Received: 29 November 2017 | Revised: 5 May 2019 | Accepted: 5 August 2019
DOI: 10.1002/nla.2268

RESEARCH ARTICLE

ECLES: A general method for local editing of parameters with linear constraints

Elisa de Cássia Silva Rodrigues[1] | **Jorge Stolfi**[2]

[1]Institute of Mathematics and Computing, Federal University of Itajubá, Itajubá, Brazil

[2]Institute of Computing, University of Campinas, São Paulo, Brazil

Correspondence
Elisa de Cássia Silva Rodrigues, Institute of Mathematics and Computing, Federal University of Itajubá, 1303 BPS Ave, Itajubá-MG 37500-903, Brazil.
Email: elisa.rodrigues@unifei.edu.br

Present Address
Elisa de Cássia Silva Rodrigues, Institute of Mathematics and Computing, Federal University of Itajubá, 1303 BPS Ave, Itajubá-MG 37500-903, Brazil.

Funding information
Brazilian government through CAPES PNPD/PPGInf/UFPR and CNPq, Grant/Award Number: 301016/92-5

Summary

We describe ECLES (*Editing by Constrained LEast Squares*), a general method for interactive local editing of objects that are defined by a list of parameters subject to a set of linear or affine constraints. The method is intended for situations where each edit action affects only a small set of the parameters; some of which (the "anchors") are to be set to new given values, whereas the rest (the "derived" ones) are to be modified so as to preserve the constraints. We use exact integer arithmetic in order to reliably determine solvability, to detect and eliminate redundancies in the set of constraints, and to ensure that the solution exactly satisfies the constraints. We also use constrained least squares to choose a suitable solution when the constraints allow multiple solutions. Unlike the usual finite element approach, the method allows editing of any set of anchors with any sufficiently large set of derived parameters. As an example of application, we show how the method can be used to edit smooth (C^1) deformations of geometric mesh models.

KEYWORDS

deformation, exact arithmetic, fraction-free, least squares, linear constraints, parameter editing

1 | INTRODUCTION

We consider here the general numerical problem of interactive local editing of parameters of some model or process, subject to linear or affine constraints. This problem occurs in many applications, such as construction and deformation of smooth splines (for terrain and computer-aided design (CAD) models, image morphing and registration, etc.), constraint-based drawing, and process control.[1–4]

We assume that the application has a finite vector $p = (p_1, p_2, \dots, p_n)$ of n real valued parameters that are subjected to a finite set of m affine equality constraints, that is, polynomial equations of degree 1, such as $3p_3 - 5p_8 = 7$. We also assume that, at each editing action, the user specifies new required values for some subset $\mathcal{A}$ of parameters (the *anchors*), and desirable but not required "hint" values for another subset $\mathcal{D}$ (the *derived* parameters). The editing software is then supposed to assign given values to the anchors, and automatically adjust the parameters in $\mathcal{D}$ so as to preserve the constraints while being as close as possible to the given hints.

The main contribution of this article is a general and robust method to perform such editing tasks, which we call ECLES (*Editing by Constrained LEast Squares*).[5,6] The ECLES algorithm uses linear system solving with exact integer arithmetic[7,8] to reliably detect and eliminate redundancies among the constraints that are relevant to each editing action and to ensure that the constraints are always exactly satisfied. It also uses constrained least squares[9] to minimize the discrepancy between the given hints and the final values chosen for the derived parameters.

An important feature of ECLES is that the sets $\mathcal{A}$ and $\mathcal{D}$ are not fixed but may be specified by the application at the start of each editing action. The cost of each editing action depends on the number $\widetilde{n} = |\mathcal{D}|$ of derived parameters and on the number $\widetilde{m}$ of constraints that refer to them and the anchors. Specifically, in the worst case, the computing time for each new choice of $\mathcal{A}$ and $\mathcal{D}$ grows proportionally to $\widetilde{m}^4\widetilde{n}$, rather than to $m^4 n$—as would be the case if the problem was solved for all n parameters and m constraints at once. Once $\mathcal{A}$ and $\mathcal{D}$ have been specified, a new solution for new anchor values and new hints for derived parameters can be computed in time proportional to $\widetilde{m}^3\widetilde{n}$, instead of $\widetilde{m}^4\widetilde{n}$.

These worst-case time bounds are so high due to the use of exact arithmetic in the analysis of redundant constraints, a choice that is justified in Section 3.3. The number of arithmetic operations actually grows proportionally to $\widetilde{m}^2\widetilde{n}$, as one would expect, but the bit size of the numbers grows proportional to $\widetilde{m}$, so the cost of each operation grows like $\widetilde{m}^2$. Nevertheless, the method remains fast enough to be used in practical editing applications, as shown in Section 7.1.

The constrained parameter editing problem is formally stated in Section 3. In Sections 4 and 5, we define the main tools used by ECLES, including fraction-free LDU matrix factoring, constrained least squares, removing redundant constraints, and checking for solvability. The ECLES algorithm proper is described in detail in Section 6. In Section 7, we briefly describe a practical application, namely, an interactive editor for 2D spline deformations of the plane. The Appendix gives additional details about the fraction-free LDU decomposition algorithm, as used in ECLES.

This article covers a part of the PhD thesis of the first author.[6] The application outlined in Section 7 was described with more details, also in a conference article.[10]

2 | RELATED WORK

There are many systems for interactive editing of objects under linear constraints.[11–15] Typical constraint-based interactive editors recompute all parameters at each user editing action and use penalty terms in order to minimize the changes to nonedited parameters.

Finite elements are well-known approach for the local editing of parameters with affine constraints. It entails precomputing a *finite element basis* of parameter change vectors that span the space of all possible changes allowed by the constraints and then giving the user a separate "knob" for each finite element. To limit the impact of each edit action, those change vectors are chosen to have small support, that is, to be nonzero only for a small number of parameters.

A typical example of this approach is the editing of splines with specified continuity order. The parameters in this case are the Bézier control points or control values that define each patch of the spline. Each finite element basis has exactly one user-adjustable value (the coefficient of the element, often presented as one of the Bézier parameters). The finite element determines the effect of that coefficient on the Bézier parameters of one or more patches. Because of the constraints, the number of adjustable "knobs" is usually much smaller than the number of parameters.

Finite elements basis for smooth splines have been extensively studied by L. Schumaker and many others.[16,17] One particular case of the finite element approach is the B-spline approach,[18] where there is only one editable control point for each patch of the spline and the resulting function is automatically continuous to the maximum possible order. However, B-splines are well defined only for meshes with the topology of a regular orthogonal grid (quadrangular or hexahedral).[19,20] Extending them to irregular and simplicial meshes is possible but rather complicated.[21]

A general limitation of the finite elements technique is that the element basis must be precomputed for the whole mesh and the impact of each "knob" is then fixed, that is, the set $\mathcal{D}$ is determined automatically by the set of "knobs" that are modified.

METAFONT is a font design system developed by Knuth in 1979[13] that can be viewed as a precursor of the constraint-based local parameter editing model. It is a programming language where function parameters may be subjected to linear and affine constraints. The language lets the user specify any subset of independent parameters, automatically solving for the others when enough parameters have been specified. However, METAFONT was not interactive.

Interactive parameter editors often must cope with user constraints that are underdeterminate, in which case they need additional criteria to choose the final values. METAFONT automatically chooses the set of derived parameters $\mathcal{D}$ so as to ensure a unique solution, adding parameters to it according to some ad-hoc rules—such as giving lower priority to parameters that were recently used as anchors $\mathcal{A}$ in previous commands.

Many **constraint-based illustration editors** are available.[11,12,14] An early example is the Juno editor developed by Nelson in 1985.[12,14] In Juno, the user could define (graphically or symbolically) geometric entities such as points, lines, and circles, specify geometric or algebraic constraints between them and give hints for any unknown coordinates or other parameters. These parameters then would be (re)computed automatically by Juno so as to satisfy the constraints, as the user edited selected points.

Those drawing editors typically solve for all constraints simultaneously by an iterative algorithm, using the current situation p as a starting guess. If the constraints are underdetermined, the solution is whatever the iterative solver converges to.

The **least squares** criterion has been used to resolve underconstrained problems. For example, in the work of Sorkine et al.,[15] it was used to globally approximate 3D triangular meshes to given data points.

ECLES is closest to the approach used in the work of Masuda et al.[22] for surface mesh editing with first-degree constraints. They also used the criterion of least squares and user-given hints to handle underconstrained situations. However, their method allows editing of a single control point at a time ($|\mathcal{A}| = 1$), and then recomputes all the remaining control points ($\mathcal{D} = \overline{\mathcal{A}}$). Our method differs from theirs by allowing the user or application to vary multiple anchors at the same time and to specify the set $\mathcal{D}$ of parameters that can be adjusted to satisfy the constraints. Moreover, we use exact arithmetic to reliably detect inconsistent or redundant constraints.

3 | STATEMENT OF THE PROBLEM

The problem that we consider in this article involves a column vector $p = (p_1, p_2, \dots, p_n)$ of n variables (the *parameters*) of an application, which are subject to a fixed set of m linear or affine equations (the *constraints*). These can be expressed as a matrix equation $Rp = q$, where R is a constant $m \times n$ coefficient matrix, and q is a column vector of m constants (possibly zero).

For example, a chemical process may have four pumps whose flow rates p_1, p_2, p_3, and p_4 must always satisfy $p_1 + p_2 = p_3$ and $p_4 = 2p_3 + 5$, that is,

$$\begin{bmatrix} 1 & 1 & -1 & 0 \\ 0 & 0 & -2 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \end{bmatrix}.$$

Any imposed change in one of the four rates, for example p_3, must then be simultaneously compensated by adjusting the rate p_4 (to satisfy the second constraint) and either p_1, or p_2, or both (to satisfy the first one). Someone, such as the designer or the operator of the system, is supposed to choose among these last three possibilities and, if both p_1 and p_2 are to be adjusted, also to somehow specify how to split the adjustment between them.

In general, let $\mathcal{N} = \{1, 2, \dots, n\}$ be the set of indices of all parameters. For each editing action, the user (or the application) must define two disjoint subsets of $\mathcal{N}$:

- $\mathcal{A}$ (*anchor set*): the indices of one or more parameters whose values will be deterministically set in this action;
- $\mathcal{D}$ (*derived set*): the indices of zero or more parameters that may be adjusted, if necessary, to satisfy the constraints.

We call the remaining indices $\mathcal{I} = \mathcal{N} \setminus (\mathcal{A} \cup \mathcal{D})$ of the parameters that should not be changed, the *fixed* or *invariant set*.

Furthermore, for each index s in $\mathcal{A} \cup \mathcal{D}$, the user must specify a new parameter value p'_s. That value is considered mandatory if s is in $\mathcal{A}$ but only a *hint* if s is in $\mathcal{D}$. The editing algorithm must then compute a new vector p'' of parameter values, that satisfies all constraints (1), and is such that p''_s is equal to p'_s if s is not in $\mathcal{D}$, and as close as possible to p'_s if s is in $\mathcal{D}$. (Note that p'_s is irrelevant if $s \in \mathcal{I}$.)

In many applications, the new anchor values and the hint values (p'_s) may be changed several times for a fixed choice of $\mathcal{A}$ and $\mathcal{D}$. For example, the anchors may be the coordinates of a control point on the screen that is being positioned by the user with the mouse. As detailed in Section 6, ECLES precomputes all working data that depends only on the sets $\mathcal{A}$ and $\mathcal{D}$, so that it can quickly recompute the new parameter vector p'' when p' changes—thus enabling fast feedback to the user, in this case.

