Mixed Semidefinite–Quadratic–Linear Programs

Jean–Pierre Haeberly* Madhu V. Nayakkankuppam†
Michael L. Overton‡

September 23, 1998

Abstract

We consider mixed semidefinite–quadratic–linear programs. These are linear optimization problems with three kinds of cone constraints, namely: the semidefinite cone, the quadratic cone and the nonnegative orthant. We outline a primal–dual path following method to solve these problems and highlight the main features of SDPpack, a Matlab package which solves such programs. We give some examples where such mixed programs arise, and provide numerical results on benchmark problems.

1 Introduction

We consider the following mixed semidefinite–quadratic–linear program (SQLP):

$$\begin{array}{llr} (1) & \max & b^T y \\ & \text{s.t.} & \\ (2) & & F_0^{(k)} + \sum_{i=1}^m y_i F_i^{(k)} \geq 0, \quad k = 1, \ldots, L \\ (3) & & \|(c^{(k)} - A^{(k)})^T y\| \leq \gamma^{(k)} - (g^{(k)})^T y, \quad k = 1, \ldots, M \\ (4) & & (A^{(0)})^T y \leq c^{(0)} \end{array}$$

where $y \in \mathbf{R}^m$ and

$$\begin{gathered} F_i^{(k)} \in \mathcal{S}^{n_k}, \quad i = 0, \ldots, m, \ k = 1, \ldots, L \\ A^{(k)} \in \mathbf{R}^{m \times p_k}, \ c^{(k)} \in \mathbf{R}^{p_k}, \ g^{(k)} \in \mathbf{R}^m, \ \gamma^{(k)} \in \mathbf{R}, \quad k = 1, \ldots, M \\ A^{(0)} \in \mathbf{R}^{m \times p_0}, \ c^{(0)} \in \mathbf{R}^{p_0}. \end{gathered}$$

The first set of constraints (2) consists of L semidefinite cone constraints of size $n_1, \ldots, n_L$ respectively. The second set of constraints (3) consists of M quadratic

*Mathematics Department, Fordham University, Bronx, NY 10458. E–mail: haeberly@murray.fordham.edu

†Courant Institute, 251 Mercer Street, New York, NY 10012. E–mail: madhu@cs.nyu.edu

‡Courant Institute, 251 Mercer Street, New York, NY 10012. E–mail: overton@cs.nyu.edu

cone constraints of size $p_1, \ldots, p_M$ respectively. The final constraint (4) consists of p_0 ordinary linear inequality constraints. We use y instead of x as the optimization variable, with maximization objective $b^T y$ instead of minimization objective $c^T x$ as used elsewhere in this book, in order to make the notation used here more compatible with the SDPpack software to be described later in this chapter.

It is convenient to rewrite the problem as follows:

$$\begin{array}{lc}
\max & b^T y \\
\text{ s.t.} & \\
(5) & F^{(k)} = F_0^{(k)} + \sum_{i=1}^m y_i F_i^{(k)}, \quad k = 1, \ldots, L \\
(6) & f^{(k)} = c^{(k)} - (A^{(k)})^T y, \quad \phi^{(k)} = \gamma^{(k)} - (g^{(k)})^T y, \quad k = 1, \ldots, M \\
(7) & f^{(0)} = c^{(0)} - (A^{(0)})^T y, \\
(8) & F^{(k)} \geq 0,\ k = 1, \ldots, L; \quad \|f^{(k)}\| \leq \phi^{(k)},\ k = 1, \ldots, M; \quad f^{(0)} \geq 0
\end{array}$$

where $F^{(k)} \in \mathcal{S}^{n_k}$ $(k = 1, \ldots, L)$; $f^{(k)} \in \mathbf{R}^{p_k}$ and $\phi^{(k)} \in \mathbf{R}$ $(k = 1, \ldots, M)$; $f^{(0)} \in \mathbf{R}^{p_0}$. We write

$$\begin{array}{rcl}
F_i & = & \mathbf{diag}(F_i^{(1)}, \ldots, F_i^{(L)}) \quad (i = 1, \ldots, m), \\
F & = & \mathbf{diag}(F^{(1)}, \ldots, F^{(L)}), \\
\text{and} & & \\
f & = & \left[\phi^{(1)}, (f^{(1)})^T, \cdots, \phi^{(M)}, (f^{(M)})^T\right]^T.
\end{array}$$

The dual optimization problem is

$$(9) \quad \min \quad \sum_{k=1}^{L} \mathbf{Tr} F_0^{(k)} Z^{(k)} + \sum_{k=1}^{M} \left[(c^{(k)})^T z^{(k)} + \gamma^{(k)} \zeta^{(k)}\right] + (c^{(0)})^T z^{(0)}$$

s.t.

$$(10) \quad \begin{bmatrix} -\sum_{k=1}^L \mathbf{Tr} F_1^{(k)} Z^{(k)} \\ \vdots \\ -\sum_{k=1}^L \mathbf{Tr} F_m^{(k)} Z^{(k)} \end{bmatrix} + \sum_{k=1}^{M} \left[A^{(k)} z^{(k)} + \zeta^{(k)} g^{(k)}\right] + A^{(0)} z^{(0)} = b$$

$$(11) \quad Z^{(k)} \geq 0,\ k = 1, \ldots, L; \quad \|z^{(k)}\| \leq \zeta^{(k)},\ k = 1, \ldots, M; \quad z^{(0)} \geq 0$$

where the dual variables are $Z^{(k)} \in \mathcal{S}^{n_k}$, $k = 1, \ldots, L$; $z^{(k)} \in \mathbf{R}^{p_k}$ and $\zeta^{(k)} \in \mathbf{R}$, $k = 1, \ldots, M$; and $z^{(0)} \in \mathbf{R}^{p_0}$. Note that the constraints $Z^{(k)} \geq 0$ are semidefinite constraints. We write

$$Z = \mathbf{diag}(Z^{(1)}, \ldots, Z^{(L)})$$

and

$$z = \left[\zeta^{(1)}, (z^{(1)})^T, \ldots, \zeta^{(M)}, (z^{(M)})^T\right]^T.$$

Let us denote the primal objective (1) by $\mathcal{P}(y)$ and the the dual objective (9) by $\mathcal{D}(Z, z, z^{(0)})$. If y and $(Z, z, z^{(0)})$ respectively satisfy the primal and dual feasibility constraints, it immediately follows that

$$\mathcal{P}(y) = \mathcal{D}(Z, z, z^{(0)}) - \mathbf{Tr} ZF - z^T f - (z^{(0)})^T f^{(0)}.$$

