

# Software manual for SCIP 3.0<sup>1</sup>

Christian Zillober  
Department of Computer Science  
University of Bayreuth  
D-95440 Bayreuth  
Germany  
Christian.Zillober@Uni-Bayreuth.De

November 2004

## Abstract

This manual describes the implementation and usage of SCIP, version 3.0, a FORTRAN77 code for non-linear programming. It is the realization of the method of moving asymptotes and the method sequential convex programming. Most of the computational work has to be done in the solution of a large sequence of linear systems. A main feature is the ability to solve either systems of the dimension of the number of variables or systems of the dimension of the number of constraints. For both possibilities there is the option to use a dense Cholesky solver, a sparse Cholesky solver or an iterative method (conjugate gradients). The decision which one is the best can be done by the computer.

The most important new aspects of this version of SCIP are a built-in sparse Cholesky-solver, a warmstart possibility and the possibility to use SCIP efficiently within a dynamic storage allocation environment.

## 1 Introduction

The method of moving asymptotes (MMA) has been introduced in 1987 by K. Svanberg [9]. In 1993 the lack of missing global convergence has been removed in the method SCP (sequential convex programming; cf. [16]) by adding a line-search procedure. Both methods have been proven to be efficient tools in the context of structural optimization (cf. [8]), since displacement dependent constraints (e.g. stresses) are approximated very well. However, problems besides this context could also be solved efficiently. In 1995, weak convergence results have been proven in [10] without a line-search. In [16] the methods have been extended to a general mathematical programming framework. Box-constraints for the variables are kept in the model because they can be handled separately and they appear in many real-world applications.

Thus, we consider the following general nonlinear programming problem:

$$\begin{aligned} \min \quad & f(x), & x \in \mathbb{R}^n, \\ \text{s.t.} \quad & h_j(x) = 0, & j = 1, \dots, m_{eq}, \\ & h_j(x) \leq 0, & j = m_{eq} + 1, \dots, M, \\ & \underline{x}_i \leq x_i \leq \overline{x}_i, & i = 1, \dots, n. \end{aligned} \tag{P1}$$

The functions  $f$  and  $h_j$  ( $j = 1, \dots, M$ ) have to be defined only on  $X := \{x \mid \underline{x}_i \leq x_i \leq \overline{x}_i, i = 1, \dots, n\}$  and are assumed to be at least twice continuously differentiable in the interior of  $X$ . The feasible region is assumed to be non-empty.

The objective function of the problem will be approximated by a uniformly convex function, inequality constraints by convex functions, equality constraints by linear functions and the box-constraints will be allowed to shrink. Then, (P1) will be replaced by a sequence of subproblems using these approximations.

In the classical approaches, the subproblems have been solved applying a dual method (cf. [9]). This approach needs the repeated solution of dense linear systems of the dimension  $M \times M$  which can be excellent for the case of large  $n$  and small  $M$ . But for a larger number of constraints this approach is disadvantageous since the linear system matrices are dense, independent of the structure of the original problem.

---

<sup>1</sup>Manual version 3.0.1

In [15] the predictor-corrector interior point method to solve the subproblems has been introduced. The interior point method has several advantages. Firstly, it is also possible to reduce to  $M \times M$  systems, i.e. the advantage of the dual approach is covered. Secondly, it is possible to reduce to  $n \times n$  systems which is desirable for a small number of variables and a large number of constraints. This is the case in many sizing problems of structural optimization. As a third possibility, reduction to certain  $(n + M) \times (n + M)$  systems is possible which can be of interest for certain sparsity patterns. It is important to notice that sparsity in the original problem can be exploited by the interior point method. This is not the case for the dual approach. SCPIP has been described in a structural optimization framework in [17]. In [18] its ability to solve very large-scale problems arising in structural optimization and the solution of elliptic control problems has been shown.

The outline of the paper is as follows. In Section 2 the approximation scheme is introduced and the methods MMA and SCP are formulated. The interior point method to solve the subproblems is described in Section 3. Section 4 contains convergence results. Section 5 describes the major changes to version 2.2 of SCPIP. In Section 6 the interface of SCPIP is introduced and in Section 7 a sample calling program is described. Finally, Section 8 contains a note about scaling.

Sections 2-4 are only a summary of some papers mentioned above, containing the underlying methods and theory. The main purpose of this paper is to introduce into the software, Sections 6 and 7.

## 2 The MMA approximation

As for most methods of nonlinear programming, the method proposed here replaces the problem (P1) by a sequence of easier to solve subproblems. For this purpose, the model functions of (P1), i.e.  $f$  and  $h_j$  ( $j = 1, \dots, M$ ) are approximated by functions  $f^k$  and  $h_j^k$  ( $j = 1, \dots, M$ ) as follows. The motivation for this approach comes from the fact, that displacement dependent constraints in the context of structural optimization are approximated very well by this scheme, but it is generally applicable. More detailed explanations can be found in [9] and the references cited therein.

The objective function  $f$  is replaced by

$$f^k(x) := f(x^k) + \sum_{I_{0,k}^+} \left[ \frac{\partial f(x^k)}{\partial x_i} \left( \frac{(U_i^k - x_i^k)^2}{U_i^k - x_i} - (U_i^k - x_i^k) \right) + \tau_i^k \frac{(x_i - x_i^k)^2}{U_i^k - x_i} \right] - \sum_{I_{0,k}^-} \left[ \frac{\partial f(x^k)}{\partial x_i} \left( \frac{(x_i^k - L_i^k)^2}{x_i - L_i^k} - (x_i^k - L_i^k) \right) - \tau_i^k \frac{(x_i - x_i^k)^2}{x_i - L_i^k} \right]. \quad (1)$$

An inequality constraint  $h_j$  ( $j = m_{eq} + 1, \dots, M$ ) is replaced by

$$h_j^k(x) := h_j(x^k) + \sum_{I_{j,k}^+} \frac{\partial h_j(x^k)}{\partial x_i} \left( \frac{(U_i^k - x_i^k)^2}{U_i^k - x_i} - (U_i^k - x_i^k) \right) - \sum_{I_{j,k}^-} \frac{\partial h_j(x^k)}{\partial x_i} \left( \frac{(x_i^k - L_i^k)^2}{x_i - L_i^k} - (x_i^k - L_i^k) \right). \quad (2)$$

$I_{j,k}^+$ , ( $I_{j,k}^-$ , resp.) is the set of indices of all components with non-negative (negative) first partial derivatives at the expansion point  $x^k$ ,  $I_{j,k}^+ := \{i | \frac{\partial h_j(x^k)}{\partial x_i} \geq 0\}$ , ( $I_{j,k}^- := \{i | \frac{\partial h_j(x^k)}{\partial x_i} < 0\}$ ), where  $h_0 := f$ .

$f^k$  and  $h_j^k$  ( $j = m_{eq} + 1, \dots, M$ ) are defined on

$$D^k := \{x | L_i^k < x_i < U_i^k, i = 1, \dots, n\}.$$

$L_i^k$  and  $U_i^k$  are parameters to be chosen with  $L_i^k < x_i^k < U_i^k$ ,  $\tau_i^k$  are positive parameters,  $i = 1, \dots, n$ .

Equality constraints  $h_j$  ( $j = 1, \dots, m_{eq}$ ) are approximated by

$$h_j^k(x) := h_j(x^k) + \sum_{i=1}^n \frac{\partial h_j(x^k)}{\partial x_i} (x_i - x_i^k). \quad (3)$$

For the equality constraints no further restrictions on  $D^k$  apply.

This means, equality constraints are linearized in its usual sense, inequality constraints and the objective function are linearized with respect to transformed variables  $\frac{1}{U_i^k - x_i}$  and  $\frac{1}{x_i - L_i^k}$ . In the objective function an additional term is added. These so defined functions have the following properties:

$f^k$  and  $h_j^k$  are first-order approximations of  $f$  and  $h_j$ , resp. I.e.

$$\begin{aligned} f^k(x^k) &= f(x^k), \quad h_j^k(x^k) = h_j(x^k), \\ \nabla f^k(x^k) &= \nabla f(x^k), \quad \nabla h_j^k(x^k) = \nabla h_j(x^k), \end{aligned}$$

for all  $j = 1, \dots, M$ . Additionally:

$$\begin{aligned} h_j^k \quad (j = m_{eq} + 1, \dots, M) &\text{ are convex, i.e. can be strictly convex.} \\ f^k &\text{ is strictly convex.} \\ f^k \text{ and } h_j^k \quad (j = 1, \dots, M) &\text{ are separable.} \end{aligned} \tag{4}$$

One particular subproblem of (P1) looks then as follows:

$$\begin{aligned} \min \quad & f^k(x), \quad x \in \mathbb{R}^n, \\ \text{s.t.} \quad & h_j^k(x) = 0, \quad j = 1, \dots, m_{eq}, \\ & h_j^k(x) \leq 0, \quad j = m_{eq} + 1, \dots, M, \\ & \underline{x}_i' \leq x_i \leq \bar{x}_i', \quad i = 1, \dots, n. \end{aligned} \tag{P_{sub}^k}$$

where  $\underline{x}_i' := \max\{\underline{x}_i, x_i^k - \omega(x_i^k - L_i^k)\}$  and  $\bar{x}_i' := \min\{\bar{x}_i, x_i^k + \omega(U_i^k - x_i^k)\}$ ,  $\omega \in ]0; 1[$  fixed.

I.e.,  $\underline{x}_i \leq \underline{x}_i' \leq x_i^k \leq \bar{x}_i' \leq \bar{x}_i$ , where  $x^k$  denotes the expansion point. For further use we define

$$X' := \{x : \underline{x}_i' \leq x_i \leq \bar{x}_i', i = 1, \dots, n\}.$$

We will now explain how the asymptotes can be chosen.

**Definition 1** *A sequence of asymptotes is called feasible, if*

1.  $L_i^k \leq x_i^k - \xi$ ,  $U_i^k \geq x_i^k + \xi$ ,  $\xi$  is a positive constant, for all  $i = 1, \dots, n$  and  $k \geq 0$ .
2.  $L_i^k \geq L_{min}$ ,  $U_i^k \leq U_{max} \forall k \geq 0$ ,  $L_{min}$ ,  $U_{max}$  finite.

The first point of this definition prevents the curvature of the approximations from going to infinity. The second part prevents the approximations from going to close to linearity. This is necessary for the approximation of the objective. It is not necessary for the constraints. Therefore, one could use different asymptotes for the objective and the constraints but, for the matter of simplicity of the presentation we use the same asymptotes for both the objective and the constraints.

**Remark 1** *Individual asymptotes for each function that has to be approximated were introduced by Fleury [2]. But with very large problems in mind we do not proceed this way, because we would have to compute and store  $2n(M - m_{eq} + 1)$  values. Therefore, it only seems to be reasonable to use different asymptotes for the objective on the one hand and the set of inequality constraints on the other hand.*

For the rest of this paper we assume a feasible sequence of asymptotes.

We will now formulate algorithm MMA.

**Algorithm 1** *MMA (method of moving asymptotes)*

*Step 0 : Choose  $x^0 \in X$ ,  $y_j^0 \geq 0, j = 1, \dots, M$ ; compute  $f(x^0), \nabla f(x^0), h_j(x^0), \nabla h_j(x^0), j = 1, \dots, M$ ; let  $k := 0$ .*

*Step 1 : Compute  $L_i^k$  and  $U_i^k$  ( $i = 1, \dots, n$ ) by some scheme; define  $f^k(x), h_j^k(x), j = 1, \dots, M$*

(cf. (1),(2) and (3)).

Step 2 : Solve  $(P_{sub}^k)(x^k)$ ; let  $(x^{k+1}, y^{k+1})$  be the solution, where  $y^{k+1}$  denotes the corresponding vector of Lagrange multipliers.

Step 3 : If  $x^{k+1} = x^k$  stop;  $(x^k, y^k)$  is the solution.

Step 4 : Compute  $f(x^{k+1})$ ,  $\nabla f(x^{k+1})$ ,  $h_j(x^{k+1})$ ,  $\nabla h_j(x^{k+1})$ ,  $j = 1, \dots, M$ , let  $k := k + 1$ , goto step 1.

Taking the solution of  $(P_{sub}^k)$  as the new iteration point does not lead to a globally convergent algorithm. One would have to introduce strong assumptions on the model functions that are far away from reality. In order to globalize the convergence of algorithms, one possibility is to introduce so-called merit functions and perform a line-search with respect to such a function. The background is that stationary points of a problem like (P1) are linked to stationary points of the unrestricted merit function. The decrease in the merit function is a measure for the progress in the iteration. For our purposes we chose the augmented Lagrangian merit function which will be introduced below.

To simplify the notation we rewrite (P1) incorporating the box-constraints into the general inequality constraints:

$$\begin{aligned} \min \quad & f(x), \quad x \in \mathbb{R}^n, \\ \text{s.t.} \quad & h_j(x) = 0, \quad j = 1, \dots, m_{eq}, \\ & h_j(x) \leq 0, \quad j = m_{eq} + 1, \dots, m, \end{aligned} \tag{P2}$$

where  $m = M + 2n$ .

Now we introduce the augmented Lagrangian function.

