

Czech Technical University in Prague  
Faculty of Electrical Engineering  
Department of Computer Science and Engineering



Dissertation Thesis Proposal DCSE-DTP-2005-04

## **MOSFET Parameter Extraction**

*Tomáš Zahradnický*

PhD program: Computer Science and Engineering

*Supervisor: Róbert Lórencz*

September 2005

This research has been supported by the Ministry of Education, Youth and Sports under the research program #J04/98:212300014.

.....  
*Tomáš Zahradnický*  
PhD student

.....  
*Róbert Lórencz*  
Supervisor

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
<b>2</b>	<b>Basic Definitions</b>	<b>4</b>
2.1	SPICE Models . . . . .	4
2.2	Error Analysis . . . . .	4
2.2.1	Numerical Errors . . . . .	4
2.2.2	The Purpose of Rounding Error Analysis . . . . .	6
2.3	Floating Point Arithmetic . . . . .	7
2.3.1	Floating Point Calculations . . . . .	9
2.4	The Arithmetic of Residual Number System . . . . .	10
2.4.1	Principles of RNS Computation . . . . .	10
2.4.2	Error Analysis in RNS . . . . .	10
2.5	Arbitrary Precision Arithmetics . . . . .	11
2.5.1	GNU MP/MPFR . . . . .	11
<b>3</b>	<b>Previous Work</b>	<b>12</b>
3.1	MOSFET Overview . . . . .	12
3.2	How MOSFETs Work . . . . .	13
3.3	MOSFET Mathematical Model . . . . .	14
3.4	MOSFET Mathematical Model Parameters . . . . .	16
3.5	Parameter Extraction Method . . . . .	18
3.6	Optimization Methods . . . . .	19
3.6.1	Simulated Annealing . . . . .	19
3.6.2	Evolutionary Computing . . . . .	20
3.6.3	Genetic Algorithms . . . . .	20
3.6.4	Multiobjective Optimization . . . . .	21
3.6.5	Ant Colony Optimization . . . . .	22
3.6.6	Tabu Search . . . . .	23
3.6.7	Levenberg-Marquardt's algorithm . . . . .	23
3.7	Other Parameter Extraction Methods . . . . .	24
3.7.1	Genetic algorithms & Multiobjective optimization . . . . .	24
3.7.2	Other LMA Algorithms . . . . .	25
3.7.3	Miscellaneous Extraction Algorithms . . . . .	25
<b>4</b>	<b>Main Results</b>	<b>26</b>
4.1	Jacobi's Matrix & Numerical Derivative Evaluation . . . . .	26
4.1.1	Numeric Derivative Approximation . . . . .	26
4.1.2	Analytical Derivative Evaluation . . . . .	28
4.2	Mdouble Arithmetic . . . . .	36
4.2.1	Mdouble Structure . . . . .	36
4.2.2	A-posteriori Error Estimates . . . . .	38
4.2.3	A-posteriori Error Estimates of Basic Operators . . . . .	38
4.2.4	A-posteriori Error Estimates of Addition & Subtraction . . . . .	38
4.2.5	A-posteriori Error Estimate of Multiplication . . . . .	38
4.2.6	A-posteriori Error Estimate of Division . . . . .	38
4.3	A-posteriori Error Estimates of Basic Functions . . . . .	39
4.3.1	A-posteriori Error Estimate of Square Root . . . . .	39

