OPTIMIZATION TECHNOLOGY CENTER

PCx User Guide (Version 1.1)[1]

by

Joseph Czyzyk, Sanjay Mehrotra, Michael Wagner, and Stephen J. Wright

Technical Report OTC 96/01

November 3, 1997

ABSTRACT

We describe the code PCx, a primal-dual interior-point code for linear programming. Information is given about problem formulation and the underlying algorithm, along with instructions for installing, invoking, and using the code. Computational results on standard test problems are tabulated. The current version number is 1.1.

Key words: linear programming, interior-point methods, software.

[1] This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

1 Introduction

PCx is a linear programming solver developed at the Optimization Technology Center at Argonne National Laboratory and Northwestern University. It implements a variant of Mehrotra's predictor-corrector algorithm [8] with the higher-order correction strategy of Gondzio [4]. This approach is the most effective one known at present for general linear programs. The bulk of PCx is written in the C programming language. Its main computational operation—solution of a sparse linear system of equations at each iteration—is performed by a call to the sparse Cholesky package of Ng and Peyton [9], which is programmed in Fortran 77. Source codes for both PCx and the Ng-Peyton linear equations solver can be found in the PCx distribution file. They are available subject to the qualifications in the copyright statement on the PCx home page on the World Wide Web (see Section 7).

Key features of PCx include

- a set of high-level data structures for linear programming constructs, designed for possible reuse in other codes;
- ability to be invoked both as a stand-alone program (with input from an MPS file) and as a callable procedure;
- a presolver;
- modular structure, which makes it easy for users to modify the code to experiment with variants of the current algorithm; and
- a simple interface to the linear equations solver, which allows straightforward linking of alternative solvers to the body of PCx.

The current version of PCx performs efficiently on the standard netlib test problems. Nevertheless, PCx should be viewed as work in progress. Features such as finite termination/basis recovery and alternative linear algebra solvers (and alternative formulations of the step equations) may be added in future versions. In making the source available, we encourage others to become involved in the development and extension of PCx.

The remaining sections of this guide contain an outline of the underlying algorithm, instructions for installing and using PCx, and computational results on standard test problems. Section 2 describes the various linear programming formulations that are accommodated by the data structures of PCx, including the formulation to which the algorithm is actually applied. Section 3 describes the algorithm, including details of termination and infeasibility detection. Section 4 discusses the major computational issue in the code—factorization of a sparse, positive definite matrix—including the modifications to the Ng-Peyton code [9] needed in this context. (Alternative factorization modules to the Ng-Peyton code will require similar modifications.) Presolver capabilities are outlined in Section 5. The user can set various algorithmic options and control the amount and type of output by means of a specifications file; details are provided in Section 6. Section 7 contains instructions for installing the code in a Unix environment, while instructions for invoking PCx as a stand-alone solver are given in Section 8. Section 9 is a brief description of the interface with the sparse Cholesky solver, showing the user how alternative solvers can be hooked up to PCx without the need to understand or modify the bulk of the code. (In addition to the Ng-Peyton solver, the PCx distribution contains the routines needed to link to IBM's WSSMP solver [5]. Unlike the Ng-Peyton solver, the WSSMP library is proprietary and must be obtained separately.) Finally, Section 10 reports on computational results for the standard netlib test set of feasible and infeasible problems, together with some new problems arising from the NEMS project at Argonne.

This guide will be updated continually as new releases of PCx are made available.

2 The Formulation

PCx accepts any valid linear program that can be specified in the MPS format. The model described in the MPS file may include upper and lower bounds, linear equality constraints, linear inequality constraints and free variables. PCx defines a data structure `MPStype` that contains a complete specification of a single linear programming problem in this general formulation. This data structure also stores the names assigned to the rows, columns, and objectives of the model specified in the MPS file.

For algorithmic purposes, however, it is convenient to work with a simpler formulation of the linear program. PCx converts the general formulation to the following simpler form:

$$\min_{x \in R^n} c^T x \ \text{ subject to } \ Ax = b \qquad \begin{array}{ll} 0 \le x_i, & i \in \mathcal{N} \\ 0 \le x_i \le u_i, & i \in \mathcal{U} \\ x_i \text{ free}, & i \in \mathcal{F}, \end{array} \tag{1}$$

where $\mathcal{N} \cup \mathcal{U} \cup \mathcal{F}$ is a partition of the index set $\{1, 2, \ldots, n\}$ into "normal," "upper-bounded," and "free" variables, respectively. The PCx data structure `LPtype` contains a single linear program in the form (1). The transformation from an `MPStype` formulation to an `LPtype` formulation is carried out internally and transparently to the user by PCx. After the solution has been found, the transformation from `MPStype` to `LPtype` is inverted to express the solution in terms of the original formulation.

Users who circumvent the MPS file and call the procedure `PCx()` directly must specify their problems in the form (1). That is, they pass an `LPtype` data structure to this procedure.

The current version of PCx carries out one more level of problem transformation before invoking the solution algorithm. The use of a normal equations formulation of the step equations (see below) implies that the model can contain no free variables. Hence, we replace each of the free variables x_i in the `LPtype` formulation by a pair of normal variables x_i^+ and x_i^-, making the substitution

$$x_i = x_i^+ - x_i^-.$$

After these substitutions are made (and the notation is redefined), the linear program has the following form:

$$\min_{x \in R^n} c^T x \ \text{ subject to } \ Ax = b \qquad \begin{array}{ll} 0 \le x_i \le u_i, & i \in \mathcal{U} \\ 0 \le x_i, & i \in \bar{\mathcal{U}}, \end{array} \tag{2}$$

where $\bar{\mathcal{U}} = \{1, 2, \ldots, n\} \backslash \mathcal{U}$. The split variables are recombined before return from `PCx()`, so the transformation between (1) and (2) is transparent to the user. The `LPtype` data structure is also used to store problems in the form (2).

The dual problem associated with (2) is

$$\begin{array}{rrcll} \displaystyle\max_{\pi \in R^m, r \in R^{|\mathcal{U}|}, s \in R^n} & \displaystyle b^T\pi - \sum_{i \in \mathcal{U}} u_i r_i & & & \\ \text{subject to} & A_{i\cdot}^T \pi + s_i - r_i & = & c_i & i \in \mathcal{U} \\ & A_{i\cdot}^T \pi + s_i & = & c_i & i \in \bar{\mathcal{U}} \\ & (r, s) & \geq & 0, & \end{array} \tag{3}$$

where π is the Lagrange multiplier vector for the equality constraint $Ax = b$, and r represents the Lagrange multipliers for the upper bounds $x_i \leq u_i$. The Karush-Kuhn-Tucker (KKT) optimality conditions for (2) and (3) are

$$A_{i\cdot}^T \pi + s_i - r_i = c_i, \quad i \in \mathcal{U} \tag{4a}$$
$$A_{i\cdot}^T \pi + s_i = c_i, \quad i \in \bar{\mathcal{U}} \tag{4b}$$
$$Ax = b \tag{4c}$$
$$x_i + w_i = u_i, \quad i \in \mathcal{U}, \tag{4d}$$
$$x_i s_i = 0, \quad i = 1, 2, \ldots, n, \tag{4e}$$
$$w_i r_i = 0, \quad i \in \mathcal{U}, \tag{4f}$$
$$(x, s, r, w) \geq 0. \tag{4g}$$

(We have introduced a vector w of slack variables for the constraint $x_i \leq u_i$.)

Like all infeasible-primal-dual algorithms, the version of Mehrotra's algorithm implemented by PCx generates a sequence of iterates

$$(x^k, \pi^k, s^k, r^k, w^k), \qquad k = 0, 1, 2, \ldots,$$

that satisfy the strict positivity condition $(x^k, s^k, r^k, w^k) > 0$. However, these points are usually infeasible; that is, the equality conditions (4a),(4b),(4c) are satisfied only in the limit as $k \to \infty$. Compliance with the complementarity conditions (4e),(4f) is measured by the *duality measure* μ, defined by

$$\mu = \frac{\sum_{i=1,\ldots,n} x_i s_i + \sum_{i \in U} w_i r_i}{n + |\mathcal{U}|}. \tag{5}$$

Note that μ is the average value of all the pairwise products $x_i s_i$, $i = 1, 2, \ldots, n$, and $r_i w_i$, $i \in \mathcal{U}$.

