OSA

MMOS - A MINIMAX OPTIMIZATION SYSTEM IN FORTRAN

OSA-86-MM-2-M

January 24, 1986

Optimization Systems Associates

163 Watson's Lane, Dundas, Ontario, Canada

MMOS - A MINIMAX OPTIMIZATION SYSTEM

IN FORTRAN

OSA-86-MM-2-M

January 24, 1986

I. INTRODUCTION

MMOS is a package of subroutines for solving linearly constrained nonlinear minimax optimization problems. The package has been designed to run on the VAX 11/780 system with the VMS 4.2 operating system and Fortran 4.3 compiler.

The user must supply a subroutine which calculates the values of all the functions as well as the first-order derivatives of all functions with respect to all variables to be optimized.

The solution is found by an iterative scheme which may involve linear programming or a quasi-Newton method employing approximate second-order derivatives. Linear constraints on variables are handled with ease.

Reference

J.W. Bandler, W. Kellermann and K. Madsen, "A superlinearly convergent minimax algorithm for microwave circuit design," IEEE Trans. Microwave Theory Tech., vol. MTT-33, December 1985, pp. 1519-1530.

II. LIST OF ARGUMENTS

1.Entry For Unconstrained Problems

The subroutine call is

```
CALL MMOS1(FDF,N,M,X,DX,EPS,MAXF,W,IW,LCH,IPR)
```

The arguments are as follows.

FDF is the name of the subroutine supplied by the user (see Subsection 3).

N is an integer set to the number of optimization variables.

M is an integer set to the number of functions.

X is a real array of length N (REAL*8 X(N)) which on entry must be set to the starting point of the variables. On exit it contains the solution.

DX is a REAL*8 variable set to the initial step length. The choice of DX is usually not critical, since it is adjusted by the package during the iteration.

EPS is a REAL*8 variable set to the required relative accuracy of the solution.

MAXF is an integer set to a limit on the number of calls to FDF (i.e., maximum number of function evaluations). On exit it contains the number of calls to FDF that have actually been performed by the package.

W is a real array (REAL*8 W(IW)) used for working space.

IW is an integer set to the length of W. Its value must be at least 2*M*N+5*N*N+4*M+8*N+7.

LCH is an integer set to the unit number of the output file. It is usually the unit number of the file SYS$OUTPUT. If it is

less than or equal to zero, no printed output will be generated by the package.

IPR is an integer which controls the printed output. Let J = ABS(IPR)/10. Every Jth function evaluation is reported in the printed output. If the least significant digit of IPR is not zero then the derivatives are also printed. If IPR is negative, the derivatives calculated by the user are verified numerically at the starting point. Any significant discrepancy will be reported.

Examples:

IPR=50. Print the values of the variables and functions for every 5 iterations.

IPR=51. Print the values of the variables, functions and derivatives for every 5 iterations.

IPR=-151. Print the variables, functions and derivatives for every 15 iterations. Verify the derivatives at the starting point.

2. Entry For Constrained Problems

The subroutine call is

```
CALL MMOS2(FDF,N,M,X,DX,EPS,MAXF,W,IW,LCH,IPR,L,LEQ,B,C,LC)
```

The arguments are identical to those defined in Subsection 1 except IW and the last five arguments. The linear constraints are defined as

$$C(I,1)*X(1)+\ldots+C(I,N)*X(N)+B(I) = 0,\ I=1,\ldots,LEQ,$$

$$C(I,1)*X(1)+\ldots+C(I,N)*X(N)+B(I) \geq 0,\ I=LEQ+1,\ldots,L.$$

IW is an integer set to the length of the working space W. Its value must be at least $2*M*N+5*N*N+4*M+8*N+4*LC+3$.

L is an integer set to the total number of linear constraints (including equality and inequality constraints).

LEQ is an integer set to the number of equality constraints.

B is a real array (REAL*8 B(LC)). The elements of B must be set to the constant terms in the linear constraints.

C is a real matrix (REAL*8 C(LC,N)). The elements of C must be set to the coefficients in the linear constraints.

LC is an integer defining the dimensions of B and C. Its value must be not less than L.

3.User's Subroutine FDF

The user is responsible for supplying a subroutine to calculate the function values and derivatives. It must have the form

```
SUBROUTINE FDF(N,M,X,DF,F)
REAL*8 X(N),F(M),DF(M,N)
```

and it must calculate the values of the functions F(J), J=1,...,M and the derivatives DF(J,I) = dF(J)/dX(I). The name FDF is a formal argument. The actual name can be arbitrary and must appear in the EXTERNAL statement in the program calling MMOS1 or MMOS2. The user can terminate the optimization and force the return from the package by setting to zero the variable MARK in the common block MML000 in the subroutine FDF

```
COMMON /MML000/ MARK
```

(on entry MARK is set to 1 by the package).

Example 1

Minimize

$$\max \{f_1, f_2, f_3\}$$

where

$$f_1 = x_1^2 + x_2^2 - 1.0,$$

$$f_2 = 3.0 - x_1^2 + x_2^2,$$

$$f_3 = x_1 - x_2 + 3.0.$$

The starting point is $x_1 = -0.5$ and $x_2 = 0.5$.

```
      PROGRAM TEST1
      PARAMETER (N=2,M=3,IW=2*M*N+5*N*N+4*M+8*N+7)
      REAL*8 X(N),W(IW),DX,EPS
      EXTERNAL FDF
      DATA X/-0.5,0.5/
      OPEN(6,FILE='SYS$OUTPUT',STATUS='UNKNOWN')
      DX=0.2
      EPS=1.D-5
      MAXF=50
      LCH=6
      IPR=10
      CALL MMOS1(FDF,N,M,X,DX,EPS,MAXF,W,IW,LCH,IPR)
      STOP
      END
C
C
C
      SUBROUTINE FDF(N,M,X,DF,F)
      REAL*8 X(N),F(M),DF(M,N),T1,T2
      T1=X(1)*X(1)
      T2=T1+X(2)*X(2)
      F(1)=T2-1.0
      F(2)=3.0-T2
      F(3)=X(1)-X(2)+3.0
      DF(1,1)=2.0*X(1)
      DF(1,2)=2.0*X(2)
      DF(2,1)=-DF(1,1)
      DF(2,2)=-DF(1,2)
      DF(3,1)=1.0
      DF(3,2)=-1.0
      RETURN
      END
```

```
DATE : **-***-****          TIME :  **:**:**
MINIMAX OPTIMIZATION SYSTEM (MMOS PACKAGE. VERSION 86.01)


INPUT DATA
----------

  NUMBER OF VARIABLES (N) . . . . . .    2

  NUMBER OF FUNCTIONS (M) . . . . . .    3

  INITIAL STEP LENGTH (DX) . . . . . . 2.000E-01

  ACCURACY REQUIRED FOR SOLUTION (EPS) . . . . . 1.000E-05

  MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) . . . . . .   50

STARTING POINT
--------------

                        VARIABLES                    FUNCTION VALUES

                  1  -5.000000000000E-01         1  -5.000000000000E-01
                  2   5.000000000000E-01         2   2.500000000000E+00
                                                  3   2.000000000000E+00

FUNCTION EVALUATION :   2

                        VARIABLES                    FUNCTION VALUES

                  1  -6.414213562373E-01         1  -1.771572875254E-01
                  2   6.414213562373E-01         2   2.177157287525E+00
                                                  3   1.717157287525E+00

FUNCTION EVALUATION :   3

                        VARIABLES                    FUNCTION VALUES

                  1  -9.242640687119E-01         1   7.085281374239E-01
                  2   9.242640687119E-01         2   1.291471862576E+00
                                                  3   1.151471862576E+00

FUNCTION EVALUATION :   4

                        VARIABLES                    FUNCTION VALUES

                  1  -1.003102972128E+00         1   1.012431145384E+00
                  2   1.003102972128E+00         2   9.875688546163E-01
                                                  3   9.937940557442E-01
```

FUNCTION EVALUATION : 5

	VARIABLES		FUNCTION VALUES
1	-1.000004799326E+00	1	1.000019197349E+00
2	1.000004799326E+00	2	9.999808026506E-01
		3	9.999904013483E-01

FUNCTION EVALUATION : 6

	VARIABLES		FUNCTION VALUES
1	-1.000000000012E+00	1	1.000000000046E+00
2	1.000000000012E+00	2	9.999999999539E-01
		3	9.999999999770E-01

SOLUTION

	VARIABLES		FUNCTION VALUES
1	-1.000000000012E+00	1	1.000000000046E+00
2	1.000000000012E+00	2	9.999999999539E-01
		3	9.999999999770E-01

NUMBER OF FUNCTION EVALUATIONS 6

EXECUTION TIME (IN SECONDS) 0.21

Example 2

The problem of a 3-section 100-percent relative bandwidth 10:1 transmission-line transformer is considered. The maximum reflection coefficient of this matching network is minimized over the frequency interval [0.5 1.5]. Details can be found in the reference.

Reference

J.W. Bandler, W. Kellermann and K. Madsen, "A superlinearly convergent minimax algorithm for microwave circuit design," IEEE Trans. Microwave Theory Tech., vol. MTT-33, December 1985, pp.1519-1530.

```
      PROGRAM TEST2
C
C       three section transmission-line transformer
C
      PARAMETER (N=6,M=11,IW=2*M*N+5*N*N+4*M+8*N+7)
      REAL*8 X(N),W(IW),FREQ(11),EPS,DX
      EXTERNAL FDF
      COMMON FREQ
      DATA X/0.8,1.5,1.2,3.0,0.8,6.0/
      OPEN(6,FILE='SYS$OUTPUT',STATUS='UNKNOWN')
      FREQ(1)=0.5
      FREQ(2)=0.6
      FREQ(3)=0.7
      FREQ(4)=0.77
      FREQ(5)=0.9
      FREQ(6)=1.0
      FREQ(7)=1.1
      FREQ(8)=1.23
      FREQ(9)=1.3
      FREQ(10)=1.4
      FREQ(11)=1.5
      EPS=1.D-6
      DX=0.25
      MAXF=200
      LCH=6
      IPR=150
      CALL MMOS1(FDF,N,M,X,DX,EPS,MAXF,W,IW,LCH,IPR)
      STOP
      END
C
C
C
      SUBROUTINE FDF(N,M,X,DF,F)
      REAL*8 X(N),F(M),DF(M,N),FREQ(11),G(6)
      COMMON FREQ
      DO 100 I=1,11
      CALL REFLCF(N,X,F(I),G,FREQ(I))
      DO 50 J=1,N
   50 DF(I,J)=G(J)
  100 CONTINUE
      RETURN
      END
C
C
C
```

```
      SUBROUTINE REFLCF(N,X,F,G,FREQN)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 X(N),G(N),AL(3),TH(3),Z(3),GW(3)
      COMPLEX*16 AI(4),V(4),A,B,C,RH,Y,VG
      BETA=0.2095844728*FREQN
      ALFA=7.4948125*BETA
      AI(4)=DCMPLX(1.0,0.0)
      V(4)=DCMPLX(10.0,0.0)
      DO 10 J=1,3
      AL(J)=7.4948125*X(2*J-1)
10    Z(J)=X(2*J)
      DO 20 J=1,3
      T=BETA*AL(4-J)
      TH(4-J)=T
      A=DCMPLX(DCOS(T),0.D0)
      B=DCMPLX(0.D0,DSIN(T)*Z(4-J))
      C=DCMPLX(0.D0,DSIN(T)/Z(4-J))
      V(4-J)=A*V(5-J)+B*AI(5-J)
20    AI(4-J)=C*V(5-J)+A*AI(5-J)
      VG=V(1)+AI(1)
      RH=1.0-AI(1)*2.0/VG
      F=CDABS(RH)
      Y=2.0*DCONJG(RH)/VG
      DO 30 J=1,3
      ST=DSIN(TH(J))
      G(2*J)=DREAL(Y*(V(J)*AI(J)-V(J+1)*AI(J+1))/VG)/(Z(J)*F)
30    G(2*J-1)=ALFA*DREAL(Y*(V(J)*AI(J+1)-V(J+1)*AI(J))/VG)/(ST*F)
      RETURN
      END
```

DATE : **-***-**** TIME : **:**:**
MINIMAX OPTIMIZATION SYSTEM (MMOS PACKAGE. VERSION 86.01)

INPUT DATA

NUMBER OF VARIABLES (N) 6

NUMBER OF FUNCTIONS (M) 11

INITIAL STEP LENGTH (DX) 2.500E-01

ACCURACY REQUIRED FOR SOLUTION (EPS) 1.000E-06

MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) 200

STARTING POINT

	VARIABLES		FUNCTION VALUES
1	8.000000000000E-01	1	2.296956854739E-01
2	1.500000000000E+00	2	6.654872421565E-02
3	1.200000000000E+00	3	2.629788111390E-01
4	3.000000000000E+00	4	3.441313288788E-01
5	8.000000000000E-01	5	3.881323270514E-01
6	6.000000000000E+00	6	3.528638816933E-01
		7	2.807203998109E-01
		8	1.808150177638E-01
		9	1.491914625001E-01
		10	1.581989844434E-01
		11	2.409199354849E-01

FUNCTION EVALUATION : 15

	VARIABLES		FUNCTION VALUES
1	1.000493781528E+00	1	1.974993547241E-01
2	1.632679661106E+00	2	3.922053543408E-02
3	1.000008347164E+00	3	1.717636769803E-01
4	3.162553846780E+00	4	1.971127079618E-01
5	9.995020527000E-01	5	1.238118240839E-01
6	6.111537800268E+00	6	2.271382723502E-03
		7	1.238102129799E-01
		8	1.971095575180E-01
		9	1.717600068779E-01
		10	3.923181788768E-02
		11	1.975147234839E-01

SOLUTION

	VARIABLES		FUNCTION VALUES
1	9.999999992498E-01	1	1.972906269228E-01
2	1.634707139318E+00	2	3.946026367410E-02
3	1.000000000158E+00	3	1.719771329266E-01
4	3.162277663615E+00	4	1.972906269227E-01
5	1.000000001065E+00	5	1.238880208045E-01
6	6.117303697955E+00	6	1.752438041922E-09
		7	1.238880208045E-01
		8	1.972906269227E-01
		9	1.719771329266E-01
		10	3.946026367405E-02
		11	1.972906269228E-01

NUMBER OF FUNCTION EVALUATIONS 19

EXECUTION TIME (IN SECONDS) 1.23

Example 3

Minimize

$$\max \{f_1, f_2, f_3\}$$

subject to

$$-3x_1 - x_2 - 2.5 \geq 0,$$

where

$$f_1 = x_1^2 + x_2^2 + x_1x_2 - 1,$$

$$f_2 = \sin(x_1),$$

$$f_3 = -\cos(x_2).$$

The starting point is $x_1 = -2$ and $x_2 = -1$.

Reference

K. Madsen and H. Schjaer-Jacobsen, "Linearly constrained minimax optimization," Mathematical Programming, vol. 14, 1978, pp.208-223.

```
      PROGRAM TEST3
      PARAMETER (N=2,M=3,L=1,LEQ=0,LC=1)
      PARAMETER (IW=2*M*N+5*N*N+4*M+8*N+4*LC+3)
      REAL*8 X(N),W(IW),B(LC),C(LC,N),DX,EPS
      EXTERNAL FDF
      OPEN(6,FILE='SYS$OUTPUT',STATUS='UNKNOWN')
      B(1)=-2.5D0
      C(1,1)=-3.0D0
      C(1,2)=-1.0D0
      X(1)=-2.0D0
      X(2)=-1.0D0
      DX=0.2
      EPS=1.D-5
      MAXF=50
      LCH=6
      IPR=-100
      CALL MMOS2(FDF,N,M,X,DX,EPS,MAXF,W,IW,LCH,IPR,L,LEQ,B,C,LC)
      STOP
      END
C
C
C
      SUBROUTINE FDF(N,M,X,DF,F)
      REAL*8 X(N),F(M),DF(M,N),X1,X2
      X1=X(1)
      X2=X(2)
      F(1)=X1*X1+X2*X2+X1*X2-1.0D0
      F(2)=DSIN(X1)
      F(3)=-DCOS(X2)
      DF(1,1)=X1+X1+X2
      DF(1,2)=X1+X2+X2
      DF(2,1)=DCOS(X1)
      DF(2,2)=0.0D0
      DF(3,1)=0.0D0
      DF(3,2)=DSIN(X2)
      RETURN
      END
```

```
DATE : **-***-****          TIME :  **:**:**
MINIMAX OPTIMIZATION SYSTEM (MMOS PACKAGE. VERSION 86.01)


INPUT DATA
----------

  NUMBER OF VARIABLES (N) . . . . . .  2

  NUMBER OF FUNCTIONS (M) . . . . . .  3

  TOTAL NUMBER OF LINEAR CONSTRAINTS (L) . . . . . .  1

  NUMBER OF EQUALITY CONSTRAINTS (LEQ) . . . . . .  0

  INITIAL STEP LENGTH (DX) . . . . . 2.000E-01

  ACCURACY REQUIRED FOR SOLUTION (EPS) . . . . . 1.000E-05

  MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) . . . . . 50

STARTING POINT
--------------

                      VARIABLES                    FUNCTION VALUES

                1  -2.000000000000E+00        1   6.000000000000E+00
                2  -1.000000000000E+00        2  -9.092974268257E-01
                                               3  -5.403023058681E-01

VERIFICATION OF PARTIAL DERIVATIVES PERFORMED.


SOLUTION
--------

                      VARIABLES                    FUNCTION VALUES

                1  -8.928571428571E-01        1  -3.303571428571E-01
                2   1.785714285714E-01        2  -7.788668934368E-01
                                               3  -9.840984453126E-01


  NUMBER OF FUNCTION EVALUATIONS . . . . . .  9

  EXECUTION TIME (IN SECONDS) . . . . . .  0.04
```

Example 4

This is the problem proposed by Brent as an example in which the continuous analog of the Newton-Raphson method is not globally convergent. The problem is to solve the system of 2 nonlinear equations

$$4(x_1 + x_2) = 0,$$

$$(x_1 - x_2)((x_1 - 2)^2 + x_2^2) + 3x_1 + 5x_2 = 0.$$

The problem is formulated as to minimize

$$\max \{f, -f\}$$

subject to

$$4x_1 + 4x_2 = 0,$$

where

$$f = (x_1 - x_2)((x_1 - 2)^2 + x_2^2) + 3x_1 + 5x_2 .$$

The starting point is $x_1 = 2$ and $x_2 = 2$.

Reference

R.P. Brent, "On the Davidenko-Branin method for solving simultaneous nonlinear equations", IBM J. Research and Development, vol. 16, 1972, pp. 434-436.

```
      PROGRAM TEST4
      PARAMETER (N=2,M=2,L=1,LEQ=1,LC=1)
      PARAMETER (IW=2*M*N+5*N*N+4*M+8*N+4*LC+3)
      REAL*8 X(N),B(LC),C(LC,N),W(IW),DX,EPS
      EXTERNAL FDF
      DATA X/2.0,2.0/
      DATA B/0.0/,C/4.0,4.0/
      OPEN(6,FILE='SYS$OUTPUT',STATUS='UNKNOWN')
      DX=0.2
      EPS=1.D-6
      MAXF=50
      ICH=6
      IPR=50
      CALL MMOS2(FDF,N,M,X,DX,EPS,MAXF,W,IW,ICH,IPR,L,LEQ,B,C,LC)
      STOP
      END
C
C
C
      SUBROUTINE FDF(N,M,X,DF,F)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 X(N),F(M),DF(M,N)
      X1=X(1)
      X2=X(2)
      R1=X1-X2
      R2=(X1-2.0)**2+X2*X2
      F(1)=R1*R2+3.0*X1+5.0*X2
      F(2)=-F(1)
      DF(1,1)=R2+(R1+R1)*(X1-2.0)+3.0
      DF(1,2)=-R2+R1*(X2+X2)+5.0
      DF(2,1)=-DF(1,1)
      DF(2,2)=-DF(1,2)
      RETURN
      END
```

```
DATE : **-***-****          TIME :  **:**:**
MINIMAX OPTIMIZATION SYSTEM (MMOS PACKAGE. VERSION 86.01)


INPUT DATA
----------

  NUMBER OF VARIABLES (N) . . . . . .  2

  NUMBER OF FUNCTIONS (M) . . . . . .  2

  TOTAL NUMBER OF LINEAR CONSTRAINTS (L) . . . . . .  1

  NUMBER OF EQUALITY CONSTRAINTS (LEQ) . . . . . .  1

  INITIAL STEP LENGTH (DX) . . . . . 2.000E-01

  ACCURACY REQUIRED FOR SOLUTION (EPS) . . . . . 1.000E-06

  MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) . . . . . 50

STARTING POINT
--------------

                    VARIABLES                    FUNCTION VALUES

                1   2.000000000000E+00       1   1.600000000000E+01
                2   2.000000000000E+00       2  -1.600000000000E+01

SOLUTION
--------

                    VARIABLES                    FUNCTION VALUES

                1   1.850371707709E-17       1   0.000000000000E+00
                2  -1.295260195396E-16       2   0.000000000000E+00


  NUMBER OF FUNCTION EVALUATIONS . . . . . .  3

  EXECUTION TIME (IN SECONDS) . . . . . .  0.01
```

Example 5

Minimize the Beale constrained function

$$f = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3$$

subject to

$$x_i \geq 0, \quad i = 1,2,3,$$

$$3 - x_1 - x_2 - 2x_3 \geq 0.$$

The starting point is $x_1 = x_2 = x_3 = 0.5$.

