Int. J. Appl. Engng Ed. Vol. 7, No. 4, pp. 264–274, 1991
Printed in Great Britain.

0742–0269/91 $3.00+0.00

A Nonlinear Programming Computer Tool*

A. EL-HAJJ
W. HARMOUSH
K. Y. KABALAN
H. MUDALLAL
Electrical Engineering Department, American University of Beirut, Beirut, Lebanon†

This paper presents a nonlinear programming interactive computer package useful as a practical tool in teaching several related courses. The input to this program is the function to be minimized and the constraints. Once the nonlinear programming method is selected, the user must supply the program with the initial values of its parameters, and the starting point. After parsing the input correctly, the program provides the user with complete information about the solution found. Graphical illustrations are also available for the evaluation of the efficiency of the selected method and the influence of the parameters values on the desired solution.

1. INTRODUCTION

THE nonlinear programming (NLP) problem, which deals with the optimization of a nonlinear objective function subject to linear and/or nonlinear constraints, usually occurs in many engineering problems as well as in several other applications. Analytical techniques for solving such problems are available but when the problem becomes large and highly nonlinear these methods become unsatisfactory and numerical techniques must be used. Several numerical techniques are available to treat these problems.

This paper presents a nonlinear programming interactive computer package that is useful to educate users while solving such problems. The input to this program is the function F(X) to minimize or maximize subject to the equality constraints H(X) = 0 and the inequality constraints $G(X) > 0$. This problem is discussed in detail in [1] and [2]. A parser reads these functions and detects any error following the rules of a grammar. The program contains an option that computes, whenever necessary, the analytical expression of the derivative of a function. This derivative routine may be characterized by its efficiency in finding the derivative of any function regardless of its form, order, and shape. The user can select among many NLP methods (three are now available in the menu), a method to solve his/her problem. For a selected method, the user is asked to provide the necessary parameters as well as the starting point. When the optimization is in process, the program displays different steps. The user can suspend the execution for: For example to watch carefully the situation in the last few steps or to change some parameters in a more convenient way for the following steps. The user can graphically display some information about the optimization process: For example, the function and/or the constraints versus iteration, which enables him/her to compare different methods and to study the efficiency of a given method for a specific problem.

* Paper accepted 22 October 1990.
† Mail address: American University of Beirut, Electrical Engineering Dept., New York Office, 850 Third Avenue, 18th Floor, New York, N.Y. 10022 U.S.A.

2. THE PARSER

The mathematical model of the problem is entered through the editor in the following form:

```
Min:  F(x,y,x, . . .);
CON:  H(x,y,z, . . .)  =  I(x,y,z, . . .);
CON:  G(x,y,z, . . .)  >  K(x,y,z, . . .);
CON:  L(x,y,z, . . .)  <  M(x,y,z, . . .);
```

The equality and inequality constraints can be entered in any order and the program transforms all inequality constraints to the greater than zero type. The editor that contains menus and windows allows efficient correction and variation of the input. Options are also available to save a function and its constraints on a disk file or to load a saved file into memory. The problem is analyzed and recognized by the computer by means of a parser. This parser converts the model from a sequence of characters to a sequence of tokens (tokens include operators, operands, variables, constants . . .). Each token is represented by a fixed length integer code. A token specifier is associated with the tokens of variables and reals types. A context free grammar is defined to specify the form or the syntax of the legal statements of the problem (Appendix A). The parser is used to recognize each statement of the mathematical model as some language construct described by the grammar. In case of a syntax error, one of 12 built error messages is displayed (Appendix A). Figure 1 shows the syntax error that is displayed after compiling the function given in the edit window.

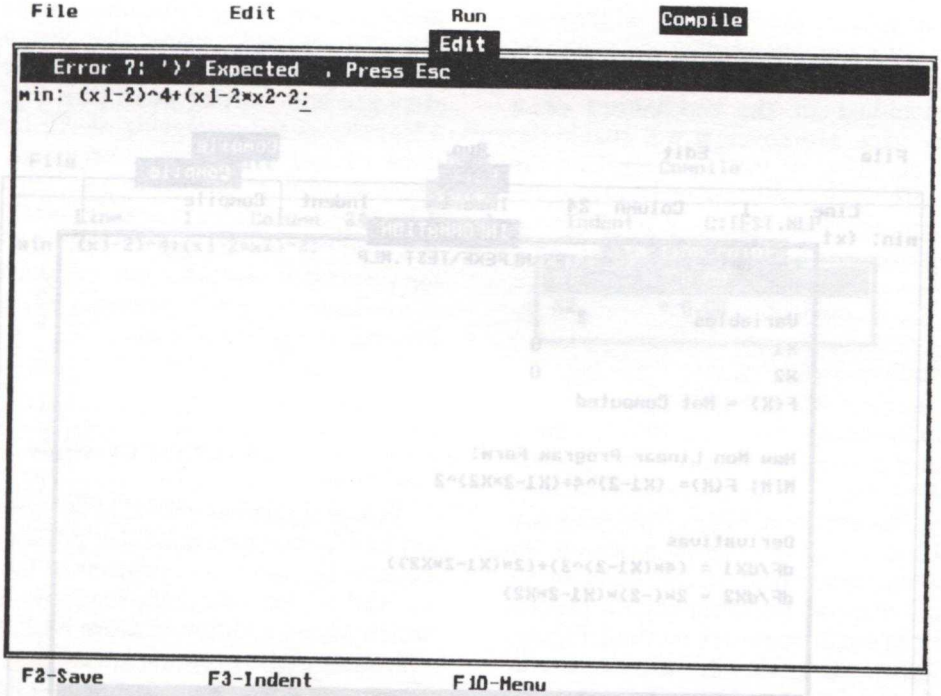


Fig. 1. Compilation of the function $F(X) = (X_1 - 2)^4 + (X_1 - 2X_2^2)$.

