## A manifold-based approach to sparse global constraint satisfaction problems

## Ali Baharev, Arnold Neumaier \& Hermann Schichl

## Journal of Global Optimization

An International Journal Dealing with
Theoretical and Computational Aspects of Seeking Global Optima and Their
Applications in Science, Management and Engineering

ISSN 0925-5001
Volume 75
Number 4

Volume 75, No. 4 - December 2019

## Journal of Global Optimization

J Glob Optim (2019) 75:949-971 DOI 10.1007/s10898-019-00805-x

4) Springer

Your article is published under the Creative Commons Attribution license which allows users to read, copy, distribute and make derivative works, as long as the author of the original work is cited. You may selfarchive this article on your own website, an institutional repository or funder's repository and make it publicly available immediately.

# A manifold-based approach to sparse global constraint satisfaction problems 

Ali Baharev ${ }^{1}$ (D) Arnold Neumaier ${ }^{1}$ • Hermann Schichl ${ }^{1}$

Received: 7 November 2018 / Accepted: 4 July 2019 / Published online: 12 July 2019
© The Author(s) 2019


#### Abstract

We consider square, sparse nonlinear systems of equations whose Jacobian is structurally nonsingular, with reasonable bound constraints on all variables. We propose an algorithm for finding good approximations to all well-separated solutions of such systems. We assume that the input system is ordered such that its Jacobian is in bordered block lower triangular form with small diagonal blocks and with small border width; this can be performed fully automatically with off-the-shelf decomposition methods. Five decades of numerical experience show that models of technical systems tend to decompose favorably in practice. Once the block decomposition is available, we reduce the task of solving the large 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 proposed method resolves this issue with the novel backsolve step. We study the effect of the decomposition on a sequence of challenging problems. Beyond a certain problem size, the computational effort of multistart (no decomposition) grows exponentially. In contrast, thanks to the decomposition, for the proposed method the computational effort grows only linearly with the problem size. It depends on the problem size and on the hyperparameter settings whether the decomposition and the more sophisticated algorithm pay off. Although there is no theoretical guarantee that all solutions will be found in the general case, increasing the so-called sample size hyperparameter improves the robustness of the proposed method.


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

[^0]
## 1 Introduction

### 1.1 Aims

We consider square nonlinear systems

$$
\begin{align*}
& F(x)=0 \\
& \underline{x} \leq x \leq \bar{x} \tag{1}
\end{align*}
$$

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 $\bar{x}$ denote the vector of lower and upper bounds, respectively on the components of $x$. 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 good approximations to all well-separated solutions. Even for problems with an infinite number of solutions, only a finite number of points need to be generated.

The task that we just posed is computationally intractable in general; we have to make further assumptions. We assume that (1) has already been ordered such that its Jacobian is in bordered block lower triangular form with small blocks and with small border width; the formal definition of bordered block lower triangular forms is given in Sect. 1.3. In Sect. 1.4 we give references how (1) can be ordered to the desired bordered block lower triangular form fully automatically and efficiently. We argue in Sect. 1.5 why the models of technical systems tend to decompose favorably in practice, and why the proposed method is expected to be useful across many engineering fields, e.g., mechanical, electrical, chemical, and aerospace engineering. Further (less limiting) assumptions are given in Sect. 1.6. The last one of our assumptions is given in Sect. 3, after the overview of the proposed method; this is necessary for better understanding of this particular assumption.

### 1.2 Terminology

We refer to the number $\operatorname{dim} x$ of components of a vector $x$ as its dimension. The structural rank of a matrix $A$ is the maximum number of nonzero entries that can be permuted onto the diagonal with suitable row and column permutations. (It is also known as the maximal size of a transversal, of a maximum assignment, or of a maximum matching in the bipartite sparsity graph of $A$.) The structural rank is an upper bound on the numerical rank of $A$. $A$ is nonsingular for some numerical values of its nonzero entries if and only if it is possible to permute the rows and columns of $A$ in such a way that the diagonal is zero free. Such a matrix is called structurally nonsingular.

In an engineering application it is usually not meaningful to distinguish two solutions that are too close due to the intrinsic uncertainty of every real-life model. We therefore call a set $P$ of points well-separated if, for any distinct points $p, q \in P$, the distance $\|p-q\|_{2}$ is above a small threshold $\delta$ specified by the user, for example $\delta=10^{-4}$.

Array slicing notation. The shorthand $p: q$ is used for the ordered index set $p, p+$ $1, \ldots, q$, where $p \leq q$. 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 components. In case of block vectors, the shorthand $v_{i: k}$ is used for a block vector with consecutive blocks $v_{j}(j=i: k)$.

A point cloud is a set of scattered points, intended to approximate a manifold.

### 1.3 Bordered block lower triangular forms

The so-called bordered block lower triangular form is illustrated in Fig. 1, and formally defined as follows. The variables in (1) are partitioned as

$$
x=\left(\begin{array}{c}
x_{1}  \tag{2}\\
\vdots \\
x_{N+1}
\end{array}\right)
$$

into subvectors $x_{i} \in \mathbb{R}^{d_{i}} \quad(i=1 \ldots N+1)$, so that $n=d_{1}+\cdots+d_{N+1}$. For notational convenience, let

$$
\begin{equation*}
x_{0}:=x_{N+1} . \tag{3}
\end{equation*}
$$

Similarly to the variables, $F$ is partitioned as

$$
F(x)=\left(\begin{array}{c}
F_{1}(x)  \tag{4}\\
\vdots \\
F_{N+1}(x)
\end{array}\right)
$$

into subfunctions $F_{i}(x) \in \mathbb{R}^{d_{i}}(i=1 \ldots N+1)$. The Jacobian of the diagonal blocks, $F_{i}^{\prime}\left(x_{i}\right)$ (see Fig. 1) are required to be structurally nonsingular.

For any bordered block lower triangular matrix, only variables from subvectors $x_{0}, \ldots, x_{i}$ $(i \leq N)$ can appear in $F_{i}(x)$ :

$$
\begin{equation*}
F_{i}(x)=F_{i}\left(x_{0}, x_{1}, \ldots, x_{i}\right) \text { for } i=1, \ldots, N . \tag{5}
\end{equation*}
$$

The motivation behind requiring a bordered block lower triangular form is that we can decompose the input system of equations (1) into a cycle-free sequence of subproblems, where the sequence is given by (5).

By construction, the diagonal blocks are structurally nonsingular. We refer to the set $S$ of arguments where some block is singular as the singular set of the system. The structural nonsingularity implies that $S$ has measure zero. For arguments $x$ outside this set, all blocks are nonsingular, and $F_{1: i}\left(x_{0: i}\right)=0(i=1, \ldots, N)$ implicitly defines a (possibly disconnected) $d_{0}$-dimensional manifold in $\mathbb{R}^{d_{i}}$, where $d_{i}=\operatorname{dim} x_{0: i}$. We refer to the full solution set of this subsystem for arguments within the original bounds as the solution manifold associated with the bordered block lower triangular form. (If the singular set $S$ is nonempty, this is a manifold only in a generalized sense since it has singularities at the points of $S$, e.g. self-crossings and cusps.) In our algorithm we resolve this manifold through a coarse discretization by a point cloud.

Equations (2)-(5) describe the block sparsity pattern shown in Fig. 1. This decomposition exists for any structurally nonsingular matrix. As we will see in Sect. 3, the usefulness of a particular block decomposition depends primarily on the border width

$$
\begin{equation*}
d:=d_{0}=d_{N+1}, \tag{6}
\end{equation*}
$$

and secondarily on the largest block size

$$
\begin{equation*}
b:=\max d_{1}, \ldots, d_{N}, \tag{7}
\end{equation*}
$$

from the point of view of the proposed method.

