Geschke *et al. Economic Structures* (2019) 8:2
https://doi.org/10.1186/s40008-019-0133-7

Journal of Economic Structures

RESEARCH Open Access

Balancing and reconciling large multi-regional input–output databases using parallel optimisation and high-performance computing

Arne Geschke[1*], Julien Ugon[2], Manfred Lenzen[1], Keiichiro Kanemoto[3] and Daniel Dean Moran[4]

*Correspondence:
arne.geschke@sydney.edu.au
[1] Integrated Sustainability Analysis, School of Physics, University of Sydney, Sydney, Australia
Full list of author information is available at the end of the article

Abstract

Over the past decade, large-scale multi-regional input–output (MRIO) tables have advanced the knowledge about pressing global issues. At the same time, the data reconciliation strategies required to construct such MRIOs have vastly increased in complexity: large-scale MRIOs are more detailed and hence require large amounts of different source data which are in return often varying in quality and reliability, overlapping, and, as a result, conflicting. Current MRIO reconciliation approaches—mainly RAS-type algorithms—cannot fully address this complexity adequately, since they are either tailored to handle certain classes of constraints only, or their mathematical foundations are currently unknown. Least-squares-type approaches have been identified to offer a robust mathematical framework, but the added complexity in terms of numerical handling and computing requirements has so far prevented the use of these methods for MRIO reconciliation tasks. We present a new algorithm (ACIM) based on a weighted least-squares approach. ACIM is able to reconcile MRIO databases of equal or greater size than then currently largest global MRIO databases. ACIM can address arbitrary linear constraints and consider lower and upper bounds as well as reliability information for each given data point. ACIM is based on judicious data preprocessing, state-of-the art quadratic optimisation, and high-performance computing in combination with parallel programming. ACIM addresses all shortcomings of RAS-type MRIO reconciliation approaches. ACIM's was tested on the Eora model, and it was able to demonstrate improved runtimes and source data adherences.

Keywords: Input–output analysis, Constrained optimisation, Matrix balancing, Least-squares approach, RAS method

1 Introduction

Reconciling[1] data from different sources to compile multi-regional input–output (MRIO) databases is the task of finding a balanced MRIO table that closely matches all considered data sources. This task can be formulated as a mathematical optimisation problem

[1] In the literature, the reconciliation of MRIO tables is often referred to as *balancing*. Strictly speaking, *balancing* limits the reconciliation task to a very narrow class of constraints. The authors are using the broader term *reconciliation* to indicate that there is no limitation in terms of classes of constraints that can be handled.

$$\min_{\boldsymbol{\tau}} f(\boldsymbol{\tau}, \boldsymbol{\tau}^0, \boldsymbol{\sigma}_{\boldsymbol{\tau}^0}, \boldsymbol{\sigma}_{\mathbf{c}}) \qquad \text{subject to} \qquad \mathbf{M}\boldsymbol{\tau} = \mathbf{c}, \qquad \mathbf{l} \leq \boldsymbol{\tau} \leq \mathbf{u}. \tag{1}$$

For this formulation, an arbitrary MRIO table $\mathbf{T}$ is vectorised into the vector $\boldsymbol{\tau}$. The vector $\boldsymbol{\tau}^0$ is the vectorised MRIO *prior* to the reconciliation process, and the optimal solution $\boldsymbol{\tau}^*$ represents the final MRIO *after* the reconciliation. The function f is the *objective function* which measures the departure of $\boldsymbol{\tau}$ from $\boldsymbol{\tau}^0$ during the reconciliation process. The matrix $\mathbf{M}$ holds the linear mathematical constraints, and the vector $\mathbf{c}$ contains the corresponding external or superior data. The final table $\boldsymbol{\tau}^*$ must adhere to the equation $\mathbf{M}\boldsymbol{\tau}^* = \mathbf{c}$. The vectors $\mathbf{l}$ and $\mathbf{u}$ contain upper and lower bounds for every element of the MRIO.

Restricting the constraints set to linear constraints does not pose a problem in MRIO reconciliation, as most conditions that are posed on the MRIO table elements by superior data (such as balancing, adhered to sector totals, matching an aggregated table) can be interpreted as linear constraints.

The formulation given in Eq. (1) also includes reliability data for $\boldsymbol{\tau}^0$ and $\mathbf{c}$, given in the vectors $\boldsymbol{\sigma}_{\boldsymbol{\tau}^0}$, and $\boldsymbol{\sigma}_{\mathbf{c}}$. We are following the interpretation of the reliability data as it has been proposed in previous publications [see, for example, Lenzen et al. (2010)]. That is, each element of an MRIO table is interpreted as the expected value of a normally distributed random variable, with the standard deviation being a measure for the reliability of each value. Hence, each element in $\boldsymbol{\tau}^0$ is accompanied by a value for its standard deviation, and these values are pooled in the vector $\boldsymbol{\sigma}_{\boldsymbol{\tau}^0}$.

The objective function f typically measures the deviation between the original unbalanced and the balanced estimate.

From a mathematical viewpoint, it is usually the problem that defines which objective function should be used to measure this deviation. For example, in order to minimise the travelled distance between two arbitrary points on a plane, a standard Pythagorean distance measure is applied. If the ways on which one can travel across the plane are restricted, for example by a road, then the Pythagorean measure does not suit the problem anymore, and the distance between two points must be measured so that the possible pathways between the two points of interest are considered correctly. After all, one cannot walk straight through a building! Only once the correct objective function is found, an appropriate algorithm to find the shortest way can be chosen.

This pathway towards finding the appropriate objective function for compiling MRIOs has been somewhat reversed: suitable algorithms were developed first, and their mathematical understanding grew in subsequent years.

Over the decades, two mathematical approaches prevailed for the task of data reconciliation during MRIO compilation. These can be broadly classified as *cross-entropy methods* and *quadratic programming (QP) methods*. In this paper, we are proposing a new algorithm for a least-squares approach, which falls into the category of QP methods.

The idea of using a least-squares objective function was first proposed in Stone et al. (1942). It was further developed into a fully quadratic model by Byron (1978, 1996) and van der Ploeg (1982, 1984, 1988), who also introduced error terms to control uncertainties [see also Harrigan and Buchanan (1984), Morrison and Thumann (1980)]. In order to motivate this function, van der Ploeg distinguishes between so-called *soft* and *hard* constraints in the matrix $\mathbf{M}$ and the vector $\mathbf{c}$.

In the following, $\mathbf{m}^T$ describes the *transposed* of the vector $\mathbf{m}$, where the different $\mathbf{m}_i^T$ are the rows of $\mathbf{M}$. Each row in $\mathbf{m}_i^T$ in $\mathbf{M}$ and its corresponding value c_i in $\mathbf{c}$ represent one constraint. As a preparatory measure, the aim is now to identify for each row of $\mathbf{M}$ if it presents a so-called *hard* or *soft* constraint. The index s is used as the counter for the soft constraints, and the index h is the counter for the hard constraints.

A row $\mathbf{m}_s^T$ of $\mathbf{M}$ and the corresponding value $\mathbf{c}_s$ of $\mathbf{c}$ are considered a *soft constraints*, if for the corresponding standard deviation $\boldsymbol{\sigma}_{\mathbf{c},s} \neq 0$ holds (standard deviation for the superior data point is not zero). Consequently, $\mathbf{m}_h^T$ and $\mathbf{c}_h$ form a *hard constraint* if $\boldsymbol{\sigma}_{\mathbf{c}}, h = 0$. Plainly speaking, a soft constraint is a constraint that can remain unfulfilled (within the limits defined by the standard deviation), whereas hard constraints must be adhered to perfectly.

Hard constraints are typically the so-called balancing constraints, which ensure that the final MRIO adheres to the condition that the total output of each sector must equals the sector's inputs. Other data, such as national statistical data or international trade data, do not have to be adhered to perfectly, and this is represented by the positive values in $\boldsymbol{\sigma}_{\mathbf{c},s} \neq 0$, which is the condition for a soft constraint. For example, the GDP for a country is never perfectly reliable, and a deviation by a few per cent does not render the final table as faulty. However, official statistics data should of course be adhered to as much as possible.

The soft and hard constraints can then be pooled in different matrices, splitting the original matrix $\mathbf{M}$ into separate matrices. Let $\mathbf{M}_{\text{soft}}$ only contain soft constraints and $\mathbf{M}_{\text{hard}}$ only contain hard constraints. The vector $\mathbf{c}$ is split into $\mathbf{c}_{\text{soft}}$ and $\mathbf{c}_{\text{hard}}$ accordingly.

Using this concept, van der Ploeg introduces error terms ε_s to the soft constraints, which account for the deviation from the corresponding value in $c_{\text{soft},s}$.

$$\mathbf{m}_{\text{soft},s}^T \boldsymbol{\tau} = c_{\text{soft},s} + \varepsilon_s.$$

Van der Ploeg elegantly reinterprets these error terms. Let $\boldsymbol{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_S)^T$ be the vector of all error terms, then

$$\begin{aligned} & \mathbf{M}_{\text{soft}}\, \boldsymbol{\tau} = \mathbf{c}_{\text{soft}} + \boldsymbol{\varepsilon} \\ \Longleftrightarrow\ & \mathbf{M}_{\text{soft}}\, \boldsymbol{\tau} - \boldsymbol{\varepsilon} = \mathbf{c}_{\text{soft}} \\ \Longleftrightarrow\ & \mathbf{M}_{\text{soft}} - \mathbf{I} \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{\varepsilon} \end{pmatrix} = \mathbf{c}_{\text{soft}}\,, \end{aligned}$$

where $\mathbf{I} \in \mathbb{R}^{S \times S}$ is the unity matrix. Note that error terms are *only* applied to soft constraints. The hard constraints are introduced to this notation by

$$\begin{pmatrix} \mathbf{M}_{\text{hard}} & \mathbf{0}^{H \times S} \\ \mathbf{M}_{\text{soft}} & -\mathbf{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{\varepsilon} \end{pmatrix} = \begin{pmatrix} \mathbf{c}_{\text{hard}} \\ \mathbf{c}_{\text{soft}} \end{pmatrix}, \tag{2}$$

where $\mathbf{0}^{H \times S}$ is a matrix with all-zero elements of dimension $H \times S$, in which H is the total number of hard constraints and S is the total number of soft constraints. By defining

$$\mathbf{p} := \begin{pmatrix} \boldsymbol{\tau} \\ \boldsymbol{\varepsilon} \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{\sigma}_{\mathbf{p}} := \begin{pmatrix} \boldsymbol{\sigma}_{\boldsymbol{\tau}} \\ \boldsymbol{\sigma}_{\mathbf{c}} \end{pmatrix}, \tag{3}$$

Equation (2) can be rewritten as

$$\underbrace{\begin{pmatrix}\mathbf{M}_{\text{hard}} & \mathbf{0}^{H\times S}\\ \mathbf{M}_{\text{soft}} & -\mathbf{I}\end{pmatrix}}_{=:\mathbf{G}}\underbrace{\begin{pmatrix}\boldsymbol{\tau}\\ \boldsymbol{\varepsilon}\end{pmatrix}}_{=:\mathbf{p}} = \mathbf{c},$$

The elements of the MRIO, now represented within the vector $\mathbf{p}$, may also be subject to boundary conditions. The most common boundary condition is the non-negativity condition, which ensure that financial transaction in an input–output table can usually not be negative (with a few exceptions).

