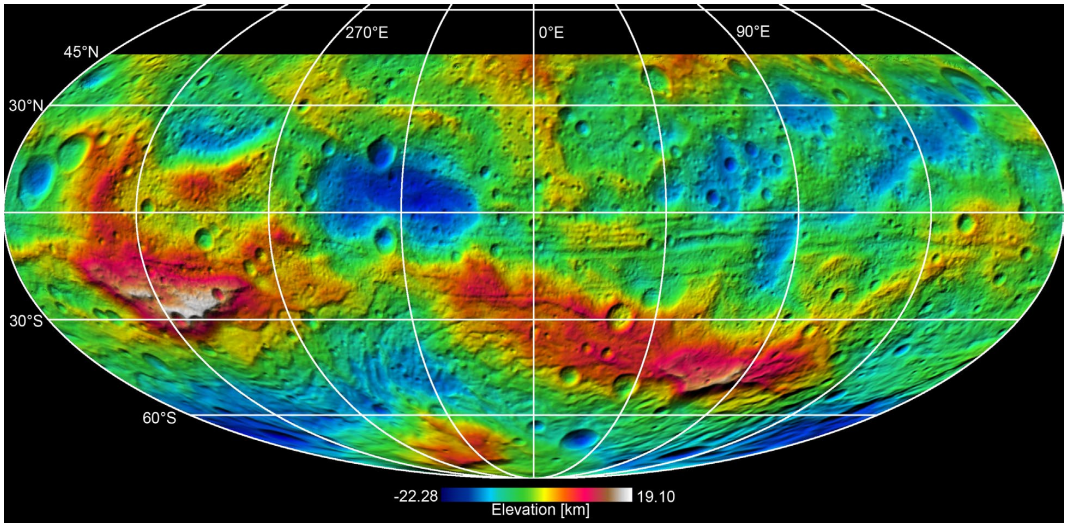



Figure 5.4: Reference DTM of Vesta (Preusker et al., 2012, fig. 2)

dard planes, such that the great circle containing the prime falls onto the Z-axis (the upper two images) or on the X-axis (the southern hemisphere). The point cloud was triangulated and rendered with the software GeoMagic, the shading and view settings were done with CloudCompare Viewer. A reference DTM based on a body-fixed solution (figure 5.4), published in Preusker et al. (2012), has been included.

### 5.2.3 Surface spherical harmonics analysis

The surface spherical harmonics of order $j = 0, 1, \ldots$ in $\mathbb{R}^3$ are given by

$$Y_j(\theta, \phi) = \sum_{m=0}^{j} P_{jm}(\sin\theta) \left(C_{jm} \cos(m\phi) + \bar{C}_{jm} \sin(m\phi)\right) \ , \ C_{jm}, \bar{C}_{jm} \in \mathbb{R}$$

when $\theta$ is the spherical latitude and $\phi$ the longitude. $P_{jm}$ is called *Legendre polynomial* for $m = 0$ and *associated Legendre polynomial* for $m > 0$ with degree $j$ (see definition Bronstein and Semendjajew (2005) 3.3.1.3.4). Since the coefficients $C_m, \bar{C}_m$ are arbitrary real numbers, $Y_j$ is as every polynomial uniquely defined by the choice of its coefficients.

**Remark:** *The typical definition in mathematics states* $\cos\theta$ *as argument of* $P_{jm}$*. It is important to note, that in this case* $\theta$ *is not the latitude. Instead, one measures the angle from the north pole* (0°) *down to the equator* (90°) *and*

*further to the south pole* (180°). *Hence,* $0 \leq \theta \leq 180$ *is the spherical co-latitude w.r.t. the north pole. Since* $\cos\theta$ *agrees with the sine value of the latitude on both hemispheres, the definition above is obtained.*

The analysis of the topography $f$ based on spherical harmonics relies on the serial development

$$f(\theta, \phi) = r = \sum_{j=0}^{\infty} Y_j(\theta, \phi)\,, \quad \forall \text{ surface points } P = (\theta, \phi, r) \tag{5.1}$$

That means, all control points are projected to the unit sphere $(\theta, \phi)$ and $f$ simply assigns $r$, the distance to the barycentre, to each point. This assignment must be unique, so obviously it is to assume

$$||P - P'|| > 0 \quad \Longleftrightarrow \quad ||(\theta, \phi) - (\theta', \phi')|| > 0\,.$$

With this necessary restriction to the topology $f$ is defined on the sphere and can be represented by equation (5.1) (Bronstein and Semendjajew, 2005, 3.3.2). The equation implies, that $f$ can be described by a set of coefficients that define the $Y_j$. Since this set is infinite, an approximation of $f$ is to use instead. The truncation

$$f_d = \sum_{j=0}^{d} Y_j \quad \text{satisfies} \quad \lim_{d\to\infty}(f - f_d) = 0 \tag{5.2}$$

where $d \in \mathbb{N}$ is called the degree of development.
The coefficients of the surface spherical harmonics have been computed based on equation (5.2) with $d = 60$ for all 82 806 control points. The system of equations can be written in matrix form as $Ac = b$ with the vector of radii on the right hand side and the unknown coefficients $c$. Weighted with the stochastic model aposteori $P_{CP}$ for the control points, the solution is obtained by inversion as

$$c = (A^T P_{CP} A)^{-1} A^T P_{CP}\, b\,.$$

In the appendix section A.4 the procedure is explained for $j = 0$ and $j = 1$, in total $61^2 = 3721$ coefficients have been computed. Table 5.5 lists the unnormalised coefficients up to order $j = 10$, that are the first 121 coefficients of the series. Figure 5.5 shows a larger spectrum (400 values), normalised to

the unit sphere radius by applying $C_{jm}/C_{00}$. The vertical lines mark the $C_{j0}$ coefficients, they are followed by $C_{j1}, \bar{C}_{j1}, C_{j1}, \bar{C}_{j1}, \ldots, C_{jj}, \bar{C}_{jj}$; the plot shows the magnitudes in both directions of the zero-line. A figure showing all of the coefficients has been dismissed because of its low weighting information.

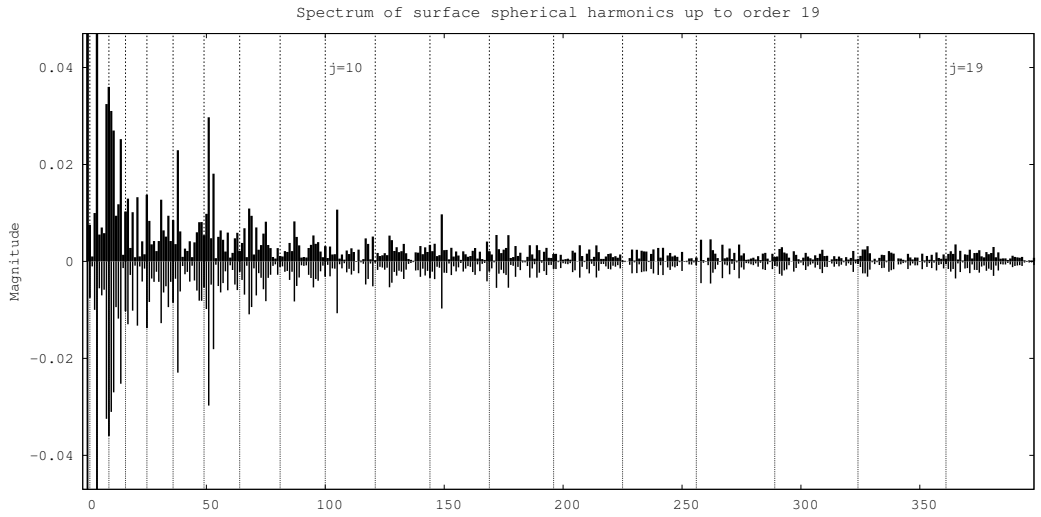



Figure 5.5: Zoom into spectrum of the surface spherical harmonics coefficients for $d = 60$ and $j = 0, \ldots, 19$ showing the first 400 values. The vertical lines mark the legendre coefficients $C_{j0}$, followed by $C_{jm}, \bar{C}_{jm}$ $(1 \leq m \leq j)$ successively. Values are normalised such that $C_{00} = 1$, $|C_{20}| = 0.23$ (the second large impulse) is the only other value above 0.04; magnitude is mirrored below zero-line for better visibility. See table 5.5 for concrete numbers.