For simplicity in describing the algorithm, we assume in the remainder of the paper that *all* primal variables have upper bounds, that is, $\mathcal{U} = \{1, 2, \ldots, n\}$. The primal and dual problems can be stated in this case as

$$\min_{x \in R^n} c^T x \qquad \text{subject to} \quad Ax = b, \;\; 0 \leq x \leq u, \tag{6}$$

and

$$\max_{\pi\in R^m, r\in R^n, s\in R^n} \quad b^T\pi - r^T u \qquad \text{subject to} \quad A^T\pi + s - r = c, \;\; (r,s)\geq 0. \tag{7}$$

The KKT conditions for (6) and (7) are

$$\begin{aligned}
A\pi + s - r &= c, && && (8a)\\
Ax &= b, && && (8b)\\
x + w &= u, && && (8c)\\
x_i s_i &= 0, && i = 1,2,\ldots,n, && (8d)\\
w_i r_i &= 0, && i = 1,2,\ldots,n, && (8e)\\
(x,s,r,w) &\geq 0. && && (8f)
\end{aligned}$$

We stress that the PCx code actually works with the formulation (2); we use the simpler form (6) in our discussion solely to avoid creating a notational jungle in the next few sections.

3 The Algorithm

Mehrotra's predictor-corrector algorithm [8] is based on Newton's method for the KKT conditions (4a)–(4e), modified to retain positivity of the (x,s,r,w) components, to incorporate a "centering" component in the search direction, and to improve the order of accuracy to which the search direction approximates the nonlinear equations (4d) and (4e). We mention just the major elements of the algorithm in this section and the next. For further details and motivation, see Wright [11].

The search direction at each iteration of Mehrotra's algorithm is obtained by solving two systems of linear equations, which have the same coefficient matrix but different right-hand sides. If we assume for simplicity that $\mathcal{U}$ in (2) is the entire index set $\{1,2,\ldots,n\}$, these *step equations* have the form

$$\begin{bmatrix} 0 & A & 0 & 0 & 0 \\ A^T & 0 & I & 0 & -I \\ 0 & I & 0 & I & 0 \\ 0 & S & X & 0 & 0 \\ 0 & 0 & 0 & R & W \end{bmatrix} \begin{bmatrix} \Delta\pi \\ \Delta x \\ \Delta s \\ \Delta w \\ \Delta r \end{bmatrix} = \begin{bmatrix} -r_b \\ -r_c \\ -r_u \\ -r_{xs} \\ -r_{wr} \end{bmatrix}. \tag{9}$$

Here A is the constraint matrix from the linear program, $X = \mathrm{diag}(x)$, $S = \mathrm{diag}(s)$, $W = \mathrm{diag}(w)$, and $R = \mathrm{diag}(r)$. The coefficient matrix is simply the Jacobian of the nonlinear equations defined by (4a)–(4e). The right-hand side for the first system of equations chooses r_u, r_c, and r_b to be the residuals for the upper-bound, dual, and primal infeasibilities, respectively; that is,

$$r_u = x + w - u, \qquad r_c = A^T\pi + s - r - c, \qquad r_b = Ax - b. \tag{10}$$

For the other right-hand side components, this first system uses

$$r_{xs} = XSe, \qquad r_{rw} = RWe, \tag{11}$$

so that the solution $(\Delta x^{\text{aff}}, \Delta\pi^{\text{aff}}, \Delta s^{\text{aff}}, \Delta r^{\text{aff}}, \Delta w^{\text{aff}})$ of this first system is the pure Newton direction for the nonlinear system of equations (4a)–(4e). This direction is often known as the *affine-scaling* direction.

The second direction is a combined centering-corrector direction, which we denote by

$$(\Delta x^{\text{cc}}, \Delta\pi^{\text{cc}}, \Delta s^{\text{cc}}, \Delta r^{\text{cc}}, \Delta w^{\text{cc}}).$$

To obtain this direction, we set the right-hand side components of (9) as follows:

$$r_u = 0, \qquad r_c = 0, \qquad r_b = 0, \tag{12}$$

$$r_{xs} = \Delta X^{\text{aff}} \Delta S^{\text{aff}} e - \sigma\mu e, \qquad r_{rw} = \Delta R^{\text{aff}} \Delta W^{\text{aff}} e - \sigma\mu e, \tag{13}$$

where μ is defined in (5) and ΔX^{aff}, ΔS^{aff}, ΔR^{aff}, and ΔW^{aff} are the diagonal matrices constructed from the affine-scaling step components Δx^{aff}, Δs^{aff}, Δr^{aff}, and Δw^{aff}, respectively. The scalar $\sigma \in [0,1]$ in (12) is chosen by a complicated heuristic that is based on the ability of the pure affine-scaling step to attain large reductions in the duality measure μ before reaching the boundary of the positive orthant for the (x, s, r, w) components. Given the affine-scaling step, we calculate the maximum step to this boundary in primal and dual variables from the definitions

$$\alpha^{\text{aff,P}} = \inf\{\alpha \in [0,1] \mid (x, w) + \alpha(\Delta x^{\text{aff}}, \Delta w^{\text{aff}}) \geq 0\}, \tag{14a}$$

$$\alpha^{\text{aff,D}} = \inf\{\alpha \in [0,1] \mid (s, r) + \alpha(\Delta s^{\text{aff}}, \Delta r^{\text{aff}}) \geq 0\}. \tag{14b}$$

We then compute the duality measure μ^{aff} at this point as

$$\mu^{\text{aff}} = \frac{1}{2n}\left[(x + \alpha^{\text{aff,P}}\Delta x)^T(s + \alpha^{\text{aff,D}}\Delta s) + (w + \alpha^{\text{aff,P}}\Delta w)^T(r + \alpha^{\text{aff,D}}\Delta r)\right]. \tag{15}$$

Finally, the value of σ is chosen to be

$$\sigma = \left(\frac{\mu^{\text{aff}}}{\mu}\right)^3. \tag{16}$$

The actual search direction is obtained by simply adding the affine-scaling direction to the centering-corrector direction; that is,

$$(\Delta x, \Delta\pi, \Delta s, \Delta r, \Delta w) = (\Delta x^{\text{aff}}, \Delta s^{\text{aff}}, \Delta r^{\text{aff}}, \Delta w^{\text{aff}}) + (\Delta x^{\text{cc}}, \Delta\pi^{\text{cc}}, \Delta s^{\text{cc}}, \Delta r^{\text{cc}}, \Delta w^{\text{cc}}). \tag{17}$$

The step taken by the algorithm is then a fraction of the maximum steps $\alpha^{\text{max,P}}$, $\alpha^{\text{max,D}}$ to the boundary in the primal and dual variables, respectively. Similarly to (14), we calculate

$$\alpha^{\text{max,P}} = \inf\{\alpha \in [0,1] \mid (x, w) + \alpha(\Delta x, \Delta w) \geq 0\}, \tag{18a}$$

$$\alpha^{\text{max,D}} = \inf\{\alpha \in [0,1] \mid (s, r) + \alpha(\Delta s, \Delta r) \geq 0\}, \tag{18b}$$

and set

$$\alpha^{\text{P}} = \gamma_{\text{P}} * \alpha^{\text{max,P}}, \qquad \alpha^{\text{D}} = \gamma_{\text{D}} * \alpha^{\text{max,D}}, \tag{19}$$

where γ_{P} and γ_{D} are two scaling factors obtained from Mehrotra's adaptive steplength heuristic [8, p. 588].

Having described all the ingredients, we can summarize the algorithm as follows:

Given $(x^0, \pi^0, s^0, r^0, w^0)$ with $(x^0, s^0, r^0, w^0) > 0$;
for $k = 0, 1, 2, \ldots$
  if termination test is satisfied
    stop;
  Set $(x, \pi, s, r, w) = (x^k, \pi^k, s^k, r^k, w^k)$ and calculate the affine-scaling direction
    from (9) by setting the right-hand side as in (10), (11);
  calculate $\alpha^{\mathrm{aff,P}}$, $\alpha^{\mathrm{aff,D}}$, μ^{aff} and σ from (14), (15), and (16);
  Calculate the centering-corrector step from (9) by setting the right-hand
    side as in (12);
  Calculate the search direction from (17);
  Calculate α^{P}, α^{D} from (18) and (19);
  Calculate new iterate as

$$(x^{k+1}, w^{k+1}) = (x, w) + \alpha^{\mathrm{P}}(\Delta x, \Delta w), \tag{20a}$$
$$(\pi^{k+1}, s^{k+1}, r^{k+1}) = (\pi, s, r) + \alpha^{\mathrm{D}}(\Delta \pi, \Delta s, \Delta r); \tag{20b}$$

end (for).

Gondzio's [4] higher-order correction strategy is used to enhance the search direction at each iteration. In this approach, additional centering/correction directions are computed by solving (9) for different right-hand sides. Rather than attempting to correct the current point to the central path in a single step, Gondzio's strategy is more conservative, aiming only to bring the pairwise products $x_i s_i$, $i = 1, 2, \ldots, n$ and $r_i w_i$, $i \in \mathcal{U}$ that are much larger than the average μ more into line. The number of centering/correction directions depends on the ratio of time required to form and factor the coefficient matrix of the main linear system (see Section 4) to the time required to perform triangular substitutions with the factors. This ratio is machine dependent and therefore leads to different results on different architectures. We refer the interested reader to Gondzio's paper for details. Our implementation draws not only on this paper and also on Gondzio's code HOPDM (version 2.13), in which slightly different heuristics from those described in the paper are used.