**Definition 2** The augmented Lagrangian function  $\Phi_\rho : \mathbb{R}^{n+m} \rightarrow \mathbb{R}$  associated to (P2) is for a fixed set of parameters  $\rho_j > 0$  ( $j = 1, \dots, m$ ) defined by:

$$\begin{aligned} \Phi_\rho \begin{pmatrix} x \\ y \end{pmatrix} &= f(x) + \sum_{j=1}^{m_{eq}} \left( y_j h_j(x) + \frac{\rho_j}{2} h_j^2(x) \right) \\ &+ \sum_{j=m_{eq}+1}^m \begin{cases} y_j h_j(x) + \frac{\rho_j}{2} h_j^2(x), & \text{if } -\frac{y_j}{\rho_j} \leq h_j(x) \\ -\frac{y_j^2}{2\rho_j}, & \text{otherwise} \end{cases}. \end{aligned}$$

The motivation to choose the augmented Lagrangian merit function results from the following two well known statements:

- A point  $\begin{pmatrix} x^* \\ y^* \end{pmatrix}$  is stationary for  $\Phi_\rho$  if and only if  $\begin{pmatrix} x^* \\ y^* \end{pmatrix}$  is stationary for (P2).
- Under some regularity conditions there is a  $\bar{\rho} > 0 \in \mathbb{R}^m$  such that  $x^*$  is a local minimizer for  $\tilde{\Phi}_\rho(x) := \Phi_\rho \begin{pmatrix} x \\ y^* \end{pmatrix}$  for all  $\rho \geq \bar{\rho}$ .

For usage in the algorithm below we define:

$$\eta_i^k := \begin{cases} \left( \frac{\partial f(x^k)}{\partial x_i} + \tau_i^k \right) \frac{2U_i^k - z_i^k - x_i^k}{(U_i^k - z_i^k)^2}, & \text{if } \frac{\partial f(x^k)}{\partial x_i} \geq 0 \\ \left( \tau_i^k - \frac{\partial f(x^k)}{\partial x_i} \right) \frac{-2L_i^k + z_i^k + x_i^k}{(z_i^k - L_i^k)^2}, & \text{if } \frac{\partial f(x^k)}{\partial x_i} < 0 \end{cases}$$

and

$$\eta^k := \min_{i=1, \dots, n} \eta_i^k. \tag{5}$$

Another important ingredient for the SCP-algorithm is the procedure to modify the penalty parameter vector in case of violation of the descent property (cf. step 6 of Algorithm 3), which will be introduced now. The motivation for subsequently used factors can be found in [16].

Moreover, we define:

$$\begin{aligned} J &:= \{j \mid 1 \leq j \leq m_{eq}\} \cup \left\{ j \mid m_{eq} + 1 \leq j \leq m; -\frac{y_j}{\rho_j} \leq h_j(x) \right\}, \\ K &:= \{j \mid 1 \leq j \leq m; j \notin J\}. \end{aligned}$$

**Algorithm 2 (Update of penalty parameters)** Let  $\kappa_1 > 1$  (const.; e.g. 2),  $\kappa_2 > \kappa_1$  (const.; e.g. 10).

$j \in J$  : if  $h_j(x^k) > 0$  and  $\nabla h_j(x^k)^T(z^k - x^k) \neq 0$  or  $h_j(x^k) < 0$  and  $\nabla h_j(x^k)^T(z^k - x^k) > 0$  :

$$\rho_j := \min \left\{ \kappa_2 \rho_j; \max \left\{ \kappa_1 \rho_j; \left| \frac{2(v_j - y_j)}{h_j(x^k)} \right| \right\} \right\}$$

else:  $\rho_j := \kappa_1 \rho_j$

$j \in K$  : if  $v_j - y_j < 0$  :

$$\rho_j := \min \left\{ \kappa_2 \rho_j; \max \left\{ \kappa_1 \rho_j; \left| \frac{y_j(v_j - y_j)4m}{\eta^k(\delta^k)^2} \right| \right\} \right\}$$

else:  $\rho_j := \kappa_1 \rho_j$

$\kappa_1$  prevents a penalty parameter from converging slowly to an upper bound.  $\kappa_2$  prevents it from going to fast to large values. Notice, that it is not necessary to compute the lowest possible penalty parameter, it is only necessary (and reasonable from a numerical point of view) to exceed a certain (unknown) value.

We will now formulate the SCP-algorithm.

**Algorithm 3 SCP (sequential convex programming)**

Step 0 : Choose  $x^0 \in \mathbf{X}$ ,  $y_j^0 \in \mathbb{R}$ ,  $j = 1, \dots, m_{eq}$ ,  $y_j^0 \geq 0$ ,  $j = m_{eq} + 1, \dots, m$ ,  $0 < c < 1$  (e.g. 0.001),

$0 < \psi < 1$  (e.g. 0.5),  $\rho_j > 0$  (e.g. 1),  $j = 1, \dots, m$ ; compute  $f(x^0), \nabla f(x^0)$ ,  $h_j(x^0)$ ,  $\nabla h_j(x^0)$ ,

$j = 1, \dots, M$ ; let  $k := 0$ .

Step 1 : Compute  $L_i^k$  and  $U_i^k$  ( $i = 1, \dots, n$ ) by some scheme; define  $f^k(x)$ ,  $h_j^k(x)$ ,  $j = 1, \dots, M$

(cf. (1),(2) and (3)).

Step 2 : Solve  $(P_{sub}^k)(x^k)$ ; let  $(z^k, v^k)$  be the solution, where  $v^k$  denotes the corresponding vector of Lagrange multipliers.

Step 3 : If  $z^k = x^k$  stop;  $\begin{pmatrix} x^k \\ y^k \end{pmatrix}$  is the solution.

Step 4 : Let  $p^k := \begin{pmatrix} z^k - x^k \\ v^k - y^k \end{pmatrix}$ ,  $\delta^k := \|z^k - x^k\|$ ,  $\eta^k$  as defined in (5).

Step 5 : Compute  $\Phi_\rho \begin{pmatrix} x^k \\ y^k \end{pmatrix}$ ,  $\nabla \Phi_\rho \begin{pmatrix} x^k \\ y^k \end{pmatrix}$ ,  $\nabla \Phi_\rho \begin{pmatrix} x^k \\ y^k \end{pmatrix}^T p^k$ .

Step 6 : If  $\nabla \Phi_\rho \begin{pmatrix} x^k \\ y^k \end{pmatrix}^T p^k > -\frac{\eta^k(\delta^k)^2}{2}$  update the penalty parameters  $\rho_j$  ( $j = 1, \dots, m$ ) according to Algorithm 2 and goto step 5; otherwise, let  $\sigma := 1$ .

Step 7 : Compute  $f(x^k + \sigma(z^k - x^k))$ ,  $h_j(x^k + \sigma(z^k - x^k))$ ,  $j = 1, \dots, M$ ,  $\Phi_\rho \left( \begin{pmatrix} x^k \\ y^k \end{pmatrix} + \sigma p^k \right)$ .

Step 8 : (Armijo condition) If  $\Phi_\rho \begin{pmatrix} x^k \\ y^k \end{pmatrix} - \Phi_\rho \left( \begin{pmatrix} x^k \\ y^k \end{pmatrix} + \sigma p^k \right) < -c\sigma \nabla \Phi_\rho \begin{pmatrix} x^k \\ y^k \end{pmatrix}^T p^k$  let  $\sigma := \sigma \cdot \psi$ , goto step 7;

else let  $\sigma^k := \sigma$ .

Step 9 : Let  $\begin{pmatrix} x^{k+1} \\ y^{k+1} \end{pmatrix} := \begin{pmatrix} x^k \\ y^k \end{pmatrix} + \sigma^k p^k$ ,  $k := k + 1$ .

Step 10: Compute  $\nabla f(x^k)$ ,  $\nabla h_j(x^k)$ ,  $j = 1, \dots, M$ , goto step 1.

### 3 Predictor-corrector interior point method

It is beyond the scope of this paper to go into the convergence theory of interior point methods for convex programming. This work has been done mainly by Nesterov and Nemirovskii ([7]) for classes of interior point methods. One of the first practical methods for nonlinear convex programming appeared in [4]. Svanberg ([11]) developed in parallel to [14] a primal dual interior point method for the solution of the MMA subproblems and modified MMA subproblems.

We chose the predictor-corrector interior point method for the solution of our convex programs. It has been proven to be one of the most efficient interior point methods for linear programming (see [5] and [6] for the

development of the method and [5] for a numerical evaluation). The basic structure of our problem does not differ very much from a linear program.

We proceed from problem ( $P_{sub}^k$ ) and modify it in order to prepare it for the application of the predictor-corrector method. For that purpose we add nonnegative slack variables wherever inequalities appear:

$$\begin{aligned}
\min \quad & f^k(x), & x \in \mathbb{R}^n, \\
\text{s.t.} \quad & h_j^k(x) = 0, & j = 1, \dots, m_{eq}, \\
& h_j^k(x) + c_{j-m_{eq}} = 0, & j = m_{eq} + 1, \dots, M, \\
& -c_j + r_j = 0, & j = 1, \dots, M - m_{eq}, \\
& \underline{x}_i' - x_i + s_i = 0, & i = 1, \dots, n, \\
& x_i - \bar{x}_i' + t_i = 0, & i = 1, \dots, n, \\
& r, s, t \geq 0.
\end{aligned} \tag{6}$$

The slacks  $s$  and  $t$  are introduced because variable  $x$  is formally free and allowed to violate its bounds. Slack  $r$  is not absolutely necessary, but with its usage we allow  $c$  and in consequence the dual of the original subproblem constraints to become 0. If we do not use slack  $r$  we need  $c$  and the dual variable  $y$  to be positive. The computational amount is touched only slightly by this modification.

The positivity is only demanded for the new introduced variables  $r, s$  and  $t$ , variables that do not appear outside the subproblems, i.e. in the main loop.

For the next step we add barrier terms corresponding to the slack variables and build the Lagrangian of the so modified problem:

$$\begin{aligned}
L_\mu(x, y_{eq}, y_{ie}, c, r, s, t, d_r, d_s, d_t) = & f^k(x) - \mu \sum_{j=1}^{M-m_{eq}} \ln r_j - \mu \sum_{i=1}^n \ln s_i - \mu \sum_{i=1}^n \ln t_i + y_{eq}^T h_{eq}^k(x) + y_{ie}^T (h_{ie}^k(x) + c) \\
& + d_r^T (-c + r) + d_s^T (\underline{x}' - x + s) + d_t^T (x - \bar{x}' + t),
\end{aligned}$$

where  $h_{eq}^k := (h_1^k, \dots, h_{m_{eq}}^k)^T$ ,  $h_{ie}^k := (h_{m_{eq}+1}^k, \dots, h_M^k)^T$  and  $y_{eq}, y_{ie}, d_r, d_s, d_t$  are the dual variable vectors to the corresponding constraints. The combination of  $y_{eq}$  and  $y_{ie}$  corresponds to the vector  $y$  of the last section, i.e.  $y = (y_{eq}, y_{ie})^T$ .  $\mu$  is a positive homotopy parameter. Formally, in this formulation all variables are free, the barrier formulation, however, needs  $r, s$  and  $t$  to be positive.

The necessary optimality condition  $\nabla L_\mu = 0$  reads then:

$$\begin{aligned}
\nabla_x : \quad & \nabla f^k(x) + J_{eq} y_{eq} + J_{ie} y_{ie} - d_s + d_t = 0, \\
\nabla_{y_{eq}} : \quad & h_{eq}^k(x) = 0, \\
\nabla_{y_{ie}} : \quad & h_{ie}^k(x) + c = 0, \\
\nabla_c : \quad & y_{ie} - d_r = 0, \\
\nabla_r : \quad & D_r R e - \mu e = 0, \\
\nabla_s : \quad & D_s S e - \mu e = 0, \\
\nabla_t : \quad & D_t T e - \mu e = 0, \\
\nabla_{d_r} : \quad & -c + r = 0, \\
\nabla_{d_s} : \quad & \underline{x}' - x + s = 0, \\
\nabla_{d_t} : \quad & x - \bar{x}' + t = 0,
\end{aligned} \tag{S0}$$

where  $R = \text{diag}(r_1, \dots, r_{M-m_{eq}})$ ;  $S, T, D_r, D_s, D_t$  resp.,  $e = (1, 1, \dots, 1)^T$  in the appropriate dimension,

$$J_{eq} = \begin{pmatrix} \frac{\partial h_1^k(x)}{\partial x_1} & \frac{\partial h_2^k(x)}{\partial x_1} & \dots & \frac{\partial h_{m_{eq}}^k(x)}{\partial x_1} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \frac{\partial h_1^k(x)}{\partial x_n} & \frac{\partial h_2^k(x)}{\partial x_n} & \dots & \frac{\partial h_{m_{eq}}^k(x)}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{n, m_{eq}} \text{ and}$$

$$J_{ie} = \begin{pmatrix} \frac{\partial h_{m_{eq}+1}^k(x)}{\partial x_1} & \frac{\partial h_{m_{eq}+2}^k(x)}{\partial x_1} & \cdots & \frac{\partial h_M^k(x)}{\partial x_1} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \frac{\partial h_{m_{eq}+1}^k(x)}{\partial x_n} & \frac{\partial h_{m_{eq}+2}^k(x)}{\partial x_n} & \cdots & \frac{\partial h_M^k(x)}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{n, M-m_{eq}}.$$

