Reliable Outer Bounds for the Dual Simplex Algorithm with Interval Right-hand Side

Christoph Fünfzig
Fraunhofer Institute for Industrial Mathematics
Kaiserslautern, Germany
Email: c.fuenfzig@gmx.de

Dominique Michelucci, Sebti Foufou
Le2i, University of Burgundy
Dijon, France
Email: {dominique.michelucci, sebti.foufou}@u-bourgogne.fr

***Abstract*—In this article, we describe the reliable computation of outer bounds for linear programming problems occuring in linear relaxations derived from the Bernstein polynomials. The computation uses interval arithmetic for the Gauss-Jordan pivot steps on a simplex tableau. The resulting errors are stored as interval right hand sides. Additionally, we show how to generate a start basis for the linear programs of this type. We give details of the implementation using OpenMP and comment on numerical experiments.**

***Keywords*-verified simplex algorithm; interval arithmetic; tableau form; OpenMP parallelization**

I. Introduction

Linear relaxation [1] is a common method to solve non-linear systems in several variables with domains $D_i \subset \mathbb{R}$, $i = 1, \ldots, N$. For the system with variables $x_1 \in [0, 1]$ and $x_2 \in [0, 1]$,

$$\begin{aligned} x_1^2 - x_2 &= 0 \\ x_2 - x_1 &\le 0 \end{aligned}$$

a linear relaxation derived from the tangent plane in $x_1 = x_2 = 0.5$ is

$$\begin{aligned} &(x_1 - 0.5) - (x_2 - 0.5) - 0.25 \ge 0 \\ &x_2 - x_1 \le 0 \end{aligned}$$

In [2], we presented linear relaxations for the monomials x_i^2 and $x_i x_j$, $i < j$ with $x_i \in D_i$, $x_j \in D_j$, which are derived from the Bernstein polynomials on domain D. The curves (x_i, x_i^2), and the surfaces $(x_i, x_j, x_i x_j)$, $i < j$ are enclosed in a polytope, called *Bernstein polytope* (Figure 2).

With linear relaxation, a quadratic system

$$F(x) = 0,\ G(x) \ge 0,\ x = (x_1, \ldots, x_N)$$

gives a linear system in the variables x_i, x_{ii}, and x_{ij}, $i < j$. For example, a lower bound for component i can be obtained by solving a linear program: minimize x_i on the linear system obtained with the Bernstein polytope for domain D.

In this article, we show how to use interval arithmetic in a tableau-form implementation of the dual simplex algorithm (Section II) to verify computations and to generate a tight lower bound on the minimum value. The tableau has floating-point entries and uses an interval right-hand side. In a pivot operation of the Gauss-Jordan algorithm, rounding errors are collected and stored in the right-hand side intervals (Section II-B). For the application of linear relaxations using the Bernstein polytope, we give two ways to generate a start basis for the occurring linear programs in Section II-C. Finally, we conclude on this work in Section IV.

A. Related Work

In an overview, there are three classes of methods for solving polynomial systems. Using computer algebra, polynomial expressions (*resultants*) can be defined, which are equal to zero if and only if the polynomials have a common root. They provide simple and effective methods for low degree problems due to the degree of the resulting expression. Decomposing an arbitrary polynomial in the ideal of the given polynomials is possible with a special generator of the ideal. Such an ordered generator (*Gröbner basis*) can be used to compute common polynomial factors. From a computational point of view, these methods need exact arithmetic. Recently, some articles [3] work towards their numerical computation with tools from interval arithmetic.

An established semi-numerical method solves a given polynomial system P with the same number of equations as variables. The method uses a continuation method [4] with a scalar parameter $t \in [0, 1]$ to deform a polynomial system P^0 with known solutions to the given system P. Finding an initial simple polynomial system P^0 for the given P is the main difficulty of the method.