Fig. 1 Bordered block lower triangular form with structurally nonsingular square blocks on the diagonal, see Eqs. (1)-(5). The square blocks along the diagonal (dark gray squares) must be structurally nonsingular. This decomposition can be computed fully automatically for any matrix that is structurally nonsingular, see Sect. 1.4. In engineering applications, the light gray area is typically sparse, and the border width and block sizes tend to be small, see Sect. 1.5


### 1.4 Creating the desired block decomposition automatically

Sparse matrix ordering algorithms are a well-researched subject with a vast literature; we only mention some key points and references here. Both the Jacobian of (1) and the square blocks along the diagonal are required to have full structural row rank. The structural rank is revealed by the Dulmage-Mendelsohn decomposition (Dulmage and Mendelsohn [18-20], Johnson et al. [30], Duff et al. [17, Ch. 6], Pothen and Fan [43], and Davis [11, Ch. 7]). This decomposition is a standard procedure, and efficient computer implementations are available, for example HSL_MC79 from the HSL [29]. Practical ordering algorithms are applied next; these include the Hellerman-Rarick family of ordering algorithms [17,21,27,28], and the algorithms of Stadtherr and Wood [45,46]. An efficient computer implementation of the Hellerman-Rarick algorithms is MC33 from the HSL [29]. Although there are subtle differences among the various ordering algorithms, they all fit the same pattern when viewed from a high level of abstraction, see Fletcher and Hall [22].

### 1.5 Tearing heuristics to create bordered block lower triangular forms

Beside the references given in Sect. 1.4, the engineering literature is also rich in sparse matrix ordering algorithms. Decomposing to bordered block lower triangular form has a long tradition in engineering applications: It is usually referred to as tearing, diakoptics, or sequential modular approach, depending on the engineering discipline. When dealing with distillation columns, tearing is called stage-to-stage or stage-by-stage calculations. Tearing dates back to the 1930's [34,48], 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 Aspen Technology, Inc. [1], MOSAICmodeling [9], Dymola [10], JModelica [37], or OpenModelica [40]. 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" [36].

The various tearing heuristics are concerned with selecting a minimal subset of variables called the torn variables; when these torn variables are moved to the border of the matrix, and the Dulmage-Mendelsohn decomposition is applied to the rest of the matrix, the blocks of the resulting bordered block lower triangular form correspond to the devices (or machines) of the technical system. The block sizes therefore tend to be $O(1)$, that is, they are typically bounded by a small constant. More than five decades of practical experience and the widespread usage of tearing show that the tearing heuristics also tend to produce a narrow border when applied to technical systems.

### 1.6 Further assumptions

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 Euclidean norm ( $\ell_{2}$-norm).

We also assume that the bound constraints $\underline{x} \leq x \leq \bar{x}$ 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.

## 2 Overview of the proposed algorithm

The algorithm builds up a point cloud sequentially, satisfying

$$
\begin{align*}
& F_{1: i}\left(x_{0: i}\right) \approx 0 \text { for } i=1, \ldots, N, \\
& \underline{x} \leq x \leq \bar{x} . \tag{8}
\end{align*}
$$

The algorithm starts with a scattered set of points $S^{(0)}$ for $x_{0}$, then eliminates the square blocks one-by-one along the diagonal in order $i=1, \ldots, N$, see Eq. (5) and Fig. 1. Solving (8) for $x_{i}$ will be referred to as forward solve.

If we applied forward solve only, the algorithm would be similar to Gaussian elimination without pivoting, which can give arbitrarily poor results even for well-conditioned linear problems [24, Ch. 3.3]. In the nonlinear case, and when propagating the point cloud within the variable bounds, the numerical issues manifest themselves in two ways:
(a) Many or all points become bound infeasible.
(b) The $x_{i}$ component of many points in the point cloud accumulate around one point or around a particular subspace. In this case, the remaining part of the feasible region is no longer adequately represented by the other points.

In both cases, the point cloud is no longer a proper approximation of the solution set of (8). Fig. 2a illustrates both issues on the test problem of Sect. 5: Some of the points are outside the feasible region (outside the so-called composition simplex), and many points have accumulated along the $(0,0)-(0,1)$ line, while the interior part of the feasible region is poorly covered.

We propose the following procedure to mitigate the numerical issues. In each iteration step, after having solved (8) for $x_{i}$, new points are inserted for a subset of $x_{i}$ uniformly at


Fig. 2 Illustrating how the backsolve step introduces new points on the test problem of Sect. $5(N=60)$. a The scattered set of points after the forward solve in a particular iteration $i$, projected to $2 \mathrm{D} ; z_{1}, z_{2}$ are components of $x_{i}$; solid lines: boundaries of the feasible region, the so-called composition simplex. b The set of points after the backsolve step in the same iteration $i$. The three gray dots show the solutions
random, let $\left(\tilde{x}_{i}\right)_{J}$ denote this subset where $|J|=d$ and $d$ was defined at (6). (The $\left(\tilde{x}_{i}\right)_{J}$ denotes those components of the vector $\tilde{x}_{i}$ whose index is in the index set $J$.) Both the index set $J$ and the values for $\left(\tilde{x}_{i}\right)_{J}$ come from a random number generator. We use the tilde to distinguish between $x_{i}$ and $\tilde{x}_{i}$ : The former values come from the forward solve, the latter from a random number generator. These newly inserted $\left(\tilde{x}_{i}\right)_{J}$ points are still lacking the components of $x_{0: i}$ that are not covered by $\left(\tilde{x}_{i}\right)_{J}$. Therefore, for a given $\left(\tilde{x}_{i}\right)_{J}$, and for each point $x_{0: i-h-1}$ in the scattered set of points $S^{(i)}$, we solve the following NLPs:

$$
\begin{align*}
\underset{y_{i-h: i}}{\operatorname{minimize}} & \left\|F_{i-h: i}\left(x_{0: i-h-1}, y_{i-h: i}\right)\right\| \\
\text { subject to } & \left(y_{i}\right)_{J}=\left(\tilde{x}_{i}\right)_{J}  \tag{9}\\
& \underline{x}_{i-h: i} \leq y_{i-h: i} \leq \bar{x}_{i-h: i}
\end{align*}
$$

to determine the missing components, where $h$ is a hyper-parameter of the algorithm ( $h$ stands for history, and it is typically a small integer). The missing components will partly come from the old point (the $x_{0: i-h-1}$ part), and the rest is the solution of the NLP above (the $y_{i-h: i}$ part); the $\left(\tilde{x}_{i}\right)_{J}$ part from the random number generator remains unchanged. The procedure of solving ( 9 ) will be referred to as backsolve. The new points are forcibly inserted in the backsolve step, it is therefore expected that there will be some amount of constraint violation in $F_{i-h: i}\left(x_{0: i-h-1}, y_{i-h: i}\right)$, which has to be tolerated. Figure 2 illustrates how the backsolve introduces new points.

The last subproblem at $i=N+1$ is different from (8) in that it is an overdetermined system, while all the other subproblems (8) are square. At $i=N+1$ the algorithm skips the forward solve step (since there is no square block to eliminate), and performs a backsolvelike step: It solves (9) with $J=\varnothing$, and with $x_{i-h: N}$ as starting points for $y_{i-h: N}$. (For the variables, the variable slices $i-h: i$ are truncated to $i-h: N$, see Sect. 1.2.)

