MOSFET MODEL PARAMETER EXTRACTION BASED ON FAST SIMULATED DIFFUSION

by

Takayasu Sakurai and A. Richard Newton

Memorandum No. UCB/ERL M90/20

16 March 1990

MOSFET MODEL PARAMETER EXTRACTION BASED ON FAST SIMULATED DIFFUSION

by

Takayasu Sakurai and A. Richard Newton

Memorandum No. UCB/ERL M90/20

16 March 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

MOSFET MODEL PARAMETER EXTRACTION BASED ON FAST SIMULATED DIFFUSION

by

Takayasu Sakurai and A. Richard Newton

Memorandum No. UCB/ERL M90/20

16 March 1990

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

MOSFET Model Parameter Extraction Based on Fast Simulated Diffusion

Takayasu Sakurai* and A. Richard Newton

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA94720, U.S.A.
*) On leave from Semiconductor Device Engineering Lab., Toshiba Corporation, Kawasaki, 210, Japan

Abstract

A new algorithm, namely a Fast Simulated Diffusion (FSD) is proposed to solve a multi-minimal optimization problem on multi-dimensional continuous space. The algorithm performs a greedy search and a random search alternately and can give the global minimum with a practical success rate. A new efficient hill-decending method which is employed as the greedy search in the FSD is proposed. When the FSD is applied to a set of standard test functions, it shows an order of magnitude faster speed than the conventional simulated diffusion. Some of the optimization problems encountered in system and VLSI designs are classified into the multi-optimal problems. A MOSFET parameter extraction problem is one of them and the proposed FSD is successfully applied to the problem with a deep sub-micron MOSFET. A program listings are also attached.

1. Introduction

Some of the VLSI design problems including transistor sizing and model parameter extraction can be considered as an minimization problem in multi-dimensional continuous space with an object function which has plural local minima. Well-established minimization procedures for convex functions like Levenberg-Marquarlt method[1], can be easily trapped in one of the local minima and thus can not find a global minimizer. Recently a method called 'simulated diffusion' (SD) has been proposed[2] to find the global minimum of a multi-minimal function on continuous space. The simulated diffusion is conceived by the stimulus of 'simulated annealing' (SA), which is for combinatorial optimization problems[3]. Although much efforts have been made to theoretically study the behavior of the SD[4,5] and it has been demonstrated theoretically that under certain conditions the method guarantees to find the global minimizer with a probability of unity, little is known about the practical aspects of the SD as an optimization procedure. Although the SD could find a global minimizer, it was very slow[2].

In this paper, a new optimization method, named Fast Simulated Diffusion (FSD), is proposed to provide a faster way to find the global minimum. The new method is successfully applied to MOSFET parameter extraction problem in the deep sub-micron regime.

In Section 2, the basic idea of the conventional SD is described. In Section 3, the algorithm of the fast SD is presented and the advantage of the fast SD over the conventional SD is clarified in Section 4. Section 5 is dedicated for the discussion on the application of the proposed fast SD method to the practical VLSI design problems, namely a MOSFET model parameter extraction problem for a circuit simulator. A conclusion is summarized in Section 6.

2. Conventional Simulated Diffusion (CSD)

First, a basic idea of the conventional simulated diffusion is described. Essentially, the SD makes use of the physical fact that a particle placed in a given potential and with Brownian motion is diffused into the global minimum of the given potential profile. The following is the more mathematical formulation of the process. An differential equation which describes a diffusion process of a particle with Brownian motion is given as

$$dx = - \nabla f(x)\, dt + \sqrt{2T}\, dw \,, \tag{1}$$

where t is time, x is the space coordinate which points the location where the particle is, $f(x)$ is a potential function in which the particle is put, ∇ is a gradient operation, dw is a Gaussian random noise and T is a temperature. The first term on the left-hand side corresponds to the drift component of the movement and the second term signifies the Brownian movement. When the temperature is high, the second term dominates and the movement of the particle is just stochastic. On the other hand, when the temperature becomes low, the first term dominates and the process approaches pure hill-descent. The second term is essential to get out of the local minima and the first term gives the tendency to minimize the function.

It has been shown[4] that with a proper cooling schedule, the probability distribution of x, $P(x)$, approaches

$$P(x) \propto exp\{-f(x) / T\} \qquad (2)$$

as t goes infinity. This means that the limit distribution is independent of the initial value and is peaked around the global minimizers of $f(x)$. This in turn means that if dx is integrated over a long period of time, x tends to converge to a global minimum of the function $f(x)$. This is the principle of the conventional simulated diffusion. Aluffi-Pentini et al.[2] numerically integrated Eq.1 to obtain the minimizer from this first principle. However, the numerical process turned out to be slow.

If there are constraints in the original minimization problem, it is possible to introduce penalization functions and make it a minimization problem without constraints[6]. Consequently, the SD can be applicable not only to the unconstrained minimization problems but also to the optimization problems with constraints.

3. Fast Simulated Diffusion (FSD)

In this work, instead of integrating Eq.1 directly, two basic modifications are made. One is the introduction of an accept/non-accept function of a Boltzman distribution type, which is commonly used in the simulated annealing. If the next point $xnext$ (= $x+dx$) gives the smaller function value than the current x, take the $xnext$. On the other hand, if $xnext$ gives the larger function value than the current x, generate a random number R in [0,1] and calculate $P=exp[-\{f(x+dx)-f(x)\}/T]$. If $R < P$, then accept the $xnext$, otherwise discard the $xnext$ and re-generate $xnext$. The higher the function value becomes in the next move, the less probable it becomes to accept the move. The introduction of this Boltzman accept/non-accept function can be validated by Eq.2 which is the Boltzmann distribution itself and it is expected to help establishing the probability distribution of Eq.2 faster than simply integrating Eq.1. In practice, the use of this accept/non-accept function prunes very 'stupid' moves to be taken otherwise and consequently accelerates the convergence.

The other modification is concerning with the generation of the next move. Instead of adding the greedy hill-descending part (the first term of Eq.1) and the random perturbation part (the second term of Eq.1), the generation of x based on a greedy method and a random method are carried out *alternately*. That is, in one time, dx is calculated by $-\nabla f(x)dt$ and the next time, dx is calculated as $\sqrt{2T}\,dw$. By generating the next move by the gradient method and the random method alternately, it is possible to achieve hill-descending even if the temperature is relatively high. In the relatively high temperature range, the random term happens to generate ineffective moves and it is probable that no improvements of $f(x)$ will be observed if the two terms are added together as in the CSD, because the hill-descending part can be hidden by the dominating random noise and all moves are possibly rejected.

Several considerations are taken other than the above-mentioned two major modifications to make the method more efficient. First, since it is expensive to calculate the direction of $\nabla f(x)$ if the space has large dimensions, $<\nabla f(x)\cdot \boldsymbol{r}>\boldsymbol{r}$ is used instead, where $\boldsymbol{r}$ is a unit vector of a randomly picked axis. This is because the expected direction of $<\nabla f(x)\cdot \boldsymbol{r}>\boldsymbol{r}$ approaches $\nabla f(x)$ in a long run[2].

Secondly, since it is difficult to choose a good value of dt, a new hill-descending method is proposed and used. The choice of dt is critical because if it is too small, the improvement of the solution is small but if it is too big, $-\nabla f(x)\,dt$ does not always give the improvement. The proposed method is described in Fig.1. First, pick a random axis direction. If the function is concave at the point along the picked axis, quadratic fitting is carried out and the minimum x in that direction is guessed and adopted as the $xnext$. If the function is convex, choose a small dx first and double the dx until $f(x+dx)$ fails to decrease from $f(x)$. The doubling process is confined up to a certain number of times (three in the following examples). It is not an objective of this new hill-descending method to obtain the exact minimum in that direction but to provide an inexpensive yet effective way to improve the solution, since there is always a possibility that the random search can give rise to a big jump and then the previous hill-descending becomes wasteful. This method is considered as an inexpensive adaptive method to determine a good value of dt.

The detailed algorithm of the FSD is shown in Fig.2. In the first several external loops (around 10 loops), the hill-descending is not taken and only random search is carried out because big jumps are accepted in the high temperature stage and the hill-descending is not effective at all.

The initialization scheme and the temperature update algorithms in [7] are adopted. That is, the initial temperature, Tinit, is determined by a statistics gathered over randomly selected Ninit points as is shown in Fig.2. The adopted temperature update algorithm (cooling schedule) is basically a geometric decrease. The theory of the SD suggests that the cooling schedule

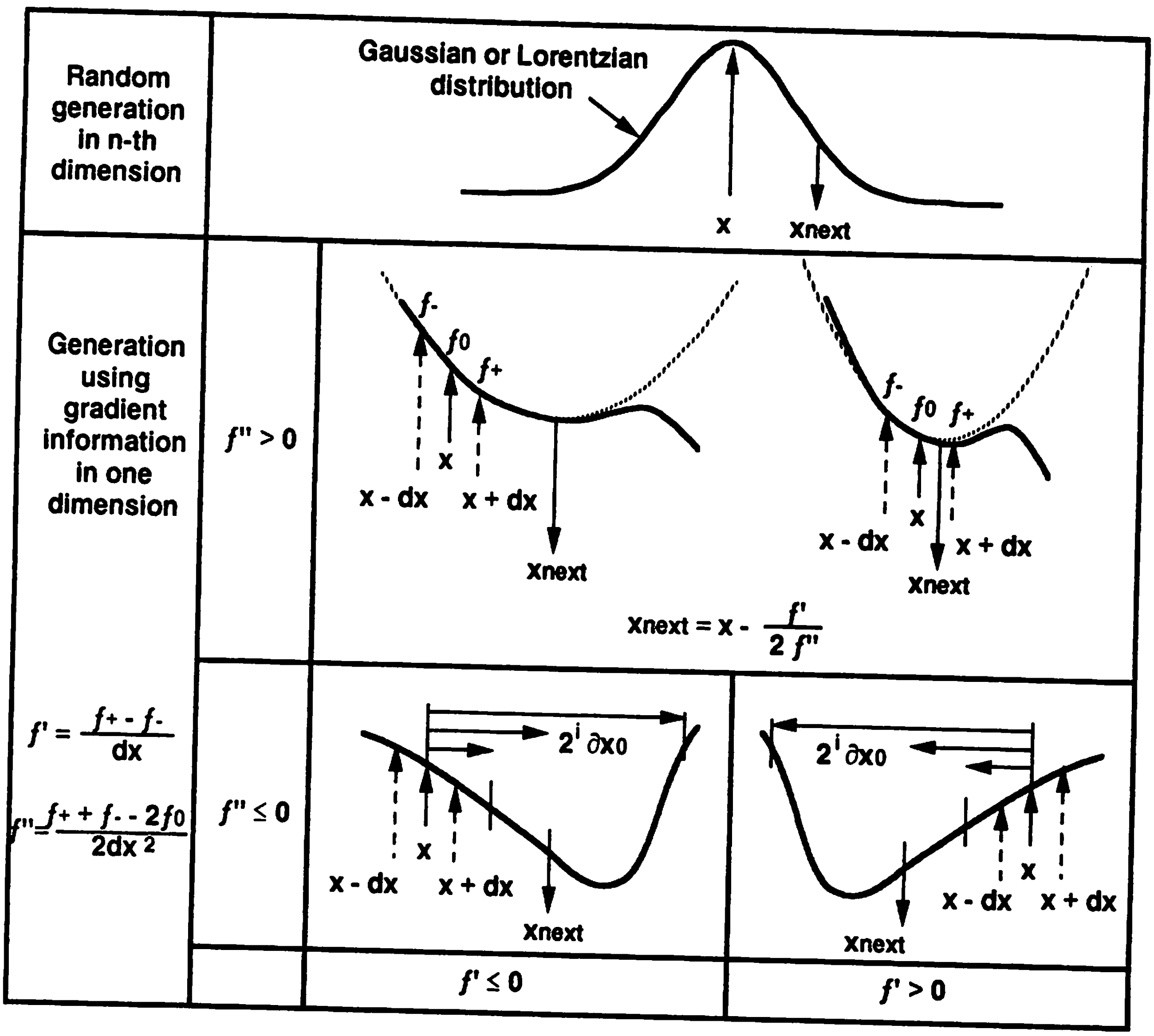

Fig. 1 Proposed hill-descending method using f' and f'' information

```
main {
    T = Tinit = (κ * (standard deviation of f(X) over randomly selected Ninit points));
                              // Set initial temperature by using huristics. κ =0.2, Ninit=200
    S = Sinit;                // Set initial S to Sinit
    Xinit = (Xinit_Given_by_User or one of those randomely selected Ninit points
            whichever gives the minimum value of f);
    Xopt = X = Xinit;         // Set initial X to the best X known
    do {                      // External loop with varying T
        iINT = 0;
        while (a certain times (ex. 15~25*dimension)) {// Internal loop with constant T
            iINT ++;
            Generate_X();          // Generate new X by simulated diffusion
            Δf= f(Xnew) - f(X);
            if (Δf < 0) {                                  // If cost decreases,
                X = Xnew;                                  // adopt the Xnew.
                if ( f(X) < f(Xopt) ) { Xopt = X }         // Save best X.
            } else {                                       // Even if cost increases,
                P = exp(-Δf / T);                          // adopt the Xnew
                R = random number in [0,1];                // according to
                if (R < P) {X = Xnew}                      // Boltzman distribution.
            }
        }
        if ( f(X) > f(Xopt) ) { X = Xopt }                 // Resume the best X.
        if (cost is not improved considerably) {
            iLast_Gasp ++;               // If cost is not improved considerably,
        } else {                         // take Last_Gasp loop,
            iLast_Gasp = 0;              // where T is increased a little
        }                                // and then decreased to freeze.
        Update_T();
        Update_S();                      // S = Sinit * (T / Tinit)^a ; a = 0.5~1(ex. a=0.75)
    } until ((iLast_Gasp > iLast_Gasp_Max) and (T / Tinit < T_Ratio_Min) )
            // until  Last_Gasp loop is taken long enough and T gets low enough.
    solution = Xopt;
}

Generate_X() {                 // generate new X
    if (iINT < mINT) {
        gradient_Flag = 0;
    } else {
        gradient_Flag = 1 - gradient_Flag;
    }
    if (gradient_Flag == 1) {
        Randomly select single variable Xi and move only in this axis.
        Generate Xnew with gradient information according to f'' and f' values.
    } else {
        Xnew = X + S * (n-th dimensional Gaussian or Lorentzian distribution)
    }
}

Update_T() {                   // update temperature
    if (iLast_Gasp = 0) {
        T_Factor = exp(-λ T / σ)  // ex. λ=0.7, σ=standard dev. of accepted f(x)
        if  (T_Factor < T_Factor_Min) { T_Factor = T_Factor_Min (=0.5)}
        T *= T_Factor
    }
    if (  1 ≤ iLast_Gasp ≤ n2)    { T *= T_Factor2 (T_Factor2 > 1, ex. 1.3) } // ex. n2=4
    if (n2 ≤ iLast_Gasp)          { T *= T_Factor1 (T_Factor1 < 1, ex. 0.75) }
}
```

Fig.2 Algorithm of Fast Simulated Diffusion should is much slower the geometric decrease to guarantee to reach the global minimum even for ill-conditioned functions[11]. However, for practical problems, the geometric cooling works well [7,12].