4.3.2	A-posteriori Error Estimate of Logarithm . . . . .	39
4.3.3	A-posteriori Error Estimate of Exponential Function . . . . .	40
4.3.4	A-posteriori Error Estimate of Generic Power Function . . . . .	40
<b>5</b>	<b>Experiments and/or Evaluation of Results</b>	<b>41</b>
5.1	Mathematical Model Approximation . . . . .	41
5.1.1	Neural Networks Conclusion . . . . .	42
5.2	Model Speedup . . . . .	43
5.2.1	Model Speedup Conclusions . . . . .	45
5.3	Root Mean Square Evaluation . . . . .	45
5.4	Comparing Various Arithmetics . . . . .	46
<b>6</b>	<b>Conclusions</b>	<b>48</b>
6.1	Parameter Extraction . . . . .	48
6.2	A-posteriori Error Estimates . . . . .	48
6.2.1	Derivative evaluation . . . . .	48
6.3	Further Research Possibilities . . . . .	49
6.3.1	Extend A-posteriori Estimates . . . . .	49
6.3.2	Perform System Level Optimization of Numeric Codes . . . . .	49
6.3.3	Make Parallel Extraction System . . . . .	49
6.3.4	Perform A-posteriori Error Estimates of RNS . . . . .	49
6.3.5	Perform Self Stability Checks . . . . .	49
<b>7</b>	<b>Bibliography</b>	<b>50</b>
<b>8</b>	<b>Relevant refereed publications of the author</b>	<b>52</b>
<b>9</b>	<b>Remaining refereed publications of the author</b>	<b>53</b>
<b>10</b>	<b>Unrefereed publications of the author</b>	<b>53</b>
<b>11</b>	<b>Dissertation Thesis</b>	<b>54</b>
11.1	Parallel, Numerically Stable MOSFET Parameter Extraction . . . . .	54
11.2	Self-Validated MOSFET Parameter Extraction . . . . .	54

## List of Figures

2.1	Normalized floating point number of $M$ set . . . . .	8
3.1	N-MOSFET Transistor . . . . .	12
3.2	N-MOSFET DC Characteristic . . . . .	12
3.3	MOSFET Internals . . . . .	13
3.4	Extraction Algorithm . . . . .	18
3.5	A typical genetic algorithm . . . . .	20
3.6	Typical multiobjective optimization algorithm . . . . .	21
3.7	A typical ACO algorithm . . . . .	22
5.1	Mismodelled DC characteristic . . . . .	41
5.2	Modelled DC characteristic . . . . .	41
5.3	Sensitivity analysis of mathematical model . . . . .	42
5.4	The progress of <i>rms</i> error . . . . .	46
5.5	Relative running error bound for <i>rms</i> error . . . . .	46
5.6	SLE Solve Time for various arithmetics . . . . .	47

## List of Tables

2.1	Floating point precision . . . . .	7
2.2	IEEE 754 arithmetic precision and exponent bounds . . . . .	10
3.1	MOSFET Model Parameters . . . . .	16
4.1	Compile time observations . . . . .	36
5.1	Various mathematical model evaluation times . . . . .	42
5.2	Profile data of parameter extraction . . . . .	43
5.3	Profile data of original MOSFET mathematical model function . . . . .	44
5.4	Profile data of improved MOSFET mathematical model function . . . . .	45

## Abbreviations

### Number sets

$\mathbb{N}$	Natural numbers set
$\mathbb{N}_0$	Natural numbers set $\cup \{0\}$
$\mathbb{Z}$	Integer numbers set
$\mathbb{Q}$	Rational numbers set
$\mathbb{F}$	Floating point numbers set
$\mathbb{I}$	Irrational numbers set
$\mathbb{R}$	Real numbers set
$\mathbb{C}$	Complex numbers set

### Common mathematical functions and operators

$10_2$	Numbers' radices are designated with a subscript
$\mathbf{b}$	Vector $\mathbf{b}$
$b_i$	$i$ -th element of vector $\mathbf{b}$
$\ \mathbf{b}\ $	Norm of vector $\mathbf{b}$
$\mathbf{A}$	Matrix $\mathbf{A}$
$a_{i,j}$	Element of matrix $\mathbf{A}$ at $i$ -th row and $j$ -th column
$\mathbf{A}^{-1}$	Inverse matrix to matrix $\mathbf{A}$
$\mathbf{A}^T$	Transposed matrix to matrix $\mathbf{A}$
$\ \mathbf{A}\ $	Norm of matrix $\mathbf{A}$
$\text{rank } \mathbf{A}$	Rank of matrix $\mathbf{A}$ — how many independent rows/columns it has
$\max\{a, b\}$	Maximum of $a$ and $b$ , $a$ when $a \geq b$ , $b$ when $a < b$
$\min\{a, b\}$	Minimum of $a$ and $b$ , $a$ when $a \leq b$ , $b$ when $a > b$