In this article, we solve quadratic polynomial systems by *branch-and-bound* using a linear relaxation for the monomials. We compute an outer bound for the optimum value of the linear program reliably. In [2], the revised simplex code *SoPlex* [5] in floating point arithmetic and a backward analysis of the final linear system for the objective value was used. The article [6] gives a comparison of linear programming codes using rational arithmetic and floating-point arithmetic. Of course, the use of floating-point arithmetic, which is used at least in parts of the code, is significantly faster than exact rational arithmetic. [7] describes how to compute a lower bound using an arbitrary linear program solver. The authors use the *weak optimality theorem of linear programming*: Any feasible point y of the dual problem $A^t y \le c$ ($\max b^t y$) gives a lower bound $b^t y$ for the minimum of the primal problem $Ax = b$, $x \ge 0$ ($\min c^t x$). As outlined in [7], the lower bounds obtained from a verified, feasible point can be away from the optimum value for ill-conditioned problems. The article [7] additionally

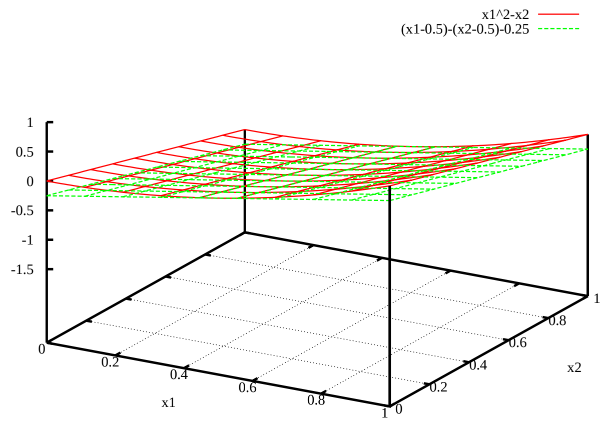


Fig. 1: Linear relaxation (dashed below) for $x_1^2 - x_2$ (solid above).

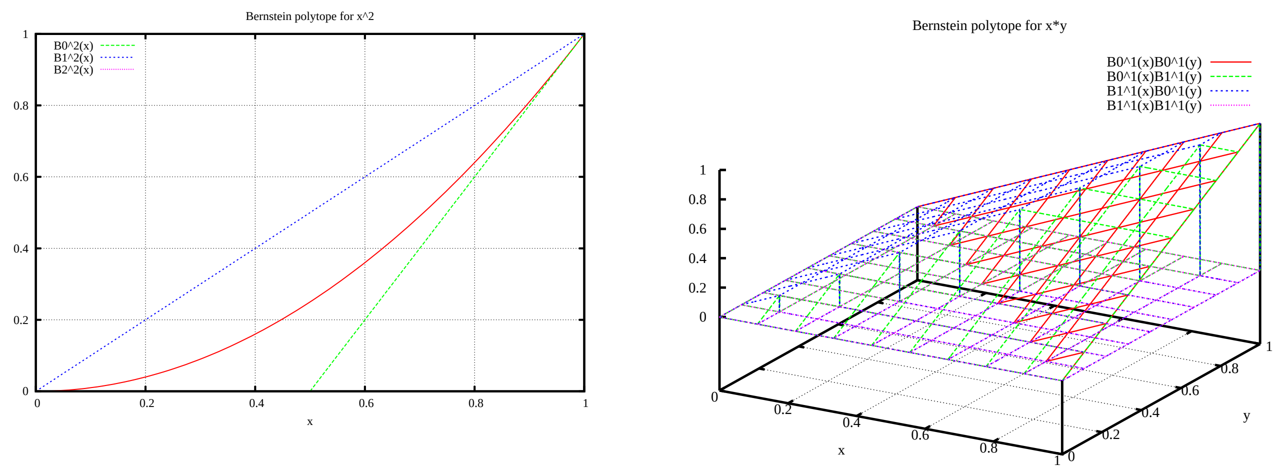


Fig. 2: Bernstein polytope enclosing the curve (left) (x, x^2): $B_0^2(x) \geq 0$, $B_1^2(x) \geq 0$, $B_2^2(x) \geq 0$. The surface (right) (x, y, xy): $B_0^1(x)B_0^1(y) \geq 0$, $B_1^1(x)B_0^1(y) \geq 0$, $B_0^1(x)B_1^1(y) \geq 0$, $B_1^1(x)B_1^1(y) \geq 0$.

considers the computation of upper bounds. Concerning the parallelization of simplex codes, [8] summarizes a number of attempts. The authors of [9] consider the parallelization of the sparse dual simplex code *CPLEX*. They make out the steepest-edge pricing rule as a good candidate for parallelization, which compares all infeasible rows based on the resulting change of the dual variables.