Reference

J. Kowalik and M.R. Osborne, Methods for Unconstrained Optimization Problems. New York: Elsevier, 1968.

```
      PROGRAM TEST5
      PARAMETER (N=3,M=1,L=4,LEQ=0,LC=4)
      PARAMETER (IW=2*M*N+5*N*N+4*M+8*N+4*LC+3)
      REAL*8 X(N),W(IW),B(LC),C(LC,N),DX,EPS
      EXTERNAL FDF
      DATA B/0.0,0.0,0.0,3.0/
      DATA C/1.0,0.0,0.0,-1.0,0.0,1.0,0.0,-1.0,
     1         0.0,0.0,1.0,-2.0/
      OPEN(6,FILE='SYS$OUTPUT',STATUS='UNKNOWN')
      X(1)=0.5
      X(2)=0.5
      X(3)=0.5
      DX=0.25
      EPS=1.D-5
      MAXF=50
      IPR=150
      LCH=6
      CALL MMOS2(FDF,N,M,X,DX,EPS,MAXF,W,IW,LCH,IPR,L,LEQ,B,C,LC)
      STOP
      END
C
C
C
      SUBROUTINE FDF(N,M,X,DF,F)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 X(N),F(M),DF(M,N)
      X1=X(1)
      X2=X(2)
      X3=X(3)
      F(1)=9.0-8.0*X1-6.0*X2-4.0*X3+2.0*(X1*(X1+X2+X3)+X2*X2)+X3*X3
      DF(1,1)=4.0*X1+2.0*(X2+X3)-8.0
      DF(1,2)=4.0*X2+2.0*X1-6.0
      DF(1,3)=2.0*(X1+X3)-4.0
      RETURN
      END
```

```
DATE : **-***-****          TIME :  **:**:**
MINIMAX OPTIMIZATION SYSTEM (MMOS PACKAGE. VERSION 86.01)


INPUT DATA
----------

  NUMBER OF VARIABLES (N) . . . . . .  3

  NUMBER OF FUNCTIONS (M) . . . . . .  1

  TOTAL NUMBER OF LINEAR CONSTRAINTS (L) . . . . . .  4

  NUMBER OF EQUALITY CONSTRAINTS (LEQ) . . . . . .  0

  INITIAL STEP LENGTH (DX) . . . . . . 2.500E-01

  ACCURACY REQUIRED FOR SOLUTION (EPS) . . . . . 1.000E-05

  MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) . . . . . 50

STARTING POINT
--------------

                    VARIABLES                    FUNCTION VALUES

                1   5.000000000000E-01       1   2.250000000000E+00
                2   5.000000000000E-01
                3   5.000000000000E-01

SOLUTION
--------

                    VARIABLES                    FUNCTION VALUES

                1   1.333333340566E+00       1   1.111111111111E-01
                2   7.777777795101E-01
                3   4.444444399622E-01

  NUMBER OF FUNCTION EVALUATIONS . . . . . .  10

  EXECUTION TIME (IN SECONDS) . . . . . .  0.04
```

Example 6

This is once again the Beale constrained function (Example 3)

$$f_1 = 9 - 8x_1 - 6x_2 - 4x_3 + 2x_1^2 + 2x_2^2 + x_3^2 + 2x_1x_2 + 2x_1x_3$$

but in this case the constraint

$$3 - x_1 - x_2 - 2x_3 \geq 0$$

which is the only constraint active at the solution, is transformed into an additional function

$$f_2 = f_1 - (3 - x_1 - x_2 - 2x_3).$$

The problem is thus to minimize

$$\max \{f_1, f_2\}$$

subject to the remaining constraints

$$x_i \geq 0, \quad i = 1,2,3.$$

The starting point is $x_i = 0.5$, $i = 1,2,3$.

Reference

J.W. Bandler and C. Charalambous, "Nonlinear programming using minimax techniques", J. Optimization Theory and Applications, vol.13, 1974, pp. 607-619.

```
      PROGRAM TEST6
      PARAMETER (N=3,M=2,L=3,LEQ=0,LC=3)
      PARAMETER (IW=2*M*N+5*N*N+4*M+8*N+4*LC+3)
      REAL*8 X(N),W(IW),B(LC),C(LC,N),DX,EPS
      EXTERNAL FDF
      DATA B/0.0,0.0,0.0/
      DATA C/1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0/
      OPEN(6,FILE='SYS$OUTPUT',STATUS='UNKNOWN')
      X(1)=0.5
      X(2)=0.5
      X(3)=0.5
      DX=0.25
      EPS=1.D-6
      MAXF=50
      IPR=150
      LCH=6
      CALL MMOS2(FDF,N,M,X,DX,EPS,MAXF,W,IW,LCH,IPR,L,LEQ,B,C,LC)
      STOP
      END
C
C
C
      SUBROUTINE FDF(N,M,X,DF,F)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*8 X(N),F(M),DF(M,N)
      X1=X(1)
      X2=X(2)
      X3=X(3)
      F(1)=9.0-8.0*X1-6.0*X2-4.0*X3+2.0*(X1*(X1+X2+X3)+X2*X2)+X3*X3
      DF(1,1)=4.0*X1+2.0*(X2+X3)-8.0
      DF(1,2)=4.0*X2+2.0*X1-6.0
      DF(1,3)=2.0*(X1+X3)-4.0
      F(2)=F(1)+X1+X2+X3+X3-3.0
      DF(2,1)=DF(1,1)+1.0
      DF(2,2)=DF(1,2)+1.0
      DF(2,3)=DF(1,3)+2.0
      RETURN
      END
```

```
DATE : **-***-****          TIME :  **-**-**
MINIMAX OPTIMIZATION SYSTEM (MMOS PACKAGE. VERSION 86.01)


INPUT DATA
----------

  NUMBER OF VARIABLES (N) . . . . . .  3

  NUMBER OF FUNCTIONS (M) . . . . . .  2

  TOTAL NUMBER OF LINEAR CONSTRAINTS (L) . . . . . .  3

  NUMBER OF EQUALITY CONSTRAINTS (LEQ) . . . . . .  0

  INITIAL STEP LENGTH (DX) . . . . . 2.500E-01

  ACCURACY REQUIRED FOR SOLUTION (EPS) . . . . . 1.000E-06

  MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) . . . . . 50

STARTING POINT
--------------

                    VARIABLES                    FUNCTION VALUES

                1   5.000000000000E-01       1   2.250000000000E+00
                2   5.000000000000E-01       2   1.250000000000E+00
                3   5.000000000000E-01

SOLUTION
--------

                    VARIABLES                    FUNCTION VALUES

                1   1.333333333332E+00       1   1.111111111111E-01
                2   7.777777777756E-01       2   1.111111111111E-01
                3   4.444444444463E-01

  NUMBER OF FUNCTION EVALUATIONS . . . . . 11

  EXECUTION TIME (IN SECONDS) . . . . . 0.05
```