The linear system to compute a Newton step for the solution of this systems reads as follows:

$$\begin{pmatrix} \nabla_{xx}L & J_{eq} & J_{ie} & & & & & & & & -I & I \\ J_{eq}^T & & & & & & & & & & & & \\ J_{ie}^T & & & I & & & & & & & & & \\ & & & & I & & & & & & -I & & \\ & & & & & D_r & & & & & R & & \\ & & & & & & D_s & & & & & S & \\ & & & & & & & D_t & & & & & T \\ & & & & & & & & -I & I & & & \\ -I & & & & & & & & & I & & & \\ I & & & & & & & & & & I & & \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y_{eq} \\ \Delta y_{ie} \\ \Delta c \\ \Delta r \\ \Delta s \\ \Delta t \\ \Delta d_r \\ \Delta d_s \\ \Delta d_t \end{pmatrix} = -\nabla L_\mu, \quad (S1)$$

where  $\nabla_{xx}L = \nabla^2 f^k(x) + \frac{d}{dx}J_{eq}y_{eq} + \frac{d}{dx}J_{ie}y_{ie}$ . We will discuss this term now a little bit more detailed. It is crucial for the effectiveness of the interior point method. For the equalities the matrix  $J_{eq}$  is constant and thus  $\frac{d}{dx}J_{eq}y_{eq} = 0$ . For the inequalities the term does not vanish:

$$J_{ie}y_{ie} = \begin{pmatrix} \sum_{j=m_{eq}+1}^M \frac{\partial h_j^k(x)}{\partial x_1} (y_{ie})_{j-m_{eq}} \\ \cdot \\ \cdot \\ \sum_{j=m_{eq}+1}^M \frac{\partial h_j^k(x)}{\partial x_n} (y_{ie})_{j-m_{eq}} \end{pmatrix} \implies \frac{\partial (J_{ie}y_{ie})_k}{\partial x_i} = \frac{\partial}{\partial x_i} \left( \sum_{j=m_{eq}+1}^M \frac{\partial h_j^k(x)}{\partial x_k} (y_{ie})_{j-m_{eq}} \right) = \sum_{j=m_{eq}+1}^M \frac{\partial^2 h_j^k(x)}{\partial x_k \partial x_i} (y_{ie})_{j-m_{eq}}.$$

Since the functions  $h_j^k$  ( $j = m_{eq} + 1, \dots, M$ ) are separable, we have:  $\frac{\partial^2 h_j^k(x)}{\partial x_k \partial x_i} = 0$  if  $k \neq i$ , i.e. these Hessians are diagonal. Thus,  $\nabla_{xx}L$  is completely determined since

$$\frac{d}{dx}J_{ie}y_{ie} = \text{diag} \left( \sum_{j=m_{eq}+1}^M \frac{\partial^2 h_j^k(x)}{\partial x_k^2} (y_{ie})_{j-m_{eq}} \right)_{k=1, \dots, n}$$

and  $f^k$  is separable, too. From a computational point of view,  $\nabla_{xx}L$  is in consequence not a  $n \times n$  matrix, but a  $n$  vector. A central point considering problems with a large number of variables. Remember, that each component of  $\nabla_{xx}L$  is strictly positive due to the convexity properties of the included functions. For the purpose of computational efficiency, let us have a look on the nonzero structure of the matrix  $J_{ie}$ . For one particular component of the gradient of a constraint we have (cf. (2)):

$$\frac{\partial h_j^k(x)}{\partial x_i} = \begin{cases} \frac{\partial h_j(x^k)}{\partial x_i} \frac{(U_i^k - x_i^k)^2}{(U_i^k - x_i^k)^2}, & \text{if } \frac{\partial h_j(x^k)}{\partial x_i} \geq 0 \\ \frac{\partial h_j(x^k)}{\partial x_i} \frac{(x_i^k - L_i^k)^2}{(x_i^k - L_i^k)^2}, & \text{if } \frac{\partial h_j(x^k)}{\partial x_i} < 0 \end{cases}. \quad (7)$$

Since the second term is in both cases always ensured to be strictly positive, the nonzero structure of  $J_{ie}$  in an arbitrary point is identical to the nonzero structure of the original constraints in the current main iteration point  $x^k$ . The same is true for the matrix  $J_{eq}$ . That means, sparsity in the original problem is preserved for the subproblems.

We will now briefly state the principles of the predictor-corrector interior point algorithm, which has been chosen for the solution of  $(P_{sub}^k)$ .

**Algorithm 4** *Predictor-corrector algorithm*

*Step 0:* Choose  $x^0 \in X'$ ,  $c^0 \geq 0$ ,  $r^0, s^0, t^0, d_r^0, d_s^0, d_t^0 > 0$ ,  $y_{ie}^0 \geq 0$ ,  $y_{eq}^0, \epsilon$  (solution tolerance). Let  $k := 0$ .

*Step 1:* Let  $\mu^0 := \max_{\substack{j=1, \dots, M-m_{eq} \\ i=1, \dots, n}} \{r_j^0(d_r^0)_j, s_i^0(d_s^0)_i, t_i^0(d_t^0)_i\}$ .

*Step 2:* Solve (S1) for the predictor step (cf. Section 3.1).

*Step 3:* Let  $\mu^{k+1} = \mu^{k+1}(\mu^k, \Delta x, \dots, \Delta d_t)$  (see below).

*Step 4:* Solve (S1) for the corrector step (cf. Section 3.2) using the results of the predictor step and  $\mu^{k+1}$ .

*Step 5:* Compute independent primal and dual stepsizes: Let  $\kappa := 10 \cdot \epsilon_{mach}$ , where  $\epsilon_{mach}$  is the machine precision.

$$\begin{aligned} \tilde{\delta}_{ip,primal} &:= \min_{\substack{j=1, \dots, M-m_{eq} \\ i=1, \dots, n}} \left\{ \left( \frac{r_j^k - \kappa}{-\Delta r_j}, \text{ where } r_j^k + \Delta r_j < \kappa \right), \right. \\ &\quad \left( \frac{s_i^k - \kappa}{-\Delta s_i}, \text{ where } s_i^k + \Delta s_i < \kappa \right), \\ &\quad \left. \left( \frac{t_i^k - \kappa}{-\Delta t_i}, \text{ where } t_i^k + \Delta t_i < \kappa \right) \right\}; \\ \tilde{\delta}_{ip,dual} &:= \min_{\substack{j=1, \dots, M-m_{eq} \\ i=1, \dots, n}} \left\{ \left( \frac{(d_r^k)_j - \kappa}{-(\Delta d_r)_j}, \text{ where } (d_r^k)_j + (\Delta d_r)_j < \kappa \right), \right. \\ &\quad \left( \frac{(d_s^k)_i - \kappa}{-(\Delta d_s)_i}, \text{ where } (d_s^k)_i + (\Delta d_s)_i < \kappa \right), \\ &\quad \left. \left( \frac{(d_t^k)_i - \kappa}{-(\Delta d_t)_i}, \text{ where } (d_t^k)_i + (\Delta d_t)_i < \kappa \right) \right\}. \\ \delta_{ip,primal} &= \begin{cases} 1, & \text{if } \tilde{\delta}_{ip,primal} \text{ is not defined} \\ \min\{1, 0.99995 \cdot \tilde{\delta}_{ip,primal}\}, & \text{else} \end{cases}, \\ \delta_{ip,dual} &= \begin{cases} 1, & \text{if } \tilde{\delta}_{ip,dual} \text{ is not defined} \\ \min\{1, 0.99995 \cdot \tilde{\delta}_{ip,dual}\}, & \text{else} \end{cases}. \\ \delta_{primal} &:= \min_{i=1, \dots, n} \left\{ \delta_{ip,primal}, \left( \frac{x_i^k - \underline{x}_i'}{-\Delta x_i}, \text{ where } x_i^k + \Delta x_i < \underline{x}_i' \right), \right. \\ &\quad \left. \left( \frac{x_i^k - \overline{x}_i'}{-\Delta x_i}, \text{ where } x_i^k + \Delta x_i > \overline{x}_i' \right) \right\}; \\ \delta_{dual} &:= \min_{j=1, \dots, M-m_{eq}} \left\{ \delta_{ip,dual}, \left( \frac{(y_{ie}^k)_j}{-(\Delta y_{ie})_j}, \text{ where } (y_{ie}^k)_j + (\Delta y_{ie})_j < 0 \right) \right\}. \end{aligned}$$

*Step 6:* Update:

$$\begin{aligned} x^{k+1} &:= x^k + \delta_{primal} \cdot \Delta x, & y_{eq}^{k+1} &:= y_{eq}^k + \delta_{dual} \cdot \Delta y_{eq}, & y_{ie}^{k+1} &:= y_{ie}^k + \delta_{dual} \cdot \Delta y_{ie}, \\ c^{k+1} &:= c^k + \delta_{primal} \cdot \Delta c, & r^{k+1} &:= r^k + \delta_{primal} \cdot \Delta r, & s^{k+1} &:= s^k + \delta_{primal} \cdot \Delta s, \\ t^{k+1} &:= t^k + \delta_{primal} \cdot \Delta t, & d_r^{k+1} &:= d_r^k + \delta_{dual} \cdot \Delta d_r, & d_s^{k+1} &:= d_s^k + \delta_{dual} \cdot \Delta d_s, \\ d_t^{k+1} &:= d_t^k + \delta_{dual} \cdot \Delta d_t. \end{aligned}$$

Let  $k := k + 1$ . If  $\|\nabla L_0\| < \epsilon$  stop, else goto step 2.

**Remark 2** • For the initialization in Step 0 one can take the values of the previous main (MMA-/SCP-) iteration. One possibly has to take care of the positivity condition.

- The factor 0.99995 prevents the components of the critical vectors from getting too close to 0.
- $\delta_{ip,primal}$  and  $\delta_{ip,dual}$  are the stepsizes to ensure the interior point condition.  $\delta_{primal}$  takes additionally the definition of the approximations into account where we have the stronger demand  $x \in X'$ .  $\delta_{dual}$  ensures that  $y_{ie} \geq 0$  throughout, to guarantee the positivity of the Hessian of the Lagrangian.

We will now consider step 2 of the algorithm above, i.e., we will analyze how the linear system (S1) can be solved for the predictor step in practice.

## Update of the homotopy parameter

The homotopy parameter  $\mu$  of step 3 of the predictor-corrector interior point algorithm (Alg. 4) is updated as:

$$\mu^{k+1} := \frac{1}{n + n + M - m_{eq}} \left[ (s^k + \delta'_{primal} \cdot \Delta s)^T (d_s^k + \delta'_{dual} \cdot \Delta d_s) + \right. \\ \left. (t^k + \delta'_{primal} \cdot \Delta t)^T (d_t^k + \delta'_{dual} \cdot \Delta d_t) + \right. \\ \left. (r^k + \delta'_{primal} \cdot \Delta r)^T (d_r^k + \delta'_{dual} \cdot \Delta d_r) \right]$$

where  $\Delta s, \Delta t, \Delta r, \Delta d_s, \Delta d_t$  and  $\Delta d_r$  are the results of the predictor step and  $\delta'_{primal}$  and  $\delta'_{dual}$  are intermediate stepsizes computed in the same way as in step 5 of Algorithm 4.

This update rule is well approved in linear programming, cf. [13] or [12], and works also very well in our context.

### 3.1 Predictor step

#### 3.1.1 Indefinite linear system

The predictor step is only used to estimate a proper value for the homotopy parameter  $\mu$  and to estimate the nonlinearity terms for the corrector step. It corresponds to an affine scaling step.

For the predictor step we solve the linear system (S1), but **without** the terms containing  $\mu$  in the right hand side.

For this purpose, we will now eliminate some variables of the linear system in order to get one that can be handled easier. Notice, that all matrices in this section that are inverted are positive diagonal matrices and thus are invertable.

Choose equation 5 of (S1):  $D_r \Delta r + R \Delta d_r = -D_r R e$ ;

$$\implies \Delta d_r = -d_r - R^{-1} D_r \Delta r. \quad (8)$$

Analogously we get for the equations 6 and 7 of (S1):

$$\Delta d_s = -d_s - S^{-1} D_s \Delta s, \quad (9)$$

$$\Delta d_t = -d_t - T^{-1} D_t \Delta t. \quad (10)$$

Substitute in equation 1 of (S1):

$$\nabla_{xx} L \Delta x + J_{eq} \Delta y_{eq} + J_{ie} \Delta y_{ie} - \Delta d_s + \Delta d_t = -\nabla f^k(x) - J_{eq} y_{eq} - J_{ie} y_{ie} + d_s - d_t;$$

$$\implies \nabla_{xx} L \Delta x + J_{eq} \Delta y_{eq} + J_{ie} \Delta y_{ie} + S^{-1} D_s \Delta s - T^{-1} D_t \Delta t = -\nabla f^k(x) - J_{eq} y_{eq} - J_{ie} y_{ie}. \quad (11)$$

Substitute in equation 4 of (S1):

$$\Delta y_{ie} + R^{-1} D_r \Delta r = -y_{ie}. \quad (12)$$

Further eliminations in the equations 8, 9 and 10 of (S1) yield:

$$\Delta r = \Delta c + c - r, \quad (13)$$

$$\Delta s = \Delta x + x - \underline{x}' - s, \quad (14)$$

$$\Delta t = -\Delta x - x + \bar{x}' - t. \quad (15)$$

Substitution in (11) and (12) yields:

$$(\nabla_{xx} L + S^{-1} D_s + T^{-1} D_t) \Delta x + J_{eq} \Delta y_{eq} + J_{ie} \Delta y_{ie} = \\ -\nabla f^k(x) - J_{eq} y_{eq} - J_{ie} y_{ie} + d_s - d_t - S^{-1} D_s (x - \underline{x}') + T^{-1} D_t (\bar{x}' - x) =: \gamma_1, \quad (16)$$

$$\Delta y_{ie} + R^{-1} D_r \Delta c = -y_{ie} + d_r - R^{-1} D_r c. \quad (17)$$

The last equation is used to eliminate  $\Delta c$ :

$$\Delta c = D_r^{-1} R (-y_{ie} + d_r - R^{-1} D_r c - \Delta y_{ie}) = D_r^{-1} R (-y_{ie} + d_r - \Delta y_{ie}) - c. \quad (18)$$

This has to be inserted in equation 3 of (S1):

$$J_{ie}^T \Delta x - D_r^{-1} R \Delta y_{ie} = -h_{ie}^k(x) - D_r^{-1} R(-y_{ie} + d_r) =: \gamma_2. \quad (19)$$

Putting all together we get the following linear system:

$$\begin{pmatrix} \nabla_{xx} L + S^{-1} D_s + T^{-1} D_t & J_{eq} & J_{ie} \\ J_{eq}^T & & \\ J_{ie}^T & & -D_r^{-1} R \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y_{eq} \\ \Delta y_{ie} \end{pmatrix} = \begin{pmatrix} \gamma_1 \\ -h_{eq}^k(x) \\ \gamma_2 \end{pmatrix}. \quad (S2)$$

This system is indefinite and has full rank provided that  $J_{eq}$  and  $J_{ie}$  are of full rank (remember  $\nabla_{xx} L > 0$ ). It's dimension is  $(n + M) \times (n + M)$ . The upper left and lower right part are diagonal. Thus, the matrix can be considered as sparse. Additional sparsity of the Jacobians improves the situation. It is now possible to solve this system with a sparse indefinite linear system solver.