B. Notation

We use $\underline{R}$ for the lower bound of an interval R and $\overline{R}$ for the upper bound. median(R) denotes the median of the interval bounds $\underline{R}$ and $\overline{R}$ computed as $[0.5\,(\underline{R} + \overline{R})]^-$. For a real number a, we denote by a^- the largest floating-point number smaller or equal to a, and by a^+ the smallest floating-point number larger or equal to a. We denote by e_k the kth vector of the canonical basis with $e_{k,k} := 1$, and 0 otherwise. As usual, an inequality between vectors, like $x \geq 0$, applies to all components i: $x_i \geq 0$.

II. Linear Programming Problem

A linear program in standard form is defined by

$$\begin{aligned} &\min c^t x \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

where A is a $m \times n$ real matrix, b is a m-component real vector, and c is a n-component real vector. The system $Ax = b$ contains the linear equality constraints, and the function $c^t x$ defines the linear objective function to be minimized. An inequality $a_1^t x \leq b_1$ is transformed into an equality by a new variable $x_s \geq 0$: $a_1^t x + x_s = b_1$, which is called a *slack variable* [10]. Note that in our case it is $m \leq n$, i.e., our problem has at least as many variables as rows due to the slack variables. There are several ways to ensure non-negativity of variables. The first way is to use symbolic substitution of the problem variables $\tilde{x_i} \rightarrow (x_i - \underline{D_i})$ if $\underline{D_i} < 0$ and change all equations and inequalities into problem variables $\tilde{x_i} \in [0, \overline{D_i} - \underline{D_i}]$. This adds a linear number of additional terms to the given problem. The second and preferred way is to substitute all variables x_i and x_{ij} locally into $\tilde{x_i} \rightarrow (x_i - \underline{D_i})$ resp. $\tilde{x_{ij}} \rightarrow (x_{ij} - \underline{D_i \cdot D_j})$ so that $x_i = \tilde{x_i} + \underline{D_i}$ resp. $x_{ij} = \tilde{x_{ij}} + \underline{D_i \cdot D_j})$. Then row r changes into $\sum_k a_{rk}(\tilde{x_k} + \underline{D_k}) = b_r$ which needs to be done only once during tableau setup.

The tableau-form implementation of the simplex algorithm (with column basis) selects a maximal subset of m linear independent columns of A (corresponding to the basic variables of x), where m is the rank of the matrix A. The subset with index set B is called a *basis*, and the corresponding submatrix $A_{*,B}$ is invertible; the rest matrix is denoted by $A_{*,N}$. Non-basic variables always have a zero value. A basis update operation maintains the reduced row-echelon form of the tableau ($x_B = A^{-1}_{*,B} b$)

$$\begin{pmatrix} c^t_B x_B & (c_B - c^t_B A^{-1}_{*,B} A_{*,B})^t & (c_N - c^t_B A^{-1}_{*,B} A_{*,N})^t \\ x_B & A^{-1}_{*,B} A_{*,B} & A^{-1}_{*,B} A_{*,N} \end{pmatrix} \tag{1}$$

which allows to look up the reduced costs in the first row, the objective function value in the first column of the first row, and the basic variable values x_B in the first column below the first row of this tableau [10]. Furthermore, we group matrix rows into equalities (without a slack variable) given by the index set E and inequalities (each having a slack variable) given by the index set I. Maintaining this form is possible with the Gauss-Jordan algorithm from numerical linear algebra. As tableau rows define basis variables, it is possible to check the non-negativity constraints $x \geq 0$ and to select a leaving variable in the simplex algorithm (*pricing* [10]). The leaving variable is replaced by an entering variable, which can be selected from the reduced costs in the first tableau row (*ratio test* [10]). For applications, this *dual form of the simplex algorithm* is beneficial [11], which changes an infeasible basis with a sub-minimum value into an optimum, feasible basis. It selects the leaving variable from the infeasible basis first and replaces it by the entering variable. In contrast the *primal form of the simplex algorithm*, selects the entering variable first, then selects the leaving variable until an optimum value is reached.

