A universal and fast method to solve linear systems with correlated coefficients using weighted total least squares

OPEN ACCESS
IOP Publishing
Measurement Science and Technology
Meas. Sci. Technol. **33** (2022) 015017 (10pp)
https://doi.org/10.1088/1361-6501/ac32ec

A universal and fast method to solve linear systems with correlated coefficients using weighted total least squares

Matthias Wurm

Physikalisch-Technische Bundesanstalt, Bundesallee 100, 38116 Braunschweig, Germany

E-mail: matthias.wurm@ptb.de

Received 18 May 2021, revised 29 September 2021
Accepted for publication 25 October 2021
Published 25 November 2021

Abstract

Especially in metrology and geodesy, but also in many other disciplines, the solution of overdetermined linear systems of the form $\boldsymbol{Ax} \approx \boldsymbol{b}$ with individual uncertainties not only in $\boldsymbol{b}$ but also in $\boldsymbol{A}$ is an important task. The problem is known in literature as *weighted total least squares*. In the most general case, correlations between the elements of $[\boldsymbol{A},\boldsymbol{b}]$ exist as well. The problem becomes more complicated and can—except for special cases—only be solved numerically. While the formulation of this problem and even its solution is straightforward, its implementation—when the focus is on reliability and computational costs—is not. In this paper, a robust, fast, and universal method for computing the solution of such linear systems as well as their covariance matrix is presented. The results were confirmed by applying the method to several special cases for which an analytical or numerical solution is available. If individual coefficients can be considered to be free of errors, this can be taken into account in a simple way. An implementation of the code in MATLAB is provided.

Supplementary material for this article is available online

Keywords: weighted total least squares, generalized total least squares, mixed total least squares, total least squares, linear systems, error propagation, covariance matrix

1. Introduction

In our laboratory at the National Metrology Institute in Germany, we work in the field of Mueller matrix ellipsometry, an optical method for the investigation of structured surfaces and thin films. While evaluating the raw data (incl. error propagation), we encountered a mathematical problem whose technical solution we think could also be relevant for other experiments.

It is a classic problem: let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ with $m > n$ be a coefficient matrix with full column-rank. $\boldsymbol{b} \in \mathbb{R}^{m\times 1}$ and $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$ are column vectors. Typically, the rows of $\boldsymbol{A}$ and $\boldsymbol{b}$ result from m different measurements performed under different configurations or conditions. $\boldsymbol{x}$ is unknown; it will be determined from

$$\boldsymbol{Ax} \approx \boldsymbol{b}. \tag{1}$$

The solution of this linear system is the central topic of this paper. The solution, its uncertainty, and also the complexity of its calculation strongly depend on the coefficients that are subject to uncertainties, and how these uncertainties are correlated with each other. One searches, therefore, for the best-fit solution $\boldsymbol{x}$, considering the covariances of the matrix $[\boldsymbol{A},\boldsymbol{b}]$ and wants to know how these input covariances propagate to

$\boldsymbol{x}$. Let $\boldsymbol{W}$ be a weight matrix, defined as the inverse of a covariance matrix $\boldsymbol{\Sigma}$, which, in turn, describes all uncertainties and mutual correlations of the elements in $[\boldsymbol{A},\boldsymbol{b}]$

$$\boldsymbol{W}^{(m\cdot(n+1)\times m\cdot(n+1))} = \boldsymbol{\Sigma}^{-1}_{[\boldsymbol{A},\boldsymbol{b}]}. \tag{2}$$

With the vectorization of a matrix performed column-wise

$$\boldsymbol{a} = \mathrm{vec}\,(\boldsymbol{A}) = \begin{pmatrix} A_{1,1} \\ \vdots \\ A_{m,n} \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_{m\cdot n} \end{pmatrix}, \tag{3}$$

the problem can be formulated as an optimization problem:

$$\min_{\boldsymbol{x}} SE_{\boldsymbol{W}}(\boldsymbol{x}) \tag{4}$$

with

$$SE_{\boldsymbol{W}} = \left\| \begin{pmatrix} \boldsymbol{da} \\ \boldsymbol{db} \end{pmatrix} \right\|^2_{\boldsymbol{W}} = \begin{pmatrix} \boldsymbol{da} \\ \boldsymbol{db} \end{pmatrix}^{\mathrm{T}} \boldsymbol{W} \begin{pmatrix} \boldsymbol{da} \\ \boldsymbol{db} \end{pmatrix} \tag{5}$$

s.t.

$$(\boldsymbol{A}+\boldsymbol{dA})\boldsymbol{x} = \boldsymbol{b}+\boldsymbol{db} \tag{6}$$

($^{\mathrm{T}}$ denotes the matrix transpose operation; *SE*, squared error). Or, with the equality constraints eliminated:

$$SE_{\boldsymbol{W}}(\boldsymbol{x}) = \begin{pmatrix} \boldsymbol{da} \\ (\boldsymbol{A}+\boldsymbol{dA})\boldsymbol{x}-\boldsymbol{b} \end{pmatrix}^{\mathrm{T}} \boldsymbol{W} \begin{pmatrix} \boldsymbol{da} \\ (\boldsymbol{A}+\boldsymbol{dA})\boldsymbol{x}-\boldsymbol{b} \end{pmatrix}. \tag{7}$$

This is the most general formulation of the problem. Depending on the structure of $\boldsymbol{W}$, which can be partitioned as

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{W_A} & \boldsymbol{W_{Ab}^{\mathrm{T}}} \\ \boldsymbol{W_{Ab}} & \boldsymbol{W_b} \end{pmatrix} \tag{8}$$

many different solutions were found.

1.1. Cases with analytical solutions

Analytical solutions were found for

- LS (*least squares*): $\boldsymbol{W_b} \sim \boldsymbol{I}$, $\boldsymbol{W_A} = \boldsymbol{0}$, and $\boldsymbol{W_{Ab}} = \boldsymbol{0}$.
- WLS (*weighted least squares*): $\boldsymbol{W_b} \sim \boldsymbol{\Sigma_b^{-1}}$, $\boldsymbol{W_A} = \boldsymbol{0}$, and $\boldsymbol{W_{Ab}} = \boldsymbol{0}$.
- TLS (*total least squares*): $\boldsymbol{W} \sim \boldsymbol{I}$.
- MTLS or LS-TLS (*mixed total least squares*): $\boldsymbol{W_A} \sim \mathrm{diag}\,([0_i, 1_j])$, $\boldsymbol{W_b} \sim \boldsymbol{I}$, and $\boldsymbol{W_{Ab}} = \boldsymbol{0}$ with $i = 1,\ldots,m\cdot n_1$, $j = m\cdot(n_1+1),\ldots,m\cdot n$, and n_1 the subset of columns in $\boldsymbol{A}$, which are assumed to be free of errors.
- GTLS (*generalized total least squares*): $\boldsymbol{W} \sim \left(\boldsymbol{P_C^{((n+1)\times(n+1))}} \otimes \boldsymbol{P_R^{(m\times m)}}\right)^{-1}$.

Here, $\boldsymbol{I}$ is the identity matrix, $\otimes$ the Kronecker product, and $\mathrm{diag}\,(\boldsymbol{v})$ generates a diagonal matrix from a vector $\boldsymbol{v}$. An overview of the methods is given in [1]. A large collection of references can be found in [2].