APPENDIX

PROGRAM LISTING

```
      SUBROUTINE mmosl(fdf,n,m,x,dx,eps,maxf,w,iw,lch,ipr)
      EXTERNAL fdf
      REAL*8 c(200),dc(1,200),x(*),w(*),dx,eps
C
C          ENTRY WITHOUT CONSTRAINTS
C
      CALL mmos2(fdf,n,m,x,dx,eps,maxf,w,iw,lch,ipr,0,0,c,dc,1)
      RETURN
      END
C
C
C
```

```
      SUBROUTINE mmos2(fdf,n,m,x,dx,eps,maxf,w,iw,lch,ipr,l,leq,c,dc,ic)
C
      EXTERNAL fdf
      REAL*8 c(*), dc(l,*), x(*), w(*) , dx, eps
      CHARACTER dattim*20
      INTEGER*4 cpu
      COMMON /mmx000/ nch,lvl,lgl,nrl
      COMMON /cmmx00/ dattim
      COMMON /mml000/ mark
C
C          ENTRY WITH LINEAR CONSTRAINTS
C
      nch=lch
      IF (lch.le.0) GOTO 10
      i=iabs(ipr)
      lvl=i/10
      lgl=i-lvl*10
      CALL lib$date_time (dattim)
      CALL header
      IF( l.gt.0 ) THEN
          write (lch,20) n,m,l,leq,dx,eps,maxf
          nrl=nrl-18
      ELSE
          write(lch,25) n,m,dx,eps,maxf
          nrl=nrl-14
      ENDIF
C
      write (lch,30)
      nrl=nrl-2
      CALL fdf (n,m,x,w(m+l),w(l))
      CALL prtfun (x,n,w,m)
      IF (lgl.gt.0) CALL prtdf (w(m+l),m,n)
C
      IF (ipr.ge.0) GOTO 10
      i=m*n+m+l
      j=i+m
      k=j+m
      CALL veridf (fdf,n,m,x,w(m+l),w(l),w(j),w(k),w(i))
   10 CALL lib$init_timer
      mark=l
C
C       CHECK INPUT QUANTITIES
C
      iwr=2*m*n+5*n*n+4*m+8*n+4*ic+3
      IF (iw.lt.iwr) THEN
          IF (lch.gt.0) write(lch,80) iwr
          RETURN
      ENDIF
      IF (n.lt.l.or.m.lt.l.or.l.lt.0.or.leq.lt.0.or.leq.gt.l.or.leq.gt.n
     l.or.ic.lt.l.or.dx.le.0.d0.or.eps.lt.0.d0.or.maxf.le.0) THEN
          IF (lch.gt.0) write(lch,70)
          RETURN
      ENDIF
```

```
C
C       SPLIT UP THE WORK AREA
C
      nl=n+1
      nn=n+n
      nf=1
      nfl=nf+m
      ndf=nfl+m
      ndfl=ndf+m*n
      nxl=ndfl+m*n
      nb=nxl+n
      nu=nb+n*n
      nr=nu+n*n
      na=nu
      ncl=na+nn*nn
      nwl=ncl+ic
      nwll=nwl+ic
      nxx=nwll+ic
      nw=nxx+n
      nwl=nw+n
      nw2=nwl+n
      nwm=nw2+n
      nas=nwm+m
      nks=nas+nl
      nks0=nks+nl
      nkstc=nks0+nl
      nkstf=nkstc+ic
      il=max0(1,ic)
      CALL mmos3 (fdf,n,m,1,leq,c,dc,il,x,dx,eps,maxf,nl,
     1nn,w(nf),w(nfl),w(ndf),w(ndfl),w(nxl),w(nb),w(nu),w(nr),w(na),
     2w(ncl),w(nwl),w(nwll),w(nxx),w(nw),w(nwl),w(nw2),w(nwm),w(nas),
     3w(nks),w(nks0),w(nkstc),w(nkstf),ifall)
C
      CALL lib$stat_timer(2,cpu)
      IF (lch.le.0) RETURN
      IF (ifall.eq.-1) THEN
        write (lch,60)
        RETURN
      ENDIF
      IF (nrl.lt.9) CALL header
      write (lch,40)
      nrl=nrl-4
      CALL prtfun (x,n,w,m)
      IF (nrl.lt.5) CALL header
      write (lch,50) maxf,real(cpu)*0.01
      RETURN
C
   20 FORMAT (/1X,'INPUT DATA'/1X,'----------'//
     1 '   NUMBER OF VARIABLES (N) ',25(2H. ),I4//
     2 '   NUMBER OF FUNCTIONS (M) ',25(2H. ),I4//
     3 '   TOTAL NUMBER OF LINEAR CONSTRAINTS (L)  ',17(2H. ),I4//
     4 '   NUMBER OF EQUALITY CONSTRAINTS (LEQ)  ',18(2H. ),I4//
     5 '   INITIAL STEP LENGTH (DX)  ',21(2H. ),1PE10.3//
     6 '   ACCURACY REQUIRED FOR SOLUTION (EPS)  ',15(2H. ),1PE10.3//
     7 '   MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) ',16(2H. ),I4/)
```

```
   25 FORMAT (/1X,'INPUT DATA'/1X,'----------'//
     1 '   NUMBER OF VARIABLES (N) ',25(2H. ),I4//
     2 '   NUMBER OF FUNCTIONS (M) ',25(2H. ),I4//
     3 '   INITIAL STEP LENGTH (DX)  ',21(2H. ),1PE10.3//
     4 '   ACCURACY REQUIRED FOR SOLUTION (EPS)  ',15(2H. ),1PE10.3//
     5 '   MAX NUMBER OF FUNCTION EVALUATIONS (MAXF) ',16(2H. ),I4/)
   30 FORMAT (1X,'STARTING POINT'/1X,'--------------')
   40 FORMAT (//9H SOLUTION/9H --------)
   50 FORMAT (//35H   NUMBER OF FUNCTION EVALUATIONS  ,21(2H. ),I4//
     1 31H   EXECUTION TIME (IN SECONDS) ,21(2H. ),1H.,F7.2/)
   60 FORMAT (//42H E M P T Y   F E A S I B L E   R E G I O N/)
   70 FORMAT (//40H I N C O R R E C T   P A R A M E T E R S/)
   80 FORMAT (//1X,'I N S U F F I C I E N T   W O R K I N G   S P A C E'
     1 //1X,'IW MUST AT LEAST BE ',I6/)
      END
C
C
C
```

```
      SUBROUTINE prtout (n,m,x,df,f,k)
C
      REAL*8 x(n), df(m,n), f(m)
      COMMON /mmx000/ lch,lvl,lgl,nrl
C
C     print results of one iteration
C
      IF (lch.le.0.or.lvl.eq.0) RETURN
      IF (mod(k,lvl).ne.0) RETURN
      IF (nrl.lt.7) CALL header
      write (lch,40) k
      nrl=nrl-2
      CALL prtfun (x,n,f,m)
      IF (lgl.eq.0) RETURN
      CALL prtdf (df,m,n)
      RETURN
C
   40 FORMAT (/1X,'FUNCTION EVALUATION :',I4)
      END
C
C
```

```
      SUBROUTINE prtfun (x,n,f,m)
C
      REAL*8 x(n), f(m)
      COMMON /mmx000/ lch,lvl,lgl,nrl
C
C     print values of variables and residual functions.
C
      IF (lch.le.0) RETURN
      k=max0(n,m)
      IF (nrl.lt.5) CALL header
      write (lch,10)
      nrl=nrl-3
      DO 40 i=1,k
        IF (nrl.le.0) CALL header
        IF (i.le.n) THEN
          IF (i.le.m) THEN
            write (lch,20) i,x(i),i,f(i)
          ELSE
            write (lch,20) i,x(i)
          ENDIF
        ELSE
          write (lch,30) i,f(i)
        ENDIF
        nrl=nrl-1
   40 CONTINUE
      RETURN
C
   10 FORMAT (/30X,9HVARIABLES,18X,15HFUNCTION VALUES/)
   20 FORMAT (18X,I4,2X,1PE19.12,5X,I4,2X,1PE19.12)
   30 FORMAT (48X,I4,2X,1PE19.12)
      END
C
C
```

```
      SUBROUTINE prtdf (g,m,n)
C
      REAL*8 g(m,n)
      COMMON /mmx000/ lch,lvl,lgl,nrl
C
C     print partial derivatives of residual functions.
C
      IF (lch.le.0) RETURN
      IF (nrl.lt.7) CALL header
      write (lch,10)
      nrl=nrl-2
      DO 60 k=1,n,10
        IF (nrl.lt.5) CALL header
        jl=k
        j2=min0(n,k+9)
        write (lch,20) (j,j=jl,j2)
        write (lch,30)
        nrl=nrl-3
        DO 50 i=1,m
          IF (nrl.le.0) CALL header
          write (lch,40) i,(g(i,j),j=jl,j2)
          nrl=nrl-1
   50   CONTINUE
   60 CONTINUE
      RETURN
C
   10 FORMAT (/1X,'  GRADIENTS ( DF.I / DX.J ) :')
   20 FORMAT (/10X,12HVARIABLES(J),10(I5,5X))
   30 FORMAT (10X,12HFUNCTIONS(I))
   40 FORMAT (10X,I6,4X,10(1PE10.2))
      END
C
C
```

```
      SUBROUTINE veridf (fdf,n,m,x,df,f,dg,dh,g)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 x(n), df(m,n), f(m), dg(m), dh(m,n), g(m)
      COMMON /mmx000/ lch,lvl,lgl,nrl
C
C     numerical verification of user-defined partial derivatives
C
      IF (lch.le.0) RETURN
      k=0
      CALL fdf (n,m,x,df,f)
      DO 60 i=1,n
        z=x(i)
        dx=1.d-6*z
        IF (dabs(dx).lt.1.d-10) dx=1.d-10
        dx2=dx+dx
        x(i)=z+dx
        CALL fdf (n,m,x,dh,f)
        DO 10 j=1,m
          dg(j)=dh(j,i)
   10   CONTINUE
        x(i)=z-dx
        CALL fdf (n,m,x,dh,g)
        x(i)=z
        DO 50 j=1,m
          y=df(j,i)
          z=f(j)-g(j)
          IF (dabs(z).le.0.5d-13*(f(j)+g(j))) z=0.0
          z=z/dx2
          IF (dabs(y).le.1.d-20.and.dabs(z).le.1.d-20) GOTO 50
          IF (dabs(z).lt.1.d-20) z=dsign(1.d-20,z)
          r=100.0*dabs((z-y)/z)
          IF (r.le.1.0) GOTO 50
          IF (dsign(1.d0,dg(j))+dsign(1.d0,dh(j,i)).eq.0.0) GOTO 50
          IF (k.ne.0) GOTO 30
          IF (nrl.lt.5) CALL header
          write (lch,20)
          nrl=nrl-4
   30     k=k+1
          IF (nrl.le.0) CALL header
          write (lch,40) j,i,y,z,r
          nrl=nrl-1
   50   CONTINUE
   60 CONTINUE
      IF (k.eq.0) THEN
        IF (nrl.lt.2) CALL header
        write (lch,70)
        nrl=nrl-2
      ENDIF
   80 RETURN
   20 FORMAT (/1X,'VERIFICATION OF PARTIAL DERIVATIVES :'//
     1 19X,52H DF.I / DX.J  :  USER DEFINED  NUMERICAL  DIFFERENCE)
   40 FORMAT (19X,I5,3X,I4,6X,1PE10.3,2X,1PE10.3,4X,0PF6.1,2H %)
```

```
   70 FORMAT (/1X,'VERIFICATION OF PARTIAL DERIVATIVES PERFORMED.')
      END
      SUBROUTINE header
C
      CHARACTER dat*11,tim*9
      COMMON /mmx000/ lch,lvl,lgl,nrl
      COMMON /cmmx00/ dat,tim
C
C     change page and print page header.
C
      IF (lch.le.0) RETURN
      nrl=55
      write (lch,30) dat,tim
      RETURN
C
   30 FORMAT (1H1/8H DATE : ,A11,10X,7HTIME : ,A9,/
     1 1X,'MINIMAX OPTIMIZATION SYSTEM (MMOS PACKAGE. VERSION 86.01)',
     2 //)
      END
C
C
C
```

```
      SUBROUTINE mmos3 (fdf,n,m,l,leq,c,dc,ic,x,dx,eps,maxf,
     1nl,nn,f,fl,df,dfl,xl,b,u,r,a,cloc,wl,wll,xx,w,wl,w2,wm,aset,
     2kset,kset0,kstatc,kstatf,ifall)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 c(ic),dc(ic,n),x(n),f(m),fl(m),wm(m),df(m,n),dfl(m,n)
      REAL*8 xx(nn),xl(n),u(n,n),r(n,n),b(n,n),a(nn,nn),cloc(ic)
      REAL*8 wl(ic),wll(ic),w(n),wl(n),w2(n),aset(nl)
      INTEGER kset(nl),kset0(nl),kstatc(ic),kstatf(m)
      LOGICAL div4,accum,shift
      EXTERNAL fdf
      COMMON /mml000/ mark
      DATA xzero,xone,xbig,xsml/0.d0,1.d0,1.d33,1.d-25/
C
C       SET SOME CONSTANTS
C
      seps=2.0**(-50)
      div4=.FALSE.
      li=l-leq
C
C       INITIALIZE
C
      keqset=0
      ncall=0
      nshift=0
      nstep=0
      fmmref=xbig
      dx0=dx
      DO 20 i=1,n
        DO 10 j=1,n
          b(i,j)=xzero
   10   CONTINUE
        b(i,i)=xone
   20 CONTINUE
C
C       FIND A FEASIBLE POINT
C
      IF (l.ne.0) THEN
        DO 40 i=1,l
          t=c(i)
          DO 30 j=1,n
            t=t+dc(i,j)*x(j)
   30     CONTINUE
          cloc(i)=t
   40   CONTINUE
        nact=0
        CALL feasi (cloc,dc,ic,leq,li,n,xx,nact,kset,aset,u,r,wl,w2,wl,
     1  wll,w,kstatc,ifall,accum,seps)
        IF (ifall.ne.0) RETURN
        DO 50 i=1,n
          x(i)=x(i)+xx(i)
   50   CONTINUE
        DO 70 i=1,l
```

```
            t=c(i)
            DO 60 j=1,n
              t=t+dc(i,j)*x(j)
   60       CONTINUE
            cloc(i)=t
   70     CONTINUE
        ENDIF
C
C         CALCULATE FUNCTION VALUES IN THE FIRST FEASIBLE POINT
C
        CALL fdf (n,m,x,df,f)
        fmm0=f(1)
        DO 90 i=1,m
          fmm0=dmax1(fmm0,f(i))
   90   CONTINUE
        xn=xzero
        DO 100 i=1,n
          xn=xn+x(i)*x(i)
  100   CONTINUE
        xn=dsqrt(xn)
        nact0=0
        ncall=1
C
C         ITERATIVE LOOP STARTS HERE
C
  110   nact=nact0
        IF (nact.ne.0) THEN
          DO 120 i=1,nact
            kset(i)=kset0(i)
  120     CONTINUE
        ENDIF
C
C         SOLVE THE LINEAR SUBPROBLEM
C
        CALL mmlpa (f,df,cloc,dc,m,n,nl,ic,leq,li,dx,xxn,xx,nact,kset,
       laset,u,r,wl,w2,fl,wm,wl,wll,kstatf,kstatc,w,seps,accum,fmm,ifall)
        IF (fmm.ge.fmm0) GOTO 410
C
C         CALCULATE FUNCTION VALUES IN THE NEW POINT
C
        DO 140 i=1,n
          xl(i)=x(i)+xx(i)
  140   CONTINUE
        CALL fdf (n,m,xl,dfl,fl)
        ncall=ncall+1
        CALL prtout (n,m,xl,dfl,fl,ncall)
        IF (mark.eq.0) GOTO 410
        fmml=fl(1)
        DO 150 i=1,m
          fmml=dmax1(fmml,fl(i))
  150   CONTINUE
C
C         REVISE THE STEP LENGTH
C
        IF ((fmm0-fmml).le.0.25*(fmm0-fmm)) THEN
```

```
          dx=0.25*xxn
          div4=.TRUE.
        ELSE
          IF (.not.div4.and.(fmm0-fmm1).gt.0.75*(fmm0-fmm)) dx=xxn+xxn
          div4=.FALSE.
        ENDIF
C
C       UPDATE THE HESSIAN APPROXIMATION
C
        DO 190 j=1,n
          w(j)=xzero
          w1(j)=xzero
  190   CONTINUE
        DO 210 i=1,nact
          k=kset(i)
          IF (k.gt.1) THEN
            kk=k-1
            t=-aset(i)
            DO 200 j=1,n
              w1(j)=w1(j)+t*df1(kk,j)
              w(j)=w(j)+t*df(kk,j)
  200       CONTINUE
          ENDIF
  210   CONTINUE
        DO 220 i=1,n
          w2(i)=w1(i)-w(i)
  220   CONTINUE
        CALL bfgs (b,n,w2,xx,w,seps)
C
C       TEST IF THE NEW POINT IS ACCEPTABLE
C
        IF ((fmm0-fmm1).le.0.01*(fmm0-fmm)) GOTO 320
C
C       COMPARE THE NEW ACTIVE SET WITH THE PRECEDING
C
        IF (nact0.eq.nact) THEN
          DO 240 i=1,nact
            k=kset(i)
            DO 230 j=1,nact
              IF (k.eq.kset0(j)) GOTO 240
  230       CONTINUE
            GOTO 250
  240     CONTINUE
          keqset=keqset+1
          GOTO 260
        ENDIF
  250   keqset=1
C
C       INTRODUCE THE NEW POINT
C
  260   nstep=nstep+1
        xn=xzero
        fmm0=fmm1
        nact0=nact
        DO 270 i=1,n
```

```
          x(i)=xl(i)
          xn=xn+x(i)**2
          DO 270 j=1,m
            df(j,i)=dfl(j,i)
  270   CONTINUE
        xn=dsqrt(xn)
        DO 280 i=1,m
          f(i)=fl(i)
  280   CONTINUE
        DO 290 i=1,nact0
          kset0(i)=kset(i)
  290   CONTINUE
        IF (l.ne.0) THEN
          DO 310 i=1,l
            t=c(i)
            DO 300 j=1,n
              t=t+dc(i,j)*x(j)
  300       CONTINUE
            cloc(i)=t
  310     CONTINUE
        ENDIF
C
C         TEST OF CONVERGENCE CRITERION
C
  320   IF (xxn.le.eps*xn) THEN
          GOTO 410
        ELSEIF (xxn.le.seps*xn) THEN
          ifail=2
          GOTO 410
        ELSEIF (xxn.le.xm50) THEN
          GOTO 410
        ELSEIF (ncall.ge.maxf) THEN
          ifail=3
          GOTO 410
        ENDIF
C
C         TEST FOR SWITCH TO STAGE-2
C
        shift=fmm0.le.fmmref.and.keqset.ge.3.and.nstep.ge.n
        IF (.not.shift) GOTO 110
        IF (nact.eq.nl) GOTO 380
C
C         TEST FOR POSITIVE DEFINITENESS OF THE HESSIAN APPROX.
C         IN A RELEVANT DIRECTION
C
        DO 340 i=1,nact
          k=kset(i)
          IF (k.le.1) THEN
            t=aset(i)
            DO 330 j=1,n
              wl(j)=wl(j)+t*dc(k,j)
  330       CONTINUE
          ENDIF
  340   CONTINUE
        DO 360 i=1,n
```

```
        t=xzero
        DO 350 j=1,n
          t=t+b(i,j)*wl(j)
  350   CONTINUE
        w(i)=t
  360 CONTINUE
      t=xzero
      DO 370 i=1,n
        t=t+w(i)*wl(i)
  370 CONTINUE
      IF (t.le.xzero) GOTO 110
C
C       shift TO STAGE-2
C
  380 nshift=nshift+1
      fmmref=fmm0-10.0*seps*dabs(fmm0)
      xxnmax=dmax1(dx0,dx+dx)
      CALL s21alq (fdf,n,m,l,leq,c,cloc,dc,ic,x,xxnmax,b,nact,
     1kset,aset,nl,kstatf,kstatc,a,xx,nn,f,df,xl,fl,dfl,wl,w2,eps,maxf,
     2ncall,xxn,nstep,seps,ifall)
      IF (ifall.gt.4) THEN
        fmm0=-xbig
        DO 390 i=1,m
          fmm0=dmax1(fmm0,f(i))
  390   CONTINUE
        dx=dmax1(dx,0.5*xxn)
        keqset=1
        GOTO 110
      ENDIF
C
C       RETURN
C
  410 maxf=ncall
      eps=xxn
      RETURN
      END
C
C
```

```
      SUBROUTINE mmlpa (f,df,c,dc,m,n,nl,ic,le,li,xnmax,xn,x,nact,kset,
     laset,u,r,dl,right,fup,dldf,cup,dldc,kstatf,kstatc,w,seps,accum,
     2fmax,ifall)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 f(m), df(m,n), c(ic), dc(ic,n), x(n), aset(nl), u(n,n),
     1r(n,n), dl(n), right(n), fup(m), dldf(m), cup(ic), dldc(ic), w(n)
      INTEGER kset(nl),kstatf(m),kstatc(ic)
      LOGICAL accum
      DATA xzero,xone,xbig/0.d0,1.d0,1.d33/
C
C     THE SUBROUTINE SOLVES A LINEARLY CONSTRAINED LINEAR MINIMAX
C     PROBLEM. THE STARTING POINT MUST BE FEASIBLE.
C
      lel=le+1
      leli=le+li
      xnmax2=xnmax*xnmax
      xn2=xzero
      eps=n*seps
      accum=.FALSE.
      ifall=0
      DO 10 i=1,n
        x(i)=xzero
   10 CONTINUE
      fmax=-xbig
      DO 20 i=1,m
        kstatf(i)=0
        t=f(i)
        IF (t.gt.fmax) THEN
          fmax=t
          kset0=i
        ENDIF
        fup(i)=t
   20 CONTINUE
      IF (leli.ne.0) THEN
        DO 30 i=1,leli
          kstatc(i)=0
          cup(i)=c(i)
   30   CONTINUE
      ENDIF
C
C       ACTIVATE INITIAL ACTIVE SET
C
      nactin=nact
      nact=0
      IF (le.ne.0) THEN
        DO 60 i=1,le
          DO 50 j=1,n
            r(j,i)=dc(i,j)
   50     CONTINUE
          kset(i)=i
          kstatc(i)=1
   60   CONTINUE
```

```
          CALL addcl (u,r,n,nact,le,right,w,accum,.FALSE.,eps)
          IF (nact.ne.le) THEN
            xn=xzero
            ifail=3
            RETURN
          ENDIF
        ENDIF
        IF (nactin.ge.lel) THEN
          DO 90 k=1,nactin
            kk=kset(k)
            IF (kk.ge.lel.and.kk.le.leli) THEN
              nact1=nact+1
              IF (nact1.gt.n) GOTO 100
              DO 80 i=1,n
                r(i,nact1)=dc(kk,i)
   80         CONTINUE
              CALL uttrns (u,n,nact,accum,r(1,nact1),w)
              CALL addcl (u,r,n,nact,1,right,w,accum,.FALSE.,eps)
              IF (nact.ge.nact1) THEN
                eps=eps+seps
                kset(nact1)=kk
                kstatc(kk)=1
              ENDIF
            ENDIF
   90     CONTINUE
        ENDIF
C
C         TRANSFORM OBJECTIVE FUNCTION GRADIENT
C
  100   kstatf(kset0)=1
        DO 110 j=1,n
          right(j)=-df(kset0,j)
  110   CONTINUE
        kset0=kset0+leli
        CALL uttrns (u,n,nact,accum,right,w)
C
C         ITERATIVE LOOP
C
C         CALCULATE MULTIPLIERS AND FIND THE LARGEST
C
  120   aset0=-xone
        IF (nact.eq.0) GOTO 240
        CALL rsolv (r,n,nact,right,aset)
        IF (nact.eq.le) GOTO 240
        amax=-xbig
        DO 130 i=lel,nact
          IF (kset(i).gt.leli) aset0=aset0-aset(i)
          IF (aset(i).gt.amax) THEN
            k=i
            amax=aset(i)
          ENDIF
  130   CONTINUE
        IF (amax.lt.xzero.and.aset0.lt.xzero) GOTO 240
        IF (amax.gt.aset0) GOTO 180
C
```

```
C       CHANGE OBJECTIVE FUNCTION
C
      DO 140 i=lel,nact
        IF (kset(i).gt.leli) THEN
          k=i
          GOTO 150
        ENDIF
  140 CONTINUE
  150 DO 170 i=1,k
        t=r(i,k)
        IF (k.ne.nact) THEN
          kl=k+1
          DO 160 j=kl,nact
            IF (kset(j).gt.leli) r(i,j)=r(i,j)-t
  160     CONTINUE
        ENDIF
        right(i)=right(i)+t
  170 CONTINUE
      kk=kset0
      kset0=kset(k)
      kset(k)=kk
C
C       DELETE ACTIVE CONSTRAINT NUMBER K
C
  180 kk=kset(k)
      IF (kk.gt.leli) THEN
        kstatf(kk-leli)=0
      ELSE
        kstatc(kk)=0
      ENDIF
      IF (.not.accum) THEN
        accum=.TRUE.
        CALL hacum (u,n,nact,w)
      ENDIF
      CALL delcl (k,u,r,n,nact,right,.TRUE.)
      eps=eps+seps
      IF (k.le.nact) THEN
        DO 200 i=k,nact
          kset(i)=kset(i+1)
  200   CONTINUE
      ENDIF
C
C       DELETE LINEAR DEPENDENCE LABELS
C
      DO 220 i=1,m
        IF (kstatf(i).eq.-2) kstatf(i)=0
  220 CONTINUE
      IF (li.eq.0) GOTO 120
      DO 230 i=lel,leli
        IF (kstatc(i).eq.-2) kstatc(i)=0
  230 CONTINUE
      GOTO 120
C
C       IS THERE AN UNBOUNDED SOLUTION ?
C
```

```
  240 IF (nact.eq.n) GOTO 490
C
C       CALCULATE THE PROJECTED GRADIENT
C
      k=nact+1
      t=xzero
      dln2=xzero
      DO 250 i=k,n
        dln2=dln2+right(i)**2
        dl(i)=right(i)
  250 CONTINUE
      IF (k.ne.1) THEN
        DO 260 i=1,nact
          t=t+right(i)**2
          dl(i)=xzero
  260   CONTINUE
      ENDIF
      t=t+dln2
      IF (t.le.xzero.or.dln2.le.eps*eps*t) THEN
        ifall=2
        GOTO 490
      ENDIF
      CALL utrns (u,n,nact,accum,dl,w)
C
C       PROJECT GRADIENTS ON THE PROJECTED GRADIENT
C
      DO 300 i=1,m
        t=xzero
        DO 290 j=1,n
          t=t+dl(j)*df(i,j)
  290   CONTINUE
        dldf(i)=t
  300 CONTINUE
      IF (leli.ne.0) THEN
        DO 320 i=1,leli
          t=xzero
          DO 310 j=1,n
            t=t+dl(j)*dc(i,j)
  310     CONTINUE
          dldc(i)=t
  320   CONTINUE
      ENDIF
C
C       CALCULATE STEP LENGTH
C
  330 sminc=xbig
      IF (li.ne.0) THEN
        DO 340 i=lel,leli
          IF (kstatc(i).eq.0) THEN
            t=dldc(i)
            IF (t.lt.xzero) THEN
              t=-cup(i)/t
              IF (t.le.sminc) THEN
                newc=i
                sminc=t
```

```
            ENDIF
          ENDIF
        ENDIF
  340   CONTINUE
      ENDIF
      sminf=xbig
      k=kset0-leli
      t0=dldf(k)
      f0=fup(k)
      DO 360 i=1,m
        IF (kstatf(i).eq.0) THEN
          t=t0-dldf(i)
          IF (t.lt.xzero) THEN
            t=(fup(i)-f0)/t
            IF (t.le.sminf) THEN
              sminf=t
              newf=i
            ENDIF
          ENDIF
        ENDIF
  360 CONTINUE
      step=dminl(sminf,sminc)
C
C       IN CASE THE STEP IS TOO LONG REDUCE AND RETURN
C
      s=step
      CALL limit (xnmax2,x,xn2,dl,dln2,s,n)
      IF (s.ne.step) THEN
        ifall=1
        step=s
        GOTO 440
      ENDIF
C
C       INCLUDE THE NEW FUNCTION/CONSTRAINT
C
      nact1=nact+1
      kk0=kset0-leli
      IF (sminf.ge.sminc) THEN
        DO 380 i=1,n
          r(i,nact1)=dc(newc,i)
  380   CONTINUE
        CALL uttrns (u,n,nact,accum,r(1,nact1),w)
        CALL addcl (u,r,n,nact,1,right,w,accum,.TRUE.,eps)
        IF (nact.ne.nact1) THEN
          kstatc(newc)=-2
          GOTO 330
        ENDIF
        kstatc(newc)=1
        kset(nact)=newc
        GOTO 430
      ENDIF
      DO 410 i=1,n
        r(i,nact1)=df(kk0,i)-df(newf,i)
  410 CONTINUE
      CALL uttrns (u,n,nact,accum,r(1,nact1),w)
```

```
      CALL addcl (u,r,n,nact,l,right,w,accum,.TRUE.,eps)
      IF (nact.ne.nactl) THEN
        kstatf(newf)=-2
        GOTO 330
      ENDIF
      kstatf(newf)=l
      kset(nact)=newf+leli
  430 eps=eps+seps
      IF (step.eq.xzero) GOTO 120
C
C       TAKE THE STEP AND UPDATE LINEAR FUNCTIONS
C
  440 fmax=-xbig
      xn2=xzero
      DO 450 i=l,n
        x(i)=x(i)+step*dl(i)
        xn2=xn2+x(i)**2
  450 CONTINUE
      DO 460 i=l,m
        t=fup(i)+step*dldf(i)
        IF (t.gt.fmax) fmax=t
        fup(i)=t
  460 CONTINUE
      IF (leli.ne.0) THEN
        DO 470 i=l,leli
          cup(i)=cup(i)+step*dldc(i)
  470   CONTINUE
      ENDIF
      IF (ifall.eq.0) GOTO 120
C
C       RETURN
C
  490 xn=dsqrt(xn2)
      nact=nact+l
      kset(nact)=kset0
      aset(nact)=aset0
      RETURN
      END
C
C
```

```
      SUBROUTINE linsys (a,b,idim,n,nr,eps)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 a(idim,idim), b(n)
      DATA xone,xsml/1.d0,1.d-25/
C
C     THE SUBROUTINE SOLVES A SYSTEM OF LINEAR EQUATIONS
C     USING GAUSSIAN ELIMINATION.
C
      nr=0
C
C       A IS CONSIDERED TO BE OF RANK K-1 if THE ABSOLUTE VALUE
C       OF THE K-TH PIVOT IS LESS THAN K*EPS.
C
      IF (n.gt.1) THEN
        GOTO 20
      ELSEIF (n.eq.1) THEN
        IF (dabs(a(1,1)).ge.xsml) THEN
          nr=1
          b(1)=b(1)/a(1,1)
        ENDIF
      ENDIF
      RETURN
C
C       EQUILIBRATION IN THE INFINITY NORM
C
   20 DO 40 i=1,n
        am=dabs(a(i,1))
        DO 30 j=2,n
          s=dabs(a(i,j))
          IF (am.lt.s) am=s
   30   CONTINUE
        IF (am.lt.xsml) am=xone
        b(i)=b(i)/am
        DO 40 j=1,n
          a(i,j)=a(i,j)/am
   40 CONTINUE
C
C       ELIMINATION
C
      nl=n-1
      DO 90 k=1,nl
        nr=k-1
C
C       FIND PIVOTAL ROW
C
        am=dabs(a(k,k))
        i0=k
        k1=k+1
        DO 50 i=k1,n
          s=dabs(a(i,k))
          IF (s.gt.am) THEN
            am=s
```

```
            i0=i