The output of the proposed (main) algorithm, after finishing the last subproblem $i=N+1$, is a point cloud approximating the solution set of (1). The implementation details of the algorithm will be discussed in Sect. 4. The algorithm of the present paper is a significant
improvement over older algorithms discussed in [3,4], both algorithmically and on the implementation level. The entire algorithm has been redesigned and rewritten from scratch, and in particular, the backsolve step is radically different. Our numerical results show several orders of magnitude improvements in speed, while achieving better robustness at the same time.

## 3 Exponential worst-case time complexity in the border width

As discussed at (8), the proposed method builds up a point cloud lying approximately on the implicitly defined solution manifold of

$$
\begin{align*}
& F_{1: i}\left(x_{0: i}\right)=0 \text { for } i=1, \ldots, N, \\
& \underline{x} \leq x \leq \bar{x}, \tag{10}
\end{align*}
$$

and aims at a point distribution such that every point on the manifold is close to a point of the cloud. (We refer back to Sect. 1.3 regarding the singular points.) For reasons of efficiency, the point cloud is constructed in a heuristic way, guided by theory.

Let $N(s)$ denote the number of boxes intersecting the solution manifold when uniformly covering the bound constraint box by a grid of boxes of side $s>0$. We call $e:=\lim _{s \rightarrow 0} e(s)$, where $e(s):=\log N(s) / \log s$, the effective dimension of the solution manifold. As a consequence, the size of a cloud with the property that every point on the manifold has distance at most $s$ to a point of the cloud grows for small $s$ proportionally to $s^{e}$. In other words, constructing the point cloud will have a time complexity that is exponential in the effective dimension $e$. Creating the point cloud is therefore computationally tractable only for small effective dimensions $e$; how small depends on the resolution $s$ needed, which fortunately is not high when (as usual) the total number of solutions of the original system is small, and the solutions are well-separated. Thus a small effective dimension $e$ is the main assumption under which our new method can operate efficiently.

If the Jacobian of (10) has full rank then (since $d$ equations are "missing" from the square system) the solution manifold has dimension $d$, the $d$-dimensional volume of the solution manifold is finite, and thus the effective dimension is $e=d$. But in general, pathological cases might be possible, such as Peano-like curves $(d=1)$ that come close to every point in the box and then have large $e$. Excluding such pathological cases, which do not arise in most applications of interest, the border width $d$ agrees with the effective dimension $e$.

In engineering applications, the presence of important bounds further decreases the effective dimension of the manifold. For example, we have the natural non-negativity bound on many variables. Each such bound will be active at many solutions, effectively amounting to an additional equation, typically decreasing the effective dimension by one. In addition, if the lower and upper bound on some variable differs by significantly less than the threshold $s$, this variable is effectively constant and also decreases the effective dimension. Such strong specifications are fairly common since the designer wants the system to perform something useful and therefore pushes the system to its limits (for example to create almost pure chemicals). We give numerical examples in Sects. 5 and 6 showing that the method is practical for certain difficult engineering applications.

To locate the solution manifold, i.e., to construct the approximating point cloud, we need to sample function values without knowing beforehand where the useful points lie that should go into the point cloud. To achieve this efficiently is the main reason why the bordered block triangular decomposition is needed. Indeed, we could sample the solution set of $F_{1: N}\left(x_{0: N}\right)=$ 0 within the bound constraints directly, without decomposition. However, the volume to be sampled then grows exponentially with the dimension $p:=\operatorname{dim} x_{0: i}$, which gets larger
and larger (ultimately $p=n$ ). This makes good sampling in this naive way prohibitively expensive. The proposed method avoids this scalability trap by sampling only at the square blocks along the diagonal (see Fig. 1): The volume to be sampled grows exponentially only with the largest block size, which is assumed to be reasonably small. In engineering applications, this assumption is usually satisfied since typically the largest block corresponds to the largest device/machine in the technical system being modeled.

## 4 Implementation details of the proposed algorithm

A high-level overview of the algorithm was already given in Sect. 2. In this section we discuss the building blocks in more detail. These building blocks are mostly implementation-level details, and there could be other ways to fill-in these low-level details that the high-level overview left open.

### 4.1 The source code of the algorithm

The most complete description of the algorithm is its source code, therefore the Python source code of the algorithm is available on GitHub [2] under the very permissive 3-Clause BSD License. For convenience the source code is distilled down to its essence, and it is given in "Appendix A" as pseudo-code too. Algorithm 1 of "Appendix A" is the core of the algorithm. We use the VA27 solver from HSL [29] to solve the equations and NLPs at each block. Since this solver cannot handle variable bounds, we enforce them with Algorithm 2. The backsolve step is given by Algorithm 3. The pseudo-code is less than 50 lines in total.

### 4.2 The farthest-first subsampling algorithm

The goal of the subsampling algorithm is to select a spatially well-distributed subset of a given scattered set of points $S$. A greedy heuristic is implemented, based on the so-called farthest-first traversal. The algorithm starts by choosing a point in $S$. We currently pick the point closest to the mean of $S$; other choices are also possible, including the random choice. Then, points are selected one-by-one, always picking that not yet chosen point next that is the farthest away from the already chosen ones, breaking ties arbitrarily. The subsampling algorithm stops when the desired sample size is reached.

### 4.3 Generating the new random points in the backsolve step

We refer back to Sect. 2, and to Fig. 2: After each forward solve we must insert new points into the sample where the manifold is not approximated properly. One way of populating such deserted areas would be inter- and extrapolation; this would assume that the spatial distribution of the points is already appropriate for inter- and extrapolation tasks, and assumes connectedness of the manifold. While this could be a viable approach, we chose a much simpler and more robust approach. Essentially we propose brute-force oversampling at the block level: We try to insert significantly more $\left(\tilde{x}_{i}\right)_{J}$ points than what we need. We do not know where to insert them, so we generate them uniformly at random within the variable bounds (brute-force). Then, the NLPs (9) of the backsolve step are solved, and those points whose objective (norm of the constraint violation) is above a user-defined threshold are
discarded. Finally, we keep only the most distant ones of the remaining points by applying the subsampling algorithm.

This approach for populating deserted areas of the manifold is very robust, and fairly simple to implement. It does not assume connectedness, and it does not assume anything about the spatial distribution of the already existing points in the sample. In fact, if we loose all points in the forward solve, the backsolve may still succeed to insert new points, and the algorithm can continue. In contrast, it is impossible to inter- and extrapolate if we have lost all our points. Since we cannot assume connectedness of the manifold, some sort of (block-level) global sampling is inevitable.

### 4.4 Efficient implementation of the backsolve step

A significant fraction of the execution time is spent in the backsolve step, solving (9). Three improvements proved to be crucial to perform the backsolve step efficiently: (i) trying only a small, carefully selected subset of all the possible combinations of the $\left(\left(\tilde{x}_{i}\right)_{J}, x_{0: i-h-1}\right)$ matches in (9) instead of trying all of them, (ii) estimating a good starting point for (9), and (iii) skipping those matches that are very likely to have above-threshold objective value (constraint violation) at the optimum, and would most likely be discarded anyway.