SCPIP does not support this approach. We will proceed with two more possibilities and will show which approach is favorable in which situation.

### 3.1.2 Positive definite system of dimension $M \times M$

First of all we define

$$\Theta := \nabla_{xx} L + S^{-1} D_s + T^{-1} D_t.$$

We proceed from (S2) by eliminating  $\Delta x$  from equation 1:

$$\Delta x = \Theta^{-1}(\gamma_1 - J_{eq} \Delta y_{eq} - J_{ie} \Delta y_{ie}). \quad (20)$$

Inserting in the second equation yields:

$$J_{eq}^T \Theta^{-1} J_{eq} \Delta y_{eq} + J_{eq}^T \Theta^{-1} J_{ie} \Delta y_{ie} = h_{eq}^k(x) + J_{eq}^T \Theta^{-1} \gamma_1. \quad (21)$$

Inserting in the third equation yields:

$$J_{ie}^T \Theta^{-1} J_{eq} \Delta y_{eq} + (J_{ie}^T \Theta^{-1} J_{ie} + D_r^{-1} R) \Delta y_{ie} = J_{ie}^T \Theta^{-1} \gamma_1 - \gamma_2. \quad (22)$$

Together we have

$$\begin{pmatrix} J_{eq}^T \Theta^{-1} J_{eq} & J_{eq}^T \Theta^{-1} J_{ie} \\ J_{ie}^T \Theta^{-1} J_{eq} & J_{ie}^T \Theta^{-1} J_{ie} + D_r^{-1} R \end{pmatrix} \begin{pmatrix} \Delta y_{eq} \\ \Delta y_{ie} \end{pmatrix} = \begin{pmatrix} h_{eq}^k(x) + J_{eq}^T \Theta^{-1} \gamma_1 \\ J_{ie}^T \Theta^{-1} \gamma_1 - \gamma_2 \end{pmatrix}. \quad (S3)$$

This system is positive definite. It's dimension is  $M \times M$ . If  $J_{eq}$  and  $J_{ie}$  are sparse then we can hope that the same is true for the matrix in (S3). Unfortunately, this cannot be ensured. In the context of linear programming this case was extensively examined. If one of the Jacobians has at least one dense column then the matrix is dense, too. On the other hand, there are techniques to overcome this situation by splitting dense columns. It is beyond the scope of this paper to outline these techniques. The reader is referred to the book of Wright [13] and the references cited therein. Notice, that it is worthwhile to think about sparse positive definite systems if the Jacobians are dense overall. Then, of course, the matrix in (S3) is also dense. In SCPIP there are three possibilities to solve the linear systems (S3). Variable SPSTRAT has to be set to 1.

- A dense Cholesky solver can be chosen. This is favorable in case of a small or medium number of constraints ( $M$ ). In this case, variable LINSYS has to be set to 1.
- A sparse Cholesky solver can also be chosen. This should be the best approach (within this three) for large  $M$  and sparse Jacobians  $J_{ie}$  and  $J_{eq}$ , if it does not lead to a dense matrix in (S3). LINSYS has to be set to 2.
- The third possibility is a conjugate gradient (cg) solver, i.e. an iterative solution method. This is favorable in case of large  $M$  and dense  $J_{ie}$  and  $J_{eq}$ , or a large  $M$  and sparse Jacobians if this leads to a dense matrix in (S3). LINSYS has to be set to 3.

### 3.1.3 Positive definite system of dimension $n \times n$

For this approach we have to assume the absence of equality constraints. That means, we proceed from equation (S2) by deleting row 2 and column 2 and eliminate then  $\Delta y_{ie}$  instead of  $\Delta x$ :

$$\Delta y_{ie} = R^{-1} D_r (J_{ie}^T \Delta x - \gamma_2). \quad (23)$$

Inserting in the first equation yields:

$$(\Theta + J_{ie} R^{-1} D_r J_{ie}^T) \Delta x = \gamma_1 + J_{ie} R^{-1} D_r \gamma_2. \quad (S4)$$

As in the last subsection, this system is positive definite. It's dimension is  $n \times n$ . The remarks about sparsity are also valid here replacing columns by rows. The impact of dense rows or columns is examined more detailed in the book of Vanderbei [12].

In SCIP there are three possibilities to solve the linear systems (S4). Variable SPSTRAT has to be set to 2. This is equivalent to the last subsection.

- A dense Cholesky solver can be chosen. This is favorable in case of a small or medium number of variables ( $n$ ). In this case, variable LINSYS has to be set to 1.
- A sparse Cholesky solver can be chosen. This should be the best approach (within this three) for large  $n$  and a sparse Jacobian  $J_{ie}$ , if it does not lead to a dense matrix in (S4). (LINSYS = 2)
- The third possibility is a cg solver. This is favorable in case of large  $n$  and a dense  $J_{ie}$ , or a large  $n$  and a sparse  $J_{ie}$  if it leads to a dense matrix in (S4). (LINSYS = 3)

### 3.1.4 Comparison of the three approaches

We will now summarize the properties of the three approaches of this section and compare it among each other as well as with the classical dual approach. The time that is spent for the repeated solution of the linear systems is crucial for the computing time of the solution of a subproblem and thus also important for the solution of the overall algorithm (together with the computing amount for the function and gradient evaluations). Notice, that the method to solve the subproblems has theoretically no influence on the number of main iterations (if we solve the subproblems up to exact optimality).

For the comparison we have to mention that approach (S4) does not work for equality constraints and that the dual approach has never been worked out for this situation. So the comparison is only fair for problems without equality constraints.

For the dual approach, finally a concave maximization problem with simple nonnegativity constraints has to be solved. But this problem is only once continuously differentiable. The second derivative has jumps and is therefore only almost everywhere existent. These jumps are not only theoretically important, they are also numerically relevant (cf. [1]). Thus, we compare a BFGS based maximization algorithm. The term sparsity

Table 1: Comparison of linear systems

	dual	(S2)	(S3)	(S4)
Dimension	$M \times M$	$(n + M) \times (n + M)$	$M \times M$	$n \times n$
Definite?	positive	indefinite	positive	positive
Sparsity	dense	sparse	?	?

in the table means the case if the Jacobians are sparse.

Let us summarize these properties: To the experience of the author, the dual approach is not relevant any longer. The advantage, that in the case of few constraints one can reduce the computing amount to the solution of  $M \times M$  systems is also governed by approach (S3). But the interior point approaches are much more stable. In case of large  $n$  and small  $M$  approach (S3) is favorable, as well as approach (S4) vice versa, if  $n$  is small and  $M$  is large. The choice is not so easy if  $n$  and  $M$  are in the same range. Then approaches (S3) and (S4) are comparable. It depends on the sparsity of  $J_{eq}$  and  $J_{ie}$ . In cases, where a sparse Jacobian causes dense positive definite systems, approach (S2) is also a practical alternative.

Which specific linear system solver should be chosen is a difficult question. For small and medium dimensions the dense Cholesky solver is usually the best choice. The sparse Cholesky option is the best approach for

large dimensions and sparse matrices. In case of large dense matrices or if the Cholesky decomposition needs too much storage, the iterative solver is the best possibility. At which moment it is the best to switch from one possibility to the other is hard to predict. It is matter of further research and numerical experience.

### 3.2 Corrector step

For the corrector step we take the equation (S0), replace the vector  $(x, \dots, d_t)$  by the new point  $(x + \Delta x, \dots, d_t + \Delta d_t)$  and put all zero order and second order terms with the unknowns into the right hand side. The right hand side of the system (S1) changes then as follows (w.r.t. the predictor step): The  $\mu$ -terms are added, as they are part of the original gradient of the Lagrangian and the nonlinear terms mentioned above are estimated by the results of the predictor step. The matrix of (S1) remains unchanged. This makes this approach attractive because the matrix has to be factorized only once using a direct solver.

The following factors have to be added to the right hand side of (S1):

$$\text{Component 1: } - \left( \sum_{j=m_{eq}+1}^M \frac{\partial^2 h_j^k}{\partial x_k^2} \Delta x_k (\Delta y_{ie})_{j-m_{eq}} \right)_{k=1, \dots, n},$$

$$\text{Component 5: } + \mu e - \underline{\Delta r} \underline{\Delta d_r},$$

$$\text{Component 6: } + \mu e - \underline{\Delta s} \underline{\Delta d_s},$$

$$\text{Component 7: } + \mu e - \underline{\Delta t} \underline{\Delta d_t},$$

where the underlined terms stand for the results of the predictor step.

The same eliminations, that have been done for the predictor system, are applied to the corrector system. Firstly, equations 5, 6 and 7 of (S1):

$$\Delta d_r = -d_r + R^{-1}(\mu e - D_r \Delta r - \underline{\Delta r} \underline{\Delta d_r}), \quad (24)$$

$$\Delta d_s = -d_s + S^{-1}(\mu e - D_s \Delta s - \underline{\Delta s} \underline{\Delta d_s}), \quad (25)$$

$$\Delta d_t = -d_t + T^{-1}(\mu e - D_t \Delta t - \underline{\Delta t} \underline{\Delta d_t}). \quad (26)$$

Substitute in equation 1 of (S1):

$$\begin{aligned} & \nabla_{xx} L \Delta x + J_{eq} \Delta y_{eq} + J_{ie} \Delta y_{ie} + S^{-1} D_s \Delta s - T^{-1} D_t \Delta t = \\ & -\nabla f^k(x) - J_{eq} y_{eq} - J_{ie} y_{ie} + S^{-1}(\mu e - \underline{\Delta s} \underline{\Delta d_s}) - T^{-1}(\mu e - \underline{\Delta t} \underline{\Delta d_t}) - \left( \sum_{j=m_{eq}+1}^M \frac{\partial^2 h_j^k}{\partial x_k^2} \Delta x_k (\Delta y_{ie})_{j-m_{eq}} \right)_{k=1, \dots, n}. \end{aligned} \quad (27)$$

Substitute in equation 4 of (S1):

$$\Delta y_{ie} + R^{-1} D_r \Delta r = -y_{ie} + R^{-1}(\mu e - \underline{\Delta r} \underline{\Delta d_r}). \quad (28)$$

Further eliminations in the equations 8, 9 and 10 of (S1) yield (as for the predictor step):

$$\Delta r = \Delta c + c - r, \quad (29)$$

$$\Delta s = \Delta x + x - \underline{x}' - s, \quad (30)$$

$$\Delta t = -\Delta x - x + \underline{x}' - t. \quad (31)$$

Substitution in equations (27) and (28) yields:

$$\begin{aligned} & (\nabla_{xx} L + S^{-1} D_s + T^{-1} D_t) \Delta x + J_{eq} \Delta y_{eq} + J_{ie} \Delta y_{ie} = \\ & -\nabla f^k(x) - J_{eq} y_{eq} - J_{ie} y_{ie} + d_s - d_t - S^{-1} D_s (x - \underline{x}') + T^{-1} D_t (\underline{x}' - x) \\ & + S^{-1}(\mu e - \underline{\Delta s} \underline{\Delta d_s}) - T^{-1}(\mu e - \underline{\Delta t} \underline{\Delta d_t}) - \left( \sum_{j=m_{eq}+1}^M \frac{\partial^2 h_j^k}{\partial x_k^2} \Delta x_k (\Delta y_{ie})_{j-m_{eq}} \right)_{k=1, \dots, n} =: \gamma_3, \end{aligned} \quad (32)$$

$$\Delta y_{ie} + R^{-1} D_r \Delta c = -y_{ie} + d_r - R^{-1} D_r c + R^{-1}(\mu e - \underline{\Delta r} \underline{\Delta d_r}). \quad (33)$$

We use this equation in order to eliminate  $\Delta c$ :

$$\Delta c = D_r^{-1}R(-y_{ie} + d_r - R^{-1}D_r c + R^{-1}(\mu e - \underline{\Delta r} \underline{\Delta d_r} - \Delta y_{ie})) = D_r^{-1}R(-y_{ie} + d_r + R^{-1}(\mu e - \underline{\Delta r} \underline{\Delta d_r} - \Delta y_{ie})) - c. \quad (34)$$

Inserting in equation 3 of (S1):

$$J_{ie}^T \Delta x - D_r^{-1}R \Delta y_{ie} = -h_{ie}^k(x) - D_r^{-1}R(-y_{ie} + d_r + R^{-1}(\mu e - \underline{\Delta r} \underline{\Delta d_r})) =: \gamma_4.$$

Putting all together we get the following linear system:

$$\begin{pmatrix} \nabla_{xx}L + S^{-1}D_s + T^{-1}D_t & J_{eq} & J_{ie} \\ J_{eq}^T & & \\ J_{ie}^T & & -D_r^{-1}R \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y_{eq} \\ \Delta y_{ie} \end{pmatrix} = \begin{pmatrix} \gamma_3 \\ -h_{eq}^k(x) \\ \gamma_4 \end{pmatrix}. \quad (S5)$$

In comparison with the corresponding predictor system (S2) the matrix remains unchanged. Only the right hand side has changed. This is important justifying an iteration that requires two linear system solutions. A direct linear solver decomposes the matrix in the predictor step and uses this decomposition in the corrector step such that there is only an additional backward substitution necessary. This is also true for the two other approaches, that resulted in the equations (S3) and (S4) for the predictor step. We will not formulate the corresponding corrector equations. They are the same replacing  $\gamma_1$  by  $\gamma_3$  and  $\gamma_2$  by  $\gamma_4$ .