Since Z and F are both positive semidefinite, $\mathbf{Tr} ZF \geq 0$; this follows using the Cholesky factorizations $Z = R^T R$ and $F = S^T S$, as

$$\mathbf{Tr} ZF = \mathbf{Tr} R^T R S^T S = \mathbf{Tr} R S^T S R^T = \mathbf{Tr}(SR^T)^T (SR^T) = \|SR^T\|_F^2,$$

where $\|\cdot\|_F$ stands for the Frobenius norm. Likewise, Cauchy–Schwarz shows that $z^T f \geq 0$, since

$$z^T f = \sum_{k=1}^{L} \zeta^{(k)} \phi^{(k)} + (z^{(k)})^T f^{(k)} \geq \sum_{k=1}^{L} \zeta^{(k)} \phi^{(k)} - \|z^{(k)}\|, \|f^{(k)}\| \geq 0.$$

Also, of course, $(z^{(0)})^T f^{(0)} \geq 0$ is implied by $z^{(0)} \geq 0$, $f^{(0)} \geq 0$. Thus we have

$$\mathcal{P}(y) \leq \mathcal{D}(Z, z, z^{(0)})$$

with equality (a zero duality gap) if and only if $\mathbf{Tr} ZF = 0$, $z^T f = 0$, and $(z^{(0)})^T f^{(0)} = 0$. These three conditions are called complementarity conditions, and may be summarized as

$$\mathbf{Tr} ZF + z^T f + (z^{(0)})^T f^{(0)} = 0. \tag{12}$$

Furthermore, again using Cholesky, we see that if $\mathbf{Tr} ZF = 0$, then the matrix product ZF must itself be zero (since $SR^T = 0$), and therefore (since Z and F are symmetric) that $ZF + FZ = 0$. (Recall that this is a block diagonal matrix equation.)

Likewise, we can also elaborate on the second complementarity condition. The condition for equality in Cauchy–Schwarz shows that if $z^T f = 0$, then

$$(z^{(k)})^T f^{(k)} + \zeta^{(k)} \phi^{(k)} = 0, \text{ with } \phi^{(k)} z^{(k)} = -\zeta^{(k)} f^{(k)}, \quad k = 1, \ldots, M.$$

Following [1], this condition may be conveniently expressed as

$$\mathbf{Arw}(z)\, \mathbf{Arw}(f)\, \mathbf{e}_q = 0$$

where $\mathbf{e}_q$ is a block vector whose kth block is $\mathbf{e}_q^{(k)} = [1, 0, \ldots, 0]^T \in \mathbf{R}^{p_k+1}$, $k = 1, \ldots, M$, and the block diagonal matrix

$$\mathbf{Arw}(z) = \mathbf{diag}(\mathbf{arw}(\zeta^{(1)}, z^{(1)}), \ldots, \mathbf{arw}(\zeta^{(M)}, z^{(M)}))$$

has blocks defined, for $\alpha \in \mathbf{R}$, $a \in \mathbf{R}^p$, by the "arrow matrix"

$$\mathbf{arw}(\alpha, a) = \begin{bmatrix} \alpha & a_1 & a_2 & \ldots & a_p \\ a_1 & \alpha & 0 & \ldots & 0 \\ a_2 & 0 & \alpha & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_p & 0 & 0 & 0 & \alpha \end{bmatrix}.$$

Finally, the third complementarity condition $(z^{(0)})^T f^{(0)} = 0$ implies the componentwise condition $(z^{(0)})_k (f^{(0)})_k = 0$, $k = 1, \ldots, p_0$, since $z^{(0)} \geq 0$ and $f^{(0)} \geq 0$. Letting $\mathbf{e}_\ell = [1, \ldots, 1]^T$, this is conveniently written as

$$\mathbf{diag}(z^{(0)})\, \mathbf{diag}(f^{(0)})\, \mathbf{e}_\ell = 0.$$

This much is just linear algebra. To go further we make use of a well known result from optimization theory [14] stating that if strictly feasible primal and dual variables exist (points which satisfy all the inequalities strictly), then the optimal primal and dual objective values are equal, and that there exist primal and dual variables achieving these optimal values. Thus, the complementarity conditions must hold at the (optimal) solutions.

In what follows we make this strict feasibility assumption and we also assume, mainly for convenience, that there are no redundant equality constraints in the dual formulation of the problem.

2 A Primal–Dual Interior–Point Method

The SQLP described in Section 1 can be efficiently solved with interior–point methods. Of the many flavors of interior–point methods [15, 17], we briefly describe the algorithm implemented in the public domain package SDPpack [2]. This is a primal-dual path-following algorithm based on the "XZ+ZX" direction of [4] (to be translated here as "ZF+FZ" given the notational differences), using a Mehrotra predictor–corrector scheme.

The optimality conditions are completely determined by the three complementarity conditions together with the primal and the dual constraints. Let **svec** denote a map from symmetric, block diagonal matrices (with block sizes $n_1, \ldots, n_L$) to column vectors, so that $\mathbf{svec}(F) \in \mathbf{R}^N$, where

$$N = \sum_{k=1}^{L} n_k(n_k + 1)/2,$$

satisfies $\mathbf{Tr} ZF = \mathbf{svec}(Z)^T \mathbf{svec}(F)$ for all symmetric block diagonal matrices Z, F. Then we may abbreviate the primal slack linear constraints (5) – (7) by

$$\begin{bmatrix} \mathbf{A}_s^T \\ \mathbf{A}_q^T \\ \mathbf{A}_\ell^T \end{bmatrix} y + \begin{bmatrix} \mathbf{svec}(F) \\ f \\ f^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_s \\ \mathbf{c}_q \\ \mathbf{c}_\ell \end{bmatrix}$$

where

$$\mathbf{A}_s = \begin{bmatrix} -\mathbf{svec}\left(\mathbf{diag}(F_1^{(1)}, \ldots, F_1^{(L)})\right)^T \\ \vdots \\ -\mathbf{svec}\left(\mathbf{diag}(F_m^{(1)}, \ldots, F_m^{(L)})\right)^T \end{bmatrix}, \quad \mathbf{c}_s = \mathbf{svec}\left(\mathbf{diag}(F_0^{(1)}, \ldots, F_0^{(L)})\right)$$