Our code applies the scaling technique of Curtis and Reid [2] to the coefficient matrix A before solving. This technique aims to minimize the deviation of the nonzero elements in the matrix from 1, which it measures by the objective function

$$\sum_{A_{ij} \neq 0} \log^2 |A_{ij}|.$$

It finds row and column scaling factors ρ_i, $i = 1, 2, \ldots, m$ and χ_j, $j = 1, 2, \ldots, n$ such that the scaled version of A (whose elements are $A_{ij}/(\rho_i \chi_j)$) minimizes this objective. Conjugate gradient turns out to be very effective when applied to the least squares problem of finding the ρ_i and χ_j factors, and convergence to an approximate solution of adequate accuracy is usually achieved in three or four iterations.

Scaling generally improves the efficiency of the algorithm, but occasionally results in poorer performance. It can be disabled by the user, as we show in Section 6.

The algorithm terminates in one of four states: `optimal`, `infeasible`, `unknown`, and `suboptimal`. `Optimal` termination occurs when the current iterate satisfies the following tests:

$$\begin{aligned}
\frac{\|(r_b, r_u)\|}{1 + \|(b^T, u^T)\|} &\leq \texttt{prifeastol}, \\
\frac{\|r_c\|}{1 + \|c\|} &\leq \texttt{dualfeastol}, \\
\frac{\left|c^T x - \left(b^T \pi - \sum_{i \in \mathcal{U}} u_i r_i\right)\right|}{1 + |c^T x|} &\leq \texttt{opttol},
\end{aligned}$$

where `prifeastol`, `dualfeastol`, and `opttol` are three tolerances whose default values are 10^{-8}, 10^{-8}, and 10^{-8}, respectively.

For the remaining termination conditions, we make use of a merit function ϕ defined by

$$\phi(\pi, x, s, w, r) = \frac{\|(r_b, r_u)\|}{\max(1, \|(b, u)\|)} + \frac{\|r_c\|}{\max(1, \|c\|)} + \frac{\left|c^T x - \left(b^T \pi - \sum_{i \in \mathcal{U}} u_i r_i\right)\right|}{\max(1, \|(b, u)\|, \|c\|)}.$$

Clearly, points (π, x, s, w, r) at which $(x, s, w, r) \geq 0$ and $\phi = 0$ are primal-dual solutions of (6), (7) and vice versa. When applied to feasible linear programs (for which a primal-dual solution is known to exist), ϕ typically decreases steadily to zero after perhaps oscillating during the first few iterations. We also maintain an array $\phi_{\min}$ whose kth element is the smallest value of ϕ encountered up to iteration k; that is,

$$\phi_{\min}[k] = \min_{\ell = 0, 1, \ldots, k} \phi(\pi_\ell, x_\ell, s_\ell, w_\ell, r_\ell).$$

Infeasible problems (that is, problems for which no primal-dual solutions exist) can be detected fairly reliably by a sharp increase in ϕ. We terminate the algorithm at iteration k with status `infeasible` if it fails the optimality test above but satisfies

$$\phi(\pi_k, x_k, s_k, w_k, r_k) \geq \max(10^{-8}, 10^5 \phi_{\min}[k]).$$

In other situations, the code is unable to resolve the question of feasibility. It exhibits slow convergence, or else the improvement in duality measure μ far outstrips the improvement in primal and dual infeasibility ($\|(r_b, r_u)\|$ and $\|r_c\|$, respectively), causing μ to lose its relationship to the true gap between the primal and dual objective function values. In both these cases, we terminate the algorithm with status `unknown`. The slow convergence test is

$$\phi_{\min}[k - 30] \geq \tfrac{1}{2} \phi_{\min}[k] \qquad \text{and } k \geq 30.$$

Blowup in infeasibility-to-duality ratio is flagged if we have

$$\frac{\|(r_b^k, r_u^k)\|}{1 + \|(b^T, u^T)\|} > \texttt{prifeastol} \quad \text{or} \quad \frac{\|r_c^k\|}{1 + \|c\|} > \texttt{dualfeastol},$$

and, in addition,

$$\frac{\max(\|(r_b^k, r_u^k)\|, \|r_c^k\|)/\mu_k}{\max(\|(r_b^0, r_u^0)\|, \|r_c^0\|)/\mu_0} \geq 10^6.$$

Finally, we terminate in `suboptimal` status if the algorithm exceeds its allotted maximum number of iterations (see `iterationlimit` in Section 6) without satisfying any of the conditions above.

4 Linear Algebra

The coefficient matrix in (9) is sparse and highly structured. With the exception of the A and A^T blocks, all blocks are either zero or diagonal. By performing simple block elimination on this system, we obtain the following alternative formulation of the step equations, known as the augmented system form:

$$\begin{bmatrix} -D^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \pi \end{bmatrix} = \begin{bmatrix} -r_c - W^{-1} r_{wr} + X^{-1} r_{xs} + W^{-1} R r_u \\ -r_b \end{bmatrix}, \tag{21}$$

where D is the positive diagonal matrix defined by $D = (S^{-1}X + W^{-1}R)^{1/2}$. The remaining components Δw, Δr, and Δs of the solution vector can be recovered as follows:

$$\begin{aligned} \Delta w &= -r_u - \Delta x \\ \Delta r &= -W^{-1}(R \Delta w + r_{wr}) \\ \Delta s &= -X^{-1}(S \Delta x + r_{xs}). \end{aligned} \tag{22}$$

The system (21) can be reduced to an even more compact form as follows by eliminating Δx to obtain

$$AD^2A^T \Delta \pi = -r_b + AD^2(-r_c - W^{-1} r_{wr} + X^{-1} r_{xs} + W^{-1} R r_u). \tag{23}$$

The component Δx can be recovered from

$$\Delta x = D^2(A^T \Delta \pi - (-r_c - W^{-1} r_{wr} + X^{-1} r_{xs} + W^{-1} R r_u)), \tag{24}$$

while the remaining step components can be obtained as before from (22).

The default version of PCx uses the formulation (23), which is often known as the *normal equations* form. A sparse Cholesky algorithm is used to factor the coefficient matrix AD^2A^T, and the solution $\Delta \pi$ is obtained by performing triangular substitutions with the Cholesky factor L. These factorizations and triangular substitutions dominate the computational cost of the algorithm. The default PCx uses the sparse Cholesky solver of Ng and Peyton [9], modified slightly to handle the small pivot elements that frequently arise during later iterations of the interior-point method. This code produces a factorization of the form

$$P(AD^2A^T)P^T = LL^T, \tag{25}$$

where P is a permutation matrix (determined independently of the numerical values in AD^2A^T during an ordering step) and L is a lower triangular matrix.

Ng and Peyton's code uses a multiple minimum degree ordering strategy identical to the one in SPARSPAK. This strategy was introduced by Liu [6]. The scheme used for symbolic factorization is partly described by Liu [7] and Gilbert, Ng, and Peyton [3]. The numerical factorization is performed by a left-looking block sparse Cholesky algorithm, as described by Ng and Peyton [9]. The code exploits hierarchical memory by splitting the supernodes into blocks that fit into available cache. (Cache size is passed to the code as a parameter.) Loop unrolling is used to make better use of registers. The release of Ng and Peyton's code used here is version 0.4 of May 1995.

Release 1.1 of PCx also contains a link the IBM code WSSMP, described by Gupta, Joshi, and Kumar [5]. This is a multifrontal sparse Cholesky package which uses a nested dissection ordering, and it appears to be especially effective for larger problems. WSSMP is currently supplied in the form of libraries for IBM RS6000 architectures. This is a proprietary code; send mail to Anshul Gupta (gupta@watson.ibm.com) for more information.

Since the nonzero structure of the matrix that we factor is the same at each interior-point iteration, the ordering and symbolic factorization operations are carried out just once, during computation of the initial point. At each interior-point iteration, the numerical factorization is performed once. Two back-substitutions are performed with these computed factors: one for the affine-scaling step, and one for the corrector-centering step.