From the coefficients of the spherical harmonics, the shape of Vesta was reconstructed. For $d = 60$ the result in kilometres is 559.57 x 560.02 x 456.69. Compared with the originally obtained CPN there is a difference in $X$ of 30 m, in $Y$ of 1.3 km and in $Z$ of 10 m, table 5.6 shows the rounded values (full km). The IAU gives the diameter of Vesta with 578 x 560 x 458 with an uncertainty of $\pm$10 km each (Archinal et al. (2011), table 6). Hence, there is a significant deviation for the semi-major axis (18 km). The coordinates of the ellipsoid's centre are (1.9, -4.6, 4.6). The dense distribution of measurements in the southern pole region mirrors the obtained offset between the center of mass and the center of figure. The images of the current data set do not sufficiently cover the northern pole region and there is a strongly unbalanced distribution of measurements, favouring the southern hemisphere. The section about the centre of figure (A.4, page 111) contains the well-known connection between the first order coefficients and the centre of figure; it is included to emphasise, that the distribution of control points projected onto the sphere should be uniform and esp. centred. This can be obtained if e.g. a DTM is used to complete the CPN.

Finally, the CPN has been reconstructed from the coefficients. Figure 5.6 shows the same point cloud as before, approximated by a series of surface spherical harmonics of degree and order 60. The views are the same as for the original CPN. The average approximation error is 378 m (min. 5 m, max. 4900 m).

**Surface spherical harmonics coefficients for Vesta**

| $j$ | $C_{j0}$ | $j$ | $C_{j0}$ | $j$ | $C_{j0}$ | $j$ | $C_{j0}$ | for $m = 0$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 260357.3 | 3 | 9368.7 | 6 | 2217.0 | 9 | -293.0 | |
| 1 | 1953.5 | 4 | 2673.9 | 7 | -1399.1 | 10 | 813.1 | |
| 2 | -60315.6 | 5 | -3574.2 | 8 | 555.0 | | | |

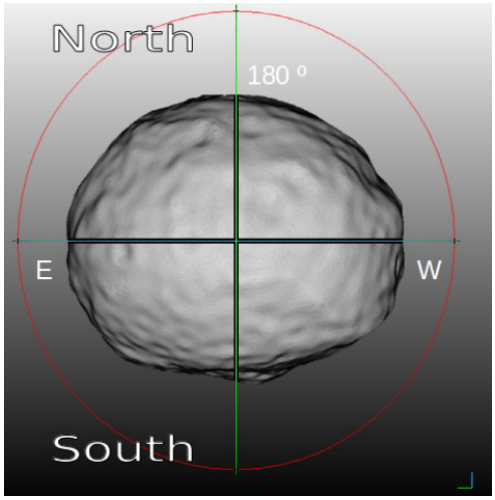
| $j$ | $m$ | $C_{jm}$ | $\bar{C}_{jm}$ | $j$ | $m$ | $C_{jm}$ | $\bar{C}_{jm}$ | for $m > 0$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | -266.2 | 2601.7 | 8 | 1 | 992.9 | -1782.3 | |
| 2 | 1 | 1441.5 | 1820.1 | 8 | 2 | 229.7 | -2840.4 | |
| 2 | 2 | 1517.7 | 8450.9 | 8 | 3 | -2448.8 | -374.9 | |
| 3 | 1 | -8080.6 | -7028.8 | 8 | 4 | 1827.7 | -631.6 | |
| 3 | 2 | 2453.5 | -3070.3 | 8 | 5 | -878.6 | 1494.5 | |
| 3 | 3 | -6568.8 | -356.7 | 8 | 6 | 2132.7 | -881.2 | |
| 4 | 1 | 3372.9 | 725.8 | 8 | 7 | -715.4 | -239.2 | |
| 4 | 2 | -2641.9 | 228.6 | 8 | 8 | 155.1 | -715.2 | |
| 4 | 3 | 3448.0 | -271.9 | 9 | 1 | -265.1 | -554.0 | |
| 4 | 4 | 1081.6 | 386.1 | 9 | 2 | 513.0 | -992.7 | |
| 5 | 1 | 2176.9 | 919.3 | 9 | 3 | 553.3 | 2150.1 | |
| 5 | 2 | 1095.8 | 553.6 | 9 | 4 | 1318.1 | -861.1 | |
| 5 | 3 | 1095.3 | 3315.0 | 9 | 5 | 207.5 | -240.4 | |
| 5 | 4 | 1668.0 | 1333.8 | 9 | 6 | 210.9 | 719.4 | |
| 5 | 5 | -2454.7 | 1095.9 | 9 | 7 | 888.9 | -1396.5 | |
| 6 | 1 | 929.1 | -5969.6 | 9 | 8 | 947.4 | -1038.0 | |
| 6 | 2 | -1612.9 | -247.8 | 9 | 9 | 531.1 | -225.7 | |
| 6 | 3 | -689.1 | -562.7 | 10 | 1 | 182.1 | 800.0 | |
| 6 | 4 | 1084.3 | -232.0 | 10 | 2 | -374.9 | 389.4 | |
| 6 | 5 | -1023.2 | 1564.4 | 10 | 3 | 2783.1 | -150.1 | |
| 6 | 6 | -2105.4 | -2112.5 | 10 | 4 | -372.9 | 95.6 | |
| 7 | 1 | -2555.7 | 7738.4 | 10 | 5 | 586.7 | -398.0 | |
| 7 | 2 | 1246.9 | 4713.2 | 10 | 6 | -706.5 | 508.7 | |
| 7 | 3 | 175.8 | -1322.0 | 10 | 7 | -22.6 | -635.5 | |
| 7 | 4 | -1664.8 | -1164.4 | 10 | 8 | 48.3 | 123.1 | |
| 7 | 5 | 532.4 | 1551.1 | 10 | 9 | -1237.4 | -943.5 | |
| 7 | 6 | -209.0 | 461.5 | 10 | 10 | 112.2 | -1337.6 | |
| 7 | 7 | -1241.1 | -1538.0 | | | | | |

Table 5.5: Unnormalised coefficients of surface spherical harmonics up to order 10 for degree of development $d = 60$, obtained by inversion with the stochastic model for adjusted control points

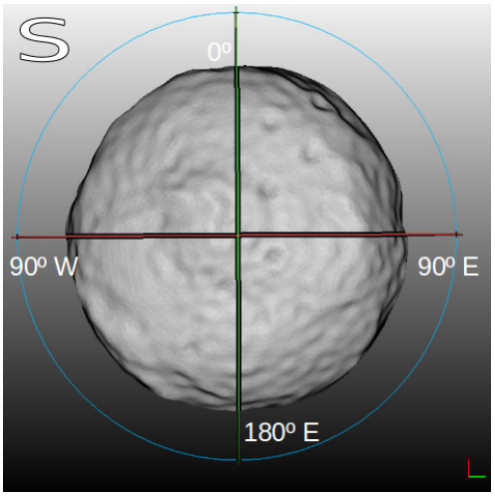
**Size parameter of Vesta**

| Source | Diameter | COE |
|---|---|---|
| CPN (best-fit) | 560 x 559 x 457 | (2, -5.5, 3.5) |
| SSH ($d = 60$) | 560 x 560 x 457 | (2, -5.0, 4.5) |
| IAU | 578 x 560 x 458 | — |

Table 5.6: Size parameter of Vesta (in km); col. 2 and 3 show the diameters and centre of the ellipsoid (COE) in X,Y,Z coordinates for a best-fit ellipsoid using the CPN and computed by surface spherical harmonics of degree and order 60 (SSH). IAU values (±10 km for each dimension) have been included from Archinal et al. (2011) for comparison.