          ENDIF
   50   CONTINUE
        IF (am.lt.2*k*eps) RETURN
C
C       INTERCHANGE EQUATIONS K AND I0
C
        IF (i0.ne.k) THEN
          DO 60 j=k,n
            s=a(k,j)
            a(k,j)=a(i0,j)
            a(i0,j)=s
   60     CONTINUE
          s=b(k)
          b(k)=b(i0)
          b(i0)=s
        ENDIF
C
C       STORE PIVOT IN AM AND ELIMINATE IN ROWS K+1 TO N
C
        am=a(k,k)
        DO 90 i=kl,n
          s=a(i,k)/am
          DO 80 j=kl,n
            a(i,j)=a(i,j)-s*a(k,j)
   80     CONTINUE
          b(i)=b(i)-s*b(k)
   90 CONTINUE
      nr=nl
      IF (dabs(a(n,n)).lt.2*n*eps) RETURN
C
C       A HAS FULL RANK
C
      nr=n
C
C       BACK SUBSTITUTION
C
      b(n)=b(n)/a(n,n)
      k=n
      DO 110 i=2,n
        kl=k
        k=k-1
        s=b(k)
        DO 100 j=kl,n
          s=s-a(k,j)*b(j)
  100   CONTINUE
        b(k)=s/a(k,k)
  110 CONTINUE
      RETURN
      END
C
C
```

```
      SUBROUTINE bfgs (b,n,y,xx,w,seps)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 b(n,n), y(n), xx(n), w(n)
      DATA xzero/0.d0/
C
C     UPDATES A HESSIAN APPROXIMATION USING BFGS-FORMULA.
C
      eps=(n+10)*seps
      DO 20 i=1,n
        t=xzero
        DO 10 j=1,n
          t=t+b(i,j)*xx(j)
   10   CONTINUE
        w(i)=t
   20 CONTINUE
      yxx=xzero
      wxx=xzero
      yn=xzero
      xxn=xzero
      wn=xzero
      DO 30 i=1,n
        wi=w(i)
        xxi=xx(i)
        yi=y(i)
        yn=yn+yi*yi
        xxn=xxn+xxi*xxi
        wn=wn+wi*wi
        yxx=yxx+yi*xxi
        wxx=wxx+wi*xxi
   30 CONTINUE
      yn=dsqrt(yn)
      xxn=dsqrt(xxn)
      wn=dsqrt(wn)
      IF (yn.eq.xzero.or.wn.eq.xzero.or.xxn.eq.xzero) RETURN
      IF (dabs(yxx).lt.eps*yn*xxn.or.dabs(wxx).lt.eps*wn*xxn) RETURN
      DO 40 i=1,n
        b(i,i)=b(i,i)+y(i)**2/yxx-w(i)**2/wxx
   40 CONTINUE
      IF (n.ne.1) THEN
        DO 50 i=2,n
          DO 50 j=1,i-1
            b(i,j)=b(i,j)+y(i)*y(j)/yxx-w(i)*w(j)/wxx
            b(j,i)=b(i,j)
   50   CONTINUE
      ENDIF
      RETURN
      END
C
C
```

```
      SUBROUTINE addcl (u,r,n,kcol,knew,right,w,accum,lright,eps)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 u(n,n), r(n,n), right(n), w(n)
      LOGICAL accum,lright
      DATA xzero/0.d0/
C
C     UPDATES HOUSEHOLDER FACTORIZATION.
C     THE NEW COLUMNS MUST HAVE BEEN TRANSFORMED AS RIGHTHAND SIDES.
C
      kl=kcol+1
      k2=kcol+knew
C
C       COLUMN LOOP STARTS HERE
C
      DO 170 k=kl,k2
        s=xzero
        t=xzero
        IF (k.ne.1) THEN
          DO 10 i=1,k-1
            t=t+r(i,k)**2
   10     CONTINUE
         ENDIF
        DO 30 i=k,n
          s=s+r(i,k)**2
   30   CONTINUE
        t=t+s
        t=dsqrt(t)
        s=dsqrt(s)
C
C       RETURN IF THE NEW COLUMN DEPENDS LINEARLY ON THE
C       PRECEDING COLUMNS
C
        IF (t.eq.xzero.or.s.lt.t*eps) RETURN
C
C       PERFORM HOUSEHOLDER TRANSFORMATION
C
        tt=r(k,k)
        t=dabs(tt)
        alfa=dsqrt(s*(s+t))
        beta=-dsign(s,tt)
        r(k,k)=beta
        w(k)=(tt-beta)/alfa
        IF (k.ne.n) THEN
          kk=k+1
          DO 40 i=kk,n
            w(i)=r(i,k)/alfa
   40     CONTINUE
C
C       TRANSFORM THE REMAINING COLUMNS
C
          IF (k.ne.k2) THEN
            DO 70 j=kk,k2
```

```
                t=xzero
                DO 50 i=k,n
                  t=t+w(i)*r(i,j)
   50           CONTINUE
                DO 60 i=k,n
                  r(i,j)=r(i,j)-t*w(i)
   60           CONTINUE
   70         CONTINUE
            ENDIF
          ENDIF
C
C         TRANSFORM THE RIGHTHAND SIDE
C
          IF (lright) THEN
            t=xzero
            DO 90 i=k,n
              t=t+w(i)*right(i)
   90       CONTINUE
            DO 100 i=k,n
              right(i)=right(i)-t*w(i)
  100       CONTINUE
          ENDIF
C
C         ACCUMULATE THE TRANSFORMATIONS IN U
C         U MUST HAVE BEEN INITIALIZED
C
          IF (.not.accum) THEN
            DO 120 i=k,n
              u(i,k)=w(i)
  120       CONTINUE
          ELSE
            DO 160 i=1,n
              t=xzero
              DO 140 j=k,n
                t=t+u(i,j)*w(j)
  140         CONTINUE
              DO 150 j=k,n
                u(i,j)=u(i,j)-t*w(j)
  150         CONTINUE
  160       CONTINUE
          ENDIF
          kcol=kcol+1
  170   CONTINUE
        RETURN
        END
C
C
```

```
      SUBROUTINE delcl (k,u,r,n,kcol,right,lright)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 u(n,n), r(n,n), right(n)
      LOGICAL lright
C
C     DELETES COLUMN NUMBER K IN THE FACTORIZED MATRIX.
C     K MUST SATISFY      1.LE.K.LE.KCOL
C     U MUST HAVE BEEN ACCUMULATED.
C
      kcol=kcol-1
      IF (k.gt.kcol) RETURN
      DO 10 j=k,kcol
        jl=j+1
        DO 10 i=1,jl
          r(i,j)=r(i,jl)
   10 CONTINUE
C
C       TRANSFORM TO UPPER TRIANGULAR FORM
C       USING STANDARD GIVENS TRANSFORMATIONS
C
      DO 60 kk=k,kcol
        kl=kk+1
        x=r(kk,kk)
        y=r(kl,kk)
        a=dsqrt(x*x+y*y)
        c=x/a
        s=y/a
        r(kk,kk)=c*x+s*y
        IF (kk.ne.kcol) THEN
          DO 20 j=kl,kcol
            x=r(kk,j)
            y=r(kl,j)
            r(kk,j)=c*x+s*y
            r(kl,j)=c*y-s*x
   20     CONTINUE
        ENDIF
        IF (lright) THEN
          x=right(kk)
          y=right(kl)
          right(kk)=c*x+s*y
          right(kl)=c*y-s*x
        ENDIF
C
C       ACCUMULATE THE TRANSFORMATIONS
C
   40   DO 50 i=1,n
          x=u(i,kk)
          y=u(i,kl)
          u(i,kk)=c*x+s*y
          u(i,kl)=c*y-s*x
   50   CONTINUE
   60 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE feasi (c,dc,ic,le,li,n,x,nact,kset,aset,u,r,dl,right,
     lcup,dldc,w,kstat,ifall,accum,seps)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 c(ic),dc(ic,n),x(n),aset(n),u(n,n),r(n,n),dl(n)
      REAL*8 right(n),cup(ic),dldc(ic),w(n)
      INTEGER kset(n),kstat(ic)
      LOGICAL accum,object
      DATA xzero,xbig/0.d0,1.d33/
C
C     THE SUBROUTINE FINDS A FEASIBLE POINT FOR A SET OF LINEAR
C     EQUALITY AND INEQUALITY CONSTRAINTS.
C
      eps=(n+10)*seps
      accum=.FALSE.
      nactin=nact
      nact=0
      lel=le+1
      leli=le+li
      DO 10 i=1,n
        x(i)=xzero
   10 CONTINUE
      ifall=0
      IF (leli.eq.0) RETURN
      DO 20 i=1,leli
        kstat(i)=0
   20 CONTINUE
C
C       MAKE ACTIVE THE EQUALITY CONSTRAINTS PLUS OTHER CONSTRAINTS
C       AS DEFINED IN KSET
C
      IF (le.ne.0) THEN
        IF (le.gt.n) GOTO 410
        DO 40 i=1,le
          right(i)=-c(i)
          DO 30 j=1,n
            r(j,i)=dc(i,j)
   30     CONTINUE
          kset(i)=i
          kstat(i)=1
   40   CONTINUE
        CALL addcl (u,r,n,nact,le,right,w,accum,.FALSE.,eps)
        IF (nact.lt.le) GOTO 410
      ELSEIF (nactin.ge.1) THEN
        DO 70 k=1,nactin
          kk=kset(k)
          IF (kk.ge.lel.and.kk.le.leli) THEN
            nactl=nact+1
            IF (nactl.gt.n) GOTO 80
            DO 60 i=1,n
              r(i,nactl)=dc(kk,i)
   60       CONTINUE
            CALL uttrns (u,n,nact,accum,r(1,nactl),w)
```

```
            CALL addcl (u,r,n,nact,l,right,w,accum,.FALSE.,eps)
            IF (nact.ge.nactl) THEN
              right(nactl)=-c(kk)
              kset(nactl)=kk
              kstat(kk)=1
            ENDIF
          ENDIF
   70   CONTINUE
      ENDIF
   80 CALL tsolv (r,n,nact,right,x)
      IF (nact.ne.n) THEN
        nactl=nact+1
        DO 90 i=nactl,n
          x(i)=xzero
   90   CONTINUE
      ENDIF
      CALL utrns (u,n,nact,accum,x,w)
C
C       UPDATE THE CONSTRAINTS
C
      IF (li.eq.0) RETURN
      DO 120 i=lel,leli
        t=c(i)
        DO 110 j=1,n
          t=t+dc(i,j)*x(j)
  110   CONTINUE
        cup(i)=t
  120 CONTINUE
C
C       initialize inequality constraint loop
C
      DO 130 i=lel,leli
        IF (cup(i).lt.xzero.and.kstat(i).eq.0) kstat(i)=-1
  130 CONTINUE
C
C       ACTIVATE VIOLATED INEQUALITY CONSTRAINTS ONE BY ONE
C       USE THE STRONGEST VIOLATED AS OBJECTIVE CONSTRAINT
C
  140 fmin=xbig
      DO 150 i=lel,leli
        IF (kstat(i).eq.-1.and.cup(i).lt.fmin) THEN
          fmin=cup(i)
          new=i
        ENDIF
  150 CONTINUE
      IF (fmin.ge.xzero) RETURN
      DO 160 i=1,n
        right(i)=dc(new,i)
  160 CONTINUE
      CALL uttrns (u,n,nact,accum,right,w)
      kstat(new)=1
C
C       CALCULATE MULTIPLIERS FOR THE NEW ACTIVE CONSTRAINT AND DROP
C       CONSTRAINTS WITH POSITIVE MULTIPLIERS IN ORDER TO ACHIEVE
C       THE DIRECTION OF STEEPEST INCREMENT
```

```
C
  170 IF (nact.eq.le) GOTO 220
      CALL rsolv (r,n,nact,right,aset)
      amax=-xbig
      DO 180 i=lel,nact
        IF (aset(i).ge.amax) THEN
          k=i
          amax=aset(i)
        ENDIF
  180 CONTINUE
      IF (amax.lt.xzero) GOTO 220
      kstat(kset(k))=0
      DO 190 i=lel,leli
        IF (kstat(i).eq.-2) kstat(i)=0
  190 CONTINUE
      IF (.not.accum) THEN
        accum=.TRUE.
        CALL hacum (u,n,nact,w)
      ENDIF
      CALL delcl (k,u,r,n,nact,right,.TRUE.)
      IF (k.le.nact) THEN
        DO 210 i=k,nact
          kset(i)=kset(i+1)
  210   CONTINUE
      ENDIF
      GOTO 170
C
C       calculate the projected gradient
C
  220 t=xzero
      dln2=xzero
      IF (nact.ne.0) THEN
        DO 230 i=1,nact
          t=t+right(i)**2
          dl(i)=xzero
  230   CONTINUE
      ENDIF
      nactl=nact+1
      IF (nact.ne.n) THEN
        DO 250 i=nactl,n
          dln2=dln2+right(i)**2
          dl(i)=right(i)
  250   CONTINUE
      ENDIF
      t=t+dln2
      IF (t.le.xzero.or.dln2.le.eps*eps*t) THEN
        s=(n+1)*dabs(c(new))
        DO 270 i=1,n
          s=s+dabs(dc(new,i)*x(i))*(n+3-i)
  270   CONTINUE
        IF (cup(new).lt.-eps*s) GOTO 410
        kstat(new)=0
        GOTO 140
      ENDIF
  280 CALL utrns (u,n,nact,accum,dl,w)
```

```
C
C       PROJECT GRADIENTS ON THE PROJECTED GRADIENT
C
      DO 300 i=lel,leli
        t=xzero
        DO 290 j=1,n
          t=t+dl(j)*dc(i,j)
  290   CONTINUE
        dldc(i)=t
  300 CONTINUE
C
C       CALCULATE STEP LENGTH "ANES" TO MAKE THE OBJECTIVE CONSTRAINT
C       EQUAL ZERO, AND CALCULATE THE STEP LENGTH "AMIN" TO THE
C       NEAREST INACTIVE CONSTRAINT UNDER CONSIDERATION
C
      anes=-cup(new)/dln2
  310 amin=xbig
      DO 320 i=lel,leli
        IF (kstat(i).eq.0) THEN
          t=dldc(i)
          IF (t.lt.xzero) THEN
            t=-cup(i)/t
            IF (t.le.amin) THEN
              amin=t
              k=i
            ENDIF
          ENDIF
        ENDIF
  320 CONTINUE
C
C       WILL THE OBJECTIVE CONSTRAINT GET ACTIVE ?
C       if NOT, MAKE ACTIVE THE CLOSEST
C
      object=anes.le.amin
      alfa=dminl(amin,anes)
      nactl=nact+1
      IF (.not.object) THEN
        DO 330 i=1,n
          r(i,nactl)=dc(k,i)
  330   CONTINUE
        CALL uttrns (u,n,nact,accum,r(1,nactl),w)
        CALL addcl (u,r,n,nact,l,right,w,accum,.TRUE.,eps)
        IF (nactl.ne.nact) THEN
          kstat(k)=-2
          GOTO 310
        ENDIF
  340   kstat(k)=1
        kset(nact)=k
      ENDIF
C
C       TAKE THE STEP
C
  350 IF (alfa.ne.xzero) THEN
        DO 360 i=1,n
          x(i)=x(i)+alfa*dl(i)
```

```
  360   CONTINUE
        DO 370 i=lel,leli
          t=cup(i)+alfa*dldc(i)
          IF (kstat(i).eq.-1.and.t.ge.xzero) kstat(i)=0
          cup(i)=t
  370   CONTINUE
      ENDIF
      IF (.not.object) GOTO 170
C
C       ACTIVATE THE OBJECTIVE CONSTRAINT
C
      DO 390 i=1,n
        r(i,nactl)=right(i)
  390 CONTINUE
      CALL addcl (u,r,n,nact,l,right,w,accum,.FALSE.,eps)
      IF (nact.ne.nactl) THEN
        kstat(new)=0
      ELSE
        kset(nact)=new
      ENDIF
      GOTO 140
C
C       NO FEASIBLE POINTS
C
  410 ifail=-1
      RETURN
      END
C
```

```
      SUBROUTINE s2lalq (fdf,n,m,l,leq,c,cloc,dc,ic,x,xxnmax,b,
     lnact,kset,aset,nl,kstatf,kstatc,dz,zz,nn,f,df,xl,fl,dfl,w,wl,eps,
     2maxf,ncall,xxn,nstep,seps,ifall)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 c(ic),cloc(ic),dc(ic,n),x(n),b(n,n),aset(nl),dz(nn,nn)
      REAL*8 zz(nn),f(m),df(m,n),xl(n),fl(m),dfl(m,n),w(n),wl(n)
      INTEGER kset(nl),kstatf(m),kstatc(ic)
      EXTERNAL fdf
      COMMON /mml000/ mark
      DATA xzero,xone,xbig,xsml/0.d0,1.d0,1.d33,1.d-25/
C
C     STAGE-2 (QUASI-NEWTON) ALGORITHM FOR LINEARLY CONSTRAINED
C     MINIMAX OPTIMIZATION.
C
      li=l-leq
      lel=leq+l
      ifall=0
      sseps=dsqrt(seps)
      kk0=kset(nact)-l
      nactl=nact-l
      nz=n+nactl
      nstep2=0
      xxn=xzero
      DO 10 i=l,m
        kstatf(i)=0
   10 CONTINUE
      IF (l.ne.0) THEN
        DO 20 i=l,l
          kstatc(i)=0
   20   CONTINUE
      ENDIF
      DO 40 i=l,nact
        k=kset(i)
        IF (k.le.l) THEN
          kstatc(k)=l
        ELSE
          kstatf(k-l)=l
        ENDIF
   40 CONTINUE
C
C       ITERATIVE LOOP STARTS HERE
C
C       SET UP THE ITERATION MATRIX AND THE RIGHTHAND SIDE
C
   50 DO 70 i=l,n
        DO 60 j=l,n
          dz(i,j)=b(i,j)
   60   CONTINUE
        zz(i)=-df(kk0,i)
   70 CONTINUE
      IF (nact.ne.l) THEN
        DO 140 j=l,nactl
```

```
            k=kset(j)
            jn=j+n
            IF (k.le.1) THEN
              zz(jn)=-cloc(k)
              DO 80 i=1,n
                dz(i,jn)=dc(k,i)
                dz(jn,i)=dz(i,jn)
   80         CONTINUE
            ELSE
              kk=k-1
              zz(jn)=f(kk)-f(kk0)
              DO 100 i=1,n
                dz(i,jn)=df(kk0,i)-df(kk,i)
                dz(jn,i)=dz(i,jn)
  100         CONTINUE
            ENDIF
            DO 120 i=nl,nz
              dz(i,jn)=xzero
  120       CONTINUE
            t=aset(j)
            DO 130 i=1,n
              zz(i)=zz(i)-t*dz(jn,i)
  130       CONTINUE
  140     CONTINUE
        ENDIF
        res0=xzero
        DO 160 i=1,nz
          res0=res0+zz(i)**2
  160   CONTINUE
        res0=dsqrt(res0)
C
C       CALCULATE THE QUASI-NEWTON STEP
C
        CALL linsys (dz,zz,nn,nz,k,seps)
        IF (k.ne.nz) THEN
          ifail=8
          RETURN
        ENDIF
C
C       CONTROL STEP LENGTH
C
        xxnl=xzero
        alfa=xone
        DO 180 i=1,n
          xxnl=xxnl+zz(i)**2
  180   CONTINUE
        xxnl=dsqrt(xxnl)
        IF (xxnl.gt.xxnmax) alfa=xxnmax/xxnl
C
C       WILL OTHER CONSTRAINTS OR FUNCTIONS BECOME ACTIVE ?
c
        step=xbig
        IF (li.ne.0) THEN
          DO 200 i=lel,l
            IF (kstatc(i).eq.0) THEN
```

```
            t=xzero
            DO 190 j=1,n
              t=t+zz(j)*dc(i,j)
  190       CONTINUE
            IF (t.lt.xzero) THEN
              t=-cloc(i)/t
              IF (t.le.step) step=t
            ENDIF
          ENDIF
  200   CONTINUE
      ENDIF
      t0=xzero
      DO 220 i=1,n
        t0=t0+zz(i)*df(kk0,i)
  220 CONTINUE
      f0=f(kk0)
      DO 240 i=1,m
        IF (kstatf(i).eq.0) THEN
          t=xzero
          DO 230 j=1,n
            t=t+zz(j)*df(i,j)
  230       CONTINUE
          t=t0-t
          IF (t.lt.xzero) THEN
            t=(f(i)-f0)/t
            IF (t.le.step) step=t
          ENDIF
        ENDIF
  240 CONTINUE
      IF (step.le.alfa) THEN
        ifall=9
        alfa=step
      ENDIF
C
C       SCALE THE STEP
C
      DO 260 i=1,nz
        zz(i)=alfa*zz(i)
  260 CONTINUE
      xxnl=dabs(alfa)*xxnl
C
C       CALCULATE FUNCTION VALUES AND RESIDUALS IN THE NEW POINT
C
      xnl=xzero
      DO 270 i=1,n
        xl(i)=x(i)+zz(i)
        xnl=xnl+xl(i)**2
  270 CONTINUE
      xnl=dsqrt(xnl)
      ncall=ncall+1
      CALL fdf (n,m,xl,dfl,fl)
      CALL prtout (n,m,xl,dfl,fl,ncall)
      IF (mark.eq.0) THEN
        ifall=4
        RETURN
```

```
      ENDIF
      daset0=xzero
      IF (nact.ne.1) THEN
        DO 280 i=nl,nz
          IF (kset(i-n).gt.1) daset0=daset0-zz(i)
  280   CONTINUE
      ENDIF
      res=xzero
      t=aset(nact)+daset0
      DO 300 i=1,n
        w(i)=-t*df(kk0,i)
        wl(i)=-t*dfl(kk0,i)
  300 CONTINUE
      IF (nact.ne.1) THEN
        DO 340 j=1,nactl
          k=kset(j)
          jn=j+n
          t=aset(j)+zz(jn)
          IF (k.le.1) THEN
            s=c(k)
            DO 310 i=1,n
              ss=dc(k,i)
              w(i)=w(i)+t*ss
              wl(i)=wl(i)+t*ss
              s=s+ss*xl(i)
  310       CONTINUE
            res=res+s*s
          ELSE
            kk=k-1
            DO 330 i=1,n
              w(i)=w(i)-t*df(kk,i)
              wl(i)=wl(i)-t*dfl(kk,i)
  330       CONTINUE
            res=res+(fl(kk0)-fl(kk))**2
          ENDIF
  340   CONTINUE
      ENDIF
      DO 360 i=1,n
        res=res+wl(i)**2
  360 CONTINUE
      res=dsqrt(res)
C
C       UPDATE THE HESSIAN APPROXIMATION
C
      DO 370 i=1,n
        wl(i)=wl(i)-w(i)
  370 CONTINUE
      CALL bfgs (b,n,wl,zz,w,seps)
C
C       TEST IF THE RESIDUAL HAS DECREASED
C
      IF (nstep2.eq.0.or.res.le.0.999*res0) GOTO 390
C
C       IF NO - TEST FOR MACHINE ACCURACY
C
```

```
      IF (xxnl.gt.sseps*(xxnmax+xnl).or.nstep2.lt.2) THEN
        ifall=5
      ELSE
        ifall=2
      ENDIF
      RETURN
C
C       IF YES - INTRODUCE THE NEW POINT
C
  390 nstep2=nstep2+1
      nstep=nstep+1
      xn=xzero
      DO 400 i=1,n
        x(i)=xl(i)
        DO 400 j=1,m
          df(j,i)=dfl(j,i)
  400 CONTINUE
      xn=xnl
      xxn=xxnl
      fmax=-xbig
      DO 410 i=1,m
        t=fl(i)
        fmax=dmaxl(t,fmax)
        f(i)=t
  410 CONTINUE
      aset(nact)=aset(nact)+daset0
      IF (aset(nact).gt.xzero) ifall=6
      IF (nact.ne.1) THEN
        DO 420 i=1,nact1
          in=i+n
          aset(i)=aset(i)+zz(in)
          IF (kset(i).gt.leq.and.aset(i).gt.xzero) ifall=6
  420   CONTINUE
      ENDIF
      IF (l.ne.0) THEN
        DO 450 j=1,l
          t=c(j)
          DO 440 i=1,n
            t=t+dc(j,i)*x(i)
  440     CONTINUE
          cloc(j)=t
  450   CONTINUE
C
C       TEST IF THE ACTIVE SET IS COMPLETE
C
        t=fmax+res
        DO 460 i=1,m
          IF (f(i).gt.t) THEN
            ifall=7
            RETURN
          ENDIF
  460   CONTINUE
      ENDIF
C
C       TEST CONVERGENCE CRITERION
```

```
C
  470 IF (xxn.le.eps*xn) THEN
        IF (nact.lt.nl) ifall=1
      ELSEIF (xxn.le.seps*xn) THEN
        ifall=2
      ELSEIF (xxn.le.xsml) THEN
        ifall=0
      ELSEIF (ncall.ge.maxf) THEN
        ifall=3
      ELSEIF (ifall.le.4) THEN
        GOTO 50
      ENDIF
      RETURN
      END
C
C
```

```
      SUBROUTINE uttrns (u,n,kcol,accum,r,w)
C
      REAL*8 t,xzero
      REAL*8 u(n,n), r(n), w(n)
      LOGICAL accum
      DATA xzero/0.d0/
C
C     TRANSFORM THE VECTOR R AS A RIGHTHAND SIDE.
C
C       IF THE TRANSFORMATIONS HAVE BEEN ACCUMULATED
C       DO SIMPLE MATRIX-MULTIPLICATION
C       ELSE TRANSFORM RIGHTHAND SIDES
C
      IF (.not.accum) THEN
        IF (kcol.ne.0) THEN
          DO 30 k=1,kcol
            t=xzero
            DO 10 i=k,n
              t=t+r(i)*u(i,k)
   10       CONTINUE
            DO 20 i=k,n
              r(i)=r(i)-t*u(i,k)
   20       CONTINUE
   30     CONTINUE
        ENDIF
      ELSE
        DO 50 i=1,n
          w(i)=r(i)
   50   CONTINUE
        DO 70 k=1,n
          t=xzero
          DO 60 i=1,n
            t=t+u(i,k)*w(i)
   60     CONTINUE
          r(k)=t
   70   CONTINUE
      ENDIF
      RETURN
      END
C
C
```

```
      SUBROUTINE utrns (u,n,kcol,accum,r,w)
C
      REAL*8 t,xzero
      REAL*8 u(n,n), r(n), w(n)
      LOGICAL accum
      DATA xzero/0.d0/
C
C     TRANSFORM THE VECTOR R OPPOSITE A RIGHTHAND SIDE.
C
      kl=kcol+1
      IF (.not.accum) THEN
        IF (kcol.ne.0) THEN
          DO 30 kk=1,kcol
            k=kl-kk
            t=xzero
            DO 10 j=k,n
              t=t+u(j,k)*r(j)
10          CONTINUE
            DO 20 j=k,n
              r(j)=r(j)-t*u(j,k)
20          CONTINUE
30        CONTINUE
        ENDIF
      ELSE
        DO 50 i=1,n
          w(i)=r(i)
50      CONTINUE
        DO 70 i=1,n
          t=xzero
          DO 60 j=1,n
            t=t+u(i,j)*w(j)
60        CONTINUE
          r(i)=t
70      CONTINUE
      ENDIF
      RETURN
      END
C
C
```

```
      SUBROUTINE rsolv (r,n,kcol,right,x)
C
      REAL*8 t
      REAL*8 r(n,n), right(n), x(n)
C
C     PERFORM BACK SUBSTITUTION ON RIGHT.
C
C
C       CALCULATE ALFA USING BACK SUBSTITUTION ON R
C
      IF (kcol.gt.0) THEN
        DO 20 k=kcol,1,-1
          kl=k+1
          t=right(k)
          IF (kl.le.kcol) THEN
            DO 10 j=kl,kcol
              t=t-x(j)*r(k,j)
10          CONTINUE
          ENDIF
          x(k)=t/r(k,k)
20      CONTINUE
      ENDIF
      RETURN
      END
C
C
```

```
      SUBROUTINE tsolv (r,n,kcol,right,x)
C
      REAL*8 t
      REAL*8 r(n,n), right(n), x(n)
C
C     PERFORM BACK SUBSTITUTION ON RIGHT USING THE
C     TRANSPOSED TRIANGULAR MATRIX.
C
      IF (kcol.eq.0) RETURN
      x(1)=right(1)/r(1,1)
      IF (kcol.ne.1) THEN
        DO 20 i=2,kcol
          t=right(i)
          DO 10 j=1,i-1
            t=t-x(j)*r(j,i)
   10     CONTINUE
          x(i)=t/r(i,i)
   20   CONTINUE
      ENDIF
      RETURN
      END
C
C
```

```
      SUBROUTINE hacum (u,n,kcol,w)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 u(n,n), w(n)
      DATA xzero,xone/0.d0,1.d0/
C
C     ACCUMULATES HOUSEHOLDER VECTORS STORED IN LOWER TRIANGLE
C     OF THE FIRST KCOL COLUMNS OF U IN AN ORTHONORMAL MATRIX U.
C     THE HOUSEHOLDER VECTORS MUST HAVE TWO NORM EQUAL TO TWO.
C     KCOL.GE.1 .
C
      kl=kcol+1
      DO 10 i=kcol,n
        w(i)=u(i,kcol)
   10 CONTINUE
      DO 20 i=kcol,n
        u(i,i)=xone-w(i)**2
   20 CONTINUE
      IF (kcol.ne.n) THEN
        DO 30 i=kl,n
          t=w(i)
          DO 30 j=kcol,i-1
            s=-t*w(j)
            u(i,j)=s
            u(j,i)=s
   30   CONTINUE
      ENDIF
      IF (kcol.eq.1) RETURN
C
      DO 100 kk=2,kcol
        k=kl-kk
        DO 50 i=k,n
          w(i)=u(i,k)
   50   CONTINUE
        t=w(k)
        kpl=k+1
        u(k,k)=xone-t*t
        DO 60 i=kpl,n
          u(i,k)=-t*w(i)
   60   CONTINUE
        DO 90 l=kpl,n
          s=xzero
          DO 70 i=kpl,n
            s=s+w(i)*u(i,l)
   70     CONTINUE
          u(k,l)=-t*s
          DO 80 i=kpl,n
            u(i,l)=u(i,l)-s*w(i)
   80     CONTINUE
   90   CONTINUE
  100 CONTINUE
      RETURN
      END
C
C
```

```
      SUBROUTINE limit (xnmax2,x,xn2,p,pn2,alfa,n)
C
      IMPLICIT REAL*8 (a-h,o-z)
      REAL*8 x(n), p(n)
      DATA xzero/0.d0/
C
C     LIMIT THE STEP LENGTH ALFA.
C
      xtp=xzero
      DO 10 i=1,n
        xtp=xtp+x(i)*p(i)
   10 CONTINUE
      b=xtp/pn2
      t=dsqrt(b*b+(xnmax2-xn2)/pn2)
      ap=t-b
      am=-t-b
      IF (alfa.gt.ap) alfa=ap
      IF (alfa.lt.am) alfa=am
      RETURN
      END
```

