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-Marquardt 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

\[ dx = - \nabla f(x) \, dt + \sqrt{2T} \, dw, \]

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, \( \nabla \) 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
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 $x_{\text{next}} (= x + dx)$ gives the smaller function value than the current $x$, take the $x_{\text{next}}$. On the other hand, if $x_{\text{next}}$ 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 $x_{\text{next}}$, otherwise discard the $x_{\text{next}}$ and re-generate $x_{\text{next}}$. The higher the function value becomes in the next move, the less probable it becomes to
accept the move. The introduction of this Boltzmann 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 \textit{alternately}. That is, in one time, \( dx \) is calculated by \(-Vf(x)dt\) 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, \( <\nabla f(x)\cdot r>r \) is used instead, where \( r \) is a unit vector of a randomly picked axis. This is because the expected direction of \( <\nabla f(x)\cdot r>r \) 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 \( x_{\text{next}} \). 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, \( T_{\text{init}} \), is determined by a statistics gathered over randomly selected \( N_{\text{init}} \) 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
Random generation in n-th dimension

Generation using gradient information in one dimension

<table>
<thead>
<tr>
<th>$f'' &gt; 0$</th>
<th>$f'' \leq 0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f' = \frac{f^+ - f^-}{dx}$</td>
<td>$f'' = \frac{f^+ + f^- - 2f_0}{2dx^2}$</td>
</tr>
</tbody>
</table>

$x_{\text{next}} = x - \frac{f'}{2f''}$

Fig. 1 Proposed hill-descending method using $f'$ and $f''$ information
main {
T = Tinit = (x * (standard deviation of f(X) over randomly selected Ninit points));
// Set initial temperature by using heuristics. x = 0.2, Ninit = 200
S = Sinit;
// Set initial S to Sinit
Xinit = (Xinit_Given_by_User or one of those randomly 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) {
X = Xnew;
if (f(X) < f(Xopt)) {Xopt = X}  // If cost decreases,
// adopt the Xnew.
} else {
P = exp(-Δf / T);
R = random number in [0,1];
// Even if cost increases,
if (R < P) {X = Xnew}  // Boltzmann distribution.
}
if (f(X) > f(Xopt)) {X = Xopt}  // Resume the best X.
if (cost is not improved considerably) {
ifLast_Gasp +=;  // If cost is not improved considerably,
} else {
ifLast_Gasp = 0;  // where T is increased a little
}  // and then decreased to freeze.
Update_T();
Update_S();
// S = Sinit * (T / Tinit)α ; α = 0.5~1 (ex. a = 0.75)
} until ((ifLast_Gasp > ifLast_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() {
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 (ifLast_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 ≤ ifLast_Gasp ≤ n2) {T *= T_Factor2 (T_Factor2 > 1, ex. 1.3)}  // ex. n2 = 4
if (n2 ≤ ifLast_Gasp) {T *= T_Factor1 (T_Factor1 < 1, ex. 0.75)}
}

Fig.2 Algorithm of Fast Simulated Diffusion
should be 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>
<thead>
<tr>
<th>Problem description</th>
<th>CSD (*1)</th>
<th>Fast Simulated Diffusion (this work)</th>
</tr>
</thead>
<tbody>
<tr>
<td># (#1,2)</td>
<td>dimension</td>
<td># of local minima</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>760</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>760</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>760</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>125</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>625</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1e5</td>
</tr>
<tr>
<td>11</td>
<td>8</td>
<td>1e8</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>1e10</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>900</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>2.7e4</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>8.1e5</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>7.6e5</td>
</tr>
<tr>
<td>17</td>
<td>6</td>
<td>1.1e7</td>
</tr>
<tr>
<td>18</td>
<td>7</td>
<td>1.7e8</td>
</tr>
</tbody>
</table>

Average 3.8 5.7e8 93009 4828 0.96 5.2


*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 in 10 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 Lorentzian 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

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

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

<table>
<thead>
<tr>
<th>problem #</th>
<th>NF1: # of function evaluation</th>
<th>Simulated Annealing-like random search</th>
<th>Simulated Diffusion with Lorentzian Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NF3: # of function evaluation</td>
<td>success rate in 10 trials (Table I *4)</td>
<td>success rate in 10 trials (Table I *4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(%)</td>
<td>(%)</td>
</tr>
<tr>
<td>1</td>
<td>7168</td>
<td>3111</td>
<td>2939</td>
</tr>
<tr>
<td>2</td>
<td>77699</td>
<td>3060</td>
<td>2387</td>
</tr>
<tr>
<td>3</td>
<td>241215</td>
<td>4131</td>
<td>2877</td>
</tr>
<tr>
<td>4</td>
<td>76894</td>
<td>5967</td>
<td>3170</td>
</tr>
<tr>
<td>5</td>
<td>183819</td>
<td>5831</td>
<td>2678</td>
</tr>
<tr>
<td>6</td>
<td>10822</td>
<td>5151</td>
<td>3609</td>
</tr>
<tr>
<td>7</td>
<td>159549</td>
<td>7701</td>
<td>3023</td>
</tr>
<tr>
<td>8</td>
<td>72851</td>
<td>11322</td>
<td>3232</td>
</tr>
<tr>
<td>9</td>
<td>49690</td>
<td>11475</td>
<td>3401</td>
</tr>
<tr>
<td>10</td>
<td>72226</td>
<td>20053</td>
<td>4108</td>
</tr>
<tr>
<td>11</td>
<td>136061</td>
<td>28689</td>
<td>7716</td>
</tr>
<tr>
<td>12</td>
<td>98985</td>
<td>33986</td>
<td>9856</td>
</tr>
<tr>
<td>13</td>
<td>23770</td>
<td>7378</td>
<td>3294</td>
</tr>
<tr>
<td>14</td>
<td>66010</td>
<td>10761</td>
<td>3446</td>
</tr>
<tr>
<td>15</td>
<td>122166</td>
<td>11424</td>
<td>4051</td>
</tr>
<tr>
<td>16</td>
<td>66365</td>
<td>14790</td>
<td>4140</td>
</tr>
<tr>
<td>17</td>
<td>98974</td>
<td>19730</td>
<td>4903</td>
</tr>
<tr>
<td>18</td>
<td>109886</td>
<td>22962</td>
<td>6295</td>
</tr>
<tr>
<td>average</td>
<td>93009</td>
<td>12640</td>
<td>4174</td>
</tr>
</tbody>
</table>

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 $I_{D,\text{model}}$ 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

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


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.
Simulated Diffusion Program for MOSFET Device Parameter Extraction

