Numer Algor
DOI 10.1007/s11075-016-0249-x

ORIGINAL PAPER

A robust approach for finding all well-separated solutions of sparse systems of nonlinear equations

Ali Baharev[1] · **Ferenc Domes**[1] · **Arnold Neumaier**[1]

Received: 24 November 2015 / Accepted: 30 November 2016

Abstract Tearing is a long-established decomposition technique, widely adapted across many engineering fields. It reduces the task of solving a large and sparse nonlinear system of equations to that of solving a sequence of low-dimensional ones. The most serious weakness of this approach is well-known: It may suffer from severe numerical instability. The present paper resolves this flaw for the first time. The new approach requires reasonable bound constraints on the variables. The worst-case time complexity of the algorithm is exponential in the size of the largest subproblem of the decomposed system. Although there is no theoretical guarantee that all solutions will be found in the general case, increasing the so-called sample size parameter of the method improves robustness. This is demonstrated on two particularly challenging problems. Our first example is the steady-state simulation a challenging distillation column, belonging to an infamous class of problems where tearing often fails due to numerical instability. This column has three solutions, one of which is missed using tearing, but even with problem-specific methods that are not based on tearing. The other example is the Stewart–Gough platform with 40 real solutions, an extensively studied benchmark in the field of numerical algebraic geometry. For both examples, all solutions are found with a fairly small amount of sampling.

Keywords Decomposition methods · Diakoptics · Large-scale systems of equations · Numerical instability · Sparse matrices · Tearing

✉ Ali Baharev
ali.baharev@gmail.com

[1] Faculty of Mathematics, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

Published online: 17 December 2016

1 Introduction

We consider square nonlinear systems

$$\begin{aligned} F(x) &= 0, \\ \underline{x} \le x &\le \overline{x}, \end{aligned} \tag{1}$$

where $F : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a continuously differentiable vector-valued function and whose Jacobian is structurally nonsingular; $\underline{x}$ and $\overline{x}$ denote the componentwise lower and upper bounds on the variables x, respectively.

From an applied point of view, it is usually not meaningful to distinguish two solutions that are too close, due to the intrinsic uncertainty of every real-life model. Therefore, the task we pose is to find a reasonably small set of points such that every solution of (1) is close to one of the points in this set. An algorithm solving this task finds in particular all well-separated solutions. Even for problems with an infinite number of solutions, only a finite number of points need to be generated. Such problems are expected to come only from defective models, and our implementation will return in such cases a diagnostic message.

Our algorithm assumes that the variables are adequately scaled. This allows us to use one of the standard norms to measure distances; unless otherwise indicated, we use the maximum norm (ℓ_∞ norm). We also assume that the bound constraints $\underline{x} \le x \le \overline{x}$ are finite and reasonable; this is needed to allow an adequate sampling of the search space. (Therefore, our method may not work well when a variable is unbounded or its upper bound is not known, and the user circumvents this by specifying a huge number such as 10^{20} as upper bound.)

Finite bound constraints are also important from an engineering perspective: These bounds often exclude those solutions of $F(x) = 0$ that either have no physical meaning or lie outside the validity of the model. In a typical technical system, all variables are bounded from below and from above. Indeed, a model is typically valid only in a finite range of the variables. Physical limitations of the devices and the design typically impose minimal and maximal geometry, load, or throughput of the devices; this implies bounds on the corresponding variables. There are also natural physical bounds, for example, the mass fractions must be between 0 and 1. In practice, not all bounds are made explicit in the model because implied bounds would be tedious for the modeler to derive by hand and to keep up-to-date when the model changes. Therefore, the bounds are conventionally not specified explicitly if they can be deduced from the model formulation; e.g., upper bounds on nonnegative variables are typically not specified if there is a constraint fixing their sum. Fully automatic and computationally cheap preprocessing can compute sufficient (but not necessarily sharp) bounds for properly specified models, see for example [13, 47, 67], or [90]. When dealing with technical systems, a variable that remains unbounded after such preprocessing is almost always a sign of a modeling mistake. We therefore assume that such a preprocessing has already been done successfully when presenting (1) to our algorithm.

Besides the (possibly infinite) number of solutions that exist, the sparsity pattern of F' decides how efficiently (1) can be solved. We therefore discuss favorable forms of sparse matrices in Section 1.1, and in Section 1.2 we present algorithms

for automatically ordering sparse matrices to the form required by the proposed algorithm. Since the paper is concerned with resolving the most serious flaw of tearing, we first briefly review it in Section 1.3. The robustness of the method will be demonstrated on problems with multiple solutions; the alternative approaches are discussed in Section 1.4. Some important specific applications are summarized in Section 1.5. The proposed method is given in Section 2, and the numerical results in Section 3. The Appendix contains practical matters, such as parameter tuning and implementation-level remarks.

1.1 Staircase triangular matrices

We call any partition of rows and columns of a square matrix A into the same number m of contiguous row blocks $R_1, \ldots, R_m$ and contiguous column blocks $C_1, \ldots, C_m$ a *block structure*. A *lower triangular block structure* (or *LTBS*) of A is a block structure that partitions A into conforming submatrices A_{jk} consisting of the entries in the contiguous rows of R_j and columns of C_k such that $A_{jk} = 0$ for $j < k$. Thus, A may be viewed as a generalization of a block lower triangular matrix to the case of possibly rectangular diagonal blocks A_{jj}. In general, this may be possible in many different ways.

We say that a square matrix $A \in \mathbb{R}^{n\times n}$ is a *staircase triangular matrix* (or has staircase triangular form) if it has no zero row or column and if the columns c_r of the last nonzero entry in row $r = 1, \ldots, n$ form a monotone increasing sequence, and the first nonzero entry in column $1, \ldots, n$ forms a monotone decreasing sequence. Staircase triangular matrices generalize staircase matrices (as surveyed, e.g., by [32]) by allowing the lower triangular part to contain additional entries, but restrict the possibilities slightly by imposing a minimal structure on the "walking profile" of the stairs. In practice, the lower triangular part is usually very sparse but often not of the form required by the traditional staircase matrices. By introducing a block boundary at every step of the stair, the staircase triangular form induces a *canonical LTBS* in a geometrically intuitive way; cf. Fig. 1. The corresponding algebraic description is as follows. By definition,

$$1 \le c_1 \le \ldots \le c_n = n.$$

Let $r_1 = 1$ and let $r_2 < \ldots < r_m$ be the list of $r = 2, \ldots, n$ with $c_{r-1} < c_r$ in increasing order. Then, the rows are partitioned into m consecutive, nonempty *row blocks*

$$R_j = \begin{cases} \{r_j : r_{j+1} - 1\} & \text{if } j = 1, \ldots, m-1, \\ \{r_j : n\} & \text{if } j = m, \end{cases}$$

and the columns into m consecutive, nonempty *column blocks*

$$C_k = \begin{cases} \{1 : c_{r_1}\} & \text{if } k = 1, \\ \{c_{r_k-1} + 1 : c_{r_k}\} & \text{if } k = 2, \ldots, m. \end{cases}$$

The shorthand $p{:}q$ is used for the index set $p, p+1, \ldots, q$, where $p \le q$. The staircase triangular form implies that the resulting block structure is lower triangular. The simplest examples of staircase triangular matrices are nonsingular lower triangular matrices where $c_r = r$, corresponding to n blocks of size 1.

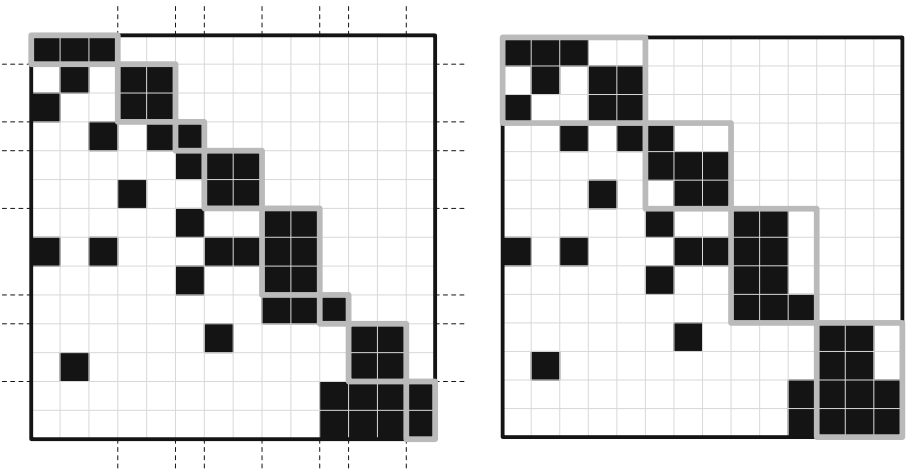

Fig. 1 *Left*: A staircase triangular matrix whose canonical LTBS has fully dense diagonal blocks. *Right*: another *LTBS* structure for the same matrix

In addition to the canonical LTBS, we may obtain LTBSs with fewer blocks by arbitrarily merging one or more consecutive row blocks and the corresponding column blocks, cf. Fig. 1.