The initial distribution of dw is chosen so that almost all the feasible space is covered by the random search at the initial stage. Such a distribution can be determined when the feasible region of x is given as a supercube, $[x_{min}, x_{max}]$. In practical problems, this feasible region is known apriori (see Section 5) or is set sufficiently large. If the randomly generated x falls out of the feasible region, it is re-generated. At the last stage of the FSD, when the object function shows little change, Last_Gasp sequence is taken where the temperature is increased a little and then decreased to freeze. The details are described in Fig.2.

In Fig.2, a multiplier S controls the random search space volume. S should be shrinked proportional to $\sqrt{T}$ as T is lowered according to the first principle of the SD, but in practice, S can be shrinked faster and is proportional to T^n (n = 0.5 ~ 1.0).

4. Comparison between the FSD and the CSD

TABLE I shows a comparison between the FSD and the CSD when they are applied to a set of standard test functions given in [2]. On the average, the FSD is about an order of magnitude faster than the CSD. Let's define a 'reachability' as a probability to be able to find out the global minimum in a finite period of time using the given algorithm. Successful trials in ten trials in TABLE I can be used as an index for the reachability. Improvement in efficiency or

TABLE I Comparison of the conventional SD and the Fast SD

problem description			CSD (*1)	Fast Simulated Diffusion (this work)		
problem # (*1,2)	dimen-sion	# of local minima	NF1: # of function evaluation	NF2: # of function evaluation (*3)	success rate in 10 trials (*4)	NF2/NF1 (%)
1	1	3	7168	3644	1.0	50.8
2	1	19	77699	2586	1.0	3.3
3	2	760	241215	3067	1.0	1.3
4	2	760	76894	2968	0.8	3.9
5	2	760	183819	2734	0.7	1.5
6	2	6	10822	4573	1.0	42.3
7	2	25	159549	3408	1.0	2.1
8	3	125	72851	3572	1.0	4.9
9	4	625	49690	3818	1.0	7.7
10	5	1e5	72226	5246	1.0	7.3
11	8	1e8	136061	9819	0.9	7.2
12	10	1e10	98985	12206	1.0	12.3
13	2	900	23770	4081	1.0	17.2
14	3	2.7e4	66010	4036	1.0	6.1
15	4	8.1e5	122166	4473	1.0	3.7
16	5	7.6e5	66365	4588	1.0	6.9
17	6	1.1e7	98974	5559	1.0	5.6
18	7	1.7e8	109886	6509	0.9	5.9
average	3.8	5.7e8	93009	4828	0.96	5.2

*1) F. Aluffi-Pentini, V.Parisi, & F.Zirilli, "Global Optimization and Stochastic Differential Equations ," J. of Optimization theory and Application, Vol.45, No.1, pp.1-16, Sept.1985.

*2) Expressions for problem # 4 and 5 presented in Aluffi-Pentini's paper seems to contain errors and hence they are modified and used.

*3) Average over 10 trials

*4) The rate of having reached to the global minimum in10 trials. Aluffi-Pentini et al's paper does not contain this information. It only gives yes or no in one trial as the reachability information.

speed might be obtained at the risk of degradation in the reachability. Judging from TABLE I, the reachability of the FSD is in the practical range.

When the first term in Eq.1 is neglected, the method becomes similar to the SA. This SA-like method is supposed to be better than the mere extension of the SA to a continuous space[8], since the random search space is decreased by a factor of $\sqrt{2T}$ as the temperature is lowered. The FSD is faster than this SA-like method as shown TABLE II because less number of 'stupid' moves are generated. In TABLE II, the results of using a Lorentzian distribution[9] instead of the Gaussian distribution are also shown. Further improvement in both speed and reachability is observed. Since the Lorentian distribution has a longer tail than the Gaussian distribution, with the Lorentzian distribution, the possibility of a big jump is rather high even at the low temperature and it helps to get out of the local minima at the final stage.

5. Application to MOSFET Model Parameter Extraction

Model parameter extraction problem is to minimize the object function

$$f(p) = \sum_{\text{various bias conditions}} weight(\text{bias condition}) \cdot |I_{D,measured} - I_{D,model}(p)| \qquad (3)$$

with the model parameters p as variables. In the above expression, I_D denotes drain current of a MOSFET and the weight function is optional. SPICE LEVEL3 MOS model is used as a MOS model in this section as an example, although the method is not restricted to a specific device models. The model parameters p that minimizes $f(p)$ is considered to be a good extracted parameter set and can be used for the circuit simulation afterwards. With the conventional

TABLE II Two modified version of the Fast Simulated Diffusion

		Simulated Annealing-like random search			Simulated Diffusion with Lorentzian Distribution		
problem #	NF1: # of function evaluation	NF3: # of function evaluation	success rate in 10 trials (Table I *4)	NF3 / NF1 (%)	NF4: # of function evaluation	success rate in 10 trials (Table I *4)	NF4 / NF1 (%)
1	7168	3111	1.0	43.4	2939	1.0	41.0
2	77699	3060	1.0	3.9	2387	1.0	3.1
3	241215	4131	0.7	1.7	2877	1.0	1.2
4	76894	5967	0.7	7.8	3170	0.8	4.1
5	183819	5831	0.7	3.2	2678	0.7	1.5
6	10822	5151	0.9	47.6	3609	1.0	33.4
7	159549	7701	0.9	4.8	3023	1.0	1.9
8	72851	11322	1.0	15.5	3232	1.0	4.4
9	49690	11475	1.0	23.1	3401	1.0	6.8
10	72226	20053	1.0	27.8	4108	1.0	5.7
11	136061	28689	0.9	21.1	7716	1.0	5.7
12	98985	33986	1.0	34.3	9856	1.0	10.0
13	23770	7378	1.0	31.0	3294	1.0	13.9
14	66010	10761	1.0	16.3	3446	1.0	5.2
15	122166	11424	1.0	9.4	4051	1.0	3.3
16	66365	14790	1.0	22.3	4140	1.0	6.2
17	98974	19730	1.0	19.9	4903	1.0	5.0
18	109886	22962	1.0	20.9	6295	1.0	5.7
average	93009	12640	0.93	13.6	4174	0.97	4.5

extraction program, the extracted parameters give the local minimum of $f(p)$ which is the nearest to the given an initial parameter set[1]. However, in practice, it is difficult to guess the initial parameter set correctly. The FSD does not require any initial value. All information needed beforehand is on the bounds, p_{min} and p_{max}, for each parameter. This is rather easy because it is known that, for example, the parameter KAPPA is in the range of 0~2. The used values for the bounds are tabulated in TABLE III. The same set of bounds is used to extract 0.25μm and 1μm MOSFET parameters.

In order to further increase the efficiency in this specific problem of parameter extraction, the search is carried out in the logarithmic space for NSUB, VMAX and NSS. This measure is taken to achieve a balanced search over a space because for example, VMAX is in the range of 1e4 ~ 1e8 and the increase from 1e4 to 1.1e4 tends to generate the similar effect on $I_{D,model}$ as the increase from 1e7 to 1.1e7 does. For other parameters, the search is made in the linear scale.

The multi-minimal nature of the object function is shown in Fig.3 together with the generated x points with the FSD. An example of the fitted drain current is shown in Fig.4 for 1μm MOSFET. Figure 5 shows another example of the parameter extraction with a 0.25μm channel-length MOSFET[10]. Good agreement is observed even down to the deep sub-micron region.

6. Conclusions

Fast simulated diffusion is proposed to provide a fast method to find a global minimum of a multi-minimal function on multi-dimensional continuous space. The fast simulated diffusion

TABLE III MOSFET model parameter extraction results

parameter name	p_{min}	p_{max}	extracted params for 1μm MOS	extracted params for 0.25μm MOS
VTO	0	1.5	0.769	0.743
UO	10	1000	900	406
NSUB	1e16	1e20	1.80e17	5.97e18
GAMMA	0.2	1.5	0.928	0.477
ETA	0	2	0.0293	0.00754
THETA	0	2	0.996	0.775
KAPPA	0	2	0.382	0.299
VMAX	1e4	1e8	5.26e7	1.81e5
XJ	1e-8	3e-8	2e-7(fixed)	2.02e-8
TOX	-	-	2e-8(fixed)	5e-9(fixed)
NFS	-	-	0(fixed)	0(fixed)
LD	-	-	0.1(fixed)	0(fixed)
W	-	-	10e-6(fixed)	4e-6(fixed)
L	-	-	1.0e-6(fixed)	0.25e-6(fixed)
# of func. eval.	-	-	4258	3114
time (min.•MIPS)	-	-	~18	~13

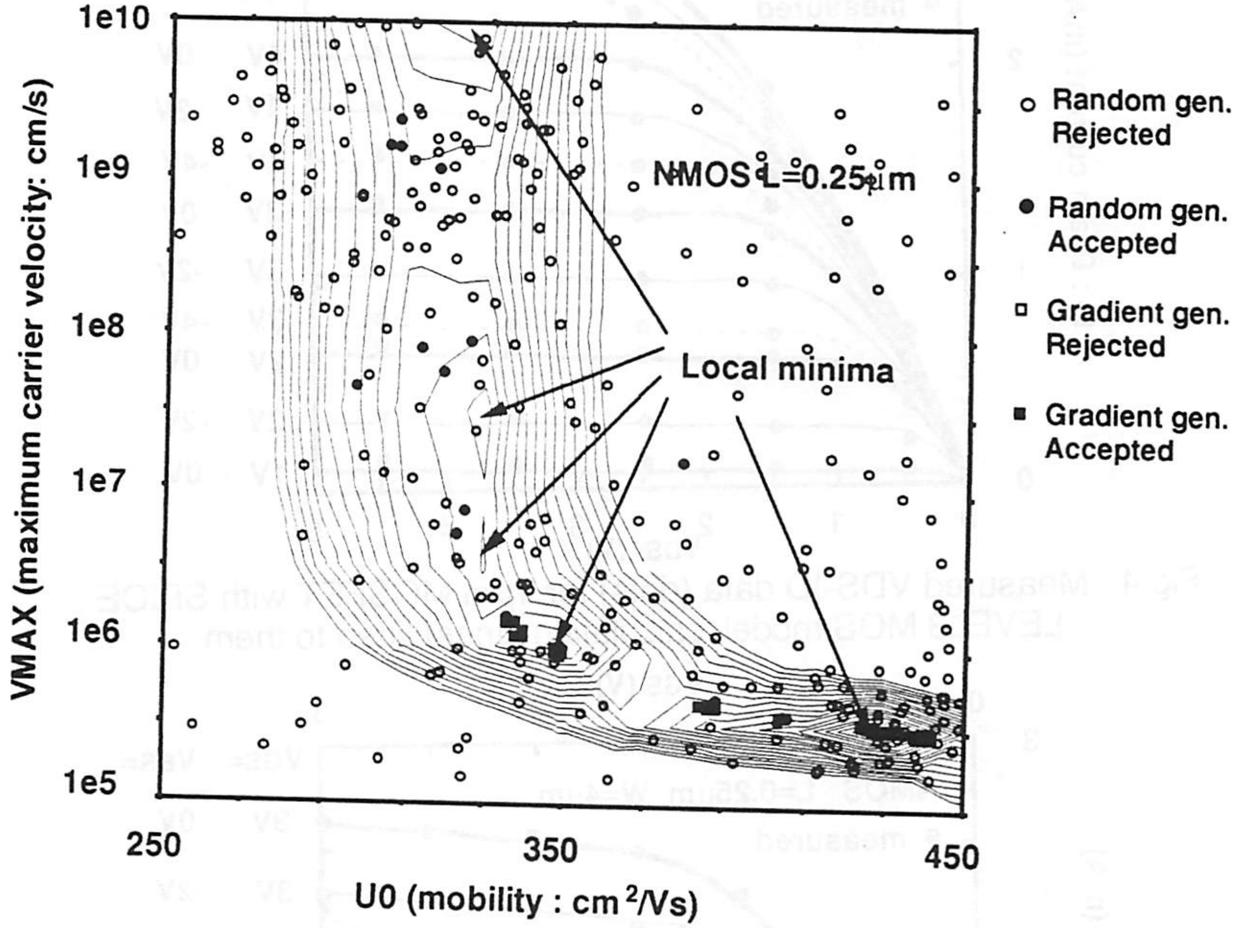

Fig.3 Multiple-minimum nature of MOS model parameter extraction problem and generated x points

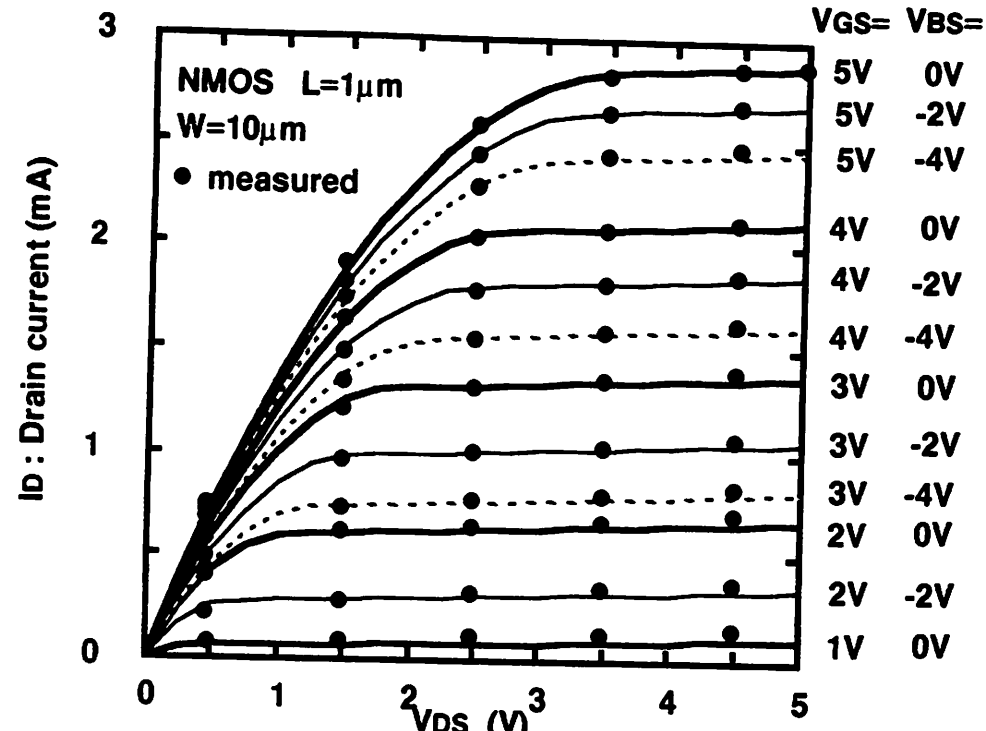

Fig.4 Measured VDS-ID data (dots) for 1μm MOSFET with SPICE LEVEL3 MOS model calculation (lines) fitted to them