A. Pricing Rule and Ratio Test

Important for the performance of the solver is the pricing rule, the ratio test, and the start basis [11]. For the pricing rule, we consider the *steepest-edge* rule

Definition 1 (Goldfarb-Forrest pricing rule): Select row r which has the most negative ratio $\frac{x_r}{|e^t_r A^{-1}_{*,B}|_2}$.
where $d = e^t_r A^{-1}_{*,B} A_{*,N}$ is the change of the dual variables per unit change of x_r (see Equation 1). The values x_r are stored as right-hand sides described in the following Section II-B, and A^{-1}_{*B} is part of the tableau.

For the ratio test, we use

Definition 2 (Harris ratio test): Select column s so that $a_{r,s}$ is minimum with $\frac{c_j}{a_{r,j}} \geq \theta_r(\epsilon)$, $a_{r,j} < 0$, $\theta_r(\epsilon) := \min\{\frac{c_j + \epsilon}{a_{r,j}} : a_{r,j} < 0\}$.
This rule chooses the element $\frac{c_s}{a_{r,s}}$ of the set $\{\frac{c_j}{a_{r,j}} \geq \theta_r(\epsilon) : a_{r,j} < 0\}$ defined by a small parameter $\epsilon > 0$. This allows to choose the denominator $a_{r,s} < 0$ with largest magnitude for the division. Note that c_j is non-negative up to rounding errors for the dual simplex algorithm always. The ratio test traverses the objective function row and the row r of the tableau in parallel.

B. Pivot Steps using Interval Arithmetic

For the collection of computer arithmetic errors during a pivot step of the Gauss-Jordan algorithm, we use interval arithmetic. Let $a_{r,s}$ be the pivot element in row r, and D_i the variable domain for variable x_i. Then the linear equation

$$\sum_j \frac{a_{r,j}}{a_{r,s}} x_j = \frac{b_r}{a_{r,s}}$$

transforms into an interval equation

$$\sum_j R_{r,j} x_j = R_r$$

where we can select a floating-point value $a_{r,j} \in R_{r,j}$ of the interval so that $R_{r,j} \subset a_{r,j} + [(\underline{R_{r,j}} - a_{r,j})^-, (\overline{R_{r,j}} - a_{r,j})^+]$. The intervals can be collected and stored as an interval right hand side R'_r.

$$\sum_j a_{r,j} x_j = R_r - \sum_j [(\underline{R_{r,j}} - a_{r,j})^-, (\overline{R_{r,j}} - a_{r,j})^+] D_j =: R'_r \tag{2}$$

With the representative $a_{r,j} :=$ median$(R_{r,j})$, the resulting interval $[(\underline{R_{r,j}} - a_{r,j})^-, (\overline{R_{r,j}} - a_{r,j})^+]$ has smallest width. Figure 3 shows the hyperplane arrangement for an example.

Similarly, a row operation as required in the Gauss-Jordan algorithm between row r and row i

$$\sum_j (a_{i,j} - a_{r,j} \frac{a_{i,s}}{a_{r,s}}) x_j = b_i - b_r \frac{a_{i,s}}{a_{r,s}}$$

can be performed in interval arithmetic

$$\sum_j R_{i,j} x_j = R_i$$

and rewritten using an interval right-hand side

$$\sum_j a_{i,j} x_j = R_i - \sum_j [(\underline{R_{i,j}} - a_{i,j})^-, (\overline{R_{i,j}} - a_{i,j})^+] D_j =: R'_i \tag{3}$$

In this form, a sufficient condition for the feasibility of x_i, $i \in B$ is $\underline{R_i} \geq \underline{D_i}$. In case $\overline{R_i} < \underline{R_i}$, it is infeasible and a candidate for the pricing rule. Otherwise some R_i have smaller and larger than $\underline{D_i}$, in which case we stop the solving process with a lower bound of the optimum value. Due to use of interval right-hand sides, it is possible to reduce the number of variables $\{x_j\}$ by replacing a variable x_j with its interval

