Quadratic Programming Methods for Tailored Reduced Hessian SQP

C. Schmid, L. Biegler

EDRC 06-159-93

Quadratic Programming Methods for Tailored Reduced Hessian SQP

Claudia Schmld and Lorenz T. Biegler *
Chemical Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213

• Author to whom correspondence should be addressed

Abstract

Reduced Hessian Successive Quadratic Programming (SQP) has been shown to be well suited for the solution of large-scale process optimization problems with many variables and constraints but only few degrees of freedom. The reduced space method Involves four major steps: an *Initial preprocessing phase* followed by an iterative procedure which requires the *solution of a set of linear equalities,* the *QP subproblem* and a *line search.* Clearly, the overall performance of the algorithm will depend directly on the robustness and computational efficiency of the techniques used to handle each of these subtasks.

The preprocessing phase solves a linear feasibility problem corresponding to the original nonlinear programming formulation. This step serves to determine an initial consistent point, to select a nonsingular set of basis variables (required for the definition of the search subspaces) and to identify linear dependency among the equality constraints. We demonstrate how Fourer's piecewise-llnear simplex techniques allow us to solve a smaller initialization problem more efficiently than is possible with standard simplex techniques.

Also, while the relative number of degrees of freedom is generally small, the *actual* number may become large. In this context, we present a new QP solver, QPKWIK, based on the dual algorithm of Goldfarb and Idnani. A unique feature of this algorithm is that it only requires the inverse Cholesky factor of the Hessian matrix to be supplied. At each iteration, this inverse Cholesky factor is obtained directly using a factorized inverse BFGS formula. The resulting solution technique for the QP subproblem is $O(n^2)$ with respect to the degrees of freedom of the problem, as opposed to most existing software which involves $O(n^3)$ operations. Further, the unconstrained optimum is dual feasible, which precludes the need for phase I calculations, and makes this method superior even for problems with few degrees of freedom. QPKWIK has been implemented so as to enhance the efficiency of the active set identification and is also able to determine a search direction when infeasible QP subproblems are encountered by relaxing the equality constraints without violating the simple bounds on the variables.

Finally, numerical results are included, both to illustrate the advantages of the proposed techniques and to assess the overall performance of the reduced Hessian method. We note that this approach is especially well-suited for process and real-time optimization and demonstrate this on several distillation problems as well as the Sunoco Hydrocracker Fractionation problem.

1. Introduction to Reduced Hessian SQP methods

Reduced Hessian Successive Quadratic Programming (SQP) has been shown to be well suited for the solution of large-scale process optimization problems with relatively few degrees of freedom. Here, we first briefly describe this method and motivate the improvements described in this study. We consider the nonlinear programming problem:

$$\begin{array}{ll} \text{Min} & f(z) \\ \text{s.t.} & h(z) = 0 \\ & z^L \le z \le z^u \end{array} \tag{PI}$$

where $f : \mathrm{SR}^n \to \mathfrak{R}$ and $h : \mathfrak{R}^n \to \mathrm{SR}^m$. This formulation is not restrictive since any inequalities can be converted to equality constraints through the addition of slack variables. At the kth iteration, the successive quadratic programming method generates a search direction d_k by solving the quadratic programming subproblem (P2).

$$\begin{array}{ll} \underset{d \in \mathfrak{R}^n}{\text{Min}} & \mathrm{Vflz J}^1 {}^{\wedge} + |d_k{}^T B_k d_k \\ \text{s.t.} & hfzj + Vh^{\wedge}zJ\, d_k = 0 \\ & z^L \le z^{\wedge} + d_k \le z^u \end{array} \tag{P2}$$

B_k denotes the Hessian of the Lagrange function or its approximation at iteration k. The reduced space SQP method results from a suitable change of basis representation applied to (P2). The new basis vectors are obtained by partitioning the search space into two subspaces;

$$Z \in \mathfrak{R}^{n \times (n-m)} \qquad \text{s.t. } \nabla h^T(z_k)\, Z_k = 0 \tag{1.1}$$

$$Y \in \mathfrak{R}^{n \times m} \qquad \text{s.t. } [Y \quad Z] \text{ is nonsIngular} \tag{1.2}$$

Thus, Y and Z together span the entire search space and the search direction d_k can be expressed as the sum of its components in the two subspaces.

$$\mathbf{|} = Y p_Y + Z p_z \tag{1.3}$$

As long as conditions (1.1) and (1.2) are satisfied, the choice of Y and Z is essentially arbitrary. Two methods in particular have received considerable attention in recent years. Let us partition the variables, z, into dependent variables $y \in \mathrm{SR}^m$, with $V_y h^T$

nonsingular, and independent variables $x \in \mathfrak{R}^{m+m}$. The coordinate basis method (Gabay, 1982; Locke *et al* 1983) uses

$$Z = \begin{bmatrix} I \\ -(\nabla_y h^T)^{-1} \nabla_x h^T \end{bmatrix} \qquad Y = \begin{bmatrix} 0 \\ I \end{bmatrix} \qquad (1.4)$$

while the orthogonal basis method (Vasantharajan and Biegler, 1988) is based on the following decomposition:

$$Z = \begin{bmatrix} I \\ -(\nabla_y h^T)^{-1} \nabla_x h^T \end{bmatrix} \qquad Y = \begin{bmatrix} \nabla_x h (\nabla_y h)^{-1} \\ I \end{bmatrix} \qquad (1.5)$$

For a detailed comparison of these two methods, see Schmid and Biegler (1993).

As derived in Schmid and Biegler (1993), it is easily seen from (1.1) and (1.3) that the linearized equality constraints in (P2) are reduced and p_y is fully determined through the solution of a set of linear equations

$$P_y = {}^{\prime\prime} \left[\nabla h_k^T Y_k \right]^{-1} h_k \qquad (1.6)$$

Once the Y space move has been calculated, terms involving py can be treated as constant and the QP subproblem (P2) reduces to

$$\text{Min} \quad [z^T \nabla f I z_k) + Z^T B Y p_y]^T p_z + \tfrac{1}{2} P_z^T Z^T B_k Z p_z$$

$$p_z \in \mathfrak{R}^{n-m}$$

$$\text{s.t.} \quad z^L - Z k - Y p_y \ \pounds \ Z p_z \ \tilde{<} \ z^u - Z j^\wedge - Y p y \qquad (P3)$$

This reduced QP is solved in the space of the independent variables and, for most process optimization applications, (P3) will be considerably smaller that (P2). Since the second order information required for the objective function of (P3) is often not available analytically, it must be obtained through other means. The reduced Hessian, $Z^T B Z$. is expected to be positive definite at the solution of the NLP. Consequently, this matrix can be approximated by positive definite quasi-Newton update formulae such as BFGS. The other second order term, $Z^T B Y p_y$, is ignored in most reduced space SQP applications because it is assumed that p_y (the "Newton step") converges to zero faster than p_z. For cases in which $Z^T B Y p_y$ is not negligible, Biegler *et al* (1993) proposed the inclusion of a second order correction term calculated via finite differences or a Broyden update formula.

The first order Kuhn-Tucker conditions which yield the multiplier values are given by

$$Y^TBYp_y + Y^TBZp_z + Y^TVhX = -Y^TVf \tag{1.7}$$

Since exact values of the multipliers are only required at the solution of the NLP, once the method has converged and $p_Y = p_z = 0$, these multipliers can be estimated using

$$X = -f^{\wedge}Vh)^{-1} Y^TVf \tag{1.8}$$

We now outline the reduced SQP algorithm which forms the basis for this paper.

A. Preprocessing Phase

Set iteration counter k to O

A.1 Evaluate the Junctions J and h as well as the gradients Vf and Vh at z_k.

A.2 Introduce slack variables and solve the linear feasibility problem (see P8) at z_k.

A.3 If all slack variables are zero go to AA. Else, perform an Armifo line search along the direction d_k generated by a Phase I linear program (P8) to minimize the Lrl penalty function (4.1). This gives a step size $a>_k$.
Set $z_{k+1} = z_{Jc} + 6)_kd_k$, Jc:=fc+j and go to A. 1.

AA Identify the basic variables as well as any redundant equality constraints which will be handled by the QP.

B. Optimization Phase

Set iteration counter k to O
Initialize the reduced Hessian (usually to the identity matrix).

B.1 Determine the LU factors for the system of basic variables and linearized equality constraints; calculate the Y space move from (1.6).

B.2 If applicable, determine the second order correction term $2^{\wedge}BYp_r$

B.3 Solve the reduced QP subproblem (P3) to yield p_r

BA Reconstruct the search direction d_k using (1.3) and estimate the Lagrange multipliers using (1.8).

B.5 If the convergence tolerance is satisfied, STOP. Else, go to C.I

C. Une Search Phase

C.1 Perform an Armijo line search along the search direction d_k obtained in BA to yield the search step a_k. The merit function we use is based on an augmented Lagrangian approach due to Biegler and Cuthrell (1985).
$$= k+1$$

C.2 Evaluate J and h as well as Vf and Vh at z_k. If the basis becomes singular return to A.2.

C.3 Update the reduced Hessian using the BFGS formula with Powell damping (Powell 1977). Go to B.I.

In this paper we focus on steps B.3 and A of the above algorithm. They are discussed in Sections 3 and 4 respectively. Step B. 1 is briefly addressed in Section *2*. Details on the decomposition strategy that is used, are included as part of the numerical results where appropriate.

In addition, each section contains numerical results for a variety of general test problems which illustrate the performance of the improvement discussed in that section. Finally, results for a number of process optimization examples are presented in Section 5. These will be used to assess the overall behavior of the algorithm and demonstrate its advantages for both process and real-time optimization. In particular, we include a case study of the Sunoco Hydrocracker Fractionation problem and compare our reduced Hessian algorithm with MINOS (Murtagh and Saunders, 1982, 1987).