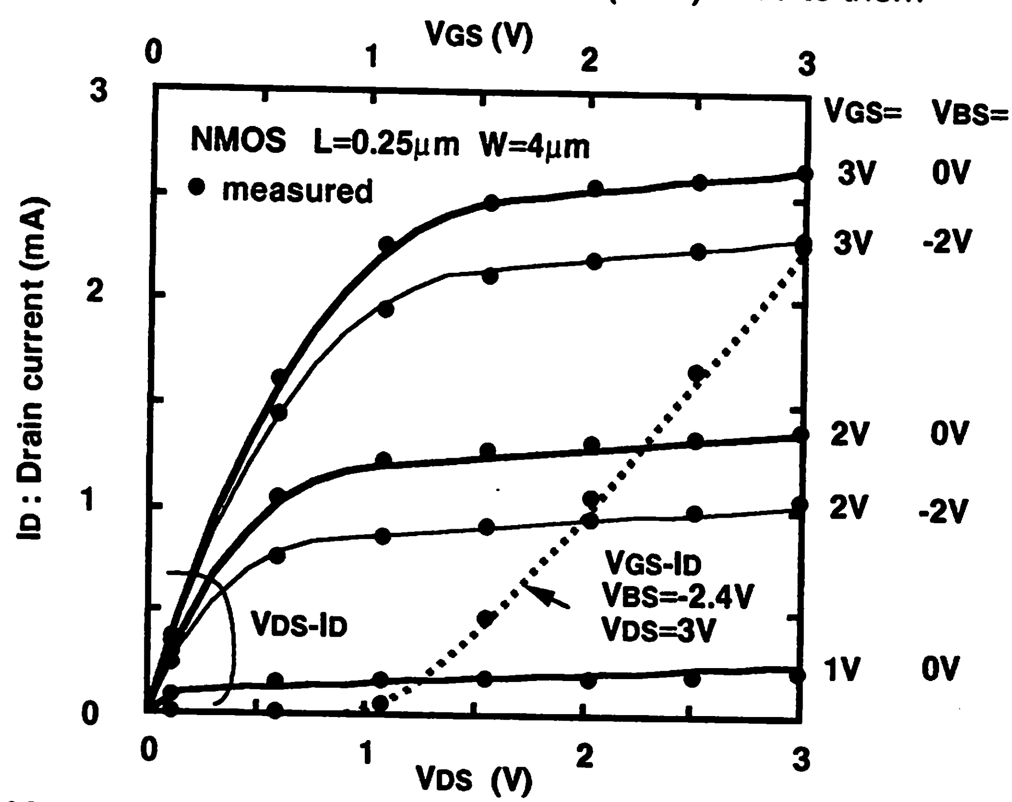

Fig.5 Measured VDS-ID data (dots) for 0.25μm MOSFET with SPICE LEVEL3 MOS model calculation (lines) fitted to them

shows about an order of magnitude faster speed over the conventional simulated diffusion, when applied to a set of standard test functions. The fast simulated diffusion is successfully applied to MOSFET model parameter extraction in the deep submicron region. The method is supposed be be applicable to other optimization problems encountered in system and VLSI designs.

Acknowledgments

The encouragement of Prof. B.Brayton, Prof. A.Sangiovanni-Vincentelli, Y.Unno, Y.Takeishi, H.Yamada and T.Iizuka throughout the course of this work is appreciated. The discussions with Prof. E.Polak and Dr. F.I Romeo on the general multi-minimal optimization problems and simulated annealing are useful and are to be acknowledged. Assistance provided by H. Ishiuchi, T. Quarles and R. Spickelmier concerning MOS physics, SPICE and computer environments is also appreciated. This work was supported by a grant from Toshiba Corporation.

References

[1] K.Doganis & D.L.Scharfetter, "General Optimization and Extraction of IC Device Model Parameters," IEEE Trans. on ED, ED-30, No.9, pp.1219-1228, Sept.1983.

[2] F.Aluffi-Pentini, V.Parisi & F.Zirilli, "Global Optimization and Stochastic Differential Equations," J. of Optimization Theory and Applications, Vol.47, No.1, pp.1-16, Sept.1985.

[3] S.Kirkparick, C.D.Gellatt & M.P.Vecchi, "Optimization by Simulated Annealing," Science Vol.220, No.4598, pp.671-680, May 1983.

[4] S.Geman & C-R Hwang, "Diffusions for Global Optimization," SIAM J. Control and Optimization, Vol.24, No.5, pp.1031-1043, Sept.1986.

[5] T-S Chiang, C-R Hwang & S-J Sheu, "Diffusion for Global Optimization in R^n," SIAM J. Control and Optimization, Vol.25, No.3, pp.737-753, May 1987.

[6] A.V.Fiacco and G.P.McCormick, "Computational Algorithm for the Sequential Unconstrained Minimization Technique for Nonlinear Programming," Management Science, 10, pp.601-617, 1964.

[7] M.D.Huang, F.Romeo and A.Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," ICCAD'86, pp.381-384, Nov.1986.

[8] M.K.Vai & M.F.D.Ng, "A Technology-Independent Device Modeling Program Using Simulated Annealing Optimization," CICC'89, pp.9.4.1-4, May 1989.

[9] H.Szu & R.Harley, "Fast Simulated Annealing," Physics Letters A, Vol.122, No.3,4, pp.157-162, Jun.1987.

[10] M.C.Jeng, P.K.Ko & C.Hu, "A Deep-Submicron MOSFET Model for Analog/Digital Circuit Simulations," IEDM'88, pp.114-117, Dec.1988.

[11] F.I.Romeo, "Simulated Annealing: Theory and Applications to Layout Problems," U.C.Berkeley ERL Memo, No.UCB/ERL M89/29, Mar.1989.

[12] G.B.Sorkin, "Simulated Annealing on Fractals: Theoretical Analysis and Relevance for Combinatorial Optimization," Proceedings of the 6th Annual MIT Conference on Advanced Research in VLSI, ed. by W.Dally, Apr.1990.