3.1 | Multidimensional parameters

In some applications, each parameter p_s is itself a vector from some Cartesian space $\mathbb{R}^d$ and each constraint is an affine equation on vectors of $\mathbb{R}^d$ (see Figure 1). Then, the constraints can be expressed more economically as

$$RP = Q, \tag{1}$$

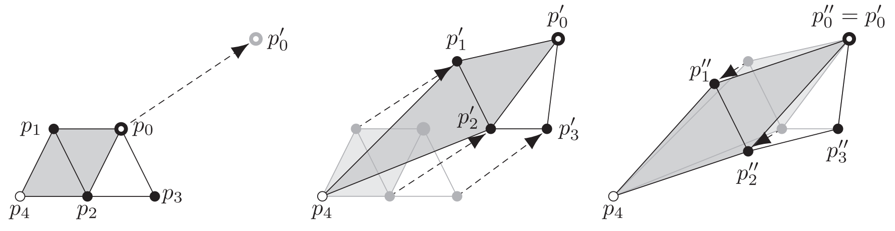

FIGURE 1 Example of a possible parameter editing action in a graphical editor. The object has five parameters, which are points of $\mathbb{R}^2$ (dots). There is one linear constraint, represented by the gray quadrilateral, that involves its four vertices: $p_1 - p_0 = p_4 - p_2$, meaning that the four points must always be the corners of a parallelogram. The anchor set is $\mathcal{A} = \{0\}$ (open bold dot), and the derived set is $\mathcal{D} = \{1, 2, 3\}$ (black dots). The left figure is the initial situation, before a user-specified change of the anchor point p_0 to p_0'. The center figure shows hints p_1', p_2', and p_3' for the derived points that could have been chosen automatically by the editor. The figure at the right shows the final positions p_1'', p_2'', and p_3'' of the derived points. Note that p_1 and p_2 had to be placed away from p_1' and p_2' in order to satisfy the constraint

where P and Q are matrices with d columns—specifically, with sizes $n \times d$ and $m \times d$. The effective number of parameters is then nd.

This is the case, in particular, of the application described in Section 7. In the 2D editing mode, each parameter is a Bézier control point ($d = 2$) of a spline from $\mathbb{R}^2$ to $\mathbb{R}^2$, and each constraint is one of the four-point Bézier conditions that determine C^1 continuity across the edges of the mesh.

We will need the following notation. If A is any matrix with r rows and s columns, $\mathcal{X}$ is any subset of $1, 2, \dots, r$, and $\mathcal{Y}$ is any subset of $1, 2, \dots s$, then $A_{\mathcal{X}}$ is the submatrix of A consisting of all the rows whose indices are in $\mathcal{X}$, and $A_{\mathcal{X}\mathcal{Y}}$ is the submatrix of $\mathcal{A}$ whose elements are all elements A_{ij} with $i \in \mathcal{X}$ and $j \in \mathcal{Y}$.

With this notation, the problem we address can be stated as follows: Given matrices R $(m \times n)$, P $(n \times d)$, and Q $(m \times d)$ that satisfy the constraint Equation (1) and an $n \times d$ matrix P', compute an $n \times d$ matrix P'' that also satisfies Equation (1), with $P''_{\mathcal{A}} = P'_{\mathcal{A}}$, $P''_{\mathcal{I}} = P_{\mathcal{I}}$, and $P''_{\mathcal{D}}$ as close as possible to $P'_{\mathcal{D}}$.

3.2 | Solvability conditions

The general problem defined in Section 3 may or may not have a solution.

We say that the problem is *strongly solvable* for a given choice of $\mathcal{A}$ and $\mathcal{D}$ if, for any current parameter vector p that satisfies all constraints and any assignment of new values p_s' for the indices s in $\mathcal{A}$, there is a solution vector p'', as defined previously. That is, the problem is strongly solvable if the set $\mathcal{D}$ of derived parameters is large enough to allow any combination of new values to be assigned to the control parameters in $\mathcal{A}$ and all constraints to be satisfied by assigning appropriate new values to all parameters in $\mathcal{D}$.

If the problem is not strongly solvable for a given choice of $\mathcal{A}$ and $\mathcal{D}$, it may still allow a solution p'' as long as the current values of the invariant parameters $p_{\mathcal{I}}$ and the new values $p'_{\mathcal{A}}$ specified for the anchors are favorable. We then say that the problem is *weakly solvable* for the current given choices of $\mathcal{A}$, $\mathcal{D}$, $p_{\mathcal{I}}$, and $p'_{\mathcal{A}}$.

Some applications (or some editing actions) may require strong solvability, whereas weak solvability may be sufficient in other cases. The ECLES method is designed to support both situations.

3.3 | The need for exact computation

The constraints imposed on the parameters may involve redundant equations. For example, we may have $p_1 + 2p_2 = 0$, $p_2 + p_3 = 5$, $2p_3 - p_1 = 10$. Note that the last equation is a linear combination of the first two.

Redundant equations usually cause problems when linear equation systems are solved with floating-point arithmetic, even when using algorithms that are numerically stable. Rounding errors during Gaussian elimination can easily turn a redundant equation into a spurious nonredundant constraint that forces absurd values to some variables or even makes

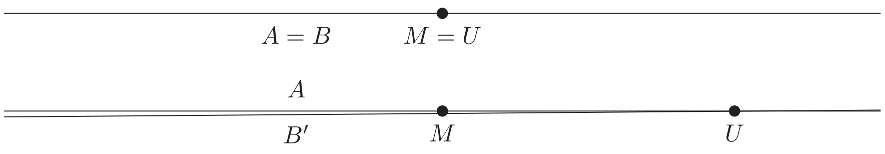

FIGURE 2 A simple hypothetical situation where redundant equations and rounding errors could cause ECLES to fail. Suppose that an unknown point U is constrained by two equations that require it to lie on both lines A and B, and otherwise as close as possible to the hint point M. If A and B are the same line (top), one of the two constraints is redundant; in that case, U can be any point on that line, in particular the given hint M. However, if the constraints are evaluated with floating-point arithmetic (bottom), the second constraint may become the equation of a slightly displaced line B'. In that case, U would have to be placed at the intersection of the two lines—which may be arbitrarily far away from M, or even at infinity

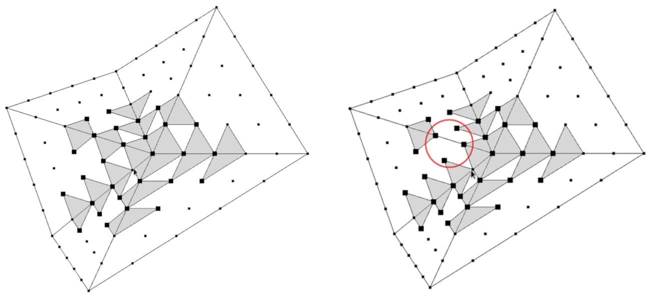

FIGURE 3 A realistic situation where redundant equations and rounding errors could cause ECLES to fail, based on the application of Section 7. The small dots are the Bézier control points of six simplicial polynomial patches of degree 5 in a spline deformation of the plane, with the six-triangle reference mesh shown. The derived points $\mathcal{D}$ are shown by the larger dots, and there is a single anchor $\mathcal{A}$, the control point just southwest of the central one. The gray quadrilaterals represent the relevant constraints that ensure C^1 continuity of the spline.[6,10] The figure at the left shows all the nonredundant constraints, as determined with exact arithmetic. The figure at the right is the set of constraints obtained when attempting to eliminate redundant constraints with floating-point arithmetic, treating matrix coefficients smaller than 10^{-12} as zero. One constraint (circled) was incorrectly assumed to be redundant and excluded from $\mathcal{E}$

the system unsolvable. Conversely, it may cause a nonredundant equation to be mistakenly identified as redundant and removed (see Figures 2 and 3).

Because of these risks, the ECLES algorithm uses exact arithmetic whenever constraints must be manipulated or used in logical decisions. Specifically, the elements of the constraint coefficient matrix R of system (1) must be integers, and those of the right-hand matrix Q and the parameter matrices P, P', and P'' must be rational. ECLES then uses exact arithmetic—specifically, the *fraction-free LDU factoring*, described in Section 4.1—to identify and discard the truly redundant equations.

If the application deals natively with floating-point numbers, the constraint coefficients and parameter values must be rounded to rational numbers with the appropriate accuracy, and then the former must reduced to integers by eliminating common denominators, before feeding them to ECLES. In that case, it is the application's problem to ensure that the resulting exact constraints are consistent and accurate for its purposes and that the initial parameter values satisfy them.

3.4 | Relevant equations and parameters

The constraints that involve parameters of $\mathcal{A}$ or $\mathcal{D}$ are called *relevant constraints*. All the other constraints, which involve only fixed parameters, will not affect or be affected by the editing action. Assuming that they are satisfied by the current parameter vector p, they will remain valid after the edit and can be ignored.

Let $\mathcal{E}$, a subset of $\mathcal{M} = \{1, 2, \ldots, m\}$, be the indices of the relevant constraints. We define the *fixed relevant* parameters, with index set $\mathcal{F} \subseteq \mathcal{I}$, as comprising the fixed parameters that occur in some relevant constraint (see Figure 4).

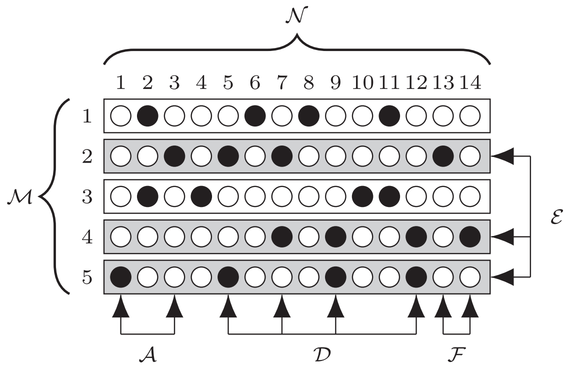

FIGURE 4 Example of the classification of $n = 14$ parameters $\mathcal{N}$ and $m = 5$ constraints $\mathcal{M}$. Each rectangle represents one constraint (one row of the matrix R). The black dots are the nonzero elements of R. Note that Equations 1 and 3 are not relevant because they do not involve any parameters of $\mathcal{A}$ or $\mathcal{D}$. Note also that the parameters 2, 4, 6, 8, 10, and 11 are fixed but not relevant (the set $\mathcal{I}$)

We can write a system that represents only the relevant constraints and unknowns of system (1):

$$SX = H \tag{2}$$

where

$$S = R_{\mathcal{E}\mathcal{D}}, \tag{3}$$

$$X = P''_{\mathcal{D}}, \text{ and} \tag{4}$$

$$H = Q_{\mathcal{E}} - R_{\mathcal{E}\mathcal{A}} P'_{\mathcal{A}} - R_{\mathcal{E}\mathcal{F}} P_{\mathcal{F}}. \tag{5}$$

Recall that $P'_{\mathcal{A}}$ (the given new values for the anchors), $P'_{\mathcal{D}}$ (the given hints for the derived parameters), and $P_{\mathcal{F}}$ (the current values of the fixed relevant parameters) are assumed given, so S and H are known matrices, and X is an unknown one. Moreover, as explained in Section 3.3, ECLES assumes that the constraint equations have integer coefficients and rational right-hand sides. Therefore, the elements of the matrix S of system (2) are integer, whereas those of H and X are rational.