--- Minimization with simulated annealing ---

OPTION BASE 0
maxnX% = 14
maxnData% = 1000
maxnEXT% = 100

DIM SHARED nxi(maxnX%), maxD(maxnX%), iniD(maxnX%),

x(maxnX%), rangeD(maxnX%),

DIM SHARED intEXTCost(maxnEXT%), iniNTACost(maxnData%),

DIM SHARED vgmn(maxnData%), vdmn(maxnData%), vbmn(maxnData%),

DIM SHARED idm(maxnData%), lgdsm(maxnData%),

DIM SHARED intX(maxnX%), optX(maxnX%), DIM SHARED nwiX(maxnX%), maxX(maxnX%), WtX<maxnX%), rangeX(maxnX%) = 1000

DIM SHARED intEXT(maxnEXT%), iniNTAC(maxnData%),

DIM SHARED tdm(maxnData%), ivgdbm(maxnData%), we»ghtm(maxnData%) = 1000

DIM SHARED mtnV(maxnX%), maxV(maxnX%), WtV(maxnX%) = 1000

DIM SHARED intXV, rtok(iMax%), ntokenMax%

DIM SHARED maxVgd, maxlds

ntokenMax% = 6

DIM token$(rtokenMax%)

2**! £ «2.V%(maxnX%*1). iV2»X%(maxnX%+1) SHARED mtnV(maxnX%), maxV(maxnX%), WtV(maxnX%) = 1000

DIM SHARED ifine%, ntokenMax%

DIM SHARED maxVgd, maxlds

ntokenMax% = 6

DIM token$(rtokenMax%)

Initializer

iX2iV%fial%) = taJP/b
A/2iX%<ial%) = U)Vo
pi = 3.141592
q = 1.6E-19
vtherm=.025
ni = 1.45E+10 • 1000000!
epssi = 8.855E-12"11.9
eppeox=8^S5E-12 " 35

ONMENU0 SUBMenucheck: MENU ON

GOTO Ido

Menucheck:

menunumber = MENU(0)

menuitem = MENU(1)

MENU


Filer:

ON menuitem GOSUB Loader, DataShower, Outer, 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 SRRunner, Manual, Contourer, ResShower

RETURN

Setter:

ON menuitem GOSUB Quitter

RETURN

Grapher:

ON menuitem GOSUB Vdslider, Vgslder

RETURN

ClipCopier:

PICTURE OFF

images = PICTURES

OPEN "CUP:PICTURE" FOR OUTPUT AS 13

PRINT#3, images

CLOSE #3

RETURN

Eraser:

WINDOW3

WINDOW2

WINDOW1

images = " CLS

RETURN

Loader:

OPEN "CUP:PICTURE" FOR OUTPUT AS #1

PRINT#1, images

RETURN

--- load data ----

iflines = FILES(1, "TEXT")
IF (lines = 0) THEN RETURN

OPEN lines FOR INPUT AS #2

--- input ----

skipLoadFlag% = 0

data% = 0

t% = 0; P% = nX% + 1

...
Simulated Diffusion Program for MOSFET Device Parameter Extraction

```
line% = 0
WHILE NOT EOF(2)
    LINE INPUT #2, inlfieS
    ine% = ine% + 1
    • parse the input line -
    CALL param(inlfieS, tokenS, rtokS, errorParseFlag%)
    IF (errorParseFlag% = 1) GOTO breakLoadLoop
    IF (rtokS(1) = "") OR (rtokS(1) = "") THEN skipLoadFlag% = 1
    THEN
        IF (rtokS(2) = "=") THEN
            -- voltage card --
            SELECT CASE UCASES(tokenS)
                CASE "VGS"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
                CASE "VDS"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
                CASE "VSB"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
                CASE "VMB"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
                CASE "VMA"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
                CASE "VTO"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
                CASE "VX"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
                CASE "VX2"
                    CALL paramRead(tokenS, 00, 00, 00, 00, mom%)
            END SELECT
        ELSE
            -- real data --
            idata% = idata% + 1
            vgsm(idata%) = vgs
            vdsm(idata%) = voB
            vbsm(idata%) = vbB
            vgdbrm(idata%) = vgdbr
            SELECT CASE mom%
                CASE 1
                    vgsm(idata%) = VAL(tokenS(1))
                CASE 2
                    vdsm(idata%) = VAL(tokenS(1))
                CASE 3
                    vbsm(idata%) = VAL(tokenS(1))
            END SELECT
        END IF
    ELSE
        IF (tokenS(1) = "=") THEN skipLoadFlag% = 0
    END IF
WEND
```

```
breakLoadLoop:
    • post-processing of the input --
    IF (idata% = 0) THEN ndata% = idata% ELSE call ermsg("No measured data.")
    IF (idata% = 0) THEN CALL ermsg("Parameters or variables missing.")
    • define markov chain length --
        minRUN% = ndata% * 50
        minRUN% = ndata% * 24
        maxRUN% = ndata% * 15
    CLOSE #2
    • put initial X into X --
        FOR (0 = 1 TO ndata%) x(0) = initX(0)
        rangeX(0) = maxX(0) - minX(0)
    RETURN
```

```
DataShower:
    • show loaded data --
        PRINT CHRS(13) + CHRS(13) + CHR$(13) + CHR$(13)
        PRINT "Click mouse to pause."
        PRINT "--- show parameter Wo ---"
        FOR iX% = 1 TO nX%
            PRINT #1, nameVS(iX2%), TDC", x(iX%)
        NEXT
        IF (MOUSE(0) <> 0) GOTO DataShowerLoop1
        FOR P% = sP% TO nP%
            PRINT #1, nameVS(iX2%), TDC", x(iX%)
        NEXT
        IF (MOUSE(0) <> 0) GOTO DataShowerLoop2
        • show measured data --
            PRINT #1, "vgs", "vdsm", "vbsm", "id", "vgdbr", "vmax",
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
            PRINT "for idata% = 1 TO ndata%"
```
Simulated Diffusion Program for MOSFET Device Parameter Extraction

vbs = vbsmf(data%)
ivgdb% = ivgcbn?/o(idaia%)

SELECT CASE fcgdftb
     CASE 1
         *—changing vgs —
         graph% = igraph% + i
         STEPv^mat/20
         PRINT #5. vgs. ids
     NEXT
     PRINT #5.
     END IF
     CASE 2
         *—changing vds —
         igraph% = igraph% + 1
         FG0aSM2s3° V*ma5r1001 STEPvdsmax/a
         PRINT#5, vds. ids
     NEXT
     PRINT #5.
     END IF
     END SELECT

'— show parameter info —
FOR idata% = 1 TO ndata%
     igraph% = igraph% + 1
     IF (lgdbm%/(data%)) THEN
         PRINT #5, "Line n_type marker label": igraph%,* 1;"* idata%
         PRINT #5, vgsm(data%)*"!1, idm(idata%)
         PRINT #5, vds(data%)*"1, idm(idata%)
     END IF
     IF (lgdbm%/(data%)) THEN
         PRINT #5, "Line n_type marker label": igraph%,* 1;"* idata%
         PRINT #5, vds(data%)*"1, idm(idata%)
         PRINT #5, vds(data%)*"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 iX%
     PRINT #4, namevS(iX%)
     PRINT #4.
     FOR iP% = iP/o TO iP%.
         IF (ss$i<>"=") THEN
             vabs = VAL(ss$)
             x(iX%) = vabs
         END IF
     NEXT
     PRINT #4.