Appendix A   Program Listing

Program source codes are shown in the following pages. The programs are written in QuickBasic Ver.1.0 for Macintosh SE/30. There are two programs. The first one is a simulated diffusion program for a set of test functions, which corresponds to Section 4 of this paper. The second one is a simulated diffusion program to extract device parameters for MOSFET LEVEL3 model which is found in SPICE 2 and SPICE3 circuit simulators. This program corresponds to Section 5 of the paper.

```
'--- Minimization with simulated annealing ---
OPTION BASE 0
maxnX% = 14
maxnData% = 1000
maxnEXT% = 100
DIM SHARED oldX(maxnX%), newX(maxnX%), oldEXTX(maxnX%)
DIM SHARED optEXTX(maxnX%), optINTX(maxnX%)
DIM SHARED minX(maxnX%), maxX(maxnX%), initX(maxnX%),
x(maxnX%), rangeX(maxnX%)
DIM SHARED histEXTCost(maxnEXT%), histINTACost(maxnData%)
DIM SHARED vgsm(maxnData%), vdsm(maxnData%), vbsm(maxnData%)
DIM SHARED idm(maxnData%), ivgdbm%(maxnData%),
weightm(maxnData%)
DIM SHARED iX2iV%(maxnX%+1), iV2iX%(maxnX%+1)
DIM SHARED minV(maxnX%), maxV(maxnX%), initV(maxnX%),
nameV$(maxnX%)
DIM SHARED iline%, ntokenMax%
DIM SHARED maxVgd, maxIds
ntokenMax% = 6
DIM token$(ntokenMax%)

Initializer:
  rseed% = 17635
  infinity = 1E+20
  infinitesmal = 1E-20
  initTK = .1
  updateTLambda = .7
  factorUpdateTLB = .5
  epsExitEXTrelCost = .02
  minnEXT% = 5
  iGaussCalled% = 0
  iGenerateXCalled% = 0
  epsRndFac = 1!
  '--- mos3 initialization ---
  nXP% = 14
  ivto% = 1: insub% = 2: igammac% = 3: iuo% = 4: itheta% = 5
  ikappa% = 6: ivmax% = 7: ieta% = 8: ixP% = 9: ild% = 10
  itox% = 11: infs% = 12: iP% = 13: iw% = 14
  ialP% = 15
  CALL initializeV(ivto%, "vto", -2!, 2!, .8)
  CALL initializeV(insub%, "nsub", 1E+16, 1E+20, 1E+18)
  CALL initializeV(igammac%, "gamma", 0!, 2!, .5)
  CALL initializeV(iuo%, "uo",10!, 1600!, 400!)
  CALL initializeV(itheta%, "theta", 0!, 2!, .7)
  CALL initializeV(ikappa%, "kappa", 0!, 2!, .5)
  CALL initializeV(ivmax%, "vmax", 100000!, 1E+08, 1000000!)
  CALL initializeV(ieta%, "eta", 0!, 2!, 1!)
  CALL initializeV(ixP%, "xj", 0!, .00001, 0!)
  CALL initializeV(ild%, "ld", 0!, .00001, 0!)
  CALL initializeV(itox%, "tox", 1E-10, .0000001, 3E-08)
  CALL initializeV(infs%, "nfs", 0!, 0!, 0!)
  CALL initializeV(iP%, "l", 0!, .0001, .000001)
  CALL initializeV(iw%, "w", 0!, .001, .00001)
  iX2iV%(ialP%) = ialP%
  iV2iX%(ialP%) = ialP%

  pi = 3.141592
  q = 1.6E-19
  vtherm = .025
  ni = 1.45E+10 * 1000000!
  epssi = 8.855E-12 * 11.9
  epsox = 8.855E-12 * 3.9

Windower:
 '--- initialize window 1, 2 ---
WINDOW 3,"Text Window", (0,30)-(480,300),1
TEXTSIZE 9
WINDOW 2,"Graphics Window", (200,30)-(480, 310),3
TEXTSIZE 9
PICTURE ON
WINDOW 1,"Report Window", (0,30)-(200, 300),1
 '--- initaiize I/O ---
TEXTSIZE 9
OPEN "scm:" FOR OUTPUT AS #1

 '--- menu ---
MENU 1,0,1,"File"
MENU 2,0,1,"Edit"
MENU 3,0,1,"Run"
MENU 4,0,0,"Params"
MENU 5,0,1,"Graph"

MENU 1,1,1,"Load": cmdkey 1,1,"L"
MENU 1,2,1,"Show Loaded Data": cmdkey 1,2,"S"
MENU 1,3,1,"Output Select": cmdkey 1,3,"O"
MENU 1,4,0,"Print": cmdkey 1,4,"P"
MENU 1,5,0,"-"
MENU 1,6,1,"Quit": cmdkey 1,6,"Q"

MENU 2,1,1,"Copy":cmdkey 2,1,"C"

MENU 3,1,1,"Simulated Diff."
MENU 3,2,1,"Manual Fit"
MENU 3,3,1,"Contour Map"
MENU 3,4,1,"Write Results"

MENU 4,1,0,"Params Set": cmdkey 4,1,"M"

MENU 5,1,1,"Vds_Id": cmdkey 5,1,"G"
MENU 5,2,1,"Vgs_Id": cmdkey 5,2,"H"

ON MENU GOSUB Menucheck: MENU ON
Idle:
  GOTO Idle

Menucheck:
  menunumber = MENU(0)
  menuitem = MENU(1)
  MENU
  ON menunumber GOSUB Filer, ClipBoarder, Runner, Setter, Grapher
RETURN

Filer:
  ON menuitem GOSUB Loader, DataShower, Outer, Quitter, Quitter
,Quitter
RETURN

Quitter:
  CLOSE
  WINDOW CLOSE 1
  WINDOW CLOSE 2
  WINDOW CLOSE 3
  PICTURE OFF
END

ClipBoarder:
  ON menuitem GOSUB ClipCopier
RETURN

Runner:
  ON menuitem GOSUB SDRunner, Manualer, Contourer, ResShower
RETURN

Setter:
  ON menuitem GOSUB Quitter
RETURN

Grapher:
  ON menuitem GOSUB VdsIder, VgsIder
RETURN

ClipCopier:
  PICTURE OFF
  image$ = PICTURE$
  OPEN "CLIP:PICTURE" FOR OUTPUT AS #3
  PRINT#3, image$
  CLOSE #3
RETURN

Eraser:
  WINDOW 3
  WINDOW 2
  WINDOW 1
  image$ = ""
  CLS
RETURN

Loader:
  '--- load data ---
  infile$ = FILES$(1,"TEXT")
  IF (infile$ = "") THEN RETURN
  OPEN infile$ FOR INPUT AS #2
  '--- input ---
  skipLoadFlag% = 0
  idata% = 0
  iX% = 0: iP% = nXP% + 1
```

```
iline% = 0
WHILE NOT EOF(2)
   LINE INPUT #2, inline$
   iline% = iline% + 1
   '---- parse the input line ----
   CALL parse(inline$, token$(), ntoken%, errorParseFlag%)
   IF (errorParseFlag% = 1) GOTO breakLoadLoop
   IF (token$(1) = "/*") OR (token$(1) = "*/") THEN skipLoadFlag% = 1
   IF (ntoken% <> 0) AND (token$(1) <> "*") AND (skipLoadFlag% = 0)
THEN
      IF (token$(2) = "=") THEN
         '---- voltage card ----
         SELECT CASE UCASE$(token$(1))
         CASE "VGS"
            IF (token$(3) = "*") THEN ivgdb% = 1 ELSE vgs = VAL(token$(3))
         CASE "VDS"
            IF (token$(3) = "*") THEN ivgdb% = 2 ELSE vds = VAL(token$(3))
         CASE "VBS"
            IF (token$(3) = "*") THEN ivgdb% = 3 ELSE vbs = VAL(token$(3))
         '---- variable parameter card ----
         CASE "VTO"
            CALL paramRead(token$(), iX%, iP%, ivto%)
         CASE "NSUB"
            CALL paramRead(token$(), iX%, iP%, insub%)
         CASE "GAMMA"
            CALL paramRead(token$(), iX%, iP%, igammac%)
         CASE "UO"
            CALL paramRead(token$(), iX%, iP%, iuo%)
            IF (minX(iV2iX%(iuo%)) <= 10!) THEN
               minX(iV2iX%(iuo%)) = 10!
            END IF
         CASE "THETA"
            CALL paramRead(token$(), iX%, iP%, itheta%)
         CASE "KAPPA"
            CALL paramRead(token$(), iX%, iP%, ikappa%)
         CASE "VMAX"
            CALL paramRead(token$(), iX%, iP%, ivmax%)
         CASE "ETA"
            CALL paramRead(token$(), iX%, iP%, ieta%)
         CASE "XJ"
            CALL paramRead(token$(), iX%, iP%, ixj%)
            IF (minX(iV2iX%(ixj%)) <= 0) THEN minX(iV2iX%(ixj%)) =
infinitesmal
         CASE "LD"
            CALL paramRead(token$(), iX%, iP%, ild%)
         CASE "TOX"
            CALL paramRead(token$(), iX%, iP%, itox%)
         CASE "NFS"
            CALL paramRead(token$(), iX%, iP%, infs%)
         CASE "L"
            CALL paramRead(token$(), iX%, iP%, il%)
         CASE "W"
            CALL paramRead(token$(), iX%, iP%, iw%)
         CASE ELSE
            msg$ = "Undefined token "+token$(1)+" in line"+STR$(iline%)
            CALL errmsg(msg$)
            GOTO breakLoadLoop
         END SELECT
      ELSE
         '---- real data ----
         idata% = idata% + 1
         vgsm(idata%) = vgs
         vdsm(idata%) = vds
         vbsm(idata%) = vbs
         ivgdbm%(idata%) = ivgdb%
         SELECT CASE ivgdb%
          CASE 1
            vgsm(idata%) = VAL(token$(1))
          CASE 2
            vdsm(idata%) = VAL(token$(1))
          CASE 3
            vbsm(idata%) = VAL(token$(1))
         END SELECT
         idm(idata%) = VAL(token$(2))
         IF (token$(3) = "") THEN
            weightm(idata%) = 1!
         ELSE
            weightm(idata%) = VAL(token$(3))
         END IF
      END IF
   ELSE
      IF (token$(1) = "*/") THEN skipLoadFlag% = 0
   END IF
WEND
breakLoadLoop:
'---- post-processing of the input ----
IF (idata% > 0) THEN ndata% = idata% ELSE CALL errmsg("No measured
data.")
IF (iP% - iX% <> 1) THEN CALL errmsg("Parameters or variables
missing.")
nX% = iX%: sP% = iP%: nP% = nXP%
'---- define markov chain length ----
'ninitRND% = nX% * 50
'maxnINT% = nX% * 24
ninitRND% = 200
maxnINT% = nX% * 15
CLOSE #2
'---- put initial X into X ----
FOR iX% = 1 TO nXP%
   x(iX%) = initX(iX%)
   rangeX(iX%) = maxX(iX%) - minX(iX%)
NEXT
RETURN

X2V:
   vto = x(iV2iX%(ivto%))
   nsub = x(iV2iX%(insub%)) * 1000000!
   gammac = x(iV2iX%(igammac%))
   uo = x(iV2iX%(iuo%)) * .0001
   theta = x(iV2iX%(itheta%))
   kappa = x(iV2iX%(ikappa%))
   vmax = x(iV2iX%(ivmax%))
   eta = x(iV2iX%(ieta%))
   xj = x(iV2iX%(ixj%))
   ld = x(iV2iX%(ild%))
   tox = x(iV2iX%(itox%))
   nfs = x(iV2iX%(infs%)) * 10000!
   l = x(iV2iX%(il%))
   w = x(iV2iX%(iw%))
RETURN

DataShower:
   '---- show loaded data ----
   WINDOW 3
   PRINT CHR$(13) + CHR$(13) + CHR$(13) + CHR$(13)
   PRINT "-------------------"
   PRINT "Click mouse to pause."
   PRINT "-------------------"
   '---- show parameter info ----
   FOR iX% = 1 TO nX%
      PRINT #1, nameV$(iX2iV%(iX%)), minX(iX%), maxX(iX%), initX(iX%)
      DataShowerLoop1:
         IF (MOUSE(0) <> 0) GOTO DataShowerLoop1
   NEXT
   FOR iP% = sP% TO nP%
      PRINT #1, nameV$(iX2iV%(iP%)), "FIX", initX(iP%)
      DataShowerLoop2:
         IF (MOUSE(0) <> 0) GOTO DataShowerLoop2
   NEXT
   '---- show measured data ----
   PRINT #1, "vgs","vds","vbs","id","weight","ivgdb"
   FOR idata% = 1 TO ndata%
      PRINT #1, vgsm(idata%), vdsm(idata%), vbsm(idata%), idm(idata%),
weightm(idata%), ivgdbm%(idata%)
      DataShowerLoop3:
         IF (MOUSE(0) <> 0) GOTO DataShowerLoop3
   NEXT
   '---- show other critical info ----
   PRINT #1, "nX%=";nX%
RETURN

ResShower:
   '---- show result data ----
   OPEN "mosfit.gr" FOR OUTPUT AS #5
   '---- Vds - Id graph ----
   vgsm(0) = infinity: vdsm(0) = infinity: vbsm(0) = infinity
   igraph% = 0
   '---- calculate vdsmax & vgsmax ----
   vdsmax = 0: vgsmax = 0
   FOR idata% = 1 TO ndata%
      IF (vgsm(idata%) > vgsmax) THEN vgsmax = vgsm(idata%)
      IF (vdsm(idata%) > vdsmax) THEN vdsmax = vdsm(idata%)
   NEXT
   FOR idata% = 1 TO ndata%
      vgs = vgsm(idata%)
      vds = vdsm(idata%)
```

```
        vbs = vbsm(idata%)
        ivgdb% = ivgdbm%(idata%)
        SELECT CASE ivgdb%
        CASE 1
          '---- changing vgs ----
          IF (vds <> vdsm(idata%-1)) OR (vbs <> vbsm(idata%-1)) OR (ivgdb%
<> ivgdbm%(idata%-1)) THEN
            igraph% = igraph% + 1
            FOR vgs = 0 TO vgsmax*1.001 STEP vgsmax/20
              GOSUB Mos3
              PRINT #5, vgs, ids
            NEXT
            PRINT #5,""
          END IF
        CASE 2
          '---- changing vds ----
          IF (vgs <> vgsm(idata%-1)) OR (vbs <> vbsm(idata%-1)) OR (ivgdb%
<> ivgdbm%(idata%-1)) THEN
            igraph% = igraph% + 1
            FOR vds = 0 TO vdsmax*1.001 STEP vdsmax/20
              GOSUB Mos3
              PRINT #5, vds, ids
            NEXT
            PRINT #5,""
          END IF
        END SELECT
      NEXT
      '---- show parameter info ----
      FOR idata% = 1 TO ndata%
        igraph% = igraph% + 1
        PRINT #5, "Line_n_type_marker_label"; igraph%;" 1";" 31";idata%
        IF (ivgdbm%(idata%) = 1) THEN
          PRINT #5, vgsm(idata%)*1! , idm(idata%)
          PRINT #5, vgsm(idata%)*1! , idm(idata%)
        END IF
        IF (ivgdbm%(idata%) = 2) THEN
          PRINT #5, vdsm(idata%)*1!, idm(idata%)
          PRINT #5, vdsm(idata%)*1!, idm(idata%)
        END IF
        PRINT #5, ""
      NEXT
      CLOSE #5
      SetCreate "mosfit.gr","MSWD"
      '---- print out parameters ----
      OPEN "mosfit.par" FOR OUTPUT AS #4
      FOR iX% = 1 TO nX%
        PRINT #4, nameV$(iX2iV%(iX%)), x(iX%)
      NEXT
      PRINT #4, ""
      FOR iP% = sP% TO nP%
        PRINT #4, nameV$(iX2iV%(iP%)), x(iP%)
      NEXT
      PRINT #4, ""
      '---- print out important info ----