1.2 Ordering to staircase triangular form

The ordering algorithms typically used assume that the input matrix is structurally nonsingular. This is revealed by the Dulmage–Mendelsohn decomposition [24–27, 45, 66, Ch. 6], and [18, Ch. 7]. This decomposition is a standard procedure, and efficient computer implementations are available, for example `HSL_MC79` from the [44]. For a structurally nonsingular square matrix, the Dulmage–Mendelsohn decomposition always produces a block lower triangular matrix with structurally nonsingular square blocks on the diagonal. These diagonal blocks are irreducible. The ordering algorithms that we discuss orders each of these diagonal blocks to staircase triangular form; the correspondingly ordered full matrix will be automatically staircase triangular as well.

Practical algorithms for ordering sparse matrices to staircase triangular form include the Hellerman–Rarick family of ordering algorithms [24, 29, 42, 43], and the algorithms of [78, 79]. An efficient computer implementation of the Hellerman–Rarick algorithms is `MC33` from the [44]. Although there are subtle differences among the various ordering algorithms, they all fit the same pattern when viewed from a high level of abstraction [31]: Algorithm 1 is the fundamental algorithm. The various ordering algorithms typically assume that the matrix is irreducible and only seem to differ in the lookahead step to break ties on line 3. Figure 2 shows an intermediate stage of the algorithm. Any version of Algorithm 1 will produce a staircase triangular matrix whose canonical diagonal blocks are fully dense.

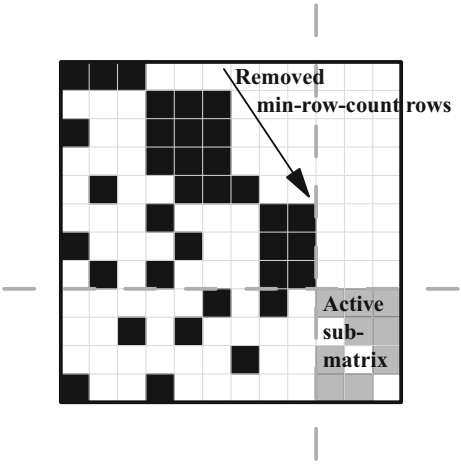

Fig. 2 An intermediate stage of Algorithm 1

Algorithm 1 The fundamental ordering algorithm by Fletcher and Hall (1993)

Input: A, a sparse irreducible matrix
Output: A permuted to staircase triangular form
// Active submatrix: The remaining part of A not ordered yet
// Row count: The number of nonzero entries in the active submatrix of a given row
1 set A as the active submatrix
2 **repeat**
3 find a row in the active submatrix with minimum row count
4 put all columns which intersect this row to the left and consider them as removed
5 update row counts in the active submatrix
6 put all rows with zero row count to the top and consider them as removed
7 **until** *all rows and columns are removed*

Some of the above cited papers on the Hellerman–Rarick algorithms and the Stadtherr–Wood algorithms include numerical results, indicating that this decomposition method is effective. Further numerical evidence shows that this approach usually gives favorable decompositions for problems from diverse fields: Performance results on 692 test problems are given in [9] for our own ordering algorithm (inspired by the fundamental Algorithm 1). These problems were taken from the COCONUT Benchmark [69], which covers a variety of applications, e.g., chemical engineering, computational chemistry, civil engineering, robotics, economics, multicommodity network flow, process design, stability analysis, VLSI chip design, and portfolio optimization.

1.3 Traditional tearing

Tearing dates back to the 1930s [50, 81] and has been widely adapted across many engineering fields since: State-of-the-art steady-state and dynamic simulation environments all implement some variant of tearing, see for example [1], Dymola [17], JModelica [55], or OpenModelica [61]. The applicability of tearing is not limited to a particular engineering discipline: It is generic, and it is used in all

state-of-the-art Modelica simulators to model "complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents" [54]. Tearing is also referred to as diakoptics or sequential modular approach depending on the discipline. When dealing with distillation columns, tearing is called stage-to-stage or stage-by-stage calculations.

We say that a square matrix A has *bordered block triangular form* if it can be written as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

with block triangular A_{11} and square A_{22}, the latter typically of fairly small size.

Numerous ordering algorithms are available to permute a spares matrix fully automatically into this form. One way of doing it is to first find a staircase ordering as in Section 1.2 and then to move those columns to the far right that spoil the upper left block triangular form. This produces a bordered block triangular form such that the diagonal blocks in A_{11} are dense. Conversely, suppose a method is available to order a matrix to bordered block triangular form with the property that the diagonal blocks in A_{11} are structurally nonsingular. Such methods are surveyed in [8] and [9]. Then, we can reorder the diagonal blocks to staircase form: We reorder each block of the whole matrix accordingly and then move the resulting border to the far left to get a matrix in staircase form.

In the traditional setup, the bound constraints in (1) are ignored, and the variables and equations are permuted with a suitable ordering algorithm such that the sparsity pattern of the Jacobian is in bordered block lower triangular form with structurally nonsingular square blocks on the diagonal, see Fig. 3.

The variables in (1) are partitioned as

$$x = \begin{pmatrix} x_0 \\ \vdots \\ x_N \end{pmatrix} \tag{2}$$

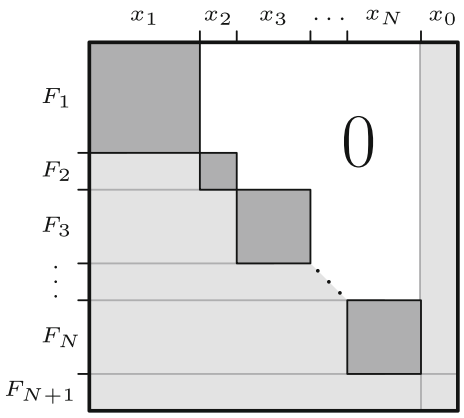

Fig. 3 Bordered block lower triangular form with structurally nonsingular square blocks on the diagonal

into subvectors $x_i \in \mathbb{R}^{d_i} \quad (i = 0 \ldots N)$, so that $n = d_0 + \cdots + d_N$. Similar to the variables, F is partitioned as

$$F(x) = \begin{pmatrix} F_1(x) \\ \vdots \\ F_{N+1}(x) \end{pmatrix} \tag{3}$$

into subfunctions $F_i(x) \in \mathbb{R}^{d_i} (i = 1 \ldots N + 1)$. Since the system (1) is square, the trailing dimension must be $d_{N+1} := d_0$. For any bordered block lower triangular matrix, only variables from subvectors $x_0, \ldots, x_i \ (i \leq N)$ can appear in $F_i(x)$:

$$F_i(x) = F_i(x_0, x_1, \ldots, x_i) \text{ for } i = 1, \ldots, N. \tag{4}$$

Equations (2)–(4) describe the block sparsity pattern shown in Fig. 3. In practice, the lower triangle is sparse.

Given the bordered block lower triangular form, the diagonal blocks are eliminated one-by-one from $i = 1$ to N, and $F_{N+1}(x)$ is considered as a function of x_0 only: $G(x_0)$, where $G : \mathbb{R}^{d_0} \mapsto \mathbb{R}^{d_0}$. Then, $G(x_0) = 0$ is solved for x_0, the only variables not eliminated. The solution vector x_0, together with the eliminated variables, give the solution to the original problem (1).

The motivation behind tearing is to save time. The diagonal blocks are typically small and dense, and specialized methods can be used to efficiently eliminate them. This can lead to substantial saving in execution time compared to solving (1) without any decomposition.

The user has to provide an initial guess for each variable when a gradient-based local solver is used to solve (1) directly. With tearing, the user has to provide an initial guess for x_0 only, which saves significant amount of time for the user. Tearing is relatively easy to implement in a component-oriented simulator such as Dymola, JModelica, or OpenModelica, and it is usually robust provided that $G(x_0)$ is well-conditioned.

The biggest flaw of tearing was recognized early [16, 80]: It can show extreme sensitivity to the initial guess for x_0, since numerical sensitivity can build up while the blocks are eliminated along the diagonal. In such cases, the Jacobian $G'(x_0)$ of the reduced system is extremely ill-conditioned, even if the Jacobian $F'(x)$ of the original system is well-conditioned. This in turn has a negative impact on the convergence properties of the methods used for solving $G(x_0) = 0$. Although several attempts were made to mitigate this issue, see for example [93, 94] and [37], it has never been resolved satisfactorily.

The sensitivity issue can become so severe that, with all the intermediate variables $x_1, \ldots x_N$ eliminated, there may not be any machine representable vector for x_0 such that $G(x_0) = 0$ is satisfied with acceptable numerical accuracy. For example, the distillation column computed in Section 3.1 is intractable with traditional tearing although the Jacobian of each solution has a condition number estimate of $< 10^9$, so that one expects from a stable method several accurate digits.

1.4 General-purpose methods for finding multiple solutions

Multistart methods try to find all solutions by starting a gradient-based local solver from multiple starting points. Guarantees regarding finding all solutions depend on the placement of the starting points. Perhaps the simplest method for bound constrained problems is the grid search: The constraint violation is evaluated at each point of a fine grid, and the best points are used as starting points for local optimization. Finding all solutions with probability one can be achieved by making the grid sufficiently dense. This naive approach is only effective in very low dimensions as the number of grid points grows exponentially with the dimension of the problem.