2. Determination of the T space move; tailoring the algorithm

As discussed above, the Y space move, p_Y, is obtained from the solution of a set of linear equations

$$Pr = - \left[\nabla h_k^T Y_k\right]^{-1} h_k \tag{1.6}$$

Using coordinate bases (1.4), this reduces to

$$Py = -V_y h_k^{-T} h_k \tag{2.1}$$

Model sparsity is maintained and determination of the Y space move is equivalent to determining a Newton step for solving $h(z) = 0$. The matrix to be factored is of dimension *mxm;* for most process engineering applications it will be very large but considerably sparse. It follows that integration of a sparse equation solver within the SQP algorithm is essential for efficient calculation of the Y space move. While a number of sparse linear equation solvers are available, the set of MA28 subroutines from the Harwell library seems adequate for our purpose. The variables and equations are first

scaled using MC19 so as to reduce the chance of ill-conditioning. At the first iteration, the equations are solved using MA28AD which determines an initial pivot sequence. MA28BD is used at subsequent iterations; it uses the same pivot sequence to solve systems with different coefficients but the same sparsity pattern. This is important since determination of the pivot sequence is computationally intensive. If the matrix becomes ill-conditioned, the algorithm returns to MA28AD and determines a new set of pivots. The advantages of sparse equation solvers, as compared to dense methods has been widely accepted in the literature.

To illustrate the performance of our reduced Hessian algorithm with respect to increasing problem size, consider Example 1, below.

Example 1.

$$Min \quad \frac{1}{2} \sum_{i=1}^{n} x_i^2$$

$$s.L \quad x_k(x_j - 1) - 10 x_j = 0 \qquad J = 1, n\text{-}100$$

where k is equal to J/100 rounded up to the nearest integer.

We can arbitrarily increase the size of the problem to be solved by increasing the
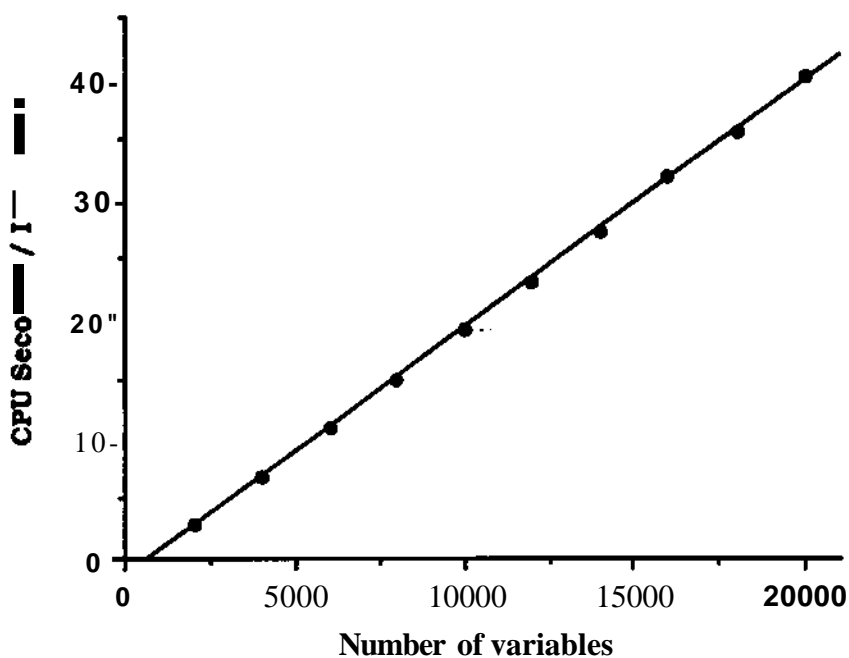

Figure 1. CPU seconds per iteration on a DEC5000/200 as a function of Increasing number of variables for Example 1.

number of variables, n. Here, the number of degrees of freedom, (n-m), remains constant at 100. Hence the increase in effort as the number of variables increases will be due, primarily, to the extra computations required to factorize the matrix $V_y h^T$, as required for the calculation of the Y space move according to (2.1). Since the number of nonzero elements in the Jacobian increases linearly as a function of n without complicating its structure, we expect our algorithm, using the sparse equations solver MA28 to yield the matrix factorizations, to scale linearly with the size of Example 1. Example 1 is solved on a DEC 5000/200; 6 to 8 iterations are requires for convergence within a tolerance of 10^{-10}, depending on problem size. CPU time per iteration as a function of n is shown in Figure 1. As expected, our algorithm scales linearly with the size of Example 1.

For specific classes of problems we can still do better, however, than is possible with a generic equation solver such as MA28. To illustrate this, we consider the example of optimizing the operating conditions of a distillation column using an equation based approach. Using constant molal overflow to simplify the state equations, the mass balance for component j on tray i of the column is given by

$$v_{i-1,j} + f_{i,j} - (1 + \frac{S_i^L}{L_i}) l_{i,j} - (1 + \frac{S_i^v}{V_i}) v_{i,j} + L_{n} = 0 \qquad (2.2)$$

where f, v and 1 denote the component feed, vapor and liquid flowrates. V and L are the overall flowrates and S refers to the contribution of sidestreams. It is easy to see that the equation for tray 1 depends exclusively on the variables for trays i-1, i and i+1. In effect, the Jacobian matrix for the model equations has a block tridiagonal structure. This structure is exploited by the Naphthali-Sandholm model, UNIDIST, which is part of the SEPSIM process simulator (Anderson *et* aL, 1991) and employs an efficient Thomas algorithm to obtain the Newton step for the distillation model. As discussed in Schmid and Biegler (1993), this *existing solution* and decomposition strategy can be integrated within the reduced SQP algorithm and used to obtain the Y space move. Thus, a tailored optimization algorithm is obtained which exploits the structure of the Jacobian as well as any economized procedure to calculate the Newton step. Numerical results for several distillation examples illustrating the benefits of using specialized solution techniques are included in Section 5.

Developing a special solution procedure for every problem is clearly an impossible endeavor. However, we find that a number of suitable and efficient Newton-based

methods are readily available and have been implemented in a very robust manner. Thus, the ease with which the reduced Hessian SQP algorithm can be tailored to take advantage of the sparsity pattern of various classes of problems provides a strong motivation for using this optimization technique.

3. Solution of the QP subproblem

Decomposition strategies for SQP are best suited for problems with few degrees of freedom, since the QP subproblem then only needs to be solved in a small subspace of the process variables. While this formulation encompasses many process engineering problems, in particular, most applications involving process flowsheets, there are situations for which the number of degrees of freedom can grow significantly. This is the case, for example, in optimal control, multiperiod design and data reconciliation. While the number of independent variables is still relatively small compared to the total number of variables, the QP could become quite large. In addition, even if the number of variables involved in the QP subproblem is small, the number of inequality constraints, arising from bounds on the variables in (PI), that need to be considered can be substantial. To successfully handle a broader range of applications it is therefore essential to consider the efficiency of the QP subproblem solution procedure.

Currently, most implementations of the reduced SQP algorithm employ QP routines such as QPSOL (Gill *et aL,* 1983), which are based on active set strategies applied to a primal algorithm for the solution of the quadratic programming problem. Considerable effort is expended in determining an initial feasible point for the QP, usually via a phase I LP solution, and in factorization of the Hessian matrix. The work discussed in this section was motivated by the algorithm proposed by Goldfarb and Idnani (1983) and the accompanying Harwell code, VE17AD, by Powell (1985). The most obvious advantage of this dual algorithm is that the unconstrained minimum of the QP objective function provides an initial dual feasible solution, and the initialization phase is thus eliminated. However, for a large number of independent variables, the effort involved in the LL^T factorization of the Hessian at each SQP iteration is still significant.

We propose a new QP solution technique, QPKWIK, based on the original algorithm by Goldfarb and Idnani but modified substantially to account for the fact that the QP is only a subproblem of the SQP algorithm and will have to be solved repeatedly. First we describe this dual method and outline the QP algorithm. We then highlight the major

advantages of QPKWIK over QPSOL and VE17AD. Finally, numerical results are Included to illustrate these differences.

3.1 A dual method for the solution of QPs

The QP subproblem (P3) can be written as follows:

$$\text{Min} \quad g^T x + \tfrac{1}{2} x^T G x$$
$$\text{s.t.} \quad A^T x \; Z \; b \tag{P4}$$
$$\mathbf{x\underline{t} \; \leq \; x \; {}^\wedge \; X\underline{u}}$$

Considering only the active inequalities and bounds, this can be rewritten as

$$\text{Min} \quad g^T x + \tfrac{1}{2} x^T G x \tag{P5}$$
$$\text{s.t.} \quad A^T x \; = \; b$$

The Karush-Kuhn-Tucker conditions for this QP are given by

$$\begin{bmatrix} \mathbf{C} & \mathbf{-A} \\ \mathbf{-A^T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} x^* \\ \mu^* \end{bmatrix} = - \begin{bmatrix} g \\ b \end{bmatrix} \tag{3.1.1}$$

$$\lambda i^* \; \pounds \; 0 \quad \text{for the active inequalities or bounds}$$

As noted by Fletcher (1987), the solution to this system of linear equations has the general form

$$\begin{bmatrix} x^* \\ \mu^* \end{bmatrix} = \begin{bmatrix} -H & D \\ D^T & -U \end{bmatrix} \begin{bmatrix} g \\ b \end{bmatrix} \tag{3.1.2}$$

The difference between the various methods that have been developed for the solution of quadratic programming problems then lies in the definitions of H, D and U. If the Hessian matrix, G, of (P5) is positive definite, explicit expressions for H, D and U are given by

$$H = G^{-1} - G^* A\,[A^T G^{-1} A]^{-1}\, G A^T G^{-1}$$
$$D = G^{-1} A [A J G^{-1} A]^{-1} \tag{3.1.3}$$
$$U = -\{A^T G^{-1} A\}^{-1}$$

Moreover, if G is positive definite, we can express it in terms of its Cholesky factors, $G = LL^T$. Then, using a QR factorization of $L^{-1} A$

$$G*A = L'TL^A = L^{-T}Q \begin{bmatrix} \cdots \\ 0 \end{bmatrix} \tag{3.1.4}$$

Also, define a matrix T such that

$$L^{-T}Q \begin{bmatrix} R \\ 0 \end{bmatrix} = L\text{-}MQ! \quad I \quad Q_2]\begin{bmatrix} R \\ n I \\ U \end{bmatrix} = p^* \quad I \quad T\ll I | \begin{bmatrix} R \\ n I \\ U \end{bmatrix} = T \ I \begin{bmatrix} R \\ \cdots \\ 0 \end{bmatrix} \tag{3.1.5}$$

Using the above definitions, (3.1.3) becomes

$$H = T^U_f T^U)^T$$
$$D = T^\circ R^{-T} \tag{3.1.6}$$
$$U = -R^{-1}R^{-T}$$

These expressions can be used, together with (3.1.2), to calculate x* and μ^*, the optimal solution of the QP (P4). once the correct active set has been identified.