View centred at 180° (vertical) and equator (horizontal)




Southern hemisphere

Figure 5.6: The 3D model was created with 82 806 control points, obtained by a series of surface spherical harmonics of degree $d = 60$. Latitude and longitude are specified by the points of figure 5.3. The point cloud is triangulated and rendered with a simulated light source, in order to make the topographic features visible.

# Chapter 6

# Conclusions and Outlook

## 6.1 Summary and conclusions

The bundle block adjustment in the inertial reference frame is a method to obtain rotational elements in a direct and analytical way from image point measurements. It is an extension of the classical method in the body-fixed reference frame that resolves the interdependence of the rotational model and the camera's external orientation (position and pointing). The functionality of the principle was tested in a case study with simulated data. The new approach turned out to improve the inner constrains of the control point network. It furthermore allows to analyse the dependencies of rotational elements and to identify parameters which are either weakly determined or not yet modelled. Applied to the real data set of Phobos, the forced libration amplitude $\lambda_0 = 1.13° \pm 0.013$ was computed with a CPN of 680 points. The result is in agreement with previous results and hence, supports the assumption of a uniform density of Phobos (Willner et al., 2010). The mean uncertainty of the CPN is about 55 m and the forward intersection error is about 13 m. The deviation in the pole axis orientation from the joint adjustment (table 5.2) indicates, that an improvement of Phobos' rotational model is possible and should be subject to a subsequent analysis. The method can be used for evaluation of further rotational elements such as the precession period or the acceleration term.

For Vesta, the pole axis orientation was computed together with a large CPN of $\approx 83\,000$ control points. The resulting values ($309.0246° \pm 0.008$, $42.2296° \pm 0.008$) are only slightly smaller then the currently adopted values of (309.031°,

42.235°) and well within the expected range of 0.01°. Also a comparison to a body-fixed adjustment was made showing that the new method is consistent with the previous results. On one hand, this confirms the current pole axis orientation of the IAU model (Archinal et al., 2013) and on the other hand, it shows the consistency of the inertial frame adjustment method. A comparing analysis of the body-fixed solution (Preusker; private data contribution) and the inertial frame solution with respect to the camera orientation data yielded another criteria of reliability: the actual differences of the corrected camera parameters are smaller than the offsets caused by the different pole axis orientation. Furthermore, the coordinates of the control points only differ by 15 m in average with a given forward intersection error of 10 m for both CPN solutions.

It has been shown, that the implemented software can handle large data sets; the costs for the resources running time are $\mathcal{O}(n^3)$ and for memory $\mathcal{O}(n^2)$ with $\frac{1}{6}n$ being the number of images. Hence, the capacity limit depends on the number of images and not on the number of control points. This is a significant improvement compared with the reference adjustment software: the computation time of 168 hours (seven days) for the adjustment of the Vesta data set is reduced to about eight hours in a first optimisation step, where the major work is done by a single core of the CPU (sequential computation). In a second optimisation step a decomposition algorithm was developed to benefit from multi-core architecture (parallel computation): the running time could be further reduced by factor 2 to four hours only on the same machine. Performance tests of the parallel algorithm showed that the choice of hardware makes a significant difference with respect to the speed-up factor – on other testing machines, a of factor 3 (maximum) and a factor of $\backsim$ 1 (minimum) were obtained. The comparison showed, that four cores are not sufficient to benefit from this new algorithm. On the other hand it turned out, that the number of parallel cores is not the most relevant criteria, too. Instead, the chip performance (in Gflop/s) and the number of cores together need to be considered. A 3-CPU node of the Xeon E5-2670 (24 cores) was showing the best performance.

Hence, even for large data sets, rotational elements can be computed in acceptable time. By means of a weight control, the rotational elements can be

kept fixed and a body-fixed bundle block adjustment is performed as a consequence. It is also possible to directly choose a body-fixed operation mode, but the additional resource requirements for the rotational elements are negligible. The new adjustment software is able to obtain the CPN solution for Vesta, the improved camera orientations and the pole axis orientation including the stochastic model aposteori in less than four hours on the reference machine. This corresponds to a speed-up factor of 42 with respect to the reference software.

## 6.2 Outlook

### Further steps

It is planned to perform a bundle block adjustment for 18 000 high resolution images mapping the equatorial region of Mercury. The focus hereby will be on the CPN solution, but simultaneously pole axis orientation and spin rate can be determined. Further geodetic products of Mercury will result from subsequent analysis.

Comet CP67, the target of the European Rosetta mission, is a very interesting target with respect to its rotational model. The available images can be used to study the rotational behaviour in greater detail with the inertial frame analysis.

The software should be enlarged to a general rotational model. All possible rotational parameters of an IAU model (Archinal et al., 2011) can be formally included. The initial rotational model as well as the parameters of interest could then be specified in a configuration file together with the weight information. Not only the pole axis coordinates $\alpha_0, \delta_0$, also the rate of precession, the rotation rate as well as the acceleration term for objects like Phobos can be resolved.

The paper of Konopliv et al. (2014) indicates that Vesta has a precessing pole, but this is a result of a theoretical consideration and the moment of precession could not be determined. It is planned to model the precession movement and to determine its period and momentum with the inertial frame adjustment method.

## GPU implementation and further optimisation

The OpenCL implementation of the decomposition algorithm in chapter 4 is fully applicable for GPU calculation. Since graphic engines can launch thousands of parallel threads, the multiplication on this computation device can be further optimised. On a CPU device, the number of threads is less than the number of images and this case, there should be only one thread assigned to each work group ($localworkgroupsize = 1$). On a GPU the size of the local work group can be increased and the work can be distributed to several threads instead. In this case further care has to be taken to synchronise the work-items of each group. With the current configuration, the algorithm was tested on a NVIDIA Quadro 600 with one GB memory. Here, only data sets of 1000 images could be tested and they are too small to pay off in terms of running time. For the Vesta data set, a card with 6 GB would be sufficient. However, the memory limitation of GPUs needs to be considered.

The algorithm computes the decomposition $(S, D)$ with $S^T DS = N_{11}^{-1}$, the upper left region of the inverse normal matrix. Considering the Vesta data set, at the current level of optimisation this part takes now a quarter of total running time on HLRN Konrad Berlin. The solution for the unknown vector $x$ is obtained quickly with the product routines for compressed sparse matrices. After that part, the diagonal of HAT matrix $AN^{-1}A^T$ is computed sequentially. This can be done in parallel and should be implemented in the future. The building of the sparse system itself, in particular the preparation step for the decomposition algorithm can be done in parallel as well. On the HLRN, with 24 cores this would reduce the running time from currently 34 minutes to about 10 minutes (per iteration) and increase the current speed-up factor of 3 to 9.

# Bibliography

C.H. Acton. Ancillary data services of NASA's Navigation and Ancillary Information Facility. *Planetary and Space Science*, 44. No. 1:66–70, 1996.