Multistart methods are applicable to large-scale systems of equations by giving up on the strong guarantees that, for example, grid search would provide. In practice, sophisticated approaches are used to place the starting points, for example, constraint consensus (to reduce infeasibility in a computationally cheap way) and clustering (to separate basins of attraction) by Smith et al. [73–75] or stochastic methods (especially population-based metaheuristics) [83]. These methods strike a balance between speed and robustness.

Another set of methods that are often successfully used to solve large-scale problems—especially if the objective function can be cheaply computed—are *evolutionary algorithms*, e.g., CMA-ES [2], if they are combined with local optimization starting from the most promising points found. Other methods that are based on similarities to natural processes (ant colony [22], particle swarm [28], etc.) can be used in a similar way.

For solving systems of equations with multiple solutions, *homotopy methods* are extensively used, especially for systems of polynomial equations. The reader is referred to [10, 11, 77, 95] for the latest developments. Mature and robust software implementations for solving polynomial systems are, for example, `Bertini` [11, 12], and `PHCpack` [86, 87]. Homotopy methods with problem-specific homotopy maps were successfully applied to large-scale industrial problems with transcendental equations [21, 51, 85]. This approach with problem-specific homotopy maps is also suitable for component-based (also referred to as component-oriented) modeling of large, complex, and heterogeneous technical systems [70; 72, Ch. 8–10]: Once the models of the devices (components) are implemented and put into a library, practically no understanding of probability—one homotopy method is required from the end-user.

Spatial branch-and-bound methods recursively split the search space into smaller parts and eliminate those parts that cannot lead to a solution better than the currently best known one. Unfortunately, their worst-case performance tends to grow exponentially with the dimension of the problem since they perform exhaustive search. (For non-convex functions, global optimization is NP-hard.) The applicability of branch-and-bound methods currently seems to be limited to fairly low-dimensional problems in the general case. Successful enclosure methods in chemical engineering include interval arithmetic [41], McCormick relaxations [53, 68], affine arithmetic [6, 76], and αBB [40].

1.5 Important specific applications

The problem of solving nonlinear systems of equations arises in the daily engineering practice, e.g., when consistent initial values for differential algebraic equation (DAE) systems are sought [63, 84], or when solving steady-state models of technical systems. A steady-state solution can be used as a consistent initial set of the DAE system [48].

Even though mature equation-based component-oriented modeling environments are available, e.g., Modelica [34, 52, 82] for multi-domain modeling of heterogeneous complex technical systems and [36] ASCEND [65] and EMSO [62] for chemical process modeling, simulation, and optimization, the steady-state initialization is still not satisfactorily resolved in the general case. Often, steady-state initialization failures can only be resolved in very cumbersome ways [3, 60, 71, 72, 89], involving user-provided good initial values for the variables. The proposed algorithm aims to eliminate this tedious process by generating good initial values fully automatically.

2 Proposed algorithm

Here, we give a formal presentation with pseudo-code through Sections 2.1–2.4; the Java source code is available online in the supplementary material [7], together with an illustrative numerical example where the steps of the algorithm are illustrated with several figures.

2.1 Input of the proposed method

The input of the proposed method is (1), together with an LTBS of its Jacobian, obtained with appropriate preprocessing. For efficiency reasons one should enforce that the diagonal blocks are structurally nonsingular and contain no zero row or column; Algorithm 1 always delivers a staircase triangular matrix whose canonical LTBS has this property. As outlined in Section 1.2, numerical evidence indicates that Algorithm 1 tends to give favorable decompositions for problems from diverse fields, that is, the size of the largest block tends to be small. The worst-case time complexity of the proposed method grows exponentially with the size of the largest block.

Let N denote the number of blocks of the input *LTBS*. Unlike in tearing, the variables are now partitioned along the block boundaries as

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} \tag{5}$$

into N subvectors $x_i \in \mathbb{R}^{n_i} (i = 1 \ldots N)$, so that $n = n_1 + \cdots + n_N$. Similarly, F is partitioned along the block boundaries as

$$F(x) = \begin{pmatrix} F_1(x) \\ \vdots \\ F_N(x) \end{pmatrix} \tag{6}$$

into N vector-valued subfunctions $F_i(x) \in \mathbb{R}^{m_i} (i = 1 \ldots N)$, and $n = m_1 + \cdots + m_N$. The motivation behind requiring an LTBS as input is that then only variables from the subvectors $x_1, \ldots, x_i$ $(i \leq N)$ can appear in $F_i(x)$:

$$F_i(x) = F_i(x_1, \ldots, x_i) \text{ for } i = 1, \ldots, N. \tag{7}$$

We view this as a cycle-free sequence of subproblems that will be handled sequentially.

2.2 The idea in a nutshell

The algorithm builds up a *point cloud*, i.e., a set of vectors satisfying

$$F_{1:i}(x_{1:i}) = 0 \tag{8}$$

by processing the blocks one-by-one along the diagonal of the LTBS. The algorithm sketched so far would have exactly the same numerical issues that tearing has. Compared to tearing, the only significant difference up to this point is that a set of points is propagated through the blocks and not just a single point. But working with a point cloud allows us to counteract conditioning problems. Inspired by our earlier results for the univariate case [5], this is achieved by redistributing the sample points after each block. This redistribution step strives to ensure in each iteration that the sample of the solution set of (8) remains representative within the bound constraints, in the sense that (a) no point of the solution set is too far from the sample, and (b) the points in the sample are well-separated.

These goals are achieved on a best effort basis. In each iteration, we first insert additional points into the sample with Algorithm 4 which involves robust sampling and sensitivity analysis; the aim of this is to achieve goal (a). After inserting additional points to the sample, its size is assumed to be greater than the user-defined sample size M_i; therefore, we have to drop some of the points from the sample until M_i points remain. (If, due to some pathological situation, the sample size is still less than or equal to M_i, we simply skip the the rest of the redistribution step and do not drop any of the points.) To achieve goal (b), we choose M_i points from the point cloud with a greedy algorithm: We choose the least infeasible point (the point with smallest $\|F_{1:i}(x_{1:i})\|_\infty$) as the first point. We iteratively continue choosing a point from

the point cloud that are furthest away from all already chosen points, breaking ties arbitrarily. When the desired sample size M_i is reached, we drop all points from the sample that have not been chosen. The sample size M_i has the biggest effect on the robustness and execution time of the method: Increasing the sample size is expected to improve robustness but at the cost of increased computational costs.

The difference of our procedure to the naive way of directly sampling the solution set of $F_{1:i}(x_{1:i}) = 0$ within the bound constraints is that in the latter approach, the volume to be sampled grows exponentially with the dimension $p := \dim x_{1:i}$, hence good sampling is prohibitively expensive once p gets large (ultimately $p = n$). The proposed method avoids this growth by sampling only at the blocks along the diagonal: The volume to be sampled grows exponentially only with the largest block size, which is usually significantly smaller than p. The biggest computational savings of the proposed method are therefore achieved here.

2.3 Pseudo-code of the proposed algorithm

The algorithm is presented in high-level pseudo-code in Algorithm 2; the reader is referred to Appendix C for practical matters, such as the choice and effect of the used-provided parameters.

When forming the subvector $v_{p:q}$ of a vector v, $p{:}q$ is cropped appropriately if necessary; that is, invalid indices are ignored. The index set $p{:}q$ is considered empty if $p > q$, and the expression $v_{p:q}$ is a valid subvector of v that has no elements. We write

$$x \oplus y := \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^{m+n} \tag{9}$$

for the concatenation of two vectors $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$.

Algorithm 2 The proposed algorithm

Input: A problem instance as defined by (1) and (5)–(7)
Parameters : M: at most M_i points are kept in iteration i
ε: tolerated constraint violation
Output: A set of initial points for starting a local solver

1 Initialize the set $S^{(0)}$ with M_0 empty vectors
2 **for** $i = 1$ **to** N **do**
3 **foreach** $x_{1:i-1} \in S^{(i-1)}$ **do**
4 Call **Algorithm 3** to solve $F_i(x_{1:i-1}, x_i) = 0$ for x_i
5 Add the resulting solutions $x_{1:i}$ to $S^{(i)}$
6 **if** $i < N$ **then**
 // The redistribution step is performed on lines 7–9
7 Call **Algorithm 4** with i, $S^{(i-1)}$, $S^{(i)}$, and add the result to $S^{(i)}$
8 Remove all $x_{1:i}$ from $S^{(i)}$ for which $\|F_{1:i}(x_{1:i})\|_\infty \geq \varepsilon$
9 Keep only the most distant M_i points from the remaining $S^{(i)}$
10 **return** $S^{(N)}$

Algorithm 3 Block elimination

Input: The block $F_i(x_{1:i-1}, x_i) = 0$ to be solved for x_i, given $x_{1:i-1}$; bound constraints $\underline{x}_i \le x_i \le \overline{x}_i$
Output: The extended vector $x_{1:i}$ with which $F_i(x_{1:i}) \approx 0$
// Either the ℓ_1 or the ℓ_2 norm is used depending on the solver, see Appendix B.1
1 **if** $\dim F_i \ge \dim x_i$ **then**
  // Square or overdetermined block
2 Solve $\min_{x_i} \|F_i(x_{1:i-1}, x_i)\|$ s.t. $\underline{x}_i \le x_i \le \overline{x}_i$
3 **return** x_i
4 **else**
  // Underdetermined block; due to implementation artifacts of the solvers, we
  // have to fix the least influential variables of x_i until the block becomes square
5 Pick k vectors for x_i with robust sampling, and store them in T
  // Latin hypercube sampling is used; $k = 5$ by default
6 Let $d = \dim x_i - \dim F_i$
7 $U \leftarrow \{\,\}$
8 **foreach** $x_i \in T$ **do**
9 Apply sensitivity analysis to find the *subvector* $(x_i)_J$ of the d least influential variables
  // J is currently identified by QR factorization of $F_i'(x)$ with column pivoting
10 Solve $\min_{y_i} \|F_i(x_{1:i-1}, y_i)\|$ s.t. $(y_i)_J = (x_i)_J$, $\underline{x}_i \le y_{,i} \le \overline{x}_i$
11 Add $x_{1:i-1} \oplus y_i$ to U
12 **return** U