As stated in the Introduction, we denote by $\widetilde{m} = |\mathcal{E}| \leq m$ the number of relevant constraints and by $\widetilde{n} = |\mathcal{D}| \leq n$ the number of derived parameters. Therefore, the sizes of S, X, and H are $\widetilde{m} \times \widetilde{n}$, $\widetilde{n} \times d$, and $\widetilde{m} \times d$, respectively.

Throughout this paper, computation costs of algorithms are expressed as a function of the sizes of the relevant equation system (such as $\widetilde{m}$, $\widetilde{n}$, and the rank r of $R_{\mathcal{E}\mathcal{D}}$), in the asymptotic worst-case sense, using D. Knuth's Θ notation.[23] For these analyses, we assume that all elements in the input matrices have a bounded size ($\Theta(1)$ bits) but allow for the growth of those numbers during the algorithms. As explained in Appendix, the numbers that arise during the algorithms will have $\Theta(r)$ bits each in the worst case. Multiplication of two r-bit numbers, by the straightforward method, takes time $\Theta(r^2)$. Thus, the asymptotic worst-case running time of each algorithm will be the number of arithmetic operations times $\Theta(r^2)$.

4 | EXACT LINEAR ALGEBRA

In this section, we describe the main exact linear algebra tools used in the ECLES method. Throughout this section, m and n are the row and column counts of the linear system that is given to these algorithms, and not the (usually larger) numbers of constraints and parameters of the original problem, as defined in Section 3.

4.1 | Fraction-free LDU factoring

To solve linear systems, ECLES uses the concept of *fraction-free LDU factoring* as developed in the works of Bareiss (1968)[24] and Turner (1995).[8] Originally, the concept was defined only for square or nonsingular matrices,[24–28] but it was recently extended in the work of Jeffrey[7] to arbitrary rectangular matrices of any rank. ECLES uses a modification of Jeffrey's algorithm, described in Appendix.

Let A be an $m \times n$ matrix of integers, with rank r. Its fraction-free LDU decomposition consists of five integer matrices: Π_{R} ($m \times m$, a row permutation matrix), L ($m \times r$), D ($r \times r$, diagonal), U ($r \times n$), and Π_{C} ($n \times n$, a column permutation matrix) such that

$$A = \Pi_{\mathsf{R}} L D^{-1} U \Pi_{\mathsf{C}}. \tag{6}$$

In practice, as usual, the matrices Π_{R} and Π_{C} can be represented as integer vectors to save space. The matrices L and U have specific structures:

$$L = \begin{pmatrix} \hat{L} \\ \tilde{L} \end{pmatrix} \quad \text{and} \quad U = \begin{pmatrix} \hat{U} & \tilde{U} \end{pmatrix}, \tag{7}$$

where $\hat{L}$ is an $r \times r$ lower triangular matrix, $\hat{U}$ is an $r \times r$ upper triangular matrix, both invertible, and $\tilde{L}$, $\tilde{U}$ are arbitrary integer matrices with sizes $(m-r) \times r$ and $r \times (n-r)$, respectively.

For example, consider the matrix A below, which has $m = 5$ rows, $n = 4$ columns, and rank $r = 3$. Its fraction-free LDU factoring could be

$$A = \begin{bmatrix} 4 & 9 & 16 & 29 \\ -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 5 & 6 & -1 & -12 \\ 5 & 10 & 15 & 20 \end{bmatrix} = \begin{bmatrix} 0&0&1&0&0 \\ 1&0&0&0&0 \\ 0&1&0&0&0 \\ 0&0&0&1&0 \\ 0&0&0&0&1 \end{bmatrix} \left[\begin{array}{ccc} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \\ \hdashline 5 & 24 & -164 \\ 5 & 20 & -120 \end{array}\right] \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix}^{-1} \left[\begin{array}{ccc:c} -1 & -6 & -16 & -19 \\ 0 & 1 & -3 & 4 \\ 0 & 0 & -80 & 0 \end{array}\right] \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix} \tag{8}$$

In the algorithms of Section 6, the fraction-free decomposition of a generic $m \times n$ matrix A or rank r are handled as a tuple $\mathbf{A} = (m, r, n, \Pi_{\mathsf{R}}, L, D, U, \Pi_{\mathsf{C}})$. Its elements are then denoted $\mathbf{A}.m$, $\mathbf{A}.r$, $\mathbf{A}.n$, $\mathbf{A}.\Pi_{\mathsf{R}}$, $\mathbf{A}.L$, $\mathbf{A}.D$, and so on.

Computation cost: In the worst case, the fraction-free LDU factoring can be computed with the `LinSys.LDUFactor` algorithm (detailed in Appendix), which, for a general $m \times n$ matrix of rank r, performs $\Theta(mrn)$ arithmetic operations. As detailed in that section, if the elements of the input matrix A have $\Theta(1)$ bits (i.e., bounded size independent of m and n), the numbers in the factor matrices L, D, and U will have $\Theta(r)$ bits. Therefore, the decomposition can be computed in $\Theta(mr^r n)$ time.

4.2 | Solving exact linear systems

The fraction-free LDU factorization can be used to compute an exact solution (with rational numbers) to a system of linear equations with integer coefficients. We assume in general that the linear system to be solved is

$$AX = B, \tag{9}$$

where A is a given $m \times n$ matrix of integers, X is an unknown $n \times d$ matrix and B is a known $m \times d$ matrix, both of rational numbers. Substituting formulas (6) and (7) into system (9), we have

$$\Pi_{\mathsf{R}} \begin{pmatrix} \hat{L} \\ \tilde{L} \end{pmatrix} D^{-1} \begin{pmatrix} \hat{U} & \tilde{U} \end{pmatrix} \Pi_{\mathsf{C}} X = B. \tag{10}$$

Let Y be the matrix $U \Pi_{\mathsf{C}} X$ $(r \times d)$. Let also $\hat{B}$ and $\tilde{B}$ be the first r and last $m - r$ rows of $\Pi_{\mathsf{R}}^{-1} B$, respectively. Equation (10) then becomes

$$\begin{pmatrix} \hat{L} \\ \tilde{L} \end{pmatrix} D^{-1} Y = \begin{pmatrix} \hat{B} \\ \tilde{B} \end{pmatrix}. \tag{11}$$

We can split the system (11) into two systems

$$\hat{L} D^{-1} Y = \hat{B} \quad \text{and} \quad \tilde{L} D^{-1} Y = \tilde{B}. \tag{12}$$

Because $\hat{L} D^{-1}$ is an $r \times r$ invertible matrix, we can compute a unique solution Y for the first system, that is, $Y = D \hat{L}^{-1} \hat{B}$. The elements of $D \hat{L}^{-1}$ turn out to be all integers.[7]

Now let $Z = \Pi_{\mathsf{C}} X$ $(n \times d)$, and let $\hat{Z}$ and $\tilde{Z}$ be the first r and last $n - r$ rows of Z. We can choose the $(n-r) \times d$ matrix $\tilde{Z}$ arbitrarily, and then compute the $r \times d$ rational matrix $\hat{Z}$ by solving $\hat{U} \hat{Z} = Y - \tilde{U} \tilde{Z}$. In particular, setting $\tilde{Z} = 0_{(n-r) \times d}$, we get $\hat{Z} = \hat{U}^{-1} Y$. Then, we get the solution by $X = \Pi_{\mathsf{C}}^{-1} Z$.

Computation cost: The running time is dominated by the computation of the $r \times r$ matrices Y and $\hat{Z}$. Because $\hat{L}$ and $\hat{U}$ are triangular $r \times r$ matrices, and D is $r \times r$ diagonal, it follows that in the worst case the matrices $Y = D \hat{L}^{-1} \hat{B}$ and $\hat{Z} = \hat{U}^{-1} Y$ can be computed in $\Theta(r^2 d)$ arithmetic operations. If the numbers in the factorization have $\Theta(r)$ bits (as discussed in Section 4.1), this computation takes $\Theta(r^4 d)$ time. The assembly of X from $\tilde{Z}$ takes $\Theta(rnd)$ time, which is the cost of copying its elements, giving $\Theta(r(r^3 + n)d)$ total time.

5 | SPECIALIZED TOOLS

In this section, we describe some additional exact linear algebra tools that are specific to the ECLES method. Recall that $\widetilde{m} = |\mathcal{E}|$ is the number of relevant constraints, and $\widetilde{n} = |\mathcal{D}|$ is the number of derived parameters.

5.1 | Checking for solvability

For each requested editing action, ECLES needs to check whether the system (2) has at least one solution. By the derivation in Section 4.2, a linear system $AX = B$ has a solution only if there is an $r \times d$ matrix Y that satisfies both Equation (12). The first equation is automatically satisfied because it is used to compute Y, as described in that section. The second equation $\widetilde{L}D^{-1}Y = \widetilde{B}$ is nonempty only if the rank r of A is less than its number of rows m, in which case either some constraints are redundant or there are incompatible constraints.

5.1.1 | Weak solvability

To verify whether the generic linear system (9) has a solution, *for the given right-hand-side matrix B*, it is necessary and sufficient to check if the second system in Equation (12) is satisfied by the computed matrix Y, that is,

$$\widetilde{L}\widehat{L}^{-1}\widehat{B} = \widetilde{B}. \tag{13}$$

Computation cost: Because $\widetilde{L}$, $\widehat{L}$, and $\widehat{B}$ have respective $(m - r) \times r$, $r \times r$, and $r \times d$ elements, in the worst case, their product can be computed in $\Theta(r^2d+(m-r)rd) = \Theta(mrd)$ arithmetic operations. In the case of system (2), were $m = \widetilde{m} = |\mathcal{E}|$ and the elements of L have $\Theta(r)$ bits, this test takes $\Theta(mr^3d)$ time.

5.1.2 | Strong solvability

Strong solvability means that the system (2) can be solved for the current values $P_\mathcal{I}$ of the fixed parameters and any assignment $P'_\mathcal{A}$ to the anchor parameters. We can write Equation (5) as $H = C - KP'_\mathcal{A}$, where $C = Q_\mathcal{E} - R_{\mathcal{E}\mathcal{F}}P_\mathcal{F}$ and $K = R_{\mathcal{E}\mathcal{A}}$. Multiplying this equation by Π_R^{-1}, we have

$$\Pi_\mathsf{R}^{-1}H = \Pi_\mathsf{R}^{-1}C - \Pi_\mathsf{R}^{-1}KP'_\mathcal{A}. \tag{14}$$

The matrices in this equation can be split as

$$\Pi_\mathsf{R}^{-1}C = \begin{pmatrix} \widehat{C} \\ \widetilde{C} \end{pmatrix} \qquad \text{and} \qquad \Pi_\mathsf{R}^{-1}K = \begin{pmatrix} \widehat{K} \\ \widetilde{K} \end{pmatrix}, \tag{15}$$

where $\widehat{C}$ and $\widehat{K}$ are the r first rows of the matrices $\Pi_\mathsf{R}^{-1}C$ and $\Pi_\mathsf{R}^{-1}K$, and $\widetilde{C}$ and $\widetilde{K}$ are the remaining rows. Then, with the analogous decomposition of $\Pi_\mathsf{R}^{-1}H$ into $\widehat{H}$ and $\widetilde{H}$ used in Section 4.2, we have two equations

$$\widehat{H} = \widehat{C} - \widehat{K}P'_\mathcal{A} \qquad \text{and} \qquad \widetilde{H} = \widetilde{C} - \widetilde{K}P'_\mathcal{A}. \tag{16}$$