In addition, the parser generates an infix notation of the expression to be used later in fetching the symbolic derivative. The function to be evaluated is pushed into a stack in postfix notation then successive elements are popped from this stack and the appropriate operators are then applied. More details about parsing and function evaluation can be found in [3] and [4].

3. THE DERIVATIVE PROCEDURE

The program computes, when necessary, analytical expressions of the derivatives of a function. The derivative procedure reads a function in its infix form and returns its partial derivatives with respect to every variable. This procedure could be applied n times to obtain the n'th partial derivatives of a function. The derivative is calculated recursively following the rules of a grammar as defined in Appendix B. A derivative rule exists for every rule of the context free grammar described in the previous section where a derivative is required. Finally, a special procedure is used to eliminate from the derivative function calculated the useless parentheses, the zero terms, and the zero factors. This derivative option in the package can be used with an introductory mathematics course teaching derivatives. Figure 2 shows the derivative obtained for the function defined in the previous section.

4. THE NLP METHODS

In this package, the constrained problem is transformed into an unconstrained one by adding to the objective function $F(X)$ a penalty term for any violation of the constraints. The problem as defined in Section 1 is then reduced to the following form:

$$\text{minimize } I(X) = F(X) + \text{Penalty}(X)$$

A possible value of the penalty function is:

$$\text{Penalty}(X) = \sum_i q_i \min(0, G_i(X)) + \sum_j q_j H_j^2(X)$$

Where the q_k are penalty constants that may be updated in each iteration.

Many unconstrained NLP methods are available to solve this problem where three of them have been implemented in this package. These methods are:

1 — The Steepest descent method
2 — The Hookes and Jeeves methods
3 — The Rosenbrock method

The program requires for each method a starting point X, the initial values of q_k as well as the initial values of the parameters that characterize each of these methods. The search starts and at each iteration the values of X, the augmented function, and the parameters are displayed. The speed of the display can be changed by the user at his/her convenience. This allows watching carefully the search process and the convergence of the method. This search can be suspended at any time by the user for many reasons: For example to look carefully at the last values displayed or to change some values or parameters that may have influence on the speed of the method or on the local solution found. The user can also display graphically some information. For

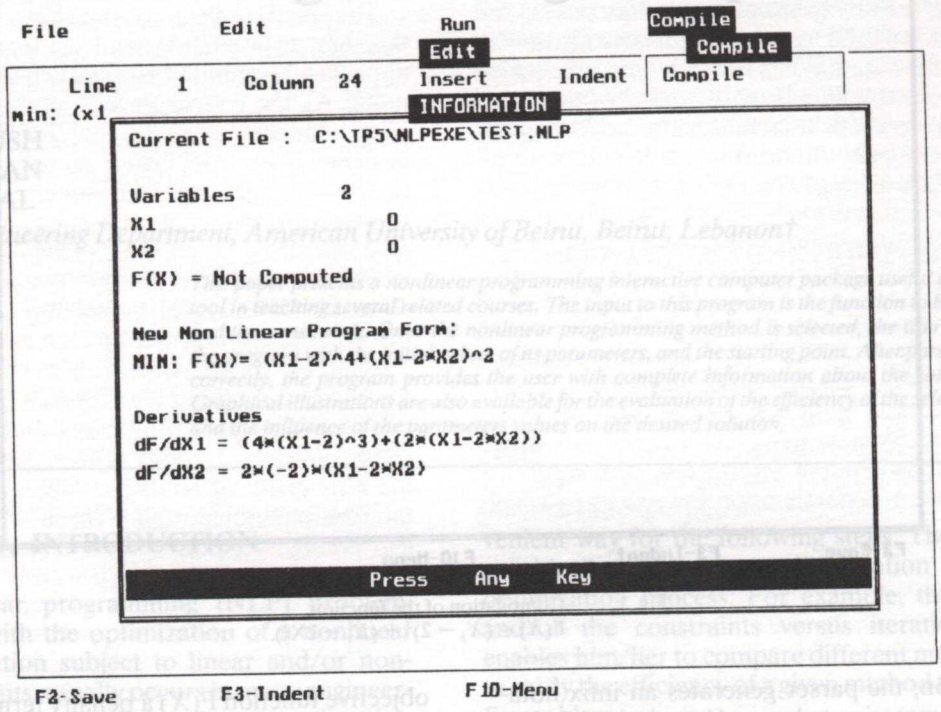


Fig. 2. Information concerning the function $F(X) = (X_1 - 2)^4 + (X_1 - 2X_2^2)$.