During the course of the QP solution, however, the active set will change from one iteration to the next. Let S_L. L =1, 2, 3, ... be this series of active sets. Let AL be the matrix containing the currently active constraint normals, and define the corresponding T_L and RL from (3.1.5). S! is chosen to be empty such that x_x is the unconstrained minimum of the objective function in (P5), and Tj is defined as $Tj = L^{-T}$. For nonempty SL, T_L is updated so that it satisfies $T_L T_L^T = L^{mT} L^l$ and an upper triangular RL, where RL $= (T_L^C)^T A_L$, is maintained.

Let x_L be the optimum point of the QP with the current active set, SL Then,

$$XL = -T_L^u(T_L^u)^T g + T_L^C R_L^{-T} b_L$$
$$\mu_L = R_L^{-1}(T_L^C)^T g + R_L^{-1} R_L^{-T} b_L \tag{3.1.7}$$

Let XL_{+1} be the optimum point of the QP with the active set S_L u constraintKNEXT- This point is also a feasible point of the active set S_L. We thus have

$$b_L = A_L^T x_{L+1}$$
$$g = -GX_{L+1} + A_I> l_{L+1} + a_{KN} EX!^KNEXT \tag{3.1.8}$$
$$= -LL^T x_{L+1} + A_L |i_{L+1} + a_{KNEXT} t$$

where t is chosen so as to ensure feasibility and will equal the multiplier, MKNEXT- Using these expressions for g and b_L, we obtain

$$\begin{aligned}
x_L &= - T_L{}^U(T_L{}^U)^T \left[- LL^T x_{L+1} + A_1\mu_{L+1} + a_{KNEXT}t \right] + T_L{}^C R_L{}^{-T} A_L{}^T x_{L+1} \\
&= T_L{}^U(T_L{}^U)^T \left[LL^T x_{L+1} - a_{KNEXT}t \right] + T_L{}^C R_L{}^{-T} A_L{}^T x_{L+1} \\
&= - T_L{}^U(T_L{}^U)^T a_{KNEXT}t + [T_L T_L{}^T - T_L{}^C(T_L{}^C)^T]LL^T x_{L+1} + T_L{}^C(T_L{}^C)^T LL^T x_{L+1} \\
&= x_{L+1} - T_L{}^U(T_L{}^U)^T a_{NEXT}t \quad\quad\quad (3.1.9)
\end{aligned}$$

$$\begin{aligned}
\mu_L &= R_L{}^{-1}(T_L{}^C)^T \left[- LL^T x_{L+1} + A_1\mu_{L+1} + a_{KNEXT}t \right] + R_L{}^{-1} R_L{}^{-T} A_L{}^T x_{L+1} \\
&= - R_L{}^{-1}(T_L{}^C)^T LL^T x_{L+1} + \mu_{L+1} + R_L{}^{-1}(T_L{}^C)^T a_{KNEXT}t + R_L{}^{-1}(T_L{}^C)^T LL^T x_{L+1} \\
&= \mu_{L+1} + R_L{}^{-1}(T_L{}^C)^T a_{KNEXT}t \quad\quad\quad (3.1.10)
\end{aligned}$$

Therefore, as constraints are added to the active set, the recursion formulas used within the dual algorithm to update the search direction and the multipliers **are** given by

$$\begin{aligned}
x_{L+1} &= x_L + T_L{}^U(T_L{}^U)^T a_{KNEXT}t \\
\mu^+ &= \begin{bmatrix} \Lambda \\ t \end{bmatrix} = H_L + \begin{bmatrix} - R_L{}^{-1}(T_L{}^C)^T a_{KNEXT} \\ 1 \end{bmatrix} t \quad\quad\quad (3.1.11)
\end{aligned}$$

3.2 QPKWIK Algorithm

Define   *NACT*   = number of active constraints (dimension of SJ

              KNEXT   = index of constraint to be added to SL

              KDROP = index of constraint to be removed from S_L

0. *Find the unconstrained minimum* $x = - G^{\prime\prime}g = - L^\Lambda L^{-1}g$
   Set $T_1 = L^{-T}$, *NACT* = 0
1. *Check for violation of any of the inactive inequality constraints*
      • *If all are satisfied, the current solution x is both feasible and optimal STOP*
      • *Otherwise, set KNEXT to the index of the most violated constraint* $M^* = I \begin{bmatrix} \mu \\ 0 \end{bmatrix}$
2. *(a) Determine the search direction in the primal space*
   $z = TjCTtfTaKNExr$
   and, if NACT>0. the search direction in the dual space
   $r = RLWIPF\ OKNEXT$
   (b) Compute step length
      (i)   Compute t_{lt} *maximum step in dual space without violating dual feasibility*
         If NACT=0 or r<0 $t_x = \sim$
         Else $ti = \min\left\{ \wedge : i) > 0 \right\}$ *KDROP* =J

(ii) Compute t_2, mintmum step in primal space such that constraint KNEXT
becomes feasible

If $|z| = O$ $t_2 = \infty$

Else $t_2 = \dfrac{(b_{KNEXT} - a_{KNEXT}^T x)}{z^T a_{KNEXT}}$

Step length, $t = min\,(UXQ)$

(c) Take step

(i) No step in primal or dual space

$t = «>$ QP is infeasible, STOP

(ii) Step in dual space

$ta{-}\varphi o$ but U is finite $i? = k^* + t \begin{bmatrix} -r \\ 1 \end{bmatrix}$

Drop constraint KDROP. update T_L and RL, set NACT = NACT-1, GOTO2(a)

(iti) Step in primal and dual space

$X = X + tZ$

$1? = It + t \begin{bmatrix} -r \\ 1 \end{bmatrix}$

If $t=t_2$ Add constraint KNEXT to $S^\wedge$ update T_L and Rb, set
       NACT=NACr+1, *GOTO1*

If $t=ti$ Drop constraint KDROP. update T_L and Rt, set NACT = NACT-1,
       GOTO2(a)

3.3 Reduced Hessian Approximation

Both QPSOL and VE17AD require the Hessian of the QP objective function to be supplied as an input parameter. In QPSOL, this matrix is projected into the space of the active constraint set and factorized via a modified Cholesky factorization. Goldfarb and Idnani, on the other hand, apply a Cholesky factorization to the original Hessian and then find the Inverse of the Cholesky factor. In both cases, the matrix factors are updated to account for the addition and removal of constraints from the active set within one QP iteration so as to avoid complete refactorization. However, $O(n^3)$ steps are required at every SQP iteration for the initial factorization of the Hessian of the QP subproblem.

The reduced SQP algorithm uses an approximation to the dense reduced Hessian, G_k, calculated via the BFGS formula. We can eliminate the factorization required by the Goldfarb and Idnani algorithm by updating the Cholesky factor L_k, where $G = L_k L_k^T$, instead. The update formula to use in this case is given by

$$N_{k+1}i \;=\; I^* \;+\; \frac{(y - cULiJs)\, cs^1 !^\wedge}{}$$

(3.3.1)

where
$$c = \left(\frac{y^T s}{s^T L_k L_k^T s}\right)^{1/2}$$

$$y = (Z^T \nabla L)^{k+1} - (Z^T \nabla L)^k$$

$$s = a\,p_z$$

N_{k+1} is not lower triangular but can easily be transformed to L_{k+1} using Givens rotations (Dennis and Schnabel, 1986). This procedure requires $O(n^2)$ operations. We can still do better, however. First, the term $1+W^* = G_k s = -(2?V^* + Z_A^T \setminus i)$ and G_k never need to be calculated. (Z_A are the rows of Z that represent active bounds.) Also, the QP algorithm works directly with the inverse of the Cholesky factor. The inverse of the above rank one update formula (3.3.1) gives the following expression through a Householder relation:

$$N_{k+1}^{-1} = L_k^{-1}\left(I - \frac{(y + c\,(Z^T \nabla \phi \quad Z_A \quad))}{s\,y} \right)$$

(3.3.2)

Again, N_{k+1}^{-J} is transformed to the lower triangular matrix $I^* {}^{\wedge - l}$ via Givens rotations. We have implemented this inverse update for QPKWIK. The update can be done efficiently using Givens rotations and requires $O(bi^2)$ operations. Also, the criterion for Powell damping remains unchanged from that generally used in the context of the BFGS formula and positive definiteness of the reduced Hessian is always maintained.

3.4 Bounds and sparsity of the reduced Jacobian

In VE17AD, all constraints, whether bounds or inequality constraints, are treated alike and must be checked for violation at each QP iteration. When we examine the structure of the QP subproblem of our reduced SQP algorithm, we find upper and lower bounds on the variables but also a large number of inequality constraints of the form

$$LB \lessgtr a^T x \leq UB$$

resulting from the original upper and lower bounds on the dependent variables. We account for this special structure within QPKWIK. If either the upper or lower bound is active, the constraint is labeled active and not checked, since both bounds cannot be active simultaneously. If both are inactive, then $a^T x$ is calculated only once and both bounds are checked for violation. This saves considerable time since the number of constraints to be checked is essentially halved.

Another option within QPKWIK, which is not Included In QPSOL or VE17AD, considers only the nonzero elements of the reduced Jacobian *A* in (P4). While the reduced Jacobian will be considerably less sparse than the full Jacobian, the number of nonzero elements may still be relatively small, especially for problems with more degrees of freedom. Therefore sparsity is checked at the first iteration and if the reduced Jacobian is found to be less than 50% dense, the calculation of $a^T x$ is restricted to the nonzero elements of *A*.