As in Section 4.2, let $Y = D\widehat{L}^{-1}\widehat{H}$. System (2) has a solution if $\widetilde{L}D^{-1}Y = \widetilde{H}$, that is, if

$$\widetilde{L}\widehat{L}^{-1}(\widehat{C} - \widehat{K}P'_\mathcal{A}) = \widetilde{C} - \widetilde{K}P'_\mathcal{A}, \tag{17}$$

which can be rearranged as

$$\widetilde{L}\widehat{L}^{-1}\widehat{C} - \widetilde{C} = (\widetilde{L}\widehat{L}^{-1}\widehat{K} - \widetilde{K})P'_\mathcal{A}. \tag{18}$$

If the matrix $\widetilde{L}\widehat{L}^{-1}\widehat{K} - \widetilde{K}$ is not zero, the right-hand side depends on $P'_\mathcal{A}$, whereas the left-hand side does not. Therefore, for strong solvability, this matrix must be zero and, then, the left-hand side must be zero too. That is, we must have

$$\widetilde{L}\widehat{L}^{-1}\widehat{C} = \widetilde{C} \qquad \text{and} \qquad \widetilde{L}\widehat{L}^{-1}\widehat{K} = \widetilde{K}. \tag{19}$$

Computation cost: Let $a = |\mathcal{A}|$ be the number of anchors, and $f = |\mathcal{F}|$ be the number of relevant fixed points. The matrices $R_{\mathcal{E}\mathcal{F}}$ and $P_{\mathcal{F}}$ have size $\widetilde{m} \times f$ and $f \times d$, with fixed-size entries, so the matrix C can be computed in time $\Theta(\widetilde{m} f d)$. Because the matrix $\widehat{C}$ has the same size as $\widehat{H}$, the product $\widetilde{L}\widehat{L}^{-1}\widehat{C}$ also requires $\Theta(r^2 d + (\widetilde{m} - r)rd) = \Theta(\widetilde{m}rd)$ arithmetic operations. The size of K is $\widetilde{m} \times a$; therefore, the product $\widetilde{L}\widehat{L}^{-1}\widehat{K}$ requires $\Theta((\widetilde{m} - r)ra + r^2 a) = \Theta(\widetilde{m}ra)$ operations. Because the elements of L have $\Theta(r)$ bits, the total time to check strong solvability is $\Theta(\widetilde{m}(fd + r^3(d + a)))$.

5.2 | Removing redundant equations

In many applications, there are often redundant constraints, especially when some of the parameters are considered fixed. Therefore, one of the subproblems that we need to solve is to identify and ignore redundant equations in a linear system $AX = B$.

This task is a side effect of computing the fraction-free LDU factoring of the matrix A. If the rank r determined during the factoring is less than m, as in example (8), the system $AX = B$ has redundant equations and can be replaced by

$$\widehat{A}X = \widehat{B}, \tag{20}$$

where $\widehat{A}$ is an $r \times n$ matrix, which is the first r rows of $\Pi_{\mathsf{R}}^{-1}A = LD^{-1}U\Pi_{\mathsf{C}}$, that is, $\widehat{L}D^{-1}U\Pi_{\mathsf{C}}$, and $\widehat{B}$ is an $r \times d$ matrix, which is the first r rows of $\Pi_{\mathsf{R}}^{-1}B$. For the matrix A of example (8), we have

$$\widehat{A} = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ 4 & 15 & -80 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -80 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -6 & -16 & -19 \\ 0 & 1 & -3 & 4 \\ 0 & 0 & -80 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} -1 & -6 & -19 & -16 \\ 1 & 5 & 15 & 19 \\ 4 & 9 & 16 & 29 \end{bmatrix}. \tag{21}$$

Computation cost: Given the fraction-free LDU decomposition of the $m \times n$ matrix A, the nonredundant system matrices $\widehat{A}$ and $\widehat{B}$ can be constructed by copying $\Theta(r(n + d))$ elements from the original matrices A and B.

5.3 | Using the hints by constrained least squares

We now describe in detail how to solve a system $AX = B$ when the rank r of the matrix A is smaller than the number of unknowns n. (Otherwise, if $r = n$, the single solution can be obtained as described in Section 4.2.) For this purpose, we use the *least squares* (*quadratic optimization*) criterion with affine constraints.[9]

Specifically, given the $n \times d$ matrix X' (whose rows, in the ECLES algorithm, will be the parameter hints P'_D), we want to find an $n \times d$ matrix X'' (in ECLES, the new parameter values P''_D) that minimizes the squared Euclidean distance between each row X''_i and the desired row X''_i, while respecting the constraints. More precisely, we want to find the $n \times d$ matrix X'' that minimizes the goal function

$$E(X'') = \sum_{i=1}^{\widetilde{n}} \sum_{j=1}^{d} (X''_{ij} - X'_{ij})^2, \tag{22}$$

while satisfying the r nonredundant equations $\widehat{A}X'' = \widehat{B}$ of the system $AX'' = B$. According to the standard least squares derivation,[6,9] the constrained minimum X'' of the goal function E defined by Equation (22) must satisfy the two linear equation systems

$$X'' = X' - \frac{1}{2}\widehat{A}^{\mathsf{T}}\Lambda \tag{23}$$

$$\widehat{A}X'' = \widehat{B}. \tag{24}$$

where $\widehat{A}^{\mathsf{T}}$ is the $n \times r$ transpose of $\widehat{A}$, and Λ is the $r \times d$ matrix of the Lagrange multipliers associated to the r nonredundant relevant constraints.

The system (24) can be solved in two steps. Namely, substituting X'' as given by Equation (23) into Equation (24), we get

$$(\widehat{A}\widehat{A}^{\mathsf{T}})\Lambda = 2(\widehat{A}X' - \widehat{B}). \tag{25}$$

Because $N = \widehat{A}\widehat{A}^{\top}$ is an $r \times r$ invertible matrix of integers numbers, we can obtain Λ by computing the fraction-free decomposition of N and then solving the system (25) as described in Section 4.2. Then, we can compute X'' by Equation (23).

Computation cost: The $r \times r$ full-rank matrix N can be computed in $\Theta(r^2 n)$ operations on numbers of $\Theta(1)$ bits, and its elements will have $\Theta(1)$ bits too. As described in Appendix, given the fraction-free LDU decomposition of N, the linear system (25) then can be solved with $\Theta(r^2 d)$ operations. The optimal solution X'' can then be computed by formula (23) with $\Theta(rnd)$ operations. Because the matrix elements in those formulas will have $\Theta(r)$ bits in the worst case, and $r \leq n$, the computation time will then be $\Theta(r(r+n)d) = \Theta(rnd)$ operations and $\Theta(r^3 nd)$ time.

6 | THE ECLES METHOD

We now describe the ECLES method proper, which solves the problem defined in Section 3, namely, replacing the parameter matrix P with a new matrix P'', so as to satisfy a set of linear constraints $RP'' = Q$, with $P''_{\mathcal{I}} = P_{\mathcal{I}}$, $P''_{\mathcal{A}} = P'_{\mathcal{A}}$, and $P''_{\mathcal{D}}$ as close as possible to $P'_{\mathcal{D}}$. In this section, m and n again are the total numbers of constraints and parameters.

6.1 | Simplified description

The ECLES method consists of two procedures, `ECLES.Initialize` and `ECLES.Update`. The `ECLES.Initialize` procedure is called when the user chooses the sets $\mathcal{A}$ and $\mathcal{D}$ of parameters to be adjusted through some editing software (the *user interface*). The `ECLES.Update` procedure is then called one or more times as the user specifies new values $P'_{\mathcal{A}}$ for the anchors and possibly new hints $P'_{\mathcal{D}}$ (see Figure 5).

The `ECLES.Initialize` procedure, detailed in Section 6.2 below, identifies the subset $\mathcal{E} \subseteq \mathcal{M}$ of relevant constraints and the set $\mathcal{F}$ of the fixed relevant parameters. In this way, the editing problem is reduced from n parameters and m constraints to the $a+\widetilde{n}+f \leq n$ relevant parameters $\mathcal{A}\cup\mathcal{D}\cup\mathcal{F}$ and the $\widetilde{m} \leq m$ relevant constraints $\mathcal{E}$. The procedure returns the coefficient matrix $S = R_{\mathcal{E}\mathcal{D}}$ of the relevant constraints, and the matrix $N = \widehat{S}\widehat{S}^{\top}$ used in the least squares step, to be used by the `ECLES.Update` procedure, both in the fraction-free factored form, namely as tuples **S** and **N** as defined in Section 4.1. Optionally, the procedure also checks whether the given sets $\mathcal{A}$ and $\mathcal{D}$ satisfy the strong solvability condition.

The `ECLES.Update` procedure, described with more detail in Section 6.3, solves the constrained least squares problem that combines the set of nonredundant relevant constraints, the specified anchor values $P'_{\mathcal{A}}$, and the hints $P'_{\mathcal{D}}$, and computes the new values $P''_{\mathcal{D}}$ for the derived parameters. If the strong solvability was not checked by `ECLES.Initialize`, it also checks the weak solvability for the particular values of $P'_{\mathcal{A}}$ given.

6.2 | The `ECLES.Initialize` procedure

The `ECLES.Initialize` procedure is given by Algorithm 1. In step 2, the `ECLES.ExtractRelevant` procedure is used to identify the set of relevant equations $\mathcal{E}$ and rewrites them in the form of the system (2). Namely, the `ECLES.ExtractRelevant` procedure identifies the equations of R that depend on any parameter of the sets $\mathcal{A}$ or $\mathcal{D}$ (the set $\mathcal{E}$ of relevant constraints) and the set $\mathcal{F}$ of fixed relevant parameters.

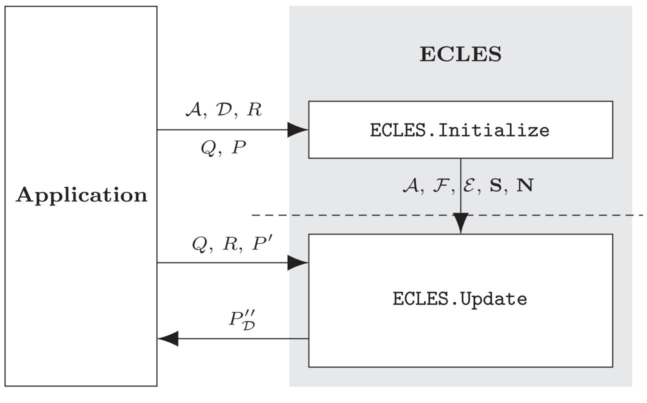
ECLES
Application
$\mathcal{A}$, $\mathcal{D}$, R
Q, P
ECLES.Initialize
$\mathcal{A}$, $\mathcal{F}$, $\mathcal{E}$, **S**, **N**
Q, R, P'
ECLES.Update
$P''_{\mathcal{D}}$

FIGURE 5 Interaction model between application and the ECLES algorithm

In step 3, `ECLES.Initialize` computes the fraction-free LDU factoring $\mathbf{S}$ of the matrix $S = R_{\mathcal{E}\mathcal{D}}$, as described in Section 4.1. In the process, it obtains the rank r of S. The first r rows of $\Pi_{\mathsf{R}}^{-1}S$ and $\Pi_{\mathsf{R}}^{-1}H$ are a set of nonredundant equations.