The advantage of using a decomposed matrix twice does not apply for the cg method. Presently, the cg solver is also applied twice but in a future release it is planned to use then the classical primal-dual interior point approach which needs only one linear system solution per iteration but usually a few iterations more than the predictor-corrector method.

## 4 Convergence results

This section provides a short summary of the most important convergence results. Details can be found in [16].

**Theorem 1** *Let the sequence  $(x^k, y^k)_{k=0,1,2,\dots}$  be produced by the SCP-algorithm, all subproblems be solvable and gradients of active constraints at the optimal points of  $(P_{sub}^k)$  be linear independent, as well as those of  $(P_{sub}^*)$  where  $(P_{sub}^*)$  is the subproblem defined in any possible accumulation point  $x^*$  of  $(x^k)_{k=0,1,2,\dots}$ .*

*Then all points  $x^k$  are in  $X$  and all  $y^k$  are in a compact set  $U$ , i.e., there is a  $y_{max}$  with  $|y_j^k|, |v_j^k| \leq y_{max}$  for all  $k \geq 0$  and  $j = 1, \dots, m$ .*

**Theorem 2** *Let the assumptions of Theorem 1 be valid.  $p^k, \eta^k$  and  $\delta^k$  are defined as in Algorithm 3 and let the choice of asymptotes be feasible.*

- a) *Then there are penalty parameters  $\bar{\rho}_j^k > 0$  ( $j = 1, \dots, m$ ) such that  $p^k$  is a direction of descent for all  $\rho \geq \bar{\rho}^k$  for the augmented Lagrangian function  $\Phi_\rho$ , that means*

$$\nabla \Phi_\rho(x^k, y^k)^T p^k \leq -\frac{\eta^k (\delta^k)^2}{2} \quad \text{for all } \rho \geq \bar{\rho}^k.$$

- b) *For each fixed  $\delta > 0$  there is a finite  $\bar{\rho}^\delta$  such that for all  $\begin{pmatrix} x^k \\ y^k \end{pmatrix}$  with  $\delta^k \geq \delta$  we have*

$$\nabla \Phi_\rho(x^k, y^k)^T p^k \leq -\frac{\eta^k (\delta^k)^2}{2} \leq -\frac{\eta \delta^2}{2} \quad \text{for all } \rho \geq \bar{\rho}^\delta.$$

The first part of this theorem shows us that we can always find penalty parameters, such that the computed search direction is a direction of descent for the augmented Lagrangian. The second part guarantees that the penalty parameters are uniformly bounded, provided that we are not too close to a stationary point.

This suffices to prove a weak convergence theorem, i.e. the existence of an accumulation point and at least one accumulation point is stationary. To be able to prove a strong convergence theorem, i.e. additionally to the weak theorem, each accumulation point is stationary, we have to conclude, that the penalty parameters are

uniformly bounded in the neighborhood of a stationary point too. For this purpose we need some additional but reasonable assumptions.

Recall that

$$J := \{j \mid 1 \leq j \leq m_{eq}\} \cup \left\{ j \mid m_{eq} + 1 \leq j \leq m; -\frac{y_j}{\rho_j} \leq h_j(x) \right\},$$

$$K := \{j \mid 1 \leq j \leq m; j \notin J\}.$$

**Theorem 3** *Let the assumptions of Theorem 1 be valid and assume a feasible choice of asymptotes. For  $\delta^k \neq 0$  we define:  $\alpha^k := \frac{\|y^k - v^k\|^2}{(\delta^k)^2}$ . Let  $\begin{pmatrix} x^k \\ y^k \end{pmatrix}$  be determined by Algorithm 3 and fulfill the following two conditions:*

- a)  $j \in J$  if and only if  $h_j^k(z^k) = 0$  ( $j = 1, \dots, m$ ) (that means, subproblems and original problem identify the same set of active constraints),
- b) there is a  $\alpha \in \mathbb{R}$  independent of  $k$ , such that  $\alpha^k \leq \alpha < \infty$ .

Then there is a  $\delta' > 0$ , such that

$$\nabla \Phi_{\rho'}(x^k, y^k)^T p^k \leq -\frac{\eta(\delta^k)^2}{2} \text{ for all } \begin{pmatrix} x^k \\ y^k \end{pmatrix} \text{ with } \delta^k \leq \delta',$$

where  $\rho'$  is a vector of penalty parameters with

$$\min_{j=1, \dots, m} \rho'_j \geq \frac{2\alpha}{\eta}.$$

Theorem 3 would suffice to prove strong convergence but especially assumption a) seems not to be necessarily fulfilled if we are far away from a stationary point. Therefore, we distinguish between Theorem 2 and Theorem 3. Notice, that we do not use any information about  $\delta^k$  in the proof of Theorem 3.

An immediate consequence of the Theorems 2 and 3 is:

**Corollary 1** *Let the sequence  $\begin{pmatrix} x^k \\ y^k \end{pmatrix}_{k=0,1,\dots}$  be produced by Algorithm 3 and let the assumptions of Theorems 2 and 3 be valid. Then the sequence of penalty parameter vectors is bounded, i.e. there is a  $\bar{\rho} < \infty$  and a  $\bar{k} < \infty$  such that  $\rho^k = \bar{\rho}$  for all  $k \geq \bar{k} \geq 0$ .*

With the results of this section it is now possible to formulate the main convergence theorem.

**Theorem 4** *Let the sequence  $\begin{pmatrix} x^k \\ y^k \end{pmatrix}_{k=0,1,\dots}$  be produced by Algorithm 3 and let the assumptions of Theorems 2 and 3 be valid. Then the sequence has at least one accumulation point and each accumulation point is stationary.*

## 5 Technical changes

New users can skip this section. Users of older versions should carefully check whether the calling routine has to be updated.

### 5.1 Technical changes from version 2.3 to version 3.0

- MA47 is no longer supported. More generally, the former option SPSTRAT=3 is no longer supported. Thus, ICNTL(7) in its old sense is not used any longer. Presently, ICNTL(7) is not used at all.
- Sparspak is no longer supported. The routines have been replaced by internal ones. Thus, ICNTL(8) in its old sense is not used any longer. Presently, ICNTL(8) is not used at all.
- A warmstart is now possible, cf. ICNTL(11).
- The combination of SPSTRAT=2 together with LINSYS=2 is not supported in this version (IERR=181).
- In case of large sparse Jacobians of constraints work can be saved if the nonzero structure of the matrix of the linear system in (S3) is fix, cf. ICNTL(12).

- Dynamic allocation of some array is possible now, e.g. if SCIPIP is used in an C/C++-environment or in a Fortran90/95-environment, cf. ICNTL(13).
- Summarizing the above, ICNTL has now to be ICNTL(13).
- In case of an unsuccessful subproblem solution (IERR=31) it is now possible to restart the last iteration.
- Several error messages are new, cf. the header of the routine.

## 5.2 Technical changes from version 2.2 to version 2.3

- The array ACTIVE may now be changed outside SCIPIP, i.e. a user can manipulate the set of active constraints.
- INFO has now to be INFO(23).
- LSUB, ISUBDIM and RINFO are new in the parameter list.
- EMACH has been skipped from the parameter list.
- ICNTL(2)=5 has been skipped.

## 6 Usage of SCIPIP

SCIPIP is a FORTRAN77 subroutine. It contains all necessary subroutines besides some linear algebra which will be mentioned below. To execute SCIPIP, the corresponding file has to be compiled and linked with the object codes for function and gradient evaluations provided by the user. All calculations within the subroutines of this file are performed in double precision arithmetic.

```
CALL SCIPIP30 (N, MIE, MEQ, IEMAX, EQMAX, X0, X_L, X_U, F_ORG, H_ORG, G_ORG, DF, Y_IE,
              Y_EQ, Y_L, Y_U, ICNTL, RCNTL, INFO, RINFO, NOUT, R_SCP, RDIM, R_SUB,
              RSUBDIM, LSCP, IDIM, LSUB, ISUBDIM, ACTIVE, MODE, IERR, IERN, IECN,
              IEDERV, IELPAR, IELENG, EQRN, EQCN, EQCOEF, EQLPAR, EQLENG, MAC-
              TIV, SPIW, SPIWDIM, SPDW, SPDWDIM, SPSTRAT, LINSYS)
```

In the table below, a '\*' means that this value has to be set by the user. A star in brackets '(\*)' means that this value has to be set by the user on request (reverse communication), a star in brackets '{\*}' means that the value may be altered by the user.

*	N	(INTEGER = I) Number of variables, at least $\geq 1$ ; not altered (NA).
*	MIE	(I) Number of inequality constraints (NA).
*	MEQ	(I) Number of equality constraints (NA).
*	IEMAX	(I) Dimension of inequality dependent arrays H_ORG, Y_IE, ACTIVE. Must be at least MIE and $\geq 1$ (NA).
*	EQMAX	(I) Dimension of equality dependent arrays G_ORG, Y_EQ. Must be at least MEQ and $\geq 1$ (NA).
*	X0(N)	(DOUBLE PRECISION = D) X0 has to contain an initial guess of the solution. On return: X0 is replaced by the last computed iterate.
*	X_L(N)	(D) Lower bounds on the variables (NA).
*	X_U(N)	(D) Upper bounds on the variables (NA).
(*)	F_ORG	(D) Contains the objective function value of the last computed iterate.
(*)	H_ORG(IEMAX)	(D) H_ORG contains the values of the inequality constraints at the last computed iterate.
(*)	G_ORG(EQMAX)	(D) G_ORG contains the values of the equality constraints at the last computed iterate.
(*)	DF(N)	(D) DF contains the gradient of the objective function at X0.
(*)	Y_IE(IEMAX)	(D) Y_IE contains the Lagrange multipliers for the inequality constraints at the last computed iterate.

Y_EQ(EQMAX)	(D) Y_EQ contains the Lagrange multipliers for the equality constraints at the last computed iterate.
Y_L(N)	(D) Y_L contains the Lagrange multipliers for the lower bounds on the variables at the last computed iterate.
Y_U(N)	(D) Y_U contains the Lagrange multipliers for the upper bounds on the variables at the last computed iterate.
* ICNTL(13)	(I) Integer array of dimension 13 to be set by the user, but all components have reasonable defaults (NA). A value 0 indicates that the defaults should be chosen. ICNTL(1): desired optimization method: 1: method of moving asymptotes (default) 2: sequential convex programming ICNTL(2): strategy for the choice of asymptotes 1: most common applicable strategy nowadays (default) 2: strategy due to the diploma thesis of Schenk 3: thought to approximate the 'choice' of the CONLIN-method 4: according to the first idea of Svanberg ([9]) ICNTL(3): maximum allowed number of iterations. Has to be greater or equal to 1. Default: 100 ICNTL(4): desired output level 1: no output 2: only final convergence analysis 3: one line of intermediate results (default) 4: more detailed and intermediate results ICNTL(5): maximum number of function calls in the line-search procedure ( $\geq 1$ ). Default: 10 ICNTL(6): if ICNTL(6) = 1, then a relaxed convergence check is applied. The program is terminated if A) the current iterate is feasible, and B) the relative change of the last two succeeding iteration points is less than RCNTL(6), and C) the absolute change of the two last objective function values is less than RCNTL(5), and D) the relative change of the two last objective function values is less than RCNTL(4). Default is the usual convergence check (Kuhn-Tucker conditions) ICNTL(7): presently not used. ICNTL(8): presently not used. ICNTL(11): A warmstart file is generated. If ICNTL(11)=1 then the file is generated only if the maximum number of iterations is reached. if ICNTL(11)=2 then the file is generated after each iteration. (default: 0) ICNTL(12): If ICNTL(12) = 1 then the structure (and size) of the interior point matrix is fix for all iterations. Then a symbolic factorization has to be performed only once. (e.g. linear equality constraints) Nothing is wrong if this parameter is not set, but work can be saved (default: 0) ICNTL(13): If ICNTL(13) = 1 then the user declares that he uses dynamic allocation (cf. ierr=13x,40,...). The difference is, that SCIP uses in this case weaker preliminary estimations for the desired memory.(default: 0)
* RCNTL(6)	(D): double precision array of dimension 6 to be set by the user, but all components have reasonable defaults (NA). A value 0 indicates that the defaults should be chosen. RCNTL(1): desired final accuracy for maximum violation of constraints and the norm of the Kuhn-Tucker residual. Has to be greater or equal than the machine precision EMACH. Default: 1.D-7 RCNTL(2): double precision number that indicates infinity. At least 1.D10. Default: 1.D30

RCNTL(3): all inequality constraints with values  $> -\text{RCNTL}(3)$  are going into the subproblem. If no active-set-strategy is desired, it is recommended to let RCNTL(3) be a large number, e.g. RCNTL(2), which is the default.  
 RCNTL(4,5,6): see ICNTL(6). If ICNTL(6) is not equal to 1, then RCNTL(4,5,6) are not used. Defaults:

RCNTL(4) = 1.D-2

RCNTL(5) = 1.D-2

RCNTL(6) = 1.D-2

If some component of ICNTL/RCNTL is set to 0, i.e. defaults are chosen, then on return the corresponding component of ICNTL/RCNTL contains this value with a minus sign, i.e. if ICNTL(1) = 0, initially, then on return ICNTL(1) = -1

INFO(23)

(I) Integer array of dimension 7 containing some problem information.