3.5 Warm starts

The first few times the quadratic subproblem is solved, the active set is likely to change considerably from one SQP iteration to the next. However, once the correct active set of the NLP (PI) has been identified, the work per iteration can be reduced by using the constraints active at the previous iteration as an initial guess for the current iteration. While QPSOL has a warm start option, there is no such provision within VE17AD.

Our new algorithm, QPKWIK, can take advantage of warm starts. However, there is a trade-off between reducing the size of the set from which potentially active constraints are selected and adding the wrong constraints which must subsequently be dropped. Apart from the effort Involved in choosing a violated constraint to add to the active set, most of the work at each QP iteration is dedicated to updating T using Givens rotations in order to maintain an upper triangular matrix R. *N-nact-1* Givens rotations are associated with adding a constraint while dropping a constraint requires *nact-kdrop-1* rotations, where n is the number of variables, *nact* the number of active constraints and *kdrop* the index of the constraint to be dropped. *Kdrop* and *nact* are bounded between 1 and n. Clearly, the penalty for dropping constraints may be considerable, and care must be taken to minimize the number of incorrect constraints that are added. With this in mind, we only use a warm start if a full search step was taken at the previous SQP iteration. To select which constraints to make active for the warm start, we order the constraints according to decreasing multiplier value scaled by the norm of the constraint gradient. The algorithm then checks down the list, adding those constraints that are violated. In this way we hope to add those constraints that will result in the greatest increase in the objective function and prevent the addition of unnecessary constraints.

3*6 Handling infeasible QPs

At the initial point, the preprocessing step discussed in Section 4 of this paper serves to guarantee the existence of a feasible region and consequently the initial QP subproblem will be feasible. At subsequent iterations, however, the algorithm may move to a point where the QP becomes infeasible. Rather than return to the preprocessing phase, it is usually sufficient to relax the feasible region and allow the algorithm to take the resulting step, in the hope that it will eventually move to a point where the QP again has a solution. Such a relaxation is possible with QPSOL which, when failing to satisfy the desired tolerance, determines a point that minimizes the total Infeasibility. While this approach may often be adequate, it cannot account for constraints which must *never* be violated. For process optimization problems, in particular, bounds on the variables usually reflect actual physical bounds which, if violated, may cause numerical difficulties. Within QPKWIK we therefore allow the feasible region to be relaxed such that the equality constraints can be violated but upper and lower bounds on the variables are always enforced. At the level of the undecomposed QP, (P2), this is achieved through the addition of an additional variable, £, and (P2) becomes

$$\underset{z \in \mathfrak{R}^n;\, \xi}{\text{Min}} \quad \nabla f^T d + \tfrac{1}{2} d^T B d + M f \xi + \xi)$$
$$\text{s.t.} \quad h\,(1 - \S) + \nabla h^T d = 0 \qquad\qquad (P6)$$
$$z^L \le z \le z^u$$
$$\% \geq 0$$

where M is a large number. After decomposition, the Y and Z space moves are calculated from

$$PY = -[{}^{\nabla h}k^{T Y}k]^{-1 h}k \qquad\qquad (3.6.1)$$

and

$$\underset{p_z \in \mathfrak{R}^{n-m}}{\text{Min}} \quad [z^T \nabla f | z_k) + (1^\wedge) Z^T B Y p_y]^T p_z + \tfrac{1}{2}|p_z{}^T Z^T B_k Z p_z + M\,(\xi + \tfrac{\xi^2}{2})$$
$$\text{s.t.} \quad z^l - 7^\wedge \le Z p_z + (1 - £)\,Y p_y \le z^u - z^\wedge \qquad\qquad (P7)$$
$$\backslash \, * \, 0$$

$$d = Z p_z + (1 - y\,Y p_y$$

Because of the laige penalty on the extra variable, \ is driven to zero whenever possible, and has a natural upper bound of 1. If £ = 0 the original quadratic programming subproblem (P3) is feasible and the additional variable does not affect the search

direction. For $0 < £ \le l_f$ the Y space move is damped, causing the linearized equality constraints to be violated. A solution to (P7) exists, though, and the SQP algorithm is able to move to a new point. At this stage, we perform a line search which minimizes the LI norm of h(x). Finally, one particular solution which may be encountered is $£ = 1$ and $p_z = 0$. This results in a zero search direction and requires the algorithm to return to the preprocessing phase before further progress can be made.

3.7 Numerical results

The differences between QPSOL and QPKWIK are illustrated with two examples. First we consider a variation of Example 1; it has a quadratic objective function and nonlinear equality constraints, with no bounds on the variables.

Example 2.

$$Mtn \quad \frac{1}{2} \sum_{i=J}^{n} x_i^2$$

$$s.t \quad x_j (x_{10+j} - 1) \qquad {}_{io+j} = 0 \qquad J'1.10 \tag{3.7.1}$$

Variables 1 to 10 are chosen to be independent and a tolerance of 10^{-15} is used. We apply the coordinate method with no correction since it requires the least operations outside of the solution of the QP subproblem and thus allows us to best isolate the work
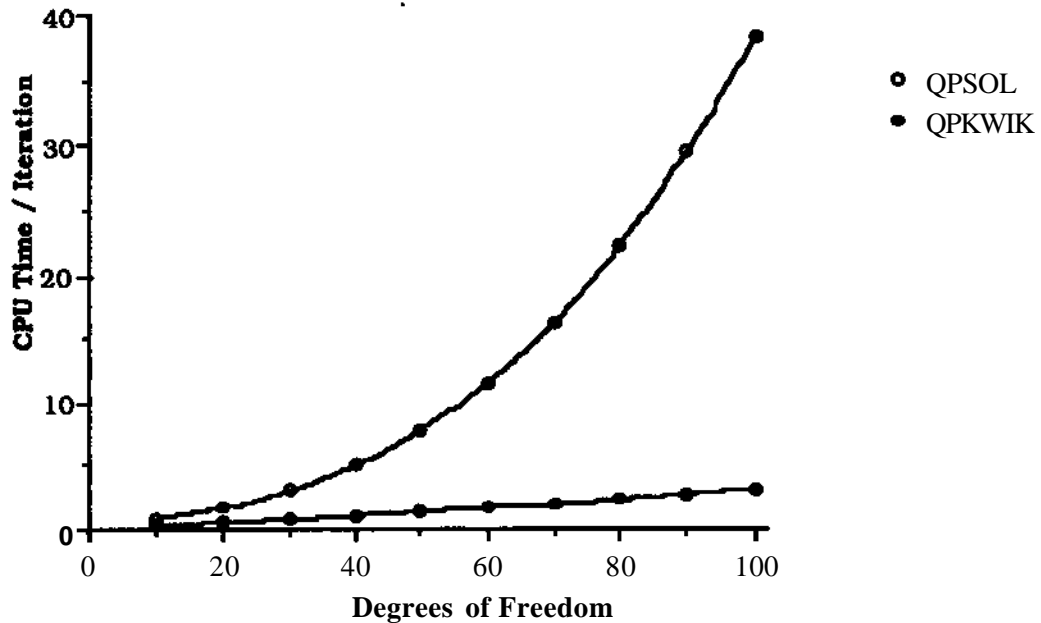

Figure 2. Comparison of CPU seconds per iteration for Example 2 using QPSOL and QPKWIK.

required to determine the Z space move. The results in Figure 2 show CPU seconds per iteration required by the QP routine on a SUN3 workstation. For this problem, there are no bounds on the variables, hence the difference in times between QPSOL and QPKWIK reflects the additional time required by QPSOL to factorize the Hessian. As expected, QPSOL requires $O(n^3)$ operations while QPKWIK requires $O(n^2)$. When the ratio of times per iteration of QPSOL over QPKWIK is plotted, the resulting graph is essentially linear.

The next example is a multiperiod problem, where the size of the problem can be adjusted by changing the number of periods, *NP*. *The* total number of variables is given by *5+3NP* and the formulation includes *2NP* equality constraints. In addition, this problem also has lower bounds on *2NP* variables, approximately half of which become active at the optimum solution. Example 3 is therefore well-suited for studying the effect of a warm start strategy.

Example 3.

$$Min \quad \sum_{k=1}^{NP} (a_k x_{k+5}^2 + b_k x_1 + c_k x_2) + x_3^2 + x_4 \cdot x_5$$

$$s.L \quad x_1^2 + x_3^2 - x_4 + x_5^2 - c_k x_{k+5} + x_{NP+k} = 0 \qquad k = 1, NP$$

$$2x_1^2 - x_2 + 3x_3^2 + x_4^2 - x_5 + a_k x_{k+5} + x_{2NP+k} = 0 \qquad k = 1, NP$$

$$x_t \geq 0 \qquad\qquad t = JVP+6,5+3JVP$$

$$a^\wedge e \, 12,61 \qquad\qquad\qquad\qquad\qquad (3.7.2)$$

$$b_k e \, [-25.-10]$$

$$c_k \, e \, [10.27]$$

The number of iterations and CPU seconds on a SUN4 required to solve Example 3 within a tolerance of 10^{-6} using coordinate bases are reported in Table 1.

NP	QPKWIK		QPSOL	
XTT>	Cold start	Warm start	Cold start	Warm start
20	22 (1.6)	22 (1.3)	23 (3.2)	23 (2.8)
45	29 (12.2)	29 (9.4)	28 (29.0)	28 (25.7)
70	25 (36.0)	25 (25.2)	35 (126.9)	35 (140.4)
95	29 (109.3)	29 (95.3)	29 (260.7)	29 (252.0)
120	30 (210.7)	30 (177.3)	27 (443.4)	27 (407.3)
145	26(356.1)	26 (266.7)	31 (961.6)	31 (800.1)
170	26 (553.9)	26 (431.1)	29 (1353.6]	29 (1310.2)

Table 1. Number of iterations and CPU seconds on a SUN4 (in brackets) for Example 3, using QPKWIK or QPSOL.