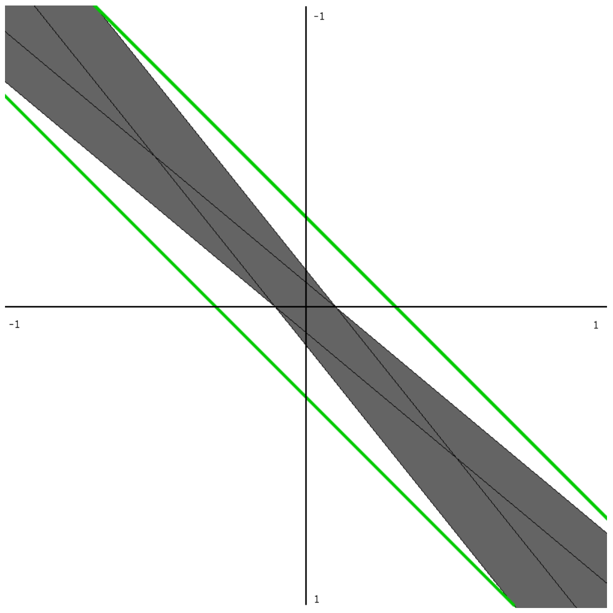


Fig. 3: Hyperplane arrangement (grey) for the interval equation $[0.8, 1.2]x + [1.0, 1.0]y + [-0.5, 0.5] = 0$. A thick hyperplane results from the selection of interval representatives $1.0x + 1.0y + [-0.7, 0.7] = 0$.

D_j. In this way, an active subset of variables of the linear program can be defined.

In the geometric view of the polytope defined by interval right-hand sides, the thick hyperplanes bound a set of polytopes, which are not necessarily of the same topology. See Figure 4 for an example, where the minimum-y vertex of the outer hyperplanes is defined by the intersection of r_1 and r_3, but the minimum-y vertex of the inner hyperplanes is defined by the intersection of r_1 and r_2.

Note that these topology changes make situations possible, where the outer polytope is non-empty but the inner polytope is empty. In such cases, the algorithm can not decide feasibility of the linear program.

C. Start Basis Generation

Inside the non-linear solver, we only have to handle objective functions of the form $x_i = e_i^t x$ for a variable index i. Start basis generation can be done by performing the Gauss-Jordan algorithm on the equation part $A_{E,*}$ of the system. This defines a subset B_E of the basis B. Note that the set B_E depends on the pivot selection strategy in the Gauss-Jordan algorithm and does not need to change during pivoting in the simplex algorithm. Also pricing rule and ratio test only consider the inequalities I and the inequalities of the Bernstein polytope.

The first possibility is to select only pivots with column index different from i. In this case, where B_E does not contain variable index i, we can easily complete the basis from the vertex with smallest value x_i of the Bernstein polytope for (x_i, x_i^2). I.e., for minimization (with $B_i^{(2)}$ the i-th quadratic

objective function	
user equalities, n vars	$u + 3c + 4d$ slack vars
u user inequalities, n vars	$u + 3c + 4d$ slack vars
$3c$ Bernstein inequalities	$u + 3c + 4d$ slack vars
$4d$ Bernstein inequalities	$u + 3c + 4d$ slack vars

Fig. 5: Tableau organization with regions for the user system and for the Bernstein polytope.

Bernstein polynomial)

$$\begin{array}{lcl} (B_0^{(2)}(x_i) & \geq & 0) \\ B_1^{(2)}(x_i) = 2(-x_{ii} + \boxed{(u_i + v_i)x_i} - u_i v_i) & \geq & 0 \\ B_2^{(2)}(x_i) = \boxed{x_{ii}} - 2u_i x_i + u_i^2 & \geq & 0 \end{array}$$

for maximization

$$\begin{array}{lcl} B_0^{(2)}(x_i) = x_{ii} + \boxed{-2v_i x_i} + v_i^2 & \geq & 0 \\ B_1^{(2)}(x_i) = 2(\boxed{-x_{ii}} + (u_i + v_i)x_i - u_i v_i) & \geq & 0 \\ (B_2^{(2)}(x_i) & \geq & 0) \end{array}$$

The second possibility is to select a pivot with column index i. In this case, where B_E contains the variable index i, we can generate a start basis from the equation row k defining variable x_i. Let $x_i + \sum_{j \neq i} a_{k,j} x_j = R_k$ be row k. If there are columns $a_{k,j} > 0$ the current basis part B_E is not optimum, and it can be changed by primal steps into an optimum basis. I.e., for each such column j with $a_{k,j} > 0$ we determine a row r such that $a_{r,0} \geq 0$ and $-a_{r,0} \frac{a_{k,j}}{a_{r,j}} < 0$ is minimum. Both strategies are compared in Section III based on a numerical example.