Our modification of the Ng-Peyton code for small pivots requires just a handful of additional lines of Fortran. A candidate pivot $M_{ii}^{(i-1)}$ is deemed to be "small" if

$$M_{ii}^{(i-1)} \leq 10^{-30} \max_{j=1,2,\cdots,m} M_{jj}^2, \tag{26}$$

where $M^{(i-1)}$ is the remaining submatrix after $i-1$ steps of the Cholesky factorization and M is the original symmetric positive semidefinite matrix. Each small pivot is replaced by the very large number 10^{128}. This substitution causes the off-diagonal elements in the ith column of the Cholesky factor L to be extremely small (essentially zero) and causes the ith component of the solution vector to be extremely small. Analysis of this technique has been performed by Wright [10].

A similar pivot modification strategy is used by the MATLAB-based code LIPSOL (see Zhang [12],[13]), which also uses Ng and Peyton's code as its computational engine.

If the matrix A contains dense columns, the product AD^2A^T may be much denser than A itself, causing the unadorned normal equations strategy to be inefficient. We modify this strategy by excluding the dense columns from the computation of AD^2A^T and accounting for them instead by using the Sherman-Morrison-Woodbury inverse updating formula. At the start of the PCx algorithm, during computation of the initial point, we partition A into "sparse" and "dense" column submatrices A_{sp} and A_{den}, respectively. The diagonal weighting matrix D can be partitioned accordingly into D_{sp} and D_{den}, so we can write

$$AD^2A^T = A_{\mathrm{sp}} D_{\mathrm{sp}}^2 A_{\mathrm{sp}}^T + A_{\mathrm{den}} D_{\mathrm{den}}^2 A_{\mathrm{den}}^T = M + A_{\mathrm{den}} D_{\mathrm{den}}^2 A_{\mathrm{den}}^T, \tag{27}$$

where we have defined M in an obvious way. By applying the Sherman-Morrison-Woodbury formula to (27), we find that

$$[M + A_{\mathrm{den}} D_{\mathrm{den}}^2 A_{\mathrm{den}}^T]^{-1}$$

$$= \quad M^{-1} - (M^{-1}A_{\rm den})\left[D_{\rm den}^{-2} + A_{\rm den}^T M^{-1} A_{\rm den}\right]^{-1} A_{\rm den}^T M^{-1}. \tag{28}$$

We apply the sparse Cholesky procedure to M alone, to obtain

$$PMP^T = LL^T, \tag{29}$$

(cf. (25)). The solution of a linear system with coefficient matrix M and right-hand side r can now be written as

$$\begin{aligned}&(AD^2A^T)^{-1}r \\ &= \quad P^TL^{-T}\left\{I - L^{-1}PA_{\rm den}\left[D_{\rm den}^{-2} + A_{\rm den}^T P^T L^{-T} L^{-1} P A_{\rm den}\right]^{-1} A_{\rm den}^T P^T L^{-T}\right\} L^{-1}Pr.\end{aligned}$$

Given L and P, the major costs of applying this formula are the cost of computing $(L^{-1}PA_{\rm den})$, the cost of a triangular substitution with L and one with L^T—a total cost of $n_{\rm den}+2$ triangular substitutions, where $n_{\rm den}$ is the number of columns in $A_{\rm den}$. For additional systems with the same coefficient matrix but different right-hand sides, the marginal cost is just two triangular substitutions.

To determine which columns are to be classified as "dense," we sort in decreasing order an array whose components are the number of nonzeros in each column. We then look at the columns for which the proportion of nonzeros is at least τ, where $\tau = 1$ for $m < 500$, $\tau = 0.1$ for $500 < m \leq 2000$, and $\tau = 0.05$ for $m > 2000$, and try to identify a gap in the sequence of nonzero counts. (In our experience, most problems that benefit from special handling of the dense columns exhibit such a gap.) Columns whose nonzero counts lie on the high side of the gap are classified as "dense."

Another feature of PCx version 1.1 is the use of a preconditioned conjugate gradient (PCG) algorithm to improve the accuracy of computed solutions for the linear system (23). Essentially, we use the computed Cholesky factorization (25) (or (29)) as the preconditioner and treat the computed solution as the first iteration of a PCG algorithm. The PCG algorithm is activated if the computed solution fails to reduce the residual by a factor `primalfeastol` or better (see Section 6), and if no small pivot modifications are required during the Cholesky factorization. If dense columns are detected in A, PCG terminates when the residual reduction factor `primalfeastol` is achieved, or after a maximum of $10n_{\rm den}$ PCG iterations, whichever comes first. If no dense columns are present in A, at most 10 PCG iterations are allowed.

5 The Presolver

Linear programming models frequently contain redundant information, as well as other information and structure that allows some components of the solution to be determined without recourse to a sophisticated algorithm. The purpose of *presolve* or *preprocessing* routines is to detect and handle these features of the input, producing a (smaller) problem to be solved by the actual linear programming algorithm. Presolvers significantly enhance the efficiency and robustness of both simplex and interior-point codes.

The presolver in PCx works with the formulation (1) stored in the `LPtype` data structure. It makes use of techniques described by Andersen and Andersen [1], checking the data for the following features:

Infeasibility. Check that $u_i \geq 0$ for each upper bound u_i, $i \in \mathcal{U}$, and that a zero row of A has a corresponding zero in the right-hand side vector b.

Empty Rows. If the matrix A has a zero row and a corresponding zero in the b, it can be removed from the problem.

Duplicate Rows. When a row of A (and the corresponding element of the right-hand side b) is simply a multiple of another row, we can delete it without affecting the primal solution.

Duplicate Columns. When a column of A is a multiple of another column, and if the two variables x_i and x_j are "normal" (that is, $i, j \in \mathcal{N}$ in the formulation (2)), the two columns can be combined. The primal variable for the combined column is either normal or free, depending on whether the columns are positive or negative multiples of each other.

Empty Columns. The corresponding element x_i can be fixed at either its lower or upper bound, depending on the sign of the cost vector coefficient c_i. If the required bound does not exist, the problem is declared to be primal unbounded.

Fixed Variables. If the variable has lower and upper bounds both zero, it can obviously be fixed at zero and removed from the problem.

Singleton Rows. If the ith row of A contains the single nonzero element A_{ij}, we clearly have $x_j = b_i/A_{ij}$, so this variable can be removed from the problem. The ith row of A (and hence the dual variable π_i) can also be removed.

Singleton Columns. When A_{ij} is the only nonzero in column j of A, and x_j is a free variable, we can express x_j in terms of the other variables represented in row i of A and eliminate it from the problem. Even if not free, x_j can be eliminated if its bounds are weaker than those implied by the ranges of the other elements represented in the row $A_{i\cdot}$.

Forced Rows. Sometimes, the linear constraint represented by row i of A forces all its variables to either their upper or lower bounds. An example would be the constraint $10x_3 - 4x_{10} + x_{12} = -4$ subject to the bounds

$$x_3 \in [0, +\infty), \qquad x_{10} \in [0, 1], \qquad x_{12} \in [0, +\infty).$$

In this case, we must have $x_3 = 0$, $x_{10} = 1$ and $x_{12} = 0$, so these three variables (and the corresponding row of A) can be eliminated.

The presolver makes multiple passes through the data, checking for each of the above features in turn. Problem reductions on one pass frequently uncover further reductions that are detected on subsequent passes. The presolver terminates when a complete pass is performed without detecting further opportunities for reduction. Each reduction operation is pushed onto a stack, which is subsequently popped after the solution of the reduced linear program is found. The effect of popping the stack is to express the solution in terms of the original, unreduced formulation.

Despite the complexity of the code, the presolver requires little CPU time in comparison with a single iteration of the interior-point solver.

Code for the presolver can be found in the file `presolve.c`. The data structures are defined in `pre.h`. This code can be used on a stand-alone basis, independently of the PCx solver, to presolve any linear program supplied in the `LPtype` format.

6 Specifications File

PCx allows many algorithmic parameters and options to be set by the user. These quantities are stored internally in a data structure of type `Parameters`.

If the user provides input to PCx via an MPS file (rather than invoking `PCx()` directly via a subroutine call), the `Parameters` data structure is allocated automatically by the program and default values are assigned to all parameters. You can override the default values by defining a specifications file, which contains a number of keywords and numerical values.

PCx searches for the specifications file in a number of locations. If the name of the MPS input file is `probname.mps`, PCx looks for the following files, in order:

`probname.spc`, `probname.specs`, `spc`, `specs`, `PCx.specs`

If more than one of these files exist, PCx uses the first file in the list above and ignores the others.

The following is a list of keywords that can be used in the specifications file, together with their default settings. The file should contain one such keyword per line, together with its corresponding numerical value or option, if appropriate. The file is processed sequentially from top to bottom, so the effect of any line in the file can be undone by a later line. For keywords with a `yes/no` argument, omission of the argument will be taken to mean `yes`. (The default setting is not necessarily `yes`.) In the descriptions below, we assume that PCx is invoked with the command

```
PCx probname
```

`boundname` {`name`} Request the bound to be the specific column `name` in `probname.mps`. Default: the first BOUND in the MPS file is used.