In the appendix, we provide in brief the formulas for some analytical solutions.

1.2. Cases with numerical solutions only

For other cases, however, only numerical solutions are available. These were often developed in the field of geodesy.

Premoli and Rastello have dealt with the *element-wise weighted total least squares* [3]. This method allows independent weights for the elements of $[\boldsymbol{A},\boldsymbol{b}]$ but no correlations i.e. $\boldsymbol{W}$ is a diagonal matrix with independent diagonal elements. Later, this method was taken up by further authors and adapted for special applications [4–6].

For the case of no correlation between $\boldsymbol{A}$ and $\boldsymbol{b}$ (i.e. $\boldsymbol{W_{Ab}} = \boldsymbol{0}$), Schaffrin and Wieser [7] solved the problem for a certain structure of $\boldsymbol{W_A}$: it must be a Kronecker product of a $m\times m$ with a $n\times n$ matrix. Mahboub [8] succeeded in removing this restriction. Amiri-Simkooei and Jazaeri showed that this *weighted total least squares* (WTLS) problem is an extension of *WLS* problem [9].

With the above methods, it is possible to fit a straight line on 2D data. This is what Krystek and Anton did in [10]. Their clou is that they transferred the problem to polar coordinates and reduced it to only one free parameter, which makes their solution very robust. They later modified their method and allowed mutual correlations [11]. Here, $\boldsymbol{W_{Ab}}$ now comes into play and is no longer equal to 0. In addition, in both implementations, they also provided a numerical estimation for the covariance matrix of the solution.

Snow [12] and Zhou *et al* [13] provided a solution for the case of a full weight matrix $\boldsymbol{W}$. Later, Zhou and Fang [14] also provided a solution for the mixed case (WLS-WTLS), in which some columns of $\boldsymbol{A}$ are assumed to be free of errors.

Despite the large number of different methods, we must conclude that our problem[1] (and certainly other problems as well) cannot be solved by any of the above methods on account of individual limitations. We have the following requirements:

(1) $\boldsymbol{W}$ must be allowed to be a full matrix i.e. correlations between $\boldsymbol{A}$ and $\boldsymbol{b}$ must be allowed.
(2) In some of our m measurements, some non-error-free components are not involved. This means it must be possible to exclude individual elements from the WTLS optimization and not only complete columns, like in WLS-WTLS or MTLS.
(3) The method should be fast since many modern problems in metrological practice deal with large matrices, and often a data preprocessing is required during a measurement to control the experimental parameters for the next measurement step[1].
(4) To allow an error-propagation analysis, a covariance matrix for the best fitting solution $\boldsymbol{x}$ must be provided.

[1] A problem in optics: the determination of Mueller matrix elements and their covariance from uncertain Fourier coefficients achieved with imperfect optical components [15]. In this application, a 140×15 matrix problem must be solved in less than 3 s.

The last point especially is of the utmost importance in metrology. In our application, the solution of the problem is an intermediate step; in terms of traceability, we must be able to determine the influence of input uncertainties on the final result and its uncertainty in accordance with the 'Guide to the expression of uncertainty in measurement' [16].

2. Numerical solution for the general case

In the general case, the weight matrix $\boldsymbol{W}$ has no special structure. But to represent a metric space, it must, of course, be symmetric and positive definite. If it is only positive semi-definite, the problem can be reformulated to only consider the relevant dimensions.

The task is now to solve equation (4). The unknowns are $\boldsymbol{x}$ and $\boldsymbol{dA}$. For a given $\boldsymbol{x}$, the necessary condition for an optimal $\boldsymbol{dA}$ is

$$\frac{\partial}{\partial da_{i=1\ldots m\cdot n}} SE_{\boldsymbol{W}} = 0. \tag{9}$$

This is a linear system, with $m\cdot n$ equations for the $m\cdot n$ unknowns of $\boldsymbol{da}$. After exploiting and applying

$$\boldsymbol{A}^{(m\times n)}\boldsymbol{x}^{(n\times 1)} = \left(\boldsymbol{x}^{\mathrm{T}} \otimes \boldsymbol{I}^{(m\times m)}\right)\ \mathrm{vec}\,(\boldsymbol{A}) \tag{10}$$

it can be converted to

$$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{V}\boldsymbol{da} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{u} \tag{11}$$

with

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{I}^{(m\cdot n\times m\cdot n)} \\ \boldsymbol{x}^{\mathrm{T}} \otimes \boldsymbol{I}^{(m\times m)} \end{pmatrix} \tag{12}$$

and

$$\boldsymbol{u} = -\begin{pmatrix} \boldsymbol{0}^{(m\cdot n\times 1)} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \end{pmatrix}. \tag{13}$$

$\boldsymbol{da}$ in equation (11) can be solved for a given $\boldsymbol{x}$ in MATLAB with the so-called backslash operator '\' as

$$\boldsymbol{da} = \left(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{V}\right)\backslash\left(\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{u}\right). \tag{14}$$

With $\boldsymbol{da}$ reshaped to $\boldsymbol{dA}$, one gets $SE_{\boldsymbol{W}}$ from equation (7). $SE_{\boldsymbol{W}}$ then can be used as merit function with a solver of your choice to find the optimal vector $\boldsymbol{x}$. However, in this form, the optimization is very expensive in terms of computing time and memory requirements. We will address this issue in section 3.

2.1. Reduction to requested elements

If individual elements of $\boldsymbol{da}$ and/or $\boldsymbol{db}$ are to be excluded from optimization (i.e. it is assumed to be free of errors), this can be done very easily: one has to just eliminate the regarding rows and columns in the matrices and vectors in equation (11).

For this purpose, let $\boldsymbol{dA}\mathbf{i}$ be given as a user-defined Boolean array, **i**ndicating the matrix elements of $\boldsymbol{dA}$ to be optimized. $\boldsymbol{db}\mathbf{i}$ is accordingly defined. Furthermore, elements of $\boldsymbol{dA}\mathbf{i}$ and $\boldsymbol{db}\mathbf{i}$ shall be set to *false* by default, if the according diagonal element of the weight matrix $\boldsymbol{W}$ equals 0. Then, equation (11) has to be solved for the subset $\boldsymbol{da}_{\boldsymbol{da}\mathbf{i}}$ of $\boldsymbol{da}$ with

$$\begin{aligned} \boldsymbol{V}_{\mathrm{red}} &= \boldsymbol{V}_{[\boldsymbol{da}\mathbf{i};\boldsymbol{db}\mathbf{i}],\boldsymbol{da}\mathbf{i}} \\ \boldsymbol{W}_{\mathrm{red}} &= \boldsymbol{W}_{[\boldsymbol{da}\mathbf{i};\boldsymbol{db}\mathbf{i}],[\boldsymbol{da}\mathbf{i};\boldsymbol{db}\mathbf{i}]} \\ \boldsymbol{u}_{\mathrm{red}} &= -\begin{pmatrix} \boldsymbol{0}_{\boldsymbol{da}\mathbf{i},1} \\ [\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}]_{\boldsymbol{db}\mathbf{i}} \end{pmatrix} \end{aligned} \tag{15}$$

('red' = reduced; *logical indexing* is used here: [T, F, T,…]→[1, 3,…]). Of course, $\boldsymbol{db}$ has to be again calculated from equation (6) taking care that *false* elements of $\boldsymbol{dA}\mathbf{i}$ lead to zeros at the respective positions in $\boldsymbol{dA}$.