      PRINT #4," c=";oldEXTCost;" T=";oldT;"ave=";aveINTACost
      PRINT #4, "E=";iEXT%;"RF";epsRndFac;"s=";sigmaINTACost
      PRINT #4, "iGasp=";iLastGasp%;"iCost=";iCost&
      PRINT #4, "iAccept";iAccept%;"gAccept";gAccept%
      PRINT #4, "exitEXTrelCost";exitEXTrelCost
      CLOSE #4
      SetCreate "mosfit.par","MSWD"
RETURN

Vdsldr:
  '---- Vds - Id graph ----
  WINDOW 3
  INPUT "Vbs"; vbs
  INPUT "Vgsmin, Vgsmax, Vgsstep"; vgsmin, vgsmax, vgsstep
  INPUT "Vdsmin, Vdsmax, Vdsstep"; vdsmin, vdsmax, vdsstep
  FOR vgs = vgsmin TO vgsmax STEP vgsstep
    FOR vds = vdsmin TO vdsmax STEP vdsstep
      GOSUB Mos3
      PRINT #1, vds, ids
    NEXT
    PRINT #1,""
  NEXT
RETURN

Vgsldr:
  '---- Vgs - Id graph ----
  WINDOW 3
  INPUT "Vds"; vds
  INPUT "Vbsmin, Vbsmax, Vbsstep"; vbsmin, vbsmax, vbsstep
  INPUT "Vgsmin, Vgsmax, Vgsstep"; vgsmin, vgsmax, vgsstep
  FOR vbs = vbsmin TO vbsmax STEP vbsstep
    FOR vgs = vgsmin TO vgsmax STEP vgsstep
      GOSUB Mos3
      PRINT #1, vgs, ids
    NEXT
    PRINT #1,""
  NEXT
RETURN

Outer:
  '---- output device select ----
  WINDOW 3
  PRINT "Output to screen(0)"
  INPUT "or new file(1) or append to a file(2) or to printer(2)"; outdev%
  CLOSE #1
  SELECT CASE outdev%
  CASE 0
    OPEN "scrn:" FOR OUTPUT AS #1
  CASE 1
    outfile$ = FILES$(0)
    IF (outfile$ = "") THEN
      CALL errmsg("File not found.")
      RETURN
    ELSE
      OPEN outfile$ FOR OUTPUT AS #1
      PRINT #1,"": PRINT #1,""
    END IF
  CASE 2
    outfile$ = FILES$(1,"TEXT")
    IF (outfile$ = "") THEN
      CALL errmsg("File not found.")
      RETURN
    ELSE
      OPEN outfile$ FOR APPEND AS #1
      PRINT #1,"": PRINT #1,""
    END IF
  CASE 3
    OPEN "lpt1:" FOR OUTPUT AS #1
  CASE ELSE
    OPEN "scrn:" FOR OUTPUT AS #1
  END SELECT
RETURN

Manualer:
  '---- manual fitting ----
  WINDOW 1
  INPUT "maximum Ids for graph"; maxIds
  vgsm(0) = infinity: vdsm(0) = infinity: vbsm(0) = infinity
  '---- calculate vdsmax & vgsmax ----
  vdsmax = 0: vgsmax = 0
  FOR idata% = 1 TO ndata%
    IF (vgsm(idata%) > vgsmax) THEN vgsmax = vgsm(idata%)
    IF (vdsm(idata%) > vdsmax) THEN vdsmax = vdsm(idata%)
  NEXT
  IF (vdsmax > vgsmax) THEN maxVgd = vdsmax ELSE maxVgd = vgsmax
  '---- maxVgd & maxIds given ----
  '---- fitting loop ----
  ManualFitLoop:
    WINDOW 1
    PRINT "Variables"
    FOR iX% = 1 TO nX%
      PRINT nameV$(iX2iV%(iX%));"=";x(iX%);
      INPUT ss$
      IF (ss$ <> "") THEN
        valss = VAL(ss$)
        x(iX%) = valss
      END IF
    NEXT
    PRINT "Parameters"
    FOR iX% = sP% TO nP%
      PRINT nameV$(iX2iV%(iX%));"=";x(iX%);
      INPUT ss$
      IF (ss$ <> "") THEN
        valss = VAL(ss$)
        x(iX%) = valss
      END IF
    NEXT
    FOR iX% = 1 TO nXP%
      PRINT nameV$(iX2iV%(iX%));"=";x(iX%)
    NEXT
    '---- plotting measured data ----
    WINDOW 2
```

```
      marksize% = 2
      FOR idata% = 1 TO ndata%
        ivgdb% = ivgdbm%(idata%)
        igraph% = igraph% + 1
        IF (ivgdbm%(idata%) = 1) THEN vv = vgsm(idata%)
        IF (ivgdbm%(idata%) = 2) THEN vv = vdsm(idata%)
        ii = idm(idata%)
        CALL User2World(vv, ii, wX%, wY%)
        SetRect rec%(0), wX%-marksize%,wY%-
marksize%,wX%+marksize%,wY%+marksize%
        IF (ivgdb% = 1) THEN
          CALL PAINTOVAL(VARPTR(rec%(0)))
        ELSE
          CALL FRAMEOVAL(VARPTR(rec%(0)))
        END IF
      NEXT
      '---- plot calculated point ----
      FOR idata% = 1 TO ndata%
        vgs = vgsm(idata%)
        vds = vdsm(idata%)
        vbs = vbsm(idata%)
        ivgdb% = ivgdbm%(idata%)
        SELECT CASE ivgdb%
        CASE 1
          '---- changing vgs ----
          IF (vds <> vdsm(idata%-1)) OR (vbs <> vbsm(idata%-1)) OR
(ivgdb% <> ivgdbm%(idata%-1)) THEN
            PENSIZE 1,1
            vgs = 0
            GOSUB Mos3
            CALL User2World(vgs, ids, wX%, wY%)
            MOVETO wX%, wY%
            FOR vgs = 0 TO vgsmax*1.001 STEP vgsmax/20
              GOSUB Mos3
              CALL User2World(vgs, ids, wX%, wY%)
              LINETO wX%, wY%
            NEXT
          END IF
        CASE 2
          '---- changing vds ----
          IF (vgs <> vgsm(idata%-1)) OR (vbs <> vbsm(idata%-1)) OR
(ivgdb% <> ivgdbm%(idata%-1)) THEN
            PENSIZE 2,2
            vds = 0
            GOSUB Mos3
            CALL User2World(vds, ids, wX%, wY%)
            MOVETO wX%, wY%
            FOR vds = 0 TO vdsmax*1.001 STEP vdsmax/20
              GOSUB Mos3
              CALL User2World(vds, ids, wX%, wY%)
              LINETO wX%, wY%
            NEXT
          END IF
        END SELECT
      NEXT
      MOVETO 5,20
      INPUT "Try Again(0) or Exit(1)"; ss$
      IF (ss$ = "1") THEN GOTO BreakManualFitLoop
      GOTO ManualFitLoop
    BreakManualFitLoop:
RETURN

Contourer:
  '---- Contour output ----
  ContourLoop:
    WINDOW 3
    FOR iX% = 1 TO nXP%
      PRINT nameV$(iX2iV%(iX%));"=";x(iX%);
      INPUT ss$
      IF (ss$ <> "") THEN x(iX%) = VAL(ss$)
    NEXT
    FOR iX% = 1 TO nXP%
      PRINT "var#=";iX%;"  ";nameV$(iX2iV%(iX%));"=";x(iX%)
    NEXT
    '---- select variables ----
    ilogXc% = 0: ilogYc% = 0: ilogZc% = 0
    iXx% = 1: PRINT "X var ";nameV$(iX2iV%(iXx%));: INPUT ss$: IF (ss$ <>
"") THEN iXx% = VAL(ss$)
    INPUT "linear or log(1)"; ss$: IF (ss$ = "1") THEN ilogXc% = 1
    minXc = minX(iXx%): PRINT "minX";minXc;: INPUT ss$: IF (ss$ <> "")
THEN minXc = VAL(ss$)
    maxXc = maxX(iXx%): PRINT "maxX";maxXc;: INPUT ss$: IF (ss$ <> "")
THEN maxXc = VAL(ss$)
    IF (ilogXc% = 1) THEN stepXc = EXP(LOG(maxXc/minXc)/10) ELSE
stepXc = rangeX(iXx%)/10
    PRINT "stepX";stepXc;: INPUT ss$: IF (ss$ <> "") THEN stepXc =
VAL(ss$)
    iXy% = 2: PRINT "Y var ";nameV$(iX2iV%(iXy%));: INPUT ss$: IF (ss$ <>
"") THEN iXy% = VAL(ss$)
    INPUT "linear or log(1)"; ss$: IF (ss$ = "1") THEN ilogYc% = 1
    minYc = minX(iXy%): PRINT "minY";minYc;: INPUT ss$: IF (ss$ <> "")
THEN minYc = VAL(ss$)
    maxYc = maxX(iXy%): PRINT "maxY";maxYc;: INPUT ss$: IF (ss$ <> "")
THEN maxYc = VAL(ss$)
    IF (ilogYc% = 1) THEN stepYc = EXP(LOG(maxYc/minYc)/10) ELSE
stepYc = rangeX(iXy%)/10
    PRINT "stepY";stepYc;: INPUT ss$: IF (ss$ <> "") THEN stepYc =
VAL(ss$)
    INPUT "Z-axis linear or log(1)"; ss$: IF (ss$ = "1") THEN ilogZc% = 1
    '---- print out input params ----
    PRINT "X var = ";nameV$(iX2iV%(iXx%))
    PRINT "min,max,step,log=";minXc;maxXc;stepXc;ilogXc%
    PRINT "Y var = ";nameV$(iX2iV%(iXy%))
    PRINT "min,max,step,log=";minYc;maxYc;stepYc;ilogYc%
    PRINT "Z-axis log=";ilogZc%
    INPUT "OK or Try Again(1)"; ss$
    IF (ss$ = "1") THEN GOTO ContourLoop
    '---- output xyz file for contour 81 ----
    OPEN "mosfit.cont" FOR OUTPUT AS #4
    IF (ilogXc% <> 0) THEN
      minXc = LOG(minXc)
      maxXc = LOG(maxXc)
      stepXc = LOG(stepXc)
    END IF
    IF (ilogYc% <> 0) THEN
      minYc = LOG(minYc)
      maxYc = LOG(maxYc)
      stepYc = LOG(stepYc)
    END IF
    scalec = (maxXc - minXc) / (maxYc - minYc)
    FOR Xc = minXc TO maxXc STEP stepXc
      IF (ilogXc% <> 0) THEN x(iXx%) = EXP(Xc) ELSE x(iXx%) = Xc
      FOR Yc = minYc TO maxYc STEP stepYc
        IF (ilogYc% <> 0) THEN x(iXy%) = EXP(Yc) ELSE x(iXy%) = Yc
        GOSUB Cost
        IF (ilogZc% <> 0) THEN Zc = LOG(retCost)/LOG(10!) ELSE Zc =
retCost
        PRINT Xc, (Yc-minYc)*scalec+minXc, Zc
        PRINT #4, Xc, (Yc-minYc)*scalec+minXc, Zc
      NEXT
    NEXT
    CLOSE #4
    SetCreate "mosfit.cont","MSWD"
    INPUT "OK or Try Again(1)"; ss$
    IF (ss$ = "1") THEN GOTO ContourLoop
RETURN

SDRunner:
  '---- set initial T & X & Cost ----
  GOSUB InitialX
  GOSUB Cost
  oldEXTCost = retCost
  GOSUB InitialT
  oldT = T
  iCost& = 0
  oldEXTCost = retCost
  GOSUB EXTReport
  '---- save oldEXTX for optEXTX, & optINTX ----
  FOR iX% = 1 TO nX%
    optEXTX(iX%) = x(iX%)
    optINTX(iX%) = x(iX%)
  NEXT
  optEXTCost = oldEXTCost
  optINTCost = oldEXTCost
  iEXT% = 0
  EXTLoop:
    '---- count-up loop counter ----
    iEXT% = iEXT% + 1
    '---- initialize random generator ----
    'RANDOMIZE rseed%
    RANDOMIZE TIMER
    iINT% = 0
    sumINTACost = 0: sumINTACost2 = 0
    iAccept% = 0: gAccept% = 0
    INTLoop:
      inner% = 1
```

```
'---- internal loop with same T ----
iINT% = iINT% + 1
'---- generate new X and calculate cost ----
GOSUB GenerateX
'---- calculate cost ----
GOSUB Cost
'---- check accept or not ----
GOSUB Accept
'GOSUB Tpoint
GOSUB Xpoint
IF (retAccept% = 1) THEN
 '---- accepted ----
 GOSUB UpdateX
 oldINTCost = retCost
 IF (iGenerateXCalled% = 0) THEN
  iAccept% = iAccept% + 1
  histINTACost(iAccept%) = retCost
 ELSE
  gAccept% = gAccept% + 1
 END IF
 '---- save current status if optimal ----
 IF (retCost < optINTCost) THEN
  optINTCost = retCost
  FOR iX% = 1 TO nX%
   optINTX(iX%) = x(iX%)
  NEXT
 END IF
 'GOSUB Tpoint
ELSE
 GOSUB ResumeOldX
END IF
'---- exit INT loop? ----
GOSUB ExitINTLoop
IF (retExitINTLoop = 1) THEN GOTO BreakINTLoop
GOTO INTLoop
BreakINTLoop:
'---- post-process of INT loop ----
inner% = 0
nINT% = iINT%
'GOTO JumpINTGreedy
'---- resume optINTX since it is minimal ----
oldINTCost = optINTCost
FOR iX% = 1 TO nX%
 x(iX%) = optINTX(iX%)
NEXT
JumpINTGreedy:
oldEXTCost = oldINTCost
'---- update optEXTX if this is optimal up to now ----
IF (oldEXTCost < optEXTCost) THEN
 optEXTCost = oldEXTCost
 FOR iX% = 1 TO nX%
  optEXTX(iX%) = x(iX%)
 NEXT
END IF
'GOTO JumpEXTGreedy
'---- resume optEXTX if current Cost is not optimal ----
IF (oldEXTCost > optEXTCost) THEN
 oldEXTCost = optEXTCost
 FOR iX% = 1 TO nX%
  x(iX%) = optEXTX(iX%)
 NEXT
END IF
JumpEXTGreedy:
retCost = oldEXTCost
'GOSUB Tpoint
histEXTCost(iEXT%) = oldEXTCost
'---- exit EXT loop? ----
GOSUB ExitEXTLoop
IF (retExitEXTLoop% = 1) THEN
 IF (iLastGasp% >= 8) THEN
  GOTO breakEXTLoop
 ELSE
  iLastGasp% = iLastGasp% + 1
  '---- resume optEXTX if current Cost is not optimal ----
  IF (oldEXTCost > optEXTCost) THEN
   oldEXTCost = optEXTCost
   FOR iX% = 1 TO nX%
    x(iX%) = optEXTX(iX%)
   NEXT
  END IF
 END IF
ELSE
 IF (histEXTCost(iEXT%) < histEXTCost(iEXT%-1)) THEN
  iLastGasp% = 0
 END IF
END IF
'---- update Temp ----
GOSUB UpdateT
'---- update epsilon for random part ----
GOSUB UpdateEpsRndFac
GOSUB EXTReport
oldT = T
GOTO EXTLoop
breakEXTLoop:
'---- post-process of EXT loop ----
GOSUB ResShower
RETURN

Accept:
'---- decide accept or reject using Boltzmann dist. ----
deltaINTCost = retCost - oldINTCost
IF (deltaINTCost < 0) THEN
 retAccept% = 1
ELSE
 boltzmann = EXP(- deltaINTCost / T)
 IF (RND(1) < boltzmann) THEN
  retAccept% = 1
 ELSE
  retAccept% = 0
 END IF
END IF
RETURN

InitialX:
'---- initialize X ----
FOR iX% = 1 TO nX%
 x(iX%) = initX(iX%)
 oldEXTX(iX%) = initX(iX%)
NEXT
RETURN

InitialT:
'---- try random search ninitRND times and guess initial T ----
minCost = infinity
maxCost = -infinity
sumCost = 0: sumCost2 = 0
FOR initRND% = 1 TO ninitRND%
 '---- random generation of X ----
 FOR iX% = 1 TO nX%
  nameVc$ = nameV$(iX2iV%(iX%))
  IF (nameVc$ = "nsub") OR (nameVc$ = "vmax") THEN
   '---- random generation in log space for nsub and vmax ----
   logminX = LOG(minX(iX%))
   logmaxX = LOG(maxX(iX%))
   lograngeX = logmaxX - logminX
   deltafacX = RND(1) * lograngeX
   deltafacX = EXP(deltafacX)
   x(iX%) = minX(iX%) * deltafacX
  ELSE
   deltaX = RND(1) * rangeX(iX%)
   x(iX%) = minX(iX%) + deltaX