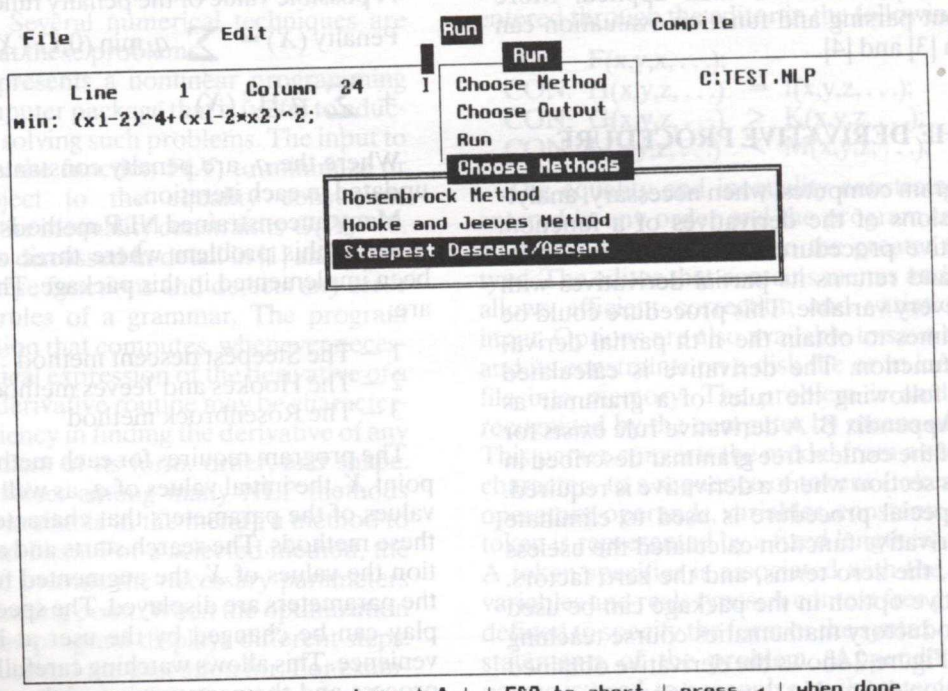
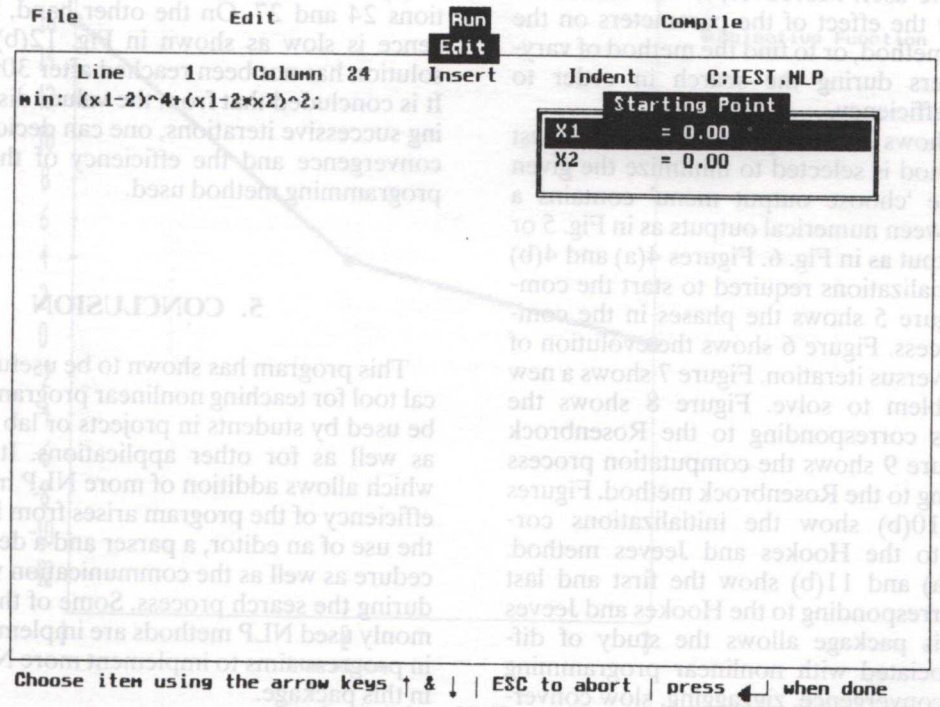


Fig. 3. Selection of a Nonlinear programming method.



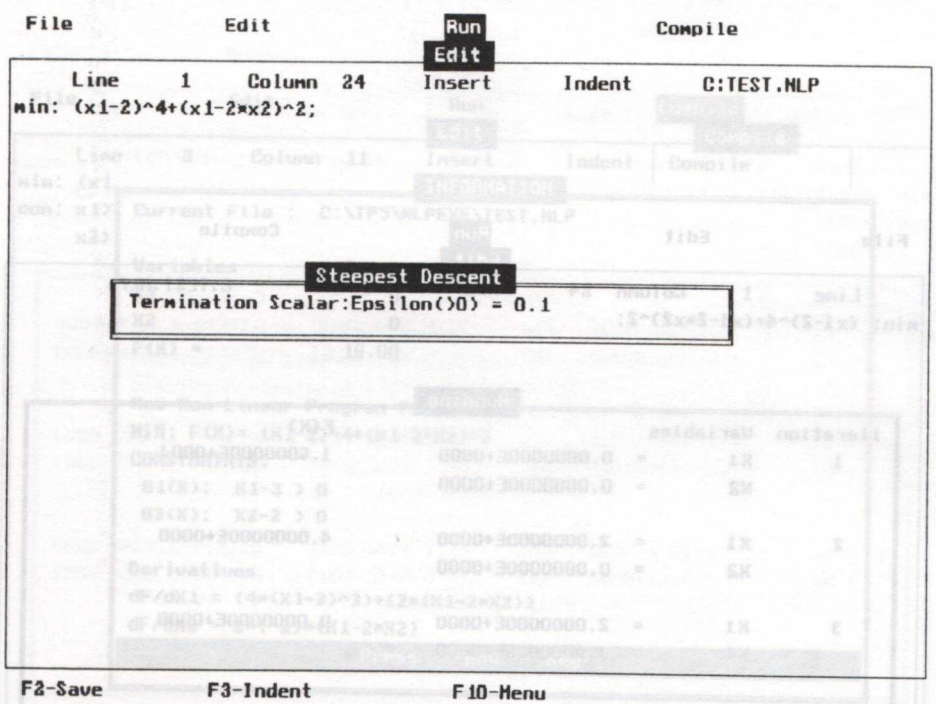


Fig. 4. (a) Starting point in minimizing F(X)
(b) Termination scalar of the solution.

example the variation of the function and the constraints versus iterations. Suspending and resuming the search can be done many times. This allows a better understanding of the mechanism of a method by the user. Moreover, it will enable the user to study the effect of the parameters on the speed of the method, or to find the method of varying parameters during the search in order to improve the efficiency.