A Superlinearly Convergent Minimax Algorithm for Microwave Circuit Design

JOHN W. BANDLER, FELLOW, IEEE, WITOLD KELLERMANN, STUDENT MEMBER, IEEE, AND KAJ MADSEN

Abstract — A new and highly efficient algorithm for nonlinear minimax optimization is presented. The algorithm, based on the work of Hald and Madsen, combines linear programming methods with quasi-Newton methods and has sure convergence properties. A critical review of the existing minimax algorithms is given, and the relation of the quasi-Newton iteration of the algorithm to the Powell method for nonlinear programming is discussed. To demonstrate the superiority of this algorithm over the existing ones, the classical three-section transmission-line transformer problem is used. A novel approach to worst-case design of microwave circuits using the present algorithm is proposed. The robustness of the algorithm is proved by using it for practical design of contiguous and noncontiguous-band multiplexers, involving up to 75 design variables.

I. Introduction

A WIDE CLASS OF microwave circuit and system design problems can be formulated as minimax optimization problems. Most commonly, the minimax functions result from lower and/or upper specifications on a performance function of interest. In practice, we form error functions at a finite discrete set of frequencies and assume that a sufficient number of sample points have been chosen so that the discrete approximation problem adequately approximates the continuous problem. This may result in a large number of minimax functions to be minimized. Therefore, a highly efficient and fast algorithm for minimax optimization is of great importance to many microwave circuit designers and engineers. It is the purpose of this paper to present such an algorithm.

The plan of the paper is as follows. In Section II, previous work in the area of nonlinear minimax optimization is briefly reviewed. The algorithm of this paper is described in more detail in Section III, where the two methods, namely, the first-order method and the approximate second-order method, are presented and the switching conditions between the two methods are given. Our attention is focused on explaining the ideas behind the algorithm and illustrating them with microwave circuit examples. A detailed discussion on the relation of the quasi-Newton iteration of the algorithm to the Powell method for nonlinear programming is given in Appendix A. Section IV contains the comparison of the present algorithm with the existing minimax algorithms using a three-section transmission-line transformer problem.

A novel approach to worst-case tolerance design of microwave circuits taking full advantage of the present algorithm is described in Section V. Previous work in this area has been concentrated on worst-case design techniques disregarding the source of the minimax functions, i.e., the discretization of a continuous problem. Our approach, which is believed to be new to the microwave tolerance design area, integrates a search technique for maxima of the response (a technique based on cubic interpolation) with the worst-case search using linearly constrained optimization.

In Section VI, an optimization procedure for practical design of contiguous- and noncontiguous-band microwave multiplexers using the present algorithm is described and illustrated by a five-channel, 11-GHz multiplexer design.

We conclude in Section VII with an assessment of the current applicability and potential impact of the algorithm in the area of microwave circuit design.

II. Review of Minimax Algorithms

A. Formulation of the Problem

The mathematical formulation of the linearly constrained minimax problem is the following. Let

$$f_j(\boldsymbol{x}) = f_j(x_1, \cdots, x_n), \qquad j = 1, \cdots, m$$

be a set of m nonlinear, continuously differentiable functions. The vector $\boldsymbol{x} \triangleq [x_1 \quad x_2 \cdots x_n]^T$ is the set of n parameters to be optimized.

We consider the optimization problem

$$\underset{\boldsymbol{x}}{\text{minimize}} F(\boldsymbol{x}) \triangleq \max_j \{ f_j(\boldsymbol{x}) \}$$

subject to

$$\begin{aligned} \boldsymbol{a}_i^T \boldsymbol{x} + b_i &= 0, \qquad i = 1, \cdots, l_{\text{eq}} \\ \boldsymbol{a}_i^T \boldsymbol{x} + b_i &\geq 0, \qquad i = l_{\text{eq}} + 1, \cdots, l \end{aligned} \tag{1}$$

where $\boldsymbol{a}_i$ and b_i, $i = 1, \cdots, l$, are constants.

B. Methods Based on Linearization

Over the past 15 years, this type of problem has been considered by many researchers. Usually, only the unconstrained minimax problem is treated, however. But, in the

Manuscript received February 13, 1985; revised May 28, 1985.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grants A7239 and G1135.

J. W. Bandler and W. Kellermann are with the Simulation Optimization Systems Research Laboratory and the Department of Electrical and Computer Engineering, McMaster University, Hamilton, Canada L8S 4L7.

K. Madsen is with the Institute for Numerical Analysis, Technical University of Denmark, Building 303, DK-2800 Lyngby, Denmark.

J. W. Bandler is also with Optimization Systems Associates, 163 Watson's Lane, Dundas, Ontario, Canada L9H 6L1.

0018-9480/85/1200-1519$01.00 ©1985 IEEE

type of methods to be described in the present paper, it is no complication and computationally costless to add the linear constraints.

Many of the minimax papers use the objective function

$$\hat{F}(\boldsymbol{x}) \triangleq \max_j |f_j(\boldsymbol{x})|$$

instead of F. There is no significant difference between these two optimization problems. We prefer (1) since it is notationally easier and more general.

One of the earliest methods for solving the minimax problem was that of Osborne and Watson [1]. At the kth iterate $\boldsymbol{x}_k$, their method uses a linear approximation of the nonlinear minimax problem, namely,

$$\underset{\boldsymbol{h}}{\text{minimize}}\, \bar{F}(\boldsymbol{x}_k, \boldsymbol{h}) \triangleq \max_j \left\{ f_j(\boldsymbol{x}_k) + \boldsymbol{f}_j'(\boldsymbol{x}_k)^T \boldsymbol{h} \right\} \quad (2)$$

where $\boldsymbol{f}_j'(\boldsymbol{x}_k)$ denotes the gradient vector of f_j w.r.t. $\boldsymbol{x}$ at the point $\boldsymbol{x}_k$. The minimizer $\boldsymbol{h}_k$ of (2) is found using linear programming and it is used in a line search. This method may be efficient but often it is inefficient. No convergence can be guaranteed and the method can even provide convergence to a nonstationary point. Madsen [2] incorporated trust regions in the Osborne and Watson method. The linearized problem (2) is solved subject to a local bound on the variable $\boldsymbol{h}$, the bound being adjusted during the iteration. No line search is used. This method has been proved to provide convergence to the set of stationary points and has a quadratic final rate of convergence when the solution is regular (Madsen and Schjaer-Jacobsen [3]). However, the rate of convergence may be very slow on singular problems.

The method of Anderson and Osborne [4] is very similar to that of Madsen. The main difference lies in the way of bounding the step length $\|\boldsymbol{h}_k\|$. A different approach was used by Bandler and Charalambous [5]. They presented an approach utilizing efficient unconstrained gradient minimization techniques in conjunction with least pth objective functions employing extremely large values of p. Charalambous and Conn [6] apply an active set strategy to obtain a direction for a line search.

All of these methods are first-order methods, i.e., the search is based on first-order derivatives only. Therefore, all of these methods have problems with singular solutions and the rate of convergence may be very slow.

C. Methods Using Second-Order Information

In order to overcome this problem, some second-order (or approximate second-order) information must be used. Hettich [7] was the first who proposed doing this. He used a Newton iteration for solving a set of equations which expresses the necessary condition for an optimum (see (6) below). However, Hettich's method is only local. It is required that the initial point is close to the solution and that the set of active functions (and constraints) is known. Han [8] suggested using nonlinear programming techniques for solving the minimax problem. He uses a nonlinear programming formulation of the minimax problem which is solved via successive quadratic programming (Powell [9]). A line search is incorporated using the minimax objective function as merit function. Overton [10] uses an approach similar to Han's but solves equality constrained quadratic problems and uses a specialized line search.

The method of Watson [11] is very similar to the method of this paper. It switches between a first- and a second-order method. The main differences between our and Watson's method are the following. Watson requires the user to provide exact first- and second-order derivatives, whereas we only require first-order derivatives. Furthermore, Watson fails to define a suitable set of criteria for switching between the first- and the second-order method. Our method has guaranteed convergence to the set of stationary points, whereas Watson's method has no such property. It can even provide convergence to a nonstationary point.

The algorithm of this paper is based on the work of Hald and Madsen [12]. It is a combination of the first-order method of Madsen [2] and an approximate second-order method. The first-order method provides fast convergence to the neighborhood of a solution. If this solution is singular, then the rate of convergence becomes very slow and a switch is made to the other method. Here a quasi-Newton method is used to solve a set of nonlinear equations that necessarily hold at a local solution of (1). This method has superlinear final convergence (see Appendix B). Several switches between the two methods may take place and the switching criteria ensure the global convergence of the combined method. Notice that the user of this algorithm is required to supply function values and first-order derivatives, whereas the necessary second derivative estimates are generated by the algorithm.

We show in Appendix A that, in the neighborhood of a local minimum of (1), our method generates the same points as the method of Powell [9] and Han [8]. However, in the latter method, a quadratic program must be solved in every iteration, whereas we have to solve only a set of linear equations or, if the solution is regular, a linear programming problem. Therefore, the computational effort used per iteration with our method is normally much smaller.

III. Description of the Algorithm

The algorithm is a combination of two methods denoted Method 1 and Method 2. Method 1 is intended to be used far away from a solution, whereas Method 2 is a local method. We first describe these two methods.