G. De Angelis. Three asteroid pole determinations. *Planetary and Space Science*, 41. No. 4:285–290, 1993.

B.A. Archinal et al. Report of the IAU working group on cartographic coordinates and rotational elements: 2009. *Celestial Mechanics and Dynamical Astronomy*, 109:101–135, 2011.

B.A. Archinal et al. Coordinate System for (4) Vesta. `http://astropedia.astrogeology.usgs.gov/download/Docs/WGCCRE/IAU-WGCCRE-Coordinate-System-for-Vesta.pdf`, Nov 2013. Pdf. Accessed: 14–10–2016.

I.N. Bronstein and K.A. Semendjajew. *Taschenbuch der Mathematik*. Harri Deutsch, 6. edition, 2005.

S. Burmeister. Phobos CPN 679. `http://dx.doi.org/10.14279/depositonce-5633`, Nov 2016a. Research data. Accessed: 16–12–2016.

S. Burmeister. Vesta CPN 82806. `http://dx.doi.org/10.14279/depositonce-5634`, Nov 2016b. Research data. Accessed: 16–12–2016.

H.J. Curnow and B.A. Wichmann. A synthetic benchmark. *Computer Journal*, 19(No 1):43–49, 1976.

W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. Springer-Verlag, Berlin Heidelberg, 2. edition, 2008.

D. Doody. *Basics of Space Flight*. Bluroof Press, Pasadena, 2011.

T.C. Duxbury and J.D. Callahan. Phobos and Deimos astrometric observations from Viking. *Astronomy and Astrophysics*, 201:169–176, 1988.

T.C. Duxbury and J.D. Callahan. Phobos and Deimos control networks. *Icarus*, 77(2):275–286, 1989.

G. Fischer. *Lineare Algebra und Analytische Geometrie*. Springer Spektrum, Wiesbaden, 2. edition, 2012.

T. Gehrels. Minor planets. I. The rotation of Vesta. *Astronomical Journal*, 72:929–938, 1967.

Y. Harada. Long-term polar motion on a quasi-fluid planetary body with an elastic lithosphere: Semi-analytic solutions of the time-dependent equation. *Icarus*, 220(2):449 – 465, 2012.

C.-H. Hsu and U. Kremer. A quantitative analysis of tile size selection algorithms. *The Journal of Supercomputing*, 27:279–294, 2004.

R.A. Jacobson. The orbits and masses of the martian satellites and the libration of Phobos. *The Astronomical Journal*, 139:668–679, 2010.

G.H. Kaplan. The IAU resolutions on astronomical reference systems, time scales, and earth rotation models: Explanation and implementation. *United States Naval Observatory Circular*, No. 179, 2005.

A.S. Konopliv et al. The Vesta gravity field, spin pole and rotation period, landmark positions, and ephemeris from the dawn tracking and optical data. *Icarus*, 240:103–117, 2014.

J.-Y. Li and J.N. Mafi. Body-Fixed Coordinate Systems for Asteroid (4) Vesta. `http://sbn.psi.edu/archive/dawn/fc/DWNVFC2_1A/DOCUMENT/VESTA_COORDINATES/VESTA_COORDINATES_131018.PDF`, Oct 2013. Pdf. Accessed: 17–10–2016.

J.-Y. Li et al. Improved measurement of asteroid (4) vesta's rotational axis orientation. *Icarus*, 211(1):528–534, 2010.

C. Ma et al. The International Celestial Reference Frame as realized by Very Long Baseline Interferometry. *The Astronomical Journal*, 116:516–546, 1998.

C. Ma et al. The second realization of the international celestial reference frame by very long baseline interferometry. Technical Note No. 35 35, International Earth Rotation and Reference Systems Service (IERS), 2009.

P. Magnusson. Distribution of spin axes and senses of rotation for 20 large asteroids. *Icarus*, 68:1–39, 1986.

I. Matsuyama et al. Rotational stability of dynamic planets with elastic lithospheres. *Journal of Geophysical Research: Planets*, 111(E2), 2006. E02003.

A. Munshi and B.R. Gaster et al. *OpenCL Programming Guide*. Pearson Education, Boston, 1. edition, 2012.

M.S. Murray and S.F. Dermott. *Solar System Dynamics*. Cambridge University Press, Cambridge, 1999.

W. Niemeier. *Ausgleichungsrechnung: Statistische Auswertemethoden*. De Gruyter, Berlin, 2. überarb. und erw. edition, 2008.

J. Oberst et al. The imaging performance of the SRC on Mars Express. *Planetary and Space Science*, 56:473–491, 2008.

J. Oberst et al. The Phobos geodetic control point network and rotation model. *Planetary and Space Science*, 102:45–50, 2014.

A. Pasewaldt et al. Astrometric observations of Phobos with the SRC on Mars Express. New data and comparison of different measurement techniques. *Astronomy and Astrophysics*, 580:A28, 2015. doi: 10.1051/0004-6361/201525957.

F. Preusker et al. Topography of Vesta from Dawn FC stereo images. *43rd Lunar and Planetary Science Conference*, 2012. id.2012.

N. Rambaux et al. Rotational motion of Phobos. *Astronomy and Astrophysics*, 548:45–50, 2012. doi: 10.1051/0004-6361/201219710.

M.S. Robinson et al. A revised control network for Mercury. *JGR*, 104 (E12): 847–852, 1999.

R. Tajeddine et al. Mimas: Strong forced longitudinal librations and constraints to its internal structure using Cassini ISS observations. *EPSC Abstracts*, Vo. 8, 2013. id.EPSC2013-391.

P.C. Thomas et al. Hyperion: rotation, shape, and geology from voyager images. *Icarus*, 117:128–148, 1995.

P.C. Thomas et al. Vesta: Spin pole, size, and shape from HST images. *Icarus*, 128:88–94, 1997.

M.S. Tiscareno, P.C. Thomas, and J.A. Burns. The rotation of Janus and Epimetheus. *Icarus*, 204:254–261, Mar 2009.

K. Willner et al. Phobos control point network, rotation and shape. *Earth and Planetary Science Letters*, 294:541–546, 2010.

# Appendix A

# Appendix

## A.1 Proof of the split technique

For a general matrix $M \in \mathcal{M}_{dim}$ with $|det(M)| > 0$ and subdivision scheme $M = \begin{bmatrix} A\,B \\ C\,D \end{bmatrix}$ the following relation holds (Bronstein and Semendjajew, 2005):

$$M^{-1} = \begin{bmatrix} A' & -A'BD^{-1} \\ -D^{-1}CA' & D^{-1} + D^{-1}CA'BD^{-1} \end{bmatrix} \tag{A.1}$$

where $A' = (A - BD^{-1}C)^{-1}$, $A \in \mathcal{M}_{dim_1}$ and $D \in \mathcal{M}_{dim_2}$, where $dim_1$ is arbitrary and $dim_2 = dim - dim_1$. Proof:
Using the same subdivision scheme for $M^{-1} = \begin{bmatrix} A'\,B' \\ C'\,D' \end{bmatrix}$, the multiplication $MM^{-1} = I$ leads to four systems of equations

$$\begin{aligned}
&1) \quad AA' + BC' &&= I_{dim_1} \\
&2) \quad AB' + BD' &&= 0 \\
&3) \quad CA' + DC' &&= 0 \qquad \Longleftrightarrow C' = -D^{-1}CA' \\
&4) \quad CB' + DD' &&= I_{dim_2} \qquad \Longleftrightarrow D' = D^{-1} - D^{-1}CB'
\end{aligned}$$

where the right hand side is the subdivision of the unit matrix $I_{dim}$.
Replacing $C'$ in equation 1) gives

$$\begin{aligned}
1') \quad AA' - BD^{-1}CA' = I \quad &\Longleftrightarrow (A - BD^{-1}C)A' = \quad I \\
&\Longleftrightarrow (A - BD^{-1}C)^{-1} = \quad A'
\end{aligned}$$

