# NUMERICAL TECHNIQUES FOR PROCESS OPTIMIZATION BY SUCCESSIVE QUADRATIC PROGRAMMING

by

M. A. STADTHERR and H.-S. CHEN
Chemical Engineering Department
University of Illinois
Urbana, Illinois 61801
USA

Recent work in process optimization has concentrated on the application of the Han-Powell algorithm for successive quadratic programming (SQP). Recently Jirapongphan (1980) and Biegler and Hughes (1981,1982) have demonstrated the effectiveness of this method on some problems in connection with the simultaneous-modular approach. Techniques for applying the Han-Powell method in connection with the equation-based approach have been described by Berna et al. (1980). In this paper we consider enhancements to the basic Han-Powell method to make it more efficient and reliable on process optimization problems. Some numerical results are presented for a new optimization code SQPHP that incorporates these enhancements. For solving process optimization problems SQPHP has been incorporated into SIMMOD, a new flowsheeting and optimization package based on the simultaneous-modular approach. We now proceed to summarize the enhancements considered. This work will be presented in more detail elsewhere.

## Solving the QP subproblems

The starting point for the development of the enhanced code SQPHP was Powell's code VF02AD, which is available as part of the Harwell Subroutine Library. VF02AD calls another Harwell Library routine VE02AD to solve the quadratic programming (QP) subproblems imbedded in the SQP algorithm. VE02AD requires $(n+m)^2 + O(n+m)$ working storages, where n is the number of variables and m the number of constraints. This is not storage efficient when the number of equality constraints is about equal to the number of variables, as is typically the case in process

optimization problems. Thus we use the equality constraints to eliminate some of the variables. This reduced QP problem is then solved using the method of Gill and Murray.

Handling bounds on variables

In the Han-Powell method, bounds on variables can be handled very efficiently by including them in the QP subproblem. However, in VF02AD bounds on variables are treated as general inequality constraints. This is inefficient both in terms of storage and CPU time. As an indication of the potential saving of CPU time, we ran the test problem used in Berna et al. (1980). This problem has 10 variables, 3 equality constraints, 8 general inequality constraints, and 20 bounds on variables. When bounds are treated as general inequality constraints, our program takes about 1.1 CPU seconds to solve it. When bounds are handled directly, our program requires only about 0.4 CPU seconds.

The stepsize procedure

Maratos (1978) has shown that there exist problems for which the stepsize procedure in the Han-Powell method causes less than the full step to be taken no matter how close the current iterate is to the solution point. In this case the superlinear rate of convergence of the Han-Powell method cannot be obtained. Several solutions to this problem have been proposed: 1. Include a projection step or second-order correction step to reduce the degree of constraint violations (Mayne, 1980; Fletcher, 1982); 2. Replace the nondifferentiable line search function used by a differentiable augmented Lagrangian function (Schittkowski, 1981; Yamashita, 1982); 3. Add a second line search function to ensure that a full step will eventually be used (Chamberlain et al., 1982). In our program we adopt the last approach and use the basic "watchdog" technique of Chamberlain et al. (1982). This approach was chosen because it is easily implemented in connection with VF02AD, and because the results of Schittkowski (1981) indicate that the usage of the differentiable augmented Lagrangian function for the line search does not improve the overall efficiency, and in fact slightly reduces the reliability of the Han-Powell method.

Scaling

One advantage of the Han-Powell method is that constraints need not be scaled. However, the scaling of the objective function and the variables will affect the performance of the method. In SQPHP the user is given the option of providing scaling factors for the variables, and of choosing one of three methods for scaling the objective. To study the performance of the Han-Powell method on a very badly scaled problem, we use the electrolytic cell problem given by Stadtherr et al. (1983). This problem has 40 variables, 37 equality constraints, 2 general inequality constraints,

and 51 bounds on variables. The variables range in size from $10^{-6}$ to $10^{5}$, so this is a very badly scaled problem. In comparing runs with and without scaling, it was found that with scaling the solution was found in 10 iterations, while without scaling 45 iterations were required. Another more severe problem is that the run without scaling would have terminated prematurely far from the solution if a slightly looser convergence tolerance had been used. For process optimization problems, the problem of premature termination can be severe because evaluation of the functions typically requires the solution of systems of nonlinear equations. Because it may not be practical to always solve these systems to very high accuracy, a loose convergence tolerance has to be used.