  END IF
 NEXT
 GOSUB Cost
 'initialTFlag% = 1
 'GOSUB Xpoint
 'initialTFlag% = 0
 IF (minCost > retCost) THEN
  '---- this is the best cost, so update initial X ----
  minCost = retCost
  FOR iX% = 1 TO nX%
   initX(iX%) = x(iX%)
  NEXT
 END IF
 '---- if this is the worst cost, update maxCost ----
 IF (maxCost < retCost) THEN minCost = retCost
 '---- calculate sum's ----
 sumCost = sumCost + retCost
 sumCost2 = sumCost2 + retCost * retCost
NEXT
'---- calculate aveCost, sigmaCost ----
N = ninitRND%
aveCost = sumCost / N
sigmaCost = SQR((sumCost2 - N * aveCost * aveCost) / (N-1))
'---- initial T = k * sigmaCost (ICCAD'86) ----
initT = initTK * sigmaCost
```

```
initCost = retCost
'PRINT #1, "initT";initT;"initCost";initCost
T = initT
'---- choose initial X as the minimum Cost X's if it is less than given initX ----
IF (minCost < oldEXTCost) THEN
  FOR iX% = 1 TO nX%
    x(iX%) = initX(iX%)
  NEXT
  oldEXTCost = minCost
  oldINTCost = minCost
ELSE
  FOR iX% = 1 TO nX%
    x(iX%) = oldEXTX(iX%)
  NEXT
  oldEXTCost = oldEXTCost
  oldINTCost = oldEXTCost
END IF
RETURN

GenerateX:
  IF (iEXT% > minnEXT%) THEN
    iGenerateXCalled% = 1 - iGenerateXCalled%
  ELSE
    iGenerateXCalled% = 0
  END IF
  IF (iGenerateXCalled% = 1) THEN
    '---- if icalled% = 1 then gradient ----
    '---- which X is moved ----
    iXv% = 1 + INT((nX% - .00001) * RND(1))
    nameVc$ = nameV$(iX2iV%(iXv%))
    IF (nameVc$ = "nsub") OR (nameVc$ = "vmax") THEN ilogXg% = 1 ELSE
ilogXg% = 0
      '---- gradient generation in log space for nsub and vmax ----
    IF (ilogXg% = 1) THEN
      minXg = LOG(minX(iXv%)): maxXg = LOG(maxX(iXv%)): Xg =
LOG(x(iXv%))
    ELSE
      minXg = minX(iXv%): maxXg = maxX(iXv%): Xg = x(iXv%)
    END IF
    rangeXg = maxXg - minXg
    '---- choose DX value ----
    DX = rangeXg * .00001
    '---- find Xopt by fitting quadratic form ----
    f0 = oldINTCost
    IF (ilogXg% = 0) THEN x(iXv%) = Xg + DX ELSE x(iXv%) = EXP(Xg + DX)
    GOSUB Cost
    fplus = retCost
    IF (ilogXg% = 0) THEN x(iXv%) = Xg - DX ELSE x(iXv%) = EXP(Xg - DX)
    GOSUB Cost
    fminus = retCost
    concave = fplus + fminus - 2 * f0
    IF (concave > 0) THEN
      '---- f" > 0 ----
      deltaXg = - DX / 2 * (fplus - fminus) / concave
      '---- limit up to limitDeltaXg ----
      IF (ABS(deltaXg) > rangeXg * .2) THEN deltaXg = SGN(deltaXg) *
rangeXg * .2
    ELSE
      limitDeltaXg = rangeXg * .03
      '---- f" <= 0 ----
      IF (fplus = fminus) THEN
        '---- f' = 0 ----
        deltaXg = 0
      ELSE
        iGenerateXLoop0% = 0
        GenerateXLoop0:
        IF (fplus > fminus) THEN
          '---- f' > 0 ----
          deltaXg = -limitDeltaXg
        ELSE
          '---- f' < 0 ----
          deltaXg = limitDeltaXg
        END IF
        IF (ilogXg% = 0) THEN x(iXv%) = Xg + deltaXg ELSE x(iXv%) =
EXP(Xg + deltaXg)
        '---- x 1/2 loop ----
        GOSUB Cost
        iGenerateXLoop0% = iGenerateXLoop0% + 1
        iInitialTFlag% = 1
        GOSUB Xpoint
        iInitialTFlag% = 0
        IF (retCost > f0) AND (iGenerateXLoop0% <= 2) THEN
          limitDeltaXg = limitDeltaXg / 2
          GOTO GenerateXLoop0
        END IF
      END IF
    END IF
    '---- update X and if X is out of range, then pull back ----
    IF (ilogXg% = 0) THEN x(iXv%) = Xg + deltaXg ELSE x(iXv%) = EXP(Xg +
deltaXg)
    IF (x(iXv%) < minX(iXv%)) THEN x(iXv%) = minX(iXv%)
    IF (x(iXv%) > maxX(iXv%)) THEN x(iXv%) = maxX(iXv%)
  ELSE
    '---- if icalled% = 0 then, random generation ----
    FOR iX% = 1 TO nX%
      nameVc$ = nameV$(iX2iV%(iX%))
      IF (nameVc$ = "nsub") OR (nameVc$ = "vmax") THEN
        '---- random generation in log space for nsub and vmax ----
        logminX = LOG(minX(iX%))
        logmaxX = LOG(maxX(iX%))
        lograngeX = logmaxX - logminX
        GenerateXLoop1:
          GOSUB GaussRnd: rndNum = retGaussRnd
          deltafacX = rndNum * lograngeX / 3.1 * epsRndFac
          deltafacX = EXP(deltafacX)
          '---- if X is out of range, then pull back ----
          IF (x(iX%) * deltafacX < minX(iX%)) THEN GenerateXLoop2
          IF (x(iX%) * deltafacX > maxX(iX%)) THEN GenerateXLoop2
        x(iX%) = x(iX%) * deltafacX
      ELSE
        GenerateXLoop2:
          'GOSUB LorentzRnd: rndNum = retLorentzRnd
          GOSUB GaussRnd: rndNum = retGaussRnd
          deltaX = rndNum * rangeX(iX%) / 3.1 * epsRndFac
          '---- if X is out of range, then pull back ----
          IF (x(iX%) + deltaX < minX(iX%)) THEN GenerateXLoop2
          IF (x(iX%) + deltaX > maxX(iX%)) THEN GenerateXLoop2
        x(iX%) = x(iX%) + deltaX
      END IF
    NEXT
  END IF
RETURN

UpdateEpsRndFac:
  '---- update epsilon for random part ----
  epsRndFac = epsRndFac * (T / oldT) ^ .75
  IF (iLastGasp% >= 1) THEN
    epsRndFac = SQR(T / initT)
  ELSE
    IF (epsRndFac <= .05) THEN epsRndFac = .05
  END IF
RETURN

UpdateEpsRndFacOld:
  '---- update epsilon for random part ----
  acceptRatio = iAccept% / nINT%
  IF (iLastGasp% = 1) THEN
    IF (epsRndFac >= .5) THEN epsRndFacUp% = 0 ELSE epsRndFacUp%
= 1
  END IF
  IF (iLastGasp% >=1) THEN
    IF (epsRndFacUp% = 0) THEN
      epsRndFac = epsRndFac * .75
    ELSE
      epsRndFac = epsRndFac * 1.5
    END IF
    IF (epsRndFac > 1) THEN
      epsRndFac = .75
      epsRndFacUp% = 0
    ELSE
      IF (epsRndFac < .1) THEN
        'epsRndFac = .15
        epsRndFacUp% = 1
      END IF
    END IF
  ELSE
    IF (iEXT% > minnEXT%) THEN acceptRatio = acceptRatio * 2
    'IF (acceptRatio < .5) THEN epsRndFac = epsRndFac *
SQR(1/factorUpdateTLB)
    'IF (acceptRatio < .5) THEN epsRndFac = epsRndFac * 2^(1/nX%)
    epsRndFac = epsRndFac * SQR(T / oldT)
    IF (epsRndFac > 1) THEN epsRndFac = 1
  END IF
RETURN

UpdateX:
```

```
    FOR iX% = 1 TO nX%
      oldX(iX%) = x(iX%)
    NEXT
  RETURN

  ResumeOldX:
    FOR iX% = 1 TO nX%
      x(iX%) = oldX(iX%)
    NEXT
  RETURN

  ExitINTLoop:
    '---- INT loop exit condition ----
    retExitINTLoop = 0
    IF (iINT% >= maxnINT%) THEN
      '---- if EXT loop count exceed limit, simply exit INT loop ----
      retExitINTLoop = 1
      RETURN
    END IF
  RETURN

  ExitEXTLoop:
    '---- EXT loop exit condition, frozen condition ----
    retExitEXTLoop% = 0
    IF (iEXT% > maxnEXT%) THEN
    '---- if EXT loop count exceed limit, simply exit EXT loop ----
      retExitEXTLoop% = 1
      RETURN
    END IF
    '---- if iEXT% < minnEXT%, continue ----
    IF (iEXT% > minnEXT%) THEN
     GOTO JumpHistoryEval
      aveCost = 0
      minCost = infinity
      maxCost = -infinity
      '---- calculate average, max, and min of recent Cost ----
      FOR iP% = iEXT% - 1 TO iEXT%
        hCost = histEXTCost(iP%)
        aveCost = aveCost + hCost
        IF (hCost > maxCost) THEN maxCost = hCost
        IF (hCost < minCost) THEN minCost = hCost
      NEXT
      aveCost = aveCost / 2
      '---- exit condition using ave and min and max ----
      exitEXTrelCost = (maxCost - minCost) / (ABS(aveCost)+infinitesmal)
      IF (exitEXTrelCost < epsExitEXTrelCost) THEN retExitEXTLoop% = 1
    JumpHistoryEval:
      delCost = histEXTCost(iEXT%) - histEXTCost(iEXT% - 1)
      exitEXTrelCost = ABS(delCost) /
(ABS(histEXTCost(iEXT%))+infinitesmal)
      IF (exitEXTrelCost < epsExitEXTrelCost) THEN retExitEXTLoop% = 1
      IF (delCost >= 0) THEN retExitEXTLoop% = 1
    END IF
  RETURN

  UpdateT:
    '---- update T by ICCAD86 ----
    IF (iLastGasp% >= 1) THEN
      IF(iLastGasp% >= 3) THEN
        T = .5 * T
      ELSE
        T = 1.6 * T
      END IF
      IF (T > initT) THEN T = .1 * T
    ELSE
      N% = iAccept%
      IF (N% >= 3) THEN
        '---- if acceptance ratio is high enough ----
        sumINTACost = 0
        FOR iU% = 1 TO N%
          sumINTACost = sumINTACost + histINTACost(iU%)
        NEXT
        aveINTACost = sumINTACost / N%
        FOR iU% = 1 TO N%
          sigmaINTACost = (histINTACost(iU%) - aveINTACost) ^ 2
        NEXT
        sigmaINTACost = SQR(sigmaINTACost / (N%-1))
        IF (sigmaINTACost = 0) THEN sigmaINTACost=infinitesmal
        factorUpdateT = EXP(-updateTLambda * T / sigmaINTACost)
        IF (factorUpdateT < factorUpdateTLB) THEN factorUpdateT =
factorUpdateTLB
        T = factorUpdateT * T
      ELSE
```

```
        T = .9 * T
        sigmaINTACost = 9999!
        aveINTACost = 9999!
      END IF
    END IF
  RETURN

  Cost:
    '---- calculating cost ----
    iCost& = iCost& + 1
    '---- summation over measured data ----
    retCost = 0
    'WINDOW 3
    FOR idata% = 1 TO ndata%
      vgs = vgsm(idata%)
      vds = vdsm(idata%)
      vbs = vbsm(idata%)
      GOSUB Mos3
      retCost = retCost + ABS(idm(idata%) - ids) * weightm(idata%)
    NEXT
    retCost = retCost + 1E-10
  RETURN

  GaussRnd:
    '---- gaussian distribution (see p.217 of NR) ----
    IF (iGaussCalled% = 0) THEN
      gaussR = 2!
      WHILE (gaussR >= 1!)
        gaussV1 = 2! * RND(1) - 1!
        gaussV2 = 2! * RND(1) - 1!
        gaussR = gaussV1 * gaussV1 + gaussV2 * gaussV2
      WEND
      gaussFac = SQR(-2 * LOG(gaussR) / gaussR)
      gaussSet = gaussV1 * gaussFac
      iGaussCalled% = 1
      retGaussRnd = gaussV2 * gaussFac
    ELSE
      retGaussRnd = gaussSet
      iGaussCalled% = 0
    END IF
  RETURN

  LorentzRnd:
    '---- Lorentzian distribution (see p.217 of NR) ----
    LorentzLoop:
      lorentzR = RND(1)
      IF (lorentzR < 3.141592 * 1.1 / 2) AND (lorentzR > 3.141592 * .9 / 2)
THEN GOTO LorentzLoop
    retLorentzRnd = TAN(3.141592 * lorentzR)
  RETURN

  EXTReport:
    '---- EXTernal loop report ----
    WINDOW 1
    PRINT #1," c=";oldEXTCost;" T=";oldT;"ave=";aveINTACost
    PRINT #1, "iE=";iEXT%;"RF";epsRndFac;"s=";sigmaINTACost
    PRINT #1, "iGasp=";iLastGasp%;"iCost=";iCost&
    PRINT #1, "nAccept";iAccept%;"gAccept";gAccept%
    PRINT #1, "relCost";exitEXTrelCost
    GOSUB XReport
   GOTO EXTReportBreak
    '---- show measured data ----
    WINDOW 2
    PRINT #1, "vgs";" vds";" vbs";" idm";" idc"
    FOR idata% = 1 TO ndata%
      vgs = vgsm(idata%)
      vds = vdsm(idata%)
      vbs = vbsm(idata%)
      GOSUB Mos3
      PRINT #1, vgsm(idata%); vdsm(idata%); vbsm(idata%); idm(idata%); ids
      EXTReportLoop:
        IF (MOUSE(0) <> 0) THEN GOTO EXTReportLoop
    NEXT
    GOSUB ResShower
  EXTReportBreak:
    IF (MOUSE(0) <> 0) THEN GOSUB ResShower
  RETURN

  XReport:
    '---- print out X values ----
    WINDOW 1
    FOR iX% = 1 TO nXP%
      PRINT #1, nameV$(iX2iV%(iX%)); x(iX%);
```

```
    IF (iX% MOD 1 = 0) THEN PRINT ""
   NEXT
   PRINT #1, ""
RETURN

Xpoint:
   '---- point a circle ----
   WINDOW 2
   SHOWPEN
   IF (iXpointCalled% = 0) THEN
      '---- write frame for the first time ----
      PENSIZE 1,1
      SetRect rect%(0),0,0,280,280
      CALL FRAMERECT(VARPTR(rect%(0)))
      iXpointCalled% = iXpointCalled% + 1
   END IF
   IF (initialTFlag% = 1) THEN marksize% = 1 ELSE marksize% = 2
   wX% = 280 * (x(1) - minX(1)) / rangeX(1)
   wY% = 280 * (1 - LOG(x(2)/minX(2)) / LOG(maxX(2)/minX(2)))
   SetRect rect%(0), wX%-marksize%,wY%-
marksize%,wX%+marksize%,wY%+marksize%
   IF (retAccept% = 1) THEN
      IF (iGenerateXCalled% = 1) THEN
         CALL PAINTRECT(VARPTR(rect%(0)))
      ELSE
         CALL PAINTOVAL(VARPTR(rect%(0)))
      END IF
   ELSE
      IF (iGenerateXCalled% = 1) THEN
