J Glob Optim (2008) 42:327–345
DOI 10.1007/s10898-008-9317-7

A scaling algorithm for polynomial constraint satisfaction problems

Ferenc Domes · Arnold Neumaier

Received: 11 March 2008 / Accepted: 13 March 2008 / Published online: 9 July 2008

Abstract Good scaling is an essential requirement for the good behavior of many numerical algorithms. In particular, for problems involving multivariate polynomials, a change of scale in one or more variable may have drastic effects on the robustness of subsequent calculations. This paper surveys scaling algorithms for systems of polynomials from the literature, and discusses some new ones, applicable to arbitrary polynomial constraint satisfaction problems.

Keywords Scaling algorithms · Constraint satisfaction problems · Optimization

1 Introduction

Good scaling is an essential requirement for the robust behavior of methods to solve sparse linear systems, to find one or all zeros of nonlinear systems, to solve optimization problems, and to find one or all solutions of constraint satisfaction problems. While the literature is dominated by work on scaling linear systems, we got interested in scaling nonlinear systems to improve the performance of our (currently still experimental) GloptLab system [4,5] for the solution of constraint satisfaction problems.

For problems involving multivariate polynomials, a change of scale in one or more variable may have drastic effects on the robustness of subsequent calculations. Therefore, it is important to have strategies for scaling polynomial systems. A few such scaling algorithms were discussed in the literature on polynomial system solving. In [21], a scaling algorithm, minimizing the sum of squares of the exponents of the coefficients is given. In [14, Chap. 5], a slightly improved scaling algorithm is presented. (These methods are presented below in Sect. 3 and further discussed in Sect. 5.)

Unfortunately, these algorithms behave poorly on badly scaled linear problems. On the other hand, a good nonlinear scaling algorithm should of course also perform well when applied to linear problems. The linear scaling problem has received a lot of attention in the

F. Domes (✉) · A. Neumaier
Faculty of Mathematics, University of Vienna, Nordbergstrasse 15, 1090 Vienna, Austria
e-mail: ferenc.domes@univie.ac.at

literature. Important old works on linear scaling are by Curtis and Reid [3] and Parlett and Landis [16].

In [15], a new pivoting strategy for Gaussian elimination was introduced. The idea was to precondition the coefficient matrix of a linear system, so as to obtain an equivalent scaled system with a matrix which is close to being diagonally dominant and hence well-scaled. The preconditioning is based on solving a linear program via a matching algorithm. It increases the accuracy of the solution and reduces the need for partial pivoting, thereby speeding up the solution process.

Theory and algorithms of the scaling algorithm in [15] were extended and adapted for preconditioning large sparse unsymmetric linear systems by Duff and Koster [7], with important applications in semiconductor device and circuit simulations [19]. A thorough experimental study of the benefits of such scaling-based preconditioning was given in [1]. Symmetrized maximum weight matchings related to optimal scaling algorithms are used in [18] and [9] for solving highly indefinite, unsymmetric linear systems; there the reordering and scaling is used as a preconditioner for incomplete LDL^T factorizations. As one can see there is a great interest and numerous applications for preprocessing techniques based on optimal scaling and associated matching algorithms.

When leaving the linear case, new considerations have to be made since the nonlinear entries should be weighted differently than the linear ones. In the present paper, a new principle for nonlinear scaling algorithms is presented which reduces in the linear case to the linear programming approach of [15]; thus we hope to gain the same benefits for polynomial problems. The problem class treated is that of polynomial constraint satisfaction problems. This includes systems of polynomial equations; but there may be fewer or more constraints than variables. The technique extends without problems to the scaling of polynomial optimization problems. Indeed, we can always treat an objective function as an additional constraint with an artificial upper bound. A derivation of two new algorithms based on the new principle is given in Sect. 4. All algorithms discussed are applied to a number of examples in Sect. 5 and compared with each other.

The new scaling algorithms are used as part of the global optimization environment GloptLab (see [4,5]).

In the context of polynomial systems, Kim and Kojima [12] also discuss scaling techniques. However, these concern the improvement of the numerical stability of auxiliary linear systems by scaling the latter, rather than scaling the coefficients of the polynomials. There are also strategies for scaling nonlinear eigenvalue problems can be found in [8], [10] and [11]. Again, these have a different nature; they exploit the scaling freedom in reducing polynomial eigenvalue problems to larger linear ones.

2 Problem specification

2.1 Notation

We shall use the following notation. $\mathbb{N}_0$ denotes the set of natural numbers including zero, and $\mathbb{R}_+$ the set of nonnegative reals. Inequalities on vectors are interpreted componentwise. The jth row of a matrix A is denoted by $A_{j:}$, and the kth column by $A_{:k}$. $\mathbb{IR}^d$ denotes the d-dimensional space of interval vectors. An interval vector $\mathbf{x} \in \mathbb{IR}^d$ is a Cartesian product of closed real intervals, representing a (bounded or unbounded) axiparallel box in $\mathbb{R}^d$. Thus each component $\mathbf{x}_i$ of $\mathbf{x}$ is a real closed interval, with

$$\mathbf{x}_i := [\underline{x}_i, \overline{x}_i].$$

The values $-\infty$ and ∞ are allowed as lower and upper bounds, respectively, to take care of one-sided bounds on variables.

We consider constraint satisfaction problems with general constraints $C(x) \in \mathbf{F}$ and bound constraints $x \in \mathbf{x}$. The m general constraint are interpreted as componentwise enclosures $C_i(x) \in \mathbf{F}_i$ $(i = 1 \ldots m)$. This includes equality constraints if $\mathbf{F}_i = [F_i, F_i]$ is a degenerate interval, and inequality constraints if one of the bounds is infinite. $\underline{F}_i \le C_i \le \overline{F}_i$ if both bounds are finite and $C_i = \overline{F}_i$ if the bounds are equal. Similarly, the n bound constraints are interpreted as enclosures $x_j \in \mathbf{x}_j$ with $j = 1 \ldots n$. Again, fixed variables and one-sided bounds on the variables are included as special cases.

$$\mathrm{mid}(\mathbf{x}) := (\underline{x} + \overline{x})/2$$

denotes the midpoint of a box $\mathbf{x}$,

$$\langle \mathbf{x} \rangle := \min_i(|\underline{x}_i|, |\overline{x}_i|)$$

denotes the magnitude and

$$|\mathbf{x}| := \max_i(|\underline{x}_i|, |\overline{x}_i|)$$

denotes the magnitude of an interval vector $\mathbf{x}$.

In this paper, all constraints are defined by polynomial expressions in standard form as a linear combination of monomials. All monomials occurring in some general constraints are collected together in a vector-valued function $q(x) : \mathbb{R}^n \to \mathbb{R}^p$ with components

$$q(x)_k = \prod_{j=1}^{n} x_j^{E_{kj}} \quad \text{for } k = 1 \ldots p.$$

Here $E \in \mathbb{N}_0^{p \times n}$ is a sparse matrix encoding the powers with which the variables appear in the monomials used. The corresponding polynomial coefficients are collected in a sparse matrix $A \in \mathbb{R}^{m \times p}$. Thus the general polynomial constraint satisfaction problem with n variables and m constraints takes the form

$$x \in \mathbf{x}, \;\; Aq(x) \in \mathbf{F} \tag{1}$$

with $q(x)$ as above, $A \in \mathbb{R}^{m \times p}$, $\mathbf{x} \in \mathbb{IR}^n$, and $\mathbf{F} \in \mathbb{IR}^m$.