$$\mathbf{A}_q = [\, g^{(1)} \quad A^{(1)} \quad \cdots \quad g^{(M)} \quad A^{(M)} \,], \quad \mathbf{c}_q = \left[\gamma^{(1)}, (c^{(1)})^T, \cdots, \gamma^{(M)}, (c^{(M)})^T\right]^T$$

and

$$\mathbf{A}_\ell = A^{(0)}, \quad \mathbf{c}_\ell = c^{(0)}.$$

The dimensions of $\mathbf{A}_s$, $\mathbf{A}_q$ and $\mathbf{A}_\ell$ are respectively $m \times N$, $m \times \sum_{k=1}^{M}(p_k + 1)$ and $m \times p_0$. The dual equality constraints (10) are then written

$$[\, \mathbf{A}_s \quad \mathbf{A}_q \quad \mathbf{A}_\ell \,] \begin{bmatrix} \mathbf{svec}(Z) \\ z \\ z^{(0)} \end{bmatrix} = b.$$

Furthermore, the complementarity conditions can be written

$$\begin{bmatrix} \mathbf{svec}(\frac{1}{2}(ZF + FZ)) \\ \mathbf{Arw}(z)\mathbf{Arw}(f)\mathbf{e}_q \\ \mathbf{diag}(z^{(0)})\mathbf{diag}(f^{(0)})\mathbf{e}_\ell \end{bmatrix} = 0$$

where $\mathbf{e}_q$ and $\mathbf{e}_\ell$ were defined in Section 1. We collect the primal and dual equality constraints and a relaxed form of the complementarity condition, called a "centering" condition, in the nonlinear system of equations

$$H(y, F, f, f^{(0)}, Z, z, z^{(0)}) = \begin{bmatrix} \begin{bmatrix} \mathbf{A}_s^T \\ \mathbf{A}_q^T \\ \mathbf{A}_\ell^T \end{bmatrix} y + \begin{bmatrix} \mathbf{svec}(F) \\ f \\ f^{(0)} \end{bmatrix} - \begin{bmatrix} \mathbf{c}_s \\ \mathbf{c}_q \\ \mathbf{c}_\ell \end{bmatrix} \\ [\, \mathbf{A}_s \quad \mathbf{A}_q \quad \mathbf{A}_\ell \,] \begin{bmatrix} \mathbf{svec}(Z) \\ z \\ z^{(0)} \end{bmatrix} - b \\ \mathbf{svec}(\frac{1}{2}(ZF + FZ) - \mu I) \\ \mathbf{Arw}(z)\mathbf{Arw}(f)\mathbf{e}_q - \mu \mathbf{e}_q \\ \mathbf{diag}(z^{(0)})\mathbf{diag}(f^{(0)})\mathbf{e}_\ell - \mu \mathbf{e}_\ell \end{bmatrix} = 0$$

where $\mu > 0$ is a parameter. It is well known that, for all $\mu > 0$, this nonlinear system of equations has a unique solution satisfying the cone inequality constraints: the set of such solutions is called the *central path* [14]. As $\mu \to 0$, this central solution converges to a (not necessarily unique) primal–dual solution of the SQLP.

The algorithm we now outline uses Newton's method to approximately follow the central path as $\mu \to 0$. For a given approximate primal–dual solution $(y, F, f, f^{(0)}, Z, z, z^{(0)})$ and central path parameter μ, Newton's method defines a "search direction" by the linear system of equations

$$\mathbf{J} \begin{bmatrix} \Delta y \\ \mathbf{svec}(\Delta F) \\ \Delta f \\ \Delta f^{(0)} \\ \mathbf{svec}(\Delta Z) \\ \Delta z \\ \Delta z^{(0)} \end{bmatrix} = -H(y, F, f, f^{(0)}, Z, z, z^{(0)}) \tag{13}$$

where the Jacobian $\mathbf{J}$ of H is the matrix

$$\begin{bmatrix} \mathbf{A}_s^T & I & 0 & 0 & 0 & 0 & 0 \\ \mathbf{A}_q^T & 0 & I & 0 & 0 & 0 & 0 \\ \mathbf{A}_\ell^T & 0 & 0 & I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{A}_s & \mathbf{A}_q & \mathbf{A}_\ell \\ 0 & I * Z & 0 & 0 & I * F & 0 & 0 \\ 0 & 0 & \mathbf{Arw}(z) & 0 & 0 & \mathbf{Arw}(f) & 0 \\ 0 & 0 & 0 & \mathbf{diag}(z^{(0)}) & 0 & 0 & \mathbf{diag}(f^{(0)}) \end{bmatrix}.$$

Here the symbol $*$ is the "symmetrized Kronecker product" [4]: for symmetric block diagonal matrices A, B, X (with block sizes $n_1, \ldots, n_L$),

$$(A * B)\,\mathbf{svec}X = \frac{1}{2}\mathbf{svec}\,(AXB + BXA)\,.$$

This linear system is not solved directly, but via an efficient method exploiting block Gaussian elimination, that is outlined in the next section.

Once primal and dual "search directions" are computed, it is necessary to compute steps along these to the relevant cone boundaries, so that subsequent iterates will respect the inequality constraints. This requires an eigenvalue computation of a block diagonal symmetric matrix to determine the step to the boundary of the semidefinite cone, the solution of M quadratic equations to determine the step to the boundary of the quadratic cone, and an ordinary ratio test to determine the step to the boundary of the positive orthant. The minimum of these step lengths determines the maximum step which can be taken. This procedure is used to determine one step length for the primal variables and another for the dual variables.

At a more detailed level, the algorithm uses a predictor–corrector scheme. This requires the following steps, which constitute one interior–point iteration:

- Given a current iterate $(y, F, f, f^{(0)}, Z, z, z^{(0)})$, compute a *predictor* search direction by solving the linear system shown above, with $\mu = 0$, for $\left(\Delta y^{\text{pred}}, \Delta F^{\text{pred}}, \Delta f^{\text{pred}}, (\Delta f^{(0)})^{\text{pred}}, \Delta Z^{\text{pred}}, \Delta z^{\text{pred}}, (\Delta z^{(0)})^{\text{pred}}\right)$.