### Miscellaneous abbreviations

<b>AU</b>	Arithmetic unit
<b>FPU</b>	Floating point unit
<b>RNS</b>	Residual number system
<b>SLC</b>	Set of linear congruencies
<b>SLE</b>	Set of linear equations

# MOSFET PARAMETER EXTRACTION

*Tomáš Zahradnický*

`zahradt@cslab.felk.cvut.cz`

Department of Computer Science and Engineering

Faculty of Electrical Engineering

Czech Technical University in Prague

Karlovo nám. 13, 121 35 Prague 2, CZ

## Abstract

This dissertation thesis proposal presents the parameter extraction methodology and verifies it on a case study of a MOSFET mathematical model parameter extraction. The presented method is based on the nonlinear least squares problem which is solved with Levenberg-Marquardt's algorithm. In order to know the accuracy of the computed solution, a-posteriori error estimates are employed to obtain a sharp, a posteriori error bound which provides an insight into algorithm's numerical stability. Dissertation thesis proposal also presents results and experiments that have been performed and compares the methodology with other parameter extraction approaches from the speed of computation, method of obtaining parameters and error points of view.

## Keywords

MOSFET, Optimization, Parameter Extraction, Regularization, A-posteriori Error Estimates

## 1 Introduction

### 1.1 Motivation

Parameter extraction is an important section of the parameter estimation theory. Its purpose is to find out the unknown values of a function which represents physical relationships of dependent and independent variables on the values of that function. The typical scenario is that we measure repeatedly values of  $f(\mathbf{x}, \mathbf{p})$  where  $\mathbf{x}$  is a known vector and  $\mathbf{p}$  are unknown parameters. Function  $f(\mathbf{x}, \mathbf{p})$  is called mathematical model and describes the dependency of  $f(\mathbf{x}, \mathbf{p})$  on its arguments. The task of parameter extraction is to find out the values of parameter vector  $\mathbf{p}$  of the mathematical model, so it represents the measured data as close as possible by means of an error function being minimal. Such error function can be a root mean square error (*rms*) or likelihood for example.

Parameter extraction can be used in various scientific areas including climatology, economy, electronic components parameter estimation, medicine, nuclear physics, neutron spectrum analysis, propulsion and traction research, stock price forecast, weather forecast, and much more. All of these scientific areas tend to model relationships of output data on input data and have to use parameter extraction in some way, though they do not necessarily need to call their methods parameter extraction.

Parameter extraction is necessary because we can rarely measure all variables of a characteristic that represents particular physical reality. Independent parameters can be measured either all at once or in groups. The former "all-at-once" measuring approach may be very

expensive or even not possible at all, hence a parameter often influences several physical effects and cannot be measured. Such parameter is called nonphysical as it does not correspond to any physical reality. The latter possibility is to measure parameters in small groups and unite groups together into a parameter set once all measurements have been conducted. The most obvious problem of the “group-by-group” approach is that each group depends on the environment data was obtained in, and parameters in groups may depend on each other (correlate). Measuring environment often includes impedance of the measuring tool, relative humidity of the lab, local lab temperature, measuring device errors, gross error and so on. Neither of the approaches mentioned offers a way of getting values of nonphysical parameters that may be involved in the mathematical model and the use of parameter extraction is the only way for obtaining values for such nonphysical parameters.

We have chosen to verify our parameter extraction approach on a case study of MOSFET. Though the parameter extraction problem is nonlinear, it needs to be linearized with Taylor series expansion which requires derivative of the mathematical model. We have chosen MOSFET for several reasons; because we have a complete measured data set including mathematical model which is continuous in its entire operating area and has derivative defined at any point.