`cachesize` {`value`} Input the size of the cache on the machine, in Kilobytes. Any value in the range 0–2048 is acceptable. Specify 0 for Cray machines. This parameter is used by the Ng-Peyton sparse Cholesky code. Default: 16.

`centerexp` {`value`} Specify the exponent to be used for calculation of the centering parameter σ in (16). Any real value in the range 1.0–4.0 is allowable. Default: 3.0.

`dualfeastol` {`value`} Specify a dual feasibility tolerance. Default: 10^{-8}.

`history` {`yes`}/{`no`} Request that a history file be written (`yes`) or not written (`no`). If `yes`, the file `probname.log` is written to the working directory (see Section 8).

`HOCorrections` {`yes`}/{`no`} Request that Gondzio's [4] higher-order corrections be used to enhance the search direction. Default: `yes`.

`inputdirectory` {`name`} If PCx is to search for the MPS input files in another directory, in addition to the current working directory, name this other directory here. Remember to include a trailing "`/`". PCx always looks first in the current working directory. If it cannot find the file there, it looks in the specified input directory. The output and history files always are written to the working directory.

`iterationlimit` {`value`} An upper limit on the number of iterations. Any positive integer is allowable. Default: 100.

`max` Maximize the objective.

`MaxCorrections` {`value`} If `HOCorrections = yes`, the parameter `MaxCorrections` is an upper limit on the number of Gondzio's higher-order corrections allowed at each iteration. If `value=0`, the maximum is determined automatically by PCx according to the relative cost of factorization and solve operations. If `HOCorrections = no`, `MaxCorrections` is ignored. Default: 0.

`min` Minimize the objective (default).

`objectivename` {`name`} Request the objective cost vector to be the specific row `name` in `probname.mps`. Default: the first row of type "N" in `probname.mps` is taken to be the objective.

`opttol` {`value`} Specify an optimality tolerance. Default: 10^{-8}.

`preprocess` {`yes`}/{`no`} Synonymous with `presolve`.

`presolve` {`yes`}/{`no`} Request that presolving be performed (`yes`) or not performed (`no`) (see Section 5). Default: `yes`.

`prifeastol` {`value`} Specify a primal feasibility tolerance. Default: 10^{-8}.

`rangename` {`name`} Request the range to be the specific column `name` in `probname.mps`. Default: the first range encountered in the MPS file is used.

`refinement` {`yes`}/{`no`} Perform preconditioned conjugate gradient refinement of the computed solution to the linear system (23) if it has a relative residual larger than the parameter `prifeastol` (`yes`) or don't perform any iterative refinement (`no`) (see Section 4). Default: `no`.

`rhsname` {`name`} Request the right-hand side to be the specific column `name` in `probname.mps`. Default: the first RHS encountered in the MPS file is used.

`scaling` {`yes`}/{`no`} If `yes`, row and column scaling is performed on the constraint matrix. Default: `yes`.

`solution` {`yes`}/{`no`} Request that a solution file be written (`yes`) or not written (`no`). If the solution file is written, it is named `probname.out` and is placed in the working directory (see Section 8). Default: `yes`.

`stepfactor` {`value`} Specify a value in the range $(0, 1)$ that is used in Mehrotra's adaptive steplength heuristic from [8, p. 118]. This value is a lower bound for γ^{P} and γ^{D} in (19). Default: 0.9.

`unrollinglevel` {`value`} Specify the level of loop unrolling. Allowable values are 1, 2, 4, and 8. (This parameter is used only in the Ng-Peyton sparse Cholesky code.) Default: 4.

If you call `PCx()` directly from your own code, you must fill out the `Parameters` data structure explicitly. This task is easier if you use the routine `*NewParameters()` to allocate the storage, since this routine assigns default values to all parameters. You can then make any desired alterations before passing the data structure to the `PCx()` routine.

7 Obtaining and Installing PCx

PCx contains material protectable under copyright laws of the United States. Permission is hereby granted to use, reproduce, prepare derivative works, and redistribute to others at no charge, provided that any changes are clearly documented and that the original PCx copyright notice, Government license and disclaimer are retained; however, any entity desiring permission to incorporate this software, or a work based on this software, into a product for sale must contact Paul Betten at the Industrial Technology Development Center, Argonne National Laboratory, Argonne, IL 60439 (phone: 630/252-4962, fax: 630/252-5230, email: betten@anl.gov). For further information, refer to the copyright notice on the software.

The source code and documentation for PCx can be obtained through the World Wide Web and anonymous ftp. The PCx home page is

`http://www.mcs.anl.gov/otc/Tools/PCx/`

This page lists the Unix systems on which PCx has been compiled and tested, and also contains the copyright statement. The PCx home page also links to the following three files:

`PCx.tar.gz`: A gzipped tar file containing the source code, a Makefile, and a README file containing installation instructions. It also contains a postscript version of this user guide.

`PCx-user.ps`: A postscript version of this user guide.

`results.ps`: The tables of computational results from Section 10 of this guide.

Executables for PCx for the SunOS, Solaris, IBM RS/6000 AIX, and SGI IRIX Unix environments can be built from source via the following procedure. Download the file `PCx.tar.gz` and place it in its own subdirectory (referred to henceforth as the "working directory"). From the working directory, unzip the file by typing

```
gunzip PCx.tar.gz
```
[1]

and then un-tar the resulting file by typing

```
tar xvf PCx.tar
```

The subdirectories `SRC/`, `DOC/`, `MAKEARCH/`, `Ng-Peyton/`, `mps/` will be created by the `tar` command. A sample specifications file named `PCx.specs` and a number of executable script files will also appear. To create the executable `PCx` that uses the default Ng-Peyton solver, type

```
build
```

Because of architectural and environmental differences, it is necessary to have a slightly different compilation procedure for each machine. The `build` script defines an environment variable `PCx_ARCH` and assigns it a value to indicate the architecture. `build` then invokes the `make` procedure, with architecture-dependent portions of the makefile being retrieved from the subdirectory `MAKEARCH/`. Since the variable `PCx_ARCH` must be defined for compiling, one should always use `build` instead of `make` to compile the program.

Executables are also available for the SunOS, Solaris, AIX, and IRIX systems. The PCx Web page also contains links to these files.

To test PCx it on one of the input files in the directory `mps/`, modify the sample specifications file `PCx.specs` if desired, then type

```
PCx afiro
```

or

```
PCx 25fv47
```

The program and documentation files can also be retrieved via anonymous ftp. Go to `ftp.mcs.anl.gov` and `cd` to `pub/neos/PCx`. The files mentioned above can be found at:

```
ftp://ftp.mcs.anl.gov/pub/neos/PCx/PCx.tar.gz
ftp://ftp.mcs.anl.gov/pub/neos/PCx/PCx-user.ps
ftp://ftp.mcs.anl.gov/pub/neos/PCx/results/results.ps
```

The executables can be found at the following URLs:

`ftp://ftp.mcs.anl.gov/pub/neos/PCx/sun4/PCx.gz` (SunOS)
`ftp://ftp.mcs.anl.gov/pub/neos/PCx/solaris/PCx.gz` (Solaris)
`ftp://ftp.mcs.anl.gov/pub/neos/PCx/rs6000/PCx.gz` (RS/6000 AIX)
`ftp://ftp.mcs.anl.gov/pub/neos/PCx/irix/PCx.gz` (SGI IRIX)
`ftp://ftp.mcs.anl.gov/pub/neos/PCx/irix64/PCx.gz` (SGI IRIX6.4)
`ftp://ftp.mcs.anl.gov/pub/neos/PCx/hp/PCx.gz` (Hewlett-Packard HPUX)
`ftp://ftp.mcs.anl.gov/pub/neos/PCx/linux/PCx.gz` (PC Linux)
`ftp://ftp.mcs.anl.gov/pub/neos/PCx/alpha/PCx.gz` (DEC Alpha)

[1] gunzip can be downloaded from `ftp://quest.jpl.nasa.gov/pub/` for compilation on a variety of architectures.

These executables can be gunzip-ed as described above to produce an executable named `PCx`.

The transfer mode should be set to `binary` by using the `bin` command in ftp before attempting to transfer `PCx.tar.gz` or any of the executable files.

If the WSSMP library [5] can be obtained, an executable `PCx` that calls this library can be created by placing the library in a directory called `./wssmp` and typing

```
build PCx_wssmp
```

8 Invoking PCx

By downloading and installing PCx on one's system (see Section 7), the user will have an executable `PCx`, a `Makefile` and a `build` script in the current working directory, together with a number of subdirectories containing source files for PCx, object files, documentation, and source files and a library for the Ng-Peyton sparse Cholesky code.