Imposing a step bound

In applying the Han-Powell method to test problems, we found that the method occasionally failed because in the first few iterations the QP subproblems predicted very large correction steps, and these large steps were not reduced by the stepsize procedure because the penalty parameters were too small. For these problems we find it is advantageous to impose a step bound restricting the stepsize during the first few iterations. SQPHP includes an option to impose such a step bound for the first three iterations.

Evaluation of functions and derivatives

VF02AD requires the evaluation of functions and derivatives at the same time, leading to some unnecessary derivative evaluations whenever a full step length is not chosen. For this reason we separate the evaluation of functions and derivatives in SQPHP.

Convergence test

VF02AD uses an absolute convergence test. In SQPHP we use a relative convergence test. This is slightly more convenient because the same value of the convergence parameter can be used for different problems, and also because it makes the convergence test independent of the scaling of the objective function as long as the absolute value of the objective is greater than one. We now proceed to describe some of the numerical tests used to study the performance of SQPHP.

Numerical tests

We use problems 1-14 from the collection of Cornwell et al. (1978) to test the numerical performance of SQPHP. These are standard test problems originating either with Himmelblau or Colville. Minkoff (1981) recently used these problems in testing VMCON, another well-known code for implementing the Han-Powell method, and GRG2, an efficient implementation of the generalized reduced gradient (GRG) method. In Table 1 below we compare SQPHP to these two codes. For purposes of comparison, the optional procedures

in SQPHP for scaling, for imposing a step bound, and for using the basic watchdog technique are disabled.

TABLE 1. COMPARISON OF SQPHP, VMCON AND GRG2 ON PROBLEMS 1-14 FROM CORNWELL ET AL.

| | Function Evaluations | Derivative Evaluations | CPU Time (Seconds) |
|---|---|---|---|
| SQPHP | 270 | 217 | 3.25 |
| VMCON | 258 | 258 | 11.33 |
| GRG2 | 4022 | 382 | 5.63 |

Several comments on the above results are in order:

1. The results for VMCON and GRG2 are taken from Minkoff (1981), are for an IBM 370/195 computer, and are based on using analytical derivatives. SQPHP was run on a CDC Cyber 175 and difference approximation derivatives were used. Both SQPHP and GRG2 returned a good solution for all of the problems, although SQPHP terminated abnormally twice, and GRG2 five times. VMCON failed to solve one problem, and in one case terminated abnormally but with a good solution.

2. From the timing data given in the LINPACK Users' Guide, the IBM 370/195 is about 70% faster than the CDC Cyber 175. To be conservative we ignore this speed difference in the discussion here.

3. SQPHP requires about the same number of function evaluations as VMCON. This is expected because both implement the Han-Powell method. However, SQPHP is considerably faster than VMCON. We attribute this largely to the fact that SQPHP handles bounds directly and uses a more efficient QP routine. Along these lines it should be noted that Lasdon (1982) has recently described a modification of VMCON that "warm-starts" the QP problems, providing a reduction in CPU time of more than 50% on a set of the Himmelblau test problems. SQPHP "cold-starts" the QP problems; incorporation of the warm-start procedure would further improve the efficiency of SQPHP, at least on problems for which a large fraction of the CPU time is spent on the QP subproblems. While the test problems used here fall into this category, our experience in using SQPHP with SIMMOD is that for actual process optimization problems, a relatively small fraction of the CPU time is spent on the QP subproblems, most of the time being spent on function and derivative evaluations. Thus the use of the warm-start procedure is likely to provide only marginal savings on actual process optimization problems.

4. Both VMCON and SQPHP require considerably fewer function and derivative evaluations than GRG2. VMCON is

considerably slower than GRG2 however, while SQPHP is faster. These results clearly demonstrate the importance of a good implementation in reducing the overhead associated with the Han-Powell method.

Effect of using the basic watchdog technique

The results presented in Table 1 are for SQPHP without use of the basic watchdog technique. We also ran SQPHP on the same 14 test problems using the watchdog technique, and found that 251 function evaluations, 218 derivative evaluations, and 3.23 seconds of CPU time were required. Clearly for this set of test problems the use of the watchdog technique has very little effect on the performance of the Han-Powell method. However, we have observed a beneficial effect on some actual process optimization problems solved with SIMMOD, and since the overhead involved in using the watchdog technique is very small, we recommend that this option be used.