The *polynomial scaling problem* now consists in finding a constraint scaling vector $r \in \mathbb{R}_+^m$ and a variable scaling vector $c \in \mathbb{R}_+^n$ such that the scaled problem

$$x \in \mathbf{x}, \;\; A^s q(x) \in \mathbf{F}^s \quad \text{with} \quad A^s_{ik} := r_i |A_{ik}| q(c)_k, \;\; \mathbf{F}^s_i := r_i \mathbf{F}_i \tag{2}$$

is well-scaled in an appropriate sense. Which properties constitute a well-scaled problem is a somewhat ill-defined matter, because it highly depends on the applications and is not easily quantifiable. Intuitively, a scaling algorithm should somehow decrease large variations between appropriately weighted sums of logarithms of the coefficients of the matrix A; the weights should reflect the expected size of the values of the monomials.

Different intuitions about well-scaledness result in different algorithms for the construction of the scaling vectors. This also makes it difficult to compare scaling algorithms. The ultimate criterion of quality is defined by the behavior of the application that uses a scaling algorithms, and thus cannot be evaluated independently of the application. Therefore it is advisable to have different algorithmic choices that enable users to choose the one most suited to their case.

3 Known methods

This section presents the two methods found in the literature, converted to the problem format and notation given in the previous section and discusses some of their weaknesses.

3.1 Watson's HOMPACK algorithm

On page 20 of [21], an algorithm is given for scaling polynomial systems $F(x) = 0$, where

$$F_i(x) = \sum_{j=1}^{n_j} p_{ij} \prod_{k=1}^{n} x_k^{d_{ijk}},\ i = 1, \ldots, m.$$

(In [21], $m = n$ but their recipe trivially extends to the case $n \neq m$.) To find the constraint scaling vector $c = 10^e$ and the variable scaling vector $r = 10^v$ (with componentwise powers), they minimize the scaling function

$$S(e, v) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n_j} \left[e_i + \log_{10} |p_{ij}| + \sum_{k=1}^{n} v_k d_{ijk} \right]^2 \tag{3}$$

by finding the solution of $\nabla S = 0$. In our problem representation (1), this amounts to find the minimum of

$$S(e, v) = \frac{1}{2} \sum_{(i,j) \in I} \left[e_i + L_{ij} + (Ev)_j \right]^2$$

with $L_{ij} := \log_{10} |A_{ij}|$ and $I = \{(i, j) \mid A_{ij} \neq 0\}$. The index sets I_i with $n_i := |I_i|$ are introduced because of the remark in [21] that all coefficients $p_{ij} = 0$ should be omitted from the calculations. The stationarity condition $\nabla S = 0$ results in an equation similar to [21, p. 20]:

$$Gw = b \tag{4}$$

with $w = \begin{pmatrix} e \\ v \end{pmatrix}$ and

$$G_{ll} = n_l, \quad G_{l,m+s} = \sum_{(l,j) \in I} E_{js}, \quad G_{m+s,l} = \sum_{(l,j) \in I} E_{js}, \quad G_{m+s,m+t} = \sum_{(:,j) \in I} E_{jt} E_{js}$$

$$b_l = \sum_{(l,j) \in I} L_{lj}, \quad b_{n+s} = \sum_{(i,j) \in I} L_{ij} E_{js},$$

where $l = 1 \ldots m, s = 1 \ldots n, t = 1 \ldots n$. Finally the scaling vectors are found by computing $c = 10^e$ and $r = 10^v$.

Remark As shown in Sect. 5 the algorithm frequently fails when the matrix E has a regular structure: The matrix G is singular whenever for any $m < k \neq t$ the equation

$$\sum_{(:,j) \in I} E_{jk} = \sum_{(:,j) \in I} E_{jt}$$

holds for all $(i, j) \in I$. For example, in the 4-dimensional linear problem where E is the 4×4 identity matrix, the matrix

$$G = \begin{pmatrix} 4 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 4 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 4 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 4 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 & 4 & 4 & 4 & 4 \end{pmatrix}$$

is singular.

3.2 Morgan's algorithm

The book [14] presents in Chap. 5 (pp. 107–119) two scaling algorithms for polynomial systems of equations. The first algorithm is called SCLGEN and is very similar to the one presented in the previous subsection. The second algorithm, called SCLCEN, is a modified version of SCLGEN, and takes the values of the bound constraints into account.

Though Morgan uses a slightly different notation, SCLGEN is a straightforward extension of the method used in Hompack. Indeed, using the same notation as in Subsect. 3.1, the objective function

$$S(e, v) = \sum_{i=1}^{m} \sum_{j=1}^{n_j} \left[e_i + \log_{10} |p_{ij}| + \sum_{k=1}^{n} v_k d_{ijk} \right]^2 + r_2 \tag{5}$$

is nearly the same as (3), with an additional term

$$r_2 := \sum_{i=1}^{m} \sum_{1 \le j' < j'' \le n_i} \left[\left(e_i + \log_{10} |p_{ij'}| + \sum_{k=1}^{n} v_k d_{ij'k} \right) - \left(e_i + \log_{10} |p_{ij''}| + \sum_{k=1}^{n} v_k d_{ij''k} \right) \right]^2,$$

penalizing the variation in magnitude between the coefficients of F_i. The suggested solution method, solving the equation $\nabla S = 0$, is also the same. For our problem representation (1), the method amounts to finding the minimum of

$$S(e, v) = \sum_{(i,j) \in I} \left[e_i + L_{ij} + (Ev)_j \right]^2 + \sum_{\substack{(i,j'),(i,j'') \in I \\ j' < j''}} \left[L_{ij'} - L_{ij''} + (Ev)_{j'} - (Ev)_{j''} \right]^2, \tag{6}$$

with $L_{ij} := \log_{10} |A_{ij}|$. This can be written as a linear least squares system, whose solution gives the scaling vectors $c = 10^e$ and $r = 10^v$. Solving $\nabla S = 0$ just means solving the normal equations, but other, more stable methods for solving the least squares problem can be used.

Note that solving the least squares problem

$$\min \|Gx - b\|, \quad G \in \mathbb{R}^{m(p+s)\times nm}, \quad x \in \mathbb{R}^{nm}, \quad b \in \mathbb{R}^{m(p+s)}, \quad s := \binom{p}{2} \tag{7}$$

could prove difficult because of the dimension of G, if the number p of the monomials is high. However this problem is reduced by the following factors:

- The matrix G is usually sparse. The dimension given in (7) is the worst case, attained only when $G_{ij} \neq 0$ for all i and j.
- In the normal equations $Cx = G^T b$, the matrix $C = G^T G$ is only $nm \times nm$.