To solve a linear program contained in the MPS file `probname.mps`, one should go to the working directory (that is, the directory in which the executable `PCx` resides) and type

```
PCx probname
```

The file `probname.mps` can reside either in the working directory or in an "input directory" defined in the specifications file (see Section 6). PCx first searches the input directory (if specified) for the given file. It searches for the file name both with and without the `.mps` extension. If it does not find the file in the input directory, it searches the working directory.

PCx optionally produces two output files named `probname.out` and `probname.log`, according to the options supplied by the user in the specifications file (see Section 6). These files are written in the working directory. They contain, respectively, the primal-dual point returned by the algorithm (provided the termination status is not `infeasible`), and a summary of iteration history, timings, preprocessor results, and sparsity statistics for the Cholesky factorization. Output is also written to standard output during execution of PCx. Essentially, the on-screen output consists of the information written to the file `probname.log`, together with error messages and warnings.

When PCx is executed as a standalone system and a runtime error is detected, the code returns a nonzero integer to the operating system. The return status indicates the type of error, as follows:

`1`: invocation error for `PCx`;

`2`: memory allocation error (usually, insufficient storage available);

`3`: error in the MPS input file;

`4`: error in the specifications file;

`5`: error detected during presolve; or

`6`: error encountered during matrix factorization, conjugate gradient iteration, sparse matrix multiplication, or dense column linear algebra.

The subroutine `PCx()` can also be invoked directly from user-written code. In this case, the user should fill out data structures that define the linear program and the algorithmic parameters. See the source code and the comments therein for details of this mode of use.

9 Interfacing with the Linear Equation Solver

This section deals with more advanced issues for users who want to experiment with different solvers for the system of linear equations that arises at each iteration. There is no need to read this part if you are content to run PCx with the default Ng-Peyton solver. If, however, you would like to try another linear equations solver, this section describes briefly the C code you need to write to make the connection between the linear equations solver and the main body of PCx.

The C header file `solver.h` defines the interface between the PCx code proper and the linear equations solver. This file names the solver-specific routines for storage allocation and for performing ordering, factorization, and solve operations. The code for these routines actually appears in another user-supplied file (see next paragraph). The names of the routines are listed separately in `solver.h` to make these routines callable from other parts of PCx.

The main requirement on the user is to provide a C file called `mysolver.c` (in the directory `./SRC`) that implements the routines listed in `solver.h`, together with any auxiliary routines that they may call. The best way to prepare this file is to examine the two templates provided with the PCx distribution: the file `Ng-Peyton.c`, which defines the link to the Ng-Peyton solver, and `wssmp.c`, which defines the link to the WSSMP library. All the relevant data is passed into the routines via the data structure `Factor` of type `FactorType`, which is defined in the header file `main.h`. This data structure contains a void pointer `ptr` that can be used to point to solver-specific information and data solver. In addition, `Factor` contains the matrix AD^2A^T, stored in the usual compressed-sparse-row (CSR) format. (Note that PCx uses Fortran-style indexing, in which row and column indices start at 1 rather than zero.)

To be a little more specific, the functions that need to be implemented in `mysolver.c` are simple memory allocation and deallocation functions for the data structure `FactorType`, the function `Order()`, which performs the symbolic factorization of the coefficient matrix, the function `Factorize()`, which performs the numerical factorization, and a function `Solve()` that uses the factorization to obtain the solution of the linear system for a given right-hand side.

The other major component to be supplied by the user is a library called `libmysolver.a` in the directory `./mysolver`. The `build` script and the `Makefile` in directory `./SRC` assume that this library is present. If desired, the `build` script can be altered so that it creates this library explicitly from a collection of source files, as is done already for the Ng-Peyton solver.

When all the files above are in place, an executable `PCx` that calls the user-supplied solver can be created by typing

```
build PCx_mysolver
```

Recall that if a sparse Cholesky technique is being used in the user-supplied solver, it will need to contain modifications to handle tiny and negative pivots, similar to the modifications described in Section 4.

To enable performance monitoring for the user-supplied solver and for PCx, uncomment the line that defines the environment variable `TIMING_PROFILE` at the start of the file `main.h`. When this variable is defined, the log file produced by the PCx run will contain detailed information about how much time was spent in different parts of the code.

10 Computational Results

We have executed PCx version 1.1 successfully in a variety of Unix environments, including

IBM RS6000/370 workstation running AIX, with 128 MB main memory and 350 MB swap space, running AIX;

Sun SPARCstation-10 running SunOS4.3, with 32 MB main memory;

Sun UltraSparc 2 running Solaris 2.x, with 200 MHz processor, 1 MB cache and 256 MB of main memory;

SGI workstation running IRIX 5.3, with 250 MHz processor, 2 MB L2-cache, 64 MB main memory.

SGI workstation running IRIX 6.4, with 195 MHz IP27 processor, 4 MB L2-cache, 4 GB main memory.

HP9000-735 workstation running HPUX-9.05, with 128 MB main memory and 125 MHz PA7150 chip.

Pentium Pro PC running PC Linux, with 48 MB main memory.

We report results from the SGI machine running IRIX 6.4. On this machine, the code was compiled with the default Fortran and C compilers (`xlf` and `cc`, respectively), using the `-O` optimization flag in both cases.

We solved a large set of test problems, both feasible and infeasible, taken for the most part from the familiar netlib set. Results obtained with the default parameter settings are shown in Tables 1–3. Each row in the tables contains the dimensions of the problem before and after presolving, measures of infeasibility and complementarity, the primal objective of the point returned by PCx, the maximum number of additional centering/corrector steps allowed at each iteration (over and above the single centering/corrector step of Mehrotra's algorithm), the number of iterations, and the CPU time. The tabulated infeasibility measure is a relative measure defined as

$$\max\left(\frac{\|(r_b, r_u)\|}{1+\|(b,u)\|}, \frac{\|r_c\|}{1+\|c\|}\right),$$

where r_b, r_u and r_c are the residuals at the final point. The tabulated complementarity measure is defined as

$$\frac{x^T s + (u-x)^T r}{1+|c^T x|}.$$

Results for the feasible problems are shown in Table 1. In most cases, PCx correctly identified the problem as feasible and returned an optimal solution. In four cases, the code terminated with status `unknown`, though in three of these cases the point returned by the code is quite close to optimality. No problems were incorrectly flagged as `infeasible`.

Results for the infeasible problems appear in Tables 2. In two cases, PCx terminates with status `unknown`; the correct status `infeasible` is reported for all other problems. In two other cases, infeasibility was detected by the preprocessor, so the interior-point solver did not need to be called at all.

The NEMS problems are instances of models in the National Energy Modeling System (NEMS) of the Energy Information Administration of the United States Department of Energy [14]. These problems are taken from NEMS modules which are used to model electricity capacity planning, petroleum marketing, and coal marketing. PCx solved these problems efficiently, as shown in Tables 3.

The improvements obtained by using higher-order corrections are not too dramatic. Part of the reason is that the factorization routine is more efficient relative to the solution routine than is the case in, for example, HOPDM (see Gondzio [4]). It follows that there is less to be gained by economizing on matrix factorizations. Significant improvements can however be observed on several problems, including `dfl001`, `pds-10`, `NEMSemm1`, and `NEMSwrld`.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. We acknowledge the contribution of Marc Wenzel, who programmed the dense-column-handling and conjugate gradient refinement features that were added for the beta-2.0 release. We are grateful also to Doug Moore for various advice and for pointing out and repairing many memory leaks in earlier releases, and to Hans Mittelmann for running many of the benchmark tests and for compiling on various architectures, including DEC Alpha and Linux. Thanks also to Erling Andersen for helpful advice and comments, and to Anshul Gupta for helping with the interface to his WSSMP code.