Then $D'$ is replaced in the second equation to obtain

$$\begin{aligned} 2') \quad AB' + BD^{-1} - BD^{-1}CB' &= 0 \\ (A - BD^{-1}C)B' + BD^{-1} &= 0 \iff (A')^{-1}B' + BD^{-1} = 0 \\ B' &= -A'BD^{-1} \,. \end{aligned}$$

If $B'$ is set into equation 4)

$$D' = D^{-1} - D^{-1}CB' = D^{-1} + D^{-1}CA'BD^{-1}$$

equation 4.3 follows. □

## A.2 Backward operation in split mode

The transformation of $S_2$ in the backward operation on page 77 is accomplished by sequentially replacing $S_1$ with its definition. Note $S_0 = L_0^{-1}$, $B_2 = \begin{pmatrix} b_{20} \\ b_{21} \end{pmatrix}$.

$$\begin{aligned} S_1 &= \begin{bmatrix} S_0, & 0 \\ 0, & I \end{bmatrix} \begin{bmatrix} I, & B_1 \\ 0, & L_1^{-1} \end{bmatrix} = \begin{bmatrix} S_0, & S_0B_1 \\ 0, & L_1^{-1} \end{bmatrix} \\ S_2 &= \begin{bmatrix} S_1, & 0 \\ 0, & I \end{bmatrix} \begin{bmatrix} I, & B_2 \\ 0, & L_2^{-1} \end{bmatrix} = \begin{bmatrix} S_1, & S_1B_2 \\ 0, & L_2^{-1} \end{bmatrix} \\ &= \begin{pmatrix} S_1, & \begin{bmatrix} S_0, & 0 \\ 0, & I \end{bmatrix} \begin{bmatrix} I, & B_1 \\ 0, & L_1^{-1} \end{bmatrix} B_2 \\ 0, & L_2^{-1} \end{pmatrix} \\ &= \begin{pmatrix} \begin{bmatrix} S_0, & S_0B_1 \\ 0, & L_1^{-1} \end{bmatrix}, & \begin{bmatrix} S_0, & 0 \\ 0, & I \end{bmatrix} \begin{bmatrix} b_{20} + B_1\, b_{21} \\ L_1^{-1}\, b_{21} \end{bmatrix} \\ 0, & L_2^{-1} \end{pmatrix} \\ &= \begin{bmatrix} L_0^{-1}, & L_0^{-1}B_1, & L_0^{-1}(b_{20} + B_1\, b_{21}) \\ & L_1^{-1}, & L_1^{-1}b_{21} \\ 0 & & L_2^{-1} \end{bmatrix} \end{aligned}$$

The computation of $B_1 = b_{10}$ is the synchronisation point for the first backward operation step (transforming the final column-group $CG = 2$):

$1a)$ $b_{20} \mathrel{+}= b_{10} * b_{21}$

$1b)\ b_{21} *= L_1^{-1}$ .

The final forward step is then the computation of $L_0^{-1}$ (and $D_0$); the final backward operation step is on the first row-group ($RG = 0$):

$2a)\ b_{10} *= L_0^{-1}$

$2b)\ b_{20} *= L_0^{-1}$

2a) and 2b) can be done in parallel straight away. 1b) has do be performed after 1a): Since both are realised in the same work-group this is easy if there is only one work-item per group. If the number will be increased, care has to be taken here. A local work-group barrier or a separate call for all work-groups can be used, analogue to step 2.
The steps above are for $n = 2$, for general $n \in \mathbb{N}$ they are as follows:

$1a)\ b_{ji} += b_{ki} * b_{jk}\ , i = 0, \ldots, k-1 \qquad \forall\, k < j < n$

$1b)\ b_{jk} *= L_k^{-1} \qquad \forall\, k < j < n$

and the final computation step is

$2)\ b_{j0} *= L_0^{-1} \qquad j = 1, \ldots, n-1\,.$

There is a work-group for each $j$ in every step; $k = n-2$ in the first iteration decreases to $k = 1$.

## A.3 Phobos tables

The tables here are additional information to section 5.1.

**initial SRC weights**

| Image No | $\sigma_X$ | $\sigma_Y$ | $\sigma_Z$ | $\sigma_\phi$ | $\sigma_\omega$ | $\sigma_\kappa$ |
|---|---|---|---|---|---|---|
| 44470005 | 1 | 1 | 1 | 0.01 | 0.01 | 0.5 |
| 43810004 | 1 | 1 | 1 | 0.01 | 0.01 | 0.5 |
| 28050024 | 1 | 1 | 1 | 0.01 | 0.01 | 0.5 |
| 46030005 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 45540005 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 43400005 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 28130005 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 38430003 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 38430004 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 28460007 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 22330005 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 38680004 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 38680005 | 1 | 1 | 1 | 0.05 | 0.05 | 0.05 |
| 7150004 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 |
| 27390005 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 |
| 46360005 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 |
| 27560005 | 1 | 1 | 1 | 0.5 | 0.5 | 0.5 |
| 44140004 | 50 | 50 | 50 | 0.5 | 0.5 | 0.5 |
| 43810005 | 50 | 50 | 50 | 0.5 | 0.5 | 0.5 |
| 43730004 | 50 | 50 | 50 | 0.5 | 0.5 | 0.5 |
| 43730005 | 50 | 50 | 50 | 0.5 | 0.5 | 0.5 |
| 27800004 | 50 | 50 | 50 | 0.5 | 0.5 | 0.5 |
| 4130002 | 100 | 100 | 100 | 0.05 | 0.05 | 0.05 |
| 4130004 | 1000 | 1000 | 1000 | 0.5 | 0.5 | 0.5 |
| 26730006 | 1000 | 1000 | 1000 | 0.5 | 0.5 | 0.5 |
| others | 1 | 1 | 1 | 0.01 | 0.01 | 0.01 |

Table A.1: SRC weighting model in metres for positions ($\sigma_X$, $\sigma_Y$, $\sigma_Z$ ) and gon for pointing data ($\sigma_\phi$, $\sigma_\omega$, $\sigma_\kappa$).

**pre-adjusted Viking weights**

| Image No | $\sigma_X$ | $\sigma_Y$ | $\sigma_Z$ | $\sigma_\phi$ | $\sigma_\omega$ | $\sigma_\kappa$ |
|---|---|---|---|---|---|---|
| 30609008 | 466.65 | 1215.66 | 944.42 | 0.325 | 0.123 | 0.296 |
| 30682813 | 447.66 | 885.56 | 905.40 | 0.199 | 0.095 | 0.156 |
| 30756611 | 448.02 | 692.62 | 746.18 | 0.144 | 0.081 | 0.104 |
| 30756617 | 460.72 | 766.88 | 850.31 | 0.154 | 0.080 | 0.111 |
| 30793514 | 1313.21 | 1199.22 | 1477.15 | 0.225 | 0.196 | 0.156 |
| 30793516 | 733.70 | 836.98 | 868.49 | 0.137 | 0.113 | 0.094 |
| 31958652 | 551.44 | 1096.07 | 1223.99 | 0.265 | 0.113 | 0.240 |
| 30627448 | 552.17 | 1376.97 | 1269.21 | 0.476 | 0.172 | 0.443 |
| 30793518 | 609.99 | 746.15 | 789.33 | 0.125 | 0.092 | 0.087 |
| 30756618 | 863.02 | 919.09 | 1136.50 | 0.195 | 0.123 | 0.173 |
| 30738070 | 609.31 | 1093.13 | 1346.96 | 0.521 | 0.189 | 0.456 |
| 30719636 | 476.17 | 807.77 | 1041.87 | 0.379 | 0.212 | 0.286 |
| 30719635 | 585.06 | 1121.50 | 1332.78 | 0.464 | 0.277 | 0.343 |
| 30682738 | 566.19 | 761.58 | 830.95 | 0.372 | 0.225 | 0.295 |
| 30682739 | 619.37 | 846.42 | 1276.26 | 0.445 | 0.362 | 0.281 |
| 30719637 | 579.59 | 1131.27 | 1319.47 | 0.442 | 0.299 | 0.316 |
| 30719639 | 781.21 | 1198.14 | 1431.54 | 0.463 | 0.344 | 0.347 |
| 33379477 | 18055.54 | 12512.32 | 15113.71 | 0.654 | 0.623 | 0.195 |
| 47290097 | 2067.44 | 2289.28 | 1817.14 | 0.215 | 0.174 | 0.107 |