CLOSE #4
SetCreate "mosfit.par".MSWD
RETURN

Vdsider:
'— Vds - Id graph —
WINDOW 3
INPUT "Vds": vds
INPUT "Vgmin, Vgmax, Vgstep": vgsmin, vgsmax, vgstep
INPUT "Vdsmin, Vdsmax, Vdsstep": vdsmin, vdsmax, vdsstep
FOR vds = vdsmin TO vdsmax STEP vdsstep
    FOR vgs = vgsmin TO vgsmax STEP vgstep
        GOSUB Mos3
        PRINT #10, vgs, vds
    NEXT
    PRINT #10.
NEXT
PRINT #10.""
RETURN

Vgskter:
'— Vgs - Id graph —
WINDOW 3
INPUT "Vds": vds
INPUT "Vgmin, Vgmax, Vgstep": vgsmin, vgsmax, vgstep
FOR vgs = vgsmin TO vgsmax STEP vgstep
    FOR vds = vdsmin TO vdsmax STEP vdsstep
        GOSUB Mos3
        PRINT #1, vgs, ids
    NEXT
    PRINT #1,""
NEXT
PRINT #1,""
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(3):" outdev%
SELECT CASE outdev%
CASE 0
OPEN "com": FOR OUTPUT AS #1
OPEN "out.txt": FOR OUTPUT AS #1
PRINT #1, "exit" FOR OUTPUT AS #1
PRINT #1,"",
PRINT #1,""
PRINT #1,""
PRINT #1,""
END IF
CASE 2
OPEN "out.txt": FOR OUTPUT AS #1
PRINT #1, "exit" FOR OUTPUT AS #1
PRINT #1,"",
PRINT #1,""
PRINT #1,""
END IF
CASE 3
OPEN "out.txt": FOR OUTPUT AS #1
PRINT #1,"",
PRINT #1,""
PRINT #1,""
PRINT #1,""
END IF
CASE ELSE
OPEN "out.txt": FOR OUTPUT AS #1
PRINT #1,"",
PRINT #1,""
PRINT #1,""
PRINT #1,""
END SELECT
RETURN

Manualer:
'— manual fitting —
WINDOW 1
PRINT "Variables".
FOR iX% = 1 TO iP%.
    IF (ss$i<>"=") THEN
        vabs = VAL(ss$)
        x(iX%) = vabs
    END IF
NEXT
PRINT #4.
NEXT
PRINT #4,""
NEXT
PRINT #4,""
NEXT
PRINT #4,""
NEXT
PRINT#4.

RETURN

Vdsider:
'— Vds - Id graph —
WINDOW 3
INPUT "Vds": vds
INPUT "Vgmin, Vgmax, Vgstep": vgsmin, vgsmax, vgstep
INPUT "Vdsmin, Vdsmax, Vdsstep": vdsmin, vdsmax, vdsstep
FOR vds = vdsmin TO vdsmax STEP vdsstep
    FOR vgs = vgsmin TO vgsmax STEP vgstep
        GOSUB Mos3
        PRINT #10, vgs, vds
    NEXT
    PRINT #10.
NEXT
PRINT #10.""
RETURN

Vgskter:
'— Vgs - Id graph —
WINDOW 3
INPUT "Vds": vds
INPUT "Vgmin, Vgmax, Vgstep": vgsmin, vgsmax, vgstep
FOR vgs = vgsmin TO vgsmax STEP vgstep
    FOR vds = vdsmin TO vdsmax STEP vdsstep
        GOSUB Mos3
        PRINT #1, vgs, ids
    NEXT
    PRINT #1,""
NEXT
PRINT #1,""
RETURN
Simulated Diffusion Program for MOSFET Device Parameter Extraction

FOR idata% = 1 TO ndata%
    vgs = vgsm(idata%)
    vds = vdsm(idata%)
    vbs = vosm(idata%)
    vgdb% = ivgdbm(idata%)
    SELECT CASE vgdb%
        CASE 1
            vgs = 0
            GOSUB MosS
            CALL User2Wodd(vgs, vds, vbs, wX%, wY%)
            FOR vds = 0 TO vdsmax STEP vdsmax/20
                GOSUB MosS
                CALL User2Wodd(vgs, vds, vbs, wX%, wY%)
            NEXT
            ENDIF
        CASE 2
            vgs = vgsm(idata%)
            FOR vgs = 0 TO vgsmax STEP vgsmax/20
                GOSUB MosS
                CALL User2Wodd(vgs, vds, vbs, wX%, wY%)
            NEXT
            ENDIF
    END SELECT
END IF
NEXT

movext = LOG(minXc)
mmaxXc = LOG(maxXc)
stepXc = LOG(stepXc)
ENDIF
IF (logXc% = 1) THEN stepXc = EXP(LOG(maxXc - minXc)/10) ELSE
    stepXc = rangeX(Xc(minXc, maxXc))/10
END IF
PRINT "stepXc = " stepXc
IF (as$ = "") THEN stepXc = VAL(as$)

FOR iX% = 1 TO nXP%
    PRINT "xyz file for contour SI";
    OPEN "mosfit.'cont" FOR OUTPUT AS #4
    FOR iY% = 1 TO nYP%
        scaleX = (maxXc - minXc)*(iX%-minXc)/scaleX
        scaleY = (maxYc - minYc)*(iY%-minYc)/scaleY
        PRINT iX%, iY%, scaleX, scaleY
    NEXT
    CLOSE #4
    SetCreate "mosfit.cont", "MSVD"
    INPUT "OK or Try Again(1)"; as$
    IF (as$ = "") THEN GOTO ContourLoop
    RETURN
CONTour:
    "Contour output";
    "Contour Loop";
    "WINDOWS";
    "FOR iX% = 1 TO nXP%";
    "PRINT name VS(Xc(minXc, maxXc))": INPUT as$
    IF (as$ = "") THEN EXIT
    NEXT
    "FOR iY% = 1 TO nYP%";
    "PRINT name VS(Yc(minYc, maxYc))": INPUT as$
    IF (as$ = "") THEN EXIT
    NEXT
    "SELECT variables";
    logXc% = 0; logYc% = 0; logZc% = 0
    IF (as$ = "") THEN EXIT
    PRINT "linear or log(1)"; INPUT as$
    IF (as$ = "") THEN EXIT
    minXc = minXc(Xc(minXc, maxXc))
    PRINT "minXc = " minXc
    EXIT
    NEXT
    EXIT
RETURN

