Failure Modes of Tearing and a Novel Robust Approach

Ali Baharev   Arnold Neumaier   Hermann Schichl

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

Abstract

State-of-the-art Modelica implementations may fail in various ways when tearing is turned on: Completely incorrect results are returned without a warning, or the software fails with an obscure error message, or it hangs for several minutes although the problem is solvable in milliseconds without tearing. We give three detailed examples and an in-depth discussion why such failures are inherent in tearing and cannot be fixed within the traditional approach.

Without compromising the advantages of tearing, these issues are resolved for the first time with staircase sampling. This is a non-tearing method capable of robustly finding all well-separated solutions of sparse systems of nonlinear equations without any initial guesses. Its robustness is demonstrated on the steady-state simulation of a particularly challenging distillation column. This column has three solutions, one of which is missed by most methods, including problem-specific tearing methods. All three solutions are found with staircase sampling.

Keywords: decomposition methods, diakoptics, large-scale systems of equations, numerical instability, sparse matrices, staircase sampling

1 Introduction

Definitions. Traditional tearing, cf. (Elmqvist, 1978; Elmqvist and Otter, 1994; Mattsson et al., 1999; Carpanzano, 2000; Cellier and Kofman, 2006; Täuber et al., 2014), is the representation of a sparse system of nonlinear equations

$$f(x) = 0, \text{ where } f:\ \mathbb{R}^n \mapsto \mathbb{R}^n, \tag{1}$$

in a permuted form where most of the variables can be computed sequentially once a small auxiliary system has been solved. More specifically, given permutation matrices P and Q such that after the transformation

$$\begin{bmatrix} g \\ h \end{bmatrix} = Pf, \qquad \begin{bmatrix} y \\ z \end{bmatrix} = Qx, \tag{2}$$

$g_i(y,z) = 0$ can be rewritten in the equivalent explicit form

$$y_i = \tilde{g}_i(y_{1:i-1}, z) \tag{3}$$

using appropriate symbolic transformations. Here the shorthand $p{:}q$ is used for the index set $p, p+1, \ldots, q$ where $p \leq q$. Equation (3) implies that the sparsity pattern of the Jacobian of Pf is

$$J = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \text{ where } A \text{ is lower triangular,} \tag{4}$$

J is therefore **bordered lower triangular**. We will use the abbreviation **BLTF** which stands for bordered lower triangular form. We refer to a particular choice of P, Q, g, h, y, and z satisfying equations (3) and (4) as an **ordering**. Given an ordering, the system of equations $f(x) = 0$ can be written as

$$\begin{aligned} g(y,z) &= 0 \\ h(y,z) &= 0. \end{aligned} \tag{5}$$

The requirement (3) that $g_i(y,z) = 0$ can be made explicit in y_i essentially means that we can obtain y from z by a nonlinear triangular solve. Substituting the result $y = \bar{g}(z)$ into h yields $h(\bar{g}(z), z) = 0$ or

$$r(z) = 0. \tag{6}$$

That is, the original nonlinear system (1) is reduced to the (usually much) smaller system $r(z) = 0$. A commonly used objective is to find an ordering that minimizes the **border width** $d := \dim z$ of J. For a given z, we call the value of $r(z)$ the residual vector or simply the **residual**.

Advanced tearing methods. There are other, more sophisticated variants of tearing, summarized in Table 1. These try to reduce the size of the final system (6) by relaxing the requirements of (3) (by allowing implicit equations for example) and/or allowing A in (4) to have a form other than lower triangular. These enhancements share that the computation of y for a given z only involves fast and numerically stable algorithms such as solving implicit univariate equations or small systems of equations. Another recent approach tries to balance between minimizing the border width and preserving the sparsity during the elimination (Magnusson and Åkesson, 2017). The reader is referred to (Baharev et al., 2017a) for an in-depth discussion of the variations on tearing. To keep the examples short and simple, we only discuss the failure modes of traditional tearing in the present paper.

Importance: initializing and solving DAE systems. 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 (Pantelides, 1988; Unger et al., 1995),

DOI
10.3384/ecp17132353

Table 1. The sparsity pattern of the Jacobian in the different variants of tearing, classified by the largest subproblem size and the level of subproblem nesting. The present paper discusses the failure modes of traditional tearing only.

Largest subproblem	Maximum level of subproblem nesting 1	 ℓ
univariate equations only	bordered lower triangular (traditional tearing)	—
$k \times k$ systems of equations	bordered block lower triangular (tearing with diagonal blocks)	nested bordered block lower triangular (hierarchical tearing)

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 (Kröner et al., 1997). Tearing usually also helps to speed up the solution process of DAE systems thanks to the reduced problem size (Elmqvist, 1978; Elmqvist and Otter, 1994; Mattsson et al., 1999; Carpanzano, 2000; Cellier and Kofman, 2006).

Even though mature equation-based component-oriented modeling environments are available, e.g., Modelica (Mattsson et al., 1998; Tiller, 2001; Fritzson, 2004) for multi-domain modeling of heterogeneous complex technical systems, and gPROMS, ASCEND (Piela et al., 1991) and EMSO (de P. Soares and Secchi, 2003) for chemical process modeling, simulation and optimization, etc., 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, requiring user-provided good initial values for the variables (Vieira and Jr, 2001; Bachmann et al., 2007; Sielemann and Schmitz, 2011; Sielemann et al., 2013; Ochel and Bachmann, 2013).

2 Demonstrative examples

Here we show the behavior of the latest release of Dymola (Version 2017 FD01 (32-bit), 2016-10-11) and OpenModelica (v1.11.0 (64-bit); February 6, 2017) on three examples. Examples 1 and 2 demonstrate that applying tearing can lead to completely incorrect results or to initialization failure. However, correct results are obtained for both examples when tearing is turned off. Example 3 is about performance: It shows that tearing can slow down the solution process drastically. Dymola can easily hang for minutes on problems that are otherwise solvable in milliseconds without tearing. The causes are discussed in Section 3. The examples trigger failure only if the tearing is performed according to the specified ordering. The Modelica source files are available in the GitHub repository of the (Online Supplement).