- Compute the primal and dual step lengths to the cone boundaries along this direction, α^{pred} and β^{pred} respectively, so that

$$\begin{pmatrix} F^{\text{pred}} = F + \alpha^{\text{pred}} \Delta F^{\text{pred}} \\ f^{\text{pred}} = f + \alpha^{\text{pred}} \Delta f^{\text{pred}} \\ (f^{(0)})^{\text{pred}} = f^{(0)} + \alpha^{\text{pred}} (\Delta f^{(0)})^{\text{pred}} \end{pmatrix}$$

and

$$\begin{pmatrix} Z^{\text{pred}} = Z + \beta^{\text{pred}} \Delta Z^{\text{pred}} \\ z^{\text{pred}} = z + \beta^{\text{pred}} \Delta z^{\text{pred}} \\ (z^{(0)})^{\text{pred}} = z^{(0)} + \beta^{\text{pred}} (\Delta z^{(0)})^{\text{pred}} \end{pmatrix}$$

are each on the boundary of the product of the semidefinite cone, the quadratic cone and the postive orthant.

- Compute

$$\mu = \frac{\big(\mathbf{Tr}\, Z^{\text{pred}} F^{\text{pred}} + (z^{\text{pred}})^T f^{\text{pred}} + ((z^{(0)})^{\text{pred}})^T (f^{(0)})^{\text{pred}}\big)^3}{\nu \big(\mathbf{Tr}\, ZF + z^T f + (z^{(0)})^T f^{(0)}\big)^2}.$$

  where $\nu = \sum_{k=1}^{M} n_k + \sum_{k=1}^{L} (p_k + 1) + p_0$. This expression for μ is known as Mehrotra's formula.

- Recompute the relaxed complementarity conditions in the right hand side of (13) using the Mehrotra formula for μ and including second-order corrections, by substituting

$$\begin{bmatrix} \mathbf{svec}(\mu I - \frac{1}{2}(ZF + FZ + \Delta Z^{\text{pred}} \Delta F^{\text{pred}} + \Delta F^{\text{pred}} \Delta Z^{\text{pred}})) \\ \mu \mathbf{e}_q - (\mathbf{Arw}(z)\mathbf{Arw}(f) + \mathbf{Arw}(z^{\text{pred}})\mathbf{Arw}(f^{\text{pred}}))\mathbf{e}_q \\ \mu \mathbf{e}_\ell - (\mathbf{diag}(z^{(0)})\mathbf{diag}(f^{(0)}) + \mathbf{diag}((z^{(0)})^{\text{pred}})\mathbf{diag}((f^{(0)})^{\text{pred}})\mathbf{e}_\ell \end{bmatrix}$$

  for the last three components in the right-hand side of (13). Solve the linear system again for a *corrector* search direction.

- Compute primal and dual step lengths α and β along the corrector search direction and update the primal and dual variables accordingly. The step lengths are chosen so that the new iterates satisfy the inequality constraints. Furthermore, they are never set to a value larger than unity. This can be done by taking the step length to be the smaller of unity and a fixed fraction (say 0.999) times the step to the boundary of the cone. If a unit primal (dual) step is taken, the new iterate also satisfies the primal (dual) equality constraints.

The choice of the sequence of values for μ greatly influences the performance of the algorithm. The method described above is Mehrotra's predictor–corrector scheme [13], originally proposed for linear programming, which significantly reduces the number of interior–point iterations needed to compute a solution of given accuracy. The predictor step helps calculate a target value for μ, while the corrector step includes some second order terms lost in the linearization of H in the predictor step. For a detailed interpretation of Mehrotra's predictor–corrector scheme, see [17, p.193–198].

3 Software

In this section, we first discuss some implementation details involved in solving the linear system arising in each interior–point iteration. Then, we highlight the main features of SDPpack (Version 0.9 Beta),[1] a package (based on Matlab 5.0) for solving SQLP, written by the authors together with F. Alizadeh and S. Schmieta.

[1] Available from the SDPpack web page: http://www.cs.nyu.edu/overton/sdppack.

3.1 Some Implementation Issues

The Newton system (13) admits a reduction to a compact Schur complement form via block Gauss elimination. Let

$$
\begin{aligned}
\mathbf{U} &= \mathbf{diag}(I * F, \mathbf{Arw}(f), f^0) \\
\mathbf{V} &= \mathbf{diag}(I * Z, \mathbf{Arw}(z), z^0) \\
\mathbf{A} &= [\mathbf{A}_s \;\; \mathbf{A}_q \;\; \mathbf{A}_\ell],
\end{aligned}
\tag{14}
$$

so that

$$
\mathbf{J} = \begin{bmatrix} \mathbf{A}^T & I & 0 \\ 0 & 0 & \mathbf{A} \\ 0 & \mathbf{V} & \mathbf{U} \end{bmatrix}.
$$

Eliminating $(\mathbf{svec}(\Delta Z), \Delta z, \Delta z^0)$ in (13), the matrix $\mathbf{J}$ transforms to

$$
\begin{bmatrix} \mathbf{A}^T & -\mathbf{V}^{-1}\mathbf{U} \\ 0 & \mathbf{A} \end{bmatrix},
$$

and on further elimination of $(\mathbf{svec}(\Delta F), \Delta f, \Delta f^0)$, we arrive at the $m \times m$ Schur complement matrix

$$
M = \mathbf{A}\mathbf{U}^{-1}\mathbf{V}\mathbf{A}^T
$$

in a linear system involving only the variable Δy. The Schur complement matrix M can be computed as the sum of the individual Schur complements for the semidefinite (M_s), the quadratic (M_q) and the linear (M_ℓ) parts. Moreover, M_s and M_q can be computed *blockwise*. Writing $\mathbf{A}_s^{(k)}$ (likewise $\mathbf{A}_q^{(k)}$) to mean the columns of $\mathbf{A}_s$ (likewise $\mathbf{A}_q$) corresponding to the k^{th} block, it can be verified that $M = M_s + M_q + M_\ell$, where

$$
\begin{aligned}
M_s &= \sum_{k=1}^{L} \mathbf{A}_s^{(k)} (I * F^{(k)})^{-1} (I * Z^{(k)}) (\mathbf{A}_s^{(k)})^T \\
M_q &= \sum_{k=1}^{M} \mathbf{A}_q^{(k)} \, \mathbf{arw}(\phi^{(k)}, f^{(k)})^{-1} \, \mathbf{arw}(\zeta^{(k)}, z^{(k)}) (\mathbf{A}_q^{(k)})^T \\
M_\ell &= \mathbf{A}_\ell \, \mathbf{diag}(f^{(0)})^{-1} \, \mathbf{diag}(z^{(0)}) \mathbf{A}_\ell^T.
\end{aligned}
\tag{15}
$$