Van der Ploeg defines the full reconciliation problem as

$$\min_{\mathbf{p}\in\mathbb{R}^N}(\mathbf{p}-\mathbf{p}^0)^T\hat{\boldsymbol{\sigma}}_{\mathbf{p}^0}^{-2}(\mathbf{p}-\mathbf{p}^0) \qquad \text{subject to} \qquad \mathbf{Gp}=\mathbf{c} \text{ and } \mathbf{l}\le\mathbf{p}\le\mathbf{u}. \tag{4}$$

Hence, the objective function that is used here is a weighted least-squares function, with the standard deviation values serving as the weights.

Here, the hat operator $\hat{\boldsymbol{\sigma}}_{\mathbf{p}^0}$ depicts the diagonalisation of the vector $\boldsymbol{\sigma}_{\mathbf{p}^0}$. It should be pointed out that van der Ploeg's formulation elegantly reinterprets the reliability values for the constraint values in $\mathbf{c}_{\text{soft}}$ as inverted reliability values for the error terms ε. Note also that the error terms are unbounded, that is $\boldsymbol{l}\in(\mathbb{R}\cup\{-\infty\})^N$ and $u\in(\mathbb{R}\cup\{+\infty\})^N$. Note further that due to its construction the values in $\boldsymbol{\sigma}_{\mathbf{p}^0}$ cannot be 0, and an inversion and squaring of the values as it is carried out in $\hat{\boldsymbol{\sigma}}_{\mathbf{p}^0}^{-2}$ does not pose a numerical problem.

This reconciliation problem fulfils the form given in Eq. (1), since the reliability data for both the MRIO elements *and* the superior data are considered.

2 RAS-type cross-entropy methods and the quadratic programming approach

Thus far, the discussion around the compiling MRIO matrices identified two different types of algorithms that were considered for this task: quadratic programming algorithms and RAS-type methods, which are specialised cross-entropy methods. RAS-type methods have so far enjoyed more attention, but there are fundamental differences between the development of the algorithm presented in this paper and the algorithms summarised in the family of RAS-type methods.

Stone (1961) presented RAS as a calculation method for one task only: balancing a matrix according to given row- and column totals. It took another 9 years before Bacharach (1970) discovered that RAS is in fact an optimisation method that solves a version of the problem given in Eq. (1). The original RAS method is limited to balancing constraints only, it can only handle positive elements in the matrix (in our case: the MRIO), and it can neither consider reliability information nor boundary conditions. To cater for various needs such as these, the algorithm has been modified and extended numerous times. For example, Gilchrist and Louis (1999) presented a *three-stage RAS* (TRAS) aimed at allowing the consideration of partially aggregated data, Junius and Oosterhaven (2003) presented a *generalised RAS* (GRAS) that allowed positive and negative elements within the MRIO. Mínguez et al. (2009) proposed the CRAS algorithm which takes into account additional information when a time series of MRIO tables is available. The most recent extension of RAS was presented by Lenzen et al. (2010). Their *Konfliktfreies RAS*

(KRAS) allows the use of arbitrary constraints and the use of reliability data for both the MRIO elements and the superior data. KRAS is the first RAS-type method capable of solving the problem given in Eq. (1). Lahr and de Mesnard (2004) and Lenzen et al. (2010) gave detailed overviews of the different RAS variants.

The development of RAS and its extensions was usually driven by the need of solving the reconciliation problem for a certain class of constraints. The mathematical interpretation of the method usually followed *after* the development of the method itself. Although this incremental development approach has enabled steady progress in MRIO compilation capabilities, the process of analysing the RAS modifications can be tedious. Such rigorous mathematical analyses allow for rational comparison of algorithms and informed reconciliation methodologies. In the case of the most recent RAS development—KRAS—the mathematical interpretation has *not* been carried out yet and the objective function of the algorithm remains unknown.

In contrast, QP-based algorithms provide a robust mathematical base. However, a general-purpose, single-step QP algorithm for MRIO compilation is currently not available.

A number of large-scale global MRIO databases such as Eora, EXIOBASE, WIOD, and GTAP [see Tukker and Dietzenbacher (2013)] were developed over recent years. These databases require large amounts of input data from varying sources. This inevitably leads to conflicting information during the data processing stage. Due to the amount of data to be considered during the reconciliation and the—compared to RAS-type approaches—relatively high complexity of van der Ploeg's QP approach, the developers of these databases have so far employed QP-type approaches—if at all—only for subproblems during the data reconciliation stage. For example, EXIOBASE, EXIOBASE2, and EXIOBASE3 were compiled using multi-step reconciliation processes. Some of the intermediate steps in this processes were completed using QP-type algorithms, with the final global balancing step being carried using a RAS-type algorithm (Wood et al. 2014, 2015; Stadler et al. 2018).

The mathematical effects of mixing both RAS-type and QP-type reconciliation techniques in a multi-step process have so not been examined. To the authors' knowledge, only the Eora database is reconciled using a reconciliation method that considers all available input data in a single-step reconciliation process (namely the KRAS method). While this removes any potential effects of using a multi-step process on the data quality, a mathematical analysis has not been carried out for KRAS.

Thanks to its rigorous mathematical properties, the quadratic approach proposed by van der Ploeg (1982, 1984, 1988), Harrigan and Buchanan (1984) and Morrison and Thumann (1980) has attracted some attention. Canning and Wang (2005) noted its flexibility and use it to construct an IRIO model. However, the lack of efficient and easy-to-use methods and algorithms for the use of this approach with larger MRIO tables and their corresponding dimensionality has prevented a more widespread adoption. But, as mentioned above, partial uptake of QP-type methods has occurred. In general, the dimension of practical reconciliation problems such as the construction of the global MRIO databases mentioned above is too large for general, single-step quadratic optimisation algorithms.

In this paper, the authors present a mathematical optimisation method that is

- based on van der Ploeg's approach and is hence based on a robust mathematical theory,
- is able to consider arbitrary linear constraints during the reconciliation of MRIO databases, and
- uses mathematical and computational parallelisation in order to allow for it to be applied to large-scale MRIO databases of equal or greater size than the currently biggest MRIO databases in existence.

To the authors' knowledge, this is the first algorithm that fulfils these three requirements and it hence firstly overcomes the problems discussed above and secondly allows for the application of a quadratic reconciliation approach to large-scale MRIOs.

The algorithm is motivated by starting from a mathematical formulation of the problem of reconciling an MRIO subject to a set of arbitrary constraints. Exploiting the structure of this formulation (in particular by interpreting the constraints geometrically), the authors present an algorithm that solves the given problem. In this paper, we will exploit the structure of the system to develop such an algorithm and apply it to reconcile one the global MRIO databases: the Eora model (Lenzen et al. 2012, 2013).

The remainder of this paper is organised as follows: Sect. 3 states the requirements for a least-squares algorithm designed to reconcile large MRIO databases, the ACIM algorithm is presented in Sect. 4, the implementation is described in Sects. 5 and 6 contains conclusions.

3 Requirements for a least-squares algorithm to reconcile large MRIO databases

System (4) is split into two subsystems: the soft constraints matrix is very large, but very sparse (a typical row may contain only a few nonzero values), while the hard constraints matrix represents balancing constraints, which are present more structure (though also denser). In this paper, we propose to use a splitting algorithm. Such algorithms allow one to solve subsystems independently from each other and use these solutions to construct a new iterate. Using this approach, we will then develop strategies for each of the soft and hard constraints. The idea of using a splitting algorithm for matrix reconciliation problem was suggested, for example, by Nagurney and Eydeland (1992).

A least-squares algorithm designed to solve the problem given in Eq. (4) must be able to handle large amounts of data, and to provide an accurate solution within an acceptable time frame. In the case of Eora, the vector $\boldsymbol{\tau}$ has around 10^9 entries and the matrix $\mathbf{G}$ has around 10^6 rows. Together, the complete set of input data for the problem in Eq. (4) occupy more than 80 GB of space. In order to accommodate for this amount of data, we design an algorithm to be run in parallel on a HPC cluster. This will enable the simultaneous reconciliation of subproblems and hence reduce runtime. Splitting methods are particularly well suited to a HPC environment, where each subsystem can be solved in parallel. At the same time, the algorithm features a number of mathematical acceleration measures that ensure faster convergence. In order to achieve this, certain geometric properties of the individual rows of $\mathbf{G}$ are considered during the calculation process.

Our findings show that given the size and complexity of the reconciliation task at hand, both the computational parallelisation and the mathematical accelerations must be applied when designing the ACIM algorithm.