III. Implementation and Numerical Experiments

We implemented the dual simplex algorithm in C/C++ using the tableau organization shown in Figure 5. The tableau can be stored as an $m \times n$ array of `double`-entries or in a sparse form as an array of m rows of index/entry pairs. The right-hand side vector is represented using the *boost* interval arithmetic library. We additionally keep a basis description consisting of arrays, *var* giving the index of the defining row for a variable index, and *row* giving the index of the variable defined in a row. Both are inverse to each other: $row[var[j]] = j$ for variable index j and $var[row[i]] = i$ for row index i. We have one pivot operation $\frac{1}{a_{r,s}} A_{r,*}$ for the pivot row r, and m row operations $A_{i,*} + f_i A_{r,*}$ for all other rows $i \neq r$. The m different row operations can be done in parallel using OpenMP. We use the basis description to exclude basis columns, which are unit vectors and therefore contain a zero value in the pivot row r. Altogether, the number of multiplications and additions for a row operation ranges from $n - m$ to n. Note that in the interval version (Equations 2 and 3), the pivoting and row operations are not entrywise as in [12] but require a reduction operation for the right-hand side interval. We avoid a parallel reduction by storing the right-hand side updates $[(\underline{R_{i,j}} - a_{i,j})^-, (\overline{R_{i,j}} - a_{i,j})^+]D_j$ in an array and perform the summation sequentially. Similarly, we perform the loop for the steepest edge pricing rule and the loop for the ratio test in parallel using OpenMP but without a critical section for the minimization.

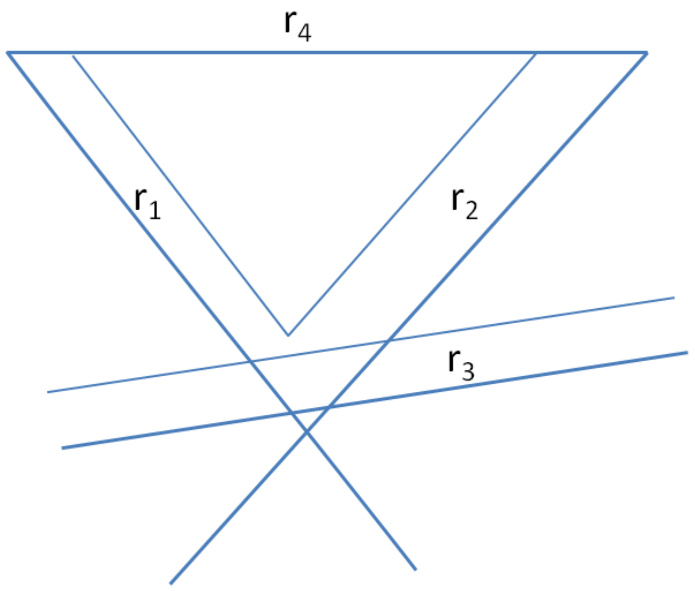


Fig. 4: Polytope bounded by thick hyperplanes r_1, r_2, r_3 and r_4. Note that the topology of the polytope built by the outer hyperplanes (thick) is not the same as the one by the inner hyperplanes (thin).

In the following, we demonstrate the different strategies for start basis generation on the example system *mixed.sys*:

$$\begin{aligned} 0.5x_1 &= x_2x_3 \\ 0.5x_2 &= x_1x_3 \\ 0.5x_3 &= x_1x_2 \end{aligned}$$

on $D_1 = D_2 = D_3 = [-1, 1]$. The basis variables are marked with a box around them. When pivoting with x_{23}, x_{13}, x_{12}, the system is in reduced row-echelon form

$$\begin{aligned} 0.5x_1 &= \boxed{x_{23}} \\ 0.5x_2 &= \boxed{x_{13}} \\ 0.5x_3 &= \boxed{x_{12}} \end{aligned}$$

and can be completed with two inequalities of the Bernstein polytope for x_{ii}, $i = 1, 2, 3$ into a start basis. Figure 6, left, shows a statistics of the interval width of the objective value in the course of pivoting. The computation as described in [2] performs one reduction of each variable before bisecting the largest interval. For this problem (tableau size $m = 19 \times n = 33$) on $D_1 = D_2 = D_3 = [-1, 1]$, it needs 47 reductions (410 pivot steps in total), 10 bisections, and the solution time is 0.30s using the sparse form (Windows 7/Visual Studio 2008, Intel Pentium P8700, Dual Core 2.53GHz). The same solving using the dense form takes 0.82s and produces different error widths. The largest condition number of the tableau is 2.0.