Having available such “derivative safe” mathematical model as presented in section 3.3, makes several portions of the extraction algorithm significantly easier and we can concentrate more on the methodology itself rather than on a particular sub-problem such as solving discontinuities in model’s derivatives which is pretty complex as it involves dealing with a special version of the mathematical model for each MOSFET operating region and usually involves smoothing functions.

Despite this simplification, MOSFET mathematical model parameter extraction still features several problems and the most important are namely the problem of solving a set of linear equations (core set) in a computer with a limited computer word and model derivative evaluation. To know more about the problem, additional knowledge is necessary and it is obtained by several analyses that are rounding error analysis, sensitivity of a parameter set analysis and statistical evaluation of the data.

The problem of solving a set of linear equations is very simple until brought to a computer. Since every computer is a finite state machine (FSM), the number of states it can represent is finite and so is the word it can represent. This fact leads to an inevitable discretization of real number set into rational numbers and only a subset of rational numbers (floating point numbers) can be represented in a computer. Most current computers support the use floating point arithmetic defined by IEEE 754 standard [20] by means of floating point unit (FPU) which handles the most common operations with `float` (32bit) and `double` (64bit) data types. Floating point arithmetic is used in most scientific evaluations and is described in section 2.3 on page 7.

Unfortunately, the floating point arithmetic introduces several factors that are unique to it such as rounding and these problems may lead to a degradation of the result or make the result useless. If we are aware that such problems exist, we can attempt to modify the algorithm to be less susceptible to the burden of rounding errors.

Although the algorithm may be numerically stable, the problem of parameter extraction is ill-conditioned and the task itself is ill-posed and cannot be solved without an additional knowledge about the problem. Being problem an ill-conditioned means that even a very small error (perturbation) to the input data can project into the output significantly. Such ill-conditioned problem often features ill-conditioned matrices if matrix based. An ill-conditioned matrix tends to be close to being singular, and a large number of solutions may potentially exist in a limited floating point arithmetic. One possible solution to the ill-conditioned problem is to find out a different problem which does not need to be ill-conditioned. In the case when we are unable to find such well-conditioned problem, we have to employ problem regularization



techniques.

To make the problem better conditioned, a technique called regularization is used to effectively lower the condition of the core set of linear equations. Once the core set of linear equations has its condition number reduced, the numerical rank of the matrix grows, the sensitivity of the problem diminishes and with the additional knowledge about the problem, we are able to find a satisfactory solution.

The proposed algorithm needs to be numerically stable to not get affected with rounding errors much and rounding error analysis can help us to replace most of potentially unstable portions of the algorithm with numerically stable ones. This ensures that rounding errors and noise in the input do not affect the computed solution in a large way.

Another important question is that the input data comes with some error. There are errors that input data comes with such as the error of measurement device and gross error followed with the error of bringing them to the computer. We are interested how errors in the input data get projected to the output and therefore an error analysis is required to be able to tell how stable the proposed problem is and how much the input error and ill-conditioning affects the solution. Dissertation thesis proposal proposes to use the running error analysis [43] which is described in section 2.2.2.3 on page 7 thorough the entire calculation so we can tell an a-posteriori error bound which is calculated concurrently with the result.

The report is organized as follows. Section 2 introduces basic definitions and terminology including SPICE model overview, common optimization methods overview, floating point introduction and a brief error analysis description. Section 3 summarizes the previous results. Section 4 describes our solution of the problem. Section 5 discusses the achieved results. Section 6 concludes with outlines for future work.

## 2 Basic Definitions

This chapter provides a brief introduction into terminology and notation that is used thorough this dissertation thesis proposal. Chapter starts with SPICE models and continues with error analysis, floating point arithmetic as described by IEEE 754 Standard [20], the arithmetic of residual number system and arbitrary precision arithmetics.