A. Method 1

This is essentially the algorithm of Madsen [2]. At the kth step, a feasible approximation $\boldsymbol{x}_k$ of a solution of (1) and a local bound Λ_k are given. In order to find a better estimate of a solution, the following linearized problem is solved:

$$\underset{\boldsymbol{h}}{\text{minimize}}\, \bar{F}(\boldsymbol{x}_k, \boldsymbol{h}) \triangleq \max_j \left\{ f_j(\boldsymbol{x}_k) + \boldsymbol{f}_j'(\boldsymbol{x}_k)^T \boldsymbol{h} \right\}$$

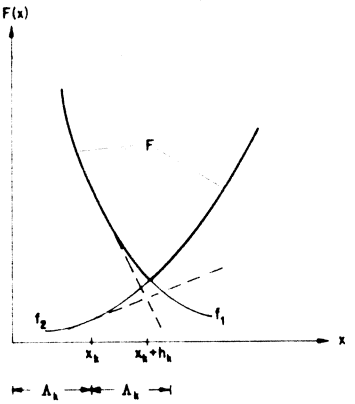


Fig. 1. An example with one variable and two functions illustrating a Method 1 iteration of the algorithm.

subject to

$$\|\boldsymbol{h}\|_\infty \leq \Lambda_k$$
$$\boldsymbol{a}_i^T(\boldsymbol{x}_k+\boldsymbol{h})+b_i=0, \qquad i=1,\cdots,l_{eq}$$
$$\boldsymbol{a}_i^T(\boldsymbol{x}_k+\boldsymbol{h})+b_i\geq 0, \qquad i=(l_{eq}+1),\cdots,l. \tag{3}$$

The solution of (3), denoted $\boldsymbol{h}_k$, is found by linear programming. Notice that $\boldsymbol{x}_k+\boldsymbol{h}_k$ is feasible. The next iterate is $\boldsymbol{x}_k+\boldsymbol{h}_k$ provided this point is better than $\boldsymbol{x}_k$ in the sense of F, i.e., if $F(\boldsymbol{x}_k+\boldsymbol{h}_k)<F(\boldsymbol{x}_k)$. Otherwise $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k$. In Fig. 1, an example with one variable, two functions, and no constraints ($l=0$) is shown. $F(\boldsymbol{x})$ is the kinked bold-faced curve. At $\boldsymbol{x}_k$, linear approximations of the two functions f_1 and f_2 are made and the solution of (3) is $\boldsymbol{h}_k$, which is found at the intersection of the two linear approximations. We assume that the local bound Λ_k is so large that it has no influence. The new point is $\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\boldsymbol{h}_k$, which is seen to be close to a local minimum of F.

The local bound Λ_k is introduced because the linear model (3) is a good approximation of (1) only in some neighborhood of $\boldsymbol{x}_k$. Therefore, it makes sense to consider only small values of $\|\boldsymbol{h}\|$ in connection with the linear model (3). The size of the bound is adjusted in every iteration based on a comparison between the decrease in the objective function F and the decrease predicted by the model (3). If the ratio between the two is small

$$F(\boldsymbol{x}_k)-F(\boldsymbol{x}_k+\boldsymbol{h}_k)\leq 0.25\left[\bar{F}(\boldsymbol{x}_k,\boldsymbol{0})-\bar{F}(\boldsymbol{x}_k,\boldsymbol{h}_k)\right] \tag{4}$$

then the bound is decreased, $\Lambda_{k+1}=\Lambda_k/4$. Otherwise, if

$$F(\boldsymbol{x}_k)-F(\boldsymbol{x}_k+\boldsymbol{h}_k)\geq 0.75\left[\bar{F}(\boldsymbol{x}_k,\boldsymbol{0})-\bar{F}(\boldsymbol{x}_k,\boldsymbol{h}_k)\right] \tag{5}$$

then $\Lambda_{k+1}=2\Lambda_k$. If neither (4) nor (5) hold, then we leave the bound unchanged, $\Lambda_{k+1}=\Lambda_k$.

Experiments have shown that the algorithm is rather insensitive to small changes in the constants used in the updating of the bound. This method has safe global convergence properties (Madsen [2]), and if the solution is regular, then the final rate of convergence is quadratic (Madsen and Schjaer-Jacobsen [13]).

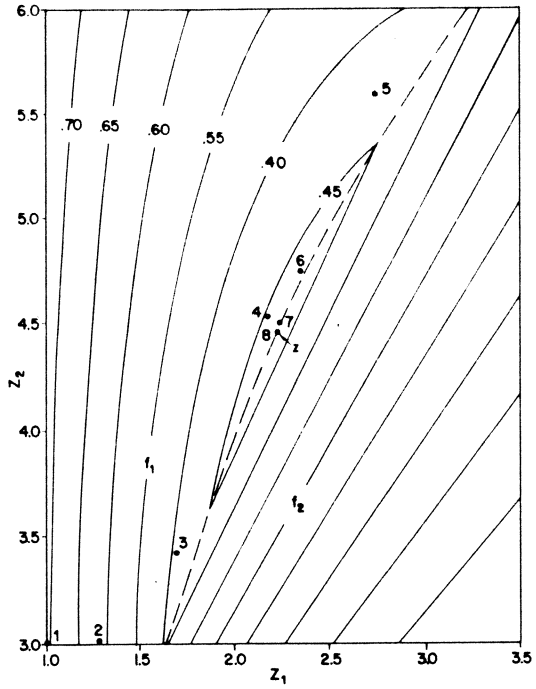


Fig. 2. Two-dimensional singular minimax problem arising from optimization of a two-section 10:1 transmission-line transformer with optimization parameters Z_1 and Z_2. The first 6 iterations are performed using Method 1 of the algorithm. Iterations 7 and 8 are performed using Method 2. The total number of iterations (function evaluations) to reach the solution with the accuracy of 10^{-6} is 11 (0.49 s on Cyber 170/815). If Method 2 is not used, 25 iterations (1.14 s of CPU time) are required to reach the solution.

When the solution is singular, however, the final convergence can be very slow. Consider the example of Fig. 2 in two variables where two functions are active at the solution $\boldsymbol{z}$(i.e., $f_j(\boldsymbol{z})=F(\boldsymbol{z})$ for two values of j). Fig. 2 shows contours for a two-section transmission-line transformer problem, where the minimax functions correspond to the reflection coefficient sampled at 11 normalized frequencies w.r.t. to 1 GHz $\{0.5,0.6,\cdots,1.4,1.5\}$. The optimization variables are characteristic impedances Z_1 and Z_2. Section lengths l_1 and l_2 are kept constant at their optimal value l_q, which is the quarter wavelength at the center frequency. According to Madsen and Schjaer-Jacobsen [3], this is a singular problem. Above the dotted line, F is equal to one of the functions f_j, $F(\boldsymbol{x})=f_1(\boldsymbol{x})$, and below the dotted line, F is equal to another function, $F(\boldsymbol{x})=f_2(\boldsymbol{x})$. At the dotted line, $f_1(\boldsymbol{x})=f_2(\boldsymbol{x})=F(\boldsymbol{x})$, and this line represents the bottom of a valley.

If f_1 and f_2 are different, then there is a kink at the bottom of the valley and a method based on linearization, such as Method 1, will provide fast convergence to this kink, as illustrated by the iterands in the figure (see point number 3). After the dotted line has been reached, however, the convergence towards $\boldsymbol{z}$ can be slow because the iterands have to follow a curve which passes the solution $\boldsymbol{z}$ in a smooth manner (with no kink). In the example, over eight iterations are needed to converge to a region close to the solution. Therefore, a method based on first derivatives only is not sufficient, in general, to give fast convergence. Some (approximate) second-order information is needed.

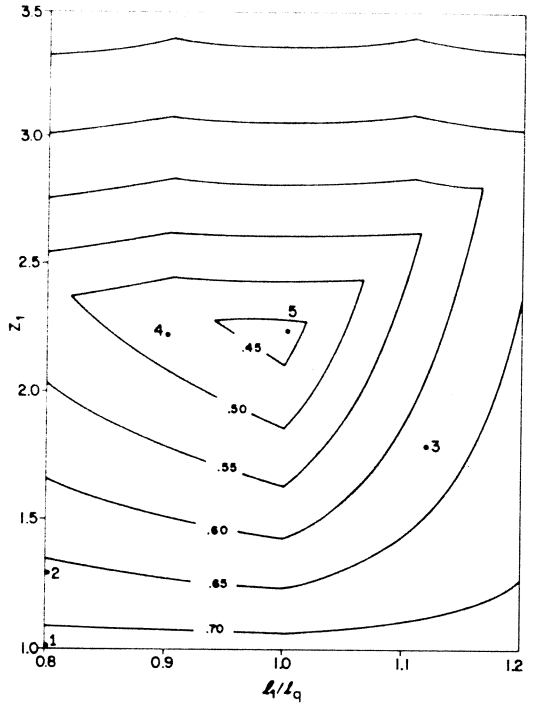


Fig. 3. Two-dimensional regular minimax problem arising from optimization of a two-section 10:1 transmission-line transformer with optimization parameters l_1/l_q and Z_1. The first 5 iterations shown are performed using Method 1. The total number of iterations to reach the solution with the accuracy of 10^{-6} is 8 (0.37 s of CPU time on Cyber 170/815).

Notice that if three functions were equal at a minimum of a two-dimensional problem, then no smooth curve through the solution exists and Method 1 provides fast (quadratic) convergence to the solution.

Fig. 3 shows contours for the same two-section transformer problem. However, the optimization variables are now l_1/l_q and Z_1. Characteristic impedance Z_2 and section length l_2/l_q are kept at their optimal values ($l_2/l_q = 1$, $Z_2 = 4.472136$). Here, the problem is regular and five iterations are sufficient to reach the vicinity of the solution.

B. Method 2

It is a local method. It is assumed that a point near a solution z is known, and that the active sets $A(z) \triangleq \{j | f_j(z) = F(z)\}$ and $C(z) \triangleq \{i | \boldsymbol{a}_i^T z + b_i = 0\}$ are known. At a local minimum z of (1), the following necessary conditions hold (see, e.g., [7]):

$$\begin{aligned} \sum_{j \in A(z)} \lambda_j \boldsymbol{f}_j'(z) - \sum_{i \in C(z)} \mu_i \boldsymbol{a}_i &= \boldsymbol{0} \\ \sum_{j \in A(z)} \lambda_j - 1 &= 0 \\ f_{j_0}(z) - f_j(z) &= 0, \qquad j \in A(z) \backslash \{j_0\} \\ \boldsymbol{a}_i^T z + b_i &= 0, \qquad i \in C(z), \end{aligned} \tag{6}$$

where the multipliers λ_j and μ_i are nonnegative and $j_0 \in A(z)$ is fixed. Method 2 is an approximate Newton method for solving the nonlinear system (6) (in the variables (z, λ, μ)). Exact first derivatives are used but the matrix $\Sigma \lambda_i f_j''(z)$ is approximated using a modified Broyden-Fletcher-Goldfarb-Shanno (BFGS) update (see Appendix A for details). In this way, an approximate Jacobian $\boldsymbol{J}_k$ is obtained at the estimate $(x_k, \lambda^{(k)}, \mu^{(k)})$ of the solution of (6). The next estimate is found by

$$\boldsymbol{J}_k \begin{bmatrix} \Delta x_k \\ \Delta \lambda^{(k)} \\ \Delta \mu^{(k)} \end{bmatrix} = -\boldsymbol{R}\left(x_k, \lambda^{(k)}, \mu^{(k)}\right)$$

$$\left(x_{k+1}, \lambda^{(k+1)}\right), \mu^{(k+1)}\right) = \left(x_k, \lambda^{(k)}, \mu^{(k)}\right) + \left(\Delta x_k, \Delta \lambda^{(k)}, \Delta \mu^{(k)}\right) \tag{7}$$

where $\boldsymbol{R}(z, \lambda, \mu) = \boldsymbol{0}$ is the vector formulation of (6).

C. The Combined Method

The combined method is the algorithm which we recommend and use in this paper. Initially, Method 1 is used and the active sets used in (6) are estimated. When a singular local minimum seems to be approached, a switch to Method 2 is made. If the Method 2 iteration is unsuccessful, Method 1 is used again. Several switches between the two methods may take place. When Method 1 is used, we say that the iteration is in Stage 1, otherwise it is in Stage 2. A detailed description of the two stages follows.

The Stage 1 Iteration: We have a point x_k, a local bound Λ_k, and a matrix $\boldsymbol{J}_k$ which should approximate the Jacobian of (6).

1) x_{k+1} and Λ_{k+1} are found using Method 1, and approximations A_{k+1} and C_{k+1} of the active sets at x_{k+1} are found via the active sets at the solution $\boldsymbol{h}_k$ of the linear model problem (3).

2) An estimate $(\lambda^{(k+1)}, \mu^{(k+1)})$ of the multipliers is found through a least-squares solution of (6) with $(x_{k+1}, A_{k+1}, C_{k+1})$ inserted for $(z, A(z), C(z))$. This estimate is used for finding a new Jacobian estimate $\boldsymbol{J}_{k+1}$ by the BFGS update as described in Appendix A.

3) A switch to Stage 2 is made if the following two conditions hold:

a) The active set estimates have been constant over three consecutive different Stage 1 iterates.
b) The components of $\lambda^{(k+1)}$ and $\mu^{(k+1)}$ are nonnegative.

The Stage 2 Iteration: $x_k, \Lambda_k, \boldsymbol{J}_k$, and active set estimates A_k, C_k are given.

1) Find $(x_{k+1}, \lambda^{(k+1)}, \mu^{(k+1)})$ and $\boldsymbol{J}_{k+1}$ using Method 2 with (A_k, C_k) inserted for $(A(z), C(z))$.

2) Let $A_{k+1} = A_k$, $C_{k+1} = C_k$, and $\Lambda_{k+1} = \Lambda_k$.

3) Switch to Stage 1 if one of the following conditions holds:

a) A function or constraint outside of A_{k+1} or C_{k+1} is active at x_{k+1}.
b) A component of $\lambda^{(k+1)}$ or $\mu^{(k+1)}$ is negative.
c) $\|\boldsymbol{R}(x_{k+1}, \lambda^{(k+1)}, \mu^{(k+1)})\| > 0.999 \|\boldsymbol{R}(x_k, \lambda^{(k)}, \mu^{(k)}\|$ (see (7) for the definition of $\boldsymbol{R}$).

TABLE I
COMPARISON OF ALGORITHMS FOR THE THREE-SECTION TRANSFORMER (NUMBER OF FUNCTION EVALUATIONS)

Algorithm		Starting Point x_0^1	Starting Point x_0^2
This algorithm[1]		18 *	21 **
Hald and Madsen [12]		21	46
Agnew [19]	Alg. I	23	64
	Alg. II	20	109
Bandler and Charalambous [20]		95	155
Charalambous and Conn [21]		162	67
Conn [22]		67	80
Madsen [2]		253	707
Madsen and Schjaer-Jacobsen [3]		29	48
This algorithm[2]		15 +	22 ++

Execution times on Cyber 170/815 in seconds are * 0.6, ** 0.7, + 0.57, ++ 0.85

"Active" frequency points selected by the cubic interpolation search
0.50000, 0.76999, 1.23001, 1.50000

[1] without cubic interpolation
[2] with cubic interpolation

This completes the description of the combined method.

It has been shown [12] that the combined method can only converge to stationary points and that the final rate of convergence is quadratic on regular problems and superlinear on singular problems (provided that the Jacobian of (6) is regular).

The results published by Hald and Madsen [12] correspond to the combined method as described here except that the PSB (Powell's symmetric Broyden) formula was used for updating J_k in Method 2. Our numerical results indicate that the use of the BFGS formula as described in Appendix A is significantly better (see Table I).

For this paper, we have used the MMLC version [14] of the algorithm based on the earlier implementation due to Hald [15].

IV. Comparison with Other Algorithms

A. The Test Problem

To compare the performance of the present algorithm to other minimax algorithms, a three-section 100-percent relative bandwidth 10:1 transmission-line transformer problem has been chosen (see Fig. 4). It is a special case of an N-section transmission-line transformer. Originally studied by Bandler and Macdonald [16], [17], this type of test problem is now widely considered.

The problem is to minimize the maximum reflection coefficient of this matching network. A detailed discussion on the formulation of direct minimax response objectives is presented in [18].

Formally, the problem is to

$$\underset{x}{\text{minimize}}\, F(x) = \max_{[0.5,1.5]} |\rho(x,\omega)| \tag{8}$$

where

$$x = [l_1/l_q \quad Z_1 \quad l_2/l_q \quad Z_2 \quad l_3/l_q \quad Z_3]^T.$$

The minimax functions represent the modulus of the reflection coefficient sampled at the 11 normalized frequencies ω (w.r.t. 1 GHz) {0.5, 0.6, 0.7, 0.77, 0.9, 1.0, 1.1, 1.23, 1.3, 1.4, 1.5}. The known quarter-wave solution is given by $l_1 = l_2 = l_3 = l_q$, $Z_1 = 1.63471$, $Z_2 = 3.16228$, $Z_3 = 6.11729$, where l_q is the quarter wavelength at the center frequency, namely,

$$l_q = 7.49481 \text{ cm for 1 GHz.}$$

The corresponding maximum reflection coefficient is 0.19729. Two starting points have been used

$$x_0^1 = [0.8 \quad 1.5 \quad 1.2 \quad 3.0 \quad 0.8 \quad 6.0]^T$$

$$x_0^2 = [1.0 \quad 1.0 \quad 1.0 \quad 3.16228 \quad 1.0 \quad 10.0]^T.$$

Gradient vectors with respect to section lengths and characteristic impedances are obtained using the adjoint network method.

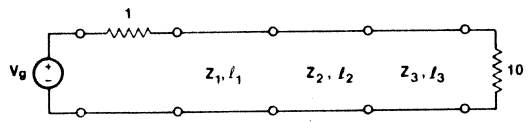


Fig. 4. Three-section, 10:1 transmission-line transformer used as a test problem to compare the performance of minimax algorithms.

B. Performance of the New Algorithm

Table I shows the performance of the new algorithm as compared to other algorithms. Table I also shows results obtained using the present algorithm with a cubic interpolation search for maxima of the response. Using this technique, the number of sample points can be reduced from 11 to 4, and we do not have to know in advance the location of frequency points corresponding to the maxima of the response. More information on the cubic interpolation search technique is given in Section V in the context of a new approach to worst-case design of microwave circuits.

To show the influence of the parameters *DX* (initial step length of the iterative algorithm) and *KEQS* (the number of successive iterations with identical sets of active residual functions that is required before a switch to Stage 2 is made), the optimization has been performed several times for different values of *DX* and *KEQS*. The resulting numbers of residual function evaluations required to achieve the accuracy $EPS = 10^{-6}$, as well as the number shifts to Stage 2 are summarized in Table II (the numbers of shifts are given in parentheses).

It can be observed that the increasing values of *KEQS* correspond to slightly increased numbers of function evaluations. Moreover, too small and too large values of *DX* require more residual function evaluations because of adjustments which are performed by the algorithm. From other experiments, it was observed that the increasing values of *KEQS* correspond, generally, to smaller numbers of shifts to Stage 2 (some too early shifts are eliminated).

V. Worst-Case Network Design

A. Preliminary Remarks

In this section, we will formulate the fixed tolerance problem (FTP) [23], [24] embodying a worst-case search and the selection of sample points for the discrete ap-

TABLE II
THE INFLUENCE OF THE CONTROLLING PARAMETERS DX AND KEQS ON THE NUMBER OF FUNCTION EVALUATIONS

	KEQS		
DX	2	3	4
0.1	21(2)	23(2)	24(2)
0.25	19(2)	18(2)	19(2)
0.5	18(2)	20(2)	22(2)
0.75	18(2)	18(2)	20(2)
1.0	21(2)	22(2)	23(2)

proximation of a continuous problem. As mentioned in the introduction, the discretization of a continuous problem may result in a large number of minimax functions to be minimized. The size of the problem increases dramatically if we want to consider tolerances on network parameters since, for each frequency point selected to represent the response 2^n (n is the number of network parameters), minimax functions have to be created if we want to consider all vertices of the tolerance region.