For the optimisation problem given in Eq. (4), define

$$\begin{aligned} \mathbf{z} &= \boldsymbol{\sigma}^{-1}\mathbf{p} \\ \mathbf{z}^0 &= \boldsymbol{\sigma}^{-1}\mathbf{p}^0 \\ \text{and} \qquad \mathbf{A} &= \mathbf{G}\boldsymbol{\sigma}. \end{aligned}$$

Then Eq. (4) simplifies to

$$\min_{\mathbf{z}\in\mathbb{R}^N} (\mathbf{z}-\mathbf{z}^0)^T(\mathbf{z}-\mathbf{z}^0) \qquad \text{subject to} \qquad \mathbf{A}\mathbf{z} = \mathbf{c} \text{ and } \boldsymbol{\sigma}^{-1}\mathbf{l} \le \mathbf{z} \le \boldsymbol{\sigma}^{-1}\mathbf{u}. \tag{5}$$

Therefore, for the sake of simplicity, $\boldsymbol{\sigma}^{-2}$ is ignored at this point. It is further assumed that the reformulation of the problem in order to obtain the form described in Eq. (5) has been carried out.

Hence, the problem simplifies to

$$\min_{\mathbf{p}\in\mathbb{R}^N} (\mathbf{p}-\mathbf{p}^0)^T(\mathbf{p}-\mathbf{p}^0) \qquad \text{subject to} \qquad \mathbf{G}\mathbf{p} = \mathbf{c} \text{ and } \mathbf{l} \le \mathbf{p} \le \mathbf{u}. \tag{6}$$

The vector $\mathbf{p}^0$ is the *initial estimate* of the problem, and $\mathbf{l}$ and $\mathbf{u}$ are the vectors of *lower* and *upper bounds*, respectively. Throughout this section, $\mathbf{G}$ will be an $(M \times N)$-dimensional matrix. Therefore, $\mathbf{p}, \mathbf{l}, \mathbf{u} \in \mathbb{R}^N$ and $\mathbf{c} \in \mathbb{R}^M$. It will be further assumed that for the rows $\|\mathbf{g}_m\|$ of $\mathbf{G}$ the equation $\|\mathbf{g}_m\| = 1$ holds for all $m = 1, \ldots, M$.

Definition 1 *(Defect and Residual)* Let $\mathbf{p}^*$ denote the solution of the problem given in Eq. (6); hence, $\mathbf{G}\mathbf{p}^* = \mathbf{c}$ and $\mathbf{l} \le \mathbf{p}^* \le \mathbf{u}$ hold. Let $\mathbf{p}^k$ be an iterate generated by an iterative optimisation algorithm in the kth iteration step. Then

$$\begin{aligned} \delta^k &:= \mathbf{p}^k - \mathbf{p}^* && \text{is called the } \textit{defect} \text{ of the } k\text{-th iteration, and} \\ \mathbf{r}^k &:= \mathbf{G}\mathbf{p}^k - \mathbf{c} && \text{is called the } \textit{residual} \text{ of the } k\text{-th iteration.} \end{aligned}$$

Unless otherwise stated, $\|\cdot\|$ denotes the Euclidean norm.

4 The ACIM algorithm

We base our algorithm on related algorithm presented in Scolnik et al. (2000). This algorithm has all the characteristics described in the previous section and is shown to have promising convergence behaviour. This algorithm is based on Cimmino's algorithm and can be described as follows: the system $\mathbf{G}\mathbf{p} = \mathbf{c}$ is split into k subsystems (or blocks) $(\mathbf{G}_1\mathbf{p} = \mathbf{c}_1, \ldots, \mathbf{G}_k\mathbf{p} = \mathbf{c}_k)$. The splitting of the original constraints matrix $\mathbf{G}$ and the corresponding right-hand side vector $\mathbf{c}$ serves two purposes:

1. Prepare the problem for the computational parallelisation by splitting $\mathbf{G}$ and $\mathbf{c}$ into blocks (or subproblems) that will later be processed in parallel, and

2. Design each subproblem so that it can be solved efficiently

Solving the subsystems $\mathbf{G}_1\mathbf{p} = \mathbf{c}_1, \ldots, \mathbf{G}_k\mathbf{p} = \mathbf{c}_k$ is the most time-intensive part in each iteration. At the same time, the proposed algorithm allows for these subsystems to be solved independently. Hence, each subsystem will be assigned to a separate computing core, and the subsystems can therefore be solved in parallel, which reduces the runtime of the algorithm tremendously.

In order to decide on how we split $\mathbf{G}$ and $\mathbf{c}$, we must understand which algorithms we will use to construct ACIM. By understanding the strengths of the different algorithms, we can then design a blocking concept for $\mathbf{G}$ and $\mathbf{c}$ such that the resulting subproblems are tailored to exploit the strengths of ACIM and it components.

ACIM is a nested algorithm, consisting of three individual algorithms, paired with mathematical acceleration measures. We will first present the different algorithms that make up ACIM, and then present the acceleration measures, and finally, using the lessons learned from the different algorithms, we will present the blocking approach.

Before we start looking at the individual algorithms, it is worthwhile looking at the geometric interpretation of the constraints sets

4.1 Geometric interpretation of the constraints set as hyperplanes

We will only present a shortened version of the theory behind hyperplanes. For more information, the reader may wish to refer to standard linear algebra literature such as Fischer (2000). It is helpful to take a closer look at the constraints $\mathbf{G}\mathbf{p} = \mathbf{c}$ of Eq. 6. Let $\mathbf{g}_m$ be an arbitrary row of the matrix $\mathbf{G}$ and let $\mathbf{g}_m \neq \mathbf{0}$ hold. The constraint corresponding to the m-th row of $\mathbf{G}$ is given by

$$\mathbf{g}_m^T\mathbf{p}^* = \mathbf{c}_m\,,$$

where $\mathbf{p}^*$, $\mathbf{p}^* \in \mathbb{R}^N$ is a solution of this constraint. Then, the set

$$H_{\mathbf{g}_m,c_m} = \{\mathbf{p} \in \mathbb{R}^N \mid \langle \mathbf{g}_m, \mathbf{p} - \mathbf{p}^* \rangle = 0\}$$

defines a hyperplane of dimension $\mathbb{R}^{N-1}$. And we obtain

Lemma 1 (Orthogonal projection onto a hyperplane) *Let $\mathbf{p} \in \mathbb{R}^N$ be a vector and let $H_{\mathbf{g},c} \subset \mathbb{R}^N$ be a hyperplane defined by the vector $\mathbf{g} \in \mathbb{R}^N$ and the value $c \in \mathbb{R}$. Then, the orthogonal projection $\mathbf{p}_p$ of $\mathbf{p}$ onto $H_{\mathbf{g},c}$ is given by*

$$\mathbf{p}_p = \mathbf{p} + \frac{c - \langle \mathbf{g}, \mathbf{p} \rangle}{\|\mathbf{g}\|^2}\mathbf{g},$$

and $\mathbf{p}_p \in H_{\mathbf{g},c}$ holds.

Proof Available in any standard literature on linear algebra, for example Fischer (2000). □

Remark 1 Lemma 1 also holds if $\mathbf{p}$ is already part of the hyperplane $H_{\mathbf{g},c}$. In this case, $\mathbf{p}$ and its projection $\mathbf{p}_p$ are identical, i.e. $\mathbf{p} = \mathbf{p}_p$.

Remark 2 Any solution $\mathbf{p}^*$ for the problem given in Eq. (4) lies in the intersection

$$\mathbf{p}^* \in \bigcap_{m=1}^{M} H_{\mathbf{g}_m,c_m}$$

of the hyperplanes defined by the rows of $\mathbf{G}$ and $\mathbf{c}$.

4.2 The components of ACIM

ACIM follows the concept of Cimmino's algorithm (see Fig. 1), which will be presented first.

Algorithm 1 *(Cimmino's Algorithm)* Define $\alpha > 0$, $\varepsilon > 0$, and w_m, $m = 1, \ldots, M$ with $\sum_{m=1}^{M} w_m = 1$. Choose an initial estimate $\mathbf{p}^0$. It is further assumed that $\|\mathbf{g}_m\| = 1$ for all $m = 1, \ldots, M$.

0. Initialise $k = 0$
1. Project $\mathbf{p}^k$ onto each hyperplane: $\mathbf{p}^k_{p,m} = \mathbf{p}^k + (c_m - \mathbf{g}_m^T \mathbf{p}^k)\mathbf{g}_m$.
   $\mathbf{p}^k_{p,m}$ is the corresponding projection of $\mathbf{p}^k$ onto the hyperplane defined by the mth row $\mathbf{g}_m^T$ of matrix $\mathbf{G}$, $m = 1, \ldots, M$.
2. Compute the barycentre of all projections: $\mathbf{p}^k_b = \sum_{m=1}^{M} w_m \mathbf{p}^k_{p,m}$.
3. Compute the new iterate: $\mathbf{p}^{k+1} = \mathbf{p}^k + \alpha(\mathbf{p}^k_b - \mathbf{p}^k)$.
4. If $\mathbf{G}\mathbf{p}^{k+1} - \mathbf{c} < \varepsilon$, a solution has been found. Set $\mathbf{p}^* = \mathbf{p}^{k+1}$ and terminate the algorithm.
   If $\mathbf{G}\mathbf{p}^{k+1} \neq \mathbf{c}$ set $k = k + 1$ and return to Step 1.

It is shown by Scolnik et al. (2000) that the algorithm converges (although it is not actually shown in Scolnik et al. (2000), it is easy to verify that the algorithm terminates after at most m iterations in a typical conjugate direction manner. However, here we apply it as an iterative algorithm) (Fig. 1).

The main step in this algorithm is the projection step 1, and our aim is to make this projection step as efficient as possible. In order to achieve this, we will attempt to split $\mathbf{G}$ into blocks such that the projection onto each of these blocks is robust and fast. This *block projection* will be carried out by Kaczmarz's algorithm (Fig. 2).