The contribution to the (i,j) entry of M_s from the k^{th} semidefinite block can be computed as $F_i^{(k)} \bullet V_j^{(k)}$, where the $n_k \times n_k$ symmetric matrix $V_j^{(k)}$ is the solution to the Lyapunov equation

$$
F^{(k)} V_j^{(k)} + V_j^{(k)} F^{(k)} = R_j^{(k)},
\tag{16}
$$

where $R_j^{(k)} = Z^{(k)} F_j^{(k)} + F_j^{(k)} Z^{(k)}$. The standard method to solve this equation is via the spectral factorization $F^{(k)} = P \, \mathbf{diag}(\lambda) \, P^T$, where P is an orthonormal matrix of eigenvectors and $\lambda = [\lambda_1, \ldots, \lambda_{n_k}]^T$, a vector of eigenvalues. The Lyapunov equation is solved by computing

$$S = (P^T R_j^{(k)} P) \circ L, \text{ where } L(s,t) = 1/(\lambda_s + \lambda_t), \text{ and} \tag{17}$$
$$V_j^{(k)} = PSP^T, \tag{18}$$

where the symbol $\circ$ in (17) denotes the Hadamard (or componentwise) product. The contribution from the k^{th} quadratic block to M_q can be computed efficiently by observing that the arrow matrix $\mathbf{arw}(\zeta^{(k)}, z^{(k)})$ is the sum of a diagonal matrix and a matrix of rank two.

3.2 Using SDPpack

We now describe the main routines of SDPpack (Version 0.9 beta); in what follows, `>>` denotes the Matlab prompt.

The problem data and an initial starting point are stored using structures:

Matlab variables	Data
`A.s`, `A.q`, `A.l`	$\mathbf{A}_s$, $\mathbf{A}_q$, $\mathbf{A}_\ell$
`C.s`, `C.q`, `C.l`	F_0, $\mathbf{c}_q$, $\mathbf{c}_\ell$
`blk.s`, `blk.q`, `blk.l`	$[n_1, \ldots, n_L]^T$, $[p_1, \ldots, p_M]^T$, p_0
`X.s`, `X.q`, `X.l`	Z, z, $z^{(0)}$
`Z.s`, `Z.q`, `Z.l`	F, f, $f^{(0)}$

For example, the following lines of Matlab code set up a randomly generated SQLP:

```
>> setpath;   % add SDPpack subdirectories to Matlab path
>> blk.s = [5 5 5];  blk.q = [10 10];  blk.l = 30;
>> m = 10;
>> sqlrnd;    % set up a random, strictly feasible SQLP
```

The constraint matrix `A.s` may also be set up using the routine `makeA`, which assumes that the matrices $F_i^{(k)}$ are stored using Matlab's cell arrays:

```
>> blk.s = [3 3]; blk.q = 3; m = 2;
>> b = [1 2]';
>> Amat{1} = speye(6);
>> Amat{2}(1:3,1:3) = sparse([1 2 3; 2 4 5; 3 5 6]);
>> Amat{2}(4:6,4:6) = sparse([1 0 0; 0 2 2; 0 2 3]);
>> A.s = makeA(Amat, blk.s);  % creates the 2 x 12 matrix A.s
>> C.s(1:3,1:3) = sparse(ones(3,3));
>> C.s(4:6,4:6) = sparse(zeros(3,3));
>> A.q = [1 2 3; 4 5 6];
>> C.q = [1 -1 -1]';
```

The routines `import` (`export`) can be used to load (save) ASCII data from (to) a file.

Note that when there is more than one semidefinite block (as in the example above with two blocks), the constraint matrices `Amat{i}` containing $-F_i$ $(i = 1, \ldots, m)$, and the cost matrix `C.s` containing $-F_0$ must be stored in sparse format.

Once the problem data are set up, the optimization parameters, which are stored in the structure `opt`, must be assigned appropriate values; these may be initialized to their default values with the routine

```
>> setopt;
```

The next step is to provide a starting point, *i.e.* the variables `X`, `y` and `Z`. Generally, it suffices to provide an infeasible starting point, *i.e.* points for which (5) – (7) and (10) need not necessarily hold, although the cone inequalities (8) and (11) must be satisfied. A default starting point (usually infeasible) may be generated with the routine

```
>> init;
```

which initializes the primal variables y to zero, the primal slack variables `Z.s`, `Z.q`, `Z.l`, and the dual variables `X.s`, `X.q`, `X.l` to `scalefac` $* (I, \mathbf{e}_q, \mathbf{e}_\ell)$, where `scalefac` is a user–defined scale factor set to a default value by `setopt`. Again, if there is more than one semidefinite block, the matrices `X.s` and `Z.s` must be stored in the sparse format.

The solver can then be invoked with the Matlab function call

```
>> [X, y, Z, iter, compval, feasval, objval, termflag] = ...
      fsql(A, b, C, blk, X, y, Z, opt);
```

or by the driver script

```
>> sql;
```

which optionally prints out additional summary information about the accuracy to which the problem was solved and the CPU time consumed.

When the algorithm terminates, a termination flag indicates the cause for termination, which is usually one of the following:

- a solution of the requested accuracy has been found. In this case, the sum of the primal infeasibility, the dual infeasibility and complementarity at the computed solution satisfies both an absolute and a relative tolerance, specified in the `opt` structure.

- an iterate $(F, f, f^{(0)})$ or $(Z, z, z^{(0)})$ violates a cone constraint — a normal phenomenon that usually happens due to numerical rounding errors in the last stages of the algorithm. In this case, the previous iterate, which *does* satisfy the cone constraints, is returned as the computed solution.

- the Schur complement matrix is numerically singular. If this occurs in the early stages of the algorithm, the most likely cause is linear dependence in the rows of the constraint matrix $\mathbf{A}$. Preprocessing can be done (cf. Section 3.3) to detect and rectify this situation.

- the algorithm fails to make sufficient progress in an iteration. The progress made in an iteration is deemed insufficient (based on parameters in the `opt` structure) if the step causes an inordinate increase in the primal or the dual infeasibilities without a commensurate decrease in complementarity. Again, the previous iterate is returned as the computed solution.