Figure 3 shows an example where the Steepest Descent method is selected to minimize the given function. The 'choose output menu' contains a selection between numerical outputs as in Fig. 5 or graphical output as in Fig. 6. Figures 4(a) and 4(b) show the initializations required to start the computation. Figure 5 shows the phases in the computation process. Figure 6 shows the evolution of the function versus iteration. Figure 7 shows a new defined problem to solve. Figure 8 shows the initializations corresponding to the Rosenbrock method. Figure 9 shows the computation process corresponding to the Rosenbrock method. Figures 10(a) and 10(b) show the initializations corresponding to the Hookes and Jeeves method. Figures 11(a) and 11(b) show the first and last iterations corresponding to the Hookes and Jeeves method. This package allows the study of difficulties associated with nonlinear programming such as nonconvergence, zigzagging, slow convergence, . . ., etc. Some difficulties can be found for a previous example using the Steepest Descent method with an inappropriate initial choice of the step size and starting point [See Fig. 12]. Figure 12(a) shows the zigzagging of $X2$ between iterations 24 and 27. On the other hand, the convergence is slow as shown in Fig. 12(b) where the solution has not been reached after 300 iterations. It is concluded that from the values displayed during successive iterations, one can decide about the convergence and the efficiency of the nonlinear programming method used.

5. CONCLUSION

This program has shown to be useful as a practical tool for teaching nonlinear programming. It can be used by students in projects or lab assignments as well as for other applications. It is modular which allows addition of more NLP methods. The efficiency of the program arises from its flexibility, the use of an editor, a parser and a derivative procedure as well as the communication with the user during the search process. Some of the most commonly used NLP methods are implemented. Work in progress aims to implement more NLP methods in this package.

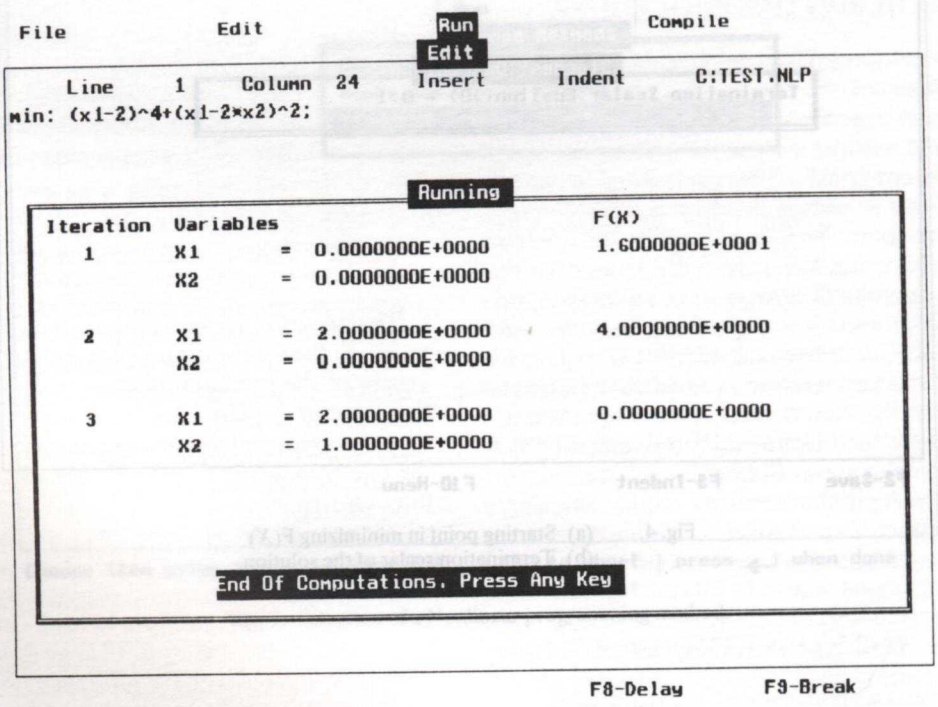


Fig. 5. Minimization process of the function F(X).

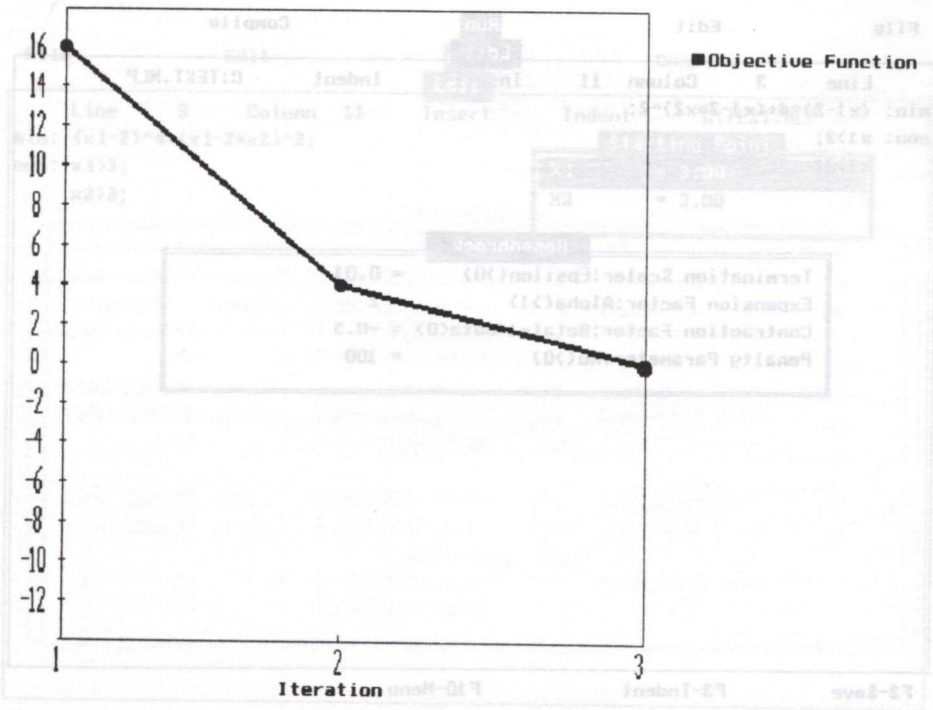


Fig. 6. Variation of F(X) versus iteration.