The robust sampling and sensitivity analysis in Algorithm 3 was necessary to overcome implementation artifacts of the solvers: In the underdetermined case, the solver may place many of the solution vectors near to each other in a particular subspace due to implementation artifacts. Making the underdetermined subsystem square by fixing the least influential variables $(x_i)_J$ proved to be sufficient to resolve this issue. The goal of the local sensitivity analysis performed on line 9 is to order the variables x_i according to their influence on $\|F_i(x_{1:i})\|_2$ with $x_{1:i-1}$ fixed. The index set J on line 9 denotes the indices of the least influential variables. QR factorization with column pivoting is performed on the Jacobian of $F_i(x_{1:i})$; the resulting permutation vector gives the desired ordering [35, p. 591].

2.4 Adding additional sample points

The goal of Algorithm 4 is to produce additional sample points. The newly introduced $x_{i-h:i}$ parts of the forced new points are separated from each other on a best effort basis. The new values for x_i are obtained with Latin hypercube sampling on line 3, which is expected to give good separation of these new x_i parts. Local sensitivity analysis is performed to find the index set J of the least influential variables (more on this shortly). Only these least influential variables of x_i are ultimately fixed on line 13(s.t. $(y_i)_J = (x_i)_J$), so that infeasibility can be most likely repaired on line 13 by minimizing the constraint violation. The fixed $(x_i)_J$ parts are also separated from each sample point $x_i^{(S)}$ (received from Algorithm 2 as input): Those $(x_i)_J$ parts that already have nearby neighbors in the sample $S^{(i)}$ are discarded on line 10.

Algorithm 4 Computing additional solutions

Input: index i of the current subproblem; for $k = i-1, i$, the set $S^{(k)}$ of sample points $x_{1:k}$ for which $F_{1:k}(x_{1:k}) \approx 0$
Parameters : p: number of forced new points
δ: threshold for two points being too close
h: maximal number of subproblems to be considered
Output: A set of points U containing new values for $x_{1:i}$ for which $F_{1:i}(x_{1:i}) \approx 0$

// Do nothing if $x_{1:i-h-1}$ is still an empty vector, i.e., we are early in the iteration
1 **if** $h+1 < i$ **then**
2 **return** $\varnothing$

// Force new values for a subvector of x_i
// Latin hypercube sampling is used in the current implementation
3 Pick p vectors for x_i with robust sampling, and add them to set R
4 Let $d = \max(1, \dim x_i - \dim F_i)$
5 $T \leftarrow \{\ \}$
6 **foreach** $x_{1:i-1} \in S^{(i-1)}, x_i \in R$ **do**
7 Apply sensitivity analysis to find the *subvector* $(x_i)_J$ of the d least influential variables
  // J is currently identified by QR factorization of $F_i'(x)$ with column pivoting
8 Add $(x_i)_J$ to T

// Discard those $(x_i)_J$ that already have nearby neighbors in $S^{(i)}$
9 **foreach** $(x_i)_J \in T$, $x_i^{(S)} \in S^{(i)}$ **do**
10 Remove $(x_i)_J$ from T if $\|(x_i)_J - (x_i^{(S)})_J\|_\infty < \delta$

// Calculate the missing history of $(x_i)_J$
// Brute-force search for the best approximating $x_{1:i-h-1} \in S^{(i-1)}$
11 $U \leftarrow \{\ \}$
12 **foreach** $(x_i)_J \in T$, $x_{1:i-h-1} \in S^{(i-1)}$ **do**
  // Resolve the last h subproblems to find the missing part $x_{i-h:i}$
  // Either the ℓ_1 or the ℓ_2 norm is used depending on the solver, see Appendix B.1
13 Solve $\min_{y_{i-h:i}} \|F_{i-h:i}(x_{1:i-h-1}, y_{i-h:i})\|$ s.t. $(y_i)_J = (x_i)_J$, $\underline{x}_{i-h:i} \le y_{i-h,i} \le \overline{x}_{i-h:i}$
14 Add $x_{1:i-h-1} \oplus y_{i-h:i}$ to U
15 **return** U

The cardinality of J is chosen to be $\max(1, \dim x_{i-h:i} - \dim F_{i-h:i})$, that is, if the subsystem is underdetermined, we treat it as we did in Algorithm 3, see Section 2.3. However, if the subsystem is square (which is a common case) or even overdetermined, we still have to perturb at least one of the variables; otherwise, the solution vector $x_{1:i}$ is most likely in the input $S^{(i)}$ already, and we will not insert any new point into our sample. (If the subsystem has multiple solutions in $x_{i-h:i}$; then, it can happen that we find a new point without perturbation.)

The entire $x_{1:i-1}$ part of the partial solution should not be recomputed on line 13, not even with inter- or extrapolations: that would potentially make the complexity of the entire algorithm $O(n^2)$ where n is the number of variables in (1). This is the reason why $x_{1:i-h-1}$ is left unchanged, and only the last h subproblems are considered on line 13.

3 Numerical results and discussion

The benchmark problems have been coded in the AMPL modeling language [33] and are available in the online supplementary material [7]. A short summary of the problems is given in Table 1.

3.1 Multiple steady-states in homogeneous azeotropic distillation

The steady-state simulation of distillation columns belongs to an infamous family of problems where tearing is often inapplicable due to numerical instability [21]. Our first example is therefore a challenging distillation column where tearing fails. This column has three solutions, one of which is missed even with problem-specific methods (that are not based on tearing).

3.1.1 Background of the problem

The model equations are the MESH equations: The component material balance (M), vapor-liquid equilibrium (E), summation (S), and heat balance (H) equations are solved. The liquid phase activity coefficient is computed from the Wilson equations. The model and its parameters correspond to the Auto model [39], except for the number of stages N and the feed stage location $N_F = N/2$. The specifications are the feed composition (methanol–methyl butyrate–toluene), the reflux ratio, and the vapor flow rate.

There are three steady-state branches: two stable steady-state branches and an unstable branch; this was experimentally verified in an industrial pilot column operated at finite reflux [23, 39]. Multiple steady-states can be predicted by analyzing columns with infinite reflux and infinite length [14, 38, 64]. These predictions for infinite columns have relevant implications for columns of finite length operated at finite reflux.

3.1.2 Published numerical results with continuation methods

Both the conventional inside-out procedure [15] and the simultaneous correction procedure [57] were reported to miss the unstable steady-state solution, see [85] and [46] (all input variables specified; output multiplicity). However, all steady-state branches

Table 1 Short summary of the benchmark problems

Name	Number of variables	Nonzeros	Blocks	Largest block size	Solutions	Note
Azeotropic Distillation	$4N$	$21N - 6$	N	5	3	Transcendental equations
Stewart–Gough Platform	9	57	3	3	40	Polynomial equations

were computed either with the AUTO software package [20] or with an appropriate continuation method [39, 46, 85]. The initial estimates were carefully chosen with the ∞/∞ analysis [14, 38], and special attention was paid to the turning points and branch switching.

3.1.3 Obtaining the lower triangular block structure

A distillation column consists of stages; partitioning the Jacobian along the stage boundaries gives the blocks. The natural order of the stages directly yields the LTBS by virtue of the internal physical layout of distillation columns. As a consequence, no preprocessing was necessary before applying the proposed method. The sparsity pattern of the Jacobian is shown in Fig. 4.

3.1.4 A note on plotting the solution vectors

To understand the displayed results of the next section, we recall how chemical engineers traditionally plot the solutions. Once a solution to the model equations is available, it is plotted as follows. A slice of the solution vector is created first: A subset of variables is selected that has the same physical meaning in each block. For example, if the variables corresponding to the temperature are selected at each block, a slice is obtained, the so-called temperature column profile, etc. Then, the block index is plotted against the selected value of the variable in this block, see on the left of Fig. 5.

In the chemical engineering literature, not a sequence of discrete points are plotted but a piecewise linear curve passing through these points, and not the block but the stage index is used, see on the right of Fig. 5. For the columns considered in this paper, block i corresponds to stage $N - i + 1$, that is, the stages are numbered in reverse order compared to the blocks. It must be emphasized that the solution is a vector of real numbers and not a continuous curve.