Next:
CLOSE #4
SetCreate "mosfit.cont", "MSVD"
INPUT "OK or Try Again(1)"; as$
IF (as$ = "") THEN GOTO ContourLoop
RETURN

SDRunner:
    "set initial T & X & Cost";
    GOSUB IntiaX
    GOSUB Cost
    okEXTCost = retCost
    GOSUB IntiNT
    okINTCost = okEXTCost
    EXT% = 0
    EXTLoop:
        "-count-up loop counter";
        EXT% = EXT% + 1
        "initialize random generator";
        RANDOMize
        RANDOMize
        INT% = 0
        sumINTCost = 0
        sumINTCost2 = 0
        IAccept% = 0
        GIAccept% = 0
        INTLoop:
            "Randomize model";
            EXIT% = EXT%
            "-Initialize random generator";
            RANDOMize
            RANDOMize
            INT% = 0
            INTLoop:
            EXIT%
            EXIT
Simulated Diffusion Program for MOSFET Device Parameter Extraction

--- internal loop with same T ---
INT% = INT% + 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 (GOSUBXCalled% = 0) THEN
    accept% = accept% + 1
  ELSE
    accept% = gAccept% + 1
  END IF
END IF
--- save current status if optimal ---
IF (optINTCost < oldINTCost) THEN
  optINTCost = retCost
  oldINT% = INT%
GOTO JumpINTGreedy
END IF

--- post-process of INT loop ---
inner% = 0
nINT% = INT%
GOTO BreakINTLoop

JumpINTGreedy:
oldEXTCost = oldINTCost
--- update optEXTX if this is optimal up to now ---
IF (oldEXTCost < optEXTCost) THEN
  optEXTCost = oldEXTCost
  FOR %x = 1 TO nx%
    x(0%) = optEXTX(0%)
  NEXT
END IF
END IF

--- resume optINTX since it is minimal ---
GOSUB ExitINTLoop
GOTO BreakINTLoop

--- post-process of EXT loop ---
histEXTCost(EXT%) = retCost
END IF

--- exit INT loop? ---
GOSUB ExitINTLoop

Accept:
--- decide accept or reject using Boltzmann dist. ---
deltaINTCost = retCost - oldINTCost
IF (deltaINTCost < 0) THEN
  retAccept% = 1
ELSE
  boltzmann = EXP(-deltaINTCost / T)
  IF (RND% < boltzmann) THEN
    retAccept% = 1
  ELSE
    retAccept% = 0
  END IF
END IF
END IF
RETURN

InitEXT:
--- try random search minRND times and guess initial T ---
minINTCost = infinity
maxCost = infinity
sumCost = 0;
sumCost2 = 0
FOR iRND% = 1 TO minRND% DO
  --- random generation of X ---
  FOR %x = 1 TO nx%
    nameV% = nameV%(iX%,V%(iX%))
    IF (nameV% = "nsub") OR (nameV% = "vsub") THEN
      logminX = LOG(minX(0%))
      logmaxX = LOG(maxX(0%))
      deltafX = RAND(1) * lograngeX
      x(0%) = minX(0%) + deltafX
    ELSE
      deltafX = RAND(1) * rangeX(0%)
      x(0%) = minX(0%) + deltafX
    END IF
  NEXT
  GOSUB Coat
  --- initial INT tag% = 0
  FOR %x = 1 TO nx%
    WX(0%) = x(0%)
  NEXT
END IF

--- update Temp ---
GOSUB UpdateT
--- update epsilon for random part ---
GOSUB UpdateEpsRandFac
GOSUB EXITReport
oldT = T
GOTO EXTLoop
breakEXTLoop:
--- post-process of EXT loop ---
GOSUB ResShower
RETURN

InitAI:
--- initialize X ---
FOR %x = 1 TO nx%
  x(0%) = initX(0%)
  histEXTCost(EXT%) = histEXTCost(EXT%-1)
END IF
RETURN
Simulated Diffusion Program for MOSFET Device Parameter Extraction

WcCost = retCost
"PRINT #1. 'Wr.WtTrWtCost
T = WfT
FOR iX% = lToX%
  x(0p/o) = WtX(W>/o)
NEXT
okEXTCost = rnhCost
okWTCost = mhCost
ELSE
  FOR iX% = lToX%
    x(iX%) = oldEXTX(iX%)
  NEXT
  okEXTCost = okEXTCost
  oldtNTCost = okEXTCost
ENDF
RETURN

GenerateX:
  F (EXT% > ninnEXT%) THEN
    GenerateXCalledVo = 1
    GenerateXCalled% = 0
  ENDIF
  IF (iGenerateXCalled% = 1) THEN
    — gradient generation in log space for nsub and vmax ----
    if (logXg% = 1) THEN
      minXg = LOG(min(x(0X%)): maxXg = LOG(max(x(0X%)): Xg = x(0X%))
    ELSE
      rangeXg = maxXg - minXg
      rangeX = rangeXg * .0001
      DX = rangeX * .0001
      — find Xopt by fitting quadratic form ----
      IF (logXg% = 0) THEN
        x(0X%) = Xg + DX ELSE x(0X%) = EXP(Xg + DX)
      GOSUB Cost
      fpus = retCost
      IF (logXg% = 0) THEN
        x(0X%) = Xg - DX ELSE x(0X%) = EXP(Xg - DX)
      GOSUB Cost
      fminus = retCost
      concave = fpus + fminus - 2 * 10
      IF (concave < 0) THEN
        f = 0
        deltaXg = DX / 2 * (fpus - fminus) / concave
        — limit up to limitDeltaXg ----
        IF (ABS(deltaXg) > rangeXg * .2) THEN deltaXg = SGN(deltaXg) * .2
      ELSE
        limitDeltaXg = rangeXg * .05
        f = 0
        IF (fpus > fminus) THEN
          f = 0
        ELSE
          deltaXg = 0
        ELSE
          GenerateXLoop0:  
          GenerateXLoop0:
        ELSE
          GenerateXLoop2:
        ELSE
          GenerateXLoop2:
      ENDIF
      ENDIF
      RETURN
    ENDIF
    UpdateEpsRndFac:
      — update epsilon for random part ----
      epsRndFac = epsRndFac * (T / okT) * .75
      IF (LastGasp% > 1) THEN
        epsRndFac = SQR(T / okT)
      ELSE
        IF (epsRndFac < .05) THEN epsRndFac = .05
      ENDIF
      RETURN
    ENDIF
    UpdateEpsRndFacOld:
      — update epsilon for random part ----
      acceptRatio = acceptRatio / nINT%
      IF (LastGasp% = 1) THEN
        IF (epsRndFac >= .5) THEN epsRndFacUp% = 0 ELSE epsRndFacUp% = .1
      END_IF
      ELSE
        epsRndFac = epsRndFac * 1.5
      ENDIF
      IF (epsRndFac < .1) THEN
        epsRndFac = .15
      ELSE
        epsRndFac = epsRndFacUp% + 1
      ENDIF
      RETURN
    ENDIF
    RETURN
FOR X% = 1 TO nX%  
oldx(X%) = x(X%)
NEXT  RETURN