As Sect. 2 is written, we try all the possible $\left(\left(\tilde{x}_{i}\right)_{J}, x_{0: i-h-1}\right)$ matches in a brute-force manner. The previous implementation of the algorithm also worked [3] this way. Numerical evidence shows that it can be very wasteful: If two distinct points in the point cloud are close in their $x_{i-h: i}$ components, it is very likely that the $\left(\left(\tilde{x}_{i}\right)_{J}, x_{0: i-h-1}\right)$ matches will have very similar objective value in (9) too; there is little to no benefit in trying both of them. An optional heuristic that we propose is to apply the subsampling algorithm of Sect. 4.2 to the points of the point cloud, considering their $x_{i-h: i}$ components only. We then try to match the points $\left(\tilde{x}_{i}\right)_{J}$ with this selected subset only. This heuristic can be disabled at the user's discretion.

We propose estimating a starting point $y_{i-h: i}$ for (9) with singular-value decomposition (SVD, see [39, Ch. 10.2]). For simplicity, and since it seems to be adequate in practice, we currently ignore during this estimation the variable bounds in (9), and we also assume that a linear approximation to (9) around the optimum is appropriate. (This estimation is crude: We set parts of $\Delta x_{i}$ to zero, although we let them change in (9) arbitrarily.)

We consider the submatrices of the Jacobian $J$ of $F(x)$ shown in Fig. 3, and defined as follows. The rows of $J_{11}$ are the row blocks $i-h: i-1$ of $J$; those of $J_{21}$, and $J_{22}$ is row block $i$ of $J$. The columns of $J_{11}$ and $J_{21}$ are the column blocks $i-h: i-1$ of $J$; those of $J_{22}$ is $i$ column block $i$ of $J$. The Jacobian is evaluated at $x_{0: i}$ (at the point $x_{0: i}$ that comes from the forward solve). In terms of these submatrices, the following linear least-squares problem is solved with SVD:

$$
\underset{\Delta x_{i-h: i-1}}{\operatorname{minimize}}\left\|\left[\begin{array}{l}
J_{11}\left(x_{i-h: i-1}\right)  \tag{11}\\
J_{21}\left(x_{i-h: i-1}\right)
\end{array}\right]\left[\Delta x_{i-h: i-1}\right]-\left[\begin{array}{c}
0 \\
J_{22}\left(x_{i}\right) \Delta x_{i}
\end{array}\right]\right\|_{2}^{2}
$$

Informally speaking, (11) solves the linear approximation to (9) in which the variable bounds are ignored, $\left(\Delta x_{i}\right)_{J}=\left(\tilde{x}_{i}\right)_{J}-\left(x_{i}\right)_{J}$, and all other components of $\Delta x_{i}$ that are not in the index set $J$ are set to zero. The solution to the linear least-squares problem (11) gives us $\Delta x_{i-h: i}$, and our estimate for $y_{i-h: i}$ is $x_{i-h: i}+\Delta x_{i-h: i}$.

The best match $\left(\left(\tilde{x}_{i}\right)_{J}, x_{0: i-h-1}\right)$ for each $\left(\tilde{x}_{i}\right)_{J}$ is always tried. For those matches for which the norm of $\left\|F_{i-h: i}\left(x_{0: i-h-1}, y_{i-h: i}\right)\right\|$ at the starting point is below the pre-defined threshold (hyperparameter), we select at most $m-1$ additional candidate $\left(\left(\tilde{x}_{i}\right)_{J}, x_{0: i-h-1}\right)$

Fig. 3 Submatrices $J_{11}, J_{21}$, and $J_{22}$ used in the starting point estimation for the backsolve step, see (11)

matches with subsampling. For each candidate match, we launch the local solver from the estimated $y_{i-h: i}$ to solve (9). The value of $m$ is an arbitrary, used-defined value; in our numerical experiments $m=20$ was used, and we did not attempt to tune this hyperparameter.

## 5 Numerical results: the effect of decomposition

We give numerical results where the computational gains, if any, are thanks to the block decomposition. The benchmark problems are coded in the AMPL modeling language [23], and are available on GitHub [2] together with the source code of the algorithm.

### 5.1 Series of test problems

The steady-state simulation of distillation columns can be a major numerical challenge [13]. Our example is a series of challenging distillation columns; these columns have 3 solutions, one of which is missed even with problem-specific methods, see Sect. 5.2. Distillation columns consist of so-called stages. The natural order of the stages directly yields the desired block structure (2) and (4) by virtue of the internal physical layout of distillation columns; no preprocessing is necessary. (Even if it was not the case, we could use any of the ordering algorithms referenced in Sect. 1.4 and 1.5 to create the block structure fully automatically.) There is a one-to-one correspondence between the stages and the blocks.

In the engineering applications it is common to optimize the total cost by varying the number of stages, which makes distillation columns perfect test problems from the perspective of the present paper: Distillation columns have a natural parameter, namely the number of stages, for examining how different numerical methods scale as the number of blocks changes. As the number of blocks is varied (within reasonable limits) each column is interesting from an engineering point of view. Let $N$ denote the number blocks. In our examples the size of each block is $4 \times 4$ except the first block which is $2 \times 2$; the problem size is $4 N$; the number of nonzeros is $25 N-10$. The manifold dimension $d=2$, and it is independent of $N$.

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 [25], except for the number of stages $N$ and the feed stage location $N_{F}$. The specifications are the feed composition (methanol-methyl butyratetoluene), 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 $[16,25]$. Multiple steady-states can be predicted by analyzing columns with infinite reflux and infinite length $[5,26,42]$. These predictions for infinite columns have relevant implications for columns of finite length operated at finite reflux.

### 5.2 Numerical results published in the literature

The published numerical results for our test problem indicate numerical difficulties. Both the conventional inside-out procedure [8] and the simultaneous correction procedure [38] were reported to miss the unstable steady-state solution, see Vadapalli and Seader [49] and Kannan et al. [31] (all input variables specified; output multiplicity). However, all steady-state branches were computed either with the AUTO software package [12] or with an appropriate continuation method $[25,31,49]$. In both cases, the initial estimates were carefully chosen with the $\infty / \infty$ analysis [5,26], and special attention was paid to the turning points and branch switching. Unfortunately, those papers do not include execution times, most likely because the computations involved human interactions too (initial estimates, turning points and branch switching).

### 5.3 The baseline for comparisons

As discussed in Sect. 5.2, the literature clearly indicates that our benchmark problems are challenging, unfortunately the execution times are not available for comparisons; we have to establish a baseline for comparisons.

### 5.3.1 Requirements for the baseline algorithm

In order to assess the quality of our new method within the prior state of the art we need to compare against a suitable baseline method with similar capabilities. We use the following criteria that such a baseline method should possess. It should be
(1) state-of-the-art;
(2) able to enumerate all solutions of large, sparse systems;
(3) able to handle transcendental equations and bound constraints,
(4) usable from an advanced modeling language without user-input beyond equations and variable bounds;
(5) a generic algorithm not tailored to a specific class of problems;
(6) easy to use without any expert knowledge;
(7) publicly available as an off-the-shelf solver.