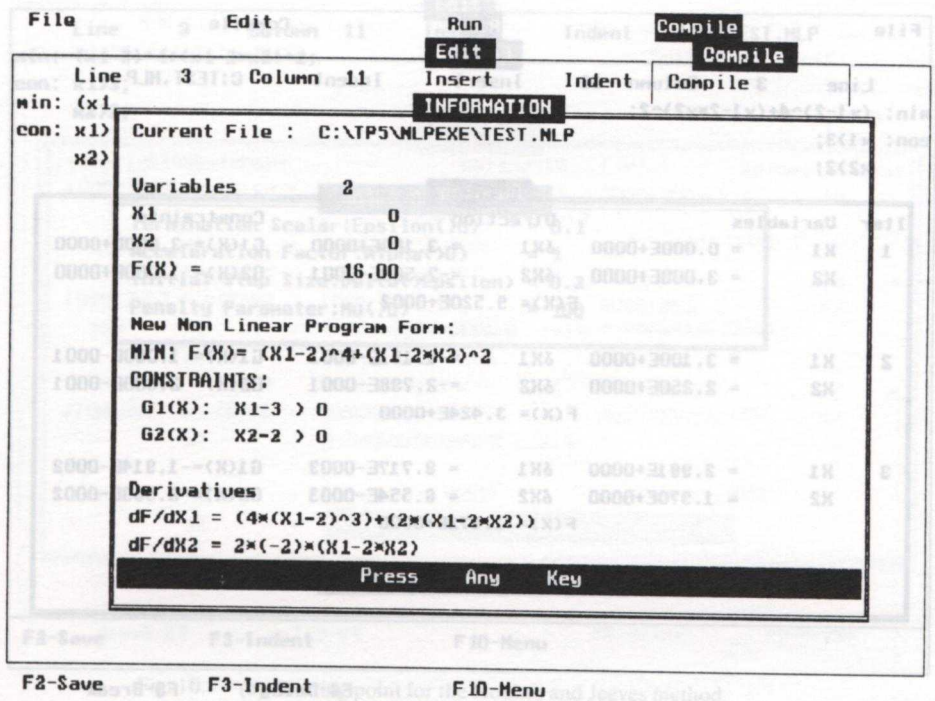


Fig. 7. Minimization of F(X) under constraints.

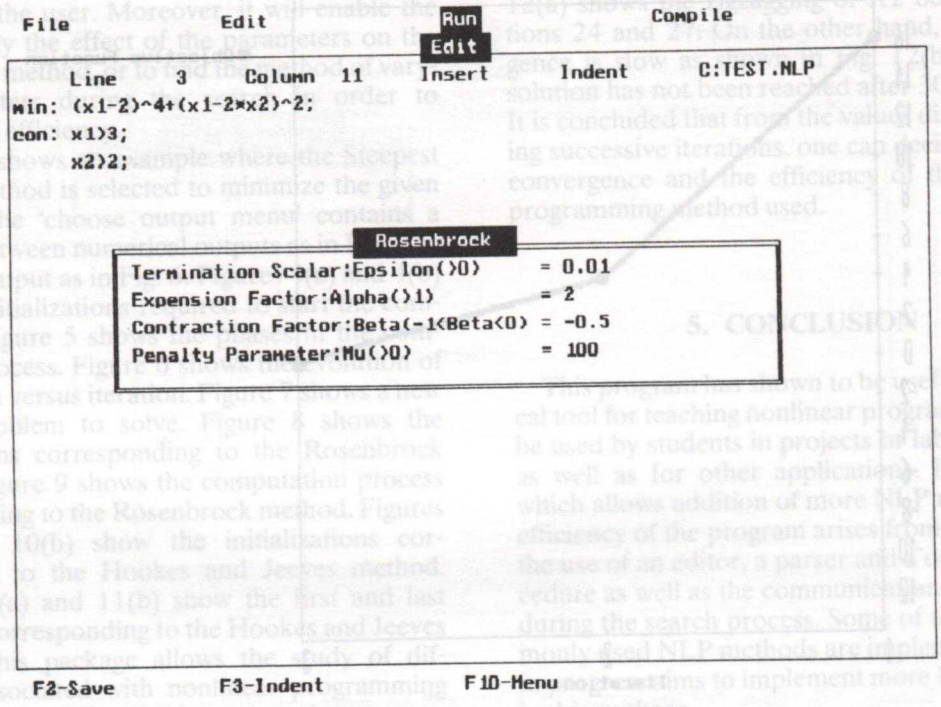


Fig. 8. Values affected to the parameters used in the Rosenbrock method.