$SE_{\boldsymbol{W}}(\boldsymbol{x})$ can then be calculated from $\boldsymbol{da}_{\boldsymbol{da}\mathbf{i}}$, $\boldsymbol{db}_{\boldsymbol{db}\mathbf{i}}$, and $\boldsymbol{W}_{\mathrm{red}}$. Note that this technique is more general than MTLS or WLS-WTLS since it allows one to assume individual elements of $[\boldsymbol{A},\boldsymbol{b}]$ to be free of errors and not only full columns.

3. Performance and robustness

In the previous chapter, the mathematical solution of the problem was given. We will now cover some technical and programming aspects to make it fast and reliable. The method was implemented in MATLAB. Nevertheless, most of the following aspects can certainly be transferred to other programming languages.

3.1. Recommended optimizer

Since the merit function (equation (7)) gives back a scalar positive value—which is the sum of SEs in the metric space defined by $\boldsymbol{W}$[2]—any multivariate optimizer should be applicable.

We recommend MATLAB's optimizer `lsqnonlin`, which is part of the optimization toolbox. Its special feature: it expects the merit function to provide an error vector $\boldsymbol{F}$ instead of the sum of squared errors $\sum F_i^2$. Here, such a vector can easily be defined:

$$\boldsymbol{F} = \boldsymbol{R}_{\boldsymbol{W}}\begin{bmatrix} \boldsymbol{da} \\ \boldsymbol{db} \end{bmatrix} \tag{16}$$

with $\boldsymbol{R}_{\boldsymbol{W}}^{\mathrm{T}}\boldsymbol{R}_{\boldsymbol{W}} = \boldsymbol{W}$ is the Cholesky decomposition of $\boldsymbol{W}$. It is obvious that $\boldsymbol{F}$ contains more details on the problem than $SE_{\boldsymbol{W}} = \sum F_i^2$. This improves convergence and therefore performance. `lsqnonlin` uses the *trust-region-reflective* algorithm by default and alternatively the *Levenberg–Marquardt*. It is a gradient-based local solver. If your

[2] Though the subscript 'red' is skipped from now on, only the *reduced* versions of the regarding matrices and vectors are used. $\boldsymbol{W}$ (=$\boldsymbol{W}_{\mathrm{red}}$) is symmetric and positive definite.

problem is difficult (i.e. many local minima), you may need to use a global solver (like *differential evolution* or *particle swarm*). In that case, there is nothing left but to use $\sum F_i^2$ instead of $\boldsymbol{F}$.

3.2. Initial values for optimization

For good conditioned problems, it is easy and cheap to provide a good guess as the initial value for the optimization from the LS solution (equation (40)) or the WLS solution (equation (41)). If the problem is close to the TLS problem, one should, of course, start with its solution as the initial guess (equation(43)). However, GTLS is the analytical solution that provides the widest possible consideration of a general weight matrix $\boldsymbol{W}$ and should, therefore, be preferred in general.

To do so, first, the necessary matrices $\boldsymbol{P_C}$ and $\boldsymbol{P_R}$ have to be derived as the solution from the minimization problem:

$$\min \left\| \boldsymbol{W}^{-1} - \boldsymbol{P_C} \otimes \boldsymbol{P_R} \right\|_F^2 \tag{17}$$

($\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm). This is known as the *nearest Kronecker product problem* and has an analytical solution [17].

Let $\mathcal{R}\left(\boldsymbol{W}^{-1}\right)$ be a certain rearrangement[3] of $\boldsymbol{W}^{-1}$ depending on the dimensions of $\boldsymbol{P_C}$ and $\boldsymbol{P_R}$, i.e. $(n+1)\times(n+1)$ and $m \times m$. From its singular value decomposition

$$\boldsymbol{U}\mathrm{diag}\left(\boldsymbol{\sigma}\right)\boldsymbol{V}^{\mathrm{T}} = \mathcal{R}\left(\boldsymbol{W}^{-1}\right) \tag{18}$$

$\boldsymbol{P_C}$ and $\boldsymbol{P_R}$ can be found from $\mathrm{vec}\left(\boldsymbol{P_C}\right) = \sigma_1 \boldsymbol{U}_{:,1}$ and $\mathrm{vec}\left(\boldsymbol{P_R}\right) = \boldsymbol{V}_{:,1}$ respectively.

It is important to note that only the product, $\boldsymbol{P_C} \otimes \boldsymbol{P_R}$ can be uniquely determined. $\boldsymbol{P_C}$ and $\boldsymbol{P_R}$ are unique except for constant scalar factors s and $1/s$ respectively. Furthermore, since $\boldsymbol{W}$ is symmetric and positive definite, $\boldsymbol{P_C}$ and $\boldsymbol{P_R}$ are symmetric, and either both positive or both negative definite. In the latter case, they can be made positive definite with $s = -1$. The absolute value of s has no impact on the GTLS solution, which then gives the best possible analytical guess as initial values for the subsequent optimization.

3.3. Rewriting equation (11)

The essential part of the solution given in the previous chapter is equation (11). The terms of this formal expression can be rewritten in a much more memory- (and, hence, time-) efficient way as

$$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{V} = \boldsymbol{W_A} + \boldsymbol{x} \otimes \boldsymbol{W_{Ab}} + \left(\boldsymbol{x} \otimes \boldsymbol{W_{Ab}}\right)^{\mathrm{T}} + \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} \otimes \boldsymbol{W_b} \tag{19}$$

and

$$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{u} = -\left(\boldsymbol{W}_{\boldsymbol{Ab}}^{\mathrm{T}} + \boldsymbol{x} \otimes \boldsymbol{W_b}\right) \cdot \left(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right). \tag{20}$$

[3] The general MATLAB code for such a rearrangement is: `reshape(permute(reshape(A,m2,m1,n2,n1),[2,4,1,3]),m1*n1,m2*n2)`.

3.4. Robust solution of equation (11)

Since $\boldsymbol{W}$ is symmetric, and positive definite, the same is true for the coefficient matrix $\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{V}$. Because of that, equation (11) can be solved efficiently and robust. Let $\boldsymbol{R}$ with

$$\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R} = \boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{V} \tag{21}$$

be the Cholesky decomposition of the coefficient matrix. Because of $\boldsymbol{R}$'s triangular shape, equation (11) can be solved with a forward-, followed by a backward-substitution. Again, with MATLAB's backslash operator, one can write these two steps as:

$$\boldsymbol{da} = \boldsymbol{R}\backslash\left(\boldsymbol{R}^{\mathrm{T}}\backslash\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{u}\right). \tag{22}$$

This solution looks more complicated than the one given in equation (14). But, actually, it is not; there are two reasons to solve $\boldsymbol{da}$ with equation (22) instead of equation (14).

(a) Applying the backslash-operator causes MATLAB to check the coefficient matrix for certain properties. If it is of triangular shape, then the substitution technique is directly applied, and all other tests are skipped. On the other hand, checking whether the Cholesky solver is applicable to equation (14) (which we already know) costs three more tests on the input matrix, which can be saved here.
(b) We explicitly will need the matrix $\boldsymbol{R}$ in the further process.