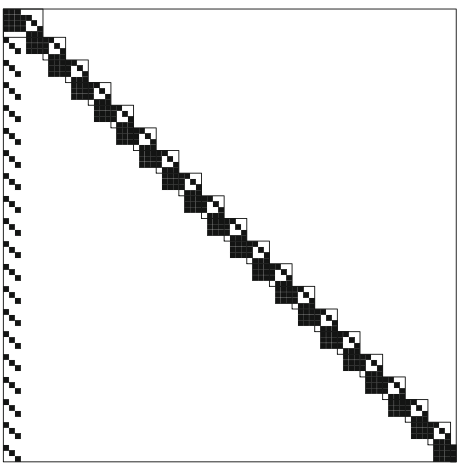

Fig. 4 Sparsity pattern of the Jacobian with the natural block structure as defined by the stages. The figure is prepared for $N = 20$ stages

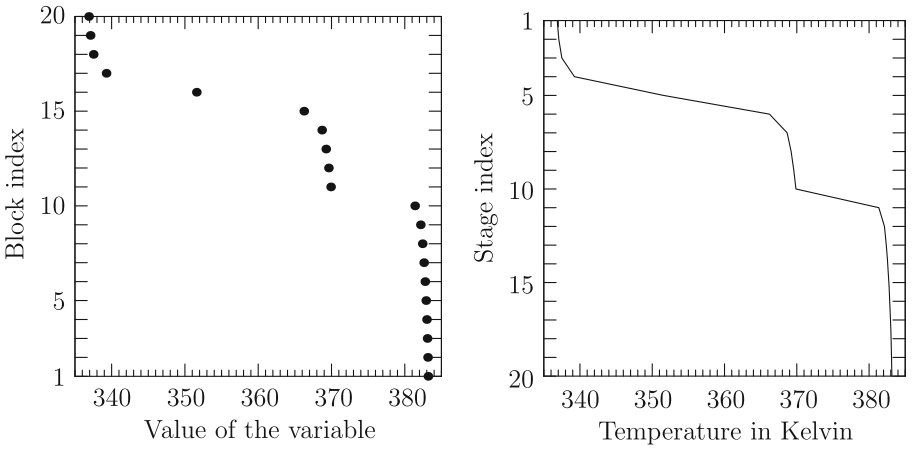

Fig. 5 Plot of a solution vector slice. *Left*: block index vs. the value of the variable in the slice. *Right*: plot of the same slice as seen in the chemical engineering literature, the so-called temperature column profile. The stages are numbered in reverse order compared to the blocks. It must be emphasized that the slice is a vector of real numbers and not a continuous curve

3.1.5 Our numerical results

With the appropriate parameter settings, all three steady-state solutions are found when IPOPT [91] is run from the starting points generated with the proposed method. As for parameter tuning of the proposed algorithm, the reader is referred to Appendix C; the effects of varying h is shown in Table 2.

For the purposes of demonstration, several plots are given for the column with $N = 20$ stages; the column with 40 stages is too long to be appropriate for illustration. Figure 6 shows the three steady-state solutions and those starting points that are the closest to them.

Figure 7 illustrates an intermediate step of the algorithm, the extension of the partial column profiles when moving from stage 11 to stage 10, that is, the result of executing the nested loop in Algorithm 2 for each point in the sample. Block i

Table 2 The effect of varying h while the initial sample size M_0 is kept fixed at 25

	$N = 20$			$N = 40$		
h	Solutions	Time in Alg. 2	Total time	Solutions	Time in Alg. 2	Total time
1	2	43	51	1	104	115
2	3	67	82	3	237	323
3	3	114	130	3	384	473

The problem being solved is the azeotropic distillation problem of Section 3.1; it has three solutions for both $N = 20$ and $N = 40$. The time is measured in seconds. Time in Alg. 2 is the time needed to generate the starting points; the total time also includes running IPOPT from each point

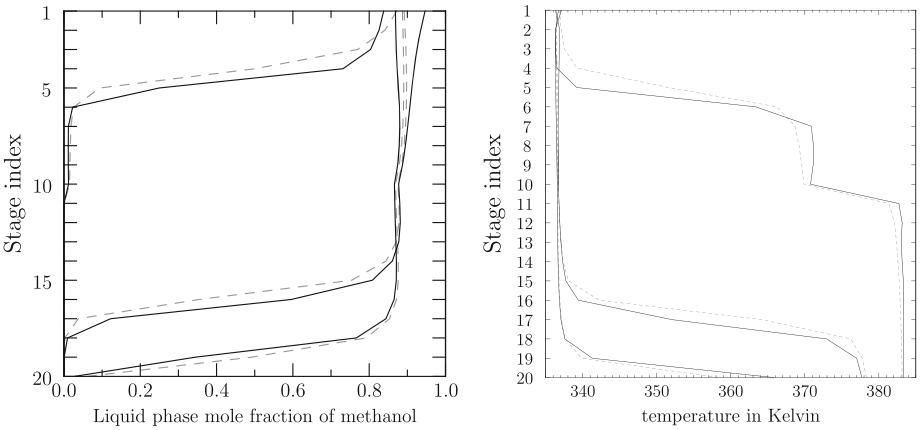

Fig. 6 The three steady-state solutions (*dashed gray lines*) and those generated starting points (*solid black lines*) that are the closest to them. The gradient-based solver IPOPT converges to the nearest solution when started from the corresponding starting point. Column description in Section 3.1

corresponds to stage $N - i + 1$; the computations are performed bottom up, starting at the reboiler (block 1, stage 20).

Figure 8 shows the effect of the distinguishing feature of the method, the redistribution. If the redistribution is disabled, the proposed method eventually boils down to the tearing (stage-by-stage method). Figure 8 is computed stage-by-stage, exactly from the same bulk composition used for Fig. 6. Two out of the three steady-state solutions are lost without the redistribution step.

Figure 9 shows one execution of the redistribution step at stage 10. New points are forced into the sample, compare with Fig. 7; then, only the most distant points are kept.

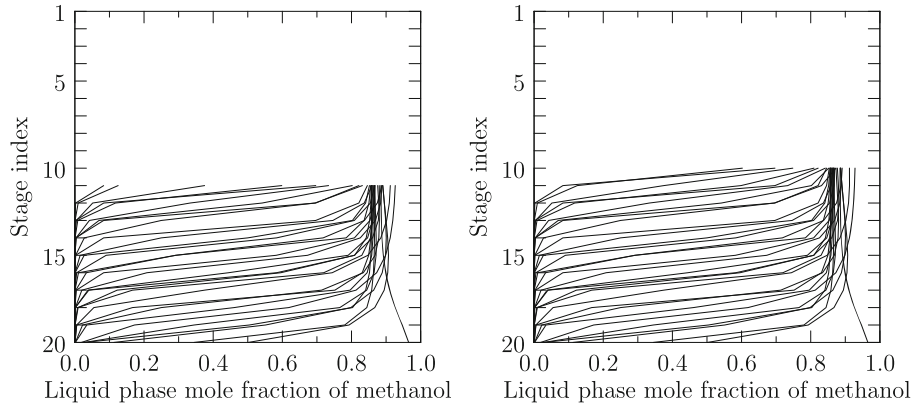

Fig. 7 Extending the partial solutions as moving from stage 11 (*left*) to stage 10 (*right*); see line 4 in Algorithm 2. Several partial column profiles are built stage-by-stage, starting from a variety of bulk compositions. Column description in Section 3.1

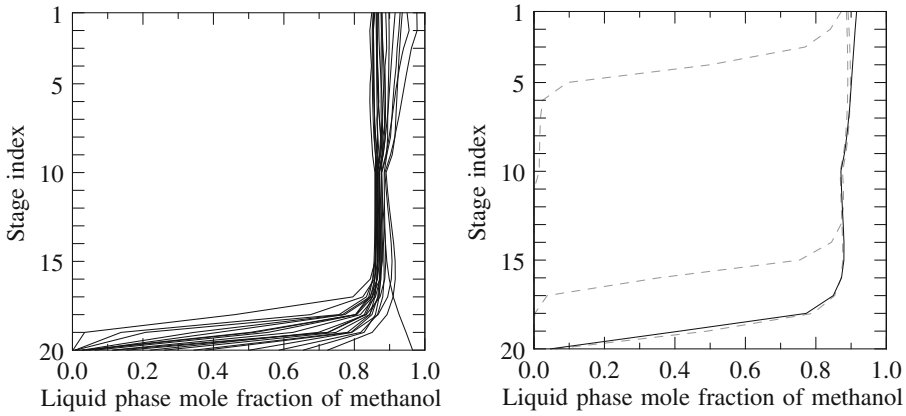

Fig. 8 The lack of redistribution. *Left*: Several column profiles are built as in the traditional stage-by-stage calculation, starting from a variety of bulk compositions. *Right*: The *dashed gray lines* show the three different steady-state solutions, two of them do not have any good approximation and therefore are not discovered. Column description in Section 3.1

3.2 Stewart–Gough platform with 40 real postures

The Stewart–Gough platform consists of two rigid bodies that are connected by six rods attached via spherical joints; it is a type of parallel-link robotic device. This problem is an extensively studied benchmark with homotopy methods, see, e.g, [77, Sec. 7.7] or [11, Sec. 6.3]. [19] published parameters with which a given Stewart–Gough platform has 40 real postures. The model equations with these parameters are available from the database of PHCpack, maintained by [88]; it is the `stewgou40` benchmark. The latest version of PHCpack [86, 87] is v2.4.25, released on 31 Aug 2016. With the hardware specified at beginning of Section 3, it requires 56.5 seconds