From the CPU seconds reported in Table 1 it is clear that QPKWIK is able to solve the QP subproblem significantly faster than QPSOL. Even though the active set changes a lot during the first few iterations, it appears that the warm start strategy outlined above is able to minimize the number of incorrect constraints added. By using activity information from the previous iteration once the algorithm gets close to the solution, the amount of work involved in identifying the active set is reduced considerably and a further reduction of 10% to 30% in CPU seconds is achieved.

A criterion which may be even more important than efElciency in assessing the quality of the QP algorithm is that of robustness. The dual algorithm initially finds the unconstrained optimum and then adds constraints to the active set until primal feasibility is achieved. In exact arithmetic, a positive definite Hessian will cause the QP objective function to increase strictly monotonically as new constraints are added and thus cycling of active sets is prevented, even with constraint degeneracies. Moreover, since the number of constraints that can be active is bounded by the number of variables in the QP, the dual algorithm converges in a finite number of steps. In practice though, the above result is not guaranteed because of round-off error. Care has been taken within QPKWIK to minimize round-off error and tests similar to those used by Powell in VE17AD are included to identify constraint degeneracies and check for increases in the objective function. Problems with cycling have never been encountered. QPSOL, on the other hand, is a primal method and constraint degeneracies can cause cycling, even with exact arithmetic. In addition, QPSOL's performance was found to be very sensitive to the adjustable parameter *FEATOU* which controls constraint infeasibility. If set too tightly, QPSOL incorrectly reports that the QP is infeasible or terminates after an excessive number of QP iterations that indicate cycling. In addition, the value of *FEATOL* can affect the number of iterations required by the SQP algorithm because it may cause different active sets to be selected. It seems, therefore, that QPKWIK is superior to QPSOL as an algorithm for the solution of the quadratic programming subproblems within a reduced Hessian SQP method, both in terms of robustness and efficiency.

4. Preprocessing

The constraints of the quadratic programming problem at each SQP iteration are obtained by a linearization of the nonlinear constraints of the original NLP about the current point. A problem which frequently arises in successive quadratic programming, especially as the problems become larger and a "good" initial point is not available, is

that of inconsistent linearizations which, in turn, gives rise to infeasible QP subproblems. A simple way to handle this situation is to relax the feasible region and solve an altered QP subproblem. Such an approach is discussed, for example, in Biegler and Cuthrell (1985). They employ the QP routine developed by Gill and Murray (1978) which first solves a Phase I LP feasibility problem. If no solution to the LP is found, the method terminates and reports the minimum infeasibility (Afl). This allows the constraint violation tolerance to be temporarily increased and the relaxed QP to be solved to optimality. It is expected that the algorithm will eventually move to a point where the constraint linearizations will no longer be inconsistent. While this strategy should be adequate as long as the constraint Infeasibility remains small, a laxge value of *MI* could lead to a very poor search direction which may be detrimental to the robustness of the method.

In addition, we also require an automatic procedure to determine a nonsingular set of basis variables such that the Y space move and the Lagrange multipliers will be uniquely determined by (1.6) and (1.8) respectively. We find it useful, therefore, to resolve both of these issues via a preprocessing phase which is executed prior to decomposition. Such an approach is presented by Vasantharajan *et aL* (1990). The authors discuss the solution of the linear feasibility problem corresponding to (P2) obtained by augmenting the linearized equality constraints with two nonnegative artificial variables p_t and n_r This yields the following LP, where the objective function is the total infeasibility, given by the sum of these artificial variables.

$$\text{Mln} \quad \sum_{i=1}^{m} (p_i + n_i)$$

$$\text{s.t} \quad h, ^) + Vh^bg \, (x - x_k) = p, - n_t \quad i = 1, m \qquad \text{(P8)}$$

$$x^L \, \pounds \, x, \leq x^u$$

$$p, n, \geq 0 \qquad V \, i$$

Now if any p_4 or nj is basic at a nonzero value, an Armijo line search is performed along the search direction d_k generated by the LP. The merit function used to obtain the search step o^ is the L-l penalty function

$$6(x) = ^ I h J \qquad (4.1)$$
$$\sum_{1=1}^{m}$$

Once the LP has produced a consistent point, we check for dependent equality constraints. This is indicated by artificial variables p_t or $r^\wedge$ that are basic at zero. Any constraints that are identified as dependent are not used in the determination of Py as this would cause the matrix $Vh_k^TY_k$ to become singular. Instead, these constraints are passed directly to the QP. Thus, the method handles constraints that are degenerate initially but may become independent at later iterations.

The numerical results obtained by Vasantharajan *et aL* (1990), using MINOS 5.1 as the sparse LP solver, show the above procedure to be robust. However, the authors note that considerable effort is expended in the preprocessing phase, thus motivating the investigation of alternate linear programming algorithms. To overcome this problem, we consider the work by Fourer (1985, 1988, 1989) on piecewise-linear programming.

With Fourer's approach, we can reformulate the linear feasibility problem in a more compact manner.

$$\text{Min} \quad \sum_{j=1}^{m} s_i$$
$$\text{s.t.} \quad h^\wedge + Vh^\wedge tx^\wedge (x - x_k) = s_t \qquad i = 1, m \qquad (P9)$$
$$^\wedge \ \pounds \ Xfc \ \leq \ X^U$$

This relaxation Introduces only half as·many additional variables, and nonnegativity constraints on these variables are no longer required; it seems natural that this formulation would solve more efficiently than (P8). Note that traditional linear simplex algorithms are unable to handle the piecewise linear form of the objective function in (P9) directly. On the other hand, the piecewise-linear (P-L) programming algorithm permits the minimization of any convex separable piecewise-linear objective, subject to linear constraints. The convex separable piecewise-linear objective is characterized as the sum of piecewise-linear functions,

$$[c/7]x = \sum_{k=1}^{X} [c/\gamma]_k \, x_k \qquad (4.2)$$

each of which is defined by an increasing sequence of breakpoints $y_k^{(h)}$

$$- V'^{21} < V'' < V^{0\prime} < \vert^m < \vert^{<2)} - \qquad \{4.3\}$$

and an increasing sequence of slopes $\underline{c}^\wedge$

$$... c_k^{\wedge *} < c_k^{\{rl)} < c_k^{(0)} < c_k^{(1)} < c^* ... \qquad (4.4)$$

Comparing P-L programming to linear approaches based on transformations, Fourer notes that each iteration of the P-L simplex algorithm represents several iterations of a linear algorithm restricted to certain bases of an equivalent linear program, while requiring substantially the same work. In addition, the criteria for selecting variables to enter and leave the basis are more flexible in P-L programming. Hence, the P-L simplex algorithm is inherently more efficient than traditional linear simplex algorithms when faced with this class of problems.

Clearly, the semi-linear objective of (P9) is a special case of the more general situation considered by Fourer. There is only one breakpoint $Y_k^{(1)} = 0$ with one finite slope $c^{\wedge}s$ - 1 to its left and one finite slope $c j^{\wedge} l$ to its right. While we do not provide any direct numerical comparison between P-L programming and the more traditional approach employed by Vasantharajan *et aL*, results obtained by Fourer indicate the potential benefits in applying the P-L simplex algorithm to the problem at hand. For a multi-stage structural design problem of the form Ax=b, x£0 where A is (397x1375) and only 0.8% dense with a semi-linear cost objective, Fourer reports that his P-L simplex algorithm, CPLP, is consistently 2-3 times faster than the XMP LP package (Marsten, 1981).

Moreover, Fourer's implementation of piecewise-linear programming takes advantage of problem sparsity, allowing us to supply only the non-zero elements of the constraint Jacobian. In addition, specification of lower and upper bounds on the variables is straightforward. On output, the code provides information on which variables are basic and nonbasic, allowing us to identify a linearly independent set of linearized equality constraints and the corresponding nonsingular set of basis variables. In the event that the initial point is infeasible, Fourer's code generates a search direction which will minimize the sum of the infeasibilities, enabling us to move to a new point. In selecting an appropriate search step, we use the same procedure adopted by Vasantharajan *et aL*.

The first issue we address is that of using the preprocessing phase to select a consistent initial point. We consider two example problems, taken from Himmelblau (1972) and Biegler and Cuthrell (1985) respectively.

Example 4. *(Himmelblau#5j*

$$\text{Min} \quad 1000 - x_1^* - 2x_2^{\prime 2} + x_3^{\;2} - X_j X_2 - x_1 x_3$$

$$s.L \quad x_1^2 + x_2^2 + x_3^2 - 25 = 0$$

$$8x_x + 14x_2 + 2x_{tr} 56 = 0$$

Inconsistent starting points

(a) $xl^{0)} = 2, i=l,2,3$

(b) $xl^{0)} = 10, i=1,2,3$

Example 5. *(Biegler and CuthreW*

$$\text{Min} \quad x_2$$

$$s.t. \quad 1 + x_1 - x_2^2 + x_3 = 0$$

$$\cdot \quad ^1 \quad \cdot_2^{\;?} \quad \cdots_4 = 0$$

$$x_2 \geq \tfrac{1}{2}$$

$$x_3. \; x_4 \; Z \; 0$$

Inconsistent starting point

$$xf^{0)} = 0, 1=1.2,3,4$$

Table 2 compares the number of iterations required for convergence using the preprocessor to ensure a consistent Initial point, or relaxing the feasible region at the level of the QP (as in P6) without first using a preprocessor. For all three cases, the preprocessor found initial consistent points and no further infeasibilities were encountered at later iterations. For relaxing the feasible region at the level of the QP, both QPKWIK and QPSOL were considered. Handling infeasibilities within QPKWIK was discussed in Section 3.6. For QPSOL, the infeasibility tolerance, *FEATOL,* was increased when an infeasible QP was encountered and the relaxed QP was then resolved.

The results in Table 2, as well as results for another set of problems which will be presented in Section 5 show that, in general, using the preprocessing step to prevent an initial inconsistent point results in the fewest number of iterations. This may not always be true because minimizing the infeasibilities may move the algorithm considerably further away from the optimum than the initial inconsistent point was. However, a preprocessing step is undoubtedly the most robust approach for handling constraint inconsistencies and should be used to guarantee an initial feasible QP subproblem. On the other hand, relaxing the problem at the level of the QP provides an inexpensive means to handle infeasibilities encountered at later iterations. With

QPKWIK, the algorithm only returns to the preprocessing phase if an infeasible QP causes the search direction to become zero.