3.5. Calculation of the Jacobian matrix

As stated earlier, MATLAB's `lsqnonlin` is gradient-based. This means that after each calculation of a candidate solution, the Jacobian matrix is calculated to estimate the candidate of the subsequent iteration. Since the accuracy requirements on the Jacobian are relaxed, it is sufficient to estimate it from *forward finite differences*, which can be obtained after n explicit calls of the vectorial merit function (equation (16)) i.e.:

$$\boldsymbol{J} = \left(\frac{\partial \boldsymbol{F}}{\partial x_i}\right)_{i=1,..,n} \approx \left(\frac{\boldsymbol{F}\left(\boldsymbol{x} + \Delta\boldsymbol{x_i}\right) - \boldsymbol{F}\left(\boldsymbol{x}\right)}{\Delta x_i}\right)_{i=1,..,n}. \tag{23}$$

The vector $\Delta\boldsymbol{x}_i$ has zero elements, barring the ith element, which equals Δx_i.

Of course, its calculation can be very time-consuming even after implementing the improvements suggested in the last two sections. The most costly one is the calculation of $\boldsymbol{R}$, the Cholesky decomposition. It requires $\sim\mathcal{O}\left((m\cdot n)^3\right)$ operations. Hence, with the forward finite-difference estimation of the Jacobian, n additional Cholesky decompositions would be necessary ($\sim\mathcal{O}\left((m\cdot n)^3\cdot n\right)$).

But this can be avoided. MATLAB's `lsqnonlin` can also be supported with an externally provided Jacobian. We will now calculate it with only $\sim\mathcal{O}\left((m\cdot n)^2\cdot n\right)$ operations.

The finite difference method will also be used now. At first, one may try to find a cheap solution to update $\boldsymbol{R}$ when an element of $\boldsymbol{x}$ updates slightly from $x_i \to x_i + \Delta x_i$. Only one method is known for this: the *rank-one update*. Unfortunately, the conditions for applicability are not fulfilled for this here because, with equation (19), it can be shown that an $\boldsymbol{x}$-update causes a rank-m update for $n = 1$ and a rank-$2m$ update for $n > 1$.

Here is the alternative solution:
first, let us define shortcuts for the expressions in equation (11) (or equations (19) and (20))

$$\begin{aligned} \boldsymbol{Y}(\boldsymbol{x}) &:= \boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{V} \\ \boldsymbol{z}(\boldsymbol{x}) &:= \boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{u}. \end{aligned} \tag{24}$$

Suppose we had updated $\boldsymbol{x} \to \boldsymbol{x} + \Delta\boldsymbol{x}_i$. Formally, $\boldsymbol{da}(\boldsymbol{x} + \Delta\boldsymbol{x}_i)$ can then be calculated as

$$\boldsymbol{da}(\boldsymbol{x} + \Delta\boldsymbol{x}_i) = \boldsymbol{Y}^{-1}(\boldsymbol{x} + \Delta\boldsymbol{x}_i)\,\boldsymbol{z}(\boldsymbol{x} + \Delta\boldsymbol{x}_i). \tag{25}$$

With

$$\boldsymbol{dY}(\Delta\boldsymbol{x}_i) := \boldsymbol{Y}(\boldsymbol{x} + \Delta\boldsymbol{x}_i) - \boldsymbol{Y}(\boldsymbol{x}) \tag{26}$$

$$\boldsymbol{dz}(\Delta\boldsymbol{x}_i) := \boldsymbol{z}(\boldsymbol{x} + \Delta\boldsymbol{x}_i) - \boldsymbol{z}(\boldsymbol{x}) \tag{27}$$

the following is surely true:

$$\boldsymbol{da}(\boldsymbol{x} + \Delta\boldsymbol{x}_i) = \left(\boldsymbol{Y}(\boldsymbol{x}) + \boldsymbol{dY}(\Delta\boldsymbol{x}_i)\right)^{-1}\left(\boldsymbol{z}(\boldsymbol{x}) + \boldsymbol{dz}(\Delta\boldsymbol{x}_i)\right). \tag{28}$$

On taking a look at equation (19), it is revealed that $\boldsymbol{dY}(\Delta\boldsymbol{x}_i)$ is—related to the spectral radius—surely small compared to $\boldsymbol{Y}(\boldsymbol{x})$ (keep in mind that $\boldsymbol{x} = \boldsymbol{0}$ is never a solution of your problem). This allows a first-order Taylor approximation for the matrix inverse:

$$\begin{aligned} \boldsymbol{da}(\boldsymbol{x} + \Delta\boldsymbol{x}_i) \approx &\left(\boldsymbol{Y}^{-1}(\boldsymbol{x}) - \boldsymbol{Y}^{-1}(\boldsymbol{x})\,\boldsymbol{dY}(\Delta\boldsymbol{x}_i)\,\boldsymbol{Y}^{-1}(\boldsymbol{x})\right) \\ &\times \left(\boldsymbol{z}(\boldsymbol{x}) + \boldsymbol{dz}(\Delta\boldsymbol{x}_i)\right). \end{aligned} \tag{29}$$

With the shortcut

$$\boldsymbol{c}_i := \boldsymbol{Y}^{-1}(\boldsymbol{x})\,\boldsymbol{dz}(\Delta\boldsymbol{x}_i) = \boldsymbol{R}\backslash(\boldsymbol{R}^{\mathrm{T}}\backslash\boldsymbol{dz}(\Delta\boldsymbol{x}_i)) \tag{30}$$

one gets

$$\begin{aligned} \Delta\boldsymbol{da}(\Delta\boldsymbol{x}_i) &:= \boldsymbol{da}(\boldsymbol{x} + \Delta\boldsymbol{x}_i) - \boldsymbol{da}(\boldsymbol{x}) \\ &\approx \boldsymbol{c}_i - \boldsymbol{Y}^{-1}(\boldsymbol{x})\,\boldsymbol{dY}(\Delta\boldsymbol{x}_i)\,(\boldsymbol{da}(\boldsymbol{x}) + \boldsymbol{c}_i). \end{aligned} \tag{31}$$

And finally

$$\Delta\boldsymbol{da}(\Delta\boldsymbol{x}_i) \approx \boldsymbol{c}_i - \boldsymbol{R}\backslash\left(\boldsymbol{R}^{\mathrm{T}}\backslash\left[\boldsymbol{dY}(\Delta\boldsymbol{x}_i)\,(\boldsymbol{da}(\boldsymbol{x}) + \boldsymbol{c}_i)\right]\right). \tag{32}$$

Note: it was not necessary to explicitly calculate the inverse of $\boldsymbol{Y}$. One can now reshape $\Delta\boldsymbol{da}(\Delta\boldsymbol{x}_i)$ to $\Delta\boldsymbol{dA}(\Delta\boldsymbol{x}_i)$ and calculate

$$\begin{aligned} \Delta\boldsymbol{db}(\Delta\boldsymbol{x}_i) = &\left[\boldsymbol{A} + \boldsymbol{dA} + \Delta\boldsymbol{dA}(\Delta\boldsymbol{x}_i)\right](\boldsymbol{x} + \Delta\boldsymbol{x}_i) \\ &- (\boldsymbol{b} + \boldsymbol{db}). \end{aligned} \tag{33}$$

The Jacobian can then be estimated as

$$\boldsymbol{J} \approx \left(\boldsymbol{R}_{\boldsymbol{W}}\begin{bmatrix} \Delta\boldsymbol{da}(\Delta\boldsymbol{x}_i) \\ \Delta\boldsymbol{db}(\Delta\boldsymbol{x}_i) \end{bmatrix}/\Delta x_i\right)_{i=1,..,n}. \tag{34}$$

Remark: for this *forward difference approximation* of the Jacobian, a step size of $\Delta x_i = \mathrm{sign}(x_i)\sqrt{\varepsilon}(1 + |x_i|)$ is used with the constant $\varepsilon = 2^{-52}$, which is the distance from 1.0 to the next larger double-precision number, and $\mathrm{sign}(x) = 1$ for $x \geqslant 0$ and $\mathrm{sign}(x) = -1$ for $x < 0$. This definition of Δx_i is based on the default definition for this approximation in MATLAB.