To our knowledge, there is currently no such solver. But the technology to create one based on traditional techniques is available; so we wrote the baseline solver ourselves. We chose

Table 1 Relative frequencies (percentages) of IPOPT finding a particular solution when starting points are generated uniformly at random between the variable bounds

| $N$ | Sol. 1 | Sol. 2 | Sol. 3 | None |
| :---: | :---: | :---: | :---: | :---: |
| 50 | 82.3 | 17.0 | 0.7 | 0.0 |
| 51 | 83.1 | 16.2 | 0.7 | 0.0 |
| 52 | 84.0 | 15.3 | 0.7 | 0.0 |
| 53 | 84.8 | 14.5 | 0.7 | 0.0 |
| 54 | 85.6 | 13.7 | 0.6 | 0.0 |
| 55 | 86.1 | 13.2 | 0.6 | 0.0 |
| 56 | 86.7 | 12.6 | 0.6 | 0.0 |
| 57 | 87.2 | 12.2 | 0.6 | 0.0 |
| 58 | 87.6 | 11.7 | 0.5 | 0.1 |
| 59 | 88.0 | 11.2 | 0.5 | 0.3 |
| 60 | 88.3 | 10.6 | 0.5 | 0.5 |
| 61 | 88.7 | 9.8 | 0.5 | 1.0 |
| 62 | 89.2 | 9.0 | 0.4 | 1.4 |
| 63 | 89.4 | 8.2 | 0.4 | 2.0 |
| 64 | 89.6 | 7.3 | 0.4 | 2.7 |
| 65 | 89.7 | 6.4 | 0.3 | 3.6 |
| 66 | 89.8 | 5.6 | 0.3 | 4.3 |
| 67 | 90.0 | 4.8 | 0.2 | 5.0 |
| 68 | 90.1 | 4.1 | 0.2 | 5.6 |
| 69 | 90.3 | 3.5 | 0.2 | 6.1 |
| 70 | 90.2 | 3.0 | 0.1 | 6.6 |
| 71 | 90.4 | 2.6 | 0.1 | 6.9 |
| 72 | 90.5 | 2.2 | 0.1 | 7.2 |
| 73 | 90.6 | 1.9 | 0.1 | 7.4 |
| 74 | 90.6 | 1.7 | 0.1 | 7.7 |
| 75 | 90.6 | 1.5 | 0.1 | 7.8 |

AMPL [23] as the modeling environment IPOPT [51] as local solver. Both are state-of-theart, and their highly polished implementation is among the fastest ones. To enumerate all solutions, we implemented multistart with uniform random sampling between the variable bounds. (Uniform sampling is adequate since all variables are scaled to be between 0 and 1.)

### 5.3.2 Results with the baseline algorithm

IPOPT was executed from 250,000 randomly generated points for $N=50 . .74$, and 500,000 points were necessary for $N=75$ to get consistent results. Table 1 shows the relative frequencies of IPOPT finding a particular solution.

The points are partitioned into consecutive batches: The first batch starts with the first point. A batch is completed when all 3 solutions are found, and then the next batch starts. Only batches completed within the allocated point budget are considered (budget: 250,000 points for each $N=50 . .74$, and 500,000 points for $N=75$ ), that is, if the last batch is unfinished, we ignore it. For a fixed $N$, the total number of iterations per completed batch fits the exponential distribution, see Fig. 4 for $N=60$. The growth rate of the expected number of iterations in a batch fits equally well with exponential and linear correlation in


Fig. 4 Histogram of the number of iterations per batch when the starting points are generated uniformly at random between the variable bounds; $N=60$. The counts are normalized to form a probability density, i.e., the area under the histogram will sum to 1 . The fitted curve corresponds to the exponential distribution
the $N=50 . .63$ regime, and it fits exponential growth rate between $N=64 . .75$, see Fig 5 . The total number of iterations IPOPT made in a batch correlates well with the total execution time, and with the number of starting points in the same batch.

### 5.4 Results with the proposed method

### 5.4.1 Illustrating the point cloud computed with the proposed method

Figure 6 shows the intermediate point cloud in iteration $i=30$ for $N=60$, projected to 2D with principal component analysis (PCA). We used Scikit-learn [41] to perform PCA and to generate the plots. Fig. 7 shows the final output of the proposed method, the generated starting points, projected to 2D with PCA. Higher principal components produce (as to be expected) more wiggly projectors that do not represent true features of the point cloud.

### 5.4.2 Illustrating the point cloud with manifold learning

We also investigated the manifold structure using manifold learning. We tried each manifold learning algorithm available in Scikit-learn: Isomap [47], locally linear embedding (LLE) [44], modified locally linear embedding (MLLE) [53], Hessian Eigenmapping (also known as Hessian-based LLE or HLLE) [15], Spectral Embedding (Laplacian Eigenmaps) [6], local tangent space alignment (LTSA) [52], and multidimensional scaling (MDS) $[7,32,33]$. (Although the t -distributed Stochastic Neighbor Embedding [35,50] algorithm, or t-SNE, also proved to be robust, we did not use t-SNE for the present paper: It was designed to artificially exaggerate structure in the data to reveal clusters, but that is undesirable in our case.)

It is not uncommon that the embeddings show false structures that in reality are not present in the data. Problem-specific knowledge was used to recognize any false structure in the embeddings as follows. We colored each point: The mole fractions of the 3 chemical components in the liquid phase on stage $k$ of the distillation column are chosen as the coordinates in the RGB color space, where the stage index $k$ is a parameter. In short, the color of the point corresponds to the chemical composition of the liquid phase on stage $k$. The


Fig. 5 Computational effort of multistart with starting points generated uniformly at random between the variable bounds. The effort is measured as the mean number of iterations per batch, averaged over 250,000 starting points. The effort growth rate fits equally well with exponential and linear correlation in the $N=50 . .63$ regime, and it fits exponential growth rate between $N=64 . .75$
theory of distillation - in particular the so-called residue curve map [14] of the mixture - tells us that for a fixed $k$ we should see a smooth color transition in the embeddings, similar to the smooth shade transition in Fig. 6. Furthermore, the coloring of the points in the embeddings should change only smoothly when $k$ is increased or decreased by 1 .

We inspected these color transitions for each algorithm offered by Scikit-learn. If the manifold learning algorithm creates a wrong embedding or false structure, it is obvious at first glance. In our numerical experience, among the manifold learning algorithms implemented in Scikit-learn, only multidimensional scaling was robust enough to consistently produce correct embeddings without any hyperparameter tuning. A possible explanation for its robustness could be that it randomly chooses the initial configuration; the other embedding techniques that we listed are based on a nearest-neighbor search which can be fooled if the points happen to have unfortunate distribution in the original high-dimensional space. The downside of multidimensional scaling is that it was by far the most computationally expensive manifold learning algorithm of all tried. Note, however, that multidimensional scaling is not part of the proposed method; it is used only for visualization.


Fig. 6 Representative sample of the 2D solution manifold of the leading subsystem $F_{0: 30}(x)=0$, generated with the proposed method $(N=60)$. The sample is projected to the planes of the principal components $y_{i}-y_{j}$

Multidimensional scaling seeks a low-dimensional representation of the data in which the distances sensibly approximate the distances in the original high-dimensional space (the between-points distances are preserved as well as possible). Using multidimensional scaling from Scikit-learn, we unfolded the discretized 2D manifold formed by the generated starting points into the 2D plane and found the embedding shown in Fig. 8.

### 5.4.3 Running a local solver from the output of the proposed algorithm

The subsampling algorithm of Sect. 4.2 selects the points in a specific order; the subsampling procedure can be used to order the points in any set $S$. This order is the so-called greedy permutation or the farthest-first traversal. When the main algorithm finishes, we propose that a local solver for large-scale, sparse problems (like IPOPT) is launched from the points of the final point cloud in this order. The numerical experiments suggest that this increases the likelihood of finding all solutions early, because we always try that point next that is the least similar to the already tried ones. As it is shown in Fig. 8, the first 3 points picked by the farthest-first heuristic suffice to find all solutions in this case. Note that in Sect. 5.3 the probability of finding the third solution was $0.5 \%$ for starting points generated uniformly at random between the variable bounds; see in Table 1, row $N=60$.