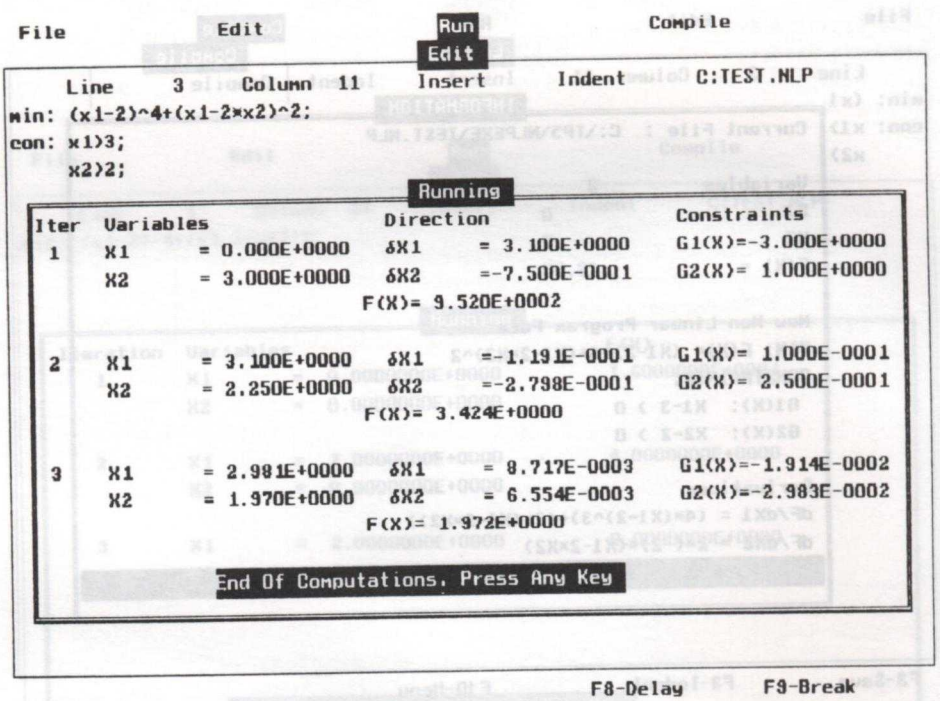
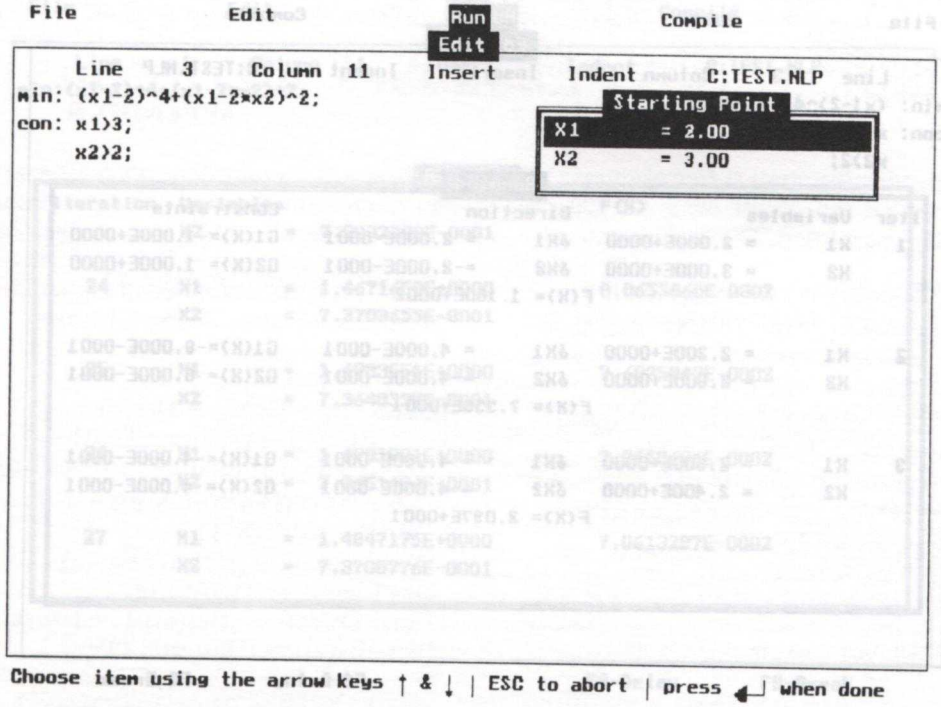


Fig. 9. Minimization of F(X) defined in Fig. 7.



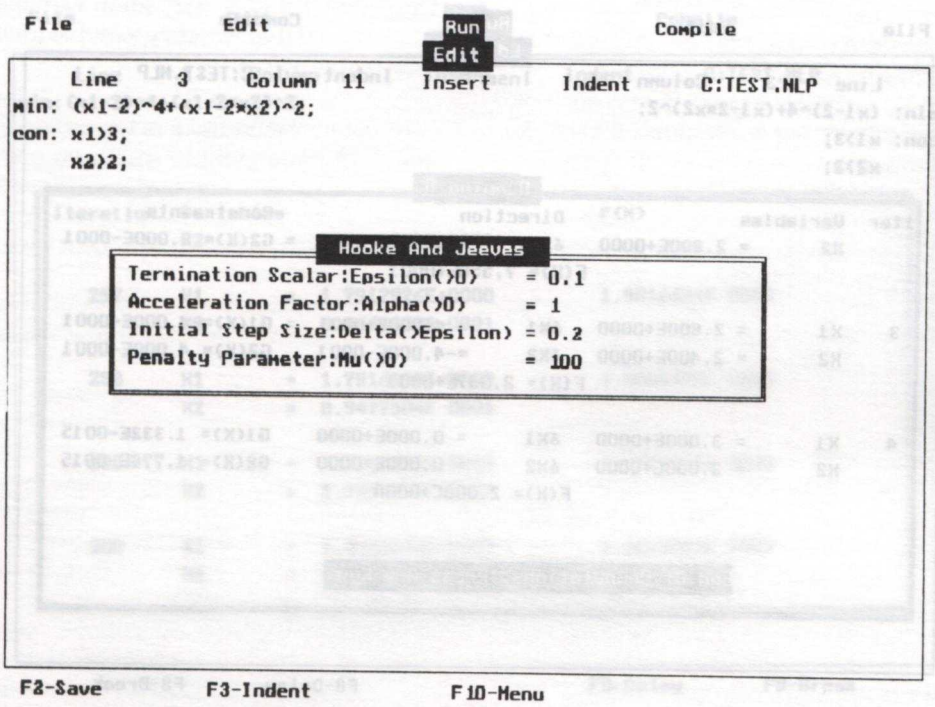
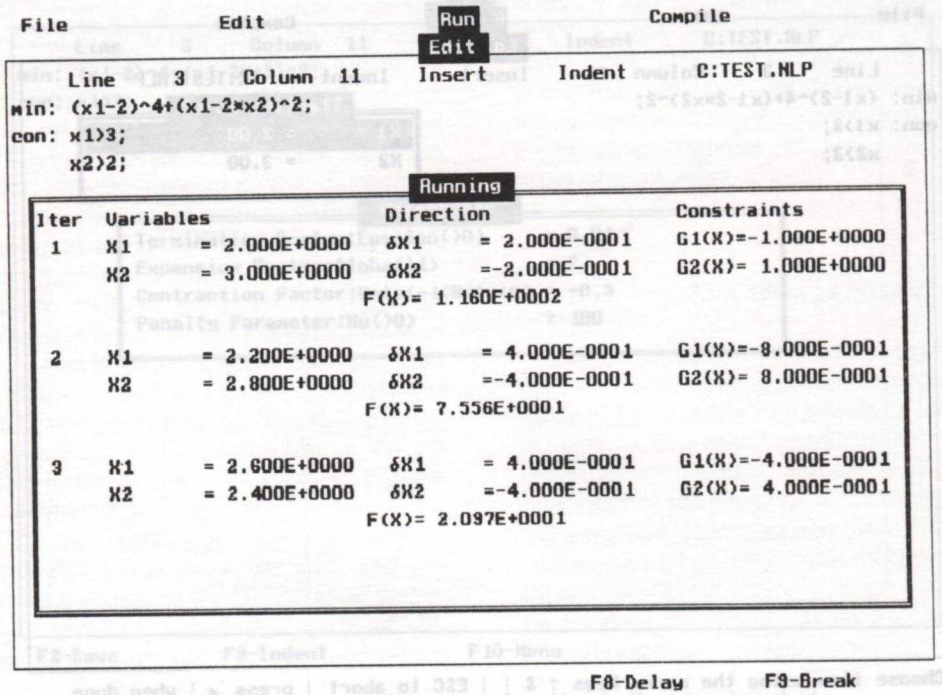