         CALL FRAMERECT(VARPTR(rect%(0)))
      ELSE
         CALL FRAMEOVAL(VARPTR(rect%(0)))
      END IF
   END IF
   HIDEPEN
RETURN

Tpoint:
   '---- point a circle ----
   WINDOW 2
   marksize% = 2
   rect%(1) = 50 * ABS(LOG(oldT / initT)) / LOG(10) - marksize%
   rect%(0) = 50 * ABS(LOG(ABS(retCost / initCost))) / LOG(10) - marksize%
   rect%(3) = 50 * ABS(LOG(oldT / initT)) / LOG(10) + marksize%
   rect%(2) = 50 * ABS(LOG(ABS(retCost / initCost))) / LOG(10) + marksize%
   IF (iGenerateXCalled% = 1) THEN
      CALL PAINTOVAL(VARPTR(rect%(0)))
   ELSE
      CALL FRAMEOVAL(VARPTR(rect%(0)))
   END IF
RETURN

Mos3:
   GOSUB X2V
   IF (tox <= 0) THEN CALL errmsg("tox negative or zero in Mos3")
   coxpu = epsox / tox
   phif = vtherm * LOG(nsub / ni)
   IF (phif <= 0) THEN
      errormsg$ = "phif < 0 ("+STR$(nsub)+") in Mos3"
      CALL errmsg(errormsg$)
   END IF
   phif2 = phif + phif
   leff = l - ld - ld
   IF (leff <= .1 * l) THEN leff = .1 * l
   phibs = phif2 - vbs
   IF (phibs <= 0) THEN phibs = infinitesmal
   sqphbs = SQR(phibs)

   '---- THRESHOLD VOLTAGE ----
   pfn = delta * pi * epssi * .5 * coxpu * w
   xd = SQR (2 * epssi / q / nsub)
   wp = xd * sqphbs
   wc = .0631353 * xj + .8013292# * wp + .01110777# * wp * wp / xj
   wpxj = wp / (xj + wp)
   fs = 1 - xj / leff * ((ld + wc) / xj * SQR (1 - wpxj * wpxj) - ld / xj)
   IF (fs <= 0) THEN
      errormsg$ = "fs negative in Mos3. Use larger nsub.
nsub="+STR$(nsub)+"  xd="+STR$(xd)
      CALL errmsg(errormsg$)
   END IF
   sigma = eta * 8.15E-22 / coxpu / leff / leff / leff
   vfb = vto - phif2 - gammac * SQR (phif2)
   vth = vfb + phif2 - sigma * vds + gammac * fs * sqphbs + pfn * phibs

   '---- ON VOLTAGE ----
   cd = coxpu * (gammac * fs / 2 / sqphbs + pfn / 2)
   mos3N = 1 + (q * nfs + cd) / coxpu
   von = vth + mos3N * vtherm

   '---- NOMINAL GATE VOLTAGE ----
   IF (vgs > von) THEN vgsx = vgs ELSE vgsx = von

   '---- SATURATION VOLTAGE ----
   us = uo / (1 + theta * (vgsx - vth))
   fb = gammac * fs / 4 / sqphbs + pfn
   arga = (vgsx - vth + .0000001) / (1 + fb)
   argb = vmax * leff / us
   vdsat = 2 * arga * argb
   argc = arga + argb + SQR(arga * arga + argb * argb)
   IF (argc <= 0) THEN argc = infinitesmal
   vdsat = vdsat / argc
   IF (vds > vdsat) THEN vdsx = vdsat ELSE vdsx = vds
   ueff = us / (1 + us * vdsx / vmax / leff)

   '---- LINEAR REGION ----
   IF (vds <= vdsat) THEN
      ids = (w / leff) * ueff * coxpu
      ids = ids * (vgsx - vth - (1 + fb) / 2 * vds) * vds
   '---- SATURATED REGION ----
   ELSE
      ep = vmax / us * (1 + vmax * leff / us / vdsat)
      arga = ep * xd * xd / 2
      argb = kappa * xd * xd * (vds - vdsat)
      argc = SQR(arga * arga + argb) + arga
      IF (argc <= 0) THEN argc = infinitesmal
      delxl = argb / argc
      '---- PUNCHTHROUGH APPROX. ----
      IF (delxl > .5 * leff) THEN
         delxl = leff * (1 - leff / 4 / delxl)
      END IF
      ids = (w / (leff - delxl)) * ueff * coxpu
      ids = ids * (vgsx - vth - (1 + fb) / 2 * vdsat) * vdsat
   END IF

   '---- SUBTHRESHOLD REGION ----
   IF (vgs < von) THEN
      vgsVonVthermn = (vgs - von) / vtherm / mos3N
      IF (vgsVonVthermn > -20) THEN vgsvon = vgsVonVthermn ELSE
vgsvon = -20
      idsexp = ids * EXP(vgsvon)
      IF (idsexp > 1E-15) THEN ids = idsexp ELSE ids = 1E-15
   END IF
RETURN

'---- parsing ----
SUB parse(s$, token$(), ntoken%, errorFlag%) STATIC
   errorFlag% = 0
   s$ = s$ + " "
   '---- clear token array ----
   FOR itoken% = 1 TO UBOUND(token$)
      token$(itoken%) = ""
   NEXT
   itoken% = 0
   WHILE 1
      '---- delete leading tabs and spaces ----
      T$ = LEFT$(s$, 1)
      WHILE ((T$ = " ") OR (T$ = CHR$(34)))
         s$ = MID$(s$, 2)
         T$ = LEFT$(s$, 1)
      WEND
      '---- searching tab or space whichever comes first ----
      inspc% = INSTR(s$, " ")
      intab% = INSTR(s$, CHR$(34))
      IF (inspc% = 0) THEN inspc% = 1000
      IF (intab% = 0) THEN intab% = 1000
      idelim% = inspc%
      IF (idelim% > intab%) THEN idelim% = intab%
      IF (idelim% = 1000) GOTO breakParseLoop
      itoken% = itoken% + 1
      'PRINT #1, s$, itoken%: INPUT a
      IF (itoken% >= ntokenMax%) THEN
         errorFlag% = 1
         msg$ = "Too many field ("+STR$(itoken%)+") in line "+STR$(line%)+"."
         CALL errmsg(msg$)
         GOTO breakParseLoop
      END IF
```

```
        '---- recognize token as an entity from the first char to the next blank ----
        tokenS(itoken%) = LEFT$(sS, idelim%-1)
        '---- update line string ----
        s$ = MID$(sS, idelim%+1)
      WEND
      breakParseLoop:
      ntoken% = itoken%
  END SUB

  '---- error message routine ----
  SUB errmsg(errormsgS) STATIC
    WINDOW 5,,(0,0)-(600,400), 4
    TEXTFACE 1
    TEXTSIZE 12
    MOVETO 100,50
    PRINT "Message : ";
    PRINT errormsg$
    BUTTON 1,1,"OK",(190,100)-(250,120),1
    errmsgLoop:
      WHILE DIALOG(0) <> 1 : WEND
      IF DIALOG(1) <> 1 THEN GOTO errmsgLoop
    WINDOW CLOSE 5
  END SUB

  SUB paramRead(tokenS(), iX%, iP%, iV%) STATIC
    IF (UCASES(tokenS(3)) = "FIX") THEN
      '---- constant parameters ----
      iP% = iP% - 1
      iX2iV%(iP%) = iV%
      iV2iX%(iV%) = iP%
      IF (tokenS(3) = "") THEN initX(iP%) = initV(iV%) ELSE initX(iP%) =
VAL(tokenS(4))
    ELSE
      IF (tokenS(3) = "") THEN minVal = minV(iV%) ELSE minVal =
VAL(tokenS(3))
      IF (tokenS(4) = "") THEN maxVal = maxV(iV%) ELSE maxVal =
VAL(tokenS(4))
      'PRINT "
iX=";iX%;"iV=";nameVS(iV%);"minVal,maxVal";minVal;maxVal;minV(iV%);max
V(iV%)
      IF (minVal = maxVal) THEN
        '---- constant parameters ----
        iP% = iP% - 1
        iX2iV%(iP%) = iV%
        iV2iX%(iV%) = iP%
        initX(iP%) = minVal
      ELSE
        '---- variables ----
        iX% = iX% + 1
        iX2iV%(iX%) = iV%
        iV2iX%(iV%) = iX%
        minX(iX%) = minVal
        maxX(iX%) = maxVal
        IF (tokenS(5) = "") THEN initX(iX%) = initV(iV%) ELSE initX(iX%) =
VAL(tokenS(5))
      END IF
    END IF
END SUB

SUB initializeV(iV%, nameVs$, minVs, maxVs, initVs) STATIC
  '---- initializing V ----
  nameVS(iV%) = nameVs$
  minV(iV%) = minVs: maxV(iV%) = maxVs: initV(iV%) = initVs
END SUB

'---- convert user coord to world coord ----
SUB User2World(x, Y, wX%, wY%) STATIC
  wX% = x / maxVgd * 240 + 20
  wY% = - Y / maxIds * 240 + 260
END SUB
```

```
'---- Minimization with simulated annealing ----
'---- initailize I/O ----
'----minnEXT is changed to 10 from 5
'----Lorentzian
OPEN "scrn:" FOR OUTPUT AS #1
OPEN "funcSD.res" FOR OUTPUT AS #2

FOR iproblem% = 1 TO 18
  P% = iproblem%
  'IF (P% <> 4) AND (P% <> 5) AND (P% <> 12) AND (P% <> 18) THEN GOTO
BreakProblemLoop
FOR icount% = 1 TO 10
Windower:
'---- initialize window 1 ----
WINDOW 2,"Graphics Window", (200,20)-(480, 300),1
WINDOW 1,"Text Window", (0,20)-(200, 300),1
TEXTSIZE 8
'INPUT "iproblem"; iproblem%

OPTION BASE 0
SELECT CASE iproblem%
CASE 0
  nX% = 2
CASE 1
  nX% = 1
CASE 2
  nX% = 1
CASE 3 TO 7, 13
  nX% = 2
CASE 8, 14
  nX% = 3
CASE 9, 15
  nX% = 4
CASE 10, 16
  nX% = 5
CASE 11
  nX% = 8
CASE 12
  nX% = 10
CASE 17
  nX% = 6
CASE 18
  nX% = 7
CASE ELSE
END SELECT
maxnX% = 14
'ninitRND% = nX% * 50
ninitRND% = 200
maxnINT% = nX% * 24
IF (maxnINT% < 100) THEN maxnINT% = 100
maxnEXT% = 100
DIM SHARED oldX(maxnX%), newX(maxnX%), oldEXTX(maxnX%)
DIM SHARED  optX(maxnX%)
DIM SHARED minX(maxnX%), maxX(maxnX%), initX(maxnX%),
X(maxnX%), rangeX(maxnX%)
DIM SHARED histINTACost(maxnINT%)

Initializer:
  iGreedy% = 1
  infinity = 1E+20
  infinitesmal = 1E-20
  initTK = .2
  updateTLambda = .7
  factorUpdateTLB = .6
  minnEXT% = 10
  iGaussCalled% = 0
  iGenerateXCalled% = 0
  epsRndFac = 1!
  pi = 3.141592

  SELECT CASE iproblem%
  CASE 0
  FOR iX% = 1 TO nX%
    minX(iX%) = -1!: maxX(iX%) = 1!
    rangeX(iX%) = maxX(iX%) - minX(iX%)
    'initX(iX%) = (maxX(iX%) + minX(iX%)) / 2
    initX(iX%) = 1!
  NEXT
  CASE 1 TO 15
  FOR iX% = 1 TO nX%+1
    minX(iX%) = -10: maxX(iX%) = 10
    rangeX(iX%) = maxX(iX%) - minX(iX%)
    initX(iX%) = 0
  NEXT
  CASE 16 TO 18
  FOR iX% = 1 TO nX%+1
    minX(iX%) = -5: maxX(iX%) = 5
    rangeX(iX%) = maxX(iX%) - minX(iX%)
    initX(iX%) = 0
  NEXT
  CASE ELSE
  END SELECT
  oldINTCost = infinity
  oldEXTCost = infinity

GOSUB SDRunner
 'INPUT a
ERASE oldX, newX, oldEXTX, optX
ERASE minX, maxX, initX, X, rangeX
ERASE histINTACost
NEXT
BreakProblemLoop:
NEXT
SetCreate "funcSD.res", "MSWD"
CLOSE
END

SDRunner:
  '---- set initial T & X & Cost ----
  GOSUB InitialX
  GOSUB Cost
  oldEXTCost = retCost
  optCost = retCost
  GOSUB InitialT
  iCost& = 0: iLastGasp% = 0
  oldEXTCost = retCost
  oldINTCost = retCost
  EXTCost = retCost
  GOSUB EXTReport
  '---- save oldEXTX for optX ----
  optCost = retCost
  FOR iX% = 1 TO nX%
    optX(iX%) = X(iX%)
  NEXT
  EXT% = 0
  EXTLoop:
    '---- count-up loop counter ----
    iEXT% = iEXT% + 1
    '---- initialize random generator ----
    RANDOMIZE TIMER
    iINT% = 0
    sumINTACost = 0: sumINTACost2 = 0
    iAccept% = 0: gAccept% = 0
    INTLoop:
      inner% = 1
      '---- internal loop with same T ----
      iINT% = iINT% + 1
      '---- generate new X and calculate cost ----
      GOSUB GenerateX
      GOSUB Cost
      '---- check accept or not ----
      GOSUB Accept
      GOSUB Tpoint
      GOSUB Xpoint
      IF (retAccept% = 1) THEN
        '---- accepted ----
        GOSUB UpdateX
        oldINTCost = retCost
        IF (iGenerateXCalled% = 0) THEN
          iAccept% = iAccept% + 1
          histINTACost(iAccept%) = retCost
        ELSE
          gAccept% = gAccept% + 1
        END IF
        '---- save current status if optimal ----
        IF (retCost < optCost) THEN
          optCost = retCost
          FOR iX% = 1 TO nX%
            optX(iX%) = X(iX%)
          NEXT
        END IF
      ELSE
        GOSUB ResumeOldX
      END IF
      '---- exit INT loop? ----
      GOSUB ExitINTLoop
```

```
        IF (retExitINTLoop% = 1) THEN GOTO BreakINTLoop
      GOTO INTLoop
      BreakINTLoop:
      '---- post-process of INT loop ----
      inner% = 0
      nINT% = iINT%
      EXTCost = oldINTCost
      originalEXTCost = EXTCost
      '---- exit EXT loop? ----
      IF (EXTCost > optCost) AND (iGreedy% = 1) THEN GOSUB
 ResumeOptX
      retCost = EXTCost
      GOSUB ExitEXTLoop
      IF (retExitEXTLoop% = 1) THEN GOTO BreakEXTLoop
      '---- greedy substitution ----
      GOSUB Tpoint
      '---- update Temp and RndFac ----
      GOSUB UpdateT
      GOSUB UpdateEpsRndFac
      GOSUB EXTReport
      oldT = T
      oldEXTCost = EXTCost
    GOTO EXTLoop
    BreakEXTLoop:
    '---- post-process of EXT loop ----
    GOSUB FinalReport
RETURN

ResumeOptX:
    '---- resume optX ----
    EXTCost = optCost
    FOR iX% = 1 TO nX%
      X(iX%) = optX(iX%)
    NEXT
RETURN

InitialX:
    '---- initialize X ----
    FOR iX% = 1 TO nX%
      X(iX%) = initX(iX%)
      oldEXTX(iX%) = initX(iX%)
    NEXT
RETURN

InitialT:
    '---- try random search ninitRND times and guess initial T ----
    maxCost = -infinity
    sumCost = 0: sumCost2 = 0
    FOR initRND% = 1 TO ninitRND%
      '---- random generation of X ----
      FOR iX% = 1 TO nX%
        X(iX%) = minX(iX%) + RND(1) * rangeX(iX%)
      NEXT
      GOSUB Cost
      IF (optCost > retCost) THEN