Morgan's second scaling algorithm SCLCEN is a modified version of SCLGEN, which uses the upper bound or the mean values of the box $\mathbf{x}$ to define the variable scaling. Let us denote the selected bound information as $\hat{x}$ then SCLCEN defines

$$v_k := \log_{10} |\hat{x}| \quad \text{for } k = 1, \ldots n,$$

(actually, Morgan does not use the absolute value, which unduly restricts the applicability of his method to variables which can take positive values) and replaces the second term r_2 of (5) with

$$r_2 := \sum_{1 \le i' \le i'' \le m} \sum_{\substack{1 \le j' \le n_{i'} \\ 1 \le j'' \le n_{i''} \\ j' \le j''}} \left[e_{i'} - e_{i''} + \log_{10} \frac{|p_{i'j'}|}{|p_{i''j''}|} + \sum_{k=1}^{n} v_k (d_{i'j'k} - d_{i''j''k}) \right]^2 .$$

The new equation now depends only on the variable vector e since v is constant. Written in our notation, the complete objective function takes the form

$$\begin{aligned} S(e) := & \sum_{(i,j) \in I} \left[e_i + L_{ij} + (Ev)_j \right]^2 \\ & + \sum_{\substack{(i',j'),(i',j'') \in I \\ i' \le i'', j' \le j''}} \left[e_{i'} - e_{i''} + L_{i'j'} - L_{i''j''} + (Ev)_{j'} - (Ev)_{j''} \right]^2 . \end{aligned}$$

The least squares problem for minimizing $S(e)$ now involves fewer variables (only e) but many more terms. However this method only computes the scaling factors for the constraints. The variable scaling is determined by the mean value of the bound constraints. This could prove bad if some of the variables are unbounded or the bounding intervals are very wide. The second problem is that $v_k := \log_{10} \hat{x}$ is not defined if $\hat{x} = 0$, thus the algorithm does not work for the very common constraint $x_i \ge 0$. Because of these problems, SCLGEN is not tested in this paper.

4 The new methods

We now discuss new methods which give a good scaling for both linear and nonlinear systems.

We search for a constraint scaling vector $r \in \mathbb{R}^m_+$ and a variable scaling vector $c \in \mathbb{R}^n_+$ such that the scaled matrix should have all entries bounded in absolute value by 1, with at least one 1 in each row not consisting of zeros only. In particular, the inequality

$$r_i |A_{ik}| q(c)_k \le 1 \tag{8}$$

holds and

$$r_i = \frac{1}{\max_k |A_{ik}| q(c)_k}$$

if the denominator is nonzero. Since scaling is intended to fix magnitudes only, we can relax this requirement by allowing the largest entries to be $O(1)$ instead of 1 and to choose the scaling factors as powers of a fixed, but arbitrary integer $b > 1$. To minimize rounding errors, b should be a power of the basis in which the arithmetic is executed. In terms of this basis, we define

$$u_i := \log_b r_i, \quad v_j := \log_b c_j$$

and

$$B_{ik} := -\log_b |A_{ik}|,$$

and substitute them into (8) we obtain

$$b^{u_i} b^{-B_{ik}} \prod_{j=1}^{n} b^{v_j E_{kj}} \leq b^0,$$

resulting in the inequality

$$u_i + \sum_{j=1}^{n} v_j E_{kj} = u_i + (Ev)_k \leq B_{ik}. \tag{9}$$

We define the index set $I := \{(i,k) \mid B_{ik} \text{ is finite}\}$ then by (9)

$$\sum_{(i,k)\in I} (u_i + (Ev)_k) \leq \sum_{(i,k)\in I} B_{ik} \leq \infty \tag{10}$$

holds. Let d_i be the number of $(i,:) \in I$ and q_k the number of $(:,k) \in I$, then by transforming the left hand side of (10) we obtain

$$\sum_{i=1}^{m} d_i u_i + \sum_{k=1}^{p} q_k (Ev)_k = d^T u + q^T E v < \infty. \tag{11}$$

If (8) holds, we may regard the expression

$$f(r,c) := d^T u + q^T E v = d^T \log_b r + q^T E \log_b c \tag{12}$$

as a measure of the quality of the scaling, in the sense that different scalings may be considered equivalent if the values of $f(r,c)$ are the same, and better scalings have higher values of $f(r,c)$. Therefore, we may take (12) as the objective function of an optimization problem with the conditions (9) (equivalent to (8)) as the constraints. This gives the linear program

$$\begin{array}{ll} \max & d^T u + q^T E v \\ \text{s.t.} & u_i + (Ev)_k \leq B_{ik} \quad \text{for all } i,k, \end{array} \tag{13}$$

with an obviously finite maximum.

To obtain a reasonable bounding box $\mathbf{v}$ and a good starting value v^0 for the variable scaling factors in our algorithms, we define the following constants. (Here the function 'round' rounds

to the nearest integers as in MATLAB, i.e., rounding in case of ambiguity to the absolutely larger integer.)

$$
\begin{array}{ll}
b & \text{base, } b > 0 \\
\mathbf{eps} & \text{machine precision} \\
\varepsilon_i := \mathbf{eps} * \min(1, 1/\max(\langle \mathbf{x} \rangle), \min(|\mathbf{x}|)) & \text{minimal scaling factor} \\
u_j := \max(\langle \mathbf{x}_j \rangle, \varepsilon) & \text{additive scaling factor for } x_j \\
r_j := 1/\min(|\mathbf{x}_j|, \varepsilon^{-1}) & \text{multiplicative scaling factor for } x_j \\
\mathbf{v}_j := [\text{round}(\log_b u_j), \text{round}(-\log_b r_j)] & v_j \text{ range interval} \\
v_j^0 := \text{round}(\overline{v}_j - \underline{v}_j) & \text{initial value for } r_j .
\end{array} \tag{14}
$$

4.1 A linear programming algorithm

Adding the bounds on the vector v defined in (14) to the linear optimization problem (13) gives the linear program

$$
\begin{array}{ll}
\max & f(u, v) := d^T u + q^T E v \\
\text{s.t.} & u_i + (Ev)_k \le B_{ik} \quad \text{for all } i, k, \\
& v \in \mathbf{v}.
\end{array} \tag{15}
$$

The linear optimization problem (15) can be solved using a linear programming package such as lpSolve [2]. The linear program has $m+n$ variables and $k+q$ inequality constraints, where k is the number of finite, non-zero entries of the matrix B (maximum mp) and q is the number of finite bounds in $\mathbf{v}$. Solving the linear program successfully results in a scaling vector $y = (u, v)^T$. The variable scaling vector c can be obtained by computing $c_j = b^{v_j}$ for $j = 1 \ldots n$ and the constraint scaling vector r from y by computing $r_i = b^{u_i}$ for $i = 1 \ldots m$.