Example 1: The residual is overly sensitive to the changes in the tear variable. We solve the following 20×20 linear system in a Newton step:

$$x_{i-1} + 10x_i + x_{i+1} = 1.2 \quad i = 1{:}20, \tag{7}$$

where $x_0 := 0.1$ and $x_{21} := 0.1$ to keep the formulas simple. The only tear variable is x_1; the residual is given by the last equation ($i = 20$). The exact solution is $x_i = 0.1$ for $i = 1{:}20$. Both Dymola and OpenModelica return completely incorrect results, for example, $x_{20} = 32.03$ and $x_{20} = 85.82$, respectively, but claim that the simulation was successful.

Example 2: The residual is insensitive to the changes in the tear variable. We solve the following 20×20 linear system in a Newton step:

$$x_{i-1} + x_i + 15x_{i+1} = 17 \qquad i = 1{:}20, \tag{8}$$

where $x_0 := 1$ and $x_{21} := 1$ to keep the formulas simple. The only tear variable is x_1; the residual is given by the last equation ($i = 20$). The exact solution is $x_i = 1$ for $i = 1 : 20$. Dymola fails with an unhelpful error message, and does not return any result. OpenModelica emits some confusing intermediate warnings and reports at the end of the computations that “simulation process finished successfully”. But it returns incorrect results; for example, x_1 still equals the initial guess, as if nothing had happened.

Example 3: Unacceptable border width, leading to very poor performance. We solve the following $N \times N$ linear system in a Newton step:

$$\sum_{i=1}^{N} x_i = N \tag{9}$$

$$x_i + x_N = 2 \qquad i = 1 : N-1, \tag{10}$$

and we assume that the only variable that can be eliminated is x_N from equation (9); this can be due to the nonlinearities of the original problem (whose Newton step we see here). All other variables are tear variables, and all other equations are residuals. For $N = 300$, the problem is solved by Dymola in 74 seconds and by OpenModelica in 37 seconds. As we argue in Sec. 3.3, the problem is solvable in milliseconds: For $N = 300$ (the largest dimension permitted in the free trial version we used), the AMPL modeling environment (Fourer et al., 2003) is faster than Dymola and OpenModelica by factors of more than 1200 and 600, respectively. The performance of the Modelica implementations rapidly deteriorates as the problem size increases: For $N = 500$, Dymola hangs for more than 6 minutes, and OpenModelica takes more than 1.5 minutes.

DOI
10.3384/ecp17132353

The examples are intentionally chosen to be easy. In challenging real-life examples, like the one in Sec. 5.3, it is hard to identify and understand the reasons of the failures, because different failure modes usually occur simultaneously and interact with each other. However, the simplicity of the present examples allows us to gain (in Section 3) important insights into the reasons *why* tearing fails or causes very poor performance. The examples were chosen to demonstrate the reasons in isolation, one at a time. This is also the reason why we picked linear examples; they should be regarded as the linear system solved in a Newton step.

3 In-depth discussion of the examples

Pathological input problems are ignored throughout this paper, for example when the system (1) has conflicting equations and as a consequence it is infeasible, when (1) is singular, when it is poorly scaled, or when the problem has a huge number of solutions, etc. While these edge cases are interesting and important, a non-tearing approach can fail in these cases too, and therefore such failures are not specific to tearing. Throughout this paper we only focus on those failure modes that are specific to tearing: We assume that the input problem (1) is feasible and properly scaled, has at most a small number of real solutions, and that these solutions can be found with an appropriate non-tearing approach using 64-bit floating-point arithmetic. Traditional tearing can fail even if all these assumptions are met.

3.1 Example 1: uncontrollable residual

The residual can become practically uncontrollable because it depends too sensitively on the tear variables. To see this we consider the system

$$x_{i-1} + 10x_i + x_{i+1} = 1.2 \quad \text{for } i = 1{:}20 \tag{11}$$

where $x_0 := 0.1$ and $x_{21} := 0.1$ to keep the formulas simple. The exact solution is $x_i = 0.1$ for $i = 1{:}20$. The coefficient matrix of the system (11) is a strictly diagonally dominant tridiagonal matrix (cf. Fig. 1 top), hence solving (11) with Gaussian elimination produces excellent results even without pivoting (Golub and van Loan (1996, Ch 3.4.10)). As it was demonstrated in Section 2, traditional tearing fails on this easy problem.

We order the coefficient matrix of (11) into BLTF with minimal border width by moving x_1 to the border, see on the bottom of Fig. 1. Given an initial guess for x_1, the formula for the forward substitution along the diagonal is:

$$x_{i+1} = -x_{i-1} - 10x_i + 1.2 \quad \text{for } i = 1{:}19, \tag{12}$$

and the residual $r := -x_{19} - 10x_{20} + 1.1$ is a univariate function of x_1, that is, we have to solve the univariate equation $r(x_1) = 0$ for x_1. Because of the factor 10 in (12), the error in our guess for x_1 is multiplied roughly by a factor of 10 in each step of the elimination according to (12). There are 19 steps in (12), meaning that the error in x_1 will be magnified roughly by a factor of 10^{19} till we compute the residual. This has catastrophic consequences. There is no machine representable number for x_1 such that after eliminating all the other variables according to (12) r is sufficiently close to zero: The two closest 64-bit floating-point numbers enclosing 0.1 give approximately 85.82 and -101.03 for x_{20}, respectively, due to the roughly 10^{19} factor magnifying the error in x_1. In other words, (11) is literally unsolvable in 64-bit floating-point arithmetic with traditional tearing, whereas solving it with Gaussian elimination is numerically stable even without pivoting. The failure is not due to a single ill-conditioned elimination step but the sequence of well-conditioned steps becoming ill-conditioned when they are chained together as in (12).