Table A.2: Weighting model for Viking images resulted from the pre-adjustment in metres for positions ($\sigma_X$, $\sigma_Y$, $\sigma_Z$ ) and gon for pointing data ($\sigma_\phi$, $\sigma_\omega$, $\sigma_\kappa$). In the initial model position weights are set to 24 km for the final two images and to 5 km for the rest; pointing weights were set to 0.4 gon.

**pre-adjusted Viking orientation**

| Image No | $X$ | $Y$ | $Z$ | $\phi$ | $\omega$ | $\kappa$ |
|---|---|---|---|---|---|---|
| 30609008 | -509701.44 | 233275.16 | -278300.01 | 267.91 | -24.74 | 203.47 |
| 30682813 | -412993.96 | 329067.86 | -280656.79 | 261.69 | -36.25 | 198.47 |
| 30756611 | -318507.28 | 358711.4 | -280030.15 | 254.78 | -43.92 | 190.33 |
| 30756617 | -363553.65 | 376609.38 | -315200.95 | 255.45 | -42.20 | 191.47 |
| 30793514 | -270506.44 | 416395.79 | -277643.96 | 248.27 | -51.31 | 183.18 |
| 30793516 | -285298.13 | 420170.54 | -289207.82 | 249.33 | -50.74 | 184.67 |
| 31958652 | 445092.62 | 149187.44 | -531627.23 | 155.33 | -13.14 | -55.83 |
| 30627448 | -426187.25 | 201295.38 | -232174.43 | 267.58 | -25.45 | 203.29 |
| 30793518 | -298170.19 | 426946.68 | -301266.05 | 250.13 | -50.47 | 186.35 |
| 30756618 | -370371.87 | 382187.65 | -319847.74 | 253.68 | -42.43 | 193.25 |
| 30738070 | 333021.49 | 121682.06 | 243868.15 | 59.90 | -17.68 | 205.10 |
| 30719636 | 198906.29 | 148837.57 | 156295.07 | 57.30 | -32.43 | 205.71 |
| 30719635 | 203879.1 | 144407.99 | 163983.05 | 58.34 | -30.99 | 205.17 |
| 30682738 | 142520.48 | 126899.79 | 146962.95 | 48.03 | -33.86 | 208.77 |
| 30682739 | 132253.28 | 127256.51 | 141588.25 | 48.91 | -34.95 | 208.09 |
| 30719637 | 189124.81 | 147705.46 | 152857.53 | 58.15 | -33.10 | 205.12 |
| 30719639 | 172517.79 | 154358.52 | 140094.59 | 58.00 | -36.17 | 204.50 |
| 33379477 | 393514.46 | 1168304.25 | 1395701.66 | 17.54 | -42.69 | 135.39 |
| 47290097 | -497585.57 | 432978.41 | 585023.8 | -44.26 | -33.50 | 186.32 |

Table A.3: Pre-adjusted Viking positions ($X$, $Y$, $Z$) in metres and pointing angles ($\phi$, $\omega$, $\kappa$) in gon.

## A.4 Centre of figure

The origin of the body-fixed coordinates is the centre of mass (COM), the physical barycentre. It agrees with the centroid or centre of figure (COF), if the body has a uniform density. Hence, a difference of COM and COF would indicate a non-uniform mass distribution. It is now shown, how the first order coefficients $(C_{10}, C_{11}, \bar{C}_{11})$ of the surface spherical harmonics are related to the center of figure and further, what assumptions about the measurements need to be made in order to perform an analysis that allows reliable conclusions.
Consider a cloud of $n$ control points and let in the following $\theta_i$ the latitude, $\phi_i$ the longitude and $r_i$ the distance to the barycenter for point $i$. The centroid is given by

$$M = \left( \frac{1}{n} \sum_{i=1}^{n} X_i, \frac{1}{n} \sum_{i=1}^{n} Y_i, \frac{1}{n} \sum_{i=1}^{n} Z_i \right) =: (M_x, M_y, M_z) \,.$$

Further let

$$M_S = \left( \frac{1}{n} \sum_{i=1}^{n} \frac{X_i}{r_i}, \frac{1}{n} \sum_{i=1}^{n} \frac{Y_i}{r_i}, \frac{1}{n} \sum_{i=1}^{n} \frac{Z_i}{r_i} \right) =: (m_x, m_y, m_z) \,,$$

the centroid of the projected points which lays inside the unit sphere. It is important to see, that $M_S$ is only related to the distribution of control points on the surface and it does not depend on the topography. For example, the Vesta data set yields

$$M = \begin{pmatrix} -1704.82 \\ 3185.18 \\ -19604.1 \end{pmatrix}, \qquad M_S = \begin{pmatrix} -0.00618 \\ 0.0132 \\ -0.0831 \end{pmatrix}.$$

From both points a strong shift towards the southern pole can be noticed. $M_S$, in polar coordinates $(-80°, 115°)$, reveals a high concentration of measurements around the south pole and that $M$ is not representative for the real planetary object. Indeed the northern pole is not sufficiently covered.
Since the integral over the unit sphere vanishes, $M_S$ is always zero, if the whole topography $f$ is considered. Hence, the first reasonable assumption to make is $M_S = 0$, which is equivalent to a balanced distribution of control points. This is achieved if to each $(\theta, \phi)$ the diametric point $(-\theta, \phi + 180°)$ is included in the point cloud, but one relies on an interpolation method or a DTM to obtain

the height value for the diametric points.
Now consider the approximation (5.2) for degree $d = 1$. The point cloud then is represented by a system of $n$ linear equations which contain the spherical harmonics of order $j = 0$ and $j = 1$.
$Y_0$ is a constant function and equal to one. Hence for $j = 0$ the $n$ equations are simply given by $C_{00} = f_1(\theta_i, \phi_i) = r_i$. For $j = 1$ it is

$$C_{00} + C_{10} \sin\theta_i + C_{11} \cos\theta_i \cos\phi_i + \bar{C}_{11} \cos\theta_i \sin\phi_i = r_i$$

for each point $i$. Performing an adjustment just for the case $d = 0$ gives the mean radius

$$C_{00} = \frac{1}{n} \sum_{i=1}^{n} r_i \ , \tag{A.2}$$

but for $d = 1$ the normal equation is

$$N(C_{00}, C_{10}, C_{11}, \bar{C}_{11})^T = \left( \sum_{i=1}^{n} r_i \sum_{i=1}^{n} Z_i, \sum_{i=1}^{n} X_i, \sum_{i=1}^{n} Y_i \right)^T \tag{A.3}$$

with the symmetric matrix

$$N = \begin{pmatrix} n & \sum z_i & \sum x_i & \sum y_i \\ * & \sum z_i^2 & \sum z_i x_i & \sum z_i y_i \\ * & * & \sum x_i^2 & \sum x_i y_i \\ * & * & * & \sum y_i^2 \end{pmatrix}, \qquad \begin{aligned} x_i &= \cos\theta_i \cos\phi_i \\ y_i &= \cos\theta_i \sin\phi_i \\ z_i &= \sin\theta_i \qquad . \end{aligned}$$