### 2.1 SPICE Models

SPICE is an acronym for Simulation Program with Integrated Circuit Emphasis and the word SPICE also denotes applications and/or models that model behavior of devices used in integrated circuits. SPICE is also a common name of an industry standard both electric and electronic simulation tool. This tool is developed in various brands and each brand possesses its own vendor specific extensions. The most common commercial versions of SPICE are B2 Spice v.5 [3] developed by BeigeBag Software which can be found at <http://www.beigebag.com/>, Orcad 9.1, and MicroSim DesignLab 8. Vast majority of commercial versions of SPICE are based on Berkeley's SPICE2 version G6 predecessor developed by University of California at Berkeley though SPICE3 is currently under development.

A class of device models denoted with the SPICE word are such ones that accurately describe the device being modeled and also can be used within SPICE tools. These models exist for a broad spectrum of analog components like capacitors, diodes, fuses, inductors, transistors (BJT, JFET, MOSFET), triacs, vacuum tubes, varistors, and digital components like comparators, opamps, and more and are usually supplied by devices's manufacturer. There also exists a lot of custom models and there may exist several models for the same device. For a list of around 20500 SPICE models look at Paul Hill's website <http://homepages.which.net/~paul.hills/Circuits/Spice/ModelIndex.html>.

Each (not only SPICE) model is characterized with a set of parameters and these parameters have to be estimated if they cannot be measured either easily or not at all. The process of estimating model parameters is called parameter extraction and a brief description of various optimization methods that can be used to control the progress of a parameter extraction algorithm follows. This dissertation thesis proposal discusses parameter extraction of one of many SPICE models that exist for a Metal Oxide Substrate Field Effect Transistor (MOSFET) and presents various gotchas that were necessary to successfully obtain model's parameters and presents possible ways of our further research. The SPICE mathematical model of the MOSFET device is described in section 3.3 on page 14.

### 2.2 Error Analysis

This section emphasizes the necessity of rounding error analysis of the parameter extraction algorithm we use together with principles of rounding errors such as where they come from. The section also briefly discusses the two most widely used deterministic rounding error analyses that are forward error analysis and backward error analysis. Among that, section discusses a-posteriori error estimates (running error analysis) which are stressed and used thorough this dissertation thesis proposal. Stochastic error analyses are not discussed in this dissertation thesis proposal.

#### 2.2.1 Numerical Errors

As mentioned before, only a subset of rational numbers known as floating point numbers can be represented within the computer. The transformation of a rational number into a computer

floating point number is described in section 2.3.1 on page 9.

One of the properties of  $\mathbb{F}$  set is that it is closed to its operations and therefore the result of each floating point operation is also a member of  $\mathbb{F}$ . The  $\mathbb{F}$  set contains few special elements that represent underflow, overflow, infinity and not a number results. Due to the fixed length of a computer word, the result of a floating point operation has to be rounded to the (usually) nearest element of  $\mathbb{F}$  and rounding error may happen in this way.

In addition to rounding and truncation errors, there are additional errors such as modeling error, error of the method, etc. and each of the mentioned errors can be generated, propagated and can get magnified with the computation or a particular order of operations as the common rules such as associativity and distributivity do not need to hold in the floating point arithmetic. A result of a floating point operation presents rather an approximation to the exact result and error analysis provides an insight at analyzed algorithm's numerical stability (or a lack of it) and the accuracy of the result.

### 2.2.1.1 The most common errors

This section describes the most common kinds of errors. These errors can either appear before the actual computation starts such as discretization error or gross error or can happen during the calculation such as roundoff error. Both kinds of errors can play significant role during the calculation and their description follows:

#### Discretization Error

Discretization is a process which transforms a continuous problem into a discrete one in order to work with it in the computer. There is an error involved with this conversion process which is called discretization error.

#### Truncation Error

Some numbers such as  $e$ ,  $\pi$  or  $\sqrt{2}$  are irrational and can be expressed with an infinite expansion. In order to bring such numbers into a computer, their infinite expansion has to be truncated and an error, known as a truncation error, appears.

#### Gross Error

Sometimes, a gross error [37] is also defined such as an error of measurement, or an error of human being when reading values from measuring device.