In sum, these five steps carried out in this chapter drastically accelerate the optimization. If one has a supported graphics processing unit (GPU) and MATLAB's parallel-computing toolbox, one can even increase the speed further, since many matrix operations used here can benefit from the parallel organization of a GPU. For this purpose, only the input matrices must be defined as `gpuArrays`. A parallel distribution to several cores of the CPU—on the other hand—cannot be sensibly implemented, as it has no positive effect on account of overhead.

We finish this chapter with some numbers on the performance: table 1 shows a small benchmark for two different machines with or without GPU support and with or without the Jacobian provided. In our application, the problem with $\boldsymbol{A}^{(140\times15)}$ is typically good conditioned, so a few ten iterations will suffice. A complete optimization, including the calculation of the covariance matrix (see next chapter) of the solution, is, therefore, finished after 2.5 s at the latest. Since we typically measure data which lead to several thousand such problems, it makes a big difference in terms of applicability and feasibility whether their solution takes a week or only a few hours.

4. Covariance matrix of the solution

As a metrologist, the writer is not only interested in the final result $\boldsymbol{x}$ but also in its statistical uncertainty. Therefore, the calculation of the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ was also implemented. One oft-used estimation of the covariance matrix calculated from the Jacobian at the optimum of a least-squares-merit function is

$$\boldsymbol{\Sigma}_{\boldsymbol{x}}^{(J)} := \left(\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}\right)^{-1}. \tag{35}$$

Or, if the weight matrix is known only up to an unknown factor:

$$\boldsymbol{\Sigma}_{\boldsymbol{x}}^{\prime(J)} := \frac{SE_W}{m-n}\left(\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}\right)^{-1}. \tag{36}$$

Table 1. Comparison of computation times for a problem with $\boldsymbol{A}^{(140\times15)}$ and a full weight matrix $\boldsymbol{W}$ on different machines. All tests have been performed with MATLAB R2020b. CUDA 11.2 was used for the tests with GPU support.

	Iteration time	
	Without	With
	Jacobian provided	
CPU1 only Intel i5 3470	2500 ms	210 ms
CPU1 + GPU1 Nvidia GeForce GTX 1050 Ti	1500 ms	103 ms
CPU2 only Intel Xeon E5-2667 v3	1500 ms	125 ms
CPU2 + GPU2 Nvidia Tesla V100	230 ms	22 ms

Since all values are available at the end of the optimization, it can be directly calculated. But while this estimation might be fine for low-dimensional problems it gets worse with increasing n.

A more reliable approximation of the covariance matrix is based on the Hessian matrix $\boldsymbol{\mathcal{H}}$:

$$\boldsymbol{\Sigma}_{\boldsymbol{x}}^{(\boldsymbol{\mathcal{H}})} := 2\boldsymbol{\mathcal{H}}^{-1}. \tag{37}$$

Or, again, if the weights are only known relative to each other:

$$\boldsymbol{\Sigma'}_{\boldsymbol{x}}^{(\boldsymbol{\mathcal{H}})} := \frac{SE_{\boldsymbol{W}}}{m-n} \cdot 2\boldsymbol{\mathcal{H}}^{-1}. \tag{38}$$

The technique applied to determine the Jacobian can also be used to calculate the Hessian. Its elements are now calculated with the *central finite differences* approximation. This is more costly than the *forward finite differences* approximation but more accurate:

$$\begin{aligned}\boldsymbol{\mathcal{H}} \approx \Big(\frac{1}{4\Delta x_i \Delta x_j}\big[&SE_{\boldsymbol{W}}(\boldsymbol{x}+\Delta\boldsymbol{x}_i+\Delta\boldsymbol{x}_j)\\ &- SE_{\boldsymbol{W}}(\boldsymbol{x}+\Delta\boldsymbol{x}_i-\Delta\boldsymbol{x}_j) - SE_{\boldsymbol{W}}(\boldsymbol{x}-\Delta\boldsymbol{x}_i+\Delta\boldsymbol{x}_j)\\ &+ SE_{\boldsymbol{W}}(\boldsymbol{x}-\Delta\boldsymbol{x}_i-\Delta\boldsymbol{x}_j)\big]\Big)_{i,j=1,\ldots,n}.\end{aligned} \tag{39}$$

Here, a step size of $\Delta x_i = \operatorname{sign}(x_i)\,\varepsilon^{1/3}\,(1+|x_i|)$ is used. Since the Hessian is symmetric, it is sufficient to calculate the $n\cdot(n+1)/2$ independent elements only. But as can been seen from equation (39), it requires four explicit calculations of $SE_{\boldsymbol{W}}$ for each element. Hence, the effort is already considerable. Therefore, the calculation of the covariance matrix was implemented as optional output.

The same is true for the condition number of $\boldsymbol{V}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{V}$. It might give you some helpful information. But, of course, it requires an additional singular value decomposition.

Before starting an optimization, one might also be interested in the condition number or the efficiency [18] of $\boldsymbol{A}$.

5. Applications

The method was implemented in MATLAB as a function, which we named `cwtls` (correlated weights total least squares). It outputs $\boldsymbol{x}$ as solution of equation (4) and—among others—its covariance $\boldsymbol{\Sigma'}_{\boldsymbol{x}}$, according to equation (38) (to be compatible with other MATLAB functions). From additional output parameters, the covariance can be calculated, according to equation (37).

Table 2 shows some application examples for `cwtls` and a comparison with—mostly analytical—alternatives. If the optimization has converged to the global minimum (which is the case for the most real-world problems, i.e. not artificially ill-conditioned, and with reasonable starting values), then, in each case, the `cwtls` solution agrees with the alternative solution—of course—up to the tolerance set as the stopping criterion of the optimization. Also, the covariance matrices provided by `cwtls` agreed with their alternative calculations. In addition, they were validated with those resulting from the numerically determined Hessian matrix at the optimum. For this purpose, the code from [19] was used.

Since, in the literature, the definitions of the covariance matrices are sometimes somewhat vague or imprecise, the corresponding formulas used here are also given.

Examples of numerical applications are given in the supplementary material (available online at stacks.iop.org/MST/33/015017/mmedia) to document agreement of `cwtls` results with the alternative solutions shown in table 2.

Table 2 can also be used as a guide, demonstrating how to prepare the inputs to solve a problem with `cwtls`.