The $*$ act as place holders for a better readability.
Scaling equation (A.3) with $\frac{1}{n}$ leads to

$$\frac{1}{n} N \left( C_{00},\, C_{10},\, C_{11},\, \bar{C}_{11} \right)^T = \left( \frac{1}{n} \sum_{i=1}^{n} r_i \,,\, M_z,\, M_x,\, M_y \right)^T \tag{A.4}$$

Since the scaling factor is applied to both sides, it does not effect the values of the coefficients. Note, that the solution vector on the righthand side contains the coordinates of $M$ and the first row of $N$ becomes $(1, m_x, m_z, m_y)$ which holds the coordinates of $M_S$. With the demand $M_S = 0$ the first row (resp. column) changes to the unit vector and equation (A.4) can be written as

$$C_{00} = \frac{1}{n} \sum_{i=1}^{n} r_i \quad \text{which is the same as (A.2)}$$

and the remaining

$$\begin{pmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i z_i \\ * & \sum y_i^2 & \sum y_i z_i \\ * & * & \sum z_i^2 \end{pmatrix} \begin{pmatrix} C_{11} \\ \bar{C}_{11} \\ C_{10} \end{pmatrix} = \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} . \tag{A.5}$$

The matrix was permuted to align the order of coefficients with the principle axes, so that the equation above can be shortened to $QC = M$. If $M = 0$ than also $C = 0$ because of the linear independence. The matrix $Q$ is the covariance matrix of the unit sphere coordinates. It is again independent of the topography and relies only on the choice of measurements. Hence, the relationship between the centroid $M$ and the first order coefficients is given by

$$C = Q^{-1}M \, .$$

Moving from a discrete point cloud to the continuous case (the whole sphere) shows, that $Q$ is constant for all topographies. Following the argumentation, that $M_S$ is always zero for the continuous case, the side entries of $Q$ also vanish and it is

$$\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} C = M \quad \Longrightarrow \quad C^T = \left( \frac{M_x}{\lambda_1}, \frac{M_y}{\lambda_2}, \frac{M_z}{\lambda_3} \right) .$$

One can show $\lambda_1 = \lambda_2 < \lambda_3$, hence $C$ is not just the scaled COF.
This is the analytical connection between the first order coefficients and the centroid. The $\lambda_i$ will vary for discrete point clouds with the choice of control points, also for the same object if points are removed or included. They do not change if only the height values are altered (scaling).
In shape studies it is a common praxis to force the coefficients to be zero since one wishes to consider a centred figure. Note, that for a balanced measurement distribution, i.e. $M_S = 0$ the coefficients will be zero, if the points are shifted to COF (see equation A.5). For $||M_S|| > 0$ this is not true, because the matrices $Q$ and $N$ are not diagonal and the values of $M_S$ affect the solution for $C$:

$$QC = M - k_C \, M_S \, , \quad k_C = \frac{1}{n} \sum r_i - C^T M_s \, .$$

Therefore, after $C$ is computed, the transformation $C^T P_i$ is used to shift the points iteratively (within a new adjustment) until $C = 0$ is reached.

**Remark:** *If a stochastic model is used for the adjustment, the solution vector changes to a weighted mean vector. Further note, that any normalisation* $\mathcal{N}(j,m)$ *of the Legendre polynomials* $P_{jm}$ *changes the values of the coefficients. This change can be reversed, if the norm is known.*