Algorithm 1 `ECLES.Initialize`

```
Input: $\mathcal{A}$: set of indices of the $a$ anchor parameters;
       $\mathcal{D}$: set of indices of the $\widetilde{n}$ derived parameters;
       $R$: $m \times n$ coefficient matrix of all constraint equations;
       $Q$: $m \times d$ matrix of independent terms of all constraints;
       $P$: $n \times d$ matrix of the current values of all parameters;
       $strong$: boolean flag requesting strong satisfiability;
Output: $\mathcal{F}$: set of indices of the $f$ relevant fixed parameters;
        $\mathcal{E}$: set of indices of the $\widetilde{m}$ relevant constraints;
        $\mathbf{S}$: fraction-free LDU factoring of $S = R_{\mathcal{E}\mathcal{D}}$;
        $\mathbf{N}$: fraction-free LDU factoring of $N = \widehat{S}\widehat{S}^{\mathsf{T}}$.
1 begin
2     $(\mathcal{E}, \mathcal{F}) \leftarrow$ ECLES.ExtractRelevant($\mathcal{A}, \mathcal{D}, R$)
3     $\mathbf{S} \leftarrow$ LinSys.LDUFactor($R_{\mathcal{E}\mathcal{D}}$)
4     $(\widehat{S}, \widetilde{S}) \leftarrow$ LinSys.SplitRows(($\mathbf{S}.\Pi_{\mathsf{R}}^{-1}$)$S$, $\mathbf{S}.r$);
5     $\mathbf{N} \leftarrow$ LinSys.LDUFactor($\widehat{S}\widehat{S}^{\mathsf{T}}$)
6     if $strong$ then
7         if not ECLES.CheckStrongSolvabitity($R, \mathcal{E}, \mathcal{A}, \mathcal{F}, Q, P, \mathbf{S}.\Pi_{\mathsf{R}}, \mathbf{S}.L, \mathbf{S}.r$) then
8             error "Strong solvability is not satisfied!"
```

If strong solvability is required, `ECLES.Initialize` calls `ECLES.CheckStrongSolvabitity` in step 7 to verify it, as described in Section 5.1.2. If the condition is not satisfied, then `ECLES.Initialize` is aborted, and the caller is notified of the problem. The caller then may, for instance, choose different sets $\mathcal{A}$ and $\mathcal{D}$, and retry the editing action. Otherwise, the procedure returns the sets $\mathcal{E}$ and $\mathcal{F}$ and the factorizations $\mathbf{S}$ and $\mathbf{N}$ of $S = R_{\mathcal{E}\mathcal{D}}$ and $N = \widehat{S}\widehat{S}^{\mathsf{T}}$.

Computation cost: The cost of `ECLES.Initialize` is dominated by the LDU factoring of the relevant constraint matrix $S = R_{\mathcal{E}\mathcal{D}}$, in step 3. According to the analysis in Appendix, that step takes $\Theta(\widetilde{m}r\widetilde{n})$ arithmetic operations on numbers with $\Theta(r)$ bits, therefore $\Theta(\widetilde{m}r^3\widetilde{n})$ time. As observed in Section 5.1.2, the strong solvability check in step 7 (if requested) requires an additional $\Theta(\widetilde{m}(fd+r^3(d+a)))$ time, where $f = |\mathcal{F}|$ is the number of fixed relevant parameters and $a = |\mathcal{A}|$ is the number of anchors. The extraction of the relevant constraints and parameters (step 2) can be done in $\Theta(\widetilde{m}(f + a + \widetilde{n}))$ with the proper data structures.

6.3 | The `ECLES.Update` procedure

The `ECLES.Update` procedure is detailed in Algorithm 2. Its inputs are the outputs of `ECLES.Initialize` ($\mathcal{E}$, $\mathcal{F}$, $\mathbf{S}$, and $\mathbf{N}$), and the matrix P'. The latter contains the new anchor values, the hints for derived parameters, and the current values of the fixed parameters.

If strong solvability was not checked in `ECLES.Initialize`, the `ECLES.Update` procedure first uses the `ECLES.CheckWeakSolvabitity` procedure (in step 5) to determine if the weak solvability condition is satisfied for the given anchor values $P'_{\mathcal{A}}$ and current fixed parameter values of $P_{\mathcal{F}}$. See Section 5.1.1. If the condition is not satisfied, `ECLES.Update` returns a message to the application; which, for example, may cancel the editing action and ask the user to provide alternative anchor values.

Otherwise, if the condition is satisfied, the procedure computes the $\widetilde{n}$ values $P''_{\mathcal{D}}$ of the derived parameters by solving the system $\widehat{S}P''_{\mathcal{D}} = \widehat{H}$, where $\widehat{S}$ is the first r rows of $(\mathbf{S}.\Pi_{\mathsf{R}}^{-1})S$ and $\widehat{H}$ is the first r rows of $(\mathbf{S}.\Pi_{\mathsf{R}}^{-1})H$. This computation is performed either in step 8, as described in Section 4.2 (procedure `LinSys.Solve`) or, if the system is underdetermined, in step 10 by constrained least squares, as described in Section 5.3 (procedure `LSQ.Solve`), taking into account the suggested values $P'_{\mathcal{D}}$ given by the user interface.

```
Algorithm 2 ECLES.Update
  Input: $\mathcal{A}$: set of indices of the $a$ anchor parameters;
         $\mathcal{F}$: set of indices of the $f$ relevant fixed parameters;
         $\mathcal{E}$: set of indices of the $\tilde{m}$ relevant constraints;
         $R$: $m \times n$ coefficient matrix of all constraint equations;
         $Q$: $m \times d$ matrix of independent terms of all constraints;
         $\mathbf{S}$: fraction-free LDU factoring of $S = R_{\mathcal{E}D}$;
         $\mathbf{N}$: fraction-free LDU factoring of $N = \widehat{S}\widehat{S}^{\top}$;
         $P'$: $n \times d$ matrix of the suggested values of all parameters.
  Output: $P''_D$: $\tilde{n} \times d$ matrix of the new values of the $\tilde{n}$ derived parameters.
1 begin
2   $H \leftarrow Q_{\mathcal{E}} - R_{\mathcal{E}\mathcal{A}}P'_{\mathcal{A}} - R_{\mathcal{E}\mathcal{F}}P_{\mathcal{F}}$
3   $(\widehat{H}, \widetilde{H}) \leftarrow$ LinSys.SplitRows($(\mathbf{S}.\Pi_{\mathrm{R}}^{-1})H, \mathbf{S}.r$)
4   $(\widehat{L}, \widetilde{L}) \leftarrow$ LinSys.SplitRows($\mathbf{S}.L, \mathbf{S}.r$)
5   if not ECLES.CheckWeakSolvabitity($\widehat{L}, \widetilde{L}, \widehat{H}, \widetilde{H}$) then
6     error "Weak solvability is not satisfied!"
7   if $\mathbf{S}.r = \mathbf{S}.n$ then
8     $P''_D \leftarrow$ LinSys.Solve ($\mathbf{S}, \widehat{H}$)
9   else
10    $P''_D \leftarrow$ LSQ.Solve ($\mathbf{N}, \widehat{H}, P'$)
```

Computation cost: The cost of ECLES.Update is dominated by the solution of the linear system in step 8 or of the constrained least squares system in step 10. As discussed in Section 4.2, the former takes $\Theta(r(r^3+\tilde{n})d)$ time, which, because $r = \mathbf{S}.n = \tilde{n}$, is the same as $\Theta(r^4 d) = \Theta(r^3\tilde{n}d)$ time. As discussed in Section 5.3, the latter takes $\Theta(r^3\tilde{n}d)$ time. So we can say that ECLES.Update takes $\Theta(r^3\tilde{n}d)$ time in the worst case.

7 | EXAMPLE APPLICATION

As a practical example application of ECLES, we built an interactive editor, which we called PrisMystic,[6,10] to manually modify 3D geometric models according to the space deformation paradigm.[29] The object to be deformed is assumed to be described by a fine mesh of triangles, the *model mesh M*. The space deformation implemented by PrisMystic is mainly determined by a spline function ϕ from $\mathbb{R}^2$ to $\mathbb{R}^2$ applied to the X and Y coordinates of the object. Two other splines ψ_0, ψ_1 from $\mathbb{R}^2$ to $\mathbb{R}$ offer independent control over the Z coordinates of the model mesh. Thus, while the objects are three-dimensional, the deformation can be described as "2.5-dimensional" (see Figure 6).

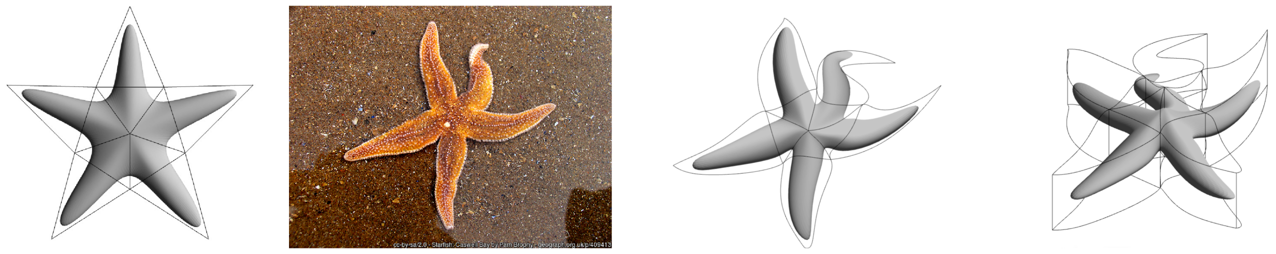

FIGURE 6 Example of using the PrisMystic editor to deform a 3D model of a starfish. At extreme left, the top view of the undeformed model mesh (gray) and of the reference mesh of the deformation splines (straight lines). At midleft, a photo of a real starfish.[30] At midright, the top view of the object mesh (gray) and control mesh (curved lines) distorted by the XY-deformation map ϕ, which was edited by hand so as to match the photo. At extreme right, an oblique view of the same, showing that each triangle of the reference mesh is the cross-section of a prismatic cell. Note that the top and bottom faces of the prisms have been displaced and stretched in the vertical direction by the two Z-deformation splines ψ_0, ψ_1, so as to obtain the desired thickness for the arms and body of the model

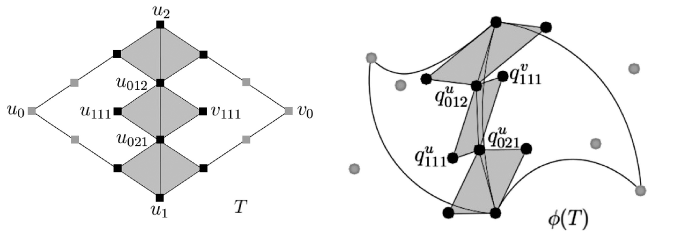

FIGURE 7 Smoothness constraints for a spline deformation ϕ of the plane of degree 3. The reference mesh (left) has two triangles u and v, and each patch of the spline has 10 Bézier control points (gray and black dots) shown at their reference positions u_{ijk}, v_{ijk}. Four control points (those with $i = 0$) are shared by the two patches, thus ensuring C^0 continuity across the common edge. The spline is defined by placing each control point u_{ijk}, v_{ijk} at new position q^u_{ijk}, q^v_{ijk} (right). Each triangle of the reference mesh is then mapped by ϕ to a curvilinear triangle, as shown. To ensure C^1 continuity, these displaced control points must satisfy three linear constraint equations, represented by the gray quadrilaterals. For instance, the quadrilateral whose corners are $q^u_{111}, q^u_{012}, q^u_{021}, q^v_{111}$ must be an affine image of the quadrilateral formed by their nominal positions $u_{111}, u_{012}, u_{021}, v_{111}$ on the reference mesh, which is also an affine image of the quadrilateral formed by the triangles u and v

The PrisMystic editor is described in detail in the first author's thesis.[6] Here, we summarize only the aspects that relate to ECLES. Each of the three deformation splines consists of triangular polynomial patches of total degree 5, joined with C^1 smoothness. All three splines are defined with respect to the same coarse 2D *reference mesh* of triangles T on the plane $\mathbb{R}^2$, with straight edges. This mesh implicitly defines a 3D mesh of triangular prismatic elements, with vertical sides and horizontal faces, which encloses the object to be deformed.