Algorithm 2 *(Kaczmarz's Algorithm)* For a problem of the form

$$\min_{\mathbf{p}\in\mathbb{R}^N} (\mathbf{p} - \mathbf{p}^0)^T (\mathbf{p} - \mathbf{p}^0) \qquad \text{subject to} \qquad \mathbf{G}\mathbf{p} = \mathbf{c} \tag{7}$$

choose an initial estimate $\mathbf{p}^0$. Set $m = 1$, $k = 0$, and define the following iteration procedure

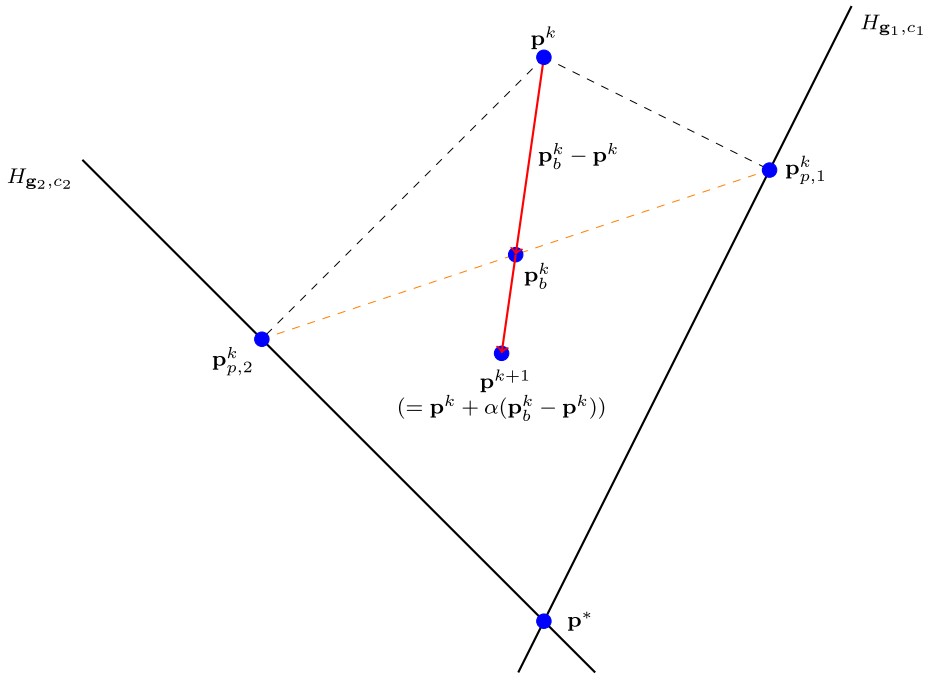

Fig. 1 Visualisation of Cimmino's algorithm. This figure shows the concept of Cimmino's algorithm for a two-dimensional problem with two constraints for the iteration step $k \to k+1$. In this case, the parameters were defined as $w_{1,2} = 0.5$ and $\alpha = 1.5$. The dashed black lines indicate the orthogonal projections of $\mathbf{p}^k$ onto the hyperplanes, and the dashed orange line indicates the calculation of the barycentre $\mathbf{p}_b^k$. A choice of different weights $w_{1,2}$ would result in the barycentre $\mathbf{p}_b^k$ being shifted along the dashed orange line

Step 1

$$\mathbf{p}^{k+1} = \mathbf{p}^k + \frac{c_m - \langle \mathbf{g}_m, \mathbf{p}^k \rangle}{\|\mathbf{g}_m\|^2} \mathbf{g}_m. \tag{8}$$

If $\mathbf{G}\mathbf{p}^{k+1} = \mathbf{c}$, terminate the algorithm with the solution $\mathbf{p}^{k+1}$.

If $\mathbf{G}\mathbf{p}^{k+1} \neq \mathbf{c}$, set $k = k+1$ and

Step 1.1 Set $m = m+1$ if $m < M$ or
Step 1.2 Set $m = 1$ if $m = M$,

and return to Step 1. Note that M is the number of constraints.

Kaczmarz's algorithm uses elements that were already present in Cimmino's algorithm: orthogonal projections onto hyperplanes. But Kaczmarz's algorithm only projects onto one hyperplane per iteration. Hence, if the hyperplanes were all orthogonal to one another, individual projections carried out in one iteration would not be voided by subsequent projections. The projection onto the next hyperplane would occur on the hyperplanes that previous iterations projected onto. Hence, Kaczmarz's algorithm can find a solution for a linear system very efficiently if the hyperplanes of the problem are orthogonal to one another. The blocking algorithm will be based on the approach to build individual blocks that only contain hyperplanes that fulfil this condition.

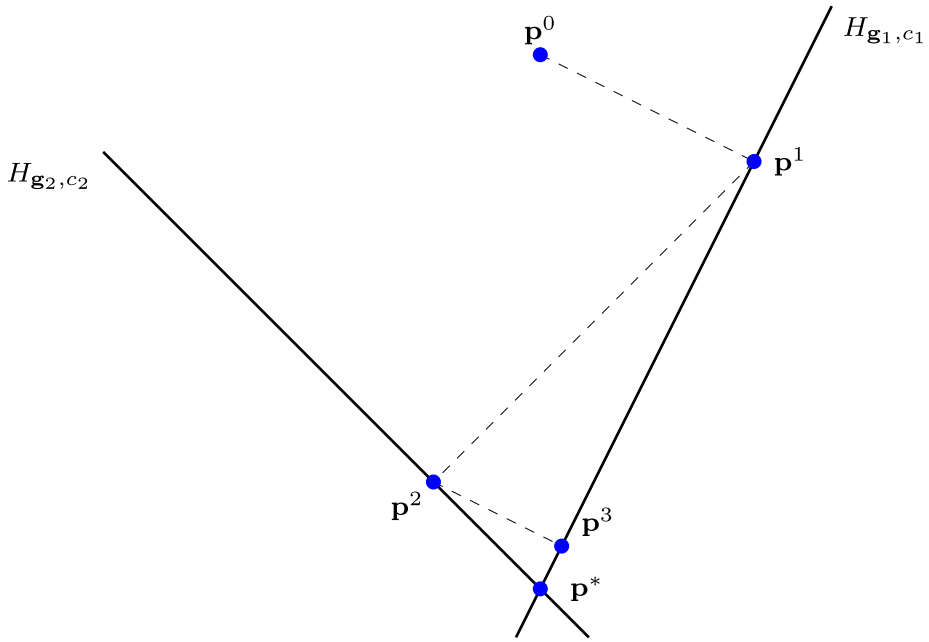

Fig. 2 Visualisation of Kaczmarz's algorithm. This visualisation shows the concept of Kaczmarz's algorithm, with two hyperplanes. The initial estimate $\mathbf{p}^0$ is projected onto the first hyperplane, yielding $\mathbf{p}^1$. The first iterate $\mathbf{p}^1$ is then projected onto the second hyperplane and so forth. Contrary to Cimmino's algorithm, the iterates produced by Kaczmarz's algorithm will always adhere to at least one of the constraints given by the rows of the matrix $\mathbf{G}$. Although the convergence appears to be a lot faster for this simple example, it can be extremely slow for larger problems in higher dimensions. The algorithm will therefore only be used for a certain class of subproblems that occur during the projection step of Cimmino's algorithm

While Cimmino's and Kaczmarz's algorithm both focus on the system of linear constraints, the boundary conditions are not considered. In order to ensure that the boundary conditions are met, we will use Hildreth's algorithm. The boundary conditions are given by

$$\mathbf{l} \leq \mathbf{p} \leq \mathbf{u}.$$

Hildreth's algorithm (Hildreth 1957) (in combination with Cimmino's algorithm) offers a way of considering the boundary constraints. This algorithm has been used by Harrigan and Buchanan (1984) and shown to work well for solving the problem at hand with inequality constraints such as the ones presented in this paper.

Algorithm 3 *(Hildreth's Algorithm for Box Constraints)* Let $\mathbf{p}^k$ be the current iterate and let $\mathbf{l}, \mathbf{u} \in \mathbb{R}^N$ be the lower and upper boundaries for the box constraints such that

$$\mathbf{l} \leq \mathbf{p}^k \leq \mathbf{u}$$

is required. Then, Hildreth's algorithm is given by the following iteration:

Step 1 For each element $p_n^k, l_n, u_n, n = 1, \ldots, N$ of the vectors $\mathbf{p}^k, \mathbf{l}, \mathbf{u}$ define

$$\hat{p}_n^{k+1} = \begin{cases} l_n & \text{if } p_n^k \leq l_n \\ p_n^k & \text{if } l_n \leq p_n^k \leq u_n \\ u_n & \text{if } p_n^k \geq u_n \end{cases}. \tag{9}$$

Step 2 Set $p^{k+1} = p^k + (\hat{p}^{k+1} - \hat{p}^k)$

4.3 Blocking the problem in subproblems

As described before, the blocking algorithm is aimed at splitting the system of linear constraints given by $\mathbf{G}$ and $\mathbf{c}$ into blocks of hyperplanes that are orthogonal to one another. The aim is to use Kaczmarz's algorithm on each of these blocks to obtain an exact solution for each of the subproblems within a very short time. The constraints matrix $\mathbf{G}$ features the so-called balancing constraints, which are represented by densely populated, near-orthogonal rows in $\mathbf{G}$, and other constraints which are sparsely populated. Additionally, the number of other constraints usually greatly outnumbers the balancing constraints. Since the balancing constraints present a near-orthogonal constraints set, they will be bundled in a single block $\mathbf{G}_1$.

For the remainder of the matrix, we propose to exploit the sparsity of the matrix to build blocks of orthogonal rows. The concept of matrix splitting is relatively straight forward. The splitting algorithm successively loops over all rows of $\mathbf{G}$ that are not assigned to $\mathbf{G}_1$, performing the following steps.

1. Test the current row $\mathbf{g}_m$ against all rows that have previously been assigned to the B blocks $\mathbf{G}_b$, $b-1,\ldots,B$. If a block is found in which $\mathbf{g}_m$ is orthogonal to all previously assigned rows, add $\mathbf{g}_m$ to this block.
2. If in each existing block $\mathbf{G}_b$ there is at least one row that is *not* orthogonal to $\mathbf{g}_m$, then open a new block $\mathbf{G}_{B+1}$ and assign $\mathbf{g}_m$ to it as its first row.