ResumeOldX:
FOR X% = 1 TO nX%  
x(X%) = oldx(X%)
NEXT  RETURN

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

ResumeOMX:
FOR iX% = lTOnX%  
x(iX%) = cldX(iX%)
NEXT  RETURN

ExitEXTLoop:
    --- end of loop condition, frozen condition ----
    retExitEXTLoop% = 0
    IF (EXT% > maxnEXT%) THEN  
        --- if LOOP count exceed limit, simply exit EXT loop ----
        retExitEXTLoop% = 1
    RETURN  END IF

UpdateT:
    --- update T by ICCAD86 ----
    IF (iLastGasp% >= 1) THEN  
        IF (iLastGasp% >= 3) THEN  
            T = 5*T  
        ELSE  
            T = 1.8*T  
        END IF
    ELSE  
        T = 1.6*T  
    END IF
    IF (T > INT%) THEN T = .1*T  
    ELSE
       Ni% = iAccept%  
       IF (Ni% >= 3) THEN  
           --- if acceptance ratio is high enough ----
           sumINTACost = 0
           FOR iU% = 1 TO nU%
               sumINTACost = sumINTACost + histINTCost(iU%)
           NEXT
           aveINTACost = sumINTACost / nU%
           sigmaINTACost = (histINTCost(iU%) - aveINTCost)^2  
           FOR iU% = 1 TO nU%
               IF (iU% = aveINTCost) THEN
                   T = 0.9 * T
               ELSE
                   T = factorUpdateT * T
               END IF
           END IF
       END IF
    ELSE
        Ni% = iAccept%
        FOR iU% = 1 TO nU%
            IF (iU% = aveINTCost) THEN
                T = 0.9 * T
            ELSE
                T = factorUpdateT * T
            END IF
        END IF
    END IF
    RETURN

Cost:
    --- calculating cost ----
    iCost% = iCost% + 1
    --- summation over measured data ----
    retCost = 0
    "WINDOW3"
    FOR idata% = 1 TO ndata%
        vgs = vgsm(idata%)
        vds = vdsm(idata%)
        vbe = vbsm(idata%)
        GOSUB MosS
        retCost = retCost + ABS(idm(idata%) - ids) * weightm(idata%)
    NEXT
    retCost = retCost + 1E-10
    RETURN

GaussRnd:
    --- gaussian distribution (see p.217 of NR) ----
    IF (GaussCalced% = 0) THEN  
        gaussR = 2
        WHILE (gaussR >= 1)  
            gaussR = 2*RND(1) - 1
        gaussFac = SQ(R(-2 * LOG(gaussR) / gaussR))
        gaussSet = gaussV1 * gaussFac
        iGaussCalced% = 1
    ELSE
        retGaussRnd = gaussV2 * gaussFac
        iGaussCalced% = 0
    END IF
    RETURN

LorentzRnd:
    --- Lorentzian distribution (see p.217 of NR) ----
    LorentzLoop:
        b = RND(1)
    RETURN  "WININ

EXTReport:
    --- EXTernal bop report ----
    "WINDOW2"
    PRINT #1, "vgs", "vds", "vbe", "idd": idd
    FOR idata% = 1 TO ndata%
        vgs = vgsm(idata%)
        vds = vdsm(idata%)
        vbe = vbsm(idata%)
        GOSUB MosS
        PRINT #1, vgsm(idata%), vdsm(idata%), vbsm(idata%): idm(idata%): ids
    NEXT
    IF (MOUSE(0) <> 0) THEN GOTO EXTReportBreak
    GOSUB ResShower
    EXTReportBreak:
        IF (MOUSE(0) <> 0) THEN GOSUB ResShower
    RETURN

XReport:
    --- print out X values ----
    "WINDOW1"
    PRINT #1, nameV$(iX%); x(X%)

T = 9*T
sigmaINTACost = 0.9999!
aveINTACost = 0.9999!
END IF
END IF
RETURN

Cost:
    --- calculating cost ----
    iCost% = iCost% + 1
    --- summation over measured data ----
    retCost = 0
    "WINDOW3"
    FOR idata% = 1 TO ndata%
        vgs = vgsm(idata%)
        vds = vdsm(idata%)
        vbe = vbsm(idata%)
        GOSUB MosS
        retCost = retCost + ABS(idm(idata%) - ids) * weightm(idata%)
    NEXT
    retCost = retCost + 1E-10
    RETURN

GaussRnd:
    --- gaussian distribution (see p.217 of NR) ----
    IF (GaussCalced% = 0) THEN  
        gaussR = 2
        WHILE (gaussR >= 1)  
            gaussR = 2*RND(1) - 1
        gaussFac = SQ(R(-2 * LOG(gaussR) / gaussR))
        gaussSet = gaussV1 * gaussFac
        iGaussCalced% = 1
    ELSE
        retGaussRnd = gaussV2 * gaussFac
        iGaussCalced% = 0
    END IF
    RETURN

LorentzRnd:
    --- Lorentzian distribution (see p.217 of NR) ----
    LorentzLoop:
        b = RND(1)
    RETURN  "WININ

EXTReport:
    --- EXTermal bop report ----
    "WINDOW2"
    PRINT #1, "vgs", "vds", "vbe", "idd": idd
    FOR idata% = 1 TO ndata%
        vgs = vgsm(idata%)
        vds = vdsm(idata%)
        vbe = vbsm(idata%)
        GOSUB MosS
        PRINT #1, vgsm(idata%), vdsm(idata%), vbsm(idata%): idm(idata%): ids
    NEXT
    IF (MOUSE(0) <> 0) THEN GOTO EXTReportBreak
    GOSUB ResShower
    EXTReportBreak:
        IF (MOUSE(0) <> 0) THEN GOSUB ResShower
    RETURN

XReport:
    --- print out X values ----
    "WINDOW1"
    PRINT #1, nameV$(iX%); x(X%)

T = 9*T
sigmaINTACost = 0.9999!
aveINTACost = 0.9999!
END IF
END IF
RETURN
Simulated Diffusion Program for MOSFET Device Parameter Extraction

IF (X% MOD 1 = 0) THEN PRINT "Xpoint:"
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Simulated Diffusion Program for MOSFET Device Parameter Extraction