A (perhaps trivial) remark for the case that an analytical solution exists: if one has absolute knowledge of the weight matrix (i.e. there is no need to rescale $\boldsymbol{\Sigma}_{\boldsymbol{x}}$ with *mse*), then one can calculate $\boldsymbol{\Sigma}_{\boldsymbol{x}}$—according to the formulas in the table—without (or before) knowing $\boldsymbol{x}$. This should be kept in mind when designing an experiment that focuses on low uncertainties for $\boldsymbol{x}$.

Please note: `cwtls` is not intended to replace existing and specialized solutions. Rather, it can be used when specialized solutions do not exist.

Table 2. Application examples for `cwtls`. Here, some special cases for which alternative—mostly analytical—solutions exist are shown. In each case, the solutions and covariance matrices provided agree with the one found by `[x,∼,∼,∼,∼,∼,∼,Cov] = cwtls(A,xini,b,W,dAi,dbi)` ($\boldsymbol{x}_{\mathrm{ini}}$, an initial guess; ***dbi***, always true). $mse = \frac{\mathrm{SE}_W}{m-n}$.

$\boldsymbol{W}$				***dA*i**	One of (possibly) several alternative solutions (MATLAB command)	$\boldsymbol{\Sigma}'_x$, covariance (either nominal or MATLAB expression to calculate it from outputs)	Reference
$\boldsymbol{W}_A$	$\boldsymbol{W}_b$	$\boldsymbol{A}$	$\boldsymbol{b}$	(True, false)			Footnote **n**um./**a**na.
Least squares							
0	$\boldsymbol{I}$	$\boldsymbol{A}$	$\boldsymbol{b}$	**F**	`x=A\b`	$mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}$	**a**
Fitting a **straight line through the origin**, only uniform errors in y							
0	$\boldsymbol{I}$	$\boldsymbol{x}_{\mathrm{in}}$	$\boldsymbol{y}_{\mathrm{in}}$	**F**	`x=A\b`	$mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}$	**a**
Fitting a **straight line through the origin**, only errors in y, uncorrelated weights							
0	$\mathrm{diag}\,(\boldsymbol{w})$	$\boldsymbol{x}_{\mathrm{in}}$	$\boldsymbol{y}_{\mathrm{in}}$	**F**	`[x,∼,∼,Cov]=lscov(A,b,w)`	$mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\mathrm{diag}\,(\boldsymbol{w})\boldsymbol{A}\right)^{-1}$	**a**
Fitting a **straight line through the origin**, only errors in y, correlated weights							
0	$\boldsymbol{V}^{-1}$	$\boldsymbol{x}_{\mathrm{in}}$	$\boldsymbol{y}_{\mathrm{in}}$	**F**	`[x,∼,∼,Cov]=lscov(A,b,V)`	$mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{A}\right)^{-1}$	**a**
Fitting a **straight line through fixed point**, errors in x and y, uncorrelated weights							[20]
$\mathrm{diag}\left(\begin{bmatrix} \boldsymbol{ux} \\ \boldsymbol{uy} \end{bmatrix}^{-2}\right)$		$\boldsymbol{x}_{\mathrm{in}} - \boldsymbol{x}_{\mathrm{fix}}$	$\boldsymbol{y}_{\mathrm{in}} - \boldsymbol{y}_{\mathrm{fix}}$	**T**	`[x,∼,∼,∼,SE,C]=wtls_line_origin (xin-xfix,yin-yfix,ux,uy)`	`SE/(m-n)*C`	**n**
Fitting a **straight line**, only uniform errors in y							
0	$\boldsymbol{I}$	$[\boldsymbol{x}_{\mathrm{in}}, 1]$	$\boldsymbol{y}_{\mathrm{in}}$	**[F,F]**	`[x,S]=polyfit(xin,yin,1)`	`(inv(S.R)*inv(S.R)')*S. normr.^2/S.df`	**a**
Fitting a **straight line**, only errors in y, correlated weights							
0	$\boldsymbol{V}^{-1}$	$[\boldsymbol{x}_{\mathrm{in}}, 1]$	$\boldsymbol{y}_{\mathrm{in}}$	**[F,F]**	`[x,∼,∼,Cov]=lscov(A,b,V)`	$mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{V}^{-1}\boldsymbol{A}\right)^{-1}$	**a**
Total least squares							
$\boldsymbol{I}$		$\boldsymbol{A}$	$\boldsymbol{b}$	**T**	`x=tls([A,b])`	$mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} - \sigma_{n+1}^2\boldsymbol{I}\right)^{-1}$	a, **a**
Mixed total least squares							[21]
$\boldsymbol{I}$		$\boldsymbol{A}$	$\boldsymbol{b}$	$[\mathbf{F}_{:,1:n1}, \mathbf{T}_{:,1:n-n1}]$	`x=mtls(A(:,1:n1), A(:,n1+1:end),b)`	See [22]	**a**
Fitting a **straight line**, uniform errors in x and y							[21]
$\boldsymbol{I}$		$[\boldsymbol{x}_{\mathrm{in}}, 1]$	$\boldsymbol{y}_{\mathrm{in}}$	**[T,F]**	`x([2,1])=mtls(A(:,2),A(:,1),b)`	See [22]	**a**
					`[x(1),x(2),∼,∼,SE,C]=… wtls_line(xin,yin,… ones(m,1),ones(m,1))`	`[C(1),C(3);C(3),C(2)]*SE/(m-n)`	[10, 23] **n**

(Continued.)

Table 2. (Continued.)

$\boldsymbol{W}$				dA**i**	One of (possibly) several alternative solutions (MATLAB command)	$\boldsymbol{\Sigma}'_x$, covariance (either nominal or MATLAB expression to calculate it from outputs)	Reference Footnote **n**um./**a**na.
$\boldsymbol{W}_A$	$\boldsymbol{W}_b$	$\boldsymbol{A}$	$\boldsymbol{b}$	(True, false)			
Fitting a **straight line**, errors in x and y, uncorrelated weights							[10, 23]
$\mathrm{diag}\left(\begin{bmatrix} \boldsymbol{ux} \\ 1 \\ \boldsymbol{uy} \end{bmatrix}^{-2}\right)$		$[\boldsymbol{x}_{\mathbf{in}}, 1]$	$\boldsymbol{y}_{\mathbf{in}}$	[**T,F**]	`[x(1),x(2),~,~,SE,C]=…` `wtls_line(xin,yin,ux,uy)`	`[C(1),C(3);C(3),C(2)]*SE/(m-n)`	**n**
Fitting a **straight line**, errors in x and y, mutually correlated weights							[11, 24]
$\boldsymbol{W}$		$[\boldsymbol{x}_{\mathbf{in}}, 1]$	$\boldsymbol{y}_{\mathbf{in}}$	[**T,F**]	`[x(1),x(2),~,~,SE,C]=wtlsc_line` `(xin,yin,s(1:m),s(m+1:end),rho)`	`[C(1),C(3);C(3),C(2)]*` `SE/(m-n)`	b, **n**
Fitting a **plane,** only uniform errors in z							
0	$\boldsymbol{I}$	$[\boldsymbol{x}_{\mathbf{in}}, \boldsymbol{y}_{\mathbf{in}}, 1]$	$\boldsymbol{z}_{\mathbf{in}}$	[**F,F,F**]	`x=A\b`	$mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}$	**a**
Fitting a **plane**, uniform errors in x, y, z							[21, 25]
$\boldsymbol{I}$		$[\boldsymbol{x}_{\mathbf{in}}, \boldsymbol{y}_{\mathbf{in}}, 1]$	$\boldsymbol{z}_{\mathbf{in}}$	[**T,T,F**]	`x([3,1,2])=mtls(A(:,3),A(:,1:2),b)`	See [22]	**a**
Generalized total least squares, column-wise correlation							[21, 25]
$(\boldsymbol{P}_C \otimes \boldsymbol{I})^{-1}$		$\boldsymbol{A}$	$\boldsymbol{b}$	**T**	`x=gtls([A,b],Pc,I)`	See footnote	c, **a**
Generalized total least squares, row-wise correlation							[21, 25]
$(\boldsymbol{I} \otimes \boldsymbol{P}_R)^{-1}$		$\boldsymbol{A}$	$\boldsymbol{b}$	**T**	`x=gtls([A,b],I,Pr)`	See footnote	c, **a**
Generalized total least squares, column- and row-wise correlation							[21, 25]
$(\boldsymbol{P}_C \otimes \boldsymbol{P}_R)^{-1}$		$\boldsymbol{A}$	$\boldsymbol{b}$	**T**	`x=gtls([A,b],Pc,Pr)`	See footnote	c, **a**
Generalized mixed total least squares, column-wise correlation							[21]
$\left(\begin{bmatrix} \boldsymbol{I} & 0 \\ 0 & \boldsymbol{P}_C \end{bmatrix} \otimes \boldsymbol{I}\right)^{-1}$		$\boldsymbol{A}$	$\boldsymbol{b}$	$[\mathbf{F}_{:,1:n1}, \mathbf{T}_{:,1:n-n1}]$	`x=gmtls(A(:,1:n1),…` `A(:,n1+1:end),b,Pc)`	n.a.	**a**