The vector $\mathbf{c}$ is split accordingly into vectors $\mathbf{c}_b$, $b=1,\ldots,B$.

The condition to detect orthogonality is: two rows of $\mathbf{G}$ are considered orthogonal if they do not share any nonzero coefficients in the same columns. This condition is stricter than the general concept of orthogonality, but it is numerically easy to achieve and proved sufficient for this application. This procedure will generate a set of orthogonal blocks $\mathbf{G}_b$ on which Kaczmarz's algorithm can be applied to find an exact solution in a finite number of steps.

The aim of blocking the matrix $\mathbf{G}$ is to reduce the computational efforts and storage requirements when implementing the final algorithm.

Definition 2 *(Block Projection)* The orthogonal projection of a vector $\mathbf{p}$ onto the subspace of hyperplanes defined by the rows of a block $\mathbf{G}_b$ is called a *block projection*. A block projection will be referred to as $\mathbf{p}_{p,b}$ (unlike the projection over a onto a single hyperplane, which was defined as $\mathbf{p}_p$ in Cimmino's algorithm).

For a block projection $\mathbf{p}_b$, the equation $\mathbf{G}_b\mathbf{p}_{p,b}=\mathbf{c}_b$ holds. Hence, $\mathbf{p}_{p,b}$ lies within the intersection of all hyperplanes defined by $\mathbf{G}_b$ and $\mathbf{c}_b$.

Remark 3 Kaczmarz's Algorithm 2 delivers this orthogonal projection for each orthogonal blocks $\mathbf{G}_b$, $b=2,\ldots,B$ in one iteration. Note that $\mathbf{G}_1$ is not generally orthogonal as it contains the balancing constraints.

4.4 Accelerating Cimmino's algorithm

Cimmino's algorithm relies on a step length, defined by the parameter α. Scolnik et al. (2000) proposed a technique to calculate the ideal step length in order to minimise the distance of the new iterate and the final solution. Using the direction

$$\mathbf{d}^k = \mathbf{p}_b^k - \mathbf{p}^k$$

of the kth iteration, Scolnik's approach is based on the idea that the line $f(\alpha)$ given by step 3 of Cimmino's algorithm

$$f(\alpha_k) = \mathbf{p}^k + \alpha_k \mathbf{d}^k$$

has a unique point at which the defect δ^{k+1} is minimal. Hence, Scolnik's line search approach is

$$\min_{\alpha_k} \left\| f(\alpha_k) - \mathbf{p}^* \right\|. \tag{10}$$

Let α^* be the solution to Eq. 10. Then, the next iterate $\mathbf{p}^{k+1}$ is given by

$$\mathbf{p}^{k+1} = \mathbf{p}^k + \alpha_k^* \mathbf{d}^k = \mathbf{p}^k + \alpha_k^* (\mathbf{p}_b^k - \mathbf{p}^k).$$

Figure 3 shows the concept of Scolnik's acceleration.

Theorem 1 (Solution for Scolnik's Line Search) *Scolnik's line search problem given in Eq. 10 has a unique solution given by*

$$\alpha_k^* = \sum_{b=1}^{B} \frac{w_b \|\mathbf{p}_{p,b}^k - \mathbf{p}^k\|^2}{\|\mathbf{p}_b^k - \mathbf{p}^k\|^2} = \sum_{b=1}^{B} \frac{w_b \|\mathbf{d}_b^k\|^2}{\|\mathbf{d}^k\|^2}.$$

The vector $\mathbf{p}_{p,b}^k$ denotes the projection of $\mathbf{p}^k$ over the block $\mathbf{G}_b$, w_b is the corresponding block weight, and B is the total number of blocks.

Proof See Scolnik et al. (2000). □

4.5 The accelerated Cimmino algorithm (ACIM)

Using the concepts introduced in this section, the final algorithm for the reconciliation of large contingency tables can be stated. It is assumed that the problem is available in the form

$$\min_{\mathbf{p} \in \mathbb{R}^N} (\mathbf{p} - \mathbf{p}^0)^T (\mathbf{p} - \mathbf{p}^0) \qquad \text{subject to} \qquad \mathbf{G}\mathbf{p} = \mathbf{c} \text{ and } \mathbf{l} \leq \mathbf{p} \leq \mathbf{u}.$$

Algorithm 4 *(ACIM: Main Algorithm for Parallel MRIO Reconciliation)* Assume that $\|\mathbf{g}_m\| = 1$ for all $m = 1, \ldots, M$.

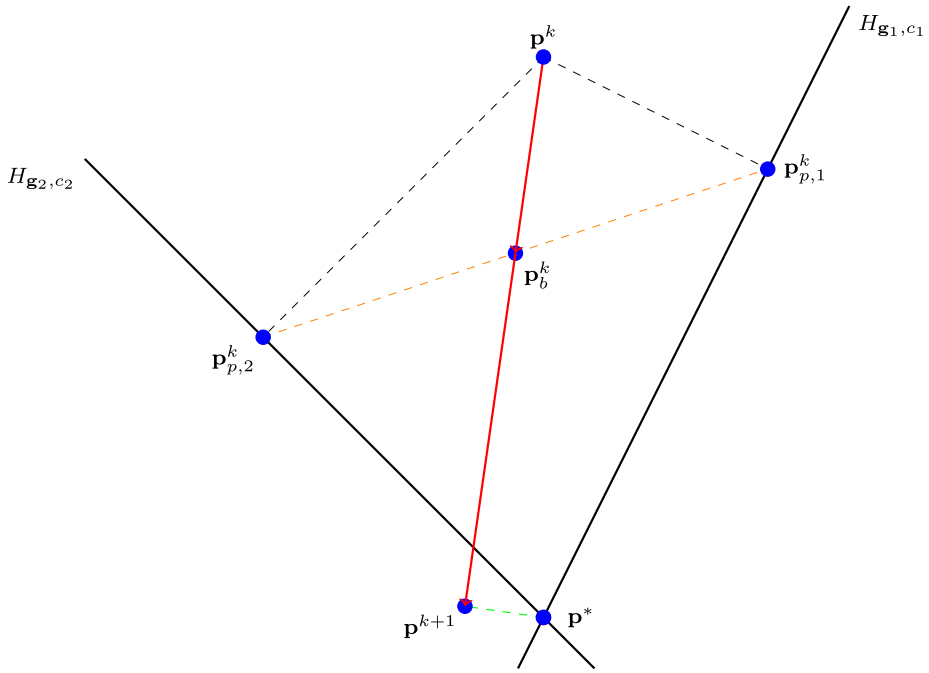

Fig. 3 Visualisation of Scolnik's acceleration. The optimal line search parameter α^* yields the new iterate $\mathbf{p}^{k+1}$. Due to the construction of Scolnik's acceleration, the vector $(\mathbf{p}^{k+1} - \mathbf{p}^k)$ is perpendicular to the vector $(\mathbf{p}^{k+1} - \mathbf{p}^*)$ (indicated by the green dashed line)

ACIM combines Cimmino's Algorithm and the acceleration measures presented in this section into a single algorithm. The ACIM's workflow is

1. Block the matrix $\mathbf{G}$ into the balancing block $\mathbf{G}_1$ and orthogonal blocks $\mathbf{G}_b$, $b = 2, \ldots, B$. Define the first iterate (this is the vectorization of the raw data table).
2. Apply Hildreth's Algorithm to the iterate to obtain a vector that is located within the given boundaries.
3. Perform one step of Cimmino's Algorithm by projection the current iterate onto the B blocks. Use Kaczmarz's Algorithm to calculate the individual blocks projections.
4. Calculate the barycentre from the B different block projections and calculate the direction vector.
5. Use Scolnik's acceleration to find the ideal steps length for the given direction and calculate the next iterate. Step back to Step 2 to start the next iteration.

A full mathematical description of the ACIM algorithm can be found in "Appendix".
The ACIM algorithm generally alternates between two different calculations:

Step 1 Ensuring that the boundary conditions $\mathbf{l} \leq \mathbf{p} \leq \mathbf{u}$ are adhered to. This is achieved by performing Hildreth's algorithm.

Step 2 Creating a sequence of iterates that converges towards a solution of $\mathbf{Gp} = \mathbf{c}$. This is achieved by performing Cimmino's algorithm on the matrix blocks $\mathbf{G}_b$ and by using Scolnik's acceleration.

Both of these steps are necessary to find a solution for the original problem given in Eq. (4). However, the sequence of iterates calculated using the ACIM algorithm may not cause a smooth convergence to the solution of the problem. The reason is that a vector that solves the constraints equation $\mathbf{Gp} = \mathbf{c}$ may not fulfil the boundary condition $\mathbf{l} \leq \mathbf{p} \leq \mathbf{u}$. Adjusting a vector $\mathbf{p}^k$ to the bounds condition using Hildreth's algorithm maybe therefore make a previously feasible solution for $\mathbf{Gp} = \mathbf{c}$ unfeasible. The ACIM algorithm will find a solution for the problem given in Eq. (4) if such a solution exists. However, it might be useful not to alternate the use of Cimmino's algorithm (including accelerations) and Hildreth's algorithm in *every* iteration of the ACIM algorithm. Instead, Cimmino's algorithm can be used to find a solution for $\mathbf{Gp} = \mathbf{c}$, followed by an adjustment of this solution to the condition $\mathbf{l} \leq \mathbf{p} \leq \mathbf{u}$ by Hildreth's algorithm. If this adjusted solution does not solve $\mathbf{Gp} = \mathbf{c}$, the Cimmino algorithm will be restarted.

This concept results in the performance of several iterations of Cimmino's algorithm between two iterations of Hildreth's algorithm. This technique will be used in the following section. We will refer to this concept as *a certain number of Cimmino iterations per Hildreth iteration*.