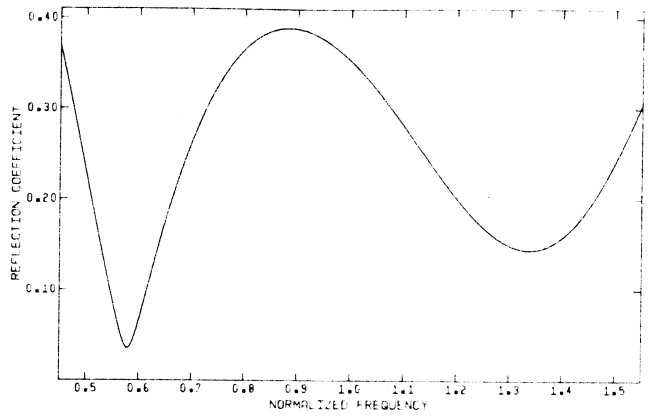
A number of methods have been proposed for solving the worst-case problem. Schjaer-Jacobsen and Madsen [24] suggest the application of interval arithmetic. Bandler *et al.* [25] and Tromp [26] described methods which rely on the assumption that the functions considered are one-dimensionally convex.

Our approach to the fixed tolerance problem is a double iterative algorithm. For each outer iteration of minimization, first a search using cubic interpolation is done to determine frequency points which are candidates for active functions, and then a number (equal to the number of selected minimax functions) of inner loop optimizations are performed to determine the worst case for each of the minimax functions.

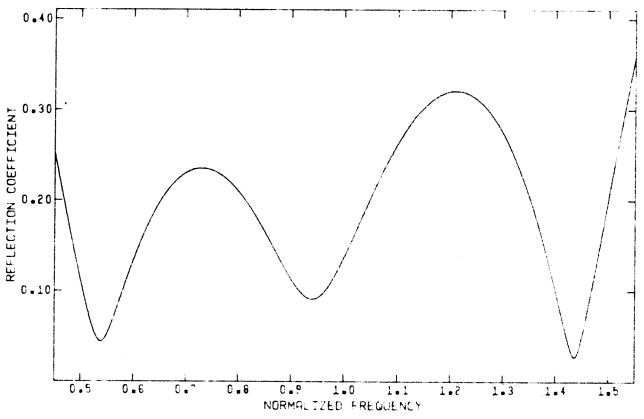
The advantage of our approach is that the worst-case search (done by means of linearly constrained optimization) and the actual minimization are linked together such that each worst-case calculation affects immediately the outer iteration of minimization.

B. Cubic Interpolation Search Technique

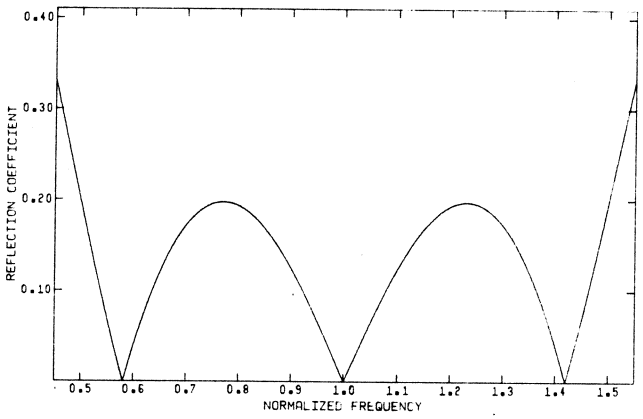
The cubic interpolation technique allows us to consider the minimum number of frequency points to adequately approximate the continuous problem. In many cases, the discretization of a continuous problem may not be adequate to give the continuous minimax solution. As illustrated in Fig. 5, the solution obtained using uniformly spaced sample points may not be optimal in the continuous minimax sense since some of the peaks of the response (or error function) have been missed. One way to overcome this difficulty is to use densely spaced sample points. This, however, may result in a prohibitively large number of minimax functions to be optimized. Therefore, it is desirable to develop a technique to locate the maxima of the response w.r.t. frequency and to track these maxima during the optimization process as they shift along the frequency



(a)



(b)



(c)

Fig. 5. (a) Response of the three-section transmission-line transformer at the starting point x_1^0. The initial sample points are 0.5, 0.8, 1.2, and 1.5. The uniformly spaced (with the step length of 0.1) search points are 0.5, 0.6,...,1.5. The sample points selected by the algorithm using cubic interpolation are 0.5, 0.884, 1.2, and 1.5. The edges of the frequency interval are kept as the sample points. The initial sample point 0.8 has been replaced by the point 0.884 since a maximum has been detected between 0.8 and 0.9. The sample point 1.2 has not been changed since no maximum has been found between points 1.2 and 1.3. (b) Response of the three-section transmission-line transformer after the first iteration of optimization. The sample points selected by the cubic interpolation search (with the step length of 0.1) are 0.5, 0.729, 1.210, and 1.5. Now the initial sample point 1.2 has been replaced by 1.210 since a maximum has been detected between points 1.2 and 1.3. The sample point 0.884 has been replaced by a new sample point 0.729 resulting from the maximum between 0.7 and 0.8. (c) Response of the three-section transformer at the solution. The sample points selected by the cubic interpolation search (with the step length of 0.1) before the last iteration of optimization are 0.5, 0.770, 1.230, and 1.5.

axis due to the changes in the values of optimization parameters. Such a technique has been developed by Bandler and Chen [27]. It is based on the cubic interpolation formulas of Fletcher and Powell [28]. For convenient reference, the formulas are given in Appendix C.

C. Fixed Tolerance Problem

To present the method for worst-case tolerance design, we will introduce some notation which basically follows that of [23].

A design consists of design data of the nominal point $\phi^0 \triangleq [\phi_1^0 \quad \phi_2^0 \cdots \phi_n^0]^T$ and a set of associated tolerances $\epsilon \triangleq [\epsilon_1 \quad \epsilon_2 \cdots \epsilon_n]^T$, where n is the number of network parameters. An outcome of a circuit is any point $\phi \triangleq [\phi_1 \; \phi_2 \cdots \phi_n]^T$ such that $\phi_i = \phi_i^0 + \epsilon_i \mu_i$, $-1 \leq \mu_i \leq 1$, $i = 1,2,\cdots,n$. The tolerance region R_ϵ is defined as $R_\epsilon \triangleq \{\phi | \phi_i = \phi_i^0 + \epsilon_i \mu_i,\ i = 1,2,\cdots,n\}$. The extreme points of R_ϵ are called the vertices and are obtained by setting $\mu_i = \pm 1$.

We consider a set of m nonlinear functions

$$f_j(\phi^0) \triangleq f(\phi^0, \omega_j), \qquad j \in J \triangleq \{1,2,\cdots,m\} \tag{9}$$

where ω_j, $j \in J$, is an independent parameter (frequency). The number of functions m is equal to

$$m = m_{\max} + 2$$

where $m_{\max}$ is the number of the maxima of the response and 2 represents the edges of the frequency interval $[\omega_l, \omega_u]$.

The fixed tolerance problem can be defined on the basis of the worst-case objective function [24] as that of determining

$$\min_{\phi^0} F(\phi^0) = \min_{\phi^0} \max_{j \in J} \left\{ \max_{\phi \in R_\epsilon} f_j(\phi) \right\}. \tag{10}$$

For each outer iteration of minimization w.r.t. ϕ^0, m frequency points are determined (by a search technique based on cubic interpolation) and m linearly constrained optimizations are performed to find the worst cases.

At the kth outer iteration of minimization, we have an approximation ϕ_k^0 of the solution and we solve m linearly constrained optimizations, where the jth problem, $j \in J$ is

$$\underset{\phi_k}{\text{minimize}} \left(-f_j(\phi_k) \right) \tag{11}$$

subject to

$$(\phi_i^0)_k - \epsilon_i \leq (\phi_i)_k \leq (\phi_i^0)_k + \epsilon_i, \qquad i = 1,2,\cdots,n.$$

Once ϕ_k^* for the jth function is determined, we can identify whether the worst-case occurred at a vertex using the following criteria.

Let

$$(y_i)_k = \left| (\phi_i^0)_k - (\phi_i^*)_k \right|. \tag{12}$$

If $|(y_i)_k - \epsilon_i| \leq 10^{-5}$, then the worst-case occurred at a vertex, for which μ_i, $i = 1,2,\cdots,n$, are easy to determine

$$\mu_i = \begin{cases} -1, & \text{if} \quad (\phi_i^*)_k \leq (\phi_i^0)_k \\ +1, & \text{otherwise} \end{cases}. \tag{13}$$

TABLE III
FIXED TOLERANCE PROBLEM FOR THE THREE-SECTION 10:1 TRANSFORMER

Number of Minimax Functions	4	
Number of Variables	6	
Required Accuracy of the Solution	10^{-5}	
Assumed Tolerances	5%	
Step Size in the Cubic Interpolation Search	0.1	
Solution Vector	$\ell_1/\ell_q = 0.96373$	$Z_2 = 3.22493$
	$Z_1 = 1.67797$	$\ell_3/\ell_q = 0.96483$
	$\ell_2/\ell_q = 0.98720$	$Z_3 = 6.04817$
"Active" Frequency Points	0.50000, 0.78726, 1.27242, 1.50000	
Maximum Refl. Coefficient	0.33589	
Number of Function Evaluations	32	
Execution Time on Cyber 170/815 (in seconds)	8.1	

The function values f_j, $j \in J$, and the gradients of f_j, $j \in J$, which are returned to the outer iteration are evaluated at a point $(\phi_j^*)_k$, i.e., were the jth worst case occurred.

D. Illustration of the Approach

The three-section transmission-line transformer is used to illustrate the approach and its validity for worst-case design. Numerical results are summarized in Table III. As expected, the nominal parameter values are different from the values obtained for the nominal design problem. The location of the two internal maxima of the response has also changed as compared to the nominal design problem. Each linearly constrained optimization to determine the worst-case for the particular frequency with the accuracy 10^{-3} requires about four iterations of the algorithm.

VI. Contiguous and Noncontiguous-Band Multiplexer Design

A. Introductory Remarks

Practical design and manufacture of contiguous- and noncontiguous-band multiplexers consisting of multicavity filters distributed along a waveguide manifold has been a problem of significant interest [29]–[31]. Recently, a general multiplexer design procedure using an extension of the normal least-squares method has been described [32].

We present here a general multiplexer optimization procedure exploiting exact network sensitivities. The simulation and sensitivity analysis aspects of the problem, together with a number of examples of multiplexer optimization (including a twelve-channel, 12-GHz multiplexer without dummy channels), have been described in [33]. All design parameters of interest, e.g., waveguide spacings, input–output, and filter coupling parameters, can be directly optimized. A typical structure under consideration is shown in Fig. 6.

B. Formulation of the Problem

A wide range of possible multiplexer optimization problems can be formulated and solved by appropriately defining specifications on common port return loss and individual channel insertion loss functions. The minimax error functions are created using those specifications, simulated exact multiplexer responses, and weighting factors.

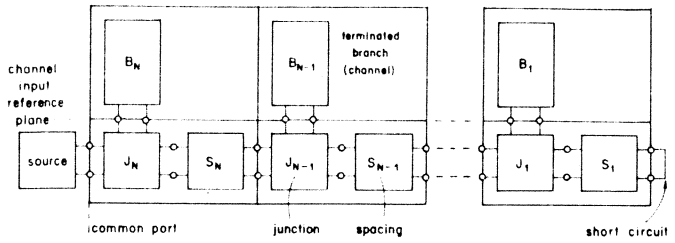


Fig. 6. The multiplexer configuration under consideration. $J_1, J_2, \cdots, J_N$ are arbitrarily defined 3-port junctions, $B_1, B_2, \cdots, B_N$ are terminated branches or channels which may each be represented in reduced cascade form, and $S_1, S_2, \cdots, S_N$ are usually waveguide spacing elements.

The minimax functions $f_j(\boldsymbol{x})$, $j \in J$, are of the form

$$w^1_{Uk}(\omega_i)\left(F^1_k(\boldsymbol{x},\omega_i) - S^1_{Uk}(\omega_i)\right) \tag{14}$$

$$-w^1_{Lk}(\omega_i)\left(F^1_k(\boldsymbol{x},\omega_i) - S^1_{Lk}(\omega_i)\right) \tag{15}$$

$$w^2_U(\omega_i)\left(F^2(\boldsymbol{x},\omega_i) - S^2_U(\omega_i)\right) \tag{16}$$

$$-w^2_L(\omega_i)\left(F^2(\boldsymbol{x},\omega_i) - S^2_L(\omega_i)\right) \tag{17}$$

where $F^1_k(\boldsymbol{x},\omega_i)$ is the insertion loss for the kth channel at the ith frequency, $F^2(\boldsymbol{x},\omega_i)$ is the return loss at the common port at the ith frequency, $S^1_{Uk}(\omega_i)(S^1_{Lk}(\omega_i))$ is the upper (lower) specification on insertion loss of the kth channel at the ith frequency, $S^2_U(\omega_i)(S^2_L(\omega_i))$ is the upper (lower) specification on return loss at the ith frequency, and $w^1_{Uk}, w^1_{Lk}, w^2_U, w^2_L$ are the arbitrary user-chosen nonnegative weighting factors.

C. Five-Channel 11-GHz Multiplexer Design

The procedure is illustrated by designing an 11-GHz, five-channel multiplexer having the center frequencies and bandwidths (similar to those of [32]) given in Table IV.

Suppose we want to design this multiplexer such that certain specifications on the common port return loss and individual channel insertion loss functions are satisfied. A lower specification of 20 dB on return loss over the passbands of all five channels should be satisfied. We want also to control return loss between channels 1 and 2, 2 and 3, and 4 and 5 in a similar way. We impose also additional specifications on insertion loss for all channels, i.e., we want the insertion loss in the transition bands not to drop below 20 dB.

We start the design process with five identical six-pole, pseudo-elliptic function filters. Starting values of the coupling coefficients for the filters are given in the following matrix [31]:

$$M = \begin{bmatrix} 0 & 0.62575 & 0 & 0 & 0 & 0 \\ 0.62575 & 0 & 0.57615 & 0 & 0 & 0 \\ 0 & 0.57615 & 0 & 0.32348 & 0 & -0.74957 \\ 0 & 0 & 0.32348 & 0 & 1.04102 & 0 \\ 0 & 0 & 0 & 1.04102 & 0 & 1.04239 \\ 0 & 0 & -0.74957 & 0 & 1.04239 & 0 \end{bmatrix}.$$

The initial spacing lengths are set equal to $\lambda_{gk}/2$ (half the wavelength corresponding to the kth center frequency).

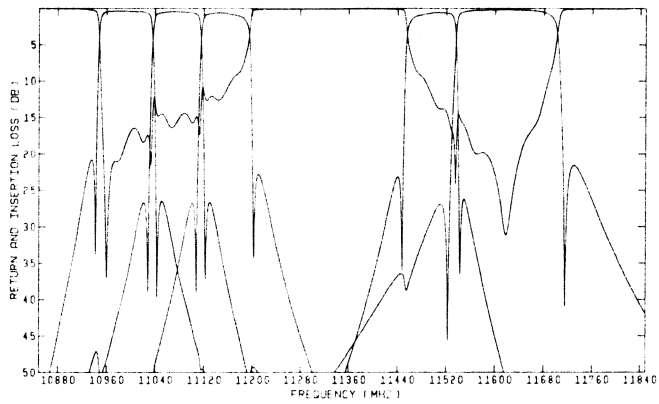


Fig. 7. Responses of the 5-channel, 11-GHz multiplexer at the starting point of the optimization process.

TABLE IV
MULTIPLEXER CENTER FREQUENCIES AND BANDWIDTHS

Channel	Center Frequency (MHz)	Bandwidth (MHz)
1	10992.5	81
2	11075.0	76
3	11155.0	76
4	11495.0	76
5	11618.5	154

The filters are assumed lossy and dispersive. Waveguide junctions are assumed nonideal.

Fig. 7 shows the responses of the multiplexer at the start of the optimization process. As we see, the specifications on the common-port return loss are seriously violated.

The optimization process is performed in several steps. First, we select only nonzero couplings, input/output transformer ratios, and filter spacings as optimization variables. This gives a total of 45 optimization variables. The error functions resulting from the multiplexer responses and specifications are created at 51 nonuniformly spaced frequency points. An improved design is obtained after 30 function evaluations (230 s on the Cyber 170/815). The responses corresponding to the first step of the optimization process are shown in Fig. 8.

In order to completely satisfy the design specifications we perform a second step of optimization in which we release additional optimization variables, i.e., cavity resonances. This gives a total of 75 nonlinear optimization variables. Using the same frequency points as in step 1 and results of the first optimization as a starting point, we continue the optimization process. After 30 additional

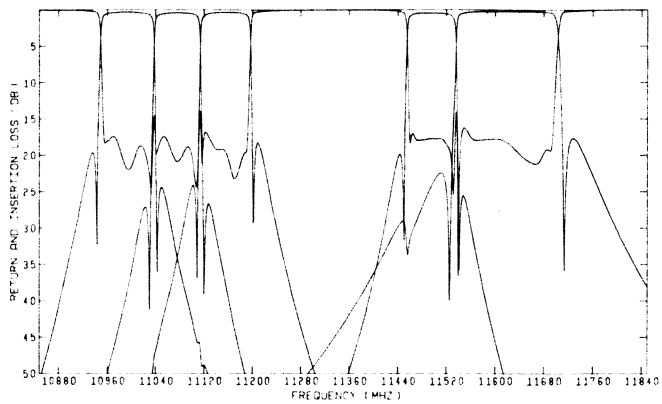


Fig. 8. Responses of the 5-channel, 11-GHz multiplexer after the first 30 iterations using 45 optimization variables and 51 nonuniformly spaced sample points.

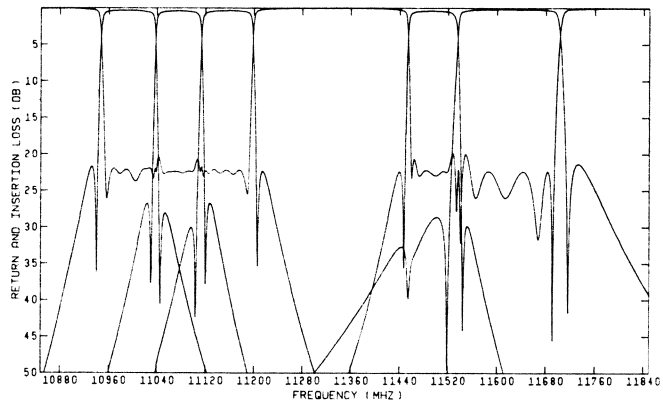


Fig. 9. Responses of the 5-channel, 11-GHz multiplexer after 30 additional iterations using 75 optimization variables and 51 minimax functions.

function evaluations (and 470 s of CPU time on the Cyber 170/815), the design specifications are satisfied and the optimized responses of the five-channel multiplexer are shown in Fig. 9. To improve the return loss response of the multiplexer, the third step of optimization is performed in which a search technique for maxima of the response is employed. This gives 66 minimax functions and the same number of variables as previously. After 25 additional function evaluations (and 360 s of CPU time on the Cyber 170/815), we obtain the final optimized responses as shown in Fig. 10.

In the approach presented, the emphasis is on achieving a maximally effective set of early iterations of optimization using a subset of all possible optimization variables. This subset should correspond to "dominant" variables of the problem. Initial selection of the variables can be facilitated by the full knowledge and experience of the designer and by an initial sensitivity analysis at selected frequency points.

VII. Conclusions

We have described a new and highly efficient algorithm for nonlinear minimax optimization problems which arise in microwave circuit design. The algorithm combines linear programming methods with quasi-Newton methods and the convergence is at least superlinear. Comparison made

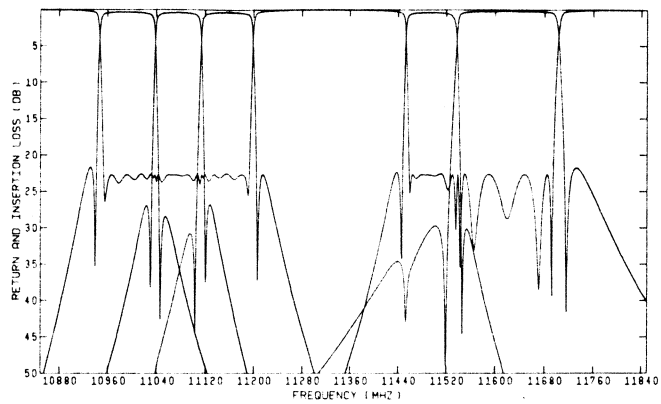


Fig. 10. The final optimized responses of the 5-channel multiplexer obtained with 75 variables and 66 minimax functions. A search technique for the peaks of the return loss response has been employed.

with the existing minimax algorithms on the classical three section transmission-line transformer problem shows clearly that this algorithm is better in terms of the number of function evaluations required to reach the solution with a desired accuracy.