[a] The TLS solution is given in equation (43). An alternative formulation is: $\boldsymbol{x}_{\mathrm{tls}} = \mathrm{tls}([\boldsymbol{A},\boldsymbol{b}]) = \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} - \sigma_{n+1}^2\boldsymbol{I}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$, with σ_{n+1}, the smallest singular value of $[\boldsymbol{A},\boldsymbol{b}]$ [1]. Simply comparing the LS and TLS expressions (without proof) leads to the TLS covariance $\boldsymbol{\Sigma}_{\mathrm{tls}}([\boldsymbol{A},\boldsymbol{b}]) = mse \cdot \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} - \sigma_{n+1}^2\boldsymbol{I}\right)^{-1}$. It agrees with the covariance matrix found numerically with `cwtls`, but differs from the one given in [26, p 242] by a factor of $m/(m-n)$. Hence, the definition given by van Huffel and Vandewalle is valid for the asymptotic case $m \to \infty$.

[b] The input parameters are connected via $\boldsymbol{\sigma} = [\boldsymbol{\sigma}_x; \boldsymbol{\sigma}_y]$, $\boldsymbol{P} = \mathrm{Corr}(\boldsymbol{\Sigma}) = \mathrm{diag}(\boldsymbol{\rho}, -m) + \mathrm{diag}(\boldsymbol{\rho}, m) + \boldsymbol{I}^{(2m\times 2m)}$, $\boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\sigma}) \cdot \boldsymbol{P} \cdot \mathrm{diag}(\boldsymbol{\sigma})$, and $\boldsymbol{W}_{[da\mathbf{i};db\mathbf{i}],[da\mathbf{i};db\mathbf{i}]} = \boldsymbol{\Sigma}^{-1}$.

[c] Refer to appendix and [25]: the GTLS covariance can be traced back to a TLS covariance: $\boldsymbol{\Sigma}_{\boldsymbol{x},\mathrm{gtls}}([\boldsymbol{A},\boldsymbol{b}]) = \frac{\boldsymbol{W}_{C11} \cdot \boldsymbol{\Sigma}_{\boldsymbol{x},\mathrm{tls}}\left(\boldsymbol{W}_R^{\mathrm{T}}[\boldsymbol{A},\boldsymbol{b}]\boldsymbol{W}_C\right) \cdot \boldsymbol{W}_{C11}^{\mathrm{T}}}{\boldsymbol{W}_{C22}^2}$.

6. Summary and conclusion

A method to solve overdetermined linear systems with arbitrary correlations of the coefficients was introduced. It was optimized in mathematical and technical aspects in terms of computational speed and robustness. It is flexible in application since it allows one to assume single coefficients to be free of error. The covariance of the solutions is also calculated. The solution itself as well as its covariance were validated by comparison with alternative solutions for special cases. The comparison covered all possible correlation scenarios. From that, it can be concluded that the method works reliably and quite fast for all sufficiently well-conditioned real-world problems.

The MATLAB code, with examples, can be found in the supplementary material. It is intended to make it available under the BSD license [27] at the MATLAB central file exchange.

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

Appendix. Special cases with analytical solutions

Known analytical solutions for some special cases are given below in their most compact form. Please note: again, only the formulas for the most important cases of well-defined problems are given. That is, it is assumed that all specified inverse matrices exist.

Least squares (LS) solution

For $\boldsymbol{W_b} \sim \boldsymbol{I}$, $\boldsymbol{W_A} = \boldsymbol{0}$, and $\boldsymbol{W_{Ab}} = \boldsymbol{0}$, so, with only deviations in $\boldsymbol{b}$ but not in $\boldsymbol{A}$ and no correlations allowed, one simply gets the best fitting least squares result as the solution of the Gaussian normal equation as

$$\boldsymbol{x}_{\mathrm{ls}} = \boldsymbol{A}^{+}\boldsymbol{b} \tag{40}$$

with $\boldsymbol{A}^{+} = \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}$, the Moore–Penrose pseudo inverse of $\boldsymbol{A}$.

Weighted least squares (WLS) solution

In the more general case for $\boldsymbol{W_b} \sim \boldsymbol{\Sigma_b}^{-1}$ with $\boldsymbol{\Sigma_b}$, the covariance matrix for $\boldsymbol{b}$, but still $\boldsymbol{W_A} = \boldsymbol{0}$, and $\boldsymbol{W_{Ab}} = \boldsymbol{0}$ the solution is given by

$$\boldsymbol{x}_{\mathrm{wls}} = \left(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\Sigma_b}^{-1}\boldsymbol{A}\right)^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{\Sigma_b}^{-1}\boldsymbol{b}. \tag{41}$$

Total least squares (TLS) solution

With $\boldsymbol{W} \sim \boldsymbol{I}$ (i.e. uniform weights in $\boldsymbol{A}$ and $\boldsymbol{b}$ and no correlations), a total least squares problem is defined. Its solution can be calculated from the singular value decomposition of $[\boldsymbol{A},\boldsymbol{b}]$:

$$\boldsymbol{U}\boldsymbol{\Sigma}\begin{pmatrix} \boldsymbol{V_A} & \boldsymbol{V_{Ab}} \\ \boldsymbol{V_{bA}} & \boldsymbol{V_b} \end{pmatrix}^{\mathrm{T}} = [\boldsymbol{A},\boldsymbol{b}] \tag{42}$$

as

$$\boldsymbol{x}_{\mathrm{tls}} = \mathrm{tls}\left([\boldsymbol{A},\boldsymbol{b}]\right) = -\boldsymbol{V_{Ab}}\boldsymbol{V_b}^{-1}. \tag{43}$$

Generalized total least squares (GTLS) solution

GTLS can deal with a certain structure of deviations in $\boldsymbol{A}$ and $\boldsymbol{b}$ and row- and column-wise correlations [5, 25]. The weight matrix is given as

$$\boldsymbol{W} \sim \left(\boldsymbol{P_C}^{((n+1)\times(n+1))} \otimes \boldsymbol{P_R}^{(m\times m)}\right)^{-1}. \tag{44}$$

Hence, GTLS assumes that all rows and/or all columns have equal covariance matrices.