5 Implementation and results

The ACIM algorithm 4 was programmed in parallel in C, using OpenMP for the parallelisation. The implemented version of ACIM will be referred to as the ACIM code.

5.1 Parallelisation of the ACIM code

Within the ACIM algorithm, two tasks can be parallelised.

- The *block projections*. The calculation of the block projection over each block $\mathbf{G}_b$ yielding a vector $\mathbf{p}_{p,b}^k$ for each block.
- The *computation of the barycentre* $\mathbf{p}_b^k = \sum_{b=1}^{B} w_b \mathbf{p}_{p,b}^k$.

Implementing these steps in parallel will have significant consequences for the memory requirement of the ACIM code, as all block projections $\mathbf{p}_{p,b}^k$ must be stored in parallel. Calculating the block projections and the barycentre in one step is theoretically possible, but a racing condition would be met if different threads attempt to add values to the memory allocated for the barycentre vector $\mathbf{p}_b^k$. It is possible to avoid data racing by forcing parallel threads to write into a certain location in memory sequentially. This option was explored for the ACIM code in order to save memory, but it was found that the performance gain obtained from the parallelisation was lost due to this measure. Hence, different block projections must be saved in memory at the same time. This impacts the system design for the parallelisation.

Since a potentially large number of block projections $\mathbf{p}_{p,b}^k$ must be added up during the calculation of the barycentre, a shared-memory architecture was chosen for the parallelisation of the ACIM code. A high-performance computing system with 12 hyperthreaded 3.4 GHz cores and 296 GB of fully shared memory is available for the execution of the ACIM code.

5.2 ACIM code: parallelisation of the block projections

For the Eora model, the blocking algorithm generates more blocks $\mathbf{G}_b$ than processors available. The calculations of the block projections are arranged within a for-loop which is parallelised using OpenMP. The fact that there are more blocks than available computing cores has significant implications on the memory management. Limiting the number of threads results in each thread getting assigned a certain amount of block projections, which are calculated by the thread sequentially. Hence, it is possible to sum up the block projections calculated by one thread within a single variable *without* risking data racing.

The sum of all block projections that were calculated by a single thread will be referred to as a *thread projection* $\mathbf{p}^{k,t}$. The index t indicates which thread the thread projections were calculated by. The total number of threads is given by the variable T. Hence, $t = 1, \ldots, T$.

The ACIM code only requires to save as many vectors in memory as there are threads, as opposed to saving as many vectors in memory as there are blocks. For the Eora model, each block projection requires about 4 GB of memory. If 24 threads calculate the block projections in parallel, around 100 GB of memory is occupied by the 24 thread projections.

5.3 ACIM code: parallelisation of the barycentre calculation

The thread projections were programmed to already contain the weighted sums of the individual block projections, and the barycentre $\mathbf{p}_b^k$ is given by

$$\mathbf{p}_b^k = \sum_{t=1}^{T} \mathbf{p}_p^{k,t}. \tag{11}$$

Depending on the size of the MRIO table, the vectors $\mathbf{p}_p^{k,t}$ can be very large. Additionally, depending on the system resources, a large number of thread projections may exist at the end of the execution of the parallel code region. Since in C the summation of different vectors must be programmed element-wise, a large number of individual element-by-element summations may have to be executed. Parallelisation can be used as an attempt to speed up this process. However, unlike the parallel block projections, the parallelisation of the sum given in Eq. (11) can involve the risk of creating a racing condition, if programmed inappropriately. In order to avoid a racing condition, each thread adds the elements of all thread projections that lie in the same row of the vector, to the corresponding element of the barycentre. Figure 4 illustrates the parallelised, element-wise summation. The workflow of the parallelised ACIM code is shown in Fig. 5.

5.4 Numerical testing: using the ACIM code for the Eora Model

This section documents the use of the ACIM code for the reconciliation of the latest version of the Eora model for the year 2000 Lenzen et al. (2012, 2013). The reconciliation data set for the Eora model has the following dimensions.

Number of elements in the iterate:	557,041,584
Number of constraints:	3,517,796
Number of nonzero elements in matrix $\mathbf{G}$:	5,334,387,811
Total size of input data files:	80.2 GB

$$
\begin{aligned}
p^k_{b,1} &= p^{k,1}_{p,1} + p^{k,2}_{p,1} + \ldots + p^{k,T}_{p,1} && \leftarrow \text{calculated by thread 1}\\
p^k_{b,2} &= p^{k,1}_{p,2} + p^{k,2}_{p,2} + \ldots + p^{k,T}_{p,2} && \leftarrow \text{calculated by thread 3}\\
p^k_{b,3} &= p^{k,1}_{p,3} + p^{k,2}_{p,3} + \ldots + p^{k,T}_{p,3} && \leftarrow \text{calculated by thread 2}\\
p^k_{b,4} &= p^{k,1}_{p,4} + p^{k,2}_{p,4} + \ldots + p^{k,T}_{p,4} && \leftarrow \text{calculated by thread 3}\\
&\vdots \qquad\qquad \vdots\\
p^k_{b,N} &= p^{k,1}_{p,N} + p^{k,1}_{p,N} + \ldots + p^{k,T}_{p,N} && \leftarrow \text{calculated by thread 1}
\end{aligned}
$$

Fig. 4 This figure visualises the parallel summation of the thread projections in order to obtain the barycentre. In this example, three threads are randomly assigned to calculate the entire sum of all corresponding values $p^{k,t}_{p,n}, t = 1, \ldots, T$ to obtain $p^k_{b,n}$. A racing condition is avoided since each thread operates on its own location in memory when saving $p^k_{b,n}$

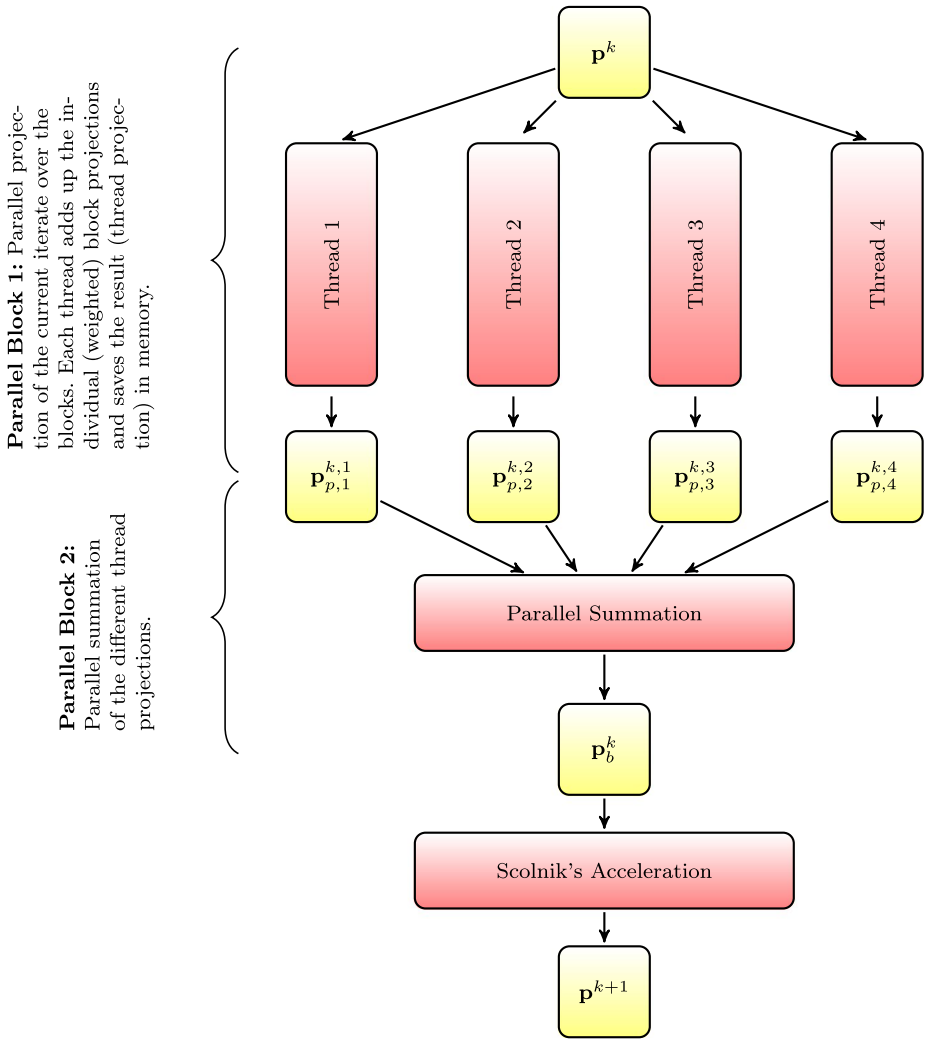

Fig. 5 Parallelisation of one iteration of the ACIM code. The red elements refer to mathematical manipulations, and the yellow elements represent variables. Parallel Block 1 calculates the thread projections in parallel. In Parallel Block 2, the individual thread projections are added up in parallel to form the barycentre

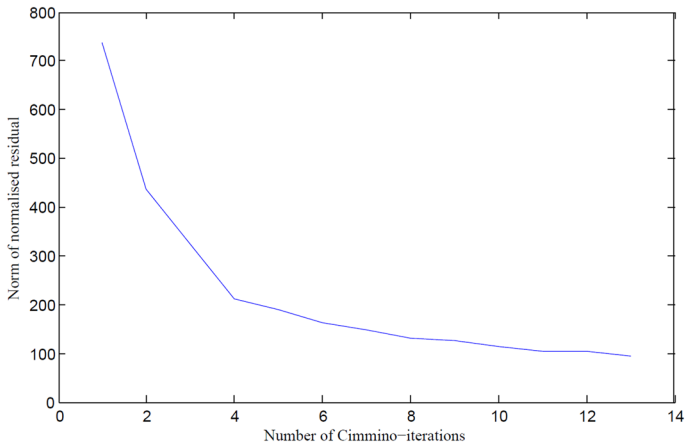

Fig. 6 This plot visualises the residual norms of the ACIM code for the Eora problem. Only Cimmino iterations were carried out, no Hildreth iterations. The code terminates after only 13 iterations

The algorithm was executed on a custom-built high-performance computing cluster with 12 hyperthreaded 3.4 GHz cores, offering a total of 24 threads and 296 GB of fully shared RAM. The ACIM code was executed using 24 threads. The maximum RAM usage was approximately 170 GB, and the code split the matrix **G** into 242 blocks. The termination criterion for the Cimmino iterations was reached once the norm of the normalised residual was below 100. The total runtime of the ACIM code required in order to reconcile the Eora MRIO was just under 3 h.