The optimization problem (15) may have an infinite number of maximizer. In this case it is desirable that that the components of the maximizer are reasonably small. To achieve this, we introduce an additional postprocessing step, which solves another linear program. Let $\hat{y} = (\hat{u}, \hat{v})$ the solution of the the linear program (15). The postprocessing linear program

$$
\begin{array}{ll}
\min & \hat{y}^T y \\
\text{s.t.} & u_i + (Ev)_k \le B_{ik} \quad \text{for all } i, k, \\
& f(u, v) \ge f(\hat{u}, \hat{v}) \\
& y \in \mathbf{y},
\end{array} \tag{16}
$$

where $y = (u, v)$ and

$$
\mathbf{y}_i = \begin{cases} [0, 0] & \text{if } \hat{y}_i = 0 \\ [-\infty, 0] & \text{if } \hat{y}_i < 0 \\ [0, \infty] & \text{if } \hat{y}_i > 0. \end{cases}
$$

yields a solution of (15) whose entries are usually small enough to be acceptable.

4.2 An iterative algorithm

If the number of nonzero entries of A is large or if we have to scale a large number of matrices, an iterative scaling method for approximately solving the linear optimization problem (13) may be preferable.

Since it follows from (9) that $u \le u(v)$, where

$$
u_i(v) := \min_k C_{ik} \quad \text{with} \quad C_{ik} := B_{ik} - (Ev)_k,
$$

the objective in (13) is maximal for $u = u(v)$. Therefore to solve (13), we have to maximize the *scaling function*

$$f(v) = d^T u(v) + q^T E v.$$

If we change the jth component of v by setting

$$\hat{v} := v + \delta I_{:j}, \tag{17}$$

the change of the scaling function is given by

$$\Delta := f(\hat{v}) - f(v) = d^T(u(\hat{v}) - u(v)) + \delta q^T E_{:j}. \tag{18}$$

with

$$u(\hat{v}) := u(v + \delta I_{:j}) = \min_k (C_{ik} - \delta E_{kj}). \tag{19}$$

Algorithm 4.1 First we initialize the n-dimensional vectors u, r, v^0 and $\mathbf{v}$ as in (14). We set $\delta := -1$, $v := v^0$ and compute

$$e := q^T E,\ C_{ik} := B_{ik} - (Ev)_k,\ m_i := \min_k C_{ik}.$$

We iterate through the components of the vector v, while variable t counts the number of unchanged components of v since the last change on v has been made. For the component v_j, $j \in \{1, \ldots, n\}$ we proceed as follows:

1. If $t > n$ terminate the iteration procedure.
2. If $v_j + \delta \in \mathbf{v}_j$
   (a) We set $\hat{v} := v + \delta I_{:j}$.
   (b) We compute $\hat{C}_{ik} := C_{ik} - \delta E_{kj}$ and $\hat{m}_i = \min_k \hat{C}_{ik}$ for $i = 1 \ldots n$ and $k = 1 \ldots p$. By (18) and (19) we have $\Delta = d^T(\hat{m} - m) + \delta e_j$.
   (c) If $\Delta > 0$ we accept $\hat{v}$ and set $v = \hat{v}$, $C = \hat{C}$, $m = \hat{m}$ and $t = 0$, then begin Step 2. anew for the same component of v.
   (d) If $\Delta \leq 0$ we set $\delta = -\delta$ and
      (i) if $\delta \geq 0$ we begin the process again for the same jth component of v.
      (ii) if $\delta < 0$ we set $t = t + 1$ and step to the $(j + 1)$th component of v.
3. If $\delta < 0$ we change the sign of δ and begin the process again for the same component of v.
4. If $\delta \geq 0$ we change the sign of δ, increase t by one, and continue with the first step, using the $(j + 1)$th component of v.

After the termination of the above iteration we have a vector v from which the variable scaling vector c can be obtained by computing

$$c_j = b^{v_j}, \quad j = 1 \ldots n,$$

and the constraint scaling vector r can be obtained by computing

$$r_i = b^{u(v)_i}, \quad i = 1 \ldots m.$$

Theorem 4.2 *Algorithm 4.1 ends after finitely many steps.*

Proof Since we accept a new v only if $\Delta = f(\hat{v}) - f(v) > 0$, implying that $f(\hat{v}) > f(v)$, the scaling function either increases, or it does not change in case we do not accept the new $v = \hat{v}$.

Since $v \in \mathbf{v}$ is bounded, $\hat{v}_j = v_j \pm \delta$ and the constant δ does not change, v can only attain a finite number of values. The termination criterion $t > n$ in the algorithm guarantees that if the value $f(v)$ of the scaling function not change in the last n steps the algorithm terminates.

Putting all this together shows that $f(v)$ increases monotonically, stays constant at most n times in a row, and takes only finitely many distinct values. Therefore the algorithm carries out a finite number of steps, and there is no possibility of cycling. This shows that the scaling algorithm is finite. □

Using the algorithm results in a scaled version A^s of the matrix A with

$$A^s_{ik} := r_i |A_{ik}| q(c)_k.$$

Since (u, v) is an approximate solution of the optimization problem (13) with integral entries, we have $A^s_{ik} \leq 1$ for $i = 1 \ldots n$ and $k = 1 \ldots m$, and, since $u = u(v)$, for each i there is at least one index k such that $A^s_{ik} > b^{-1}$. Thus the matrix will be well-scaled if b is not large.

In practice, it is reasonable to remove the variables x_k where $\mathbf{x}_k$ is narrow before the computation of the scaling. We proceed as follows. Let K be the list of indices k with $\underline{v}_k = \overline{v}_k$ (which defines the meaning of $\mathbf{x}_k$ being narrow) in increasing order. For indices in K, v_k is determined by the bounds, and can therefore be eliminated from the problem. Thus the problem (1) changes into

$$\hat{x} \in \hat{\mathbf{x}}, \quad \hat{A} q(\hat{x}) \in \mathbf{F}, \tag{20}$$

with $\widehat{x} = (x_j) j \notin K$, $\hat{A}_{ij} = (\mathbf{x}_k^{E_{kj}} A_{ij})$. After that we remove all *constant monomials* (x_k^0 for all k) from $q(\widehat{x})$ by bringing them to the left hand side by substituting them from F obtaining $\hat{F}$ and changing q to $\hat{q}$. Finally we remove the constraints where all the coefficients are zero, thereby possibly changing the dimension of $\hat{A}$ and $\hat{F}$. Doing the above reduces the computation time. Let v^0 be the starting value vector for v as in (14) and let $\hat{v}$ be the solution vector computed by scaling the reduced system (20). The the solution vector of the original system can be written as

$$v_k = \begin{cases} v^0_{i(k)} & \text{for } k \in K, \\ \hat{v}_{j(k)} & \text{for } k \notin K, \end{cases}$$

where $i(k)$ denotes the position of k in the list K, $j(k)$ denotes the position of k in the increasingly sorted list complementary to K. Then the variable scaling vector is $c = b^v$ and the constraint scaling vector is $r = b^{u(v)}$.

5 Numerical examples

We illustrate the preceding algorithms with as number of examples. In the following, we refer to the algorithms presented in Subsects. 3.1, 3.2, 4.1, and 4.2 by the names HOMPACK, MORGAN, SCALELP, and SCALEIT, respectively.

The examples are chosen to illustrate different features; first a badly scaled nonlinear system from the literature, followed by a reduced version of the same system; then a badly scaled linear system from the literature; finally a randomly generated quadratic system and a nearly degenerate problem are presented. The basis $b = 10$ is used in all calculations.