        '---- this is the best cost, so update initial X ----
        optCost = retCost
        FOR iX% = 1 TO nX%
          initX(iX%) = X(iX%)
        NEXT
      END IF
      '---- if this is the worst cost, update maxCost ----
      IF (maxCost < retCost) THEN optCost = retCost
      '---- calculate sum's ----
      sumCost = sumCost + retCost
      sumCost2 = sumCost2 + retCost * retCost
    NEXT
    '---- calculate aveCost, sigmaCost ----
    N = ninitRND%
    aveCost = sumCost / N
    sigmaCost = SQR((sumCost2 - N * aveCost * aveCost) / (N-1))
    '---- initial T = k * sigmaCost (ICCAD'86) ----
    initT = initTK * sigmaCost
    initCost = retCost
    PRINT #1, "initT";initT;"initCost";initCost
    T = initT
    oldT = T
    '---- choose initial X as the minimum Cost X's if it is less than given initX ----
    IF (optCost < oldEXTCost) THEN
      FOR iX% = 1 TO nX%
        X(iX%) = initX(iX%)
      NEXT
      oldEXTCost = optCost
      oldINTCost = optCost
    ELSE
      FOR iX% = 1 TO nX%
        X(iX%) = oldEXTX(iX%)
      NEXT
      oldEXTCost = oldEXTCost
      oldINTCost = oldEXTCost
    END IF
RETURN

GenerateX:
    IF (EXT% > minnEXT%) THEN
      iGenerateXCalled% = 1 - iGenerateXCalled%
    ELSE
      iGenerateXCalled% = 0
    END IF
    IF (iGenerateXCalled% = 1) THEN
      '---- if icalled% = 1 then gradient ----
      '---- which X is moved ----
      iXv% = 1 + INT((nX% - .00001) * RND(1))
      '---- choose DX value ----
      rangeXg = rangeX(iXv%)
      DX = rangeXg * .00001
      Xg = X(iXv%)
      '---- find Xopt by fitting quadratic form ----
      f0 = oldINTCost
      X(iXv%) = Xg + DX
      GOSUB Cost
      fplus = retCost
      X(iXv%) = Xg - DX
      GOSUB Cost
      fminus = retCost
      concave = fplus + fminus - 2 * f0
      IF (concave > 0) THEN
        '---- f" > 0 ----
        deltaXg = - DX / 2 * (fplus - fminus) / concave
        '---- limit up to limitDeltaXg ----
        IF (ABS(deltaXg) > rangeXg * .1) THEN deltaXg = SGN(deltaXg) *
rangeXg * .1
      ELSE
        '---- f" <= 0 ----
        limitDeltaXg = rangeXg * .005
        oldDeltaXg = 0
        iGenerateXLoop0% = 0
        GenerateXLoop0:
        IF (fplus >= fminus) THEN
          '---- f' >= 0 ----
          deltaXg = -limitDeltaXg
        ELSE
          '---- f' < 0 ----
          deltaXg = limitDeltaXg
        END IF
        X(iXv%) = Xg + deltaXg
        '---- x 2 loop ----
        GOSUB Cost
        iGenerateXLoop0% = iGenerateXLoop0% + 1
        IF (retCost < f0) THEN
          oldDeltaXg = deltaXg
          IF (iGenerateXLoop0% <= 3) THEN
            limitDeltaXg = limitDeltaXg * 2
            GOTO GenerateXLoop0
          END IF
        END IF
        deltaXg = oldDeltaXg
      END IF
      '---- update X and if X is out of range, then pull back ----
      X(iXv%) = Xg + deltaXg
      IF (X(iXv%) < minX(iXv%)) THEN X(iXv%) = minX(iXv%)
      IF (X(iXv%) > maxX(iXv%)) THEN X(iXv%) = maxX(iXv%)
    ELSE
      '---- if icalled% = 0 then, random generation ----
      FOR iX% = 1 TO nX%
        GenerateXLoop2:
          GOSUB LorentzRnd: rndNum = retLorentzRnd / 2
          'GOSUB GaussRnd: rndNum = retGaussRnd
          deltaX = rndNum * rangeX(iX%) / 3.1 * epsRndFac
          '---- if X is out of range, then pull back ----
          IF (X(iX%) + deltaX < minX(iX%)) THEN GenerateXLoop2
          IF (X(iX%) + deltaX > maxX(iX%)) THEN GenerateXLoop2
        X(iX%) = X(iX%) + deltaX
      NEXT
    END IF
RETURN
```

```
Accept:
  '---- decide accept or reject using Boltzmann dist. ----
  deltaINTCost = retCost - oldINTCost
  IF (deltaINTCost < 0) THEN
    retAccept% = 1
  ELSE
    boltzmann = EXP(- deltaINTCost / T)
    IF (RND(1) < boltzmann) THEN
      retAccept% = 1
    ELSE
      retAccept% = 0
    END IF
  END IF
RETURN

UpdateX:
  FOR iX% = 1 TO nX%
    oldX(iX%) = X(iX%)
  NEXT
RETURN

ResumeOldX:
  FOR iX% = 1 TO nX%
    X(iX%) = oldX(iX%)
  NEXT
RETURN

ExitINTLoop:
  '---- INT loop exit condition ----
  retExitINTLoop% = 0
  IF (iINT% >= maxnINT%) THEN
    '---- if EXT loop count exceed limit, simply exit INT loop ----
    retExitINTLoop% = 1
    RETURN
  END IF
RETURN

ExitEXTLoop:
  '---- EXT loop exit condition, frozen condition ----
  retExitEXTLoop% = 0
  IF (iEXT% > maxnEXT%) THEN
  '---- if EXT loop count exceed limit, simply exit EXT loop ----
    retExitEXTLoop% = 1
    RETURN
  END IF
  '---- if iEXT% < minnEXT%, continue ----
  IF (iEXT% < minnEXT%) THEN RETURN
  '---- takes care of LastGasp ----
  delCost = EXTCost - oldEXTCost
  relCost =  delCost / (ABS(EXTCost)+infinitesmal)
  IF (iGreedy% = 1) THEN
    '---- greedy case ----
    IF (relCost < -.02) THEN
      iLastGasp% = 0
    ELSE
      iLastGasp% = iLastGasp% + 1
    END IF
  ELSE
    '---- drift case ----
    IF (EXTCost < optCost) THEN
      iLaspGasp% = 0
    ELSE
      IF (ABS(relCost) <  .001) THEN
        iLastGasp% = iLastGasp% + 1
      ELSE
        IF (iLastGasp% >= 1) THEN iLastGasp% = iLastGasp% + 1
      END IF
    END IF
    '---- resume optX if current Cost is not optimal ----
    IF (iLastGasp% = 4) AND (EXTCost > optCost) THEN GOSUB
ResumeOptX
    IF (EXTCost < optCost) THEN iLastGasp% = 0
  END IF
  '---- exit EXT loop if iLastGasp% and epsRndFac conditions are met ----
  IF (iLastGasp% >= 14) AND (epsRndFac < .099) THEN retExitEXTLoop%
= 1
RETURN

UpdateEpsRndFac:
  '---- update epsilon for random part ----
  IF (iLastGasp% >= 3) THEN
    epsRndFac = (T / initT)
  ELSE
    epsRndFac = (T / initT) ^ .75
    IF (epsRndFac <= .03) THEN epsRndFac = .03
  END IF
RETURN

UpdateT:
  '---- update T by ICCAD86 ----
  IF (iLastGasp% >= 1) THEN
    IF(iLastGasp% >= 4) THEN
      T = .75 * T
    ELSE
      IF (iGreedy% = 1) THEN T = 1.3 * T
    END IF
  ELSE
    '=======
    'ABS(relCost) = aRelCost
    'IF (aRelCost > 1!) THEN T = .5 * T
    'IF (aRelCost > .3) THEN T = .9 * T
    'IF (aRelCost >
    'RETURN
    N% = iAccept%
    IF (N% >= 3) THEN
      '---- if acceptance ratio is high enough ----
      sumINTACost = 0
      FOR iU% = 1 TO N%
        sumINTACost = sumINTACost + histINTACost(iU%)
      NEXT
      aveINTACost = sumINTACost / N%
      FOR iU% = 1 TO N%
        sigmaINTACost = (histINTACost(iU%) - aveINTACost) ^ 2
      NEXT
      sigmaINTACost = SQR(sigmaINTACost / (N%-1))
      IF (sigmaINTACost = 0) THEN sigmaINTACost=infinitesmal
      factorUpdateT = EXP(-updateTLambda * T / sigmaINTACost)
      IF (factorUpdateT < factorUpdateTLB) THEN factorUpdateT =
factorUpdateTLB
      T = factorUpdateT * T
    ELSE
      T = .9 * T
      sigmaINTACost = 9999!
      aveINTACost = 9999!
    END IF
  END IF
RETURN

Cost:
  '---- problems----
  SELECT CASE iproblem%
  CASE 0
    xx = X(1): yy = X(2)
    retCost = xx*xx + 2*yy*yy - .3*COS(3*pi*xx)-.4*COS(4*pi*yy)+.7
  CASE 1
    xx = X(1)
    retCost = xx^6 -15 * xx^4 + 27 * xx^2 + 250
  CASE 2
    xx = X(1)
    CALL G1(xx, retCost)
  CASE 3
    xx = X(1): yy = X(2)
    CALL G1beta(xx, yy, 0!, retCost)
  CASE 4
    xx = X(1): yy = X(2)
    CALL G1beta(xx, yy, .5, retCost)
  CASE 5
    xx = X(1): yy = X(2)
    CALL G1beta(xx, yy, 1!, retCost)
  CASE 6
    xx = X(1): yy = X(2)
    retCost = (4 - 2.1*xx^2 + xx^4/3) * xx^2 + xx*yy + (-4 + 4*yy^2) * yy^2
  CASE 7 TO 9
    GOSUB G2
    retCost = retG2
  CASE 10 TO 12
    GOSUB G3
    retCost = retG3
  CASE 13 TO 18
    GOSUB G4
    retCost = retG4
  CASE ELSE
    PRINT "No corresponding problem": RETURN
  END SELECT
  iCost& = iCost& + 1
```

```
RETURN

G2:
   k2 = 10
   a2 = 1
   y1 = 1 + (X(1) - 1) / 4
   retG2 = k2 * (SIN(pi * y1)) ^ 2
   FOR iGX% = 1 TO nX%-1
      yi = 1 + (X(iGX%) - 1) / 4
      yi1 = 1 + (X(iGX%+1) - 1) / 4
      retG2 = retG2 + (yi-a2)^2 * (1 + k2*(SIN(pi*yi1))^2)
   NEXT
   retG2 = retG2 + (yi1 - a2)^2
   retG2 = pi/nX% * retG2 + .001
RETURN

G3:
   k3 = 10
   a3 = 1
   y1 = X(1)
   retG3 = k3 * (SIN(pi * y1)) ^ 2
   FOR iGX% = 1 TO nX%-1
      yi = X(iGX%)
      yi1 = X(iGX%+1)
      retG3 = retG3 + (yi-a3)^2 * (1 + k3*(SIN(pi*yi1))^2)
   NEXT
   retG3 = retG3 + (yi1 - a3)^2
   retG3 = pi/nX% * retG3 + .001
RETURN

G4:
   k4 = .1: k5 = 1!: a4 = 1: l0 = 3: l1 = 2
   x1 = X(1)
   retG4 = (SIN(pi*l0*x1)) ^ 2
   FOR iGX% = 1 TO nX%-1
      xi = X(iGX%)
      xi1 = X(iGX%+1)
      retG4 = retG4 + (xi-a4)^2 * (1 + k5*(SIN(pi*l0*xi1))^2)
   NEXT
   retG4 = retG4 + (xi1-a4)^2 * (1 + k5*(SIN(pi*l1*xi1))^2)
   retG4 = k4 * retG4 + .001
RETURN

GaussRnd:
   '---- gaussian distribution (see p.217 of NR) ----
   IF (iGaussCalled% = 0) THEN
      gaussR = 2!
      WHILE (gaussR >= 1!)
         gaussV1 = 2! * RND(1) - 1!
         gaussV2 = 2! * RND(1) - 1!
         gaussR = gaussV1 * gaussV1 + gaussV2 * gaussV2
      WEND
      gaussFac = SQR(-2 * LOG(gaussR) / gaussR)
      gaussSet = gaussV1 * gaussFac
      iGaussCalled% = 1
      retGaussRnd = gaussV2 * gaussFac
   ELSE
      retGaussRnd = gaussSet
      iGaussCalled% = 0
   END IF
RETURN

LorentzRnd:
   '---- Lorentzian distribution (see p.217 of NR) ----
   LorentzLoop:
      lorentzR = RND(1)
      IF (lorentzR < 3.141592 * 1.1 / 2) AND (lorentzR > 3.141592 * .9 / 2)
THEN GOTO LorentzLoop
   retLorentzRnd = TAN(3.141592 * lorentzR)
RETURN

EXTReport:
   '---- EXTernal loop report ----
   WINDOW 1
   PRINT #1,"ip=";iproblem%;"c=";EXTCost;" T=";T
   PRINT #1, "E=";iEXT%;"RF";epsRndFac;"iCost";iCost&
   PRINT #1, "iGasp=";iLastGasp%;"relCost";relCost
   PRINT #1, "iAccept";iAccept%;"gAccept";gAccept%
   GOSUB XReport
   PRINT #1, ""
   EXTReportLoop:
      IF (MOUSE(0) <> 0) THEN GOTO EXTReportLoop
RETURN

FinalReport:
   '---- Final resume ----
   WINDOW 1
   PRINT #2,"ip=";iproblem%;"ic=";icount%;"
T=";oldT;"E=";iEXT%;"RF";epsRndFac;
   PRINT #2, "iCost=";iCost&;"c=";optCost
   '---- print out X values ----
   PRINT #2, "X: ";
   FOR iX% = 1 TO nX%
      PRINT #2, USING "##.#### "; optX(iX%);
   NEXT
   PRINT #2, ""
RETURN

XReport:
   '---- print out X values ----
   PRINT #1, "X: ";
   FOR iX% = 1 TO nX%
      PRINT #1, USING "##.#### "; X(iX%);
      IF (iX% MOD 5 = 0) THEN PRINT ""
   NEXT
   PRINT #1, ""
RETURN

Xpoint:
   '---- point a circle ----
   WINDOW 2
   marksize% = 2
   rect%(0) = 300 * (-minX(2) + X(2)) / rangeX(2) - marksize%
   rect%(1) = 300 * (-minX(1) + X(1)) / rangeX(1) - marksize%
   rect%(2) = 300 * (-minX(2) + X(2)) / rangeX(2) + marksize%
   rect%(3) = 300 * (-minX(1) + X(1)) / rangeX(1) + marksize%
   IF (retAccept% = 1) THEN
      CALL PAINTOVAL(VARPTR(rect%(0)))
   ELSE
      CALL FRAMEOVAL(VARPTR(rect%(0)))
   END IF
RETURN

Tpoint:
   '---- point a circle ----
   WINDOW 2
   marksize% = 2
   rect%(1) = 50 * ABS(LOG(oldT / initT)) / LOG(10) - marksize%
   rect%(0) = 50 * ABS(LOG(ABS(retCost / initCost))) / LOG(10) - marksize%
   rect%(3) = 50 * ABS(LOG(oldT / initT)) / LOG(10) + marksize%
   rect%(2) = 50 * ABS(LOG(ABS(retCost / initCost))) / LOG(10) + marksize%
   IF (inner% = 0) THEN
      CALL PAINTOVAL(VARPTR(rect%(0)))
   ELSE
      CALL FRAMEOVAL(VARPTR(rect%(0)))
   END IF
RETURN

SUB G1(xx, retG1) STATIC
   retG1 = 0
   FOR i% = 1 TO 5
      retG1 = retG1 + i% * COS((i%+1) * xx + 1)
   NEXT
END SUB

SUB G1beta(xx, yy, beta, retG1beta) STATIC
   CALL G1(xx, dummy)
   CALL G1(yy, retG1beta)
   retG1beta = retG1beta * dummy
   retG1beta = retG1beta + beta * ((xx - .42513)^2 + (yy + .1997)^2)
END SUB
```