Problem	QPSOL	QPKWIK	Preprocessor + QPKWIK
1(a)	10	11	11
1(b)	14	13	11
2	18	6	6

Table 2. Number of iterations for convergence for Examples 4 and 5.

As mentioned above, the preprocessing phase also serves to select a nonsingular set of basis variables. As the systems become larger, the identification of such a set by hand is increasingly difficult if not impossible, and an automatic procedure for making the assignment becomes invaluable. For all the problems considered, Fourer's piecewise-linear programming approach applied to (P9) was successful in selecting a suitable set of dependent variables. In addition, tests were performed on problems that included linearly dependent constraints and, in all cases, the redundant constraints were correctly identified. One issue which does arise, though, is how to incorporate user knowledge in the selection of the basis. Consider, for example, a system of equality constraints which Incorporates a square block for which the assignment of appropriate dependent variables is straightforward. Clearly, this information would contribute to a reduction in the effort associated with the solution of (P9). With this in mind, we have included an additional step in the preprocessing phase which allows for the specification of some or all of the dependent variables. Fourer's code then serves to verify the selection and, if necessary, to complete it. The piecewise-linear programming algorithm requires an initial basis and an initial feasible point to be supplied. By default, the slack variables provide such a basis and a feasible point is obtained by equating these variables to the right hand side and all others to zero. If a subset of the dependent variables are prespecified, the system of equations is partially solved to yield a feasible point consistent with the modified basis. The potential benefits of such a partial preselection are illustrated using a problem taken from Lalee (1992).

The results in Table 3 give the CPU seconds on a SUN3 for the preprocessing phase as a function of the number of preassigned basic variables. In all cases, variables 3 to 300 constitute the final set.

Example 6. (Lalee)

$$Min \quad \sum_{t=1}^{299} (x_t + x_{t+1})^2$$

$$s.t \quad X_j + 2X_j^\wedge + 3x_{J+2} - 1 = 0 \qquad J = 1, \dots 298$$

$$x_P = -4 \quad x_f^w = 1, OI$$

Number of preassigned basic variables	0	100	200	298
CPU **time** (s)	75.1	49.2	24.6	1.6

Table 3. CPU seconds on a SUN3 as a function of the number of preassigned basic variables for Example 6.

The above results indicate that even a partial identification of the dependent variables yields considerable savings in the time required to solve (P9). Hence, the user should include any information that may be available.

5. Numerical results for Process Optimization

In this section we present results for three sets of process **optimization problems.** Firstly, we consider a series of distillation column examples. **These** serve **to illustrate** the concept of tailoring the algorithm to take advantage of problem structure, as described in Section 2. In addition, the distillation model is such that the constraint linearizations tend to be inconsistent unless a very good initial point is available. Next, we provide a comparison of our algorithm with MINOS (Murtagh and Saunders, 1982, 1987) for the optimization of the Sunoco Hydrocracker Fractionation Plant as presented in Bailey *et aL* (1992). Finally, we study the effect of optimizing the DIB distillation column which constitutes a subproblem of the fractionation example.

The first set of problems we present constitutes four distillation examples described in more detail in Schmid and Biegler (1993). We consider the separation of a binary benzene-toluene mixture and a ternary benzene-toluene-xylene mixture. Operating conditions of both 12 and 36 trays have been studied for both cases. This gives rise to models with *(nstxnk+2)* variables and *nstxnk* equality constraints where *nst* is the number of trays and nfc is the number of components. The pressure and reflux ratio are selected as the independent variables. The objective function (to be minimized) consists of the weighted utility requirements minus the product rate, and also includes

a penalty term to ensure that the pressure does not deviate excessively from 1 atm. Additional constraints are included to ensure that at least 98% pure benzene is retrieved and less than 2% is lost at the bottom of the column. Lower bounds maintain all variables at positive values.

The first set of results, presented in Table 4 were obtained by initializing the problem using the Naphthali-Sandholm strategy given within the UNIDIST package (Anderson et aL, 1991); the initial point is infeasible but consistent. We use these results to illustrate the effect of tailoring the algorithm to exploit problem structure as mentioned in Section 2. We compare the CPU seconds using the sparse Gauss elimination package MA28 to the tailored method using a Thomas algorithm. Results for dense Gauss elimination are also included.

Problem	Iterations	CPU Time (s)		
		Dense Gauss Elimination	Sparse Gauss Elimination	Tailored Method
2 components 12 trays	16	43.9	38.2	35.2
3 components 12 trays	10	50.2	38.6	34.7
2 components 36 trays	11	184.7	76.7	70.8
3 components 36 trays	12	507.8	131.7	119.0

Table 4. Number of iterations and CPU seconds on a SUN3 for convergence of various distillation problems

Passing from dense to sparse Gauss elimination we observe a reduction of 13% to 74% in CPU seconds. When we replace the sparse solver by a tailored algorithm we gain a further 8% to 10%. This is noteworthy since the tailored approach required very little modification of the existing distillation package. These results illustrate the potential benefit of tailoring the algorithm to take advantage of the special mathematical structure of various classes of problems.

The four distillation examples were also initialized at a point further from the optimum, which is not only infeasible but also inconsistent. Table 5 has the same format as Table 2 in Section 4. It compares the number of SQP iterations required for convergence

using the preprocessor to generate an initial consistent point as opposed to relaxing the problem at the level of the QP.

Problem	QPSOL	QPKWIK	Preprocessor + QPKWIK
2 components 12 trays	21	21	16
3 components 12 trays	24	18	13
2 components 36 trays	15	17	17
3 components 36 trays	92	28	16

Table B. Number of iterations for convergence for various distillation examples.

The results in Table 5 show that, for this set of example problems, a preprocessor is not **essential for convergence** of the algorithm. However, in **general, using a Phase I** preprocessor does result in the fewest SQP iterations.

We **now** compare **our** reduced Hessian algorithm to MINOS, a nonlinear programming software package developed by Murtagh and Saunders (1982). The **problem we consider** is that of determining the optimal operating conditions of the Sunoco Hydrocracker Fractionation Plant; we use the model presented by Bailey *et aL* (1992). This case study is based on an existing process and is typical of real-time optimization problems. The fractionation plant, shown in Figure 3 is used to separate the effluent stream from a hydrocracking unit. The portion of **the** fractionation plant which **is represented** by **the** model is highlighted in Figure 3; it includes the absorber, stripper, debutanizer, C_3/C_4 splitter and deisobutanizer. Details on the individual units may be found in Bailey *et aL.*

In addition to solving the optimization problem from the given initial point, a two-step procedure was also considered. As in Bailey *et* aL., we first solve a single square parameter case in order to fit the model to an operating point. The optimization is then performed starting from the good initial point thus obtained. In an on-line system, the solution to the parameter case would be readily available, since it constitutes the current operating conditions. Besides the equality constraints used to represent the

individual units, a number of simple bounds are also included in the model. These
bounds fall into three main categories:

(a) Bounds representing actual physical limits (e.g.. nonnegativity constraints on
    temperatures) are included to prevent numerical problems.

(b) For the optimization cases, bounds placed on key variables prevent the solution
    from moving too far from the starting point. These bounds are specified so that
    the control system would be able to take the maximum step and bring the plant
    back to steady-state over a one to three hour span.

(c) Upper and lower bounds which are used to fix certain variables:

    (i)   Variables fixed in both the parameter and optimization cases constitute
          variables that are not part of the optimization but are included for consistency.
          They include the composition and thermal conditions of the feed streams
          entering the absorber/stripper,

    (ii) Certain variables are fixed only for the parameter case. In effect, enough
          variables must be fixed for each piece of equipment such that a square system
          results,

    (ill) Finally, a number of variables are calculated during the parameter case and
          then fixed for the optimization. Typically, these Include heat exchange
          coefficients, heat loss correction factors and catalyst activities.

The problem statistics are summarized in Table 6.

	Parameter Case	Optimization Case
Number of variables	2891	2891
Number of equality constraints	2836	2836
Number of Jacoblan elements	24123	24123
Number of fixed variables: Type (l)	42	42
Type(ii)	13	0
Type (ill)	0	3
Number of independent variables	0	10

Table 6. Problem statistics for the hydrocracker fractionation plant problem.

The objective function which drives the operating conditions of the plant must account
for the costs of energy and provide a measure of the value added to the raw materials

through processing. The form of the objective function used for this study is given by (5.1). Details on each of the four terms may be found in Bailey *et aL* .