For the purpose of better understanding the examples below, instead a matrix A, we define the *exponent matrix* of the problem (1) by

$$\mathrm{ex}(A) := \begin{pmatrix} q(x)^T \\ \log_{10}(A) \end{pmatrix},$$

and will usually give $\mathrm{ex}(A)$ in place of A. The entries of $\mathrm{ex}(A)$ with value $-\infty$ are indicated by an $*$.

5.1 A badly scaled system from the literature

A real word example from the paper [13] a chemical combustion system is presented in [20]. In [20] it is advised to apply a scaling algorithm to

$$\begin{aligned}
&x_2 + 2x_6 + x_9 + 2x_{10} - 10^{-5} = 0,\\
&x_3 + x_8 - 3.0 \times 10^{-5} = 0,\\
&x_1 + x_3 + 2x_5 + 2x_8 + x_9 + x_{10} - 5.0 \times 10^{-5} = 0,\\
&x_4 + 2x_7 - 10^{-5} = 0,\\
&0.5140437 \times 10^{-7} x_5 - x_1^2 = 0,\\
&0.1006932 \times 10^{-6} x_6 - x_2^2 = 0,\\
&0.7816278 \times 10^{-15} x_7 - x_4^2 = 0,\\
&0.1496236 \times 10^{-6} x_8 - x_1 x_3 = 0,\\
&0.6194411 \times 10^{-7} x_9 - x_1 x_2 = 0,\\
&0.2089296 \times 10^{-14} x_{10} - x_1 x_2^2 = 0
\end{aligned} \tag{21}$$

before solving it. Writing (21) in the form (1) results in the exponent matrix

$$\begin{pmatrix}
x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2\\
* & 0 & * & * & * & 0.3 & * & * & 0 & 0.3 & -5 & * & * & * & * & * & *\\
* & * & 0 & * & * & * & * & 0 & * & * & -4.5 & * & * & * & * & * & *\\
0 & * & 0 & * & 0.3 & * & * & 0.3 & 0 & 0 & -4.3 & * & * & * & * & * & *\\
* & * & * & 0 & * & * & 0.3 & * & * & * & -5 & * & * & * & * & * & *\\
* & * & * & * & -7.3 & * & * & * & * & * & * & 0 & * & * & * & * & *\\
* & * & * & * & * & -7 & * & * & * & * & * & * & 0 & * & * & * & *\\
* & * & * & * & * & * & -15 & * & * & * & * & * & * & 0 & * & * & *\\
* & * & * & * & * & * & * & -6.8 & * & * & * & * & * & * & 0 & * & *\\
* & * & * & * & * & * & * & * & -7.2 & * & * & * & * & * & * & 0 & *\\
* & * & * & * & * & * & * & * & * & -15 & * & * & * & * & * & * & 0
\end{pmatrix}$$

without bounds on the variables and with $\mathbf{F}_i = [0, 0]$ for all $i = 1, \ldots, m$.

For this example, Meintjes and Morgan [13] suggest to scale using HOMPACK, and indeed both HOMPACK and MORGAN perform well. Our methods give similar results and the entries are especially small in the quadratic and in the bilinear part of the scaled exponent matrix, a property we would except after the application of a good scaling. We show the results for HOMPACK and SCALEIT:

The scaling problem solved with the method HOMPACK using the gradient equation (Subsect. 3.1) results in the scaling factors

$$\begin{aligned}
\log_a(x_s)^T &:= (-4.6 \;\; -1.5 \;\; 6.9 \;\; -13 \;\; 1.8 \;\; -3.9 \;\; 10 \;\; -9.1 \;\; -1.1 \;\; 5.5),\\
\log_a(c_s)^T &:= (1.1 \;\; 2.3 \;\; 0.62 \;\; 2.3 \;\; 7.3 \;\; 7 \;\; 15 \;\; 6.8 \;\; 7.2 \;\; 5.1e-),
\end{aligned}$$

and the scaled exponent matrix

$$\begin{pmatrix}
x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2 \\
* & -0.43 & * & * & * & -2.5 & * & * & -0.018 & 6.9 & -3.9 & * & * & * & * & * & * \\
* & * & 9.1 & * & * & * & * & -6.9 & * & * & -2.3 & * & * & * & * & * & * \\
-3.9 & * & 7.5 & * & 2.8 & * & * & -8.2 & -0.5 & 6.1 & -3.7 & * & * & * & * & * & * \\
* & * & * & -10 & * & * & 13 & * & * & * & -2.7 & * & * & * & * & * & * \\
* & * & * & * & 1.8 & * & * & * & * & * & * & -1.8 & * & * & * & * & * \\
* & * & * & * & * & -3.9 & * & * & * & * & * & * & 3.9 & * & * & * & * \\
* & * & * & * & * & * & 10 & * & * & * & * & * & * & -10 & * & * & * \\
* & * & * & * & * & * & * & -9.1 & * & * & * & * & * & * & 9.1 & * & * \\
* & * & * & * & * & * & * & * & -1.1 & * & * & * & * & * & * & 1.1 & * \\
* & * & * & * & * & * & * & * & * & -9.2 & * & * & * & * & * & * & 9.2
\end{pmatrix}.$$

The scaling problem solved with the method ScaleIt (Subsect. 4.2) results in the scaling factors

$$\begin{aligned}
\log_a(x_s)^T &:= (-4 \quad -3 \quad -3 \quad -7 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0), \\
\log_a(c_s)^T &:= (-0.3 \quad 0 \quad -0.3 \quad -0.3 \quad 7.3 \quad 6 \quad 14 \quad 6.8 \quad 7 \quad 10),
\end{aligned}$$

and the scaled exponent matrix

$$\begin{pmatrix}
x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2 \\
* & -3.3 & * & * & * & 0 & * & * & -0.3 & 0 & -5.3 & * & * & * & * & * & * \\
* & * & -3 & * & * & * & * & 0 & * & * & -4.5 & * & * & * & * & * & * \\
-4.3 & * & -3.3 & * & 0 & * & * & 0 & -0.3 & -0.3 & -4.6 & * & * & * & * & * & * \\
* & * & * & -7.3 & * & * & 0 & * & * & * & -5.3 & * & * & * & * & * & * \\
* & * & * & * & 0 & * & * & * & * & * & * & -0.71 & * & * & * & * & * \\
* & * & * & * & * & -1 & * & * & * & * & * & * & 0 & * & * & * & * \\
* & * & * & * & * & * & -1.1 & * & * & * & * & * & * & 0 & * & * & * \\
* & * & * & * & * & * & * & 0 & * & * & * & * & * & * & -0.18 & * & * \\
* & * & * & * & * & * & * & * & -0.21 & * & * & * & * & * & * & 0 & * \\
* & * & * & * & * & * & * & * & * & -4.7 & * & * & * & * & * & * & 0
\end{pmatrix}.$$