Fig. 10. (a) Starting point for the Hookes and Jeeves method.
(b) Parameters required for the Hookes and Jeeves method.



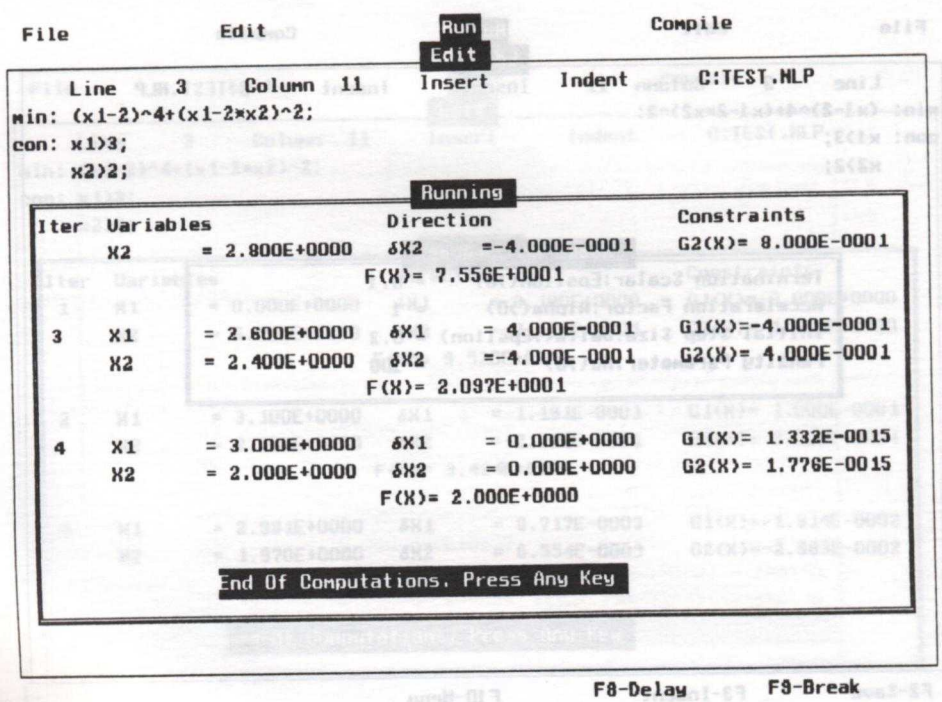
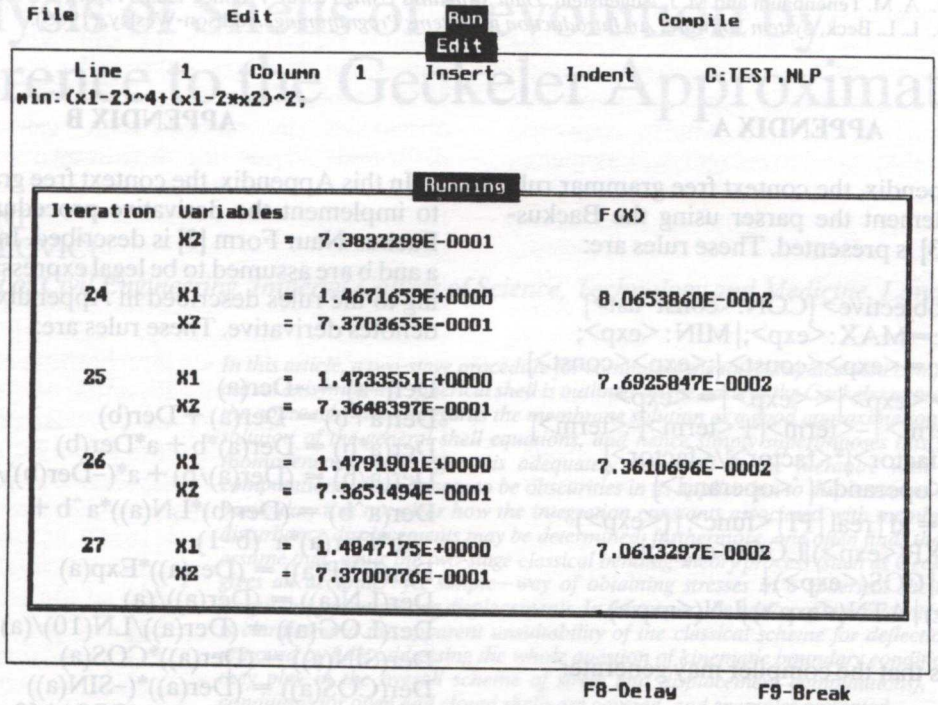


Fig. 11. (a) First iterations in the Hookes and Jeeves method.
(b) Last iterations in the Hookes and Jeeves method.