$$P = X^{z_* c G}_{i \in G} + X^{z c E}_{i \in E} + \mathbf{1}^{N_P}_{m=1} Z^{z cPm}_{i \in P_m} \cdot u \qquad \wedge$$

where   P   = profit,

        C^G = value of the feed and product streams valued as gasoline,

        $C^\wedge$ = value of the feed and product streams valued as fuel,

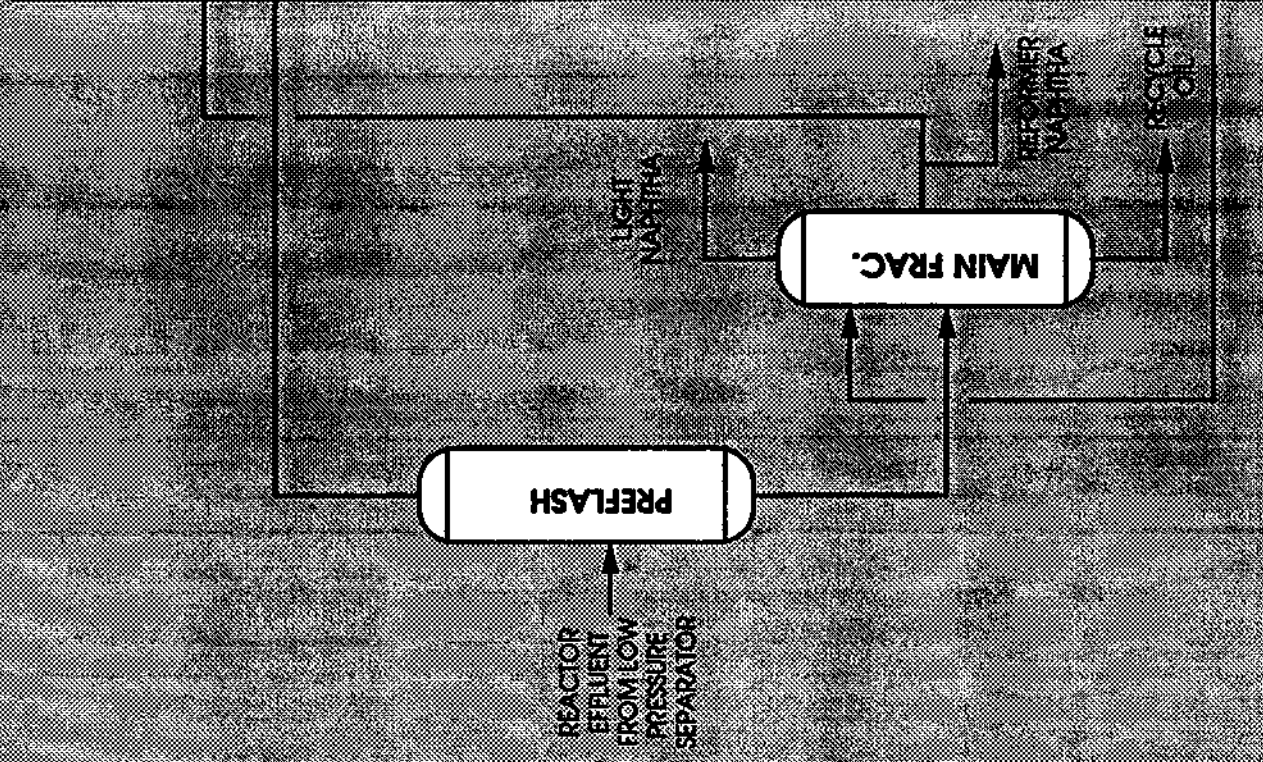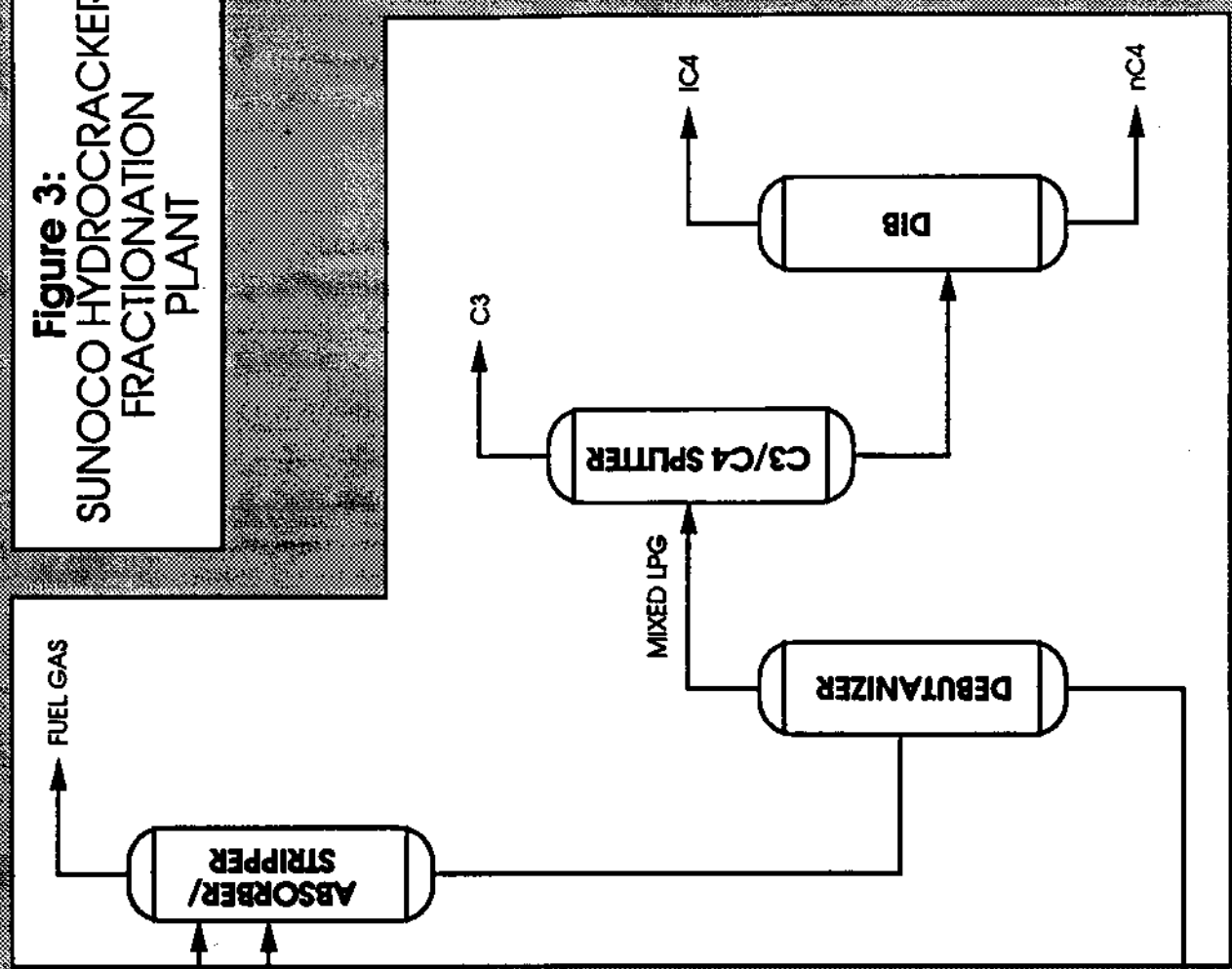   *Cf^m*   = value of pure component feed and products, and

        U   = utility costs.

In addition to the base optimization case (Case 1), the effect of heat exchanger fouling on the optimal solution was considered (Cases 2 and 3) as was the effect of changing market conditions (Cases 4 and 5). The effect of fouling is simulated by reducing the heat exchange coefficients for the debutanizer and splitter feed/bottoms exchangers. Changing market conditions are reflected by an increase in the price for propane (Case 4) or an increase in the base price for gasoline together with an increase in the octane credit (Case 5). The numerical values for the above parameters are included in Table 7.

All cases were solved on a DEC 5000/200 using a convergence tolerance of 10^{-8}. The results are reported in Table 7, where Unfeasible Initialization" indicates initialization at the original initial point while the "Parameter Initialization" results were obtained using the solution to the parameter case as the initial point. Coordinate bases were used and 10 of the 13 variables which are fixed only during the parameter case were used as the independent variables for the optimization cases. We also report the results obtained by Bailey *et aL.* using MINOS. In Table 7 their results are converted to equivalent CPU seconds for a DEC 5000/200. In all cases, both our method and their MINOS cases terminated at the same optimal solution. For an interpretation of the objective function values, see Bailey *et aL* (1992).

From Table 7 it is apparent that, for this problem, our algorithm is at least as robust and considerably more efficient than MINOS. Let us now consider each of the sets of results in turn. Reduced Hessian SQP was 8 times faster than MINOS as far as the parameter case is concerned. This is to be expected since, for a square system, SQP simply reduces to Newton's method for solving sets of nonlinear equations while MINOS is unable to take advantage of the fact that the problem solution is completely

Figure 3:
SUNOCO HYDROCRACKER
FRACTIONATION PLANT

	CaseO	Case 1	Case 2	CaseS	Case 4	Case 5
	Base Parameter	Base Optimization	Fouling 1	Fouling 2	Changing Market 1	Changing Market 2
Heat Exchange Coefficient (TJ/d°C)						
Debutanizer Feed/Bottoms	$6.565 \times 10^{"4}$	$6.565 \times 10^{"4}$	$5.000 \times 10^{"4}$	$2.000 \times 10^{"4}$	$6.565 \times 10^{"4}$	$6.565 \times 10^{"4}$
Splitter Feed/Bottoms	$1.030 \times 10^{'3}$	$1.030 \times 10^{"3}$	$5.000 \times 10^{"4}$	$2.000 \times 10^{"4}$	$1.030 \times 10^{"3}$	$1\text{-}030 \times 10^{"3}$
Pricing						
Propane ($/m^3)	180	180	180	180	300	180
Gasoline Base Price ($/m^3)	300	300	300	300	300	350
Octane Credit ($/(RON m^3))	2.5	2.5	2.5	2.5	2.5	10
Profit	230968.96	239277.37	239267.57	236706.82	258913.28	370053.98
Change from base case ($/d.%)	-	8308.41 (3.6%)	8298.61 (3.6%)	5737.86 (2.5%)	27944.32 (12.1%)	139085.02 (60.2%)
Infeuible Initialization						
MINOS						
Iterations (Major/Minor)	5/275	9/788	-	-	-	-
CPU Time (s)	182	5768	-	-	-	-
SQP						
Iterations	5	20	12	24	17	12
CPU Time (s)	23.3	80.1	54.0	93.9	69.8	54.2
Parameter Inftisiizntion						
MINOS						
Iterations (Major/Minor)	n/a	12 / 132	14 / 120	16 / 156	11 / 166	11/76
CPU Time (s)	n/a	462	408	1022	916	309
SQP						
Iterations	n/a	13	8	18	11	10
CPU Time (s)	n/a	58.8	43.8	74.4	52.5	49.7
Time SQP / Time MINOS (%)	12.8%	12.7%	10.7%	7.3%	5.7%	16.1%

Table 7. Numerical results for the Sunoco Hydrocracker Fractionation Plant problem.

determined by the constraints. Bailey *et oL* (1992) only report one result for an optimization case which was initialized at the original "infeasible initialization". When this MINOS result is compared to the SQP result, there is a difference of almost two orders of magnitude. It seems that, for this problem, SQP is less sensitive to a poor initial point than MINOS. However, as mentioned earlier, the solution to the parameter case would be available in an on-line system, so it is more important to compare the "parameter initialization[11] results. Here, the results in Table 7 indicate an order of magnitude improvement in CPU times when comparing our algorithm to MINOS.