Each patch of the spline is determined by its Bézier controls.[16] Each control is a single real value ($d = 1$) for the Z-deformation splines ψ_0 and ψ_1, and a point of the plane ($d = 2$) for the XY-deformation spline ϕ. The C^1 continuity condition of each spline is expressed as a collection of homogeneous linear equations on those controls. Each equation, called a *quadrilateral constraint*, involves four Bézier controls straddling an edge of the reference mesh (see Figure 7). Each of the three splines can be edited independently. In the rest of this section, we will consider only the editing of the XY deformation ϕ; the editing of the other two splines is basically the same.

The connection between the graphical user interface of the PrisMystic editor and the ECLES general method is another algorithm (2DSD) that we developed specifically for this application—the editing of simplicial splines with C^1 smoothness constraints.[6,10] Figure 8 shows the control flow between the user, the PrisMystic graphical user interface, the spline deformation editing algorithm 2DSD, and the general parameter editing method ECLES.

As expected by ECLES, the coordinates of the n Bézier control points, in arbitrary order, are packaged by 2DSD as the rows of an $n \times d$ matrix P, and the m quadrilateral constraints are encoded by 2DSD as the matrix equation $RP = Q$, where R is an $m \times n$ matrix and Q is an $m \times d$ matrix of zeros. The coefficients of the constraints are derived from the coordinates of the vertices's of the reference mesh, which are integers with a fixed bit size and, therefore, are themselves integers with $\Theta(1)$ bits.

Each edit action starts with the user selecting one or more Bézier control points to be the anchor set $\mathcal{A}$. The graphical interface also lets the user explicitly select another set $\mathcal{S}$ of control points that should be included in the derived set $\mathcal{D}$. These two sets are given to the `2DSD.Select`. This procedure automatically expands $\mathcal{S}$ into a set $\mathcal{D}$ of derived control points, large enough to ensure that the strong solvability condition of ECLES is satisfied. The 2DSD method then calls `ECLES.Initialize` to extract the relevant constraints and parameters for this editing action and to compute the fraction-free LDU factorization of the constraint matrix.

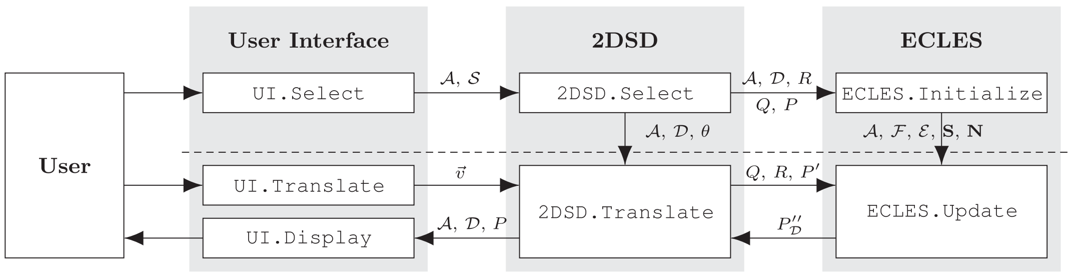
User Interface
2DSD
ECLES
User
UI.Select
UI.Translate
UI.Display
2DSD.Select
2DSD.Translate
ECLES.Initialize
ECLES.Update
$\mathcal{A}, \mathcal{S}$
$\mathcal{A}, \mathcal{D}, R$
Q, P
$\mathcal{A}, \mathcal{D}, \theta$
$\mathcal{A}, \mathcal{F}, \mathcal{E}, \mathbf{S}, \mathbf{N}$
$\vec{v}$
Q, R, P'
$\mathcal{A}, \mathcal{D}, P$
$P''_{\mathcal{D}}$

FIGURE 8 Control flow for a typical editing action (translation of anchor points)

During the editing action, the user will be continuously adjusting the position of the anchors, for example by dragging them around with the mouse (an "anchor translation" action). The `2DSD.Translate` procedure receives each new displacement vector v of the mouse, and converts that to a new anchor point position P'_i for each i in $\mathcal{A}$. That procedure also computes an appropriate hint position Pi' for each i in $\mathcal{D}$. It then calls `ECLES.Update` to compute the adjusted positions $P''_{\mathcal{D}}$ for the derived points.

7.1 | Implementation and performance

The PrisMystic editor and its underlying algorithms (2DSD, ECLES, exact linear algebra and exact least squares) were implemented in the `C++` language by the authors.[31] We used the library FLINT (Fast Library for Number Theory) for arithmetic operations on integers and rational numbers of arbitrary size.[32] The following tests were performed on a DELL laptop with four CPUs, an Intel Core i7 – 7500U with 2.70 GHz clock, 15.6GB memory, and an Intel HD 620 (Kaby Lake GT2) graphics card running Ubuntu Linux software.

We measured the performance of the ECLES algorithms by repeatedly performing a specific PrisMystic editing action on the starfish mesh model shown in Figure 6 (20,480 triangles), using C^1 deformation splines of degree 5 with the reference mesh shown at extreme left. The *XY*-deformation spline ϕ has 10 simplicial patches, with a total of $n = 151$ distinct Bézier control points (21 points per patch, some of them shared by two or more patches), each with $d = 2$ coordinates: 302 real parameter values in total. There are $m = 50$ quadrilateral constraints, expressed by equations with integer coefficients with at most 25 bits each.

In this test, we specified one anchor point ($|\mathcal{A}| = 1$), at the center of the control mesh, and left only five control points fixed ($|\mathcal{F}| = 5$), at the very tip of each arm. Thus, there were $\widetilde{n} = |\mathcal{D}| = 145$ derived points to be adjusted by ECLES. All $\widetilde{m} = 50$ constraints were relevant, and ECLES found that the rank of the matrix $S = R_{\mathcal{E}\mathcal{D}}$ (50×145) was $r = 48$. That is, there were two redundant constraints, and the solution space for $P''_{\mathcal{D}}$ had dimension $145 - 48 = 97$.

Therefore, in each edit action, `ECLES.Initialize` had to factor the $\widetilde{m} \times \widetilde{n} = 50 \times 145$ integer matrix S with 25-bit elements. The largest numbers in the resulting factorization (in the $\mathbf{S}.U$ matrix) had 40 bits. The elements of the 48×48 matrix $N = \widehat{A}\widehat{A}^{\top}$ had at most 50 bits, and the largest numbers in its factors (in the $\mathbf{N}.U$ matrix) had 640 bits. Then, at each position of the mouse, `ECLES.Update` had to solve the constrained least squares problem for $\widetilde{n} \times d = 145 \times 2$ unknowns, using the $\mathbf{N}$ matrix factorization.

This edit action was repeated twenty times, in such a way that `ECLES.Initialize` was executed from scratch each time. For each edit action, the anchor point was displaced with the mouse to five different positions, resulting in 100 calls to `ECLES.Update`. In these tests, we measured an average running time of about 120 ms for each call of the procedure `ECLES.Initialize`, and about 47 ms for each call of `ECLES.Update`. The latter was fast enough to allow smooth dragging of the anchors with real-time feedback, that is, display of the reference and model meshes with the adjusted parameter values P'' for each position of the mouse. A real-time video of this test is available.[31]

The cost of `2DSD.Translate` is basically (a) the cost of `ECLES.Update`, plus (b) the time to recompute the coordinates of all vertices of the model mesh M that were affected by the `ECLES.Update`, plus (c) the cost of rerendering the mesh M. Item (a) is completely independent of the size of the M and depends only on the number of control points and constraints of the deformation that are affected by the editing action. Items (b) and (c) depend on the size of the model mesh M; however, they must be paid by any mesh deformation method, so they are not relevant to the evaluation of ECLES.

In particular, item (b) requires enumerating each triangle of the reference mesh T that shares one of the affected Bézier points, and recomputing the deformation $\phi(v)$ for every vertex v of M that lies inside that triangle. The computation of ϕ (by the De Casteljau algorithm) takes constant time, independently of how many points or triangles are affected. Therefore, item (b) is proportional to the number of vertices of M that lie in the triangles of T affected by the action (and not on the total size of M). In theory one could optimize also item (c) by repainting only the triangles of M that had at least one vertex $\phi(v)$ recomputed. However, visibility, self-occlusions (and shadows, if used) are significant complications. Anyway, any such optimization should be made outside of 2DSD and would apply equally to any deformation editing method.

In the tests above, all triangles of T were affected, so all 20,480 vertices of M had to be recomputed by `2DSD.Translate`, which took 22 ms (item (b)) (not counting the 47 ms of `ECLES.Update`). In order to confirm the analysis above, we repeated the test with four larger model meshes, with 41,050; 80,598; and 162,246 triangles, obtained from M by two-fold subdivision of each triangle, successively. The cost of ECLES did not change (as expected), and that of `2DSD.Translate` to recompute the coordinates of the vertices of the model mesh M increased to about 41 ms, 82 ms, and 164 ms, respectively (see Figure 9).

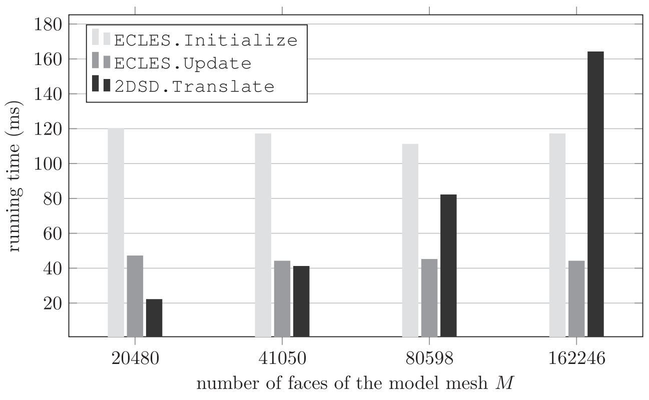

FIGURE 9 Average running time of the procedures `ECLES.Initialize`, `ECLES.Update` and, `2DSD.Translate` (recomputing of vertices coordinates), varying the number of faces of the model mesh M

7.2 | Other applications

The 2DSD algorithm[6] can be adapted to other applications such as editing 2.5D and 3D space deformation.[33] The method described can be used also to edit any spline function ϕ from $\mathbb{R}^s$ to $\mathbb{R}^d$, defined over a fixed reference mesh of s-dimensional "straight" simplicial cells. This approach could be used, for example, to model altitude in a smooth digital terrain ($s = 2$, $d = 1$), temperature in a region of three-space ($s = 3$, $d = 1$), or displacements of an elastic three-dimensional object under load ($s = 3$, $d = 3$). The requirement that a spline of degree g must have continuity of order $r \geq 1$ between adjacent mesh cells can be expressed by a set of *bipyramid constraints* of the form $p = \sum_{k=0}^{s} \alpha_k q_k$, where p and the q_i are Bernstein-Bézier control points of the spline in $\mathbb{R}^d$, and the α_i are fixed real coefficients defined by the geometry of the reference mesh. Each such constraint is therefore d fixed affine constraints on the d coordinates of those control points in $\mathbb{R}^d$. The same approach can also be used to edit smooth spline surfaces in three-space ($s = 2$, $d = 3$), built from triangular Bézier patches, as long as the topology of the surface is that of planar mesh with straight edges.

8 | CONCLUSIONS

The ECLES algorithm, described in this paper, is a general method for interactive editing of parameters subject to linear or affine constraints. Thanks to the use of exact integer arithmetic, the method is insensitive to redundancies among constraints and avoids failures due to roundoff errors.