Computational results for the NETLIB test set and the NEMS problems on an R10000 SGI workstation (195 MHz IP27 processor, L2-cache 4MB, Main memory 4GB, running IRIX 6.4)

Legend: * = terminated with `unknown` status,
† = infeasibility detected during preprocessing,
$_{max}$ = maximization problems

Table 1: NETLIB: Feasible NETLIB problems

Name	Before Preprocessing Rows	Before Preprocessing Cols	After Preprocessing Rows	After Preprocessing Cols	Relative Infeas	Relative Compl	Primal Objective	Max Add. Corr.	Iters	CPU Time [secs]
25fv47	821	1876	788	1843	1.8e-11	1.3e-07	5.501846e+03	1	22	1.47
80bau3b	2262	12061	2140	11066	1.1e-11	7.6e-07	9.872244e+05	0	37	4.47
NL	7039	15325	6665	14680	1.8e-12	9.9e-07	1.229265e+06	1	35	29.27
adlittle	56	138	55	137	8.2e-14	3.4e-11	2.254950e+05	0	12	0.02
afiro	27	51	27	51	1.9e-11	7.2e-15	-4.647531e+02	0	8	0.01
agg	488	615	390	477	3.0e-09	6.2e-11	-3.599177e+07	1	17	0.36
agg2	516	758	514	750	3.4e-11	4.8e-10	-2.023925e+07	1	20	0.75
agg3	516	758	514	750	4.5e-11	3.1e-11	1.031212e+07	1	19	0.72
bandm	305	472	240	395	1.3e-12	1.5e-08	-1.586280e+02	1	14	0.15
beaconfd	173	295	86	171	3.7e-12	2.4e-13	3.359249e+04	1	10	0.06
blend	74	114	71	111	4.6e-13	5.7e-15	-3.081215e+01	0	10	0.02
bnl1	643	1586	610	1491	1.2e-11	3.2e-08	1.977630e+03	0	39	0.96
bnl2	2324	4486	1964	4008	3.1e-09	8.2e-08	1.811237e+03	1	31	5.94
boeing1	351	726	331	697	5.0e-09	2.6e-09	-3.352136e+02	1	18	0.32
boeing2	166	305	126	265	2.1e-08	1.1e-09	-3.150187e+02	0	14	0.07
bore3d	233	334	81	138	4.2e-14	4.8e-13	1.373080e+03	0	16	0.04
brandy	220	303	133	238	1.0e-05	6.1e-15	1.518510e+03	1	16	0.15
capri	271	482	241	436	1.8e-10	2.4e-12	2.690013e+03	0	19	0.17
cycle	1903	3371	1420	2773	7.6e-09	2.3e-12	-5.226393e+00	1	21	2.32
czprob	929	3562	671	2779	3.7e-10	1.1e-07	2.185197e+06	0	26	0.57
d2q06c	2171	5831	2132	5728	4.8e-08	2.8e-07	1.227842e+05	1	24	9.08
d6cube	415	6184	403	5443	2.4e-09	4.5e-09	3.154917e+02	1	16	3.10
degen2	444	757	444	757	7.4e-14	8.4e-13	-1.435178e+03	1	11	0.51
degen3	1503	2604	1503	2604	3.6e-10	1.8e-09	-9.872940e+02	2	14	8.37
dfl001	6071	12230	5984	12143	1.5e-10	2.4e-07	1.126640e+07	4	39	707.42
e226	223	472	198	429	9.6e-12	1.8e-08	-2.586493e+01	1	16	0.18
etamacro	400	816	334	669	1.4e-14	3.1e-08	-7.557152e+02	0	25	0.42
fffff800	524	1028	322	826	5.8e-10	7.4e-08	5.556796e+05	1	25	0.68
finnis	497	1064	438	935	4.2e-12	1.1e-08	1.727911e+05	0	25	0.39
fit1d	24	1049	24	1049	9.0e-14	1.7e-07	-9.146377e+03	1	17	0.60
fit1p	627	1677	627	1677	3.3e-09	4.6e-08	9.146378e+03	0	17	0.58
fit2d	25	10524	25	10524	4.7e-14	2.4e-08	-6.846429e+04	1	23	7.58
fit2p	3000	13525	3000	13525	4.4e-08	1.8e-06	6.846441e+04	0	19	4.21
forplan	161	492	121	447	4.1e-08	3.9e-10	-6.642190e+02	1	20	0.35
ganges	1309	1706	1113	1510	4.4e-07	9.7e-09	-1.095857e+05	0	17	0.73
gfrd-pnc	616	1160	590	1134	9.7e-15	2.2e-11	6.902236e+06	0	18	0.20
greenbea *	2392	5598	1933	4164	1.6e-03	4.8e-07	-7.255534e+07	1	43	5.96
greenbeb *	2392	5598	1932	4154	1.4e-05	1.7e-10	-4.302260e+06	1	37	4.65
grow15	300	645	300	645	3.5e-07	8.4e-15	-1.068709e+08	1	21	0.55
grow22	440	946	440	946	4.1e-05	2.1e-10	-1.608343e+08	1	22	0.89
grow7	140	301	140	301	2.9e-09	1.6e-09	-4.778781e+07	1	17	0.21
israel	174	316	174	316	1.4e-12	1.3e-08	-8.966448e+05	1	19	0.52
kb2	43	68	43	68	3.0e-10	2.0e-16	-1.749900e+03	0	13	0.02
lotfi	153	366	133	346	7.4e-10	1.0e-15	-2.526471e+01	0	15	0.06

	Before		After					Max		CPU
	Preprocessing		Preprocessing		Relative	Relative	Primal	Add.		Time
Name	Rows	Cols	Rows	Cols	Infeas	Compl	Objective	Corr.	Iters	[secs]
maros-r7	3136	9408	2152	7440	2.3e-11	1.5e-11	1.497185e+06	2	14	30.10
maros	846	1966	655	1437	2.8e-08	7.1e-11	-5.806374e+04	0	20	0.57
mod2 *	34774	66409	28760	56347	1.1e-05	9.1e-05	4.557503e+07	1	57	170.45
modszk1	687	1620	665	1599	7.4e-09	1.6e-12	3.206196e+02	0	22	0.49
nesm	662	3105	654	2922	1.4e-09	2.2e-07	1.407604e+07	1	25	1.58
pds-10	16558	49932	15648	48780	2.6e-10	3.5e-06	2.672717e+10	3	35	557.58
perold	625	1506	593	1374	6.9e-07	4.4e-08	-9.380755e+03	1	31	1.28
pilot.ja	940	2267	810	1804	4.1e-05	3.0e-08	-6.113136e+03	1	29	3.35
pilot	1441	4860	1368	4543	3.4e-07	2.5e-07	-5.574897e+02	2	31	23.20
pilot.we	722	2928	701	2814	1.5e-11	1.2e-07	-2.720107e+06	0	46	1.58
pilot4	410	1123	396	1022	4.1e-05	5.1e-09	-2.581139e+03	1	46	1.91
pilot87	2030	6680	1971	6373	5.2e-07	6.6e-07	3.017105e+02	3	30	75.92
pilotnov	975	2446	848	2117	4.0e-06	1.3e-11	-4.497276e+03	1	16	1.67
radex	16	26	15	25	3.7e-13	6.2e-14	3.584229e+05	0	8	0.00
recipe	91	204	64	123	1.9e-10	3.9e-16	-2.666160e+02	0	9	0.02
sc105	105	163	104	162	8.7e-09	1.5e-16	-5.220206e+01	0	10	0.02
sc205	205	317	203	315	1.1e-11	1.1e-13	-5.220206e+01	0	11	0.07
sc50a	50	78	49	77	2.4e-11	9.5e-16	-6.457508e+01	0	8	0.01
sc50b	50	78	48	76	2.8e-09	7.4e-11	-7.000000e+01	0	6	0.01
scagr25	471	671	469	669	2.9e-12	2.2e-13	-1.475343e+07	0	18	0.18
scagr7	129	185	127	183	1.8e-13	7.9e-09	-2.331390e+06	0	14	0.04
scfxm1	330	600	305	568	1.2e-06	1.6e-09	1.841676e+04	0	17	0.19
scfxm2	660	1200	610	1136	2.2e-08	5.3e-13	3.666026e+04	0	20	0.43
scfxm3	990	1800	915	1704	5.1e-08	3.6e-12	5.490125e+04	0	20	0.64
scorpion	388	466	340	412	5.0e-14	6.2e-16	1.878125e+03	0	12	0.09
scrs8	490	1275	421	1199	6.9e-13	1.1e-08	9.042970e+02	0	22	0.27
scsd1	77	760	77	760	7.3e-15	2.1e-15	8.666667e+00	0	9	0.07
scsd6	147	1350	147	1350	1.1e-14	2.8e-09	5.050000e+01	0	12	0.15
scsd8	397	2750	397	2750	6.4e-15	9.2e-08	9.050001e+02	0	11	0.27
sctap1	300	660	284	644	1.0e-13	9.7e-14	1.412250e+03	0	16	0.12
sctap2	1090	2500	1033	2443	2.3e-15	4.0e-15	1.724807e+03	0	14	0.44
sctap3	1480	3340	1408	3268	7.0e-16	1.3e-13	1.424000e+03	0	15	0.67
seba	515	1036	448	901	2.4e-13	6.8e-09	1.571160e+04	2	12	2.60
share1b	117	253	112	248	2.6e-08	3.4e-10	-7.658932e+04	0	19	0.08
share2b	96	162	96	162	2.1e-11	1.6e-14	-4.157322e+02	0	17	0.05
shell	536	1777	487	1451	3.5e-13	1.6e-11	1.208825e+09	0	21	0.28
ship04l	402	2166	292	1905	5.8e-14	2.8e-13	1.793325e+06	0	13	0.20
ship04s	402	1506	216	1281	1.0e-11	7.0e-09	1.798715e+06	0	13	0.13
ship08l	778	4363	470	3121	5.7e-14	8.9e-11	1.909055e+06	0	16	0.39
ship08s	778	2467	276	1604	3.1e-13	3.0e-11	1.920098e+06	0	12	0.16
ship12l	1151	5533	610	4171	4.3e-14	2.0e-07	1.470188e+06	0	16	0.51
ship12s	1151	2869	340	1943	5.0e-14	3.6e-13	1.489236e+06	0	13	0.21
sierra	1227	2735	1212	2705	4.8e-16	8.0e-10	1.539436e+07	0	21	0.79
stair	356	614	356	532	1.3e-09	2.6e-08	-2.512670e+02	1	13	0.41
standata	359	1274	314	796	6.7e-15	5.4e-09	1.257700e+03	0	13	0.10
standgub	361	1383	314	796	6.7e-15	5.4e-09	1.257700e+03	0	13	0.10
standmps	467	1274	422	1192	2.9e-14	3.2e-15	1.406017e+03	0	26	0.31
stocfor1	117	165	102	150	4.5e-13	3.6e-11	-4.113198e+04	0	12	0.03
stocfor2	2157	3045	1980	2868	8.4e-11	1.0e-07	-3.902441e+04	0	20	0.95
stocfor3	16675	23541	15362	22228	1.8e-09	6.1e-08	-3.997678e+04	0	31	12.80
truss	1000	8806	1000	8806	1.5e-13	1.8e-09	4.588158e+05	0	20	2.24
tuff	333	628	257	567	5.5e-09	5.5e-08	2.921478e-01	1	18	0.32
vtp.base	198	346	72	111	1.5e-08	2.6e-09	1.298315e+05	0	11	0.02
wood1p *	244	2595	171	1718	1.8e-05	3.8e-05	1.442944e+00	2	21	3.35
woodw	1098	8418	708	5364	6.2e-11	1.2e-10	1.304476e+00	0	31	2.24
world *	34506	67147	28652	58027	2.2e-02	3.2e-04	7.214980e+07	1	61	181.65