F(iX% MOD 1 = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Simulated Diffusion Program for MOSFET Device Parameter Extraction

F(iX% MOD 1 = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Simulated Diffusion Program for MOSFET Device Parameter Extraction

F(iX% MOD 1 = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN

Xpoint:
'— point a circle —
WINDOW 2
SHOWPEN
IF (XpointCalled% = 0) THEN PRWT#1.8
RETURN
Simulated Diffusion Program for MOSFET Device Parameter Extraction

--- recognize token as an entity from the first char to the next blank ---
tokenS(tokenS%) = LEFTS(sS, idelim%+1)
--- update line string ---
sS = MID$S(sS, idelim%+1)
WEND
breakParselLoop:
tokenS% = tokenS%
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 errormsgS
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 = Window
END SUB

SUB paramRead(tokenS%, D%, IP%, IV%, VV%) STATIC
IF (UCASES(tokenS(3)) = "FIX") THEN
    --- constant parameters ---
    P% = P% + 1
    D25% = D25% + IV%
    N25% = N25% + P%
    IF (tokenS(3) = "") THEN intX(IP%) = intY(IV%) ELSE intX(IP%) = VAL(tokenS(4))
    ELSE
        IF (tokenS(3) = "") THEN minVal = minV(VV%) ELSE minVal = VAL(tokenS(3))
        IF (tokenS(4) = "") THEN maxVal = maxV(VV%) ELSE maxVal = VAL(tokenS(4))
        IF (tokenS(5) = "") THEN print "X(0%) = minVal: maxVal: minVal: maxVal: minVal: maxVal"
        VV%(IP%) = IF (minVal = maxVal) THEN
            --- constant parameters ---
            P% = P% + 1
            D25% = D25% + IV%
            N25% = N25% + P%
            intX(IP%) = minVal
        ELSE
            --- variables ---
            D% = D% + 1
            D25% = D25% + IV%
            N25% = N25% + P%
            minX(D%) = minVal
            maxX(D%) = maxVal
        IF (tokenS(5) = "") THEN internX(D%) = intY(VV%) ELSE intX(D%) = VAL(tokenS(5))
    ENDIF
END IF
END IF
END SUB

SUB initializeV(IV%, nameVsS, minVs, maxVs, intVs) STATIC
    --- initializing V ---
    nameVsS(IV%) = nameVsS
    minIV(IV%) = minVs: maxIV(IV%) = maxVs: intIV(IV%) = intVs
END SUB

--- convert user coord to world coord ---
SUB User2World(x, y, wX%, wY%) STATIC
    wX% = x / maxVgd * 240 + 20
    wY% = y / maxIda * 240 + 20
END SUB
Simulated Diffusion Program for a Set of Test Functions

--- Minimization with simulated annealing ---
--- initialize I/O ---
--- minEXT is changed to 10 from 5 ---
--- Lorentzian
OPEN "scm*" FOR OUTPUT AS #1
OPEN funcSD.res* FOR OUTPUT AS #2

FOR problem% = 1 TO 18
% = problem%
IF (% = 4) AND (% = 5) AND (% = 12) AND (% = 18) THEN GOTO BreakProblemLoop

FOR count% = 1 TO 10
Windower:
    "--- initialize window 1 ---
    WINDOW 1: "Graphics Window", (200,200)-(480,300),1
    WINDOW 2: "Text Window", (0.20),(200,300),1
TEXTSIZE 8
"INPUT "problem": problem%

OPTION BASE 0
SELECT CASE problem%
CASE 0
    rh% = 2
CASE 1
    rh% = 1
CASE 2
    rh% = 0
CASE 3 TO 7, 13
    rh% = 2
CASE 8, 14
    rh% = 3
CASE 9, 15
    rh% = 4
CASE 10, 16
    rh% = 5
CASE 11
    rh% = 6
CASE 12
    rh% = 7
CASE 13
    rh% = 8
CASE 14
    rh% = 9
CASE 15
    rh% = 10
CASE 16 FORM 17
    rh% = 10
CASE ELSE
    ENDSELECT

maxn% = 14
minINT% = 0
maxINT% = 100
DIM SHARED oldX(maxn%), newX(maxn%), oldEXTX(maxn%), optX(maxn%)
DIM SHARED minX(maxn%), maxX(maxn%), initX(maxn%), rangeX(maxn%)
DIM SHARED histINTACost(maxn%)

Initializer:
iGreedy% = 1
infinity = 1E+20
infintesimal = 1E-20
intTK = .2
updateTImeImax = .7
factorUpdateTIma = .6
minEXT% = 10
iGenerateXcalled% = 0
iGenerateXcalled% = 0
epsRNDFac = .5
pi = 3.141592

SELECT CASE problem%
CASE 0
    FOR rh% = 1 TO nX%
        minX(rh%) = -1: maxX(rh%) = 11
        rangeX(rh%) = maxX(rh%) - minX(rh%)
        initX(rh%) = (maxX(rh%) + minX(rh%)) / 2
        intX(rh%) = 1
    NEXT
CASE 1 TO 17
    FOR rh% = 1 TO nX%
        minX(rh%) = -10: maxX(rh%) = 10
        rangeX(rh%) = maxX(rh%) - minX(rh%)
        intX(rh%) = 0
    NEXT
CASE ELSE
    ENDSELECT

END
Simulated Diffusion Program for a Set of Test Functions

IF (retExitINTLoop%=1) THEN GOTO BreakINTLoop
GOTO INTLoop

BreakINTLoop:
"--- post-process of INT loop ---
inner%=0
INT%=initINT%
EXTCost = oldINTCost
originalEXTCost = EXTCost
"--- exit INT loop? ---
IF (EXTCost > optCost) AND (Greedy%=1) THEN GOSUB ResumeOptX

ResumeOptX:
resetCost = EXTCost
GOSUB ExtEXTLoop
IF (retExitEXTLoop%=1) THEN GOTO BreakEXTLoop

BreakEXTLoop:
"--- post-process of EXT loop ---
GOSUB FinaReport
RETURN

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

InitiaX:
"--- initialize X ---
FOR n% = 1 TO nX%
X(n%) = intX(n%)
oldEXTX(n%) = iniX(n%)
NEXT
RETURN

InitiaT:
"--- try random search nintRand times and guess initial T ---
maxCost = infinity
sumCost = 0; sumCost2 = 0
FOR limitRand% = 1 TO limitRand%
"--- random generation of X ---
FOR n% = 1 TO nX%
X(n%) = iniX(n%)
rangeX = rangeX(n%)
deltaX = rangeX * .00001
Xg = X(n%)
"--- fhdXopt by fitting quadratic form ---
fO = oldINTCost
Xg + deltaX
GOSUB Cost
fpbBaretCost
Xg - deltaX
GOSUB Cost
frenusaretCost
concave = (fpbB + fO) - 2 * fO
F (concave > 0) THEN
"--- f>0 ---
deltaXg = deltaX / 2 * (fO + fplus) / concave
ELSE
"--- f<=0 ---
deltaXg = deltaX / 2
ENDF
Xg + deltaXg
GOSUB Cost
ENDF