Numerical experiments also show that the final constraint violations are non-distinctive with respect to the goodness of the starting points: Below a certain threshold, the constraint violations are due to the random perturbations applied in the backsolve step, and they do not


Fig. 7 The starting points generated by the proposed method $(N=60)$. The points are projected to the planes of the principal components $y_{i}-y_{j}$. There would be 3 well-separated clusters of points in the ideal case, however, the last two equations $F_{N+1}(x)=0$ only trim the 2D solution manifold of the leading subsystem $F_{0: N}(x)=0$ due to mild ill-conditioning. One cluster is nevertheless fairly small and well-separated

Fig. 8 The starting points (circles and squares) generated by the proposed method ( $N=60$ ), embedded into the 2D with plane with multidimensional scaling. The 3 crosses show the 3 solutions. Each cluster of starting points yields the solution it surrounds when the IPOPT solver is started from there. The first 3 points picked by the farthest-first heuristic of Sect. 4.2 are marked with $1,2,3$; in this case, they suffice to find all solutions. Note that the farthest-first heuristic measures distances in the original high-dimensional space

convey any information regarding the goodness of the starting points. In other words, the constraint violation is not a good metric for ordering the final starting points; we propose the farthest-first traversal instead.

### 5.5 Comparisons: The effect of decomposition

The effect of the decomposition (2)-(5) can be studied by requesting all solutions for a given column length, and comparing the execution times of the proposed method with the baseline multistart algorithm (no decomposition). As we discussed in Sect. 5.3, if the starting points are generated uniformly at random within the variable bounds, the computational efforts grow exponentially for $N \geq 64$. For the proposed method, the computational efforts grow linearly, thanks to the decomposition. It depends on the problem size (column length) and on the hyperparameter settings whether the decomposition, and the more sophisticated algorithm pays off; see the left column of Fig. 9, comparing the execution times.

## 6 Numerical results: reusing shared substructure

A frequent task in engineering is to solve a series of related square systems $F^{\ell}(x)=0$, where the number $N_{\ell}$ of blocks of the $\ell$ th problem and hence the Jacobian varies, but the equations in the first $B_{\ell}$ blocks of $F^{\ell}$ and $F^{\ell+1}$ are identical; the remainder may deviate arbitrarily. If $B_{\ell}$ is close to $N_{\ell}$, the major part of the point cloud can be reused without any change.

We give numerical results where the computational gains, if any, are thanks to the reused substructure. The benchmark problems and the baseline algorithm are the same as in Sect. 5. The difference is that all solutions to 10 different columns with consecutive length are required. The shared substructure can be reused with the proposed method. This results in significant gains compared to our baseline multistart method, see the right column of Fig. 9. As previously, it depends on the problem size, and on the hyperparameter settings whether the decomposition, and the more sophisticated algorithm pays off.

## 7 Future work

Nonlinear programming with optionally varying $N$ and $\ell$ Another common application in the field of engineering is to augment the leading subsystem of $F_{1: N}\left(x_{0: N}\right)=0$ of (1) with an objective function and ask for all global optima.

$$
\begin{array}{ll}
\min & G^{(N, \ell)}\left(x_{0: N}\right) \\
\mathrm{s.t.} & F_{1: N}\left(x_{0: N}\right)=0  \tag{12}\\
\underline{x} \leq x \leq \bar{x}
\end{array}
$$

The basic algorithm, sketched in Sect. 2, is applied to $F_{1: N}\left(x_{0: N}\right)=0$ up until and including block $N$ as before to obtain a point cloud, approximately satisfying the constraints of (12). Then, a local solver is executed from the points of the point cloud, targeting the nonlinear program (12). As for the computational savings with varying $N$ and $\ell$, the same arguments hold as in the previous paragraphs: Whether the leading underdetermined subsystem is augmented with $d$ additional equations (making it square), or with an objective function, the point cloud for the shared leading subsystem $F_{1: N}\left(x_{0: N}\right)=0$ can be reused either way.

Chemical engineering use cases include the following scenarios. The steady-state model of a distillation column is given without the reflux specification, that is, the system of equations is underdetermined by one degree of freedom. First, we compute a point cloud, approximately satisfying this system, with Alg. 1 halting at line 10. Then, the engineer can specify

- the reflux ratio, that is, provide one more equation, making the system square;


Fig. 9 Comparing the execution times of the proposed method to multistart with randomly generated starting points between the variable bounds. For multistart, the mean execution time for a batch is given, averaged over 250,000 starting points. The execution times for the proposed method are all-inclusive: IPOPT is launched from the first 6 points picked by the farthest-first heuristic which suffices to find all 3 solutions. Left side: All solutions are required for the given column lengths. Right side: All solutions to 10 different columns with consecutive length are required; the execution times are plotted at the longest column. Rows from top to bottom: $M_{\text {keep }}=100,200,400$; for the meaning of the algorithmic parameters $h$ (history), $M_{\text {keep }}$ (at most this many new points are inserted in each iteration) see the pseudo-code in "Appendix A"

- or the reflux molar flow rate, making the system square;
- or append an objective function, and look for the cost optimal reflux ratio.

In each case, the engineer can (re)use the point cloud for free. There is still some additional computational work left due to the newly added equation or objective function, but that work is negligible compared to computing the point cloud. We expect significant computational benefits where a precomputed point cloud can be reused several times at no additional cost thanks to shared substructure.

Acknowledgements Open access funding provided by Austrian Science Fund (FWF).
Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