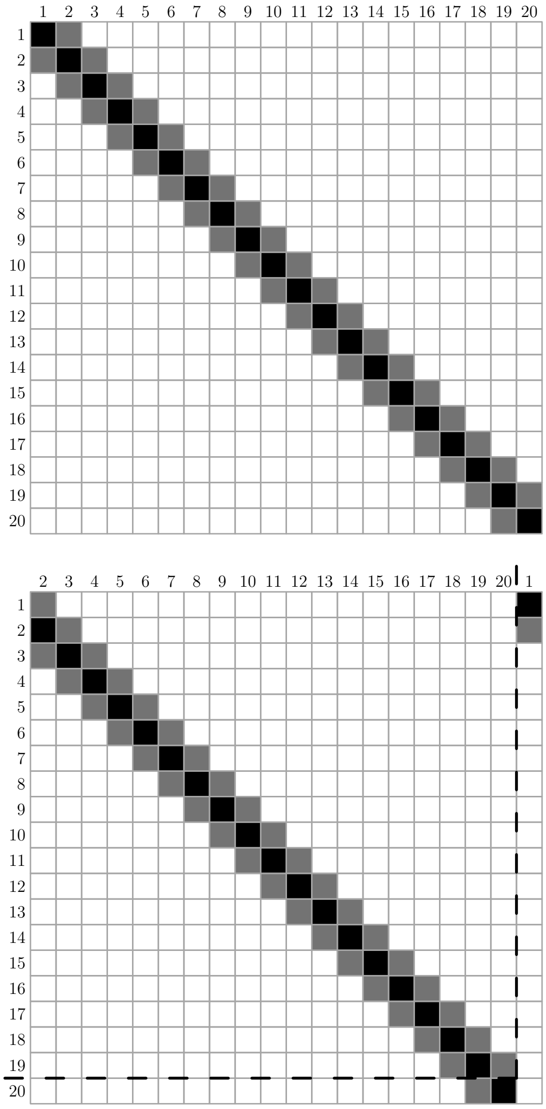

Figure 1. Top: The sparsity pattern of the coefficient matrix of problem (11). Black entries correspond to 10, gray entries to 1. Bottom: The same matrix ordered to bordered lower triangular form. The leading lower triangular submatrix, surrounded by dashed lines, is singular to working precision in 64-bit floating point arithmetic.

A similar behavior in the nonlinear case makes the problem practically unsolvable with iterative solvers, even if the original problem is easy to solve without tearing. Distillation columns are real-life examples where such failures happen, c.f. Doherty et al. (2008).

For those familiar with linear algebra: The condition number estimate of the coefficient matrix of (11) is 1.5 (symmetric, strictly diagonally dominant tridiagonal matrix), whereas the condition number estimate of the leading lower triangular matrix of the BLTF is $9 \cdot 10^{16}$, meaning that it is singular to working precision in 64-bit floating point arithmetic. See also Golub and van Loan (1996, Ch 3.3, and 3.5.4).

3.2 Example 2: insensitive residual

It can also happen that the residual shows practically no response to changes in the tear variables. Such an example is the following:

$$x_{i-1} + x_i + 15x_{i+1} = 17 \qquad i = 1{:}20, \tag{13}$$

where $x_0 := 1$ and $x_{21} := 1$ to keep the formulas simple. It is easy to see that the solution is $x_i = 1$ $(i = 1 : 20)$. Solving (13) with a non-tearing approach is not a challenge.

Making x_1 the only tear variable, and moving it to the border makes the resulting BLTF have minimal border width, see Fig. 2. Given an initial guess for x_1, the formula for the forward substitution along the diagonal is:

$$x_{i+1} = \frac{1}{15}(-x_{i-1} - x_i + 17) \qquad \text{for } i = 1{:}19, \tag{14}$$

and the residual

$$r := -x_{19} - x_{20} + 2 \tag{15}$$

is a univariate function of x_1, that is, we have to solve the univariate equation

$$r(x_1) = 0 \tag{16}$$

for x_1. As it can be seen from (14), the error in our estimate for x_1 is divided roughly by a factor of 15 in each step of the recursion, that is, the error attenuates in an exponential rate. As a consequence, we get $r = 0.0000000000$ (with 10 decimals) for both $x_1 = -1$ and $x_1 = 3$. This is unacceptable, since x_1 and many of the eliminated variables are still very far from the solution. The reason of the failure is that the value of $r(x_1)$ provides no information about the desired update of x_1: The final equation (16) is satisfied even with grossly erroneous x_1 values.

In the nonlinear case, similar issues can lead to failures of the tearing approach. Distillation columns are again real-life examples where such failures happen. In fact, distillation columns are difficult for tearing methods because one part of the column can magnify the error in the tear variables with exponential rate (similarly to (12)), while the remaining part attenuates it with an exponential rate (similarly to (14)). This in turn can trigger two failure modes of tearing at the same time: the one described in this section, and the one from the previous section.

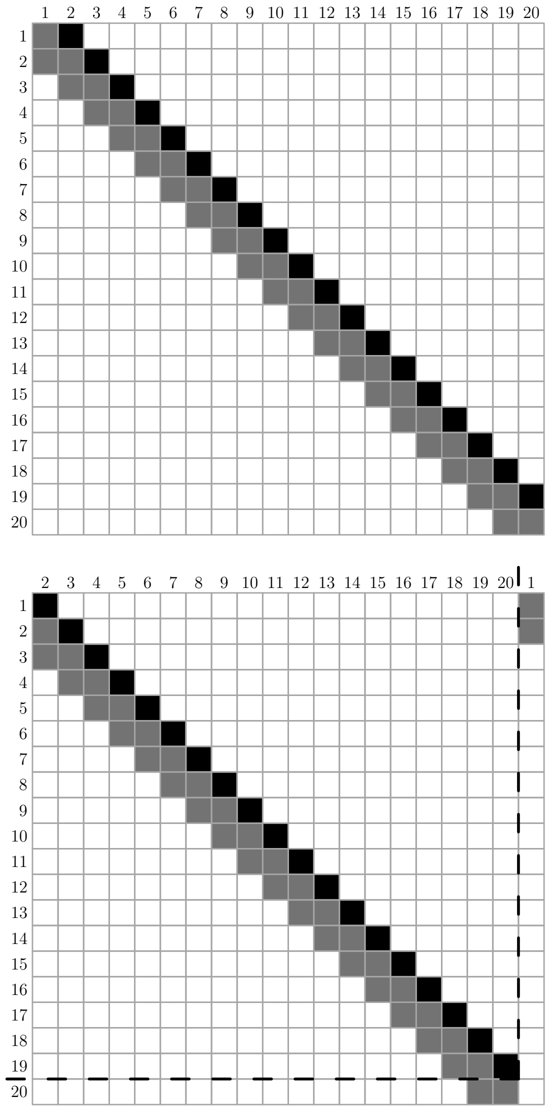

Figure 2. Top: The sparsity pattern of the coefficient matrix of problem (13). Black entries correspond to 15, gray entries to 1. Bottom: The same matrix ordered to bordered lower triangular form.