ENDF
ELSE
Xg + deltaXg
Xg - deltaXg
ENDF
RETURN

GenerateX:
IF (EXT% > maxEXT%) THEN
GenerateXCalled% = 1 - GenerateXCalled%
ELSE
GenerateXCalled% = 0
END IF
IF (GenerateXCalled% = 1) THEN
"--- if called% = 1 then gradient ---
rangeXg = rangeX(n%)
Xg = X(n%)
"--- find Xopt by fitting quadratic form ---
f0 = oldINTCost
Xg + deltaX
GOSUB Cost
fpbBaretCost
Xg - deltaX
GOSUB Cost
frenusaretCost
concave = (fpbB + fO) - 2 * fO
F (concave > 0) THEN
"--- f>0 ---
deltaXg = deltaX / 2 * (fO + fplus) / concave
ELSE
"--- f<=0 ---
deltaXg = deltaX / 2
ENDF
Xg + deltaXg
GOSUB Cost
ENDF

ENDF
ELSE
Xg + deltaXg
Xg - deltaXg
ENDF
RETURN

GenerateX:
IF (EXT% > maxEXT%) THEN
GenerateXCalled% = 1 - GenerateXCalled%
ELSE
GenerateXCalled% = 0
END IF
IF (GenerateXCalled% = 1) THEN
"--- if called% = 1 then gradient ---
rangeXg = rangeX(n%)
Xg = X(n%)
"--- find Xopt by fitting quadratic form ---
f0 = oldINTCost
Xg + deltaX
GOSUB Cost
fpbBaretCost
Xg - deltaX
GOSUB Cost
frenusaretCost
concave = (fpbB + fO) - 2 * fO
F (concave > 0) THEN
"--- f>0 ---
deltaXg = deltaX / 2 * (fO + fplus) / concave
ELSE
"--- f<=0 ---
deltaXg = deltaX / 2
ENDF
Xg + deltaXg
GOSUB Cost
ENDF

ENDF
ELSE
Xg + deltaXg
Xg - deltaXg
ENDF
RETURN

GenerateX:
IF (EXT% > maxEXT%) THEN
GenerateXCalled% = 1 - GenerateXCalled%
ELSE
GenerateXCalled% = 0
END IF
IF (GenerateXCalled% = 1) THEN
"--- if called% = 1 then gradient ---
rangeXg = rangeX(n%)
Xg = X(n%)
"--- find Xopt by fitting quadratic form ---
f0 = oldINTCost
Xg + deltaX
GOSUB Cost
fpbBaretCost
Xg - deltaX
GOSUB Cost
frenusaretCost
concave = (fpbB + fO) - 2 * fO
F (concave > 0) THEN
"--- f>0 ---
deltaXg = deltaX / 2 * (fO + fplus) / concave
ELSE
"--- f<=0 ---
deltaXg = deltaX / 2
ENDF
Xg + deltaXg
GOSUB Cost
ENDF

ENDF
ELSE
Xg + deltaXg
Xg - deltaXg
ENDF
RETURN

GenerateX:
IF (EXT% > maxEXT%) THEN
GenerateXCalled% = 1 - GenerateXCalled%
ELSE
GenerateXCalled% = 0
END IF
IF (GenerateXCalled% = 1) THEN
"--- if called% = 1 then gradient ---
rangeXg = rangeX(n%)
Xg = X(n%)
"--- find Xopt by fitting quadratic form ---
f0 = oldINTCost
Xg + deltaX
GOSUB Cost
fpbBaretCost
Xg - deltaX
GOSUB Cost
frenusaretCost
concave = (fpbB + fO) - 2 * fO
F (concave > 0) THEN
"--- f>0 ---
deltaXg = deltaX / 2 * (fO + fplus) / concave
ELSE
"--- f<=0 ---
deltaXg = deltaX / 2
ENDF
Xg + deltaXg
GOSUB Cost
ENDF

ENDF
ELSE
Xg + deltaXg
Xg - deltaXg
ENDF
RETURN

GenerateX:
IF (EXT% > maxEXT%) THEN
GenerateXCalled% = 1 - GenerateXCalled%
ELSE
GenerateXCalled% = 0
END IF
IF (GenerateXCalled% = 1) THEN
"--- if called% = 1 then gradient ---
rangeXg = rangeX(n%)
Xg = X(n%)
"--- find Xopt by fitting quadratic form ---
f0 = oldINTCost
Xg + deltaX
GOSUB Cost
fpbBaretCost
Xg - deltaX
GOSUB Cost
frenusaretCost
concave = (fpbB + fO) - 2 * fO
F (concave > 0) THEN
"--- f>0 ---
deltaXg = deltaX / 2 * (fO + fplus) / concave
ELSE
"--- f<=0 ---
deltaXg = deltaX / 2
ENDF
Xg + deltaXg
GOSUB Cost
ENDF

ENDF
ELSE
Xg + deltaXg
Xg - deltaXg
ENDF
RETURN
Simulated Diffusion Program for a Set of Test Functions

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% = 1 TO n%

okX(i%) = X(i%)

NEXT
RETURN

ResumeOldX:

FOR% = 1 TO n%

X(i%) = oldX(i%)

NEXT
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitEXTLoop:

- EXT loop exit condition, frozen condition ---

  retExitEXTLoop% = 0

  IF (EXT% > maxnEXT%) THEN

  retExitEXTLoop% = 1

  RETURN

END IF
END IF
RETURN

Cost:

- problems ---

  SELECT CASE probtem%
  
  CASE 0
  
  x = X(1); y = X(2)
  
  retCost = x*x*x + 2*x*y*y - 3.0*COS(3.0*x*x) - 4.0*COS(4.0*y*y) + .7
  
  CASE 1
  
  x = X(1)
  
  CALL G1(xx, retCost)
  
  CASE 2
  
  x = X(1); y = X(2)
  
  CALL G1beta(xx, yy, 0!, retCost)
  
  CASE 3
  
  x = X(1); y = X(2)
  
  CALL G1beta(xx, yy, 1!, retCost)
  
  CASE 4
  
  x = X(1); y = X(2)
  
  CALL G1beta(xx, yy, 5!, retCost)
  
  CASE 5
  
  x = X(1); y = X(2)
  
  CALL G1beta(xx, yy, 6!, retCost)
  