# Appendix B

# Code examples for Compressed Sparse Matrices

The following pages contain in addition to section 4.1 the C++ class definition of a compressed sparse matrix (csm) and selected product routines of the class.

```
/*  SparseMatrix.h
 *  Created on: Nov 21, 2012    Author: burm_st
 */
#ifndef SPARSEMATRIX_H_
#define SPARSEMATRIX_H_
#include <math.h>

class SparseMatrix {
public:
        SparseMatrix(SparseMatrix *SIn);
        SparseMatrix(bool known_size, int nnz, int rows, int cols, bool ccf);
        SparseMatrix(int size, int n, int m);
        SparseMatrix(double *m, int n, int col, bool ccf);            // if #nnz is unknown
        SparseMatrix(double *m, int n, int col, int size, bool ccf);  // if #nnz is known
        SparseMatrix(double *tLTM, int dim, bool utm);  // sparse the transpose of this LTM
        virtual ~SparseMatrix();

        double *VALUES;
        int *COLUMNS;
        int *FOARS;
        int  size, dim_n, dim_m;
        bool ccf;
        // for building sparse matrices on the fly
        bool fixed_size;
        int mem_size;
        int mem_rows;
        int buffer_default_size;

        void add(double val, int col, int row);
        void add_denseLine(double *line, int row, int dim);
        void addLine(double *val, int *col, int row, int length);
        void add2Lines(double *val, int *col, int row1, int length1Line);
        void fix_dimensions();
        void finish();
        void check_memory(int to_add, int r);

        void transpose();

        double* product(SparseMatrix *SM);              // with another (crf) Sparse-Matrix
        double* product(SparseMatrix *SM, bool ccf);    // dito, generalized format
        double* product(double *v, int length_v);               // with a vector
        double* product(double *m, int rows, int columns);      // with a matrix
        double* doATPA(double *diagOfP, int n);
        double* doATA();
        void productDiagXthis(double *diag, int n);     // diag is diagonal vector only
        SparseMatrix* product_toccf(SparseMatrix *SM);

        void clean();   // keep memory structure, reset values to default
        void detach();  // set memory pointers to NULL
        double* resparse();                                     // reconstruct matrix
};
#endif /* SPARSEMATRIX_H_ */
```

```
/*****************************************
 * PRODUCT METHODS  assuming that THIS is crf
 *****************************************/

double* SparseMatrix::product(SparseMatrix *SM) {
        double m_ij;
        int j_end, k_end, i, j, tj, k;
        if(dim_m != SM->dim_n) {
                cout << "product impossible: columns don't match rows of second matrix.
exit product()." << endl;
                return NULL;
        }
        double *P = new double[dim_n*SM->dim_m]();
//newMatrix(dim_n,SM.dim_m);

        switch(ccf) {
        case 0:
                for (i = 0; i < dim_n; ++i) {
                        for (j = FOARS[i], j_end = FOARS[i+1]; j < j_end; ++j) {
                                m_ij = VALUES[j];
                                for (k = SM->FOARS[tj=COLUMNS[j]], k_end =
SM->FOARS[tj+1]; k < k_end; ++k) {
                                        P[i*SM->dim_m+SM->COLUMNS[k]] +=
m_ij*SM->VALUES[k];
                                }
                        }
                }
                break;

        case 1:
                for (i = 0; i < dim_m; ++i) {
                        for (j = FOARS[i], j_end = FOARS[i+1]; j < j_end; ++j) {
                                m_ij = VALUES[j];
                                for (k = SM->FOARS[i], k_end = SM->FOARS[i+1]; k < k_end;
++k) {
                                        P[COLUMNS[j]*SM->dim_m+SM->COLUMNS[k]] +=
m_ij*SM->VALUES[k];
                                }
                        }
                }
                break;
        }
        return P;
}

/**
 *  intended to multiply a diagonal matrix whose dimension matches dim_n with the SM from
 *  left i.e. useful to calculate PA, when this equals the spardes A-matrix and diag the
 *  diagonal vector of P
 */
void SparseMatrix::productDiagXthis(double *diag, int n) {
```

```
        int j_end, k_end, i, j, tj, k;
        if(dim_n != n) {
                cout << "product impossible: dimension doesn't match rows of second
matrix. exit product()." << endl;
                return;
        }
        for (i = 0; i < dim_n; ++i) {
                for (j = FOARS[i], j_end = FOARS[i+1]; j < j_end; ++j) {
                        VALUES[j] *= diag[i];
                }
        }
}

/**
 * note: this routine currently only supports a crf Matrix
 *
 * @param:  double **ATPb - the pointer of the result (this)^transposed*P*b
 *                                      the variable *ATPb will be allocated inside this
routine!!!
 *          double *b     - a vector of length n
 *          double *P     - a vector of length n which is treated like a diagonal matrix
 *
 * @return  double *ATPA  - a matrix of dim_m x dim_m (this)^transposed*P*(this)
 */
double* SparseMatrix::doATPA(double *P, int n) {
        if (n != dim_n) {
                cout << "please provide vector of length dim_n = " << dim_n  << endl;
                cout << "doATPA() currently only supports crf Matrix A. exit here..." <<
endl;
                return NULL;
        }

        int i,j,k,j_end,k_end,col_j;

        double *AtPA = new double[dim_m*dim_m](), m_ij;
        for (i=0; i < dim_n; i++) {
                for (j=FOARS[i], j_end = FOARS[i+1]; j < j_end; j++) {
                        m_ij = VALUES[j]*P[i], col_j = COLUMNS[j];
                        for (k=FOARS[i], k_end = FOARS[i+1]; k < k_end; k++)
                                AtPA[col_j*dim_m + COLUMNS[k]] += m_ij*VALUES[k];
                }
        }
        return AtPA;
}

// beware: works only for ccf = false correctly
double* SparseMatrix::doATA() {

        double *AtA = new double[dim_m*dim_m](), val_j;
        int i,j,k, col_j;
        for (i=0; i < dim_n; i++) {
                for (j=FOARS[i]; j < FOARS[i+1]; j++)
```

```
                        for (val_j = VALUES[j], col_j = COLUMNS[j], k=FOARS[i]; k <
FOARS[i+1]; k++)
                                AtA[col_j*dim_m + COLUMNS[k]] += val_j*VALUES[k];
        }
        return AtA;
}

/*****************************************
 * PRODUCT METHODS where THIS is crf or ccf
 *****************************************/

/**
 *  Matrix product of THIS x *SM,
 *      where SM is treated as crf (case 0) or ccf (case 1)
 *
 *  Note: if ccf is false the standard procedure product(SM) is called
 *                if ccf is true THIS should be ccf too, otherwise
 *        SM is resparsed and sparsed again to crf
 */
double* SparseMatrix::product(SparseMatrix *SM, bool ccf) {

        if(dim_m != SM->dim_n) {
                cout << "product impossible: columns don't match rows of second matrix.
exit product()." << endl;
                return NULL;
        }
        double m_ij;
        int j_end, k_end, i, j, tj, k;
        double *P = NULL;                       //newMatrix(dim_n,SM.dim_m);

        switch(ccf) {

        case 0:
                return product(SM);

        case 1:
                if (this->ccf) {                // works !!
                        P = new double[dim_n*SM->dim_m]();
                        for (i = 0; i < SM->dim_m; ++i) {
                                for (j = SM->FOARS[i], j_end = SM->FOARS[i+1]; j < j_end;
++j) {
                                        tj = SM->COLUMNS[j];
                                        m_ij = SM->VALUES[j];
                                        // get column tj of this
                                        for (k = FOARS[tj], k_end = FOARS[tj+1]; k <
k_end; ++k)
                                                P[COLUMNS[k]*SM->dim_m+i] +=
m_ij*VALUES[k];
                                }
                        }
                        break;
                }
```

```
                else {  // umständlich aber korrekt....
                        double *tmp = SM->resparse();
                        k = SM->size, i = SM->dim_n, j = SM->dim_m;
                        SM->clean();
                        SM->construct(tmp,i,j,k,false);
                        P = product(SM);
                        SM->clean();
                        SM->construct(tmp,i,j,k,true);
                        delete tmp;
                        break;
                }
        }
        return P;
}

// multiply THIS with vector v (THIS x v)
double* SparseMatrix::product(double *v, int length_v) {
        if(dim_m != length_v) {
                cout << "product impossible: columns don't match vector length. exit
product()." << endl;
                return NULL;
        }
        double *thisXv = new double[dim_n]();
        int i, j, j_end;

        switch(ccf) {

        case 0: // crf
                for (i = 0; i < dim_n; ++i) {
                        for (j = FOARS[i], j_end = FOARS[i+1]; j < j_end; ++j) {
                                thisXv[i] += VALUES[j]*v[COLUMNS[j]];
                        }
                }
                break;
        case 1:
                for (i = 0; i < dim_m; ++i) {
                        for (j = FOARS[i], j_end = FOARS[i+1]; j < j_end; ++j) {
                                thisXv[COLUMNS[j]] += VALUES[j]*v[i];
                        }
                }
        }
        return thisXv;
}

// multiply THIS with matrix m of [rows, columns]
double* SparseMatrix::product(double *m, int rows, int columns) {
        if(dim_m != rows) {
                cout << "product impossible: columns don't match rows of right matrix.
exit product()." << endl;
        }
        double m_ij;
        int j_end, i, j, tj, k;
```

```
        double *P = new double[dim_n*columns]();

        switch(ccf) {

        case 0:
                for (i = 0; i < dim_n; ++i) {
                        for (j = FOARS[i], j_end = FOARS[i+1]; j < j_end; ++j) {
                                m_ij = VALUES[j];
                                for (k = 0; k < columns; ++k) {
                                        P[i*columns+k] += m_ij*m[COLUMNS[j]*columns+k];
                                }
                        }
                }
                break;
        case 1:
                for (i = 0; i < dim_m; ++i) {
                        for (j = FOARS[i], j_end = FOARS[i+1]; j < j_end; ++j) {
                                m_ij = VALUES[j];
                                tj = COLUMNS[j];
                                for (k = 0; k < columns; ++k) {
                                        P[tj*columns+k] += m_ij*m[i*columns+k];
                                }
                        }
                }
        }
        return P;
}

/*****************************************
 * PRODUCT METHODS with SPARSE RESULT
 *****************************************/

// ccf x ccf -> ccf
SparseMatrix* SparseMatrix::product_toccf(SparseMatrix *SM) {
        SparseMatrix *SP = new SparseMatrix(false,size+SM->size,dim_n,SM->dim_m,true);
        int dim = dim_n, j, k, j2, j2_end;
        buffer_default_size = dim;
        double *P = new double[dim](), val_j;
        for (int i=0; i<SM->dim_m; i++) {
                for (j= SM->FOARS[i]; j < SM->FOARS[i+1]; j++) {
                        k = SM->COLUMNS[j];
                        val_j = SM->VALUES[j];
                        for (j2 = FOARS[k], j2_end = FOARS[k+1]; j2 < j2_end; j2++)
                                P[COLUMNS[j2]] += VALUES[j2]*val_j;
                }
                SP->check_memory(dim,0);
                SP->add_denseLine(P,i,dim);
        }
        SP->finish();
        return SP;
}
```