5.2 Reduced version of the same system

By removing the obvious slack variables from the system (21) and setting $b = 10$ we obtain the simplified example

$$\begin{aligned}
&x_2 + 0.2013864 \times 10^6 x_2^2 + 0.6194411 \times 10^7 x_1x_2 + 0.4178592 \times 10^{14} x_1x_2^2 - 10^{-5} = 0, \\
&x_3 + 0.1496236 \times 10^6 x_1x_3 - 3.0 \times 10^{-5} = 0, \\
&x_1 + x_3 + 0.1028087 \times 10^8 x_1^2 + 0.2992472 \times 10^6 x_1x_3 \\
&\quad + 0.6194411 \times 10^7 x_1x_2 + 0.2089296 \times 10^{14} x_1x_2^2 - 5.0 \times 10^{-5} = 0, \\
&x_4 + 0.1238882 \times 10^8 x_1x_2 - 10^{-5} = 0.
\end{aligned} \tag{22}$$

In this case the number of constraints is reduced to 5 and the number of variables to 4. Writing (21) in the form (1) again results in the exponent matrix

$$\begin{pmatrix}
x_1 & x_2 & x_3 & x_4 & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2 \\
* & 0 & * & * & -5 & * & 7.3 & * & * & 7.21 & 15 \\
* & * & 0 & * & -4.52 & * & * & * & 6.82 & * & * \\
0 & * & 0 & * & -4.3 & 7.59 & * & * & 7.13 & 7.21 & 14.7 \\
* & * & * & 0 & -5 & * & * & 15.4 & * & * & *
\end{pmatrix}.$$

The scaling problem solved with the method HOMPACK using the gradient equation (Subsect. 3.1) results in the scaling factors

$$\log_a(x_s)^T := (0.49 \quad 1.12 \quad 0.33 \quad -3.47),$$
$$\log_a(c_s)^T := (-6.12 \quad -1.15 \quad -5.38 \quad -5.79e-016),$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2 \\ * & -5 & * & * & -11.1 & * & 3.42 & * & * & 2.7 & 10 \\ * & * & -0.821 & * & -5.67 & * & * & * & 6.49 & * & * \\ -4.89 & * & -5.05 & * & -9.68 & 3.19 & * & * & 2.56 & 3.43 & 10.4 \\ * & * & * & -3.47 & -5 & * & * & 8.47 & * & * & * \end{pmatrix}.$$

The scaling problem solved with the method MORGAN (Subsect. 3.2) results in the scaling factors

$$\log_a(x_s)^T := (-6.48 \quad -6.05 \quad -6.57 \quad -10.2)$$
$$\log_a(c_s)^T := (5.16 \quad 5.77 \quad 5.56 \quad 6.73),$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2 \\ * & -0.887 & * & * & 0.162 & * & 0.362 & * & * & -0.156 & 0.52 \\ * & * & -0.799 & * & 1.25 & * & * & * & -0.451 & * & * \\ -0.92 & * & -1.02 & * & 1.26 & 0.193 & * & * & -0.367 & 0.239 & 0.614 \\ * & * & * & -3.47 & 1.73 & * & * & 1.73 & * & * & * \end{pmatrix}.$$

The scaling problem solved with the method SCALELP (Subsect. 4.1) results in the scaling factors

$$\log_a(x_s)^T := (-7 \quad -7 \quad -7 \quad -15)$$
$$\log_a(c_s)^T := (6.02 \quad 7 \quad 6.32 \quad 14.6),$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2 \\ * & -0.981 & * & * & 1.02 & * & -0.683 & * & * & -0.773 & 0 \\ * & * & 0 & * & 2.48 & * & * & * & -0.175 & * & * \\ -0.68 & * & -0.68 & * & 2.02 & -0.09 & * & * & -0.554 & -0.472 & 0 \\ * & * & * & -0.408 & 9.59 & * & * & 0 & * & * & * \end{pmatrix}.$$

The scaling problem solved with the method SCALEIT (Subsect. 4.2) results in the scaling factors

$$\log_a(x_s)^T := (-1 \; 0 \quad -4 \quad -15)$$
$$\log_a(c_s)^T := (-7.3 \quad -1.82 \quad -6.21 \; 14.6),$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & 1 & x_1^2 & x_2^2 & x_4^2 & x_1x_3 & x_1x_2 & x_1x_3^2 \\ * & -7.3 & * & * & -12.3 & * & 0 & * & * & -1.09 & -1.32 \\ * & * & -5.82 & * & -6.35 & * & * & * & 0 & * & * \\ -7.21 & * & -10.2 & * & -10.5 & -0.618 & * & * & -4.08 & 0 & -0.528 \\ * & * & * & -0.408 & 9.59 & * & * & 0 & * & * & * \end{pmatrix}.$$

All methods seem to produce reasonable results. However, the scaled exponent matrix produced by HOMPACK still contains big entries in the quadratic and bilinear part, indicating a rather non-optimal scaling.

5.3 A linear example

The next example is a 4-dimensional linear system $Ax = b$, with matrix

$$A := \begin{pmatrix} 1 & a^4 & a^2 & 1 \\ a^4 & a^4 & 1 & a^8 \\ a^2 & 1 & a^8 & a^{10} \\ 1 & a^8 & a^{10} & 1 \end{pmatrix}, \quad b := \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

The case $a = 10^5$ of this matrix was presented in [17] as an example of a difficult to scale matrix. It was also used in [15] to demonstrate the excellent behavior of their LP-based linear scaling algorithm. The optimal scaling vectors

$$c^s = (1, \quad a^{-3}, \quad a^{-4}, \quad a^{-5}) \quad \text{and} \quad x^s = (a^{-1}, \quad a^{-4}, \quad a^{-5}, \quad a^{-6})^T \tag{23}$$

found in [15] (for $a = 10^5$) result in the permuted and scaled matrix

$$A^s := \begin{pmatrix} 1 & a^{-3} & a^{-8} & a^{-1} \\ a^{-1} & 1 & a^{-3} & a^{-6} \\ a^{-6} & a^{-1} & 1 & a^{-11} \\ a^{-3} & a^{-8} & a^{-1} & 1 \end{pmatrix}.$$

The scaling problem solved with the HOMPACK does not produce any useful results since the matrix G is singular (see Subsect. 3.1). If solved with the normal equation and rounding errors the method results in the same scaling factors as the MORGAN algorithm discussed below.

The scaling problem solved with the method MORGAN (Subsect. 3.2) results in the scaling factors

$$\begin{aligned} \log_a(x_s)^T &:= (7.31 \quad 5.06 \quad 4.06 \quad 4.31) \\ \log_a(c_s)^T &:= (-7.19 \quad -9.19 \quad -10.2 \quad -9.69), \end{aligned}$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ 1.13 & 1.88 & -1.12 & -1.87 \\ 2.13 & -0.125 & -5.12 & 3.13 \\ -0.875 & -5.12 & 1.88 & 4.13 \\ -2.37 & 3.38 & 4.38 & -5.37 \end{pmatrix}.$$

The scaling problem solved with the method SCALELP (Subsect. 4.1) results in the scaling factors

$$\begin{aligned} \log_a(x_s)^T &:= (-4 \quad -6 \quad -8 \quad -10) \\ \log_a(c_s)^T &:= (2 \quad 0 \quad 0 \quad -2), \end{aligned}$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ -1 & 0 & -4 & -7 \\ 0 & -2 & -8 & -2 \\ -2 & -6 & 0 & 0 \\ -6 & 0 & 0 & -12 \end{pmatrix}.$$

The scaling problem solved with the method ScaleIt (Subsect. 4.2) results in the scaling factors

$$\begin{aligned} \log_a(x_s)^T &:= (3 \quad 0 \quad 0 \quad -1) \\ \log_a(c_s)^T &:= (-4 \quad -7 \quad -9 \quad -10), \end{aligned}$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ 0 & 0 & -2 & -4 \\ 0 & -3 & -7 & 0 \\ -4 & -9 & -1 & 0 \\ -7 & -2 & 0 & -11 \end{pmatrix}.$$

To verify which of the scalings are equivalent we can compute the scaling measure defined by (12). The scaling vectors (23) from [15] and the scaling vectors derived by the methods ScaleLP and ScaleIt all have the value of -122. This indicates that for this problem the different methods lead to equivalent scalings.