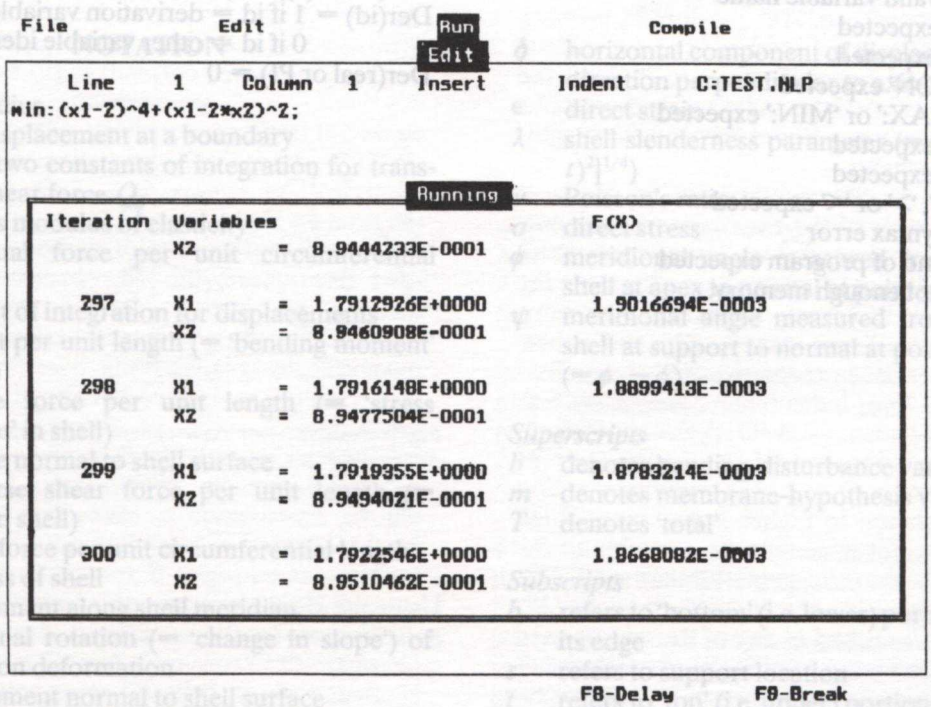


Fig. 12. (a) Zigzagging of $X2$ between iterations 24 and 27.
(b) Slow convergence of the method used.

REFERENCES

1. R. L. Fox, *Optimization Methods for Engineering Design*, Addison-Wesley, (1971).
2. M. S. Bazarraa and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, John Wiley, (1979).
3. A. M. Tenenbaum and M. J. Augenstein, *Data Structures Using Pascal*, Prentice-Hall, (1986).
4. L. L. Beck, *System Software: An Introduction to Systems Programming*, Addison-Wesley, (1985).

APPENDIX A

In this Appendix, the context free grammar rule used to implement the parser using the Backus-Naur Form [3] is presented. These rules are:

<prog>::=<objective> {CON:<const–list>}
<objective>::=MAX:<exp>;| MIN: <exp>;
<const–list>::=<exp><const> {;<exp><const>}
<const>::= > <exp> | < <exp> | = <exp>
<exp>::=<term> | –<term>{+ <term>|–<term>}
<term>::= <factor>{*<factor>|/<factor>}
<factor>::=<operand>{ ^<operand>}
<operand>::= id | real | PI | <func> | (<exp>)
<func>::=EXP(<exp>)|LOG(<exp>)|
SIN(<exp>) | COS(<exp>) |
TAN(<exp>) | ATN(<exp>) |LN(<exp>).

List of errors that the compiler may generate:

Error 1 : Symbol too long
Error 2 : Invalid variable name
Error 3 : ':' expected
Error 4 : ';' expected
Error 5 : 'CON' expected
Error 6 : 'MAX:' or 'MIN:' expected
Error 7 : ')' expected
Error 8 : '(' expected
Error 9 : '=', '>' or '<' expected
Error 10 : syntax error
Error 11 : end of program expected
Error 12 : not enough memory.

APPENDIX B

In this Appendix, the context free grammar used to implement the derivative procedure using the Backus-Naur Form [3] is described. In these rules, a and b are assumed to be legal expressions according to the rules described in Appendix A, and Der denotes derivative. These rules are:

Der(–a) = –Der(a)
Der(a+b) = Der(a) + Der(b)
Der(a*b) = Der(a)*b + a*Der(b)
Der(a/b) = (Der(a)/b) + a*(–Der(b))/b ^ 2
Der(a ^ b) = (Der(b)*LN(a))*a ^ b +
b*Der(a)*a ^ (b–1)
Der(EXP(a)) = (Der(a))*Exp(a)
Der(LN(a)) = (Der(a))/(a)
Der(LOG(a)) = (Der(a))/LN(10)/(a)
Der(SIN(a)) = (Der(a))*COS(a)
Der(COS(a)) = (Der(a))*(–SIN(a))
Der(TAN(a)) = (Der(a))/COS(a) ^ 2
Der(ATN(a)) = (Der(a))/(1+(a) ^ 2)
Der(id) = 1 if id = derivation variable identifier
0 if id = other variable identifier
Der(real or PI) = 0