As already stated, (13) is not a challenging problem; it is just that traditional tearing fails. For those familiar with linear algebra: Although problem (13) is mildly ill-conditioned, the condition number estimate is $7 \cdot 10^{11}$, one can still get the result with several accurate significant digits in 64-bit floating point arithmetic (Golub and van Loan (1996, Ch 3.3, and 3.5.4)).

3.3 Example 3: unacceptably wide border

Wide border due to tearing incautiously. The primary motivation behind tearing is to speed up the solution process (Dymola User Manual, Ch. 8.8.2, pp. 433-434). However, tearing can significantly hurt performance, especially if it is applied without any caution; Example 3 at equations (9) and (10) in Sec. 2 is just one example of that. Here the problem is that the border width is proportional to the size of the original problem. In case of Dymola, it slows down the model generation and compilation drastically; for $N = 500$, the software hangs for more than 6 minutes. OpenModelica hangs for issues that are most likely independent of tearing, but we failed to track down the exact causes.

In comparison, solving the instance $N = 300$ with AMPL takes only 61 milliseconds, although AMPL adequately performs all relevant tasks, namely:
(i) reading and parsing the model file written in the AMPL modeling language,
(ii) instantiating the model,
(iii) flattening,
(iv) compiling the C code,
(v) generating binary opcodes for the virtual AMPL stack machine,
(vi) launching the external solver and executing the code to compute the solution, and
(vii) writing back the results for the modeling environment.
AMPL and the Modelica implementations have to go through basically the same steps during the solution process of Example 3; the computational work to be done is essentially the same for each modeling environment.

For those familiar with linear algebra: The problem here is that tearing results in catastrophic fill-in (Duff et al., 1986, Ch. 7). AMPL and the solver it invokes, IPOPT (Wächter and Biegler, 2006) with MA27 from (HSL, 2017), avoid this by not perform tearing, and by using proper sparse data structures and sparse linear algebra. As far as we can tell, the state-of-the-art Modelica implementations seem to perform $O(n^2)$ or more operations, and this can hurt performance already on relatively small problems. See also (Duff et al., 1986, Ch. 5.8) regarding the so-called $O(n^2)$ traps.

Wide border due to trying to create a maximum-weight diagonal. Let A denote the coefficient matrix of (11). The reason why tearing failed in Section 3.1 is that the largest entries of A became off-diagonal after A was ordered to BLTF, and the elimination happened along the diagonal. The straightforward attempt to fix this is to mimic complete pivoting (Golub and van Loan (1996, Ch. 3.4.8)): We order A into BLTF but instead of having a minimal border width, our objective is to have a maximum-weight diagonal on the lower triangular part. Indeed, such approaches were proposed in the past, see for example Westerberg and Edie (1971a,b) and Gupta et al. (1974).

Although creating a maximum-weight diagonal mitigates the issue of uncontrollable residual, it can easily lead to the opposite problem, to the issue of the insensitive residual: The example of Sec. 3.2 has a maximum-weight diagonal and tearing fails on that easy problem. Also, compare Fig. 1 with Fig. 2 where the subdiagonal became maximum-weight. In short, creating a maximum-weight diagonal can turn one failure mode to another.

However, there is another issue that creating a BLTF with maximum-weight diagonal can also cause: It can produce a BLTF whose border width is proportional to the size of the input matrix, whereas if we minimized the border width, the border width would be a small constant, independent of the problem size. Table 2 and Figure 3 show examples of such disastrous cases. Since the final system (6) is dense, this means that tearing turns (in the course of the elimination) a sparse problem into a dense problem whose size is proportional to the original problem. Such dense problems become intolerably expensive to solve as their size grows.

4 Failing due to a single elimination step

In the previous section we discussed failure modes where a sequence of eliminations led to the failure. In this section we show additional examples where tearing fails due to a single elimination step, because it leads to an undefined operation (Sec. 4.1) or to a floating-point exception (Sec. 4.2), or it is multivalued (Sec. 4.3). We comment on the usual workaround as seen in the state-of-the-art Modelica environments, and propose a novel and better alternative in Sec. 4.4.

4.1 Undefined elimination step

For the sake of demonstration let us assume that in an elimination step in (3) we want to eliminate x_3 from

$$x_1 - x_2 x_3 = 0, \tag{17}$$

so we rearrange (17) as

$$x_3 := \frac{x_1}{x_2}. \tag{18}$$

However, this symbolic transformation is invalid if $x_2 = 0$. If x_2 happens to be 0 during the iteration in the tear variables, eliminating x_3 according to (18) would lead to division by zero, whereas the original equation (17) does not suffer from this issue. It is not only division that is problematic: Another example of this kind of failure is a negative argument to the logarithm function during the iteration in the tear variables (when working over the set of real numbers). In general, arguments outside the domain of the functions involved lead to failure of traditional tearing.

4.2 Floating-point exception in an elimination step

Floating-point exceptions can easily occur in systems involving an exponential. For example, let the tear variables be $x_{41} := 440$ and $x_{43} := 0.0$, and the elimination steps are:

$$\begin{aligned} x_{42} &:= \exp(x_{41} + 273.15) \\ x_{44} &:= x_{42} x_{43}. \end{aligned} \tag{19}$$

Table 2. Unacceptably wide border when the matrix is ordered to BLTF with maximum-weight diagonal on the lower triangular part.