With the Cholesky factors and their inverse

$$\boldsymbol{C_C}^{\mathrm{T}}\boldsymbol{C_C} = \boldsymbol{P_C} \qquad \boldsymbol{C_R}^{\mathrm{T}}\boldsymbol{C_R} = \boldsymbol{P_R} \tag{45}$$

$$\boldsymbol{W_C} = \boldsymbol{C_C}^{-1} \qquad \boldsymbol{W_R} = \boldsymbol{C_R}^{-1} \tag{46}$$

the GTLS problem can be transformed to an ordinary TLS problem. From

$$\boldsymbol{x'}_{\mathrm{tls}} = \mathrm{tls}\left(\boldsymbol{W_R}^{\mathrm{T}}[\boldsymbol{A},\boldsymbol{b}]\,\boldsymbol{W_C}\right) \tag{47}$$

its TLS-solution, the GTLS-solution can be determined as

$$\boldsymbol{x}_{\mathrm{gtls}} = \mathrm{gtls}\left([\boldsymbol{A},\boldsymbol{b}],\boldsymbol{P_C},\boldsymbol{P_R}\right) = \frac{\boldsymbol{W_{C11}}\boldsymbol{x'}_{\mathrm{tls}} - \boldsymbol{W_{C12}}}{\boldsymbol{W_{C22}}}. \tag{48}$$

Here, $\boldsymbol{W_C}$ was partitioned as

$$\boldsymbol{W_C} = \begin{array}{c} \begin{matrix} n & 1 \end{matrix} \\ \begin{pmatrix} \boldsymbol{W_{C11}} & \boldsymbol{W_{C12}} \\ \boldsymbol{0} & \boldsymbol{W_{C22}} \end{pmatrix} \end{array} \begin{matrix} n \\ 1 \end{matrix}. \tag{49}$$

ORCID iD

Matthias Wurm https://orcid.org/0000-0002-1675-4339

References

[1] Markovsky I and van Huffel S 2007 Overview of total least-squares methods *Signal Process.* **87** 2283–302

[2] Markovsky I 2010 Bibliography on total least squares and related methods *Stat. Interface* **3** 329–34

[3] Premoli A and Rastello M L 2002 The parametric quadratic form method for solving TLS problems with elementwise weighting *Total Least Squares and Errors-in-Variables Modeling* (Berlin: Springer) pp 67–76

[4] Markovsky I, Rastello M L, Premoli A, Kukush A and van Huffel S 2006 The element-wise weighted total least-squares problem *Comput. Stat. Data Anal.* **50** 181–209

[5] Schuermans M, Markovsky I, Wentzell P D and van Huffel S 2005 On the equivalence between total least squares and maximum likelihood PCA *Anal. Chim. Acta* **544** 254–67

[6] Schuermans M, Markovsky I and van Huffel S 2007 An adapted version of the element-wise weighted total least squares method for applications in chemometrics *Chemom. Intell. Lab. Syst.* **85** 40–46

[7] Schaffrin B and Wieser A 2008 On weighted total least-squares adjustment for linear regression *J. Geod.* **82** 415–21

[8] Mahboub V 2012 On weighted total least-squares for geodetic transformations *J. Geod.* **86** 359–67

[9] Amiri-Simkooei A and Jazaeri S 2012 Weighted total least squares formulated by standard least squares theory *J. Geod. Sci.* **2** 113–24

[10] Krystek M and Anton M 2007 A weighted total least-squares algorithm for fitting a straight line *Meas. Sci. Technol.* **18** 3438

[11] Krystek M and Anton M 2011 A least-squares algorithm for fitting data points with mutually correlated coordinates to a straight line *Meas. Sci. Technol.* **22** 035101

[12] Snow K B 2012 Topics in total least-squares adjustment within the errors-in-variables model: singular cofactor matrices and prior information *PhD Thesis* Ohio State University, Columbus, OH

[13] Zhou Y, Kou X, Zhu J and Li J 2014 A Newton algorithm for weighted total least-squares solution to a specific errors-in-variables model with correlated measurements *Stud. Geophys. Geod.* **58** 349–75

[14] Zhou Y and Fang X 2016 A mixed weighted least squares and weighted total least squares adjustment method and its geodetic applications *Surv. Rev.* **48** 421–9

[15] Wurm M, Grunewald T, Teichert S, Bodermann B, Reck J and Richter U 2020 Some aspects on the uncertainty calculation in Mueller ellipsometry *Opt. Express* **28** 8108–31

[16] BIPM, IEC, IFCC, ILAC, ISO, IUPAC, IUPAP and OIML 2008 Evaluation of measurement data—guide to the expression of uncertainty in measurement Joint Committee for Guides in Metrology, JCGM 100: 2008

[17] van Loan C F and Pitsianis N 1993 Approximation with Kronecker products *Linear Algebra for Large Scale and Real-time Applications* (Berlin: Springer) pp 293–314

[18] Eichhorn H 1989 Efficiency: a new concept in parameter estimation *Bull. Astron. Inst. Czech.* **40** 394–5

[19] d'Errico J 2021 Adaptive robust numerical differentiation (MATLAB Central File Exchange) (available at: www.mathworks.com/matlabcentral/fileexchange/13490-adaptive-robust-numerical-differentiation) (Accessed 9 March 2021)

[20] Wurm M Slightly modified code of [23], (unpublished)

[21] Houtzager I 2021 Total least squares with mixed and/or weighted disturbances (MATLAB Central File Exchange) (available at: www.mathworks.com/matlabcentral/fileexchange/50332-total-least-squares-with-mixed-and-or-weighted-disturbances) (Accessed 9 March 2021)

[22] Branham R L Jr 1999 A covariance matrix for total least squares with heteroscedastic data *Astron. J.* **117** 1942

[23] Anton M 2021 Weighted total least squares straight line fit (MATLAB Central File Exchange) (available at: www.mathworks.com/matlabcentral/fileexchange/17466-weighted-total-least-squares-straight-line-fit) (Accessed 9 March 2021)

[24] Anton M 2021 weighted total least squares for mutually correlated coordinates (MATLAB Central File Exchange) (available at: www.mathworks.com/matlabcentral/fileexchange/30193-weighted-total-least-squares-for-mutually-correlated-coordinates) (Accessed 9 March 2021)

[25] Rhode S, Usevich K, Markovsky I and Gauterin F 2014 A recursive restricted total least-squares algorithm *IEEE Trans. Signal Process.* **62** 5652–62

[26] van Huffel S and Vandewalle J 1991 *The Total Least Squares Problem: Computational Aspects and Analysis* (Philadelphia, PA: Society for Industrial and Applied Mathematics)

[27] Regents of the University of California *Open Source Initiative* (available at: https://opensource.org/licenses/BSD-3-Clause) (Accessed 12 November 2021)