Figure 6 shows the development of the residual norm for the Eora model for Cimmino iterations only (no boundary adjustment by Hildreth iterations). Despite the size of the problem, the residual norm decreases at a very high rate, especially during the first iterations, and reaches the termination criterion after only 13 iterations.

The next step is to introduce Hildreth iterations to the calculations (Fig. 7). The plot shows an oscillating behaviour of the residual norms over the iterations. This is once again due to the fact that once the Cimmino iterations have found a feasible solution and terminate, a Hildreth step is performed in order to ensure that the boundary conditions are adhered to. The resulting iterate causes a residual norm that is above the termination criterion, and hence, Cimmino iterations were restarted. Figure 8 shows only the residual norms obtained from Hildreth iterations. Each iterate belonging to these residual norms adheres to the boundary conditions. The iterates converge smoothly towards a feasible solution, and the ACIM code terminates.

Table 1 shows the minimum runtime for three different sections of the ACIM code within one iteration.

It is noteworthy that the calculation of the balancing constraints requires a large portion of the entire runtime for the calculation of the block projections. In order to speed up the ACIM code in the future, most of the attention should be directed to speeding up the balancing process.

Also, Table 1 shows that the ACIM code finishes the balancing task for the entire Eora model in less than 80 s (if no bounds are defined at Hildreth's algorithm does not have to be used). This is a significant improvement in performance compared to earlier methods such as RAS.

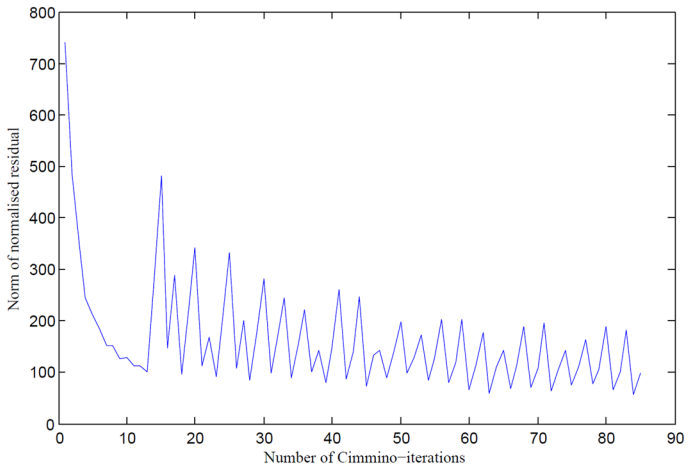

Fig. 7 This plot visualises the residual norms of the ACIM code for the Eora problem. A maximum of 100 Cimmino iterations were allowed between two Hildreth iterations. Both the Cimmino- and the Hildreth iterations were terminated when a residual norm of less than 100 was achieved

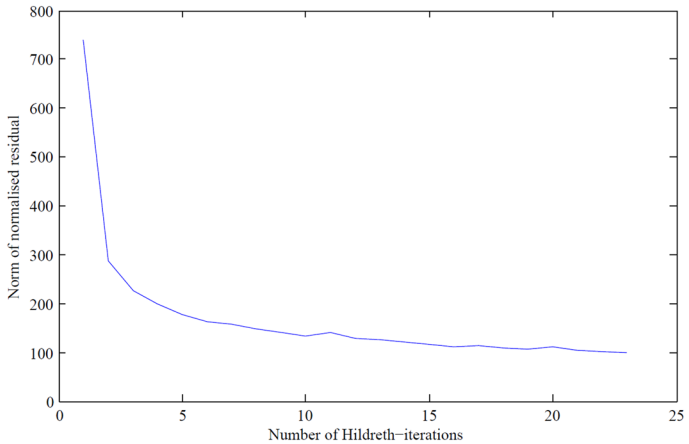

Fig. 8 This plot visualises the residual norms of the Hildreth iterations of the ACIM code, for the Eora problem. Each of the residual norms belongs to an iterate adhering to the boundary conditions

Table 1 Runtime results of the ACIM code for the Eora model

Code section	Duration (s)
Block projections (all orthogonal blocks)	80.74
Balancing constraints	73.98
Parallel summation	5.28

6 Conclusions and outlook

We have presented a parallelised least-squares-type algorithm (ACIM) for the reconciliation of large MRIO databases. ACIM uses the combination of mathematical acceleration measures and parallel programming executed on high-performance computing hardware to achieve a acceptable runtimes. To our knowledge, the ACIM algorithm is currently the only least-squares algorithm that is capable of reconciling global MRIOs of the size of the Eora database with satisfactory accuracy and runtime.

Future research will focus on enhancing the speed and accuracy as well as the numerical robustness of the ACIM algorithm further. Similar to current version of the ACIM algorithm, this will be achieved by a combination of introducing further computational measures and mathematical features. As a first step, the parallelisation of the code will be optimised in order to improve the load balancing and ultimately the overall performance of the ACIM code. The introduction of libraries such as Intel's Math Kernel Library[2] (MKL), the Basic Linear Algebra Subprograms[3] (BLAS) or CHOLMOD[4] to the code, will reduce the runtime further. Additional mathematical features will include the introduction of additional, more efficient accelerations as well as more efficient use of the geometric structure of the different constraints given in matrix **G**.

Authors' contributions

AG and JU wrote the algorithm. JU developed the mathematical framework for the algorithm. AG tested and analysed the algorithm and wrote the paper. ML provided the ground work for the development of the algorithm. KK and DDM ensured the flawless deployment on the high-performance computing systems at the University of Sydney, and ensured the operation of the Eora/algorithm interface. All authors read and approved the final manuscript.

Author details

[1] Integrated Sustainability Analysis, School of Physics, University of Sydney, Sydney, Australia. [2] School of Information Technology, Faculty of Science, Engineering and Built Environment, Deakin University, Melbourne, Australia. [3] Research Institute for Humanity and Nature, Kyoto, Japan. [4] Industrial Ecology Program, Department of Energy and Process Technology, Norwegian University of Science and Technology, Trondheim, Norway.

Acknowledgements

This work was supported by the Australian Research Council through its Discovery Projects DP0985522 and DP130101293, and Linkage Infrastructure, Equipment and Facilities Grant LE160100066, and the National eResearch Collaboration Tools and Resources project (NeCTAR) through its Industrial Ecology Virtual Laboratory VL201. The Eora database is available under www.worldmrio.com. Sebastian Jursazek expertly managed the advanced computation requirements, and Charlotte Jarabak from the University of Sydney's SciTec Library collected data.

Competing interest

The authors declare that they have no competing interests.

[2] For information on MKL, see http://software.intel.com/en-us/articles/intel-mkl/.

[3] For information on BLAS, see http://www.netlib.org/blas/.

[4] For information on CHOLMOD, see http://www.cise.ufl.edu/research/sparse/cholmod/.

Appendix: The ACIM algorithm

The variable names used in Algorithm 4 are identical to those introduced throughout this paper. For the reader's convenience, a summary of the variable names used within the Main Algorithm is stated before the algorithm.

Dimensions and counters

k Iteration counter.
N Length of vector $\mathbf{p}$; number of elements in the MRIO plus the number of soft constraints.
n Counter for N-dimensional objects; $1 \leq n \leq N$.
M Total number of constraints.
m Counter for m-dimensional objects; $1 \leq m \leq M$.
B Total number of blocks. This variable grows as the blocks are generated during the blocking algorithm. B remains unchanged once the total number of blocks has been calculated.
b Counter for the blocks; $1 \leq b \leq B$.
M_b Number of rows in block b. The block $\mathbf{G}_b$ has the dimension $\mathbb{R}^{M_b \times N}$.

Scalars, vectors and matrices

$\mathbf{p}^k$ Current iterate in iteration step k. The vector $\mathbf{p}^k$ has the length N.
$\mathbf{p}^k_{p,b}$ Projection of the iterate $\mathbf{p}^k$ onto the bth block $\mathbf{G}_b$.
$\mathbf{p}^k_p$ Barycentre of all block projections in the kth iteration step.
$\mathbf{G}$ Constraint matrix of the problem.
$\mathbf{G}_b$ bth block of $\mathbf{G}$.
$\mathbf{l}$ Vector of lower boundaries; $\mathbf{l} \in \mathbb{R}^N$.
$\mathbf{u}$ Vector of lower boundaries; $\mathbf{u} \in \mathbb{R}^N$.
$\mathbf{d}^k$ Search direction of the kth iteration.
$\mathbf{d}^k_b$ Block direction of the bth block in the kth iteration. $\mathbf{d}^k_b = \mathbf{p}^k_{p,b} - \mathbf{p}^k$.
α^*_k Optimal step length for Scolnik's line search.

Algorithm 5 *(ACIM: Main Algorithm for Parallel MRIO Reconciliation)*

Step 0. **Preparation:**
Define $\alpha > 0$, $\varepsilon > 0$, *and* w_m, $m = 1, \ldots, M$ *with* $\sum_{m=1}^{M} w_m = 1$. *Assign the initial estimate* $\mathbf{p}^0$. *Initialise* $k = 0$.

Step 1. **Perform Hildreth's Algorithm 3**
Update the individual elements p_n^{k+1} *of* $\mathbf{p}^{k+1}$ *according to the box constraints defined by the lower and upper boundary vectors* $\mathbf{l}$ *and* $\mathbf{u}$ *as follows.*

$$p_n^{k+1} = \begin{cases} l_n & \text{if } p_n^k \leq l_n \\ p_n^{k+1} & \text{if } l_n \leq p_n^{k+1} \leq u_n \\ u_n & \text{if } p_n^k \geq u_n \end{cases} .$$

Set $k_{hil} = k_{hil} + 1$.