Matrix	Minimal border width	Border width with max-weight diagonal	Pattern in Figure (3)
Tridiagonal	1	$\frac{1}{2}n$	top row
Pentadiagonal	2	$\frac{2}{3}n$	(not shown)
Arrowhead	1	$n-1$	bottom row

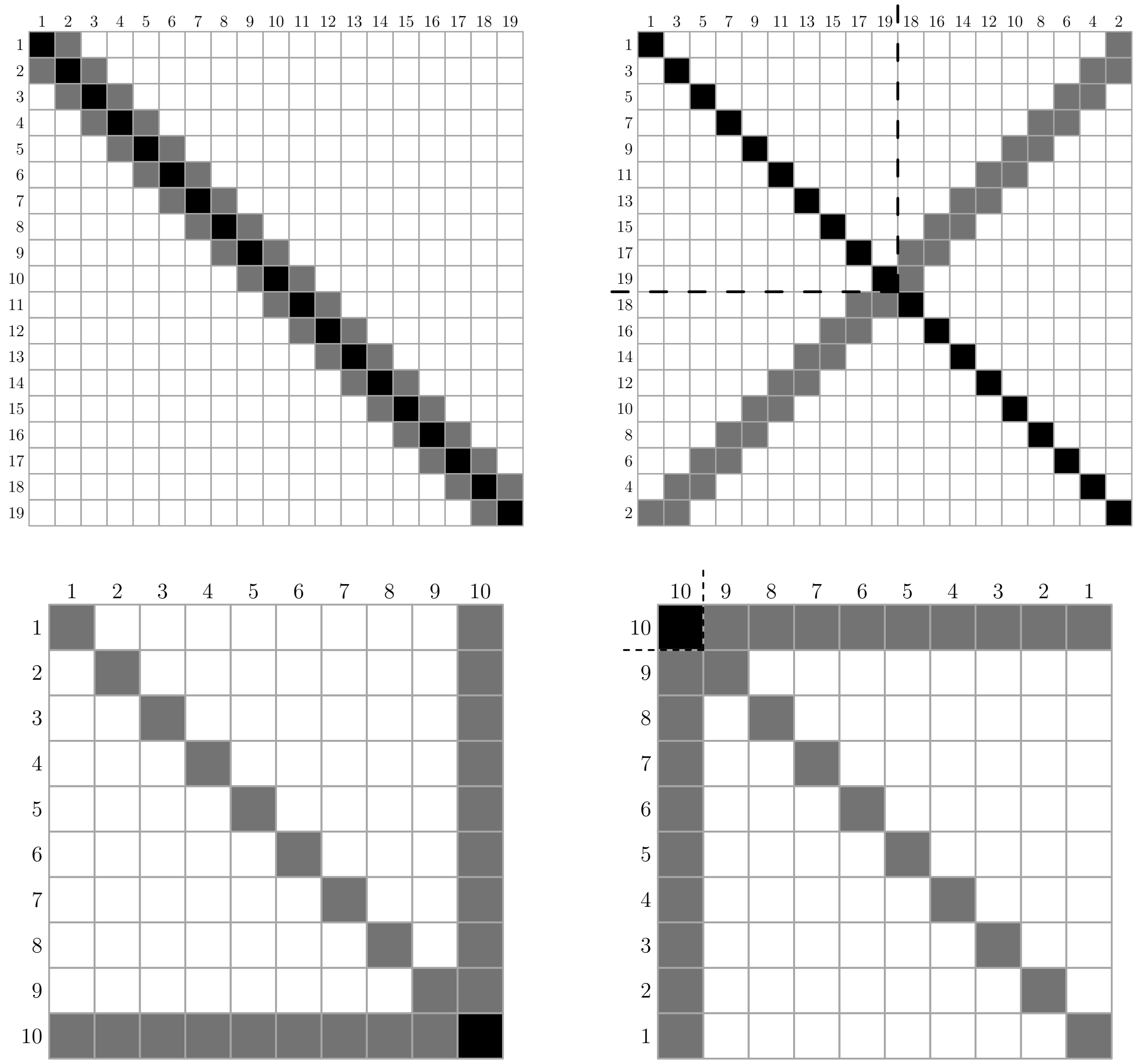

Figure 3. The left column shows the input matrices, the right column the corresponding matrices ordered to BLTF with maximum-weight diagonal; black entries correspond to 10, gray entries to 1. The tridiagonal matrix of size n will have a border width $\frac{n}{2}$ (top row). In the worst case, when the optimal ordering is the arrowhead matrix (bottom row), the border width is $n-1$.

Unlike previously, the elimination steps are mathematically sound here. However, in 64-bit floating-point arithmetic we get: $x_{42} = \texttt{inf}$ and $x_{44} = \texttt{nan}$, where `inf` and `nan` stand for infinity and not a number, see (IEEE 754). The eliminations cannot be continued as x_{42} does not have any correct significant digits, and x_{44} is not a number. Unfortunately, similar failures are not at all uncommon in practice, especially with thermodynamic models.

Getting a floating-point exception due to a single elimination step is an extreme case. More common is that the error in the tear variables is amplified during a sequence of elimination steps, as already discussed in Sections 3.1, and this leads to an interaction between failure modes by triggering a floating-point exception. For example, let us assume that x_{41} is not a tear variable, but an eliminated one, and the sequence of eliminations leading up to x_{41} yields the value 440 for x_{41} due to amplification of the error in the tear variables. This is similar to Example 1 of Sec. 3.1 where the value of x_{20} was three orders of magnitude off compared to the true value. In Example 1 it was the residual that became uncontrollable, here it is a floating-point exception that causes the ultimate failure.