- the norm of $(F, f, f^{(0)})$ or of $(Z, z, z^{(0)})$ exceeds a threshold (defined by the user in the `opt` structure). Primal (dual) unboundedness is an indication of dual (primal) infeasibility.

For full details, we refer the reader to the SDPpack User Manual [2]; see also Table 1.

Due to the loop overhead in Matlab, each block diagonal matrix is stored as one large sparse matrix, instead of as individual blocks. This has the advantage that a sparse block in such a matrix is efficiently handled using Matlab's sparse matrix operations. On the other hand, a dense block suffers the penalty involved in sparse matrix access. The loop overhead in Matlab also hinders the effective use of sparsity in the formation of the Schur complement matrix. For instance, in the truss topology design problems, most of the blocks in every constraint matrix are completely zero, and hence do not contribute to the Schur complement at all (see (15)), but this could not be exploited effectively. A stand–alone C code currently under development incorporates some techniques to improve performance on sparse problems.

3.3 Miscellaneous Features

The package provides several support routines which include:

- the routines `scomp`, `pcond` and `dcond` to examine complementarity and degeneracy conditions [3, 6]. These may be used, in particular, to check whether the computed primal and the dual solutions are unique.

- the routine `makesql` to generate random SQLP's with solutions having prescribed "ranks". This routine lets the user specify

  - for each semidefinite block k, the rank of $F^{(k)}$ (likewise $Z^{(k)}$),
  - for each quadratic block k, whether $f^{(k)}$ (likewise $z^{(k)}$) lies in the interior of the quadratic cone, or on its boundary, or is identically zero, and
  - for the linear block, the number of components of $f^{(0)}$ (likewise $z^{(0)}$) that are positive.

  Since nondegeneracy automatically implies bounds on ranks of the solution [3], this routine is particularly useful in generating test problems whose solutions have specified degeneracy and complementarity properties.

- the routine `preproc` to examine the consistency of the constraints. If they are consistent, the redundant constraints, if any, are eliminated via a QR factorization of the constraint matrix $\mathbf{A}$ (see (14)).
- the specialized solvers `dsdp` and `lsdp` (based on the "XZ method" [9, 10]) that exploit the special structure of problems arising in combinatorial optimization (for example, MAX–CUT relaxations and the Lovász θ–function). In these applications, the number of semidefinite blocks $L = 1$ and the matrices $F_1^{(1)}, \ldots, F_m^{(1)}$ are all of the form $e_i e_i^T$ or $e_i e_j^T + e_j e_i^T$. Here e_i stands for the unit vector with 1 in the i^{th} location and zeros everywhere else.

More information about any routine is available by typing `help <routine name>` from the Matlab prompt.

4 Applications

We consider two examples of SQLP.

Robust least squares This problem is taken from [5]. Given $B, \Delta B \in \mathbf{R}^{r \times s}$ $(r > s)$ and $d, \Delta d \in \mathbf{R}^r$, we want to compute the *unstructured robust least squares* solution $x \in \mathbf{R}^s$ that minimizes the worst case residual

$$\min_x \max_{\| [\Delta B \;\; \Delta d] \|_F \le \rho} \|(B + \Delta B)x - (d + \Delta d)\|$$

where the perturbation $[\Delta B \;\; \Delta d]$ is unknown, but the bound ρ on its norm is known. It is shown in [5] that this problem can be formulated using quadratic cone constraints. Adding componentwise bound constraints, say $x_{\min} \le x \le x_{\max}$, converts this into an SQLP with a quadratic and a linear component.

Clock mesh design This circuit design problem, taken from [16], deals with the propagation of a clock signal in integrated circuits. A rectangular mesh with fixed grid points is given. The interconnecting wire segment between grid points i and j is assumed to be a rectangular strip of a fixed length, but variable width d_{ij}. The electrical characteristics of each wire segment are modeled as a π–segment, *i.e.* a grounded input capacitance, a grounded output capacitance and a connecting conductance, all of which are assumed to depend affinely on the design parameters d_{ij}. A synchronous clock signal (of fixed frequency) applied to a fixed subset of the grid points has to be propagated to the entire mesh with minimal delay. There is an inherent tradeoff between the *dominant delay* (see [16] for the definition) and the power consumption of the circuit. The goal is to optimally choose the widths d_{ij} of the rectangular wire segments so that the power consumption of the circuit is minimized for a fixed value of the dominant delay that we are willing to tolerate. It is shown in [16] that this can be formulated as an SDP. Physical limitations on the width of the segments further impose bound constraints of the form $d_{\min} \leq d_{ij} \leq d_{\max}$, which results in an SQLP with a semidefinite and a linear block. Now, consider a slight variation of this problem: the objective is the same as before, but now, we have a rectangular mesh whose grid spacing is not fixed. Here the wire widths are fixed, but the spacing l_i between successive columns and r_j between successive rows of grid points are variables with $l_{\min} \leq l_i \leq l_{\max}$ and $r_{\min} \leq r_i \leq r_{\max}$. Let us further impose a restriction on the maximum allowable size of the circuit by imposing a bound on the length of the diagonal of the rectangular mesh. This last bound constraint is a convex quadratic constraint, thus resulting in an SQLP with semidefinite, quadratic and linear components.

5 Numerical Results

In this section, we present numerical results on a variety of test problems, both randomly generated and from applications. The latter come from control theory (LMI problems) and truss topology design. The randomly generated problems include general dense problems, the unstructured robust least squares problem, and problems with specific sparsity structure such as those arising from the clock mesh design application.

In the tables, the first column (#) is the problem number, the second column is the dimension of the primal slack variable

$$\dim = \sum_{k=1}^{L} n_k(n_k+1)/2 + \sum_{k=0}^{M}(p_k+1) + p_0,$$

m is the dimension of y, `iter` is the number of interior–point iterations, r_d, r_p and r_c are respectively the dual infeasibility norm (see (10)), the primal infeasibility norm (see (5) – (7)) and the complementarity $\mathbf{Tr}(ZF) + z^T f + (z^{(0)})^T f^{(0)}$ evaluated at the computed solution. The quantity `scalefac` is the factor by which the default starting point $((I, \mathbf{e}_q, \mathbf{e}_\ell), 0, (I, \mathbf{e}_q, \mathbf{e}_\ell))$ is scaled. When `scalefac` is omitted, it means that the default value of 100 was used. The quantities r_p, r_d and r_c are all shown on a log (base 10) scale. The last column (`tf`) is the termination flag (denoted `termflag` in SDPpack User Manual). A brief description of this flag is given in Table 1; for a more detailed explanation, consult the SDPpack User Manual [2]. The asterisks, if any, next to a termination flag indicate the number of *restarts* that were necessary: if the primal or the dual step lengths get very small, the starting point is scaled with a larger value of `scalefac`, the fraction of the step taken towards the boundaries of the cones is made more conservative (*i.e.* reduced), and the algorithm is restarted. The primal and the dual objective values, denoted $\mathcal{P}(y)$ and $\mathcal{D}(Z, z, z^{(0)})$ in the tables, are provided only for the LMI and the truss topology design problems,[2] and not for any of the randomly generated ones.