  CASE 6
  
  x = X(1); y = X(2)
  
  retCost = (4 - 2.1*x*x - x*x*x + 2.0*x*y + 4.0*y*y) + 4.0
  
  CASE 7 TO 9
  
  GOSUB 9
  
  retCost = retG3
  
  CASE 10 TO 12
  
  GOSUB 12
  
  retCost = retG4
  
  CASE 13 TO 18
  
  GOSUB 18
  
  retCost = retG4
  
  CASE ELSE
  
  PRINT "No corresponding problem": RETURN
  
  END SELECT
  
  iCost& = iCost& + 1

ELSE

  epsRndFac = (T / intIT) ^ .75
  
  IF (epsRndFac <= .03) THEN epsRndFac = .03
  
  END IF
  
  RETURN

UpdateT:

- update T by ICCAD86 ---

  IF (LastGasp% > 1) THEN
  
  retExitEXTLoop% = 1
  
  T = .5 * T
  
  ELSE
  
  T = .9 * T
  
  IF (relCost > .3) THEN
  
  T = .9 * T
  
  ELSE
  
  T = .9 * T
  
  END IF
  
  RETURN

END IF
END IF
RETURN

UpdateEpsRndFac:

- update epsilon for random part ---

  IF (LastGasp% >= 14) AND (epsRndFac < .00001) THEN retExitINTLoop% = 1

RETURN

UpdateINTCost:

FOR% = 1 TO n%

okX(i%) = oldX(i%)

NEXT
RETURN

ResumeOptX:

FOR% = 1 TO n%

iOpt% = 0

NEXT
RETURN

ResumeOldOptX:

FOR% = 1 TO n%

oldOptX(i%) = X(i%)

NEXT
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitEXTLoop:

- EXT loop exit condition, frozen condition ---

  retExitEXTLoop% = 0

  IF (EXT% > maxnEXT%) THEN

  retExitEXTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitEXTLoop:

- EXT loop exit condition, frozen condition ---

  retExitEXTLoop% = 0

  IF (EXT% > maxnEXT%) THEN

  retExitEXTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN

ExitINTLoop:

- INT loop exit condition ---

  retExitINTLoop% = 0

  IF (INT% >= maxINT%) THEN

  retExitINTLoop% = 1

  RETURN

END IF
END IF
RETURN
Simulated Diffusion Program for a Set of Test Functions

RETURN

**G2:**
\[ k2 = 10 \]
\[ a2 = 1 \]
\[ y1 = 1 + (X(1) - 1)/4 \]
\[ retG2 = k2 \times (\text{SW}(\pi \times y1)) \]
\[ FOR \ i = 1 \ TO \ n_i \%-1 \]
\[ y2 = 1 + (X(i) - 1)/4 \]
\[ retG2 = retG2 + (y2 - a2) \times (1 + k2 \times (\text{SW}(\pi \times y2))) \]
\[ NEXT \]
\[ retG2 = retG2 + (y2 - a2) \]
\[ retG2 = retG2 + 0.001 \]
RETURN

**G3:**
\[ k3 = 10 \]
\[ a3 = 1 \]
\[ y1 = X(1) \]
\[ retG3 = k3 \times (\text{SW}(\pi \times y1)) \]
\[ FOR \ i = 1 \ TO \ n_i \%-1 \]
\[ y2 = X(i) \]
\[ y21 = X(i+1) \]
\[ retG3 = retG3 + (y2 - a3) \times (1 + k3 \times (\text{SW}(\pi \times y21))) \]
\[ NEXT \]
\[ retG3 = retG3 + (y21 - a3) \]
\[ retG3 = retG3 + 0.001 \]
RETURN

**G4:**
\[ k4 = 1; k5 = 1; a4 = 1; \]
\[ y1 = X(1) \]
\[ retG4 = (\text{SIN}(\pi \times y1)) \]
\[ FOR \ i = 1 \ TO \ n_i \%-1 \]
\[ y2 = X(i) \]
\[ y21 = X(i+1) \]
\[ retG4 = retG4 + (y2 - a4) \times (1 + k5 \times (\text{SIN}(\pi \times y21))) \]
\[ NEXT \]
\[ retG4 = retG4 + (y21 - a4) \]
\[ retG4 = retG4 + k4 \times retG4 + 0.001 \]
RETURN

GaussRand:
--- gaussian distribution (see p.217 of NR) ---
\[ \text{IF} (\text{GaussCalled} = 0) \text{ THEN} \]
\[ \text{gaussR} = 2 \]
\[ \text{WHILE} (\text{gaussR} > 1) \]
\[ \text{gaussV1} = 2 \times \text{RND}(1) - 1 \]
\[ \text{gaussV2} = 2 \times \text{RND}(1) - 1 \]
\[ \text{gaussR} = \text{gaussV1} \times \text{gaussV1} + \text{gaussV2} \times \text{gaussV2} \]
\[ \text{END} \]
\[ \text{gaussFac} = \text{SQRT}(2 \times \text{LOG}(\text{gaussR}) / \text{gaussR}) \]
\[ \text{gaussSet} = \text{gaussV1} \times \text{gaussFac} \]
\[ \text{GaussCalled} = 1 \]
\[ \text{GaussRand} = \text{gaussV2} \times \text{gaussFac} \]
\[ \text{ELSE} \]
\[ \text{GaussRand} = \text{gaussSet} \]
\[ \text{GaussCalled} = 0 \]
\[ \text{END} \]
RETURN

LorentzRand:
--- Lorentzian distribution (see p.217 of NR) ---
\[ \text{LorentzRand} \]
\[ \text{lorentzR} = \text{RND}(1) \]
\[ \text{IF} (\text{lorentzR} < 3.141592 \times 1.1 / 2) \text{ AND} (\text{lorentzR} > 3.141592 \times .9 / 2) \]
\[ \text{THEN} \text{GOTO LorentzLoop} \]
\[ \text{lorentzRand} \]
\[ \text{lorentzR} = \text{TAN}(3.141592 \times \text{lorentzR}) \]
RETURN

EXTReport:
--- External loop report ---
\[ \text{WINDOW1} \]
\[ \text{PRINT} #1, "Problem:="; \text{"EXTCost"}; \text{" Time=";} \text{T} \]
\[ \text{PRINT} #1, "Problem:="; \text{"EXTCost"}; \text{"{RET} RandFac; \text{"Cost="}\text{COST} \]
\[ \text{PRINT} #1, "{RET} RandFac; \text{"Cost="}\text{COST} \]
\[ \text{PRINT} #1, "Accept\% unacceptable"; \text{"Accept\% unacceptable"} \]
\[ \text{GO} \]
\[ \text{SUB} \]
\[ \text{EXTReport} \]
\[ \text{IF} (\text{MOUSE}(0) = 0) \text{ THEN} \text{GOTO EXTReportLoop} \]
RETURN