## A Pseudo-code of the implemented algorithms

```
Algorithm 1: The proposed algorithm (top level, main algorithm)
    Input: \(P\) : A problem instance as defined by (1), see also Fig. 1
    Output: A set of starting points \(S^{(N+1)}\) for a local solver
    Parameters : \(M_{0}\) : Number of points to generate for the initial set \(S^{(0)}\)
                \(M_{\text {keep }}\) : We add at most this many new points in each iteration \(i\)
                \(h\) : The number blocks to re-solve in backsolve
                \(\epsilon_{\text {last }}\) : The acceptable constraint violation in the last step
    Initialize the set of \(x_{0}\) points, \(S^{(0)}\), uniformly at random with \(M_{0}\) points
    for \(i=1\) to \(N\) do
        // Forward solve, see Sect. 2:
        foreach \(x_{0: i-1} \in S^{(i-1)}\) do
            // Square system solved from random starting point, variable bounds ignored:
            Solve \(F_{i}\left(x_{0: i-1}, x_{i}\right)=0\) for \(x_{i}\) (with \(x_{0: i-1}\) fixed)
            Add the resulting \(x_{0: i}\) vector to \(S^{(i)}\) (even if it is bound infeasible)
            // Call backsolve to add more points, see Eq.(9) and Fig. 2 in Sect. 2:
            Call Algorithm 3 with \(S^{(i)}\); that algorithm returns a new set of points \(T\)
            Append \(T\) to \(S^{(i)}\)
            // Try to repair the bound infeasible points by small perturbations:
            Call Algorithm 2 with \(S^{(i)}\), and then replace \(S^{(i)}\) with the returned set
            // All points in \(S^{(i)}\) are bound feasible now; the too infeasible ones were discarded.
            // Backsolve oversampled the search space (brute-force), discard the excess points:
            Apply subsampling in \(x_{i}\) to the points inserted by Alg. 3, keep at most \(M_{\text {keep }}\) of them
            // Only the \(x_{i}\) components of the new points were considered in the subsampling.
            // We have \(\left|S^{(i)}\right| \leq M_{0}+i \cdot M_{\text {keep }}\) points at this line.
    \(i=N+1\)
    // Reached the last \(d\) equations with no new variables; try to reduce \(\left\|F_{N+1}(x)\right\|\) :
    foreach \(x_{0: i-1} \in S^{(i-1)}\) do
        // Overdetermined system, \(x_{i-h: N}\) as starting point, variable bounds ignored:
        \(\min _{y_{i-h: N}}\left\|F_{i-h: i}\left(x_{0: i-h-1}, y_{i-h: N}\right)\right\|\)
        Let \(x_{0: N}^{*}:=\left(x_{0: i-h-1}, y_{i-h: N}\right)\) denote the optimal solution
        Add \(x_{0: N}^{*}\) to \(S^{(i)}\) if \(\left\|F_{i-h: i}\left(x_{0: N}^{*}\right)\right\| \leq \epsilon_{\text {last }}\)
    // Try to repair the bound infeasible points by small perturbations:
    Call Algorithm 2 with \(S^{(i)}\), then replace \(S^{(i)}\) with the returned set
    return \(S^{(i)}\)
```

```
Algorithm 2: Repairing bound infeasibility; workaround due to VA27 from [29]
    Input: \(P, h, i\), and \(S^{(i)}\) from Alg. 1
    Output: Set of points \(W\), all \(x_{0: i} \in V\) are bound feasible and \(\left\|F_{i-h: i}\left(x_{0: i}\right)\right\| \leq \epsilon\)
    Parameters : \(\delta\) : Threshold above which we do not try to repair bound infeasibility
                    \(d\) : The manifold dimension in (1)
                            \(\epsilon\) : Tolerated constraint violation
    // If \(i=N+1, x_{0: i}\) is cropped to be \(x_{0: N}\).
    // Keep only those points that have sufficiently small bound violations:
    Compute the subset \(T\) of \(S^{(i)}\) for which the \(L_{2}\)-norm of bound violations is less then \(\delta\)
    // The already bound feasible points are temporarily ignored till line 8:
    Split \(T\) into set \(U\) of the bound feasible points, and set \(V\) of the bound infeasible ones
    Project each point in \(V\) back to the nearest boundary of the bound feasible region
    // We now fix those components that changed the most during the projection,
    // and try to reduce the constraint violations by changing the remaining components:
    foreach \(x_{0: i} \in V\) do
            Minimize \(\left\|F_{i-h: i}\left(x_{0: i}\right)\right\|\) with the \(d\) most changed components fixed
            // If less then \(d\) components changed during the projection,
            // fix some at random until \(d\) components are fixed.
            Save the resulting point in \(W\)
    Project each point in \(W\) back to the nearest boundary of the bound feasible region
    Merge \(U\) into \(W\)
    For each \(x_{0: i} \in W\) re-evaluate \(\left\|F_{i-h: i}\left(x_{0: i}\right)\right\|\)
    Discard all \(x_{0: i} \in W\) for which \(\left\|F_{i-h: i}\left(x_{0: i}\right)\right\|>\epsilon\)
    return the remaining set of points \(W\)
```

```
Algorithm 3: Backsolve
    Input: \(P, h, i\), and \(S^{(i)}\) from Alg. 1
    Output: New set of points \(T\), see Fig. 2 for an example
    Parameters : \(M_{b a c k}\) : Number of \(\left(\tilde{x}_{i}\right)_{J}\) points to generate at random, see also Sect. 2
                \(\epsilon_{\text {linear }}\) : Residual threshold for candidate matches
                    \(\epsilon_{n l p}:\) Residual threshold after solving the nonlinear program (9)
                        \(m: 20\) in our experiments, see Sect. 4.4
    Generate the \(\left(\tilde{x}_{i}\right)_{J}\) points with a random number generator, \(M_{\text {back }}\) points in total
    if \(i \leq h\) then
        // We get a square system after fixing \(\left(\tilde{x}_{i}\right)_{J}\). We use random starting points,
        // and we ignore the variable bounds and all the points in \(S^{(i)}\) :
        Solve \(F_{1: i}\left(x_{0: i}\right)=0\) for \(x_{0: i}\) with \(\left(\tilde{x}_{i}\right)_{J}\) fixed (ignore variable bounds)
        Add the results to the new set of points \(T\) (even if bound infeasible)
    if \(i>h\) then
        Optional: Compute a subsample \(\hat{S}^{(i)}\) of \(S^{(i)}\) with farthest-first subsampling of Sect. 4.2 // When the
        optional subsampling heuristic is disabled: \(\hat{S}^{(i)}:=S^{(i)}\).
        // \(F_{i-h: i}\left(x_{0: i-h-1}, x_{i-h: i}\right)=0\) is square before fixing \(\left(\tilde{x}_{i}\right)_{J}\), overdetermined after that.
        // Unlike when \(i \leq h\), here we compute starting points, and we use \(\hat{S}^{(i)}\) for that.
        // For each \(\left(\left(\tilde{x}_{i}\right)_{J}, x_{0: i-h-1}\right)\) pair we estimate the optimal solution of (9)
        // by its linear approximation (11):
        foreach \(x_{0: i} \in \hat{S}^{(i)}\) do
            Compute the pseudo-inverse of \(\left[\begin{array}{l}J_{11} \\ J_{21}\end{array}\right]\), cf. (11) in Sec. 4.4
            foreach \(\left(\tilde{x}_{i}\right)_{J}\) in the new randomly generated points do
                Compute the starting point \(\hat{y}_{i-h: i}\) for (9) by solving (11), see in Sec. 4.4
                Save the residual \(\left\|\left[\begin{array}{l}J_{11} \\ J_{21}\end{array}\right]\left[\Delta x_{i-h: i-1}\right]-\left[\begin{array}{c}0 \\ J_{22} \Delta x_{i}\end{array}\right]\right\|_{2}^{2}\) together with \(\hat{y}_{i-h: i}\)
        // Select the candidate matches \(\left(\left(\tilde{x}_{i}\right)_{J}, x_{0: i-h-1}\right)\) :
        foreach \(\left(\tilde{x}_{i}\right)_{J}\) in the new randomly generated points do
            // We view \(x_{0: i-h-1}\) and \(\hat{y}_{i-h: i}\) as a function of \(\left(\tilde{x}_{i}\right)_{J}\), as on line 10
            Given \(\left(\tilde{x}_{i}\right)_{J}\), always select the match \(\left(x_{0: i-h-1}, \hat{y}_{i-h: i}\right)\) with the smallest residual
            From those matches whose residual is less then \(\epsilon_{\text {linear }}\), select at most \(m-1\) points
            // The additional \(m-1\) points are selected with farthest-first subsampling in \(x_{i}\)
            Add the selected matches to the candidate matches
        // Solve the nonlinear programs (9) from their estimated starting point:
        For each candidate match, solve (9) for \(y_{i-h, i}\) starting from \(\hat{y}_{i-h: i}\)
        Add the result to the set of points \(T\) if \(\left\|F_{i-h: i}\left(x_{0: i-h-1}, y_{i-h: i}\right)\right\| \leq \epsilon_{n l p}\) at the optimum
    // Try to repair the bound infeasible points:
    Call Algorithm 2 with \(T\), then replace \(T\) with the returned set
    // All points in \(T\) are bound feasible now, and satisfy \(F_{1: i}\left(x_{0: i}\right) \approx 0\)
    return \(T\)
```