Table 2: NETLIB: Infeasible NETLIB problems

Name	Before Preprocessing Rows	Before Preprocessing Cols	After Preprocessing Rows	After Preprocessing Cols	Relative Infeas	Relative Compl	Primal Objective	Max Add. Corr.	Iters	CPU Time [secs]
bgdbg1	348	629	249	509	1.8e+02	4.0e+00	4.155802e+01	0	6	0.05
bgetam	400	816	334	669	8.4e+01	1.2e+01	-3.571285e+04	0	7	0.14
bgindy	2671	10880	2657	10866	4.6e+01	2.6e+00	1.059302e+09	1	8	13.29
bgprtr	20	40	20	40	1.9e-01	3.3e-01	8.008869e+06	0	6	0.00
box1	231	261	231	261	5.9e-02	1.0e+00	5.775809e+02	0	4	0.02
ceria3d	3576	4400	3576	4400	8.0e-02	7.4e-02	-9.975419e-01	1	12	4.77
chemcom	288	744	288	744	4.9e+02	1.9e+00	3.908033e+05	0	8	0.09
cplex1	3005	5224	3005	5224	5.0e+07	9.2e+00	-2.701093e+09	0	5	0.44
cplex2 *	224	378	224	378	3.1e-06	8.4e-06	6.550750e-01	0	35	0.22
ex72a	197	215	197	215	4.2e-01	1.0e+00	4.579770e+02	0	4	0.02
ex73a	193	211	193	211	4.1e-01	1.0e+00	4.449144e+02	0	4	0.02
forest6	66	131	66	131	9.2e+01	6.5e-01	4.139797e+05	0	11	0.02
galenet	8	14	5	9	4.7e+01	9.4e-01	0.000000e+00	0	5	0.00
gosh	3792	13455	3479	12502	1.7e+01	1.1e+01	4.141377e+02	1	13	11.22
gran †	2658	2525								
greenbea-i	2393	5596	1933	4153	1.1e+04	6.6e+00	2.199821e+03	1	9	1.50
itest2	9	13	9	13	2.0e+01	4.6e-01	0.000000e+00	0	5	0.00
itest6	11	17	10	15	5.0e+05	1.1e+00	8.730497e+05	0	5	0.00
klein1	54	108	54	108	3.2e+03	1.5e+01	0.000000e+00	1	23	0.08
klein2	477	531	477	531	3.0e+04	3.7e+02	0.000000e+00	2	22	5.18
klein3	994	1082	994	1082	9.5e+04	1.4e+03	0.000000e+00	3	27	47.56
mondou2	312	604	259	467	8.0e+00	4.0e+00	6.313890e+08	0	8	0.05
pang	361	741	333	685	1.1e-02	3.6e+00	2.108127e+04	0	28	0.34
pilot4i	410	1123	396	1022	1.5e+04	5.7e-01	-1.377993e+03	1	33	1.37
qual	323	464	305	441	1.5e+00	3.4e-02	-5.882060e+04	1	27	0.32
reactor	318	808	269	602	9.9e+00	6.4e-01	-3.285658e+05	0	9	0.11
refinery	323	464	303	439	3.5e+01	6.4e-01	-5.227856e+04	0	20	0.21
vol1	323	464	305	441	2.3e+00	1.3e-01	-6.111366e+04	1	26	0.32
woodinfe †	35	89								

Table 3: NEMS problems

Name	Before Preprocessing Rows	Before Preprocessing Cols	After Preprocessing Rows	After Preprocessing Cols	Relative Infeas	Relative Compl	Primal Objective	Max Add. Corr.	Iters	CPU Time [secs]
NEMSafm	334	2348	322	1402	1.2e-14	1.7e-11	-6.792374e+03	0	17	0.18
NEMScem	651	1712	479	1540	3.3e-10	1.9e-08	8.977233e+04	0	19	0.30
NEMSemm1	3945	75352	3230	41048	5.1e-15	3.6e-06	5.129614e+05	1	64	166.40
NEMSemm2	6943	48878	4526	26754	1.6e-10	2.2e-06	5.810806e+05	0	37	10.77
NEMSpmm1	2372	8903	2227	7145	2.4e-08	4.8e-08	3.274158e+05	1	38	11.45
NEMSpmm2	2301	8734	2081	7944	3.5e-08	1.9e-07	2.917948e+05	1	40	12.92
NEMSwrld	7138	28550	5621	23706	3.3e-13	1.6e-06	-2.603093e+02	2	42	182.22

References

[1] E. D. ANDERSEN AND K. D. ANDERSEN, *Presolving in linear programming*, Mathematical Programming, 71 (1995), pp. 221–245.

[2] A. R. CURTIS AND J. K. REID, *On the automatic scaling of matrices for Gaussian elimination*, J. Inst. Maths Applics, 10 (1972), pp. 118–124.

[3] J. R. GILBERT, E. NG, AND B. W. PEYTON, *An efficient algorithm to compute row and column counts for sparse cholesky factorization*, SIAM Journal on Matrix Analysis and Applications, 15 (1994), pp. 1075–1091.

[4] J. Gondzio, *Multiple centrality corrections in a primal-dual method for linear programming*, Computational Optimization and Applications, 6 (1996), pp. 137–156.

[5] A. Gupta, M. Joshi, and V. Kumar, *WSSMP: Watson Symmetric Sparse Matrix Package*, IBM Research Report RC 20923 (92699), IBM, July 1997.

[6] J. W.-H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software, 11 (1985), pp. 141–153.

[7] ——, *The role of elimination trees in sparse factorization*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 134–172.

[8] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization, 2 (1992), pp. 575–601.

[9] E. Ng and B. W. Peyton, *Block sparse Cholesky algorithms on advanced uniprocessor computers*, SIAM Journal on Scientific Computing, 14 (1993), pp. 1034–1056.

[10] S. J. Wright, *The Cholesky factorization in interior-point and barrier methods*, Preprint MCS–P600–0596, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., May 1996.

[11] ——, *Primal-Dual Interior-Point Methods*, SIAM Publications, Philadelphia, Pa, 1997.

[12] Y. Zhang, *User's Guide to LIPSOL*, Department of Mathematics and Statistics, University of Maryland Baltimore County, Baltimore, Maryland, July 1995.

[13] ——, *Solving large-scale linear programs by interior-point methods under the MATLAB enviroment*, Technical Report TR96-01, Department of Mathematics and Statistics, University of Maryland Baltimore County, Baltimore, Md, 1996.

[14] *Annual Energy Outlook 1996*, Energy Information Administration, U. S. Department of Energy, Washington, DC 20585, 1996. Document DOE/EIA-0383(96).