[2] Available from the SDPpack web page.

All the runs were performed on a Sparc Ultra II workstation (200 MHz CPU with 192 MB of memory). Unless explicitly noted otherwise in the tables, all the problems were solved with SDPpack's options set to their default values.

Random dense problems Table 2 shows results for problems with only semidefinite constraints. Table 3 shows results for problems with only quadratic cone constraints. In Table 2, each problem has three blocks of equal size; the five runs tabulated there correspond to block sizes of 20, 40, 60, 80 and 100. Each problem in Table 3 also has three blocks of equal size; the block sizes for the five runs are 50, 100, 150, 200 and 250.

Clock mesh design This problem from [16] was described in Section 4. These are problems with one semidefinite block and a linear block. The constraint matrices are extremely sparse, and the spatial mesh adjacency relation they describe gives them a banded structure; the results are shown in Table 4.

Robust Least Squares This problem from [5] was described in Section 4. Numerical results for this problem class are shown in Table 5.

Truss topology design These problems are pure SDP's that arise in truss topology design. The constraint matrices here are extremely sparse even for the moderately sized problems. The results are summarized in Table 6.

LMI problems These problems are pure SDP's that arise in control applications. On the problems where the final complementarity residual is unsatisfactory (see Table 7), the dual iterates indeed get large, indicating that the primal problem may be infeasible or weakly infeasible [11].

6 Other Computational Issues

Several enhancements to the basic interior–point iteration of Section 2 have been proposed, and some used with success, in linear programming. We are concerned with extending some of these techniques to SQLP, namely

- Gondzio's multiple centrality corrections [7], and
- Mehrotra's higher order corrections [12].

The main motivation behind considering these enhancements for SQLP is the ratio of the cost of forming and factorizing the Schur complement matrix relative to the cost of a backsolve. For SQLP, this ratio is typically much larger than for LP. Thus, having expended the effort involved in forming and factorizing the Schur complement matrix, it is worthwhile to improve the quality of the search direction by solving multiple linear systems with the same coefficient matrix, but different right hand sides. Both the above schemes can be extended to SQLP, and are described in detail elsewhere [8]. In our preliminary experiments, adapting Gondzio's multiple centrality corrections to SQLP reduces iteration count and CPU time, roughly to the same extent as reported for LP. On the other hand, Mehrotra's higher order corrections reduce the number of iterations and the overall CPU time significantly; this observation is different from LP experience, where the overall CPU time is typically not reduced.

7 Conclusions

In this chapter, we have described mixed semidefinite–quadratic–linear programs. These are linear optimization problems with mixed cone constraints, namely the semidefinite cone, the quadratic cone and the nonnegative orthant. This problem class includes pure semidefinite programming, convex quadratically constrained quadratic programming and linear programming as special cases. We have presented a primal–dual interior–point algorithm for such problems and have described the salient features of SDPpack (Version 0.9 beta), a software package that implements this algorithm. A couple of examples have been given to illustrate how such mixed programs arise in practice, and the numerical results shown here demonstrate the performance of SDPpack on several classes of test problems. Finally, we have briefly listed some enhancements of interior–point algorithms for LP; their extensions to SQLP have been implemented in a new stand–alone C code currently under development.

Acknowledgments. We thank Pascal Gahinet, Arkadii Nemirovskii and Lieven Vandenberghe for supplying us with the LMI, the truss, and the clock mesh design problems respectively.

References

[1] I. Adler and F. Alizadeh. Primal–dual interior point algorithms for convex quadratically constrained and semidefinite optimization problems. Technical Report RRR 46–95, RUTCOR, Rutgers University, New Brunswick, NJ 08903, December 1995.

[2] F. Alizadeh, J.–P. A. Haeberly, M. V. Nayakkankuppam, M. L. Overton, and S. Schmieta. SDPpack User Guide (Version 0.9 Beta). Technical Report 737, Courant Institute of Mathematical Sciences, New York NY, June 1997. Also available from the SDPpack web page: http://www.cs.nyu.edu/faculty/overton/sdppack.

[3] F. Alizadeh, J.–P. A. Haeberly, and M. L. Overton. Complementarity and nondegeneracy in semidefinite programming. *Mathematical Programming (Series B)*, 1997.

[4] F. Alizadeh, J.–P. A. Haeberly, and M. L. Overton. Primal–dual interior–point methods for semidefinite programming: convergence rates, stability and numerical results. *SIAM Journal on Optimization*, 8(3):746–768, 1998.

[5] L. El Ghaoui and H. Lebret. Robust solutions to least–squares problems with uncertain data. *SIAM Journal on Matrix Analysis and Applications*, 18(4):1035–1064, 1997.

[6] L. Faybusovich. Linear systems in Jordan algebras and primal–dual interior–point algorithms. *Journal of Computational and Applied Mathematics*, 86:149–175, 1997.

[7] J. Gondzio. Multiple centrality corrections in a primal–dual method for linear programming. *Computational Optimization and Applications*, 6:137–156, 1996.

[8] J.–P. A. Haeberly, M. V. Nayakkankuppam, and M. L. Overton. Extending Mehrotra and Gondzio higher order corrections to mixed semidefinite–quadratic–linear programming. *Optimization Methods and Software.* To appear.

[9] C. Helmberg, F. Rendl, R. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996.

[10] M. Kojima, S. Shindoh, and S. Hara. Interior point methods for the monotone linear complementarity problem in symmetric matrices. *SIAM Journal on Optimization*, 7(1):86–125, 1997.

[11] Z.-Q. Luo, J. F. Sturm, and Z. Zhang. Duality and self–duality for conic convex programming. Technical report, Econometric Institute, Erasmus University, Rotterdam (The Netherlands), March 1996.