ECLES is more flexible and intuitive than the finite element approach, because it lets the user choose the dependent and independent parameters in each editing action and provide "ideal" values for the dependent ones, in case the problem turns out to be underconstrained. Constrained least squares optimization is used to select the "best" solution in this case.

The algorithm was implemented and, as a demonstrative example, used as the core of an interactive editor for 2.5D deformations of object models described by fine triangle meshes.

ACKNOWLEDGEMENT

This work is supported by the Brazilian government through grants CAPES PNPD/PPGInf/UFPR and CNPq (301016/92-5). The authors declare to have no potential conflict of interests.

ORCID

Elisa de Cássia Silva Rodrigues https://orcid.org/0000-0002-4649-1832
Jorge Stolfi https://orcid.org/0000-0002-3159-4839

REFERENCES

1. Islam B, Inam T, Kaliyaperumal B. Overview and challenges of different image morphing algorithms. Int J Adv Res Comput Sci Electron Eng. 2013;2(4). http://www.ijarcsee.org/index.php/IJARCSEE/article/view/368
2. Sotiras A, Davatzikos C, Paragios N. Deformable medical image registration: a survey. IEEE Trans Med Imaging. 2013;32(7):1153–1190.

3. Wolberg G. Image morphing: a survey. Vis Comput. 1998;14(8–9):360–372.
4. Zitová B, Flusser J. Image registration methods: a survey. Image Vis Comput. 2003;21(11):977–1000.
5. Rodrigues ECS, Stolfi J. ECLeS: a flexible and general method for local editing of parameters with linear constraints. Proceedings of the Workshop of Works in Progress (SIBGRAPI '15); 2015 Aug 26–29; Salvador, Brazil. Washington, DC: IEEE Computer Society. p. 1–4.
6. Rodrigues ECS. Mathematical and computational methods for modeling and editing deformations. Campinas, Brazil: Institute of Computing - University of Campinas; 2017.
7. Jeffrey DJ. LU factoring of non-invertible matrices. ACM Commun Comput Algebra. 2010;44(1/2):1–8.
8. Turner PR. A simplified fraction-free integer Gauss elimination algorithm. 1995.
9. Fletcher R. Practical methods of optimization. 2nd ed. New York, NY: Wiley-Interscience; 1987.
10. Rodrigues ECS, Stolfi J. Interactive editing of $\mathbf{C}^1$-continuous 2D spline deformations using ECLES method. Proceedings of the 29th Conference on Graphics, Patterns and Images (SIBGRAPI'16); 2016 Oct 4–7; São Paulo, Brazil. Washington, DC: IEEE Computer Society. http://gibis.unifesp.br/sibgrapi16
11. Botsch M, Kobbelt L. An intuitive framework for real-time freeform modeling. ACM Trans Graph. 2004;23(3):630–634.
12. Heydon A, Nelson G. The Juno-2 constraint-based drawing editor. Technical Report 131a, Digital Systems Research. 1994.
13. Knuth DE. Tex and metafont, new directions in typesetting. Stanford, CA: American Mathematical Society and Digital Press; 1979.
14. Nelson G. Juno, a constraint-based graphics system. SIGGRAPH Comput Graph. 1985;19(3):235–243.
15. Sorkine O, Cohen-Or D. Least-squares meshes. Proceedings of the Shape Modeling International 2004; 2004 Jun 7–9; Washington, DC: IEEE Computer Society. p. 191–199.
16. Lai MJ, Schumaker LL. Spline functions on triangulations. New York, NY: Cambridge University Press; 2007.
17. Xu D. Incremental algorithms for the design of triangular based spline surfaces. Philadelphia, PA: Faculties of the University of Pennsylvania; 2002.
18. Farin G. Curves and surfaces for CAGD: A practical guide. 5th ed. San Francisco, CA: Morgan Kaufmann Publishers; 2002.
19. Boor C. Splines as linear combinations of B-splines: a survey. New York, NY. 1976.
20. Cui Y, Feng J. Real-time B-spline free-form deformation via GPU acceleration. Comput Graph. 2013;37(1–2):1–11.
21. He Y, Gu X, Qin H. Fairing triangular B-splines of arbitrary topology. Proceedings of the Pacific Graphics '05; 2005 Oct 12–14; Macao, China. p. 153–156.
22. Masuda H, Yoshioka Y, Furukawa Y. Interactive mesh deformation using equality-constrained least squares. Comput Graph. 2006;30(6):936–946.
23. Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 3rd ed. San Francisco, CA: The MIT Press; 2009.
24. Bareiss EH. Sylvester's identity and multistep integer-preserving Gaussian elimination. Math Comput. 1968;22:565–565.
25. Dureisseix D. Generalized fraction-free LU factorization for singular systems with kernel extraction. Linear Algebra Appl. 2011.
26. Nakos GC, Turner PR, Williams RM. Fraction-free algorithms for linear and polynomial equations. SIGSAM Bulletin. 1997;31(3):11–19.
27. Piziak R, Odell PL. Matrix theory: From generalized inverses to Jordan form. Boca Raton, FL: Chapman & Hall/CRC Press; 2007. (Chapman and Hall/CRC Pure and Applied Mathematics Series). https://books.google.com.br/books?id=8LsrAAAAYAAJ
28. Zhou W, Jeffrey DJ. Fraction-free matrix factors: new forms for LU and QR factors. Front Comput Sci China. 2008;2(1):67–80.
29. Marschner S, Shirley P. Fundamentals of computer graphics. 4th ed. Natick, MA: A. K. Peters; 2016.
30. Brophy P. Starfish: Caswell Bay. Available from: https://www.geograph.org.uk/reuse.php?id=409413. Accessed August 10, 2019.
31. Rodrigues ECS. Mathematical and computational methods for modeling and editing deformations. Available from: https://www.ic.unicamp.br/~erodrigues/. Accessed April 15, 2019.
32. Hart W, Johansson F, Pancratz S. FLINT: fast library for number theory. Version 2.4.0. 2013. http://flintlib.org
33. Gain J, Bechmann D. A survey of spatial deformation from a user-centered perspective. ACM Trans Graph. 2008;27(4):1–21.
34. Geddes KO, Czapor SR, Labahn G. Algorithms for computer algebra. Norwell, MA: Kluwer Academic Publishers; 1992.
35. Middeke J, Almohaimeed A, Jeffrey DJ. Common factors in fraction-free matrix reduction. Proceedings of the 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC '13); 2013 Sep 23–26; Timisoara, Romania. Washington, DC: IEEE Computer Society. p. 76–80. https://doi.org/10.1109/SYNASC.2013.17

How to cite this article: Rodrigues ECS, Stolfi J. ECLES: A general method for local editing of parameters with linear constraints. *Numer Linear Algebra Appl.* 2019;26:e2268. https://doi.org/10.1002/nla.2268

APPENDIX

FRACTION-FREE LDU FACTORING

We now describe the technique used to solve linear systems with integer coefficients, using fraction-free LDU factoring. The basic idea was described in the work of Jeffrey,[7] using Turner's simplification technique (see Appendix A.4). We adapted his algorithm to use full pivoting and column permutation, instead of partial pivoting, so as to handle rank-deficient matrices.

In this section, as in Section 4, the symbols m and n denote the number of rows and columns of the input matrix A, rather than the number of constraints and parameters of the original problem. As always, the computation cost is expressed in terms of m, n, and the matrix rank r, in the asymptotic worst-case sense.[23]

A.1 | Main LDU factoring procedure

The main procedure is `LinSys.LDUFactor` (Algorithm 3), which factors the matrix A into matrices Π_R, L, D^{-1}, U, and Π_C, as specified in Section 4.1. Recall that the matrices Π_R and Π_C can be represented as integer vectors to save space.

Algorithm 3 `LinSys.LDUFactor`

```
Input: $A$: $m \times n$ matrix of integers;
Output: $\mathbf{A}$: tuple $(m, r, n, \Pi_\mathsf{R}, L, D, U, \Pi_\mathsf{C})$ with the fraction-free factorization of $A$
1  begin
2      $\Pi_\mathsf{R} \leftarrow I_{m\times m}$; $L \leftarrow I_{m\times m}$; $D \leftarrow I_{m\times m}$; $U \leftarrow A$; $\Pi_\mathsf{C} \leftarrow I_{n\times n}$
3      $i \leftarrow 1$; $j \leftarrow n$
4      while $i \le m$ and $i \le j$ do
5          $(j, \Pi_\mathsf{R}, L, U, \Pi_\mathsf{C}) \leftarrow$ LinSys.Pivot$(i, j, m, n, \Pi_\mathsf{R}, L, U, \Pi_\mathsf{C})$
6          if $j \ge i$ then
7              if $i \ge 2$ then
8                  $(L, D, U) \leftarrow$ LinSys.SimplifyURowTurner$(i-1, i, n, m, L, D, U)$
9              $L_{ii} \leftarrow L_{ii}U_{ii}$ ; $D_{ii} \leftarrow D_{ii}U_{ii}$
10             for $t$ from $i+1$ to $m$ do
11                 $(L, D, U) \leftarrow$ LinSys.EliminateVariable$(i, t, n, L, D, U)$
12                 $(D, U) \leftarrow$ LinSys.SimplifyURowTurner$(i, t, n, D, U)$
13             $i \leftarrow i + 1$
14     $r \leftarrow i - 1$; $(L, D, U) \leftarrow$ LinSys.RemoveNullParts$(L, D, U, r)$
15     $\mathbf{A} \leftarrow (m, r, n, \Pi_\mathsf{R}, L, D, U, \Pi_\mathsf{C})$
```

In steps 8 and 12, this procedure uses Turner's GCD-free simplification method (`LinSys.SimplifyURowTurner`), described in Appendix A.4, to remove common factors from the coefficients of each equation. In step 11 of `LinSys.LDUFactor`, the auxiliary procedure `LinSys.EliminateVariable` modifies the matrix U so that each row t (from $i + 1$ to m) does not depend on the permuted variable i. On input to that procedure, invariants I1 and I3 below are valid. On output, row t of U, L_{ti}, and D_{tt} are modified so that invariants I1 and I3 are still valid, and $U_{ti} = 0$.

When the procedure reaches step 14, the rank r of the matrix is the number of nonzero rows of the triangularized matrix U. In step 14 of `LinSys.LDUFactor`, the procedure `LinSys.RemoveNullParts` discards all zero rows and columns of the resulting matrices L, D, and U.

The main loop invariant of this algorithm (I1) states that formula (6) is valid, that is, the matrices Π_R, L, D^{-1}, U and Π_C, in that order, are a factorization of the input array A. This invariant is true before and after every step of the algorithm. A secondary loop invariant (I2) says that the matrices L and U are partially triangulated and the rank of A is at least i. Specifically,

- the first $i - 1$ rows and columns of U are an upper triangular matrix with nonzero elements in the diagonal;
- the elements in rows i to m and columns 1 to $i - 1$ and $j + 1$ to n of U are zero;
- the first $i - 1$ rows and columns of L are a lower triangular matrix with nonzero elements in the diagonal;

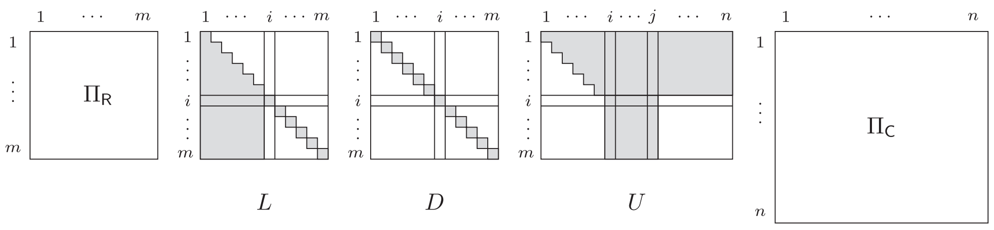

FIGURE A1 Invariant I2 of Algorithm 3. The gray parts are elements, which may be nonzero

- all elements of the diagonal of D are nonzero;
- the elements in rows 1 to $i-1$ and columns i to m of L are zero;
- rows i to m and columns i to m of L are an identity matrix.