Effect of imposing a step bound

We use here problems 15-21 from Cornwell et al. (1978), for which Rosen is the original source. SQPHP was run using a step bound parameter of $10^6$ (no step bound), 1, and 0.1. With no step bound SQPHP failed to solve all of the problems; with a step bound parameter of 1, it failed to solve two of the problems; and with a step bound parameter of 0.1 all the problems were solved. On problems 1-14, all the problems are solved without using a step bound; we find that actually imposing a step bound anyway has very little effect on numerical efficiency. We conclude that by imposing step bounds during the first few iterations the reliability of the Han-Powell method can be improved without significantly affecting its overall efficiency. More detailed results from this study and the others discussed above will be made available elsewhere.

Application to process optimization

SQPHP has been incorporated into SIMMOD, a new flowsheeting and optimization package that uses the simultaneous-modular approach. Details regarding the theory and implementation of SIMMOD, as well as detailed numerical results for several flowsheeting and optimization problems will be given elsewhere. There are several possible strategies for formulating a nonlinear programming (NLP) problem using the simultaneous modular approach. These differ most significantly in whether to handle the stream connection equations inside or outside the NLP routine. For instance, Jirapongphan (1980) passes all stream connection equations to the NLP routine. Thus he has a very large NLP problem to solve, but each function evaluation requires only one pass through each module. On the other hand, the Q/LAP, CFV, and RFV methods of Biegler and Hughes (1981,1982) in effect eliminate all the connection equations outside the NLP

routine. Thus, they solve a very small NLP problem, but each function evaluation now requires a complete process simulation. In SIMMOD we use a new method that attempts to combine the good features of these two approaches, namely a small NLP problem and function evaluations requiring only one pass through each module. The strategy used is similar in principle to that used in Biegler and Hughes' IPOSEQ method; however they find this method to be less effective than RFV or CFV. Our new method, used in connection with SQPHP to solve the nonlinear programming problems, appears more promising. For instance, using SIMMOD we have solved the ammonia synthesis optimization problem in 6 CPU seconds on a CDC Cyber 175. Other studies report times larger by more than an order of magnitude. For instance Jirapongphan reports a solution time of 311 seconds on an IBM 370/168. While the various solution times reported in the literature are not directly comparable because of the different computers used, and the different set of modules used in each case, these considerations cannot account for the order of magnitude improvement found using SIMMOD.

## Concluding Remarks

Successive quadratic programming using the Han-Powell method represents a very powerful technique for process optimization. The new code SQPHP represents an enhancement of the Han-Powell method to make it more efficient and reliable, especially in connection with process optimization problems.

## References


Berna, T. J., M. H. Locke, and A. W. Westerberg, AIChE J., 26, 37 (1980).

Biegler, L. T. and R. R. Hughes, Chem. Eng. Prog., 77(4), 76 (1981); AIChE J., to be published (1982); Comput. Chem. Eng., to be published (1982).

Chamberlain, R. M., C. Lemarechal, H. C. Pederson, and M. J. D. Powell, Math. Prog. Study, 16, 1 (1982).

Cornwell, L. W., P. A. Hutchison, M. Minkoff, and H. K. Schultz, Argonne National Lab., Report TM-320 (1978).

Fletcher, R., in Lecture Notes in Mathematics, Vol. 912 (1981).

Jirapongphan, S., Ph. D. Thesis, M.I.T. (1980).

Lasdon, L. S., presentation at AIChE Annual Meeting, Los Angeles (1982).

Maratos, N., Ph. D. Thesis, Imperial College (1978).

Mayne, D. Q., in Lecture Notes in Mathematics, Vol. 773 (1980).

Minkoff, M., in Nonlinear Programming Symposium 4 (ed. O. L. Mangasarian et al.), Academic Press (1981).

Schittkowski, K., Numer. Math., 8, 883 (1981).

Stadtherr, M. A., G. D. Cera, and R. C. Alkire, Comput. Chem. Eng., to be published (1983).

Yamashita, H., Math. Prog., 23, 75 (1982).