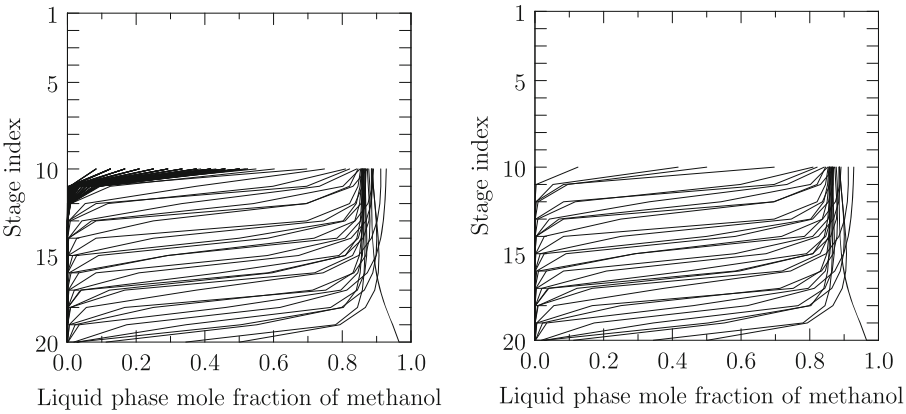

Fig. 9 The redistribution step at stage 10. *Left*: New values are inserted for mole fractions of methanol, cf. Fig. 7. *Right*: Only the most distant column profile extensions are kept. Column description in Section 3.1

for solving this particular benchmark in its so-called blackbox mode for non-experts. It is very likely that in the hands of an expert, this software can solve the problem faster. Here, we show how all the 40 real solutions can be found with the proposed method. Even though the problem characteristics that favor our method (large, sparse problem) are not present, our method performs reasonably on this benchmark.

3.2.1 Our results

Some preprocessing is necessary before applying the proposed method. As all variables are coordinates of unit vectors and therefore must be in the interval $[-1, 1]$, the latter defines the bound constraints.

The Jacobian can be permuted fully automatically into staircase triangular form. The optimal pattern, shown in Fig. 10, is found on the root node of the search tree of the method of [9], in less than 5 ms. However, in this particular case, using such an ordering algorithm is an overkill: One can easily bring the Jacobian into staircase triangular form with pen and paper by making the first equation the fourth. As it can be seen in Fig. 10, the sparsity pattern has very little structure that the proposed method can exploit: It only has three blocks on the diagonal of the canonical LTBS.

As for setting the initial sample size M_0, the following strategy gives a reasonable approach. The user should set M_0 to be ten times the expected number of solutions, and at least 25. If more than the expected number of solutions were found, M_0 should be doubled iteratively, and the proposed method run again with this larger initial sample size. We keep doubling M_0 and rerunning the algorithm until all previously found solutions were found *and* no new solutions, or we reach a pre-defined limit for M_0.

Let us pretend that we expect only one solution, and we start with an initial sample size of 25 accordingly. Since 21 solutions are found, see Table 3, we iteratively double the initial sample size and rerun the proposed algorithm, until all previously found solutions are found with $M_0 = 400$ and no new solutions. The entire procedure takes 54.3 s. If we know a priori that there are at most 40 real solutions, see for example

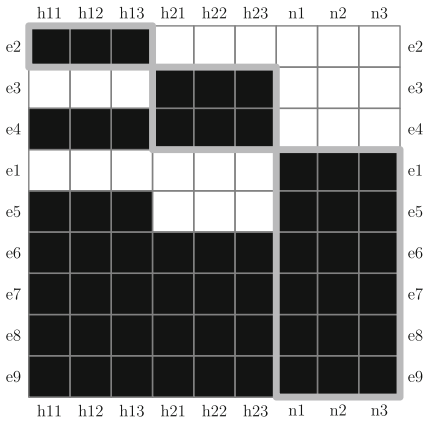

Fig. 10 The sparsity pattern of the Stewart–Gough benchmark problem (`stewgou40`) in staircase triangular form and with canonical LTBS

Table 3 The effect of iteratively doubling the initial sample size M_0, while keeping h fixed at 2

M_0	Solutions found	Accumulated solutions	Time (s)	Cumulative time (s)
25	21	21	3.3	3.3
50	30	33	4.9	8.2
100	35	39	7.3	15.5
200	39	40	12.5	28.0
400	40	40	26.3	54.3

The problem being solved is the Stewart–Gough platform with 40 solutions

[30, 49, 56], or [92], we can stop at $M_0 = 200$, requiring 28.0 s. Alternatively, if we anticipate 40 solutions and start with $M_0 = 40 \times 10$ accordingly, we finish in 26.3 s.

Acknowledgments Open access funding provided by Austrian Science Fund (FWF). The research was funded by the Austrian Science Fund (FWF): P23554, and P27891-N32. Support by the Austrian Research Promotion Agency (FFG) under project number 846920 is thankfully acknowledged. We are grateful to the anonymous reviewers for feedback that lead to improvements in the presentation of the goals, the background, and the implementation of the proposed method.

Appendix

A software and hardware environment

All computations have been carried out with the following hardware and software configuration. Processor: Intel(R) Core(TM) i5-3320M CPU at 2.60 GHz; operating system: Ubuntu 14.04.3 LTS with 3.13.0-67-generic kernel; Java (TM) SE Runtime Environment (build 1.8.0_101-b13) and Java HotSpot (TM) 64-Bit Server VM (build 25.101-b13, mixed mode). The implementation is sequential (single-threaded); cf. Appendix D.

B Local solvers used

B.1 Solving the nonlinear subproblems

The actual norm in the auxiliary Algorithms 3 and 4 depends on the solver being used to solve the optimization problem. In our implementation, the user can choose between IPOPT and LMBOPT [59]; the former uses the ℓ_1 norm and the latter the ℓ_2 norm. The choice of the solver for solving the subproblems had no effect on the robustness of the algorithm in our numerical experience.

B.2 Solving the original system from the generated starting points

Our solver of choice for solving the input problem from the generated starting points is IPOPT; however, due to implementation artifacts, this solver is not appropriate for the Stewart–Gough benchmark of Section 3.2. Even when supplied with a solution, IPOPT first wanders off then back while it is reducing the dual infeasibility. As a consequence, it already starts losing some of the solutions when provided with starting points that were obtained from the true solutions by applying small random perturbations of at most 0.004 in the maximum norm. The minimum ℓ_∞ distance between any two solutions is approximately 0.155. Other solvers, e.g., LMBOPT and VA27 from [44], do not have any difficulty with such small random perturbations. We therefore used LMBOPT for this particular benchmark. However, this solver squares the condition number, and the estimated condition number of the Jacobian at some of the solutions is of order 10^6. This necessitated some post-processing: The solutions returned by LMBOPT had to be polished with textbook Newton iteration.

B.3 Notes on the execution time

Implementing the proposed method is a major undertaking; the most difficult part is the implementation of the function and Jacobian evaluation of the subproblems. In order to reduce the implementation effort, we use at present our existing Java code that was originally created for researching inclusion algebras [58, Ch. 2.2] with abstract datatypes and our existing Java wrappers for the IPOPT and LMBOPT solvers for large-scale and sparse problems, that are comparably inefficient for small, dense subproblems to which they are applied here. Although this code reuse dramatically reduced the time needed to implement a prototype, the resulting software is unacceptably slow. Profiling shows that the execution times given in the present paper mostly reflect the performance flaws of our current research prototype. We therefore started a complete rewrite from scratch [4]: We are currently reimplementing the function and Jacobian evaluations in the C programming language, and for solving the subproblems, we are switching to VA27. Unlike IPOPT and LMBOPT, the latter solver is tailored for small and dense problems. This reimplementation is still an ongoing process.

C Parameter tuning

Algorithms 2 and 4 have parameters that were explicitly indicated in the pseudocode but were left unspecified; here, we discuss (i) the influence of these parameters on the robustness and execution time of the method, (ii) how they were set in our numerical experiments, (which we also consider reasonable default settings), (iii) and outline appealing future research directions to set them adaptively and automatically.

In our numerical experience, the sample size M is the most important, and h in the redistribution step is the second most important parameter of the method with respect

to their influence on robustness and execution time. The sample size M determines the resolution of the solution set. With the parameter h, one controls the number of subproblems among which the constraint violation of a newly forced point is spread. Increasing either one of these parameters is expected to increase robustness at the expense of increased computational effort; this is illustrated by Tables 2 and 3.

C.1 Setting the sample size

We used $M_i = c \cdot i + M_0$ in our numerical experiments, where c and M_0 are constants; it is left to the user to specify M_0 and c. The choice $M_i = i + 25$ proved to be appropriate for the most difficult distillation column considered in this paper and, for all other, easier problems in our test set (that are not discussed here) with one exception. The `stewgou40` benchmark, to be discussed in Section 3.2, has 40 solutions, and it was necessary to iteratively double M_0 to achieve a sufficient resolution of the solution set, see Table 3.

One can probably find a parametric formula that gives a reasonable default value for M_i as a function of the key characteristics such as the size of the largest block, the size of block i, and the number of blocks N. The parameters of such a formula with the right qualitative form should be fitted and cross-validated by running the algorithm on a large benchmark set, consisting of diverse test problems. This is the subject of future research.