4.3 Multivalued elimination step

In the equation

$$x_1^2 + 2x_1x_2 - 1 = 0, \tag{20}$$

the elimination of x_1 requires to solve this equation for x_1. However, there are two possibilities to perform this:

$$x_1 = -x_2 + \sqrt{x_2^2+1} \quad \text{and} \quad x_1 = -x_2 - \sqrt{x_2^2+1}. \tag{21}$$

To continue the remaining eliminations, we would have to know which solution for x_1 will remain feasible, or continue with both possibilities for x_1. If we ignore the fact that the elimination step is multivalued, we either risk losing solutions, or we risk that we continue with that value for x_1 that becomes infeasible in later eliminations. This failure mode is simply ignored in state-of-the-art Modelica implementations, for example, "[the solver] *hopefully* returns the solution closest to the guess value" (Dymola User Manual, Ch. 8.9.2, p. 442); the emphasis is ours. If we want to find all well-separated solutions of a nonlinear system of equations, the kind of applications discussed in (Baharev et al., 2016), this is unacceptable.

4.4 Avoiding floating-point exceptions, undefined and multivalued elimination steps

The commonly seen workaround in state-of-the-art Modelica tools to avoid undefined and multivalued elimination steps is to chose only linearly appearing variables with non-zero coefficients as tear variables. Although this workaround is easy to implement, it is also overly limiting in the choice of the tear variables. This can lead to a BLTF with unacceptably wide border and eventually to very poor performance, because it excludes variables that are perfectly eligible to become a tear variable. The same holds for disallowing division by variables in an attempt to avoid undefined elimination steps and floating-point exceptions. As for multivalued eliminations, it is left to chance whether a feasible solution is found or not, see Sec. 4.3.

It is moderately easy to avoid single elimination steps that can potentially become problematic depending on the actual values of the variables involved: In Baharev et al. (2017b) we proposed a novel pre-processing technique based on interval arithmetic for recognizing single-step eliminations that are *guaranteed* to be single-valued and numerically well-behaved, irrespective of the actual value of the variables involved. (Of course, it does not prevent a sequence of eliminations from becoming ill-conditioned.) Our approach offers more flexibility in the choice of tear variables than the common workaround, and it does that without risking any numerically troublesome operation. The increased flexibility in the choice of the tears can help to reduce the border width of the BLTF significantly. As for multivalued elimination steps, nothing is left to chance in our approach. For those familiar with linear algebra: Our method is, in some sense, a nonlinear extension of threshold pivoting (Duff et al., 1986, Ch. 4.4).

5 A novel robust approach

Staircase sampling was inspired by (Baharev and Neumaier, 2014), and proposed in (Baharev et al., 2016) to mitigate all of the issues listed in Sections 3 and 4. A detailed presentation of this method is outside the scope of this paper; here we only sketch the basic idea.

The subsystem

$$g(y,z) = 0 \tag{22}$$

in (5) is an underdetermined system of equations; it has infinitely many solutions per our assumptions in the first paragraph of Section 3. The aim of staircase sampling is to find a small set of points such that every solution of (22) is close to one of the points in this set. We call this small set of points the sample: It is an approximation to a sample from the infinitely many solutions of (22). The sample is built up incrementally, similarly to the usual tearing approach. Staircase sampling requires finite and reasonable lower and upper bounds on all of the variables; this is needed to allow an adequate sampling of the search space.

Staircase sampling starts with an *entire set* of values for z (a scattered set of points between the variable bounds), and not just with a single value for z as in the usual tearing approach. The algorithm then proceeds similarly to the common tearing algorithms, and it performs eliminations. A minor difference compared to (3) is that staircase sampling solves small nonlinear systems in the elimination steps, that is, it performs block elimination. The fundamental difference is that after each block elimination step, the points are redistributed, and a subvector of y is recomputed as necessary. The goal of this redistribution algorithm is to improve the spatial distribution of the points between the variable bounds: It discards points that are too close, and it inserts new points where the search space has become deserted. The details of the redistribution algorithm are discussed in (Baharev et al., 2016). Staircase sampling returns a set of scattered points, satisfying (22) fairly well. In the current implementation, those points are chosen that violate (5) the least, and they are used as a starting point for large-scale sparse solvers that target solving (1) directly. In the future, interpolation and extrapolation on the complete set of scattered points will be used to find starting points that approximately satisfy (1).

5.1 How staircase sampling resolves the failure modes of traditional tearing

We now compare staircase sampling to traditional tearing from the point of view of the the failure modes; the items are enumerated in the same order as in Sections 3 and 4.

1. As shown in Sections 3.1 and 3.2, the error in our initial estimate for z can grow or attenuate exponentially even in simple cases, ultimately leading to the failure of traditional tearing. Staircase sampling breaks this exponential change in the redistribution step; the error accumulation in an exponential rate is not possible.

2. As a consequence of the previous point, we can minimize the border width in our ordering algorithm without having to worry about the exponential error growth rate during the eliminations. An exact ordering algorithm to minimize the border width is given in (Baharev et al., 2017b). Furthermore, staircase sampling works on so-called staircase triangular matrices, and those matrices allow more flexibility in the orderings than the BLTFs do.

3. A single block elimination step can also fail, however, this is usually not an issue. Staircase sampling works with a set of points, losing some of them is typically not a problem: New points are inserted after each block elimination step in the redistribution algorithm, which makes up for the lost points.

4. Staircase sampling builds up a set of solution vectors, not just a single solution vector at a time as in traditional tearing. As a consequence, multivalued elimination steps are handled naturally.

5.2 A note on plotting vectors in 2 dimensions

Here we explain how the starting points and solution vectors will be plotted in the next section. We first select a subset of the variables according to an appropriate rule; for example, we select the methanol composition in each device of the system that we are simulating. Let us assume that we have selected a 20-dimensional subset. We then draw each 20-dimensional vector as a curve in 2 dimensions by connecting the points (x_i, i) $(i = 1{:}20)$ with adjacent indices, as shown in Fig. 4. The connecting lines have no meaning, but allow us to plot without ambiguity several vectors in one figure.