INFO(1): number of evaluations of Lagrangian function values

INFO(2): number of evaluations of Lagrangian gradients

INFO(3): necessary value for RDIM

INFO(4): necessary value for RSUBDIM

INFO(5): necessary value for IDIM

INFO(6): necessary value for SPIWDIM

INFO(7): necessary value for SPDWDIM

\*\*\* for INFO(6) and INFO(7) see also SPSTRAT/LINSYS \*\*\*

INFO(8): necessary value for ISUBDIM

INFO(9..19): internal use

INFO(20): current iteration number

INFO(21): number of iterations for the solution of the last subproblem

INFO(22): actually chosen SPSTRAT (in case of SPSTRAT=0, initially)

INFO(23): actually chosen LINSYS (in case of SPSTRAT=0, initially)

RINFO(5)

(D) Double precision array of dimension 5 containing some problem information.

RINFO(1): residual of the subproblem of the last main iteration

RINFO(2): maximum violation of constraints

RINFO(3): stepsize in last main iteration

RINFO(4): norm of the difference of the last two iteration points

RINFO(5): norm of the gradient of the Lagrangian with respect to x at the last iteration point

\* NOUT

(I) Indicates output unit number (NA).

R\_SCP(RDIM)

(D) Double precision working array of dimension at least RDIM.

\* RDIM

(I) Must be at least  $28*N+11*IEMAX+9+9*EQMAX+IELPAR$  (NA).

R\_SUB(RSUBDIM)

(D) Double precision working array of dimension at least RSUBDIM.

\* RSUBDIM

(I) Must be at least  $22*N+41*IEMAX+26*EQMAX+IELPAR+EQLPAR$  (NA).

L\_SCP(IDIM)

(I) Integer working array of dimension at least IDIM.

\* IDIM

(I) Must be at least  $7*N+8*IEMAX+4*EQMAX+IELPAR+6$  (NA).

L\_SUB(ISUBDIM)

(I) Integer working array of dimension at least ISUBDIM.

\* ISUBDIM

(I) Must be at least  $2*N+3*IEMAX+2*EQMAX+IELPAR$  (NA).

{\*} ACTIVE(IEMAX)

(I) Integer array indicating the inequality constraints which are considered as active (see RCNTL(3)). Constraint active  $\leftrightarrow$  ACTIVE() = 1: only these gradients have to be updated (active set strategy). A user is allowed to change ACTIVE before computing gradients, i.e. if IERR = -2.

\* MODE

(I) (NA).

1: Usual subroutine call for function values and gradients

2: Reverse communication

\* IERR

(I) Has initially to be 0. Do not alter IERR outside the routine, e.g. in case of reverse communication!

On return:

<0 : Reverse communication:

-1: function values are requested

-2: gradients are requested

-3: INFO(6) and INFO(7) have been computed. The user can now dimension SPIW and SPDW dynamically. If this is not desired, just rejump into SCPIP after IERR = -3.

0: successful computation

1: maximum number of iterations performed

2:  $N \leq 0$

3:  $MIE < 0$  or  $MEQ < 0$  or  $IEMAX < 1$  or  $EQMAX < 1$

4:  $MIE > IEMAX$  or  $MEQ > EQMAX$

5: IELPAR or EQLPAR  $< 1$

6: for at least one component, the lower bound is greater or equal to the upper bound. Greater makes no sense and equal disables the interior point subproblem solver!

7: no suitable choice of MODE!

8: RDIM too small, cf. INFO(3)

9: RSUBDIM too small, cf. INFO(4)

901: ISUBDIM too small, cf. INFO(8)

10: IDIM too small, cf. INFO(5)

11: the chosen strategy for the choice of the asymptotes is only suitable for positive variables and lower bounds!

120, 121 (12/0,12/1): SPDWDIM too small! In this phase, INFO(7) is an estimation of the space needed; but still an estimation!

130, 131, 132, 133, 134, 135 (13/0..5): SPIWDIM too small! In this phase, info(6) is an estimation of the space needed; but still an estimation! Error numbers are distinguished to return to the right place in case of dynamic allocation.

14: the user provided for at least one constraint an empty column in the Jacobian. Check the gradients! (There is now a workaround and thus no error is reported.)

15: the Jacobian matrices are not stored columnwise

16: the Jacobian matrices are not stored correctly. For at least one column the components are out of order!

161: (16/1) the Jacobian matrices are not stored correctly. More columns than constraints are provided.

18: SPSTRAT=2 works only without equality constraints.

181: SPSTRAT=2 together with LINSYS=2 presently not allowed.

191: (19/1) no suitable choice for SPSTRAT

192: (19/2) no suitable choice for LINSYS

20: the feasible region of the current subproblem is a singleton. One possible reason is, that the feasible region of the original problem is empty!

21: for at least one component, lower and upper bound of a subproblem are almost equal. The interior-point subproblem solver is not applicable!

22: line-search needs more than ICNTL(5) function evaluations, possibly the gradients are wrong

23: the norm of the gradient of the Lagrangian is close to 0 and the maximum of the artificial variables is  $\geq 1$ . Together, it is very likely, that the feasible region is empty!

24: the current value for IELENG/EQLENG is larger than IELPAR/EQLPAR

30: error in solution of subproblem. Unclear reason for the error!

31: the subproblem could not be solved within maximum allowed iterations! Restart is possible, cf. manual.

32: the subproblem could not be solved due to arithmetic overflow!

40: error in the construction of the predictor matrix in the subproblem. SPIWDIM is too small. INFO(6) is now a quite accurate estimation of the space needed.

41: error in the construction of the predictor matrix in the subproblem. SPDWDIM is too small. INFO(7) is now a quite accurate estimation of the space needed.

42: error in the ordering of the predictor matrix in the subproblem. The reason is unclear.

43: error in the solution of the predictor system in the subproblem. Error in memory handling. Reason unclear.

45: error in the construction of the predictor matrix in the subproblem. The matrix is not positive definite: either there is something wrong with the data or there are numerical reasons (bad scaling, e.g.).

In case of IERR = 22, H\_ORG, G\_ORG and F\_ORG do not contain the values at the last iterate but values of intermediate results. If the values at the last iterate are needed, the user has to recompute in the main program. X0, DF, IEDERV and EQCOEF contain the values at the last iterate.

- (\*) IERN(IELPAR) (I) IERN contains the row numbers, i.e. the index of the component of the entries of the Jacobian of the inequality constraints (NA).
- (\*) IECN(IELPAR) (I) IECN contains the column numbers, i.e. the number of the inequality of the entries of the Jacobian of the inequality constraints (NA).
- (\*) IEDERV(IELPAR) (D) IEDERV contains the values of the entries of the Jacobian of the inequality constraints. Only nonzero elements have to be stored, but zero elements are allowed. These three arrays IERN, IECN and IEDERV are expected to be sorted by function numbers and inside one function by component numbers, i.e. it is expected that the Jacobian is stored columnwise (NA).
- \* IELPAR (I) Dimension of arrays IERN, IECN and IEDERV. Must be at least IELENG and  $\geq 1$  (NA).
- (\*) IELENG (I) Actual number of entries in IEDERV (NA).
- (\*) EQRN(EQLPAR) (I) EQRN contains the row numbers, i.e. the index of the component of the entries of the Jacobian of the equality constraints (NA).
- (\*) EQCN(EQLPAR) (I) EQCN contains the column numbers, i.e. the number of the equality of the entries of the Jacobian of the equality constraints (NA).
- (\*) EQCOEF(EQLPAR) (D) EQCOEF contains the values of the entries of the Jacobian of the equality constraints. Only nonzero elements have to be stored, but zero elements are allowed. These three arrays EQRN, EQCN and EQCOEF are expected to be sorted by function numbers and inside one function by component numbers, i.e. it is expected that the Jacobian is stored columnwise (NA).
- \* EQLPAR (I) Dimension of arrays EQRN, EQCN and EQCOEF. Must be at least EQLENG and  $\geq 1$  (NA).
- (\*) EQLENG (I) Actual number of entries in EQCOEF (NA).
- (\*) MACTIV (I) Number of constraints considered as active in the subproblem model (depends on RCNTL(3)). Only this number of constraints is used in the subproblem. This affects the required working space, see SPDWDIM below, and thus the subproblem solution method.
- SPIW(SPIWDIM) (I) Integer working array of dimension at least SPIWDIM. This array is used for the subproblem solution. Its necessary dimension is dependent on the subproblem solution strategy. It can be varied after each subproblem solution (in case of reverse communication), i.e. the subproblem solution method can be changed after each main iteration. After IERR = -3 one can dimension SPIW and SPDW dynamically with the values of INFO(6) and INFO(7).
- \* SPIWDIM (I) Has to be at least (NA):  
 SPSTRAT=1:  
   LINSYS=1,3: 1  
   LINSYS=2: (IELENG+EQLENG)\*5+2\*N+1 (estimation)  
 SPSTRAT=2:  
   LINSYS=1,3: 1  
   LINSYS=2 : 5\*IELENG+2\*MACTIV+1 (estimation)
- SPDW(SPDWDIM) (D) Double precision working array of dimension at least SPDWDIM. For the handling see SPIW.

the reason to distinguish between SPDW and R.SUB is, that the storage for SPDW and SPIW is variable dependent on the subproblem solver, whereas R.SUB needs a fixed amount.

- \* SPDWDIM (I) Has to be at least (NA):  
 SPSTRAT=1, LINSYS=1: (MACTIV+MEQ)\*\*2  
 SPSTRAT=1, LINSYS=2: 10\*(IELENG+EQLENG) (estimation)  
 SPSTRAT=1, LINSYS=3: 1  
 SPSTRAT=2: LINSYS=1: N\*N  
 SPSTRAT=2: LINSYS=3: 1  
 the estimated amounts should be enlarged to a maximum possible value, because it has to contain the decomposition of a sparse matrix. The value above is the formally lowest possible one and will usually not suffice in practice.
- \* SPSTRAT (I) Subproblem solution strategy (NA):  
 0: Automatic choice by the program  
 1: Approach (S3)  
 2: Approach (S4)
- \* LINSYS (I) Linear systems solver that should be applied. Only valid for SPSTRAT = 1,2 (NA):  
 1: Dense Cholesky solver  
 2: Sparse Cholesky solver.  
 3: CG solver  
 Notice, that for SPSTRAT=1 with LINSYS=2, the necessary storage for the subproblem solver can only be estimated in advance, such that an error can occur when a linear system will be decomposed.

Notice, that whenever gradients are required, a previous call of function values was made at the same iterate X0 and F\_ORG, H\_ORG and G\_ORG contain function values of objective and constraints at X0. X0, F\_ORG, H\_ORG and G\_ORG are not allowed to be altered during the computation of gradients. Let us now explain some input parameters in detail.

## X0

X0 has to fulfill the box-constraints, i.e.  $X.L \leq X0 \leq X.U$ . If this condition is not fulfilled, then the variable will be set on its violated bounds.

## X.L and X.U

X.L and X.U have to contain some values. If no bounds exist, we recommend to initialize X.L with -RCNTL(2) and X.U with RCNTL(2).

## ICNTL(1)

ICNTL(1) = 1 denotes the original MMA method (Algorithm 1), ICNTL(1) = 2 denotes SCP, MMA with line-search (Algorithm 3).

## ICNTL(2)

There are four different strategies for the choice of the asymptotes implemented (cf. step 1 of Algorithms 1 and 3). This is only thought for very experienced users. Otherwise we strongly recommend the choice of ICNTL(2) = 1.

In the following, let  $k$  be the iteration index throughout.

**ICNTL(2) = 1:** This scheme is used nowadays in most cases:

$k = 0, 1$  : If  $\underline{x}_i = -\text{RCNTL}(2)$  or  $\bar{x}_i = \text{RCNTL}(2)$ :

$$L_i^k = x_i^k - \max(1, |x_i^k|),$$

$$U_i^k = x_i^k + \max(1, |x_i^k|),$$

else:

$$L_i^k = x_i^k - 0.5(\bar{x}_i - \underline{x}_i),$$

$$U_i^k = x_i^k + 0.5(\bar{x}_i - \underline{x}_i).$$

$k = 2, 3, \dots$  : If  $\text{sign}(x_i^k - x_i^{k-1}) = \text{sign}(x_i^{k-1} - x_i^{k-2})$  :

$$L_i^k = x_i^k - 1.15 \cdot (x_i^{k-1} - L_i^{k-1}),$$

$$U_i^k = x_i^k + 1.15 \cdot (U_i^{k-1} - x_i^{k-1}).$$

If  $\text{sign}(x_i^k - x_i^{k-1}) \neq \text{sign}(x_i^{k-1} - x_i^{k-2})$  :

$$L_i^k = x_i^k - 0.7 \cdot (x_i^{k-1} - L_i^{k-1}),$$

$$U_i^k = x_i^k + 0.7 \cdot (U_i^{k-1} - x_i^{k-1}).$$

**ICNTL(2) = 2:** This is a scheme based on the traditional ICNTL(2) = 4, modified for general variables. It is only recommended if the lower and upper bounds on the variables are reasonable, i.e. if they are by the same magnitude and not just 'very small' or 'very large' numbers.