When pivoting with x_1, x_{13}, x_{12}, the system is in reduced row-echelon form

$$\begin{aligned} 0.5\boxed{x_1} &= x_{23} \\ 0.5x_2 &= \boxed{x_{13}} \\ 0.5x_3 &= \boxed{x_{12}} \end{aligned}$$

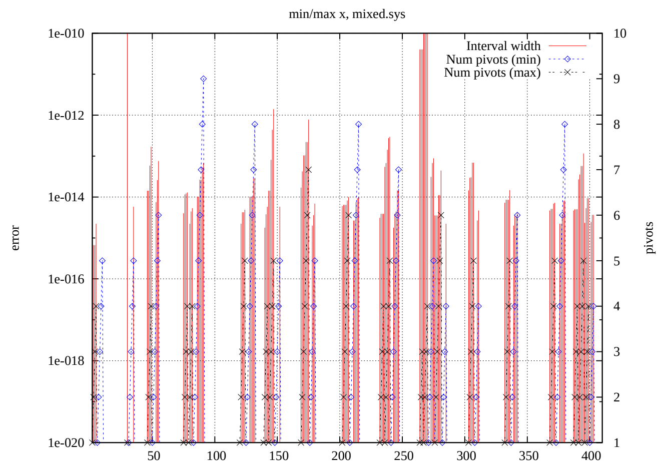
and can be completed with three inequalities of the Bernstein polytope x_{23} into a start basis. Note that the reduced row-echelon form for variables x_2 and x_3 is similar and thus omitted here. Figure 6, right, shows a statistics of the interval width of the objective value in the course of the pivot steps. For this problem, the computation performs 47 reductions (275 pivot steps in total), 10 bisections, and the solution time is 0.32s. The same solving using the dense form takes 0.72s. The worst condition number of the tableau is 846.3, and it results in larger objective value intervals, i.e., worse lower bounds. In the comparison, the second start basis results in less pivot steps in the dual simplex iteration. Tableaus of larger condition number normally occur if very small pivots were chosen. This can be necessary for Bernstein inequalities corresponding to very small intervals D_i. It is possible to replace such a Bernstein polytope by a thick plane or a thick line as described in [2].

In general, the tableau method tends to populate rows quickly. On the system *mixed.sys*, the user and Bernstein region of the tableau get filled approximately 60% to 80%.

For comparison with the revised simplex implementation *SoPlex*, we compute a rigorous lower bound $b^ty + e$ using the duality gap

$$e = \min\{(c^t - y^{*\,t}A)x : x \in D\}$$

The primal solution vector x^* is directly available, and the corresponding dual solution vector y^* can be derived from the constraint slackness at x^*. When solving the system *mixed.sys*, the code performs around 500 pivot steps, which are fast due to the revised simplex implementation. But large duality gap sizes (larger than 10^{-10}) occur for linear programs, where no bound reduction could be achieved due to an early termination in *SoPlex*.



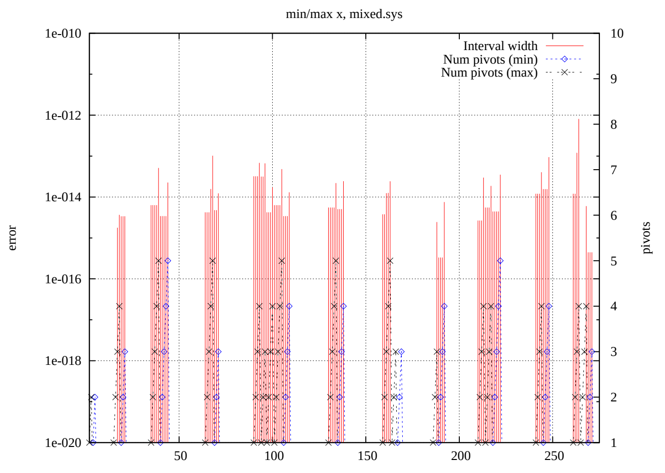


Fig. 6: System *mixed.sys* with start basis from Bernstein polytope for x_{ii} (left) and from the equation system itself (right). Errors are derived from the interval width of the objective value.