We have presented a novel approach to worst-case tolerance design of microwave circuits integrating a cubic interpolation based search technique for maxima of the response with the worst-case search using linearly constrained optimization. The validity of the approach has been demonstrated by solving a fixed tolerance problem for a three-section transmission-line transformer. We emphasize that our approach does not require the designer to know in advance the location of frequency points corresponding to the maxima of the response and significantly reduces the number of sample points adequately approximating the continuous response. This aspect of our approach is particularly important since it can significantly reduce the number of minimax functions for which the worst cases have to be found.

The robustness of the algorithm presented makes possible the practical design of contiguous- and noncontiguous-band microwave multiplexers. To our knowledge, our work is the first successful attempt to use gradient-based optimization for multiplexer design, as well as being the largest nonlinear optimization process ever demonstrated on microwave circuit design for a reasonable computational cost.

We feel that the algorithm presented will have a significant impact on microwave circuit design techniques and practices allowing the designer to consider problems of greater size than usually done in the past, including tolerances on circuit parameters.

Appendix A
Quasi-Newton Iteration and the Powell Method

Here the details of the approximate Newton iteration used in Method 2 are given. Furthermore, it is shown that close to a local minimum, Method 2 generates the same points as Powell's sequential quadratic programming method [9] applied to the nonlinear programming formula-

tion of (1). Therefore, our method has the same local convergence properties as that of Powell.

We consider one iteration of Method 2. For simplicity, we use the notation $x = x_k$, $\lambda = \lambda_k$, $\mu = \mu_k$, $A = A(z)$, and $C = C(z)$. In a Newton iteration for solving (6), we should use the Jacobian

$$R'(x,\lambda,\mu) = \begin{bmatrix} \sum_{j\in A} \lambda_j f_j''(x) & E & -F \\ 0\,0\cdots 0 & 1\,1\cdots 1 & 0\,0\cdots 0 \\ G^T & 0 & 0 \\ F^T & 0 & 0 \end{bmatrix} \tag{A1}$$

where E has the columns $f_j'(x)$, $j \in A$, F has the columns a_i, $i \in C$, and G has the columns $f_{j_0}'(x) - f_j'(x)$, $j \in A \setminus \{j_0\}$. Only the upper left-hand block involves more than first derivatives. In Method 2, this block is approximated by an updating formula, whereas the exact values are used in the other blocks of R'.

The Lagrangian function corresponding to (1) is

$$L(x,\lambda,\mu) = \sum_{j=1}^{m} \lambda_j f_j(x) - \sum_{i=1}^{l} \mu_i \left[a_i^T x + b_i\right] \tag{A2}$$

so the upper left-hand block of (A1) is $L_{xx}''(x,\lambda,\mu)$ since $\lambda_j = 0$ for $j \notin A$.

This block is approximated by the BFGS formula with the modifications of Powell [9] that keep the approximation positive definite. Thus, the matrix J_k of (7) is

$$J_k = \begin{bmatrix} B_k & E & -F \\ 0\,0\cdots 0 & 1\,1\cdots 1 & 0\cdots 0 \\ G^T & 0 & 0 \\ F^T & 0 & 0 \end{bmatrix} \tag{A3}$$

where B_k is updated through

$$B_{k+1} = B_k - B_k s s^T B_k / \left[s^T B_k s\right] + y y^T / \left[s^T y\right]$$

with

$$s = x_{k+1} - x$$
$$y = L_x'(x_{k+1},\lambda,\mu) - L_x'(x,\lambda,\mu). \tag{A4}$$

An iteration of Method 2 is now given by (A3), (7), and (A4) with

$$R(x,\lambda,\mu) = \begin{bmatrix} L_x'(x,\lambda,\mu) \\ \sum \lambda_j - 1 \\ e \\ f \end{bmatrix} \tag{A5}$$

where e has the components $f_{j_0}(x) - f_j(x)$, $j \in A \setminus \{j_0\}$ and f has the components $a_i^T x + b_i$, $i \in C$.

Now consider the nonlinear programming formulation of problem (1)

$$\underset{(x,\delta)}{\text{minimize}}\; G(x,\delta) \triangleq \delta$$

subject to

$$\begin{aligned} \delta - f_j(x) &\geq 0, && j = 1,\cdots,m \\ a_i^T x + b_i &= 0, && i = 1,\cdots,l_{\text{eq}} \\ a_i^T x + b_i &\geq 0, && i = (l_{\text{eq}}+1),\cdots,l. \end{aligned} \tag{A6}$$

Assume that Powell's sequential quadratic programming method is used to solve (A6). At the iterate $(x,\delta) = (x_k, \delta_k)$, the following subproblem is solved:

$$\underset{(h,p)}{\text{minimize}}\; Q(h,p) = p + h^T \bar{B}_k h$$

subject to

$$\begin{aligned} \delta + p - \left[f_j(x) + f_j'(x)^T h\right] &\geq 0, && j = 1,\cdots,m \\ a_i^T(x+h) + b_i &= 0, && i = 1,\cdots,l_{\text{eq}} \\ a_i^T(x+h) + b_i &\geq 0, && i = (l_{\text{eq}}+1),\cdots,l \end{aligned} \tag{A7}$$

with $\bar{B}_k$ being a positive definite estimate of the Hessian of the Lagrangian. $\bar{B}_k$ has the dimensions $n \times n$. Actually, it should be $(n+1)(n+1)$, but the row and column corresponding to p are left out for notational convenience since they have no influence. The next iterate is $(x_{k+1}, \delta_{k+1}) = (x,\delta) + \alpha(h,p)$, where (h,p) denote the solution of (A7), and close to a solution $\alpha = 1$ is necessary for fast convergence. We assume that (x,δ) is close to a solution (z,δ^*) and that $\alpha = 1$. Assume further that x is so close to z that the active constraints at the solution of (A7) are the same as at the solution of (A6). These are identified by the indices $j \in A = A(z)$ and $i \in C = C(z)$.

Using these assumptions, we can find the solution of (A7) using the Kuhn–Tucker conditions. They give

$$\begin{bmatrix} \bar{B}_k h \\ 1 \end{bmatrix} - \sum_{j\in A} \bar{\lambda}_j \begin{bmatrix} -f_j'(x) \\ 1 \end{bmatrix} - \sum_{i\in C} \bar{\mu}_i \begin{bmatrix} a_i \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{aligned} p - \left[f_j(x) + f_j'(x)^T h\right] &= 0, && j \in A, \\ a_i^T(x+h) + b_i &= 0, && i \in C. \end{aligned} \tag{A8}$$

This is equivalent to the system

$$\bar{B}_k h + \sum_{j\in A} \bar{\lambda}_j f_j'(x) - \sum_{i\in C} \bar{\mu}_i a_i = 0$$

$$\sum_{j\in A} \bar{\lambda}_j - 1 = 0$$

$$\left[f_{j_0}(x) + f_{j_0}'(x)^T h\right] - \left[f_j(x) + f_j'(x)^T h\right] = 0, \quad j \in A \setminus \{j_0\}$$

$$a_i^T(x+h) + b_i = 0, \qquad i \in C. \tag{A9}$$

Using (A3), (A5), and (A2) and a small calculation, it is seen that (A9) is the same as (7) with $h = \Delta x_k$, $\bar{\lambda} = \lambda^{(k+1)}$ and $\bar{\mu} = \mu^{(k+1)}$ (provided that $\bar{B}_k = B_k$). Thus, the point $x + h$ found by Powell's method is the same as the point x_{k+1} found by Method 2. Furthermore, Powell uses $\bar{\lambda}$ and $\bar{\mu}$ as the new multiplier estimates so also here there is coincidence with Method 2.

Finally, the matrices $\bar{B}_k$ and B_k are updated through the same formula. This is seen from the fact that the Lagrangian of (A6) is

$$\bar{L}(w,\lambda,\mu) = \delta - \sum_{j=1}^{m} \lambda_j \left[\delta - f_j(x)\right] - \sum_{i=1}^{l} \mu_i \left[a_i^T x + b_i\right] \tag{A10}$$

with $w = (x, \delta)$. In Powell's method, $\bar{B}_k$ is updated by (A4) with $\bar{L}$ instead of L. Therefore, it is seen from (A2) and (A10) that the updates of $\bar{B}_k$ and B_k are identical.

Consequently, Method 2 is identical to Powell's method in its final stages provided that the matrices to be updated are initialized in the same way.

Appendix B
Superlinear Convergence

If the sequence $\{r_k\}$ converges to r^* in such a way that

$$\lim_{k \to \infty} \frac{|r_{k+1} - r^*|}{|r_k - r^*|} = \beta < 1$$

the sequence is said to converge linearly to r^* with convergence ratio β.

The case where $\beta = 0$ is referred to as superlinear convergence.

Appendix C
Cubic Interpolation Formula

As a well-known fact, a maximum of a continuous differentiable function $e(\omega)$ is characterized by $e' \triangleq \partial e/\partial \omega = 0$ and $\partial^2 e/\partial \omega^2 < 0$. This implies a change in the sign of $\partial e/\partial \omega$ and, in the neighborhood of the maximum, $\partial e/\partial \omega$ decreases as frequency increases. It follows that if there exist two points $\omega_1 < \omega_2$ such that

$$e'_{\omega_1} > 0 \quad \text{and} \quad e'_{\omega_2} < 0$$

at least one maximum of $e(\omega)$ lies between ω_1 and ω_2. If ω_1 and ω_2 are close enough to exclude the existence of multiple maxima, the location of the detected maximum can be estimated by the cubic interpolation formula [28]

$$\omega_{\max} = \omega_2 - \frac{(\omega_2 - \omega_1)[x - y - e'_{\omega_2}]}{e'_{\omega_1} - e'_{\omega_2} + 2x} \tag{C1}$$

where

$$y = -e'_{\omega_1} - e'_{\omega_2} + 3\frac{e(\omega_2) - e(\omega_1)}{\omega_2 - \omega_1} \tag{C2}$$

and

$$x = \left[y^2 - e'_{\omega_1} e'_{\omega_2}\right]^{1/2}. \tag{C3}$$

Acknowledgment

The authors are grateful to S. H. Chen and S. Daijavad of McMaster University for their high-level contribution to the success of this research program. They also acknowledge the contribution of Dr. W. M. Zuberek, currently with the Memorial University of Newfoundland, in the early stage of the work on worst-case circuit design. The assistance of M. L. Renault of McMaster University in preparing the numerical examples is also gratefully acknowledged.

References

[1] M. R. Osborne and G. A. Watson, "An algorithm for minimax optimization in the nonlinear case," *Comput. J.*, vol. 12, pp. 63–68, 1969.

[2] K. Madsen, "An algorithm for minimax solution of overdetermined systems of nonlinear equations," *J. IMA*, vol. 16, pp. 321–328, 1975.

[3] K. Madsen and H. Schjaer-Jacobsen, "Singularities in minimax optimization of networks," *IEEE Trans. Circuits Syst.*, vol. CAS-23, pp. 456–460, 1976.

[4] D. H. Anderson and M. R. Osborne, "Discrete, nonlinear approximations in polyhedral norms: A Levenberg-like algorithm," *Num. Math.*, vol. 28, pp. 157–170, 1977.

[5] J. W. Bandler and C. Charalambous, "Practical least pth optimization of networks," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-20, pp. 834–840, 1972.

[6] C. Charalambous and A. R. Conn, "An efficient method to solve the minimax problem directly," *SIAM J. Numerical Analysis*, vol. 15, pp. 162–187, 1978.

[7] R. Hettich, "A Newton-method for nonlinear Chebyshev approximation," in *Approximation Theory* (Lecture Notes in Mathematics, vol. 556), R. Schaback and K. Scherer, Eds. Berlin: Springer, 1976, pp. 222–236.

[8] S. P. Han, "Variable metric methods for minimizing a class of non-differentiable functions," *Mathematical Programming*, vol. 20, pp. 1–13, 1981.

[9] M. J. D. Powell, "The convergence of variable metric method for nonlinearly constrained optimization calculations," in *Nonlinear Programming*, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, Eds. New York: Academic Press, 1978, pp. 27–63.

[10] M. L. Overton, "Projected Lagrangian algorithms for nonlinear minimax and l_1 optimization," Stanford Univ., Dep. Comput. Sci., Rep. STAN-CS-79-752, 1979.

[11] G. A. Watson, "The minimax solution of an overdetermined system of non-linear equations," *J. IMA*, vol. 23, pp. 167–180, 1979.

[12] J. Hald and K. Madsen, "Combined LP and quasi-Newton methods for minimax optimization," *Mathematical Programming*, vol. 20, pp. 49–62, 1981.

[13] K. Madsen and H. Schjaer-Jacobsen, "Linearly constrained minimax optimization," *Mathematical Programming*, vol. 14, pp. 208–223, 1978.

[14] J. W. Bandler and W. M. Zuberek, "MMLC—A Fortran package for linearly constrained minimax optimization," Dept. Elec. and Comput. Eng., McMaster University, Hamilton, Canada, Rep. SOS-82-5, 1982.

[15] J. Hald, "MMLA1Q, a Fortran subroutine for linearly constrained minimax optimization," Institute for Numerical Analysis, Tech. Univ. Denmark, Rep. NI-81-01, 1981.

[16] J. W. Bandler and P. A. Macdonald, "Optimization of microwave networks by razor search," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-17, pp. 552–562, 1969.

[17] J. W. Bandler and P. A. Macdonald, "Cascaded noncommensurate transmission-line networks as optimization problems," *IEEE Trans. Circuit Theory*, vol. CT-16, pp. 391–394, 1969.

[18] J. W. Bandler, "Optimization methods for computer-aided design," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-17, pp. 533–552, 1969.

[19] D. Agnew, "Improved minimax optimization for circuit design," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 791–803, 1981.

[20] J. W. Bandler and C. Charalambous, "New algorithms for network optimization," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-21, pp. 815–818, 1973.

[21] C. Charalambous and A. R. Conn, "Optimization of microwave networks," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-23, pp. 834–838, 1975.

[22] A. R. Conn, "An efficient second order method to solve the (constrained) minimax problem," Dept. Combinatorics and Optimization, Univ. of Waterloo, Waterloo, Ontario, Rep. CORR 79-5, 1979.

[23] J. W. Bandler, P. C. Liu, and H. Tromp, "A nonlinear programming approach to optimal design centering, tolerancing and tuning," *IEEE Trans. Circuits Syst.*, vol. CAS-23, pp. 155–165, 1976.

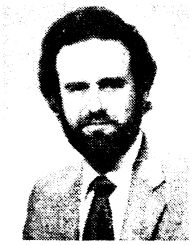
[24] H. Schjaer-Jacobsen and K. Madsen, "Algorithms for worst-case tolerance optimization," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 775–783, 1979.

[25] J. W. Bandler, P. C. Liu, and J. H. K. Chen, "Worst case network tolerance optimization," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-23, pp. 630–641, 1975.

[26] H. Tromp, "The generalized tolerance problem and worst case search," in *Proc. IEEE Conf. Computer-Aided Design of Electronic and Microwave Circuits and Systems* (Hull, England), 1977, pp. 72–77.

[27] J. W. Bandler and S. H. Chen, "Interactive optimization of multi-coupled cavity microwave filters," Dept. Elec. and Comput. Eng., McMaster University, Hamilton, Canada, Rep. SOS-84-13-R, 1984.
[28] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Computer J.*, vol. 6, pp. 163–168, 1963.
[29] A. E. Atia, "Computer-aided design of waveguide multiplexers," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-22, pp. 332–336, 1974.
[30] M. H. Chen, F. Assal, and C. Mahle, "A contiguous band multiplexer," *COMSAT Tech. Rev.*, vol. 6, pp. 285–306, 1976.
[31] M. H. Chen, "A 12-channel contiguous band multiplexer at *Ku*-band," in *1983 IEEE Int. Microwave Symp. Dig.* (Boston), pp. 77–79.
[32] R. G. Egri, A. E. Williams, and A. E. Atia, "A contiguous-band multiplexer design," in *1983 IEEE Int. Microwave Symp. Dig.* (Boston), pp. 86–88.
[33] J. W. Bandler, S. H. Chen, S. Daijavad, and W. Kellermann, "Optimal design of multicavity filters and contiguous-band multiplexers," *Proc. 14th Eur. Microwave Conf.* (Liege, Belgium), 1984, pp. 863–868.

✠



John W. Bandler (S'66–M'66–SM'74–F'78) was born in Jerusalem, Palestine, on November 9, 1941. He studied at Imperial College of Science and Technology, London, England, from 1960–1966. He received the B.Sc. (Eng.), Ph.D., and D.Sc. (Eng.) degrees from the University of London, London, England, in 1963, 1967, and 1976, respectively.

He joined Mullard Research Laboratories, Redhill, Surrey, England, in 1966. From 1967 to 1969, he was a Postdoctorate Fellow and Sessional Lecturer at the University of Manitoba, Winnipeg, Canada. He joined McMaster University, Hamilton, Canada, in 1969, where he is currently a Professor of Electrical and Computer Engineering. He has served as Chairman of the Department of Electrical Engineering and Dean of the Faculty of Engineering. He currently directs research, which has received substantial support by the Natural Sciences and Engineering Research Council of Canada under its Operating and Strategic Grants Awards, in the Simulation Optimization Systems Research Laboratory.

Dr. Bandler is also currently President of Optimization Systems Associates. He has provided consulting services and software to numerous organizations in the electronic, microwave, and electrical power industry, specializing in advanced applications of simulation, sensitivity analysis, and mathematical optimization techniques.

He is a contributor to *Modern Filter Theory and Design*, Wiley-Interscience, 1973. He has over 200 publications, four of which appear in *Computer-Aided Filter Design*, IEEE Press, 1973, one in *Microwave Integrated Circuits*, Artech House, 1975, and one in *Low-Noise Microwave Transistors and Amplifiers*, IEEE Press, 1981. Dr. Bandler was an Associate Editor of the IEEE TRANSACTIONS ON MICROWAVE THEORY AND TECHNIQUES (1969–1974). He was Guest Editor of the Special Issue of the IEEE TRANSACTIONS ON MICROWAVE THEORY AND TECHNIQUES on Computer-Oriented Microwave Practices (March 1974).

Dr. Bandler is a Fellow of the Institution of Electrical Engineers (Great Britain) and a member of the Association of Professional Engineers of the Province of Ontario (Canada).

✠

Witold Kellermann (S'82) was born in Opole, Poland, on August 8, 1954. He received the M.Sc. degree in electrical engineering from the Technical University of Wroclaw, Wroclaw, Poland, in 1978.

From 1978 to 1980, he was a Research and Teaching Assistant in the Department of Electrical Engineering, Higher School of Engineering, Opole, Poland. Since 1981, he has been employed as a Teaching Assistant in the Department of Electrical and Computer Engineering, McMaster University, Hamilton, Canada.

At McMaster University, he joined the Simulation Optimization Systems Research Laboratory, where he has been working under the supervision of Dr. J. W. Bandler. Mr. Kellermann was awarded an Ontario Graduate Scholarship for 1984/85. He is currently completing his Ph.D. requirements in electrical engineering from McMaster University.

His research interests are in the area of optimization-based computer-aided design of circuits and systems.

✠

Kaj Madsen was born in Denmark in 1943. He received his Cand. Scient. degree in mathematics from the University of Aarhus in 1968.

He has been a Lecturer in numerical analysis and computer science since 1968, apart from the academic year 1973 to 1974, when he was at AERE Harwell, Didcot, England. Most of the time he has been with the Institute for Numerical Analysis at the Technical University of Denmark, but during the years 1981–1983, he was at the Computer Science Department at Copenhagen University. During the summer of 1978, he visited McMaster University, Hamilton, Canada. His fields of interest in teaching and research are programming languages, optimization, and interval analysis.