$k = 0, 1$  :  $L_i^k = \underline{x}_i - 0.1(\bar{x}_i - \underline{x}_i)$ ,

$$U_i^k = \bar{x}_i + 0.1(\bar{x}_i - \underline{x}_i).$$

$k = 2, 3, \dots$  : If  $\text{sign}(x_i^k - x_i^{k-1}) = \text{sign}(x_i^{k-1} - x_i^{k-2})$  :

$$L_i^k = x_i^k - \frac{x_i^{k-1} - L_i^{k-1}}{0.7},$$

$$U_i^k = x_i^k + \frac{U_i^{k-1} - x_i^{k-1}}{0.7}.$$

If  $\text{sign}(x_i^k - x_i^{k-1}) \neq \text{sign}(x_i^{k-1} - x_i^{k-2})$  :

$$L_i^k = x_i^k - 0.7(x_i^{k-1} - L_i^{k-1}),$$

$$U_i^k = x_i^k + 0.7(U_i^{k-1} - x_i^{k-1}).$$

**ICNTL(2) = 3:** This strategy is thought to simulate the CONLIN approximation [1]. This strategy makes only sense if there are lower bounds and an initial guess greater than 0.

$k = 0, 1, 2, \dots$  :  $L_i^k = 0$ ,

$$U_i^k = \text{RCNTL}(2).$$

**ICNTL(2) = 4:** This was one of the first proposals by Svanberg (1987). This strategy makes only sense if there are lower bounds and an initial guess greater than 0.

$k = 0, 1, 2, \dots$  :  $L_i^k = 0.7 \cdot x_i^k$ ,

$$U_i^k = \frac{x_i^k}{0.7}.$$

## ICNTL(5)

ICNTL(5) is the maximum number of stepsize updates in step 8 of Algorithm 3. Each such a step needs an additional evaluation of function values, whereas gradient values are only needed for the point belonging to the accepted stepsize.

## ICNTL(6), RCNTL(4), RCNTL(5) and RCNTL(6)

If ICNTL(6)=0, then the usual check for convergence is done, i.e. the Kuhn-Tucker conditions are checked as shown above.

If ICNTL(6)=1, a relaxed convergence check is done. The program is stopped if the following four conditions are fulfilled simultaneously:

1. The current iterate has to be feasible, i.e.  $h_j(x) \leq \text{RCNTL}(1)$ ,  $j = m_{eq} + 1, \dots, M$  and  $|h_j(x)| \leq \text{RCNTL}(1)$ ,  $j = 1, \dots, m_{eq}$ .
2.  $\frac{\|x^k - x^{k-1}\|_\infty}{\|x^{k-1}\|_\infty} \leq \text{RCNTL}(6)$ , where  $k$  is the current iteration index.
3.  $|f(x^k) - f(x^{k-1})| \leq \text{RCNTL}(5)$ .
4.  $\frac{|f(x^k) - f(x^{k-1})|}{|f(x^{k-1})|} \leq \text{RCNTL}(4)$ .

## ICNTL(11)

By appropriate setting of ICNTL(11) the user can decide to produce a warmstart file. The advantage is that the optimization process can be restarted with the same data which he had at the end of the first attempt. If no warmstart file has been generated the last computed point can be used but many other (internal) data are not available.

If ICNTL(11)=1 a warmstart file is generated if the maximum number of iterations has been reached, cf. ICNTL(3). If ICNTL(11)=2 a warmstart file is generated after each successful iteration.

**Notice:** the writing to the warmstart file can be time and storage consuming. It should be clearly justified whether it makes sense to do it.

To the experience of the author warmstart files can be helpful, i.e. time saving, in the period of code development if errors have to be expected. Later on it should be avoided, at least the writing of files after each iteration.

**How to restart:** In case of a warmstart the file *warmstart.txt* has been defined. It may not be changed! If the user wants to restart he has to set IERR=-4 and enter SCPIP. SCPIP returns with IERR=-4. Now (not before!) the user may change some data (ICNTL, RCNTL). Then he has to set IERR=-5 and to return to SCPIP again. Notice, that if the reason for the termination was the maximum number of iterations the user has to change ICNTL(3) appropriately. Also notice that the output file (cf. ICNTL(4)) will be overwritten. Thus, it has to be saved by the user if necessary.

Examples: for a problem with 150 variables, one constraint and IELENG=150 the size of the warmstart file is about 150 KB. For 48600 variables, 48601 constraints and IELENG=490000 the size is about 90 MB.

Code example (warmstart.txt exists):

```
...
      IERR = -4
...
100  CALL SCPIP30(...)
...
      IF (IERR .EQ. -4) THEN
          ICNTL(3) = 100
          RCNTL(3) = 1.D-5
          IERR = -5
          GOTO 100
      END IF
```

## ICNTL(12)

This is presently only valid for SPSTRAT=1 and LINSYS=2 or perhaps if SPSTRAT=0, i.e. if the linear system (S3) is used. If ICNTL(12)=1 the user states that the nonzero structure of the matrix of the linear system is fixed throughout. Clearly, this makes only sense if the system is large and sparse. If the structure is not fix then some data are computed and the matrix of (S3) is symbolically factorized in each subproblem once. This is not necessary if the structure is fix, then this work has to be done only once in the first subproblem. Thus, much work can be saved.

This option should be used only if the user is sure about his situation. Otherwise errors have to be expected. For example, the structure is fixed throughout, if there are only linear equality constraints or if there are linear inequality constraints **and** the active set strategy does not change the active set throughout (e.g. if the active set strategy is not used).

## ICNTL(13)

Dynamic allocation of the arrays SPIW, SPDW, R\_SCP, R\_SUB, LSUB and LSCP is possible. If ICNTL(13)=1 the user states that he uses dynamic allocation. In this case weaker preliminary estimations of some arrays are used and the user can provide first of all less storage than stated above for these arrays.

If there is too few storage SCPIP returns with the following errors:

IERR=40, 130, 131, 132, 133, 134, 135: SPIWDIM is too small. Allocate INFO(6) for SPIW/SPIWDIM.

The various error numbers are necessary such that SCPIP knows where to go back after allocation.

IERR=41, 120, 121: SPDWDIM is too small. Allocate INFO(7) for SPDW/SPDWDIM.

IERR=8: RDIM is too small. Allocate INFO(3) for R\_SCP/RDIM.

IERR=9: RSUBDIM is too small. Allocate INFO(4) for R\_SUB/RSUBDIM.

IERR=901: ISUBDIM is too small. Allocate INFO(8) for LSUB/ISUBDIM.

IERR=10: IDIM is too small. Allocate INFO(5) for LSCP/IDIM.

If no dynamic allocation is used these error messages have to lead to an abort.

After the allocation is done jump back into SCPIP. Don't change IERR.

Example (Fortran95):

```
...
INTEGER SPIWDIM
INTEGER, ALLOCATABLE:: SPIW(:)
...
SPIWDIM = 1
ALLOCATE (SPIW(SPIWDIM))
...
100 CALL SCPIP30 (...)

IF ((IERR .EQ. 40) .OR. (IERR .EQ. 130) .OR. (IERR .EQ. 131) .OR. (IERR .EQ. 132)&
.OR. (IERR .EQ. 133) .OR. (IERR .EQ. 134)) THEN
  DEALLOCATE (SPIW)
  SPIWDIM=INFO(6)
  ALLOCATE (SPIW(SPIWDIM))
  GOTO 100
ENDIF
...
```

## RCNTL(1)

RCNTL(1) denotes the desired accuracy to accept an iteration point as a solution. RCNTL(1) is important for two conditions: feasibility and stationarity. I.e., the following has to be fulfilled:

$$|h_j(x)| \leq \text{RCNTL}(1), \quad j = 1, \dots, m_{eq},$$

$$h_j(x) \leq \text{RCNTL}(1), \quad j = m_{eq} + 1, \dots, M,$$

$$\|\nabla f(x) + \sum_{j=1}^{m_{eq}} (y_{eq})_j \nabla h_j(x) + \sum_{j=m_{eq}+1}^M (y_{ie})_{j-m_{eq}} \nabla h_j(x) - \sum_{i=1}^n (y_l)_i + \sum_{i=1}^n (y_u)_i\| \leq \text{RCNTL}(1).$$

$y_{eq}$  corresponds to Y\_EQ,  $y_{ie}$  corresponds to Y\_IE,  $y_l$  corresponds to Y\_L and  $y_u$  corresponds to Y\_U. Notice, that the box-constraints are always fulfilled due to the construction of the algorithm.

### RCNTL(3)

For the inequality constraints  $h_j(x) \leq 0, j = m_{eq} + 1, \dots, M$ , an active-set strategy can be performed. In the definition of a subproblem ( $P_{sub}^k$ ), only those constraints are considered, which fulfill the following conditions:

**If RCNTL(3)  $\geq 0$ :**

All constraints with

$$h_j(x) > -\text{RCNTL}(3),$$

because for the other constraints it is assumed that they are not active and thus are not necessary to formulate the subproblem.

**If RCNTL(3)  $< 0$ :**

All constraints with

$$h_j(x) > \text{RCNTL}(3),$$

and, additionally, all those constraints, that had a nonzero Lagrangian multiplier Y\_IE(j) in the preceding iteration.

In both cases, the usage of RCNTL(3) needs good knowledge of the problem to be able to justify which (negative) function value indicates that the corresponding constraint will most probably not be active in the solution. If no active-set strategy is desired, let  $\text{RCNTL}(3) = \text{RCNTL}(2)$ .

### ACTIVE

The array ACTIVE represents the result of an active set strategy. ACTIVE is closely related to RCNTL(3). It is an integer array which indicates for each inequality constraint whether it is considered as (potentially) active (ACTIVE(j) = 1) or not. It is not intended to be sharp in a strongly mathematical sense of an active constraint. It is thought to contain all violated and active constraints as well as all those constraints that become likely active or violated in subsequent iterations.

Only for inequality constraints with ACTIVE(j) = 1, it is necessary to compute gradient values. If only function values are requested by the program (IERR=-1), then it is expected that **all** function values are updated. If ACTIVE is also used for the function values it is the responsibility of the user to control whether a constraint that is inactive becomes active in later iterations.

A user is allowed to change ACTIVE in his environment. Clearly, this is only possible for MODE=2. The user may change ACTIVE before computing the gradients, i.e. if IERR=-2, not before! Otherwise, ACTIVE is overwritten by the active set strategy of SCIP.

This can make sense if, e.g., a constraint is not detected as potentially active by SCIP but the user knows or expects that this constraint will be active at the optimum. Then he can change ACTIVE(i), which has been returned by SCIP with value 0 to ACTIVE(i)=1. In this case the user has also to update MACTIV and in consequence also SPIWDIM and SPDWDIM if these dimensions have been calculated accurately before.

### MODE

With this variable it is distinguished whether a normal subroutine call performs function and gradient evaluations (MODE=1) or if reverse communication is performed (MODE=2).

In case of MODE = 1, the user has to provide subroutines SCPFCT and SCPGRD to be called by SCIP: SCPFCT (N,MIE,MEQ,X0,F\_ORG,G\_ORG,H\_ORG),

Input: N,MIE,MEQ,X0,

Output:F\_ORG,G\_ORG,H\_ORG.

SCPGRD (N,MIE,MEQ,X0,F\_ORG,G\_ORG,H\_ORG,ACTIVE,DF,IERN,IECN, IEDERV,IELENG,EQRN,  
EQCN,EQCOEF,EQLENG),

Input: N,MIE,MEQ,X0,F\_ORG,G\_ORG,H\_ORG,ACTIVE,

Output: DF,IERN,IECN,IEDERV,IELENG,EQRN,EQCN,EQCOEF,EQLENG.

All variables have the same meaning as for SCPIP.

In case of  $MODE = 2$ , SCPIP returns whenever function or gradient evaluations are required. In case of function evaluations, variable  $IERR$  is set to  $-1$ . Then the user has to provide function values ( $F\_ORG$ ,  $G\_ORG$ ,  $H\_ORG$ ) at the current point  $X0$ . The same has to be done in case of  $IERR = -2$ , then gradients are required ( $IERN$ ,  $IECN$ ,  $IEDERV$ ,  $IELENG$ ,  $EQRN$ ,  $EQCN$ ,  $EQCOEF$ ,  $EQLENG$ ). In this case, variable  $ACTIVE$  is important. In case of  $IERR = -3$  the values of  $INFO(6)$  and  $INFO(7)$  have been computed and can now be used to set  $SPIWDIM$  and  $SPDWDIM$  dynamically after each iteration. If this is not desired, maybe because the user trusts the values set at the beginning, then it can just be jumped back into SCPIP. **Notice**, that whenever gradients are required, a previous call of function values was made at the same iterate  $X0$  and  $F\_ORG$ ,  $H\_ORG$  and  $G\_ORG$  contain function values of objective and constraints at  $X0$ .  $X0$ ,  $F\_ORG$ ,  $H\_ORG$  and  $G\_ORG$  are not allowed to be altered during the computation of gradients.

### **IERN, IECN, IEDERV, IELENG**

These arrays define the Jacobian matrix of the inequality constraints considered as active, i.e. for all inequalities  $J$  with  $ACTIVE(J) = 1$ . Sparse storage format as follows is used, i.e. only nonzero values have to be stored. For problems with full Jacobians this format has to be used also.

$IERN$  has to contain the row numbers of the Jacobian, i.e. the index of the corresponding component.  $IECN(I)$  contains the number of the function of entry  $I$ . Its value is stored in array  $IEDERV(I)$ .  $IELENG$ , finally, denotes the total number of entries in the three arrays  $IERN$ ,  $IECN$  and  $IEDERV$ , which are of equal length  $IELENG$ . It is no problem if zero values are stored.

The Jacobian has to be stored columnwise, i.e. firstly all entries of constraint 1, then all entries of constraint 2 and so on. Within one function the entries have to be sorted by component numbers.

### **EQRN, EQCN, EQDERV, EQLENG**

For these arrays the same is valid as for the arrays  $IERN$ ,  $IECN$ ,  $IEDERV$  and  $IELENG$ , replacing inequalities by equalities. Since an equality is always to be considered as active, no use of the array  $ACTIVE$  is made here.