#### Convergence Error

This error obviously appears when we prematurely terminate a process which converges to some value or from a divergent process.

#### Overflow and Underflow Errors

We can avoid these floating point errors by appropriately scaling the data we have and by employing normalization techniques.

#### Roundoff Error

May happen if the length of a result of one floating point operation is larger (due to a normalization process of floating point numbers) than the defined length and, as the floating point set  $\mathbb{F}$  is closed to its operations, the result has to be rounded to the nearest element of  $\mathbb{F}$ . This incurs errors as each arithmetic step of an algorithm, and the propagation and possible amplification of errors incurred in preceding steps [37].

### 2.2.1.2 Floating point errors

Floating point errors are consequences of a calculation performed in the floating point arithmetic and these include:

#### Cancellation error

Cancellation error occurs when we subtract two very close floating point numbers. The result of the cancellation is practically only the error of the numbers consequently leads to the amplification of the errors of the subtracted numbers.

#### Inherent error

Inherent error is a built-in error which appears at the beginning of the computation as a result of the data being not exact.

#### Roundoff

Roundoff may cause an error if we add/subtract one number of large magnitude with a number of a small magnitude. The result may be that the sum/difference is only the number with the larger magnitude and the other number actually either does not contribute to the sum/difference at all or contributes very little.

#### Under/Overflow

Cancellation error in the denominator of a fraction can often cause this kind of errors when the value of denominator is close to zero and the result will get dramatically magnified which can lead to overflow error. In the reverse case, where the denominator is huge, we can get so small result that we are unable to represent it and in that case underflow occurs.

## 2.2.2 The Purpose of Rounding Error Analysis

The purpose of either error analysis is to show the existence of some a priori or a posteriori error bound as a measure of the effects caused by rounding errors committed in the algorithm. The error bound for given algorithm does not need to exist and the question whether it exists is important [18]. If the bound exists, we desire to make it small. However, if the bound does not exist, error analysis should reveal features of the algorithm that characterize any potential instability and thereby suggest how the instability can be cured or avoided [18]. The two error analyses are briefly outlined within the next four sections.

### 2.2.2.1 Forward error analysis

Forward error analysis [18, 43, 42] of an algorithm inspects each step of that algorithm and determines its absolute (or relative) partial error bound. Combining the obtained partial bounds together, we obtain a forward error bound of the computed solution in a form  $|\bar{y} - y|$ , where  $\bar{y}$  denotes the computed value of  $y$ . The error bound estimated by this analysis is usually much larger than the actual error as it does not depend on the actual rounding error committed and presents rather the worst-case behavior. The forward error analysis is usually difficult to apply on a complex algorithm.

### 2.2.2.2 Backward error analysis

Backward error analysis [6, 8, 18, 37, 43] is the method of bounding the backward error  $|\bar{x} - x|$  of the computed solution by treating rounding errors as perturbations in the data instead of

tracking absolute/relative errors at each step of the algorithm. The purpose of this analysis is to show that the computed solution  $\bar{y}$  is exact for some neighboring problem  $\bar{x}$ , which we obtain by perturbing the data we already have. There may exist an infinite number of  $\bar{x}$ s, one or no  $\bar{x}$  at all.

### 2.2.2.3 A-posteriori estimates (Running error analysis)

Forward error bound does not depends on the actual rounding errors committed. If we want a sharper, a-posteriori error bounds, we can use technique known as running error analysis [18, 43] to find a-posteriori error estimates, where the error bound is computer concurrently with the solution and stands atop of the idea that:

$$|x S y - \text{fl}(x S y)| \leq u |\text{fl}(x S y)|, \quad (2.1)$$

where  $S$  stands for some operator. The advantage of this approach is that the error bound is easily computed and the actual immediate quantities are used. Running error bound is smaller than an a priori one because it depends on the actual errors committed and does not cover the worst-case. Errors committed during the running error bound evaluation are negligible for  $nu \ll 1$  since we do not need many significant digits of an error estimate. This dissertation thesis proposal evaluates a-posteriori error estimates in its `mdouble` arithmetic and bounds for common arithmetic operations and functions are derived in section 4.2.2 on page 38.