## References

1. Aspen Technology, Inc.: Aspen Simulation Workbook, Version Number: V7.1. Burlington, MA, USA. EO and SM Variables and Synchronization, p. 110 (2009)
2. Baharev, A.: ManiSolve: A manifold-based approach to solve systems of equations (2019). URL https:// github.com/baharev/ManiSolve
3. Baharev, A., Domes, F., Neumaier, A.: A robust approach for finding all well-separated solutions of sparse systems of nonlinear equations. Numer. Algorithms 76, 163-189 (2017). https://doi.org/10.1007/s11075-016-0249-x. ISSN 1572-9265
4. Baharev, A., Neumaier, A.: A globally convergent method for finding all steady-state solutions of distillation columns. AIChE J. 60, 410-414 (2014)
5. 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)
6. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput. 15(6), 1373-1396 (2003). https://doi.org/10.1162/089976603321780317
7. Borg, I., Groenen, P.J.F.: Modern Multidimensional Scaling. Springer, New York (2005)
8. 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)
9. Bublitz, S., Esche, E., Tolksdorf, G., Mehrmann, V., Repke, J.U.: Analysis and decomposition for improved convergence of nonlinear process models in chemical engineering. Chem. Ing. Tech. 89(11), 1503-1514 (2017)
10. Dassault Systèmes AB. Dymola—Dynamic Modeling Laboratory. User Manual, vol. 2., Ch. 8. Advanced Modelica Support (2014)
11. Davis, T.A.: Direct methods for sparse linear systems. In: Higham, N.J. (ed.) Fundamentals of Algorithms. SIAM, Philadelphia (2006)
12. 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 (1995)
13. Doherty, M.F., Fidkowski, Z.T., Malone, M.F., Taylor, R.: Perry's Chemical Engineers' Handbook, Chapter 13, 8th edn, p. 33. McGraw-Hill Professional, New York (2008)
14. Doherty, M.F., Fidkowski, Z.T., Malone, M.F., Taylor, R.: Perry's Chemical Engineers' Handbook, Chapter 13, 8th edn, p. 69. McGraw-Hill Professional, New York (2008)
15. Donoho, D.L., Grimes, C.: Hessian eigenmaps: locally linear embedding techniques for high-dimensional data. Proc. Natl. Acad. Sci. 100(10), 5591-5596 (2003) ISSN 0027-8424. URL https://www.pnas.org/ content/100/10/5591
16. Dorn, C., Güttinger, T.E., Wells, G.J., Morari, M.: Stabilization of an unstable distillation column. Ind. Eng. Chem. Res. 37, 506-515 (1998)
17. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct Methods for Sparse Matrices. Clarendon Press, Oxford (1986)
18. Dulmage, A.L., Mendelsohn, N.S.: Coverings of bipartite graphs. Can. J. Math. 10, 517-534 (1958)
19. Dulmage, A.L., Mendelsohn, N.S.: A structure theory of bipartite graphs of finite exterior dimension. Trans. R. Soc. Can. Sec. 3(53), 1-13 (1959)
20. Dulmage, A.L., Mendelsohn, N.S.: Two algorithms for bipartite graphs. J. Soc. Ind. Appl. Math. 11, 183-194 (1963)
21. Erisman, A.M., Grimes, R.G., Lewis, J.G., Poole, W.G.J.: A structurally stable modification of HellermanRarick's $P^{4}$ algorithm for reordering unsymmetric sparse matrices. SIAM J. Numer. Anal. 22, 369-385 (1985)
22. Fletcher, R., Hall, J.A.J.: Ordering algorithms for irreducible sparse linear systems. Ann. Oper. Res. 43, 15-32 (1993)
23. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming. Brooks/Cole, Belmont (2003)
24. Golub, G.H., van Loan, C.F.: Matrix Computations, 3rd edn. The Johns Hopkins University Press, Baltimore (1996)
25. 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)
26. Güttinger, T.E., Morari, M.: Comments on "multiple steady states in homogeneous azeotropic distillation". Ind. Eng. Chem. Res. 35, 2816-2816 (1996)
27. Hellerman, E., Rarick, D.C.: Reinversion with preassigned pivot procedure. Math. Program. 1, 195-216 (1971)
28. 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, New York (1972)
29. HSL: A collection of Fortran codes for large scale scientific computation (2016). URL http://www.hsl.rl. ac.uk
30. Johnson, D.M., Dulmage, A.L., Mendelsohn, N.S.: Connectivity and reducibility of graphs. Can. J. Math. 14, 529-539 (1962)
31. 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)
32. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika 29(1), 1-27 (1964a). https://doi.org/10.1007/BF02289565. (ISSN 1860-0980)
33. Kruskal, J.B.: Nonmetric multidimensional scaling: a numerical method. Psychometrika 29(2), 115-129 (1964b). https://doi.org/10.1007/BF02289694. (ISSN 1860-0980)
34. Lewis, W.K., Matheson, G.L.: Studies in distillation. Ind. Eng. Chem. 24, 494-498 (1932)
35. Maaten, L.V.D., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. 9(Nov), 2579-2605 (2008)
36. Modelica: Modelica and the modelica association. https://www.modelica.org/, 2018. [Online; Accessed 14 Oct 2018]
37. Modelon, A.B.: JModelica.org User Guide, version 2.2. https://jmodelica.org/downloads/UsersGuide. pdf, 2018. [Online; Accessed 14 Oct 2018]
38. Naphthali, L.M., Sandholm, D.P.: Multicomponent separation calculations by linearization. AIChE J. 17, 148-153 (1971)
39. Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer, New York (2006)
40. OpenModelica: Openmodelica user's guide. https://openmodelica.org/doc/OpenModelicaUsersGuide/ latest/omchelptext.html, 2018. [Online; Accessed 14 Oct 2018]
41. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825-2830 (2011)
42. Petlyuk, F.B.: Distillation Theory and Its Application to Optimal Design of Separation Units. Cambridge University Press, Cambridge (2004)
43. Pothen, A., Fan, C.J.: Computing the block triangular form of a sparse matrix. ACM Trans. Math. Softw. 16, 303-324 (1990)
44. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500), 2323-2326 (2000). (ISSN 0036-8075)
45. 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 (1984)
46. 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 (1984)
47. Tenenbaum, J.B., Silva, V.D., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290(5500), 2319-2323 (2000). (ISSN 0036-8075)
48. Thiele, E., Geddes, R.: Computation of distillation apparatus for hydrocarbon mixtures. Ind. Eng. Chem. 25, 289-295 (1933)
49. Vadapalli, A., Seader, J.D.: A generalized framework for computing bifurcation diagrams using process simulation programs. Comput. Chem. Eng. 25, 445-464 (2001)
50. Van Der Maaten, L.: Accelerating t-SNE using tree-based algorithms. J. Mach. Learn. Res. 15(1), 32213245 (2014)
51. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. 106, 25-57 (2006)
52. Zhang, Z.Y., Zha, H.Y.: Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. J. Shanghai Univ. (English Edition) 8(4), 406-424 (2004). https://doi.org/10.1007/s11741-004-0051-1. (ISSN 1863-236X)
53. Zhang, Z., Wang, J.: MLLE: modified locally linear embedding using multiple weights. In: Schölkopf, B., Platt, J.C., Hoffman, T. (eds.) Advances in Neural Information Processing Systems, vol. 19, pp. 1593-1600. MIT Press, Cambridge (2007)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.


[^0]:    The research was funded by the Austrian Science Fund (FWF): P27891-N32. The authors wish to thank Saskia Bublitz (TU Berlin) for her valuable feedback on the manuscript.

    Ali Baharev
    ali.baharev@gmail.com
    1 Faculty of Mathematics, University of Vienna, Oskar-Morgenstern-Platz 1, 1090 Vienna, Austria