### **R\_SCP, R\_SUB, SPDW**

The reason to distinguish between these three working arrays is that  $R\_SCP$  has to be kept during one optimization run of SCPIP, whereas the storage for  $R\_SUB$  and  $SPDW$  can be freed after each iteration. I.e. if  $IERR = -2$ , or in case of  $ICNTL(1) = 1$ , if  $IERR = -1$ , then a subproblem solution has been finished and the storage of  $R\_SUB$  and  $SPDW$  can be freed. This can be important if the storage is needed for function or gradient evaluations. The storage has to be reallocated before SCPIP is reentered again.  $R\_SUB$  and  $SPDW$  are separated because the length of  $RSUBDIM$  is fixed, whereas the necessary length of  $SPDW$  can vary with the number of active constraints and so on, cf. definition of  $SPDWDIM$ .

### **LSCP, LSUB, SPIW**

For these arrays the same is valid as for  $R\_SCP$ ,  $R\_SUB$  and  $SPDW$ .

### **SPSTRAT**

$SPSTRAT$  denotes the subproblem solution strategy.  $SPSTRAT = 1$  is approach (S3),  $SPSTRAT = 2$  is approach (S4).

The user has the possibility to set  $SPSTRAT = 0$  which means that the program decides itself for the best combination of  $SPSTRAT = 1,2$  and  $LINSYS$ .

## LINSYS

For the approaches (S3) and (S4), i.e. for  $SPSTRAT = 1,2$ , there are three possibilities to solve the linear system.  $LINSYS = 1$  denotes the usage of a dense Cholesky solver.  $LINSYS = 2$  means the usage of a sparse Cholesky solver.  $LINSYS = 3$  denotes the usage of an iterative conjugate gradient solver.

## Restart in case of IERR=31

In case of an unsuccessful subproblem solution ( $IERR=31$ ) it is possible to restart the last iteration. This can be useful if possibly the estimation of active constraints was too optimistic, i.e. too few constraints entered the subproblem ( $ACTIVE(I) = 1$ ). After SCPIP finished with  $IERR=31$  the user may change  $ACTIVE$  (it seems only to be useful to add constraints, not to delete). In this case, also the gradient information has to be updated ( $IERN$ ,  $IECN$ ,  $IEDERV$ ,  $IELENG$  and  $MACTIV$ ). After changing this information rejump into SCPIP without changing  $IERR$ .

## 7 Example

In this section we figure out how a calling program has to look like. Which parameters have to be set and so on. We chose an easy example, # 34 of the collection of Hock and Schittkowski [3]. It has  $N = 3$  variables and  $MIE = 2$  constraints. It is implemented using reverse communication. Direct calling of the routines to compute function and gradient values would be possible in this case by setting  $MODE = 1$ .

Objective function:  $f(x) = -x_1$ .

Constraints:  $h_1(x) = e^{x_1} - x_2$ ,

$h_2(x) = e^{x_2} - x_3$ ,

$0 \leq x_1 \leq 100$ ,

$0 \leq x_2 \leq 100$ ,

$0 \leq x_3 \leq 10$ .

Starting point:  $x^0 = (0, 1.05, 2.9)$ .

```

program main
c   Sample driving program for problem #34 in Hock/Schittkowski

implicit none

integer nn,mi,me,ielpar,eqlpar,rd,rd2,rd3,id,id2,id3

parameter(nn=100,mi=10,me=10)           !nn maximum number of variables,
                                         !mi inequality constraints
                                         !me equality constraints

parameter(ielpar=mi*10,eqlpar=me*10)    !ielpar maximum number of entries in
                                         !Jacobian of inequalities. eqlpar same
                                         !for equalities

parameter(rd=1000,rd2=1000,rd3=1000,id=1000,id2=1000,id3=1000)
                                         !maximum for working arrays

integer          n,mie,meq,iemax,eqmax,icntl(13),info(23),nout,
+               rdim,rsubdim,i_scp(id),idim,i_sub(id2),isubdim,
```

```

+         active(mi),mode,ierr,iern(ielpar),iecn(ielpar),
+         ieleng,eqrn(eqlpar),eqcn(eqlpar),eqleng,mactiv,
+         spiw(id3),spiwdim,spdwdim,spstrat,linsys,i
double precision x0(nn),x_l(nn),x_u(nn),f_org,h_org(mi),g_org(me),
+         df(nn),y_ie(mi),y_eq(me),y_l(nn),y_u(nn),
+         rcntl(6),rinfo(5),r_scp(rd),r_sub(rd2),
+         iederv(ielpar),eqcoef(eqlpar),spdw(rd3)

n = 3
mie = 2
meq = 0

iemax = max(mie,1)
eqmax = max(meq,1)

x0(1) = 0.d0
x0(2) = 1.05d0
x0(3) = 2.9d0

x_l(1) = 0.d0
x_l(2) = 0.d0
x_l(3) = 0.d0

x_u(1) = 1.d2
x_u(2) = 1.d2
x_u(3) = 1.d1

icntl(1) = 1
icntl(2) = 1
icntl(3) = 100
icntl(4) = 1
icntl(5) = 10
icntl(6) = 0
icntl(11) = 0
icntl(12) = 0
icntl(13) = 0

rcntl(1) = 1.d-7
rcntl(2) = 1.d30
rcntl(3) = 1.d30
rcntl(4) = 0.d0
rcntl(5) = 0.d0
rcntl(6) = 0.d0

ierr = 0
nout = 7
rdim = rd
rsubdim = rd2
idim = id
isubdim = id2
spiwdim = id3
spdwdim = rd3

mode = 2

spstrat = 1
linsys = 1

```

```

100  call scpip30 (n,mie,meq,iemax,eqmax,x0,x_l,x_u,f_org,h_org,
+           g_org,df,y_ie,y_eq,y_l,y_u,icntl,rcntl,info,
+           rinfo,nout,r_scp,rdim,r_sub,rsubdim,i_scp,idim,
+           i_sub,subdim,active,mode,ierr,iern,iecn,iederv,
+           ielpar,ieleng,eqrn,eqcn,eqcoef,eqlpar,eqleng,mactiv,
+           spiw,spiwdim,spdw,spdwdim,spstrat,linsys)

      if (ierr .eq. 0) then
        print*, 'solution obtained'
      else if (ierr .eq. -1) then
        call scpfct (n,mie,meq,x0,f_org,g_org,h_org)
        goto 100
      else if (ierr .eq. -2) then
        call scpgrd (n,mie,meq,x0,f_org,g_org,h_org,active,df,iern,
+           iecn,iederv,ieleng,eqrn,eqcn,eqcoef,eqleng)
        goto 100
      else if (ierr .eq. -3) then
c now you have the chance to adopt spiwdim and spdwdim with the information
c of info(6) and info(7). we just jump back here.
        goto 100
      else
        print*, 'scpip finished with an error! ierr = ',ierr
      end if

      stop
      end

```

\*\*\*\*\*

```

      subroutine scpfct (n,mie,meq,x,f_org,g_org,h_org)

```

```

c   input:  n,mie,meq,x
c   output: f_org,g_org,h_org

```

```

      implicit none

```

```

      integer          n,mie,meq
      double precision f_org,g_org(*),h_org(*),x(*)

```

```

      f_org = -x(1)

```

```

      h_org(1) = dexp(x(1)) - x(2)
      h_org(2) = dexp(x(2)) - x(3)

```

```

      return
      end

```

\*\*\*\*\*

```

      subroutine scpgrd (n,mie,meq,x,f_org,g_org,h_org,active,df,iern,
+           iecn,iederv,ieleng,eqrn,eqcn,eqcoef,eqleng)

```

```

c   input:  n,mie,meq,x,f_org,g_org,h_org,active
c   output: df,iern,iecn,iederv,ieleng,eqrn,eqcn,eqcoef,eqleng

```

```

      implicit none

```

```

      integer          n,mie,meq,active(*),iern(*),iecn(*),ieleng,

```

```

+          eqrn(*),eqcn(*),eqleng,numbercon
double precision f_org,g_org(*),h_org(*),x(*),df(*),iederv(*),
+          eqcoef(*)

df(1) = -1.d0
df(2) = 0.d0          !gradient of objective fully stored!
df(3) = 0.d0

ieleng = 0
numbercon = 0

if (active(1) .eq. 1) then
  numbercon=numbercon+1          !one more active constraint
c constraint 1, component 1:
  ieleng = ieleng + 1          !derivative always > 0
  iecn(ieleng) = numbercon      !function number
  iern(ieleng) = 1             !component number
  iederv(ieleng) = dexp(x(1))   !value

c constraint 1, component 2:
  ieleng = ieleng + 1          !derivative always > 0
  iecn(ieleng) = numbercon      !function number
  iern(ieleng) = 2             !component number
  iederv(ieleng) = -1.d0       !value
endif

if (active(2) .eq. 1) then
  numbercon=numbercon+1          !one more active constraint
c constraint 2, component 2:
  ieleng = ieleng + 1          !derivative always > 0
  iecn(ieleng) = numbercon      !function number
  iern(ieleng) = 2             !component number
  iederv(ieleng) = dexp(x(2))   !value

c constraint 2, component 3:
  ieleng = ieleng + 1          !derivative always > 0
  iecn(ieleng) = numbercon      !function number
  iern(ieleng) = 3             !component number
  iederv(ieleng) = -1.d0       !value
endif

return
end

```

## 8 Scaling

It is important to say a word about scaling. Badly scaled problems are hard to solve for any nonlinear programming algorithm. But how to scale is problem dependent. A good knowledge of the behavior of objective function and constraints is crucial to apply a useful scaling procedure. Thus, it is in the responsibility of the user to think on this question. It is beyond the scope of this paper to go into details here. We refer to the nonlinear programming literature.

We give here just one example of a well tried scaling procedure, applied in the context of structural optimization. An objective function weight can be scaled by its initial value if the initial design is a reasonable one. This is usually the case:

$$f_{\text{scaled}}(x) := \frac{f(x)}{f(x^0)}.$$

I.e., the scaling factor  $\frac{1}{f(x^0)}$  is applied to all subsequent iterations.

The most important constraints in structural optimization are stress constraints ( $\sigma(x)$ ). Usually, there are realistic lower and upper bounds on stresses in elements, that means the bounds are not just any large or small numbers. In this case, the following scaling procedure can be applied:

Original constraints:

$$\underline{\sigma}(x) \leq \sigma(x) \leq \bar{\sigma}(x).$$

Scaled constraints:

$$1 - \frac{\sigma(x)}{\underline{\sigma}(x)} \leq 0,$$
$$\frac{\sigma(x)}{\bar{\sigma}(x)} - 1 \leq 0.$$

## Acknowledgement

I am grateful to Wolfgang Kraus and Markus Inkeroinen for pointing me to several errors and inconsistencies in the code and the manual.

## References

- [1] C. Fleury. CONLIN: an efficient dual optimizer based on convex approximation concepts. *Structural Optimization*, 1:81–89, 1989.
- [2] C. Fleury. First and second order convex approximation strategies in structural optimization. *Structural Optimization*, 1:3–10, 1989.
- [3] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes*. Lecture Notes in Economics and Mathematical Systems, No. 187. Springer-Verlag, Berlin, Heidelberg, New York, 1981.
- [4] F. Jarre and M. Saunders. A practical interior-point method for convex programming. *SIAM J. on Optimization*, 5:149–171, 1995.
- [5] I.J. Lustig, R.E. Marsten, and D.F. Shanno. On implementing Mehrotra’s predictor-corrector interior point method for linear programming. *SIAM J. Optimization*, 2(3):435–449, 1992.
- [6] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optimization*, 2(4):575–601, 1992.
- [7] Y.E. Nesterov and A.S. Nemirovskii. *Interior Point Polynomial Methods in Convex Programming*. SIAM publications, 1994.
- [8] K. Schittkowski, Ch. Zillober, and R. Zotemantel. Numerical comparison of nonlinear programming algorithms for structural optimization. *Structural Optimization*, 7:1–19, 1994.
- [9] K. Svanberg. The Method of Moving Asymptotes – a new method for structural optimization. *Int. J. Num. Meth. Eng.*, 24:359–373, 1987.
- [10] K. Svanberg. A globally convergent version of MMA without linesearch. In N. Olhoff and G.I.N. Rozvany, editors, *Proceedings of the First World Congress of Structural and Multidisciplinary Optimization*, pages 9–16. Pergamon, 1995.
- [11] K. Svanberg. Two primal-dual interior-point methods for the MMA subproblems. Technical Report TRITA/MAT-98-OS12, Department of Mathematics, KTH, Stockholm, 1998.
- [12] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 1996.
- [13] S.J. Wright. *Primal-Dual Interior-Point Methods*. SIAM publications, 1997.
- [14] Ch. Zillober. A practical interior point method for a nonlinear programming problem arising in sequential convex programming. Technical Report TR98-1, Informatik, Universität Bayreuth, WWW: [www.uni-bayreuth.de/departments/math/~czillober/papers/tr98-1.ps](http://www.uni-bayreuth.de/departments/math/~czillober/papers/tr98-1.ps), 1998.

- [15] Ch. Zillober. A combined convex approximation – interior point approach for large scale nonlinear programming. *Optimization and Engineering*, 2(1):51–73, 2001.
- [16] Ch. Zillober. Global convergence of a nonlinear programming method using convex approximations. *Numerical Algorithms*, 27(3):256–289, 2001.
- [17] Ch. Zillober. SCPIP — an efficient software tool for the solution of structural optimization problems. *Structural and Multidisciplinary Optimization*, 24(5):362–371, 2002.
- [18] Ch. Zillober, K. Schittkowski, and K. Moritzen. Very large-scale optimization by sequential convex programming. *Optimization Methods and Software*, 19(1):103–120, 2004.