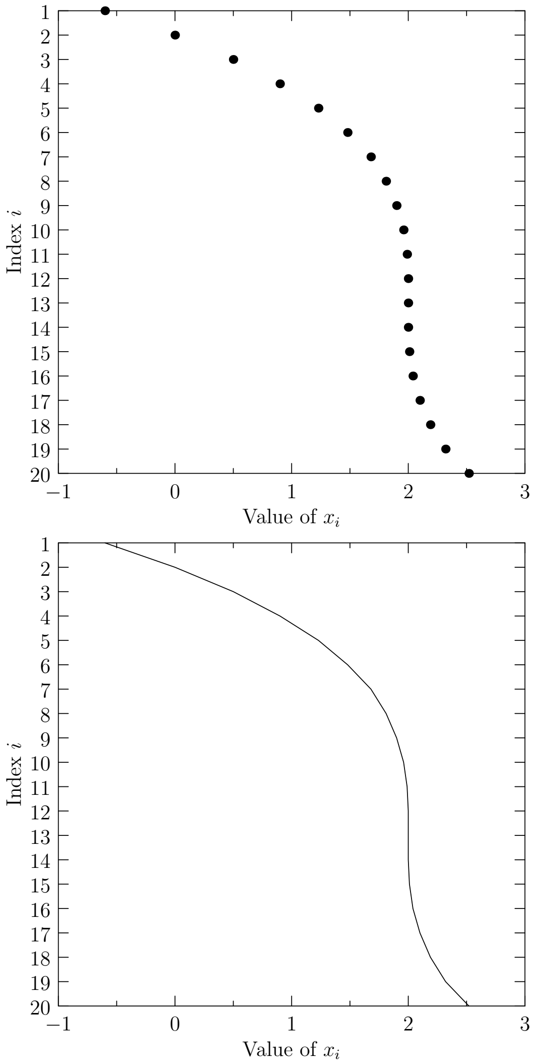

Figure 4. Top: Plotting the 20-dimensional vector x in 2 dimensions by placing a dot at (x_i, i) for $i = 1{:}20$. Bottom: To indicate that the dots belong to the same vector, we connect the neighboring points with linear lines, and we may omit the dots. The connecting lines have no meaning, but allow us to plot without ambiguity several vectors in one figure.

5.3 Demonstration test case

The robustness of staircase sampling is demonstrated on a particularly challenging distillation column. The model and its parameters correspond to the Auto model (Güttinger et al., 1997). The problem has three steady-state solutions: two stable steady-state branches and an unstable branch. Both the inside-out procedure (Boston and Sullivan, 1974) and the simultaneous correction procedure (Naphthali and Sandholm, 1971) were reported to miss the unstable steady-state solution, see (Vadapalli and Seader, 2001) and (Kannan et al., 2005). However, all steady-state branches were computed either with the AUTO software

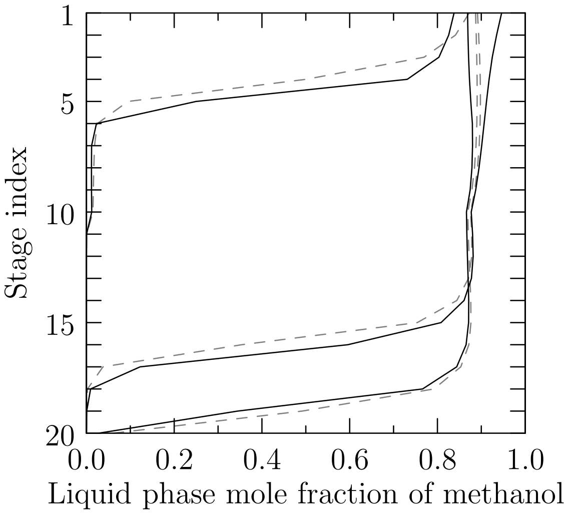

Figure 5. The three steady-state solutions (dashed gray lines) and those generated starting points (solid black lines) that are the closest to them. The gradient-based solver IPOPT converges to the nearest solution when started from the corresponding starting point.

package (Doedel et al., 1995) or with an appropriate continuation method (Güttinger et al., 1997; Vadapalli and Seader, 2001; Kannan et al., 2005). The initial estimates were carefully chosen with the ∞/∞ analysis (Bekiaris et al., 1993; Güttinger and Morari, 1996), and special attention was paid to the turning points and branch switching.

Our goal with staircase sampling was to find all solutions automatically, without any initial estimates, without relying on any domain-specific knowledge, and without any human interaction. This goal was achieved: All three steady-state solutions are found when IPOPT (Wächter and Biegler, 2006) is run from the starting points generated with staircase sampling. Figure 5 shows the three steady-state solutions of a 20-stage column and those starting points that are the closest to them; see also Sec. 5.2 as to how the solution vectors are plotted. For a more detailed discussion of this example, and for other examples see (Baharev et al., 2016).

Despite these promising results, the practical applicability and limitations of staircase sampling are yet to be explored, and a benchmark suite with real-world problems would be needed for that.

Acknowledgement

The research was funded by the Austrian Science Fund (FWF): P27891-N32. Support by the Austrian Research Promotion Agency (FFG) under project numbers 846920 and 853930 is thankfully acknowledged.

References

B. Bachmann, P. Aronßon, and P. Fritzson. Robust initialization of differential algebraic equations. In *1st International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (Berlin; Germany; July 30; 2007)*, Linköping Electronic Conference Proceedings, pages 151–163. Linköping University Electronic Press; Linköpings universitet, 2007.

A. Baharev, H. Schichl, and A. Neumaier. Decomposition methods for solving nonlinear systems of equations. Submitted, 2017a. URL `http://reliablecomputing.eu/baharev_tearing_survey.pdf`.

A. Baharev, H. Schichl, and A. Neumaier. Ordering matrices to bordered lower triangular form with minimal border width. Submitted, 2017b. URL `http://reliablecomputing.eu/baharev_tearing_exact_algorithm.pdf`.

Ali Baharev and Arnold Neumaier. A globally convergent method for finding all steady-state solutions of distillation columns. *AIChE J.*, 60:410–414, 2014.

Ali Baharev, Ferenc Domes, and Arnold Neumaier. A robust approach for finding all well-separated solutions of sparse systems of nonlinear equations. *Numerical Algorithms*, pages 1–27, 2016. doi:10.1007/s11075-016-0249-x. URL `https://doi.org/10.1007/s11075-016-0249-x`.