Incidentally, this shows that, contrary to the statement made in [15, p. 13], A need not have a unique best scaling when the dominant transversal is unique.

5.4 A nonlinear example

This example is a 3-dimensional quadratic problem, with

$$\mathrm{ex}(A) := \begin{pmatrix} x_1 & x_2 & x_3 & x_1^2 & x_1x_2 & x_1x_3 & x_1x_2 & x_2^2 & x_2x_3 & x_1x_3 & x_2x_3 & x_3^2 \\ 8 & 4 & 2 & 3 & 8 & 0 & 8 & 6 & 2 & 5 & 1 & 3 \\ 0 & 5 & 6 & 3 & 0 & 6 & 5 & 0 & 2 & 9 & 7 & 7 \\ 7 & 6 & 2 & 4 & 9 & 7 & 4 & 5 & 8 & 3 & 3 & 1 \\ 1 & 5 & 7 & 1 & 6 & 7 & 4 & 2 & 4 & 6 & 7 & 1 \end{pmatrix}.$$

Because of the symmetrical nature of the powers of the monomials, the method Hompack (Subsect. 3.1) cannot be applied since the matrix

$$G := \begin{pmatrix} 12 & 0 & 0 & 0 & 7 & 7 & 7 \\ 0 & 12 & 0 & 0 & 7 & 7 & 7 \\ 0 & 0 & 12 & 0 & 7 & 7 & 7 \\ 0 & 0 & 0 & 12 & 7 & 7 & 7 \\ 7 & 7 & 7 & 7 & 196 & 196 & 196 \\ 7 & 7 & 7 & 7 & 196 & 196 & 196 \\ 7 & 7 & 7 & 7 & 196 & 196 & 196 \end{pmatrix}$$

from (4) is singular.

The scaling problem solved with the method MORGAN (Subsect. 3.2) results in the scaling factors

$$\log_a(x_s)^T := (-0.0397 \quad -0.00397 \quad 0.21),$$
$$\log_a(c_s)^T := (-4.26 \quad -4.26 \quad -5.01 \quad -4.35),$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_1^2 & x_1x_2 & x_1x_3 & x_1x_2 & x_2^2 & x_2x_3 & x_1x_3 & x_2x_3 & x_3^2 \\ 3.7 & -0.268 & -2.05 & -1.34 & 3.69 & -4.09 & 3.69 & 1.73 & -2.06 & 0.907 & -3.06 & -0.843 \\ -4.3 & 0.732 & 1.95 & -1.34 & -4.31 & 1.91 & 0.692 & -4.27 & -2.06 & 4.91 & 2.94 & 3.16 \\ 1.95 & 0.982 & -2.8 & -1.09 & 3.94 & 2.16 & -1.06 & -0.0218 & 3.19 & -1.84 & -1.81 & -3.59 \\ -3.39 & 0.649 & 2.86 & -3.43 & 1.61 & 2.82 & -0.391 & -2.36 & -0.141 & 1.82 & 2.86 & -2.93 \end{pmatrix}.$$

The scaling problem solved with the method SCALELP (Subsect. 4.1) results in the scaling factors

$$\log_a(x_s)^T := (-2 \quad 0 \quad -1)$$
$$\log_a(c_s)^T := (-6 \quad -6 \quad -7 \quad -6),$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_1^2 & x_1x_2 & x_1x_3 & x_1x_2 & x_2^2 & x_2x_3 & x_1x_3 & x_2x_3 & x_3^2 \\ 0 & -2 & -5 & -7 & 0 & -9 & 0 & 0 & -5 & -4 & -6 & -5 \\ -8 & -1 & -1 & -7 & -8 & -3 & -3 & -6 & -5 & 0 & 0 & -1 \\ -2 & -1 & -6 & -7 & 0 & -3 & -5 & -2 & 0 & -7 & -5 & -8 \\ -7 & -1 & 0 & -9 & -2 & -2 & -4 & -4 & -3 & -3 & 0 & -7 \end{pmatrix}.$$

The scaling problem solved with the method SCALEIT (Subsect. 4.2) results in the scaling factors

$$\log_a(x_s)^T := (-1 \quad 0 \quad 0)$$
$$\log_a(c_s)^T := (-7 \quad -8 \quad -8 \quad -7),$$

and the scaled exponent matrix

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_1^2 & x_1x_2 & x_1x_3 & x_1x_2 & x_2^2 & x_2x_3 & x_1x_3 & x_2x_3 & x_3^2 \\ 0 & -3 & -5 & -6 & 0 & -8 & 0 & -1 & -5 & -3 & -6 & -4 \\ -9 & -3 & -2 & -7 & -9 & -3 & -4 & -8 & -6 & 0 & -1 & -1 \\ -2 & -2 & -6 & -6 & 0 & -2 & -5 & -3 & 0 & -6 & -5 & -7 \\ -7 & -2 & 0 & -8 & -2 & -1 & -4 & -5 & -3 & -2 & 0 & -6 \end{pmatrix}.$$

If we have additional bound constraints

$$\mathbf{x} := \begin{pmatrix} [1, 2] \\ [2, 3] \\ [7, 10] \end{pmatrix}$$

the first two methods result in the same scaling vectors but the SCALELP and the method SCALEIT scaling vectors change to:

$$\log_a x^s := \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$$
$$\log_a c^s := \begin{pmatrix} -8 & -10 & -9 & -8 \end{pmatrix}.$$

This is good for optimization problems because the tight bound constraints give us an explicit knowledge about the variable scaling.

5.5 A nearly degenerate problem

Our final problem consists of a single constraint

$$4x_1^2 + 4Nx_1x_2 + 12x_1x_3 - 28x_1x_4 + (1+N^2)x_2^2 + (6N-2)x_2x_3 \\ -(10N+14)x_2x_4 + 11x_3^2 - 32x_3x_4 + 75x_4^2 + 2x_2 + 2Nx_3 + 26 \le 0.$$

We discuss the case $N = 5 \times 10^6$. The HOMPACK scaling algorithm does not yield any scaling factors because to numerical instabilities, but the MORGAN algorithm yields seemingly acceptable scaling factors

$$\log_a(x_s)^T := (-0.232 \ -5.92 \ -1.17 \ -0.846)$$
$$\log_a(c_s)^T := -0.109.$$

(Of course, neither of these algorithms were originally designed for non-square problems.)