Step 2. **If** $\|\mathbf{G}\mathbf{p}^{k+1} - \mathbf{c}\| = 0$,
A solution has been found. Set $\mathbf{p}^* = \mathbf{p}^k$ *and terminate the algorithm.*
End If

Step 3. **Block Projections (Cimmino's Algorithm):**

Step 3.1. *Project* $\mathbf{p}^k$ *onto the block of balancing constraints* $\mathbf{G}_1$ *using* **Kaczmarz's Algorithm 2**, *to obtain* $\mathbf{p}_{p,1}^k$.
Let M_1 *be the total amount of rows assigned to the balancing constraints matrix* $\mathbf{G}_1$. *Let* $\mathbf{g}_{1,m}$ *be the* m*-th row of* $\mathbf{G}_1$ *and let* c_m^1 *be the* m*-th element of* $\mathbf{c}^1$. *Set the iteration counter* $h = 1$, $m = 1$ *and* $\mathbf{p}_{p,1}^m = \mathbf{p}^k$.

Step 3.1.1. *Calculate*

$$\mathbf{p}_{p,1}^{m+1} = \mathbf{p}_{p,1}^m + c_m^1 - \langle \mathbf{g}_m, \mathbf{p}_{p,1}^{m+1} \rangle \mathbf{g}_m .$$

Step 3.1.2. **If** $m = M_1$,
If $\|\mathbf{G}_1 \mathbf{p}_{p,1}^{M_1} - \mathbf{c}^1\| < \varepsilon_{kacz}$
STOP. The block projection $\mathbf{p}_{p,1}^k$ *over the balancing constraints is given by* $\mathbf{p}_{p,1}^{M_1}$.
Else
Restart Kaczmarz's Algorithm: Set $h = h + 1$, $m = 1$ *and* $\mathbf{p}_{p,1}^m = \mathbf{p}_{p,1}^{M_1}$ *and return to step* ***Step 3.1.1.***
End If
Else
Set $m = m + 1$ *and return to step* ***Step 3.1.1.***.
End If

Step 3.2. *For* $b = 2, \ldots, B$, *project* $\mathbf{p}^k$ *onto each block* $\mathbf{G}_b$ *using one iteration of* **Kaczmarz's Algorithm 2**:
Let M_b *be the total amount of rows assigned to* $\mathbf{G}_b$. *Let* $\mathbf{g}_{b,m}$ *be the* m*-th row of* $\mathbf{G}_b$ *and let* c_m^b *be the* m*-th element of* $\mathbf{c}^b$. *Set* $m = 1$ *and* $\mathbf{p}_{p,b}^m = \mathbf{p}^k$.

Step 3.2.1. *Calculate*

$$\mathbf{p}_{p,b}^{m+1} = \mathbf{p}_{p,b}^m + c_m^b - \langle \mathbf{g}_m, \mathbf{p}_{p,b}^{m+1} \rangle \mathbf{g}_m .$$

Step 3.2.2. **If** $m = M_b$
STOP. The block projection $\mathbf{p}_{p,b}^k$ *for the block* $\mathbf{G}_b$ *is given by* $\mathbf{p}_{p,b}^{M_b}$.
Else
Set $m = m + 1$ *and return to step* ***Step 3.2.1.***.
End If

Set $k_{cim} = k_{cim} + 1$.

Step 4. **Compute the barycentre of all projections**

$$\mathbf{p}_b^k = \sum_{b=1}^{B} \frac{1}{B} \mathbf{p}_{p,b}^k \qquad \text{where } B \text{ is the total number of blocks,}$$

and the search direction

$$\mathbf{d}^k = \mathbf{p}_b^k - \mathbf{p}^k .$$

Step 5. **Perform Scolnik's Acceleration**

Step 5.1. *Calculate the block directions*

$$\mathbf{d}_b^k = \mathbf{p}_{p,b}^k - \mathbf{p}^k \qquad \text{for } b = 1, \ldots, B \ .$$

Step 5.2. *Calculate the ideal step length* α_k^*

$$\alpha_k^* = \sum_{b=1}^{B} \frac{\|\mathbf{d}_b^k\|^2}{B\|\mathbf{d}^k\|^2} \ ,$$

and the new iterate

$$\mathbf{p}^{k+1} = \mathbf{p}^k + \alpha_k^* \mathbf{d}^k \ .$$

Step 6. **If** $\|\mathbf{G}\mathbf{p}^{k+1} - \mathbf{c}\| = 0$,
Set $\mathbf{p}^* = \mathbf{p}^k$ *and terminate the algorithm.*
Else
Set $k = k + 1$ *and start the next iteration by returning to Step 3.*
End If

Theorem 2 (Convergence of ACIM) *The Main Algorithm 4 converges towards the solution to the problem*

$$\min_{\mathbf{p}\in\mathbb{R}^N} (\mathbf{p} - \mathbf{p}^0)^T (\mathbf{p} - \mathbf{p}^0) \qquad \text{subject to} \qquad \mathbf{G}\mathbf{p} = \mathbf{c} \text{ and } \mathbf{l} \le \mathbf{p} \le \mathbf{u}. \tag{12}$$

Proof See Scolnik et al. (2000). □

Publisher's Note

Received: 8 May 2018 Accepted: 7 January 2019
Published online: 19 January 2019

References

Bacharach M (1970) Biproportional matrices and input–output change. Cambridge University Press, Cambridge
Byron R (1996) Diagnostic testing and sensitivity analysis in the construction of social accounting matrices. J R Stat Soc Ser A (Stat Soc) 159(1):133–148
Byron RP (1978) The estimation of large social account matrices. J R Stat Soc Ser A (Gen) 141(3):359–367
Canning P, Wang Z (2005) A flexible mathematical programming model to estimate interregional input–output accounts. J Reg Sci 45(3):539–563
Fischer G (2000) Lineare algebra. Vieweg Studium, Braunschweig
Gilchrist D, Louis LS (1999) Completing input–output tables using partial information, with an application to Canadian data. Econ Syst Res 11(2):185–193
Harrigan F, Buchanan I (1984) A quadratic programming approach to input–output estimation and simulation. J Reg Sci 24:339–358
Hildreth C (1957) A quadratic programming procedure. Naval Res Logist Q 4(1):79–85
Junius T, Oosterhaven J (2003) The solution of updating or regionalizing a matrix with both positive and negative entries. Econ Syst Res 15:87–96
Lahr ML, de Mesnard L (2004) Biproportional techniques in input–output analysis: table updating and structural analysis. Econ Syst Res 16(2):115–134
Lenzen M, Gallego B, Wood R (2010) Matrix balancing under conflicting information. Econ Syst Res 21:23–44

Lenzen M, Kanemoto K, Moran D, Geschke A (2012) Mapping the structure of the world economy. Environ Sci Technol 46(15):8374–8381
Lenzen M, Moran D, Kanemoto K, Geschke A (2013) Building EORA: a global multi-region input-output database at high country and sector resolution. Econ Syst Res 25(1):20–49
Mínguez R, Oosterhaven J, Escobedo F (2009) Cell-corrected RAS method (CRAS) for updating or regionalizing an input–output matrix. J Reg Sci 49:329–348
Morrison W, Thumann RG (1980) A Lagrangian multiplier approach to the solution of a special constrained matrix problem. J Reg Sci 20:279–292
Nagurney A, Eydeland A (1992) A splitting equilibration algorithm for the computation of large-scale constrained matrix problems: theoretical analysis and applications. In: Amman HM, Belsley DA, Pau LF (eds) Computational economics and econometrics, advanced studies in theoretical and applied econometrics, vol. 22. Springer, Netherlands, pp 65–105
Scolnik H, Echebest N, Guardarucci MT, Vacchino MC (2000) A class of optimized row projection methods for solving large nonsymmetric linear systems. Appl Numer Math 41(4):499–513
Stadler K, Wood R, Bulavskaya T, Södersten CJ, Simas M, Schmidt S, Usubiaga A, Acosta-Fernández J, Kuenen J, Bruckner M, Giljum S, Lutter S, Merciai S, Schmidt JH, Theurl MC, Plutzar C, Kastner T, Eisenmenger N, Erb KH, Koning A, Tukker A (2018) EXIOBASE 3: developing a time series of detailed environmentally extended multi-regional input–output tables. J Ind Ecol 22(3):502–515. https://doi.org/10.1111/jiec.12715
Stone R (1961) Input–output and national accounts. Organisation for European Economic Co-operation, Paris
Stone R, Champernowne DG, Meade JE (1942) The precision of national income estimates. Rev Econ Stud 9(2):111–125
Tukker A, Dietzenbacher E (2013) Global multiregional input–output frameworks: an introduction and outlook. Econ Syst Res 25(1):1–19
van der Ploeg F (1982) Reliability and the adjustment of sequences of large economic accounting matrices. J R Stat Soc Ser A 145(2):169–194
van der Ploeg F (1984) Generalized least squares methods for balancing large systems and tables of national accounts. Rev Public Data Use 12:17–33
van der Ploeg F (1988) Balancing large systems of national accounts. Comput Sci Econ Manag 1:31–39
Wood R, Hawkins TR, Hertwich EG, Tukker A (2014) Harmonising national input–output tables for consumption-based accounting—experiences from EXIOPOL. Econ Syst Res 26(4):387–409. https://doi.org/10.1080/09535314.2014.960913
Wood R, Stadler K, Bulavskaya T, Lutter S, Giljum S, de Koning A, Kuenen J, Schütz H, Acosta-Fernández J, Usubiaga A, Simas M, Ivanova O, Weinzettel J, Schmidt J, Merciai S, Tukker A (2015) Global sustainability accounting–developing EXIOBASE for multi-regional footprint analysis. Sustainability 7(1):138