N. Bekiaris, G. A. Meski, C. M. Radu, and M. Morari. Multiple steady states in homogeneous azeotropic distillation. *Ind. Eng. Chem. Res.*, 32:2023–2038, 1993.

J. F. Boston and S. L. Sullivan. A new class of solution methods for multicomponent, multistage separation processes. *Can. J. Chem. Eng.*, 52:52–63, 1974.

Emanuele Carpanzano. Order reduction of general nonlinear DAE systems by automatic tearing. *Mathematical and Computer Modelling of Dynamical Systems*, 6(2):145–168, 2000.

François E Cellier and Ernesto Kofman. *Continuous system simulation*. Springer Science & Business Media, 2006.

R. de P. Soares and A. R. Secchi. EMSO: A new environment for modelling, simulation and optimisation. In *Computer Aided Chemical Engineering*, volume 14, pages 947–952. Elsevier, 2003.

E. J. Doedel, X. J. Wang, and T. F. Fairgrieve. 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.

M. F. Doherty, Z. T. Fidkowski, M. F. Malone, and R. Taylor. *Perry's Chemical Engineers' Handbook*, chapter 13, page 33. McGraw-Hill Professional, 8th edition, 2008.

I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.

Dymola User Manual. Volume 2. Dymola 2017 FD01, Dassault Systèmes AB, 2016.

H. Elmqvist and M. Otter. Methods for tearing systems of equations in object-oriented modeling. In *Proceedings ESM'94, European Simulation Multiconference, Barcelona, Spain, June 1–3*, pages 326–332, 1994.

Hilding Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, May 1978.

Robert Fourer, David M. Gay, and Brian Wilson Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Brooks/Cole USA, 2003.

Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, USA, 3rd edition, 1996.

gPROMS. Process Systems Enterprise Limited, gPROMS. `https://www.psenterprise.com`, 2017. [Online; accessed 21-Jan-2017].

Prem K. Gupta, Arthur W. Westerberg, John E. Hendry, and Richard R. Hughes. Assigning output variables to equations using linear programming. *AIChE Journal*, 20(2):397–399, 1974.

T. E. Güttinger and M. Morari. Comments on “multiple steady states in homogeneous azeotropic distillation”. *Ind. Eng. Chem. Res.*, 35:2816–2816, 1996.

T. E. Güttinger, C. Dorn, and M. Morari. Experimental study of multiple steady states in homogeneous azeotropic distillation. *Ind. Eng. Chem. Res.*, 36:794–802, 1997.

HSL. A collection of Fortran codes for large scale scientific computation., 2017. URL `http://www.hsl.rl.ac.uk`.

IEEE 754. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 1–70, Aug 2008. doi:10.1109/IEEESTD.2008.4610935.

A. Kannan, M. R. Joshi, G. R. Reddy, and D. M. Shah. Multiple-steady-states identification in homogeneous azeotropic distillation using a process simulator. *Ind. Eng. Chem. Res.*, 44: 4386–4399, 2005.

A. Kröner, W. Marquardt, and E.D. Gilles. Getting around consistent initialization of DAE systems? *Computers & Chemical Engineering*, 21(2):145–158, 1997.

F. Magnusson and J. Åkesson. Symbolic elimination in dynamic optimization based on block-triangular ordering. *Optimization Methods and Software*, 2017. doi:10.1080/10556788.2016.1270944. Published online: 17 Jan 2017.

S. Mattsson, H. Elmqvist, and M. Otter. Physical system modeling with Modelica. *Control. Eng. Pract.*, 6:501–510, 1998.

S. E. Mattsson, M. Otter, and H. Elmqvist. Modelica hybrid modeling and efficient simulation. In *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, volume 4, pages 3502–3507, 1999.

L. M. Naphthali and D. P. Sandholm. Multicomponent separation calculations by linearization. *AIChE J.*, 17:148–153, 1971.

L. A. Ochel and B. Bachmann. Initialization of equation-based hybrid models within OpenModelica. In *5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (University of Nottingham; Nottingham, UK; April 19, 2013)*, Linköping Electronic Conference Proceedings, pages 97–103. Linköping University Electronic Press; Linköpings universitet, 2013.

Online Supplement, 2017. URL `https://github.com/baharev/failure-modes-of-tearing`.

C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988.

P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND: An object-oriented computer environment for modeling and analysis: The modeling language. *Computers & Chemical Engineering*, 15(1):53–72, 1991.

M. Sielemann and G. Schmitz. A quantitative metric for robustness of nonlinear algebraic equation solvers. *Mathematics and Computers in Simulation*, 81(12):2673–2687, 2011.

M. Sielemann, F. Casella, and M. Otter. Robustness of declarative modeling languages: Improvements via probability-one homotopy. *Simulation Modelling Practice and Theory*, 38: 38–57, 2013.

P. Täuber, L. Ochel, W. Braun, and B. Bachmann. Practical realization and adaptation of Cellier's tearing method. In *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 11–19, New York, NY, USA, 2014. ACM.

M. Tiller. *Introduction to physical modeling with Modelica*. Springer Science & Business Media, 2001.

J. Unger, A. Kröner, and W. Marquardt. Structural analysis of differential-algebraic equation systems — theory and applications. *Computers & Chemical Engineering*, 19(8):867–882, 1995.

A. Vadapalli and J. D. Seader. A generalized framework for computing bifurcation diagrams using process simulation programs. *Comput. Chem. Eng.*, 25:445–464, 2001.

R.C. Vieira and E.C. Biscaia Jr. Direct methods for consistent initialization of DAE systems. *Computers & Chemical Engineering*, 25(9–10):1299–1311, 2001.

A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

A. W. Westerberg and F. C. Edie. Computer-aided design, Part 1 Enhancing Convergence Properties by the Choice of Output Variable Assignments in the Solution of Sparse Equation Sets. *The Chemical Engineering Journal*, 2:9–16, 1971a.

A. W. Westerberg and F. C. Edie. 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.