See Figure A1. Loop invariant I2 is true before step 4 and after step 13, inclusive.

In addition, after step 5, an additional condition (invariant I3) holds: Element U_{ii} is nonzero, and the submatrix consisting of rows $i+1$ to m and columns i to j of U has at least one nonzero element in each column. The factorization is complete when $j < i$, that is, rows i to m of U are zero.

Computation cost: The outermost loop (steps 5–13) is executed r times, and the innermost loop (steps 11–12) is executed $m-i$ times in each iteration, or a total of $\Theta(mr)$ times in the asymptotic worst-case sense. The running time of the algorithm is dominated by step 5, which (as discussed in Appendix A.3 below) executes $\Theta(n)$ arithmetic operations for each call, or $\Theta(mrn)$ operations in total. Because the numbers have $\Theta(r)$ bits, the procedure takes $\Theta(mr^3n)$ time. The output matrices L, D, and U have $\Theta(r(m+n))$ elements with $\Theta(r)$ bits each.

Algorithm 4 `LinSys.Pivot`

```
Input: $i$: current row in the factoring process;
       $j$: last column of interest in $U$;
       $m$: number of rows in matrix $A$;
       $n$: number of columns in matrix $A$;
       $\Pi_R, L, D, U, \Pi_C$: factors satisfying invariants I1 and I2.
Output: $j$: last column of interest in $U$;
        $\Pi_R, L, D, U, \Pi_C$: updated factoring.
1  begin
2      $x \leftarrow i+1$; $y \leftarrow i$; $s \leftarrow i$; $t \leftarrow i$;
3      $(U, \Pi_C, j) \leftarrow$ LinSys.GatherNonzeroColumns$(i, j, U, \Pi_C)$
4      if $j < i$ then
5          return $(j, \Pi_R, L, U, \Pi_C)$
6      $g \leftarrow U_{ii}$
7      for $p$ from $i$ to $m$ do
8          for $q$ from $i$ to $j$ do
9              if $U_{pq} \neq 0$ and $(g = 0$ or $|U_{pq}| < |g|)$ then
10                 $g \leftarrow U_{pq}$; $s \leftarrow p$; $t \leftarrow q$
11     if $g = 0$ then
12         $j \leftarrow i-1$
13         return $(j, \Pi_R, L, U, \Pi_C)$
14     if $i \neq t$ then
15         $(U, \Pi_C) \leftarrow$ LinSys.SwapColumns$(i, t, U, \Pi_C)$
16     if $i \neq s$ then
17         $(\Pi_R, L, D, U) \leftarrow$ LinSys.SwapRows$(i, s, \Pi_R, L, D, U)$
```

A.2 | Pivoting

The auxiliary procedure `LinSys.Pivot` swaps the rows and columns of L, D, U (and Π_{R}, Π_{C}) so as to bring the chosen pivot element U_{ks} to position U_{ii}. The pivot is the nonzero element in rows i to m and columns i to j of U with the *smallest* absolute value—a heuristic that helps limit the bit size of matrix entries. The procedure also ensures invariant I3, namely, that all columns of that submatrix are nonzero, by permuting columns and reducing j if necessary. If all those elements are zero, the procedure returns with $j < i$.

Computation cost: The running time of each call to this procedure is dominated by the innermost loop (steps 9–10), which, in the worst case, is executed $\Theta(mn)$ times. Because the variables g and U_{pq} may have $\Theta(r)$ bits, and comparison of such numbers takes $\Theta(r)$ time, the running time is $\Theta(mrn)$.

A.3 | Variable elimination

The auxiliary procedure `LinSys.EliminateVariable` is the integer version of standard Gaussian elimination step. It replaces a specified row t of matrix U (between rows $i+1$ and m) by a suitable integer linear combination of rows i and t, so that row t becomes independent of the (permuted) variable with index i. It also modifies the matrices L and D so as to preserve the factorization invariant I1.

Algorithm 5 `LinSys.EliminateVariable`

```
Input: i: current row in the factoring process;
       t: next row in the factoring process;
       n: number of columns in matrix A;
       L, D, U: matrices of the factoring.
Output: L, D, U: updated factoring.
1 begin
2     $L_{ti} \leftarrow L_{tt} U_{ti}$
3     $D_{tt} \leftarrow D_{tt} U_{ii}$
4     for $s$ from $i+1$ to $n$ do
5         $U_{ts} \leftarrow (U_{ii}\, U_{ts}) - (U_{ti}\, U_{is})$
6     $U_{ti} \leftarrow 0$
```

Computation cost: This procedure performs $\Theta(n)$ operations on integers with $\Theta(r)$ bits. Because the elements of U have $\Theta(r)$ bits, the running time is $\Theta(r^2 n)$.

A.4 | Common divisor elimination in LDU factoring

A practical problem when using straightforward fraction-free Gauss–Jordan elimination is that the bit size of the integers generated during the elimination grows exponentially with the number of equations.[34] In general, if the input numbers have t bits (excluding sign), the final matrices will have $t \cdot 2^{r-1}$ bits, where r is the rank (the number of nonredundant equations; see Figure A2 [top]).

The size of the numbers can be greatly reduced by dividing each computed row by the GCD of its coefficients. With this optimization, we observe experimentally that the bit size of the elements in the coefficients matrix grows linearly with the number of equations (specifically, close to $t \cdot r$), instead of exponentially (see Figure A2 [bottom]).

In 1968, Bareiss[24] observed that some common factors in the computed rows are predictable and can be avoided by using complex formulas in the elimination step. In 1995, Turner[8] found a simpler method to find common factors. Specifically, after elimination of variable i from rows $i+1$ to m, the diagonal element of row $i-2$ of the partially triangulated matrix divides all the elements of the new rows i to m. Other authors have identified additional common factors in the matrices L, D, and U that can be eliminated without computing the GCD.[35] Even though Turner's algorithm does not eliminate all common factors, it still reduces the growth in the bit size of elements to linear instead of exponential,[8,24,34] and is only a little less effective than removing the GCD (see Figure A2 [bottom]).

Turner's algorithm, which we used in our implementation, has the advantage that the determinant of the original matrix U is automatically detected as the final value of U_{nn} (apart from sign changes, if pivoting is used). It is implemented by the

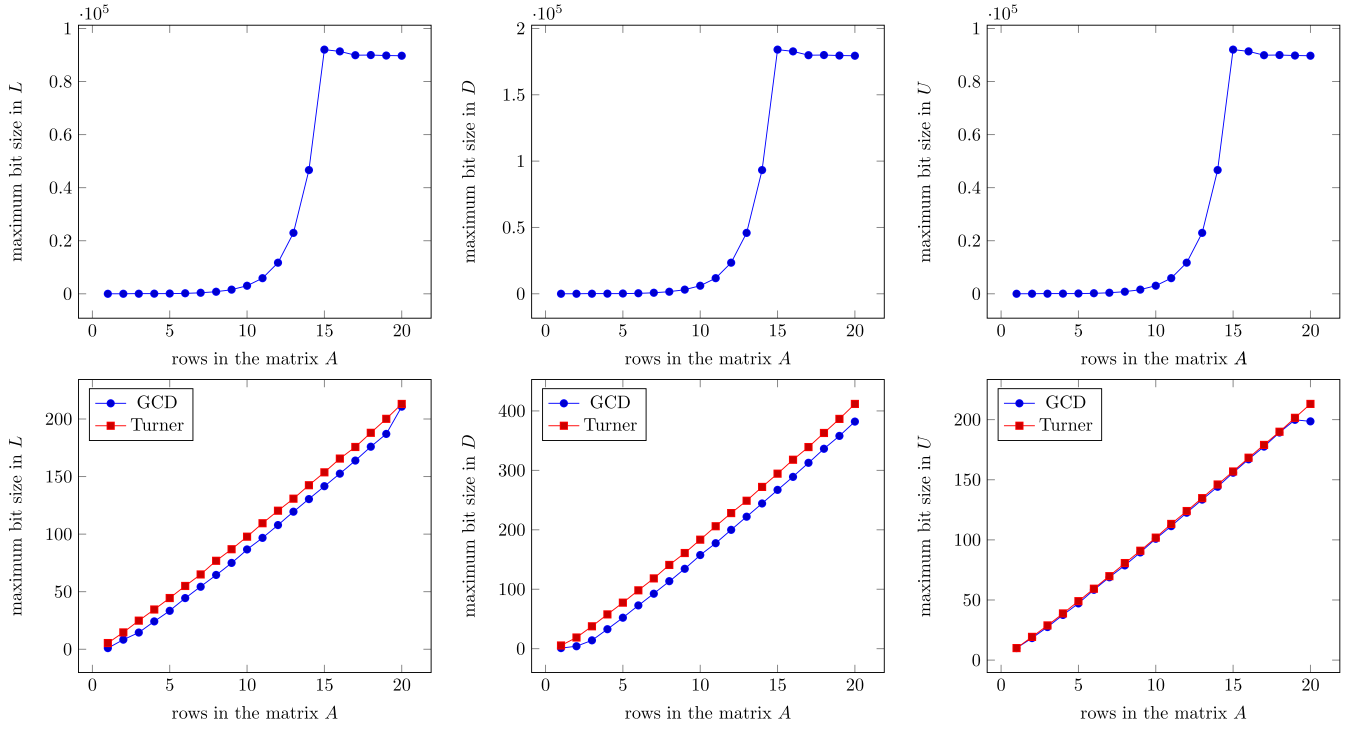

FIGURE A2 Maximum bit size of computed numbers during integer LDU factorization without any common divisor simplification (top), and with the GCD and Turner simplification methods (bottom). The horizontal axis is the number of rows m of the given matrix A. The vertical axis is the maximum bit size among all entries observed while factoring 1,000 test matrices. The elements were randomly chosen 10-bit signed integer elements; therefore, most matrices had rank $r = \min m, n$. In the top graphs, all matrices had $n = 15$ columns. Note that the bit size stops growing when m exceeds n. The slight decrease after that point is due to the greater chances of finding a small pivot as m increases. In the bottom graphs, all matrices had $n = 20$ columns

procedure `LinSys.SimplifyURowTurner` that is called on each iteration i of `ECLES.Initialize`, for each row $t > i$. Note that it does not modify the matrix L.

Algorithm 6 `LinSys.SimplifyURowTurner`

```
Input: $i$: current row in the factoring process;
       $t$: next row in the factoring process;
       $n$: number of columns in matrix $A$;
       $D, U$: factoring matrices.
Output: $D, U$: updated factoring.
1 begin
2     if $i \geq 2$ then
3         for $s \leftarrow i + 1$ to $n$ do
4             $U_{ts} \leftarrow U_{ts}/U_{i-1,i-1}$
5         $D_{tt} \leftarrow U_{tt}/U_{i-1,i-1}$
```

Computation cost: The procedure `LinSys.SimplifyURowTurner` executes $\Theta(n)$ arithmetic operations. If the numbers have $\Theta(r)$ bits, it runs in $\Theta(r^2 n)$ time.