Using the SCALELP method, we obtain good scaling factors

$$\log_a(x_s)^T := (6 \ \ 0 \ \ 6 \ \ 5)$$
$$\log_a(c_s)^T := -13.3979.$$

It turns out that the scaling makes an essential difference in a global optimization technique for box reduction described in [6]. This technique is based on the verification that a quadratic constraint is strictly convex, and a subsequent enclosure of the ellipsoid describing the constraint by a box.

In the present example, the convexity verification phase fails due to the poor condition of the Hessian matrix of the constraint ($\text{cond}_2 A = 10^{21}$). Note that the constraint is equivalent to

$$(2x_1 + Nx_2 + 3x_3 - 7x_4)^2 + (x_2 - x_3 - 5x_4 + 1)^2 \\ +(x_3 + N + 1)^2 + (x_4 + 5)^2 \le (N-1)^2$$

which is manifestly convex.

If we scale the variables according to the results of SCALELP before applying the box reduction techniques, strict convexity is verified, and we find the bound constraints

$$x \in ([-146, 1442], [-510^8, 510^7], [-310^{-5}, 10^{-12}], [-10^{-2}, 10^{-3}])^T$$

for the originally unconstrained problem. (On the other hand, when scaling with the scaling factors found by the MORGAN algorithm, the convexity test still fails. This happens because the linear part of the matrix scaled by using the MORGAN method still contains big entries.)

5.6 Discussion

The numerical examples above show that the older scaling algorithms designed for scaling polynomial systems may perform poorly when applied to linear systems or to problems with fewer constraints than variables. Due to some structural problems the method HOMPACK using the gradient equation suggested in [21] even fails to obtain a scaling vector. For the linear example the results of the method MORGAN can not be trusted, due to the numerical instabilities reported by MATLAB. Both of our algorithms produce results which are equivalent to the optimal scaling found in [15]. For nonlinear examples the two other algorithms produce similar results, but both ignore the bound constraints. The alternative method of MORGAN algorithm uses bound information but in a too naive way. Our methods are optimal

for the linear case. For the nonlinear case each algorithm suits, giving different scalings which could suit different applications. Computing the number of operations needed to evaluate the objective functions by the different methods results in

Method	Cost of function evaluation
HOMPACK	$p(n-1)+3n_A$
MORGAN	$\leq p(n-1)+3n_A+4s$
SCALELP	$p(n-1)+(n+m+1)+3n_A$
SCALEIT	$\leq 2n+n(m+n+1)$

where n_A denotes the non zero entries in A and $s := \binom{p}{2}$. Thus, for larger problems only HOMPACK, SCALELP or SCALEIT are suitable, and for very large problems SCALEIT is the only alternative.

6 Conclusion

In view of the stability problems for HOMPACK and MORGAN, and their breakdown for linear problems, for nonlinear problems with a symmetric structure, and for problems with fewer constraints than variables, only SCALELP or SCALEIT are suitable as general purpose scaling methods. For very large problems, the higher complexity of SCALELP makes SCALEIT the method of choice.

Acknowledgements We thank an anonymous referee whose numerous suggestions markedly improved the presentation of the paper.

References

1. Benzi, M., Haws, J.C., Tuma, M.: Preconditioning highly indefinite and nonsymmetric matrices. SIAM J. Sci. Comput. **22**, 1333–1353 (2000)
2. Buttrey, S.: The lpSolve package. http://cran.mirroring.de/doc/packages/lpSolve.pdf (2007)
3. Curtis, A.R., Reid, J.K.: On the automatic scaling of matrices for gaussian elimination. IMA J. Appl. Math. **10**(1), 118–124 (1972)
4. Domes, F.: Verified global optimization with Gloptlab. http://www.mat.univie.ac.at/dferi/ICIAM07.pdf (2007)
5. Domes, F.: GloptLab, a configurable framework for the rigorous global solution of quadratic constraint satisfaction problems. in preparation (2007–2008)
6. Domes, F., Neumaier, A.: Directed cholesky factorizations and applications. submitted (2008)
7. Duff, I.S., Koster, J.: The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. SIAM J. Matrix Anal. Appl. **20**, 889 (1999)
8. Fan, H.-Y., Lin, W.-W., Van Dooren, P.: Normwise scaling of seond order polynomial matrices. SIAM J. Matrix Anal. Appl. **26**, 252–256. http://www.inma.ucl.ac.be/publi/258602.pdf (2004)
9. Hagemann, M., Schenk, O.: Weighted matchings for the preconditioning of symmetric indefinite linear systems. SIAM J. Sci. Comp. **28**, 403–420. http://www.computational.unibas.ch/cs/scicomp/ (2006)
10. Higham, N.J., Mackey, D.S., Tisseur, F.: The conditioning of linearizations of matrix polynomials. SIAM J. Matrix Anal. Appl. **28**, 1005–1028. http://eprints.ma.man.ac.uk/669/ (2006)
11. Higham, N.J., Mackey, D.S., Tisseur, F., Garvey, S.D.: Scaling, sensitivity and stability in the numerical solution of quadratic eigenvalue problems. Int. J. Num. Math. Eng. **73**, 344–360. http://eprints.ma.man.ac.uk/997/ (2008)
12. Kim, S., Kojima, M.: Numerical stability of path tracing in polyhedral homotopy continuation methods. Computing **73**, 329–348. http://math.ewha.ac.kr/~skim/Research/B-390.pdf (2004)
13. Meintjes, K., Morgan, P.: Chemical equilibrium systems as numerical test problems. ACM Trans. Math. Software **16**, 143–151 (1990)

14. Morgan, A.: Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems. Prentice-Hall (1987)
15. Olschowka, M., Neumaier, A.: A new pivoting strategy for Gaussian elimination. Linear Algebra Appl. **240**, 131–151 (1996)
16. Parlett, B.N., Landis, T.L.: Methods for scaling to double stochastic form. Linear Algebra Appl. **48**, 53–79 (1982)
17. Rice, J.R.: Matrix Computation and Mathematical Software. McGraw-Hill (1981)
18. Schenk, O., Gartner, K.: On fast factorization pivoting methods for sparse symmetric indefinite systems. Elec. Trans. Num. Anal. **23**, 158–179. http://informatik.unibas.ch/personen/schenk_o.html (2006)
19. Schenk, O., Rollin, S., Gupta, A.: The effects of unsymmetric matrix permutations and scalings in semiconductor device and circuit simulation. Computer-Aided Des. Integ. Circuits Syst. **23**, 400–411. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1269862. (2004)
20. Verschelde, J., Verlinden, P., Coolsi, R.: Homotopies exploiting newton polytopes for solving sparse polynomial systems. SIAM J. Numer. Anal. **31**, 915–930. http://scitation.aip.org/getpdf/servlet/GetPDFServlet?filetype=pdf&id=SJNAAM000031000003000915000001&idtype=cvips&prog=normal. (1994)
21. Watson, L.T., Terry, L.: HOMPACK: a suite of codes for globally convergent homotopy algorithms. http://deepblue.lib.umich.edu/dspace/bitstream/2027.42/8204/5/ban6930.0001.001.pdf (1985)