C.2 Setting *h*

The second most important parameter of the proposed method is h in the redistribution step: After forcing a new point into the sample, the last h blocks are simultaneously resolved to minimize the constraint violation resulting from the forceful insertion. If h is chosen too small (for example $h = 1$), we may fail to reduce the constraint violation sufficiently, and the new forced points will be discarded in Algorithm 2 on line 8; this can lead to poor resolution of the solution set, and some of the solutions will be lost eventually. Setting h to a large value ($h \gg 5$) spoils the performance, since the last h blocks are resolved simultaneously, and the increased computational effort does not yield any visible improvement in robustness. In the current implementation, it is left to the user to specify h. The $h = 4$ choice works for all the test problems in our test set.

Similarly to M, it is subject of future research to work out a rule for choosing a default value for h. In particular, the following adaptive rule seems appealing. In our numerical experience, the remaining constraint violation in Algorithm 4, line 13 decreases rapidly as h is increased. Therefore, the algorithm could start with a small value for h at each block (for example $h = 2$), and increment it until either the tolerated constraint violation ϵ of Algorithm 2 or a user-defined cutoff for h (for example $h = 5$) is reached. The optimization problem in Algorithm 4 on line 13 can be quickly resolved after incrementing h if the gradient-based solver is started from the previous solution. Other, more sophisticated adaptive rules are also possible.

C.3 Setting the remaining parameters

With M and h fixed, the other parameters have negligible effect on the robustness and performance of the method in our experience; therefore, practically no effort was made to tune these other parameters. For the purposes of documentation only, the following settings were used: $\epsilon = 2/M_0$ in Algorithm 2, and $p = \left|S^{(i)}\right|$, and $\delta = 2/M_0$ in Algorithm 4.

D Parallelization

Profiling shows that most of the time is spent solving the subproblems at the blocks as expected: In Algorithm 2 in the loop on lines 3–5, and in Algorithm 4 in the loop on lines 12–14. Since the optimization problems solved in these loops are completely independent, this means that they can be solved in parallel; the bottleneck of the computations is parallelizable. We have not implemented this improvement; our current implementation is still sequential (single-threaded) because the underlying implementation is unfortunately not thread-safe.

References

1. Aspen Technology, Inc (2009) Aspen Simulation Workbook, Version Number: V7.1. Burlington, MA, USA. EO and SM Variables and Synchronization, p. 110
2. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. In: 2005. The 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1769–1776. IEEE (2005)
3. Bachmann, B., Aronßon, P., Fritzson, P.: Robust initialization of differential algebraic equations. In: 1st International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, vol. 2007, pp. 151–163, Linköping University Electronic Press; Linköpings universitet, Linköping Electronic Conference Proceedings (2007)
4. Baharev, A.: https://sdopt-tearing.readthedocs.io, Exact and heuristic methods for tearing (2016)
5. Baharev, A., Neumaier, A.: A globally convergent method for finding all steady-state solutions of distillation columns. AIChE J. **60**, 410–414 (2014)
6. Baharev, A., Kolev, L., Rév, E.: Computing multiple steady states in homogeneous azeotropic and ideal two-product distillation. AIChE J. **57**, 1485–1495 (2011)
7. Baharev, A., Domes, F., Neumaier, A.: Online supplementary material of the present manuscript. http://www.baharev.info/finding_all_solutions.html (2016a)
8. Baharev, A., Schichl, H., Neumaier, A.: Decomposition methods for solving nonlinear systems of equations, http://reliablecomputing.eu/baharev_tearing_survey.pdf, submitted (2016b)
9. Baharev, A., Schichl, H., Neumaier, A.: Ordering matrices to bordered lower triangular form with minimal border width, http://reliablecomputing.eu/baharev_tearing_exact_algorithm.pdf, submitted (2016c)
10. Bates, D.J., Hauenstein, J.D., Sommese, A.J.: Efficient path tracking methods. Numer. Algorithm. **58**(4), 451–459 (2011)
11. Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W.: Numerically Solving Polynomial Systems with Bertini, Software, Environments and Tools, vol 25. SIAM, Philadelphia, PA (2013)
12. Bates, D.J., Newell, A.J., Niemerg, M.: BertiniLab: A MATLAB interface for solving systems of polynomial equations. Numer. Algorithm. **71**(1), 229–244 (2016)

13. Beelitz, T., Frommer, A., Lang, B., Willems, P.: Symbolic–numeric techniques for solving nonlinear systems. PAMM **5**(1), 705–708 (2005)
14. Bekiaris, N., Meski, G.A., Radu, C.M., Morari, M.: Multiple steady states in homogeneous azeotropic distillation. Ind. Eng. Chem. Res. **32**, 2023–2038 (1993)
15. Boston, J.F., Sullivan, S.L.: A new class of solution methods for multicomponent, multistage separation processes. Can. J. Chem. Eng. **52**, 52–63 (1974)
16. Christensen, J.H.: The structuring of process optimization. AIChE J. **16**(2), 177–184 (1970)
17. Dassault Systèmes, A.B.: Dymola—Dynamic Modeling Laboratory. User Manual. Vol. 2., Ch. 8. Advanced Modelica Support (2014)
18. Davis, T.A.: Direct methods for sparse linear systems. In: Higham, N.J. (ed.) Fundamentals of Algorithms. SIAM, Philadelphia, USA (2006)
19. Dietmaier, P.: The Stewart-Gough platform of general geometry can have 40 real postures, pp. 7–16. Springer, Netherlands, Dordrecht (1998)
20. Doedel, E.J., Wang, X.J., Fairgrieve, T.F.: AUTO94: Software for continuation and bifurcation problems in ordinary differential equations. Technical Report CRPC-95-1, Center for Research on Parallel Computing, California Institute of Technology, Pasadena CA 91125 (1995)
21. Doherty, M.F., Fidkowski, Z.T., Malone, M.F., Taylor, R. Perry's Chemical Engineers' Handbook, 8th edn., p. 33. McGraw-Hill Professional (2008). chapter 13
22. Dorigo, M., Birattari, M., Stützle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)
23. Dorn, C., Güttinger, T.E., Wells, G.J., Morari, M.: Stabilization of an unstable distillation column. Ind. Eng. Chem. Res. **37**, 506–515 (1998)
24. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct methods for sparse matrices. Clarendon Press, Oxford (1986)
25. Dulmage, A.L., Mendelsohn, N.S.: Coverings of bipartite graphs. Can. J. Math. **10**, 517–534 (1958)
26. Dulmage, A.L., Mendelsohn, N.S.: A structure theory of bipartite graphs of finite exterior dimension. Trans. Royal Soc. Can. Sec. **3**(53), 1–13 (1959)
27. Dulmage, A.L., Mendelsohn, N.S.: Two Algorithms for Bipartite Graphs. J. Soc. Ind. Appl. Math. **11**, 183–194 (1963)
28. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995. MHS'95, pp. 39-43. IEEE (2002)
29. Erisman, A.M., Grimes, R.G., Lewis, J.G., Poole, W.G.J.: A structurally stable modification of Hellerman-Rarick's P^4 algorithm for reordering unsymmetric sparse matrices. SIAM J. Numer. Anal. **22**, 369–385 (1985)
30. Faugère, J.C., Lazard, D.: Combinatorial classes of parallel manipulators. Mech Mach. Theory **30**(6), 765–776 (1995)
31. Fletcher, R., Hall, J.A.J.: Ordering algorithms for irreducible sparse linear systems. Ann. Oper. Res. **43**, 15–32 (1993)
32. Fourer, R.: Staircase matrices and systems. SIAM Rev. **26**(1), 1–70 (1984)
33. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, Brooks/Cole, USA (2003)
34. Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press (2004)
35. Golub, G.H., Van Loan, C.F. Matrix computations, 3rd edn. The Johns Hopkins University Press, Baltimore, USA (1996)
36. gPROMS. Process Systems Enterprise Limited, gPROMS. http://www.psenterprise.com, [Online; accessed 17-November-2015] (2015)
37. Gupta, P.K., Westerberg, A.W., Hendry, J.E., Hughes, R.R.: Assigning output variables to equations using linear programming. AIChE J.ournal **20**(2), 397–399 (1974)
38. Güttinger, T.E., Morari, M.: Comments on multiple steady states in homogeneous azeotropic distillation. Ind. Eng. Chem. Res. **35**, 2816–2816 (1996)
39. Güttinger, T.E., Dorn, C., Morari, M.: Experimental study of multiple steady states in homogeneous azeotropic distillation. Ind. Eng. Chem. Res. **36**, 794–802 (1997)
40. Guzman, Y.A., Hasan, M.M.F., Floudas, C.A.: Computational comparison of convex underestimators for use in a branch-and-bound global optimization framework. In: Rassias, T.M., Floudas, C.A., Butenko, S. (eds.) Optimization in Science and Engineering, pp. 229–246. Springer, New York, USA (2014)