The last set of numerical results deals with optimization of the deisobutanizer (DIB) column, a sub-unit of the Sunoco Hydrocracker Fractionation Plant. A mixture of butane and iso-butane containing small amounts of propane and iso-pentane enters the column. Iso-butane is retrieved at the top of the column while the bottoms product is rich in butane. The model developed by Bailey *et aL* Includes 361 variables and 351 equality constraints, with 2211 nonzero elements in the constraint Jacobian. As for the optimization of the full plant, we solve both the parameter and optimization cases. 10 variables are fixed for the parameters to give a square system. Two of these are freed for the optimization case; these provide a natural choice of independent variables. For the optimization case, bounds on the variables of types (a) and (b)$_f$ as discussed above, are also included. Bounds such as nonnegativity of the exit flowrates are not expected to be active at the solution but are required to prevent numerical difficulties during the course of the optimization. The second class of bounds defines the operating limits of the system; these constraints are frequently active at the solution. Here, upper bounds on the amount of butane and iso-butane in the top and bottom streams respectively are included so as to ensure a minimum purity of the exit streams. In addition, bounds are placed on the reflux ratio as well as the heat duties of the reboiler and condenser. These bounds reflect the range of normal operating conditions of the DIB column and characterize the boundaries of the region for which we are confident of the validity of the model.

We first solve the parameter case (Case 0), and the base optimization case (Case 1). We then resolve the problem with a different objective function. Instead of maximizing profit, we wish to determine the maximum purity of iso-butane in the overhead product (Case 2) or of butane in the bottoms product (Case 3), given the above operating bounds. Finally, we remove the bounds on the reflux ratio and on the heat duties (Case 4). The results are given in Table 8 below. We indicate the profit at the

solution, as well as the mole fraction of iso-butane in the overhead stream and of butane in the bottom stream. In addition, we also report the value of the variables which are constrained by operating bounds and indicate which bounds become active at the solution.

	CaseO	Case 1	Case 2	Case 3	Case 4
Reflux Ratio	9.05	8.00 (LB)	10.00 (UB)	9.56	5.40
Top Heat Duty	0.716	0.670	0.790	0.79 (UB)	0.444
Bottom Heat Duty	0.734	0.692	0.808	0.812	0.465
Overhead Butane	0.019	0.05 (UB)	0.016	0.039	0.05 (UB)
Bottom lso-Butane	0.053	0.012	0.05 (UB)	0.012	0.05 (LB)
Overhead lso-Butane	0.924	0.896	0.927	0.906	0.895
Bottom Butane	0.820	0.844	0.823	0.849	0.812
Profit ($/d)	443.336	663.373	264.476	350.725	1167.011
Change In profit from base case ($/d. %)	n/a	220.037 (49.63%)	-178.860 (-40.34%)	-92.611 (-20.89%)	723.675 (163.23%)
SQP Iterations	5	8	5	12	9
CPU seconds on a DEC 5000/200	3.3	4.1	3.6	4.7	4.3

Table 8. Numerical results for the DIB column.

Comparing the results for Case 0 and Case 1_f we observe an increase of almost 50% in the profit as a result of optimizing the operating conditions of the DIB column. The solution is constrained by the lower bound on the reflux ratio and the upper bound on the mole fraction of butane in the overhead stream. The results for Case 2 and Case 3 give us an upper bound on the maximum purity we can achieve, given the operating bounds on the variables. The maximum mole fraction of iso-butane in the overhead stream is 0.927, as opposed to 0.896 for Case 1. The maximum purity of butane is 0.849, only slightly higher than 0.844, the mole fraction obtained for Case 1. In both cases, the profit is reduced. However, we have not accounted for the possibility that the increase in purity may increase the sales price of the product. The point here was simply to determine the achievable limits on purity. Finally, the results for Case 4 indicate that by removing the bounds on the reflux ratio and the heat duties we are able to almost double the profit as compared to Case 1. This suggests that it may be worthwhile to investigate the physical effect of relaxing these bounds. In particular.

the validity of the model at these new operating conditions must be verified and certain parameters may have to be readjusted.

6. Conclusions

In this study we consider key steps of reduced Hessian SQP for the implementation of a robust and efficient algorithm for large-scale process optimization. The main focus is on the solution of the quadratic programming subproblem which arises at each SQP iteration. The algorithm we present, QPKWIK, allows for direct updating of the inverse Cholesky factor of the reduced Hessian matrix and is based on a dual algorithm which does not require an initial feasible point to be determined for the QP. Thus, the method is superior to standard primal methods, even when the QP subproblem is small. Various other features are also included within QPKWIK which allow the reduced Hessian method to perform better as the number of degrees of freedom of the problem becomes larger. In particular, we take advantage of the doubly-bounded nature of the QP constraints, account for the fact that these constraints may be sparse and also include a warm start option. A numerical comparison of QPKWIK and QPSOL, a commercial algorithm commonly used to solve QP subproblems is very encouraging. In addition, QPKWIK is able to handle infeasible QPs which may occur during the course of the algorithm. Unlike QPSOL, QPKWIK only allows the equality constraints but never the bounds to be violated as this could cause the SQP algorithm to experience numerical difficulties.

The second issue we address is that of a preprocessing phase which generates an initial feasible point as well as a nonsingular set of basis variables. Here we incorporate Fourer's piecewise-linear simplex algorithm, allowing for more efficient solution of the linear feasibility problem than is possible with standard simplex methods.

The numerical results in Section 5 indicate that our reduced Hessian SQP algorithm is able to solve problems which are both large and present the numerical characteristics typical of process optimization models. By considering different initial points, we demonstrate the ability of our algorithm to determine the optimal solution, even when it is initialized relatively far from the solution at a point where the constraint linearizations may be inconsistent. Moreover, the comparison with MINOS for the optimization of the Sunoco Hydrocracker Fractionation Plant is very encouraging. The results indicate that our reduced Hessian algorithm is at least as robust and an order of magnitude faster than MINOS for this set of problems. Finally, results for a series of

distillation examples illustrate the benefit of tailoring the solution algorithm to take advantage of the mathematical structure of the process model.

Acknowledgements

References

Bailey, J. K., A. N. Hrymak, S. S. Treiber and R B. Hawkins, "Nonlinear Optimization of a Hydrocracker Fractionation Plant/[1] to appear, *Comput chem. Engng,* (1993).

Berna, T., M. H. Locke and A. W. Westerberg, "A New Approach to Optimization of Chemical Processes," *AIChE J..* 26, p. 37 (1980).

Biegler, L. T. and J. E. Cuthrell, "Improved infeasible path optimization for sequential modular simulators - II. The optimization algorithm," *Comput chem. Engng* 9, 257 (1985).

Biegler, L. T., J. Nocedal and C. Schmid, "Reduced Hessian Strategies for Large-Scale Nonlinear Programming," Working paper (1993).

Dennis, J. E. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice Hall, New Jersey (1983).

Fourer, R, A simplex algorithm for piecewise-linear programming I: Derivation and proof. *Math. Prog.* 33, 204 (1985).

Fourer, R, A simplex algorithm for piecewise-linear programming II: Finiteness, feasibility and degeneracy. *Math. Prog.* 33, 204 (1985).

Fourer, R, A simplex algorithm for piecewise-linear programming III: Computational analysis and applications. Technical report 86-03, Northwestern University (1989).

Gabay, D., "Reduced Quasi-Newton Methods with Feasibility Improvement for Nonlinearly Constrained Optimization," *Math Programming Study,* 16, p. 18 (1982).

Gill, P. E. and W. Murray, Numerically stable methods for QP. *Math. Prog.* 14, 349 (1978).

Gill, P. E., W. Murray, M. A. Saunders and M. H. Wright, *User's Guide for SOL/QPSOL: A Fortran Package for Quadratic Programming.* Technical Report SOL 83-7 (1983).

Goldfarb, D. and A. Idnani, "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs," *Mathematical Programming,* 27 (1983) 1-33.

Himmelblau, D. M., *Applied Nonlinear Programming.* McGraw-Hill, New York (1972).

Lalee, M., "Algorithms for Nonlinear Optimization/[1] PhD Dissertation, Northwestern University, 1992.

Locke, M. H. , R Edahl and A. W. Westerberg, "An Improved Successive Quadratic Programming Optimization Algorithm for Engineering Design Problems," *AIChEJ.,* 29, 5, (1983).

Marsten, R E. The design of the XMP linear programming library. ACM Transactions on Mathematical Software 7, 481 (1981).

Murtagh, B. A. and M. A. Saunders, "A Projected Lagrangian Algorithm and Its Implementation for Sparse Nonlinear Constraints[11], *Math Prog Study,* 16, 84-117, 1982.

Murtagh, B. A. and M. A. Saunders, "MINOS 5.1 User's Guide", Technical Report SOL 83-2OR, Stanford University, 1987.

Powell, M. J. D., "A Fast Algorithm for Nonlinear Constrained Optimization Calculations," *1977 Dundee Conference on Numerical Analysis,* (1977).

Powell, M. J. D., "ZQPCVX: a Fortran subroutine for convex quadratic programming,[11] Report DAMTP/1983/NA17, Department of Applied Mathematics and Theoretical Physics, University of Cambridge (1983).

Schmid, C. and L. T. Biegler, "Acceleration of Reduced Hessian Methods for Large-Scale Nonlinear Programming," to appear. *Computers and Chemical Engineering* (1993).

Vasantharajan, S. and L. T. Biegler, "Large-Scale Decomposition for Successive Quadratic Programming," *Computers and Chemical Engineering,* 12, p. 1089·(1988).

Vasantharajan, S., J. Viswanathan and L. T. Biegler, [tf]Reduced Successive Quadratic Programming implementation for large-scale optimization problems with smaller degrees of freedom," *Comput chem. Engng* 14,907(1990).