[12] S. Mehrotra. Higher order methods and their performance. Technical Report Technical Report 90–16R1, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, 1991.

[13] S. Mehrotra. On the implementation of a primal–dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 1992.

[14] Y. E. Nesterov and A. Nemirovskii. *Interior–Point Polynomial Algorithms in Convex Programming*, volume 13 of *Studies in applied mathematics.* SIAM, Philadelphia, PA, 1994.

[15] Y. E. Nesterov and M. J. Todd. Primal–dual interior–point methods for self–scaled cones. *SIAM Journal on Optimization*, 8:324–364, 1998.

[16] L. Vandenberghe, S. Boyd, and A. El Gamal. Optimal wire and transistor sizing for circuits with non-tree topology. In *Proceedings of the IEEE International Conference on Computer–Aided Design*, San Jose, CA, November 1997.

[17] S. J. Wright. *Primal–Dual Interior–Point Methods.* SIAM, Philadelphia, 1997.

Table 1: Legend for termination flag `tf`

`tf`	Description
–2	$(Z, z, z^{(0)})$ exceeds user–defined limit.
–1	$(F, f, f^{(0)})$ exceeds user–defined limit.
0	A solution of requested accuracy has been found.
1	$(Z, z, z^{(0)}$ or $(F, f, f^{(0)})$ is outside the cone or on its boundary.
2	F has a nonpositive eigenvalue.
3	Schur complement matrix is numerically singular.
4	Insufficient progress.
5	Primal or dual step length is too small.
6	Exceeded maximum number of iterations allowed.
7	Failed data validation checks.

Table 2: Randomly generated with semidefinite blocks only.

#	dim	m	`iter`	r_d	r_p	r_c	CPU secs	`tf`
1	630	60	14	–13	–14	–14	1.88e+01	0
2	2460	120	13	–13	–13	–12	1.57e+02	0
3	5490	180	15	–12	–13	–11	8.45e+02	0
4	9720	240	13	–12	–13	–12	2.13e+03	0
5	15150	300	14	–10	–13	–07	5.72e+03	4

Table 3: Randomly generated with quadratic blocks only.

#	dim	m	`iter`	r_d	r_p	r_c	CPU secs	`tf`
1	150	50	8	–14	–15	–10	5.40e–01	0
2	300	100	8	–13	–14	–13	1.29e+00	0
3	450	150	8	–13	–13	–10	3.48e+00	0
4	600	200	8	–12	–14	–11	7.00e+00	0
5	750	250	8	–13	–13	–10	1.28e+01	0

Table 4: Clock mesh design problems (`scalefac` = 1000): #1 is an 8×8 mesh, and #2 a 12×12 mesh

#	dim	m	`iter`	r_d	r_p	r_c	CPU secs	`tf`
1	3609	144	16	–13	–16	–09	1.92e+02	0
2	14989	312	21	–12	–16	–09	2.84e+03	0

Table 5: Unstructured robust least squares problems (`scalefac` = 1000).

#	dim	m	`iter`	r_d	r_p	r_c	CPU secs	`tf`
1	1103	102	7	–13	–13	–09	3.32e+00	0
2	2203	202	7	–12	–13	–09	1.74e+01	0
3	3303	302	7	–12	–13	–09	4.93e+01	0
4	4403	402	7	–12	–12	–09	1.02e+02	0
5	5503	502	7	–12	–12	–09	1.86e+02	0

Table 6: Truss topology design.

#	dim	m	`iter`	r_d	r_p	r_c	$\mathcal{D}(Z,z,z^{(0)})$	$\mathcal{P}(y)$	CPU secs	`tf`
1	19	6	10	–14	–15	–11	9.00e+00	9.00e+00	4.20e–01	0
2	331	58	12	–13	–14	–11	1.23e+02	1.23e+02	4.81e+00	0
3	91	27	15	–14	–15	–08	9.11e+00	9.11e+00	1.86e+00	4
4	37	12	11	–13	–16	–11	9.01e+00	9.01e+00	6.10e–01	0
5	1816	208	15	–10	–14	–09	1.33e+02	1.33e+02	7.28e+01	0
6	901	172	28	–07	–13	–07	9.01e+02	9.01e+02	5.85e+01	4
7	451	86	29	–06	–13	–08	9.00e+02	9.00e+02	2.40e+01	4
8	6271	496	18	–10	–14	–11	1.33e+02	1.33e+02	1.13e+03	0

Table 7: LMI problems from $\mathbf{H}_\infty$ control (`scalefac` = 1000).

#	d	m	`iter`	r_d	r_p	r_c	$\mathcal{D}(Z,z,z^{(0)})$	$\mathcal{P}(y)$	CPU secs	`tf`
1	41	13	12	–08	–12	–10	–2.03e+00	–2.03e+00	7.60e–01	2
2	51	13	14	–09	–11	–08	–1.10e+01	–1.10e+01	9.20e–01	4
3	51	13	16	–05	–12	–10	–5.70e+01	–5.69e+01	9.90e–01	1
4	51	13	15	–06	–10	–11	–2.75e+02	–2.75e+02	9.10e–01	2
5	51	13	17	–04	–12	–09	–3.63e+02	–3.63e+02	1.04e+00	1
6	51	13	33	–02	–11	–08	–4.49e+02	–4.49e+02	2.00e+00	2
7	51	13	11	–05	–11	–07	–3.91e+02	–3.91e+02	6.90e–01	1
8	51	13	15	–05	–11	–09	–1.16e+02	–1.16e+02	9.10e–01	1
9	51	13	17	–07	–13	–11	–2.36e+02	–2.36e+02	1.05e+00	2
10	66	21	27	–05	–07	–05	–1.09e+02	–1.09e+02	2.30e+00	–1
11	97	31	23	–07	–07	–04	–6.60e+01	–6.59e+01	3.23e+00	–1
12	120	43	25	–08	–07	–01	–2.00e–01	–1.78e–01	5.31e+00	–1
13	178	57	29	–04	–09	–02	–4.44e+01	–4.44e+01	8.73e+00	2
14	227	73	30	–07	–09	–03	–1.30e+01	–1.30e+01	1.39e+01	5
15	273	91	18	–06	–08	–02	–2.39e+01	–2.40e+01	2.23e+01	–1*
16	51	13	17	–07	–13	–11	–2.36e+02	–2.36e+02	1.08e+00	2