IV. Conclusion

In this paper, we presented a new way to implement the dual simplex algorithm in tableau form with pivot steps using interval arithmetic for the direct computation of a reliable lower bound. Such an algorithm can be used for example in a polynomial system solver using linear relaxations. Compared to a lower bound computed from the duality gap, it is not so much affected by the condition number of the given linear program and an early termination of the simplex code. But due to the use of the tableau form it performs more floating-point operations than a revised simplex implementation (e.g., SoPlex [5]) for a sparse input system due to the gradual increase of non-zeros during pivot steps.

The computation is based on the Gauss-Jordan algorithm and performs pivot steps in interval arithmetic that are parallelizable with OpenMP. We always avoid critical sections for reduction operations. In the steepest edge pricing rule and the ratio test, a suboptimal choice does not produce wrong results and we could not observe any performance degradation. For tableau storage, we choose row major order, which avoids recomputing the row factors for a row operation. We prefer sparse storage (as rows of index/entry pairs) over dense (as an array). To avoid further fill-in, we make use of a separate basis description so that the locations of unit vectors in the tableau are known.

For large systems, it is possible to work with an active set of variables $\{x_j\}$ and replace all others with their interval D_j, which is a major benefit of using an interval right-hand side. But with such wide intervals the topology changes, outlined in Figure 4, need to be handled.

Acknowledgements

This research work has been funded by NPRP grant number NPRP 09-906-1-137 from the Qatar National Research Fund (a member of The Qatar Foundation).

References

[1] R. Kearfott, “Discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization,” *Optimization Methods and Software*, vol. 21, no. 5, 2006, pp. 715–731.

[2] Ch. Fünfzig, D. Michelucci, S. Foufou, “Nonlinear systems solver in floating-point arithmetic using LP reduction,” in *ACM/SIAM Symposium on Solid and Physical Modeling*, 2009, pp. 123–134.

[3] M. Bodrato, A. Zanoni, “Intervals, syzygies, numerical Gröbner bases : A mixed study,” in *CASC 2006 Proceedings*, V. G. Ganza, E. W. Mayer, and E. V. Vorozhtsov, Eds., vol. 4194. LNCS, Springer, September 2006, pp. 64–76.

[4] J. Verschelde, “Polynomial homotopy continuation with PHCpack,” *ACM Communications in Computer Algebra*, vol. 44, no. 4, 2010, pp. 217–220.

[5] R. Wunderling, “SoPlex library version 1.4.2,” 1996.

[6] C. Keil, “A comparison of software packages for verified linear programming,” *Preprint Institute of Reliable Computing, Hamburg University of Technology*, 2008.

[7] ——, “Lurupa – rigorous error bounds in linear programming,” in *Algebraic and Numerical Algorithms and Computer-assisted Proofs, Dagstuhl Seminar Proceedings (Number 05391)*, B. Buchberger, S. Oishi, M. Plum, and S. Rump, Eds., July 2006.

[8] J. Hall, “Towards a practical parallelisation of the simplex method,” *Computational Management Science*, vol. 7, no. 2, 2010, pp. 139–170.

[9] R. E. Bixby, A. Martin, “Parallelizing the dual simplex method,” *INFORMS Journal on Computing*, vol. 12, no. 1, Jan. 2000, pp. 45–56.

[10] C. Papadimitriou, K. Steiglitz, *Combinatorial optimization: Algorithms and Complexity*. Dover, 1998.

[11] R. Bixby, “Solving linear and integer programs,” in *Block Course Combinatorial Optimization at Work, Berlin*, M. Grötschel, Ed., 2009.

[12] S. F. McGinn, R. E. Shaw, “Parallel Gaussian Elimination using OpenMP and MPI,” in *16th Annual Int. Symp. on High Performance Computing Systems and Applications, Moncton, Canada*, June 2002, pp. 169–176.