41. Gwaltney, C.R., Lin, Y., Simoni, L.D., Stadtherr, M.A.: Interval methods for nonlinear equation solving applications. Wiley, Chichester, UK (2008)
42. Hellerman, E., Rarick, D.C.: Reinversion with preassigned pivot procedure. Math Programm. **1**, 195–216 (1971)
43. Hellerman, E., Rarick, D.C.: The partitioned preassigned pivot procedure (P^4). In: Rose, D.J., Willoughby, R.A. (eds.) Sparse Matrices and their Applications, The IBM Research Symposia Series, pp. 67–76. Springer, US (1972)
44. HSL: A collection of Fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk (2016)
45. Johnson, D.M., Dulmage, A.L., Mendelsohn, N.S.: Connectivity and reducibility of graphs. Can. J. Math. **14**, 529–539 (1962)
46. Kannan, A., Joshi, M.R., Reddy, G.R., Shah, D.M.: Multiple-steady-states identification in homogeneous azeotropic distillation using a process simulator. Ind. Eng. Chem. Res. **44**, 4386–4399 (2005)
47. Kearfott, R.B.: Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems. Computing **47**(2), 169–191 (1991)
48. Kröner, A., Marquardt, W., Gilles, E.: Getting around consistent initialization of DAE systems? Comput. Chem. Eng. **21**(2), 145–158 (1997)
49. Lazard, D.: On the representation of rigid-body motions and its application to generalized platform manipulators, pp. 175–181. Springer, Netherlands, Dordrecht (1993)
50. Lewis, W.K., Matheson, G.L.: Studies in distillation. Ind. Eng. Chem. **24**, 494–498 (1932)
51. Malinen, I., Tanskanen, J.: Homotopy parameter bounding in increasing the robustness of homotopy continuation methods in multiplicity studies. Comput. Chem. Eng. **34**(11), 1761–1774 (2010)
52. Mattsson, S., Elmqvist, H., Otter, M.: Physical system modeling with Modelica. Control Eng. Pract. **6**, 501—-510 (1998)
53. Mitsos, A., Chachuat, B., Barton, P.I.: McCormick-based relaxations of algorithms. SIAM J. Optim. **20**(2), 573–601 (2009)
54. Modelica: Modelica and the modelica association. https://www.modelica.org/, [Online; accessed 10-October-2016] (2016)
55. Modelon, A.B.: JModelica.org User Guide, verison 1.17. http://www.jmodelica.org/page/236, [Online; accessed 10-October-2016] (2016)
56. Mourrain, B.: The 40 ëgeneric positions of a parallel robot. In: Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation, ACM, NY, USA, ISSAC '93, pp. 173–182, doi:10.1145/164081.164120 (1993)
57. Naphthali, L.M., Sandholm, D.P.: Multicomponent separation calculations by linearization. AIChE J. **17**, 148–153 (1971)
58. Neumaier, A.: Interval methods for systems of equations. Cambridge University Press, Cambridge (1990)
59. Neumaier, A., Azmi, B.: LMBOPT – A limited memory method for bound-constrained optimization, http://www.mat.univie.ac.at/neum/ms/lmbopt.pdf, in preparation (2017)
60. Ochel, L.A., Bachmann, B.: Initialization of equation-based hybrid models within OpenModelica. In: 5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (University of Nottingham), pp. 97–103. Linköping University Electronic Press; Linköpings universitet, Linköping Electronic Conference Proceedings, Nottingham, Uk (2013)
61. OpenModelica: Openmodelica user's guide. https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/omchelptext.html, [Online; accessed 10-October-2016] (2016)
62. De P Soares, R., Secchi, A.R.: EMSO: A new environment for modelling, simulation and optimisation. In: Computer Aided Chemical Engineering, vol. 14, pp. 947–952. Elsevier (2003)
63. Pantelides, C.C.: The consistent initialization of differential-algebraic systems. SIAM J. Sci. Stat. Comput. **9**(2), 213–231 (1988)
64. Petlyuk, F.B.: Distillation theory and its application to optimal design of separation units. Cambridge University Press, Cambridge, UK (2004)
65. Piela, P.C., Epperly, T.G., Westerberg, K.M., Westerberg, A.W.: ASCEND: An object-oriented computer environment for modeling and analysis: the modeling language. Comput. Chem. Eng. **15**(1), 53–72 (1991)
66. Pothen, A., Fan, C.J.: Computing the block triangular form of a sparse matrix. ACM Trans. Math. Softw. **16**, 303–324 (1990)

67. Schichl, H., Neumaier, A.: Interval analysis on directed acyclic graphs for global optimization. J. Global Optim. **33**, 541–562 (2005)
68. Scott, J.K., Stuber, M.D., Barton, P.I.: Generalized McCormick relaxations. J. Glob. Optim. **51**(4), 569–606 (2011)
69. Shcherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.H., Nguyen, T.V.: Benchmarking global optimization and constraint satisfaction codes. In: Bliek, C., Jermann, C., Neumaier, A. (eds.) Global Optimization and Constraint Satisfaction, Lecture Notes in Computer Science, vol. 2861, pp. 211–222. Springer, Berlin Heidelberg (2003). http://www.mat.univie.ac.at/neum/glopt/coconut/Benchmark/Benchmark.html
70. Sielemann, M.: Device-oriented modeling and simulation in aircraft energy systems design. Dissertation, TU Hamburg, Hamburg (2012). https://doi.org/10.15480/882.1111
71. Sielemann, M., Schmitz, G.: A quantitative metric for robustness of nonlinear algebraic equation solvers. Math. Comput. Simul. **81**(12), 2673–2687 (2011)
72. Sielemann, M., Casella, F., Otter, M.: Robustness of declarative modeling languages: improvements via probability-one homotopy. Simul. Modell. Pract. Theory **38**, 38–57 (2013)
73. Smith, L.: Improved placement of local solver launch points for large-scale global optimization. PhD thesis, Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE). Carleton University, Ontario, Canada (2011)
74. Smith, L., Chinneck, J., Aitken, V.: Constraint consensus concentration for identifying disjoint feasible regions in nonlinear programmes. Optim. Methods Softw. **28**(2), 339–363 (2013a)
75. Smith, L., Chinneck, J., Aitken, V.: Improved constraint consensus methods for seeking feasibility in nonlinear programs. Comput. Optim. Appl. **54**(3), 555–578 (2013b)
76. Soares, R.P.: Finding all real solutions of nonlinear systems of equations with discontinuities by a modified affine arithmetic. Comput. Chem. Eng. **48**, 48–57 (2013)
77. Sommese, A.J., Wampler II, C.W.: The numerical solution of systems of polynomials arising in engineering and science. World Scientific (2005)
78. Stadtherr, M.A., Wood, E.S.: Sparse matrix methods for equation-based chemical process flowsheeting–I: Reordering phase. Comput. Chem. Eng. **8**(1), 9–18 (1984a)
79. Stadtherr, M.A., Wood, E.S.: Sparse matrix methods for equation-based chemical process flowsheeting–II: Numerical Phase. Comput. Chem. Eng. **8**(1), 19–33 (1984b)
80. Steward, D.V.: Partitioning and tearing systems of equations. J. Soc. Indust. Appl. Math. Ser. B Numer. Anal. **2**(2), 345–365 (1965)
81. Thiele, E., Geddes, R.: Computation of distillation apparatus for hydrocarbon mixtures. Ind. Eng. Chem. **25**, 289–295 (1933)
82. Tiller, M.: Introduction to physical modeling with Modelica. Springer Science & Business Media (2001)
83. Ugray, Z., Lasdon, L., Plummer, J., Glover, F., Kelly, J., Martí, R.: Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. INFORMS J. Comput. **19**(3), 328–340 (2007). doi:10.1287/ijoc.1060.0175
84. Unger, J., Kröner, A., Marquardt, W.: Structural analysis of differential-algebraic equation systems —– theory and applications. Comput. Chem. Eng. **19**(8), 867–882 (1995)
85. Vadapalli, A., Seader, J.D.: A generalized framework for computing bifurcation diagrams using process simulation programs. Comput. Chem. Eng. **25**, 445–464 (2001)
86. Verschelde, J.: Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. ACM Trans. Math. Softw. **25**(2), 251–276 (1999)
87. Verschelde, J.: Polynomial homotopy continuation with phcpack. ACM Commun. Comput. Algebra **44**(3/4), 217–220 (2011)
88. Verschelde, J.: The database of polynomial systems. http://homepages.math.uic.edu/jan/demo.html (2016)
89. Vieira, R.Jr., E. B.: Direct methods for consistent initialization of DAE systems. Comput. Chem. Eng. **25**(9–10), 1299–1311 (2001)
90. Vu, X.H., Schichl, H., Sam-Haroud, D.: Interval propagation and search on directed acyclic graphs for numerical constraint solving. J. Glob. Optim. **45**(4), 499 (2008)
91. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Programm. **106**, 25–57 (2006)
92. Wampler, C.W.: Forward displacement analysis of general six-in-parallel sps (Stewart) platform manipulators using soma coordinates. Mech. Mach. Theory **31**(3), 331–337 (1996)

93. Westerberg, A.W., Edie, F.C.: Computer-aided design, Part 1 enhancing convergence properties by the choice of output variable assignments in the solution of sparse equation sets. Chem. Eng. J. **2**, 9–16 (1971a)
94. Westerberg AW, Edie FC: Computer-aided design, part 2 an approach to convergence and tearing in the solution of sparse equation sets. Chem. Eng. J. **2**(1), 17–25 (1971b)
95. Wu, W., Reid, G.: Finding points on real solution components and applications to differential polynomial systems. In: Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation, ACM, NY, USA, ISSAC '13, pp. 339–346 (2013)