## 2.3 Floating Point Arithmetic

Since most of our computations happen in the floating point arithmetic, it is important to define it together with the common terms we use thorough this dissertation thesis proposal such as roundoff unit, (sub)normal numbers, etc. We will denote a set of floating point number as  $\mathbb{F}$  and it is obvious that  $\mathbb{F} \subset \mathbb{Q} \subset \mathbb{R}$ . All elements of the floating point set  $\mathbb{F}$  have the form:

$$f = (-1)^s \cdot m \cdot 2^{e-t}, \quad (2.2)$$

where  $s$  is the sign and  $s \in \{0, 1\}$ ,  $m$  stands for mantissa<sup>1</sup>,  $e$  for an exponent, 2 is the base of the arithmetic, and  $t$  is the precision. This text does not tell much about additional possible bases such as 10 or 16, and sticks with the IEEE 754 Standard. For more information about floating point number arithmetic refer to [18, 20]. The Standard defines three kinds of precisions known as *single*, *double* and *extended* precision but we will not deal with the *extended* precision much in this text. The following table wraps all three types of the floating point numbers:

	sign [b]	Exponent [b]	Mantissa [b]	Precision [dig]	Total [b]
<b>single</b>	1	8	23	8	32
<b>double</b>	1	11	52	16	64
<b>extended</b>	1	15	64	20	80

Table 2.1: Floating point precision

Another way of expressing a floating point number  $f$  is

$$f = (-1)^s \cdot 2^e \left( \frac{d_1}{2} + \frac{d_2}{4} + \dots + \frac{d_t}{2^t} \right) = (-1)^s 2^e \times .d_1 d_2 d_3 d_4 \dots d_t, \quad (2.3)$$

---

<sup>1</sup>Mantissa is sometimes called significand to prevent confusion with logarithms.

where  $d_i$  are individual digits of  $m$  and each  $d_i \in \{0, 1\}$ . Since  $f$  can be written in a number of ways ( $2 = 1, 0_2 \cdot 2^1 = 0, 1_2 \cdot 2^{10_2} = 0, 01_2 \cdot 2^{11_2} = \dots$ ), the equation (2.2) is ambiguous. To deal with this problem and to use mantissa effectively to keep maximum amount information, normalized numbers have been introduced. Mantissa of a floating point number can be expressed also as

$$m = d_1 d_2 d_3 \dots d_t. \quad (2.4)$$

and number  $f$  is pronounced normalized (normal) if and only if  $d_1 = 1$  and that yields  $m \geq 2^{t-1}$ , where  $t$  is the precision. Zero is the only exception of normalized number where  $m = 0$ .

**Example 1.** The problem of normalized numbers is that they are inequally spaced and their distance is wider with the distance from zero. For a floating point system where mantissa  $t = 3$  and  $e_{min} = -1$  and  $e_{max} = 3$ . Normalized floating point numbers are those, whose mantissa  $m > 2^{t-1}$  and 0. Since precision is 3, number whose mantissa is normalized are 4, 5, 6, and 7. The set of all normalized floating point numbers is an union of subsets  $M_4$  through  $M_7$  assuming we omit all floating point numbers with the negative sign since this set is symmetric around zero. That yields:

$$\begin{aligned} M &= M_4 \cup M_5 \cup M_6 \cup M_7 \cup \{0\} \\ M_4 &= \left\{ 4 \cdot 2^{e-3} \mid e = -1 \dots 3 \right\} = \{0.25, 0.5, 1, 2, 4\} \\ M_5 &= \left\{ 5 \cdot 2^{e-3} \mid e = -1 \dots 3 \right\} = \{0.3125, 0.625, 1.25, 2.5, 5\} \\ M_6 &= \left\{ 6 \cdot 2^{e-3} \mid e = -1 \dots 3 \right\} = \{0.375, 0.75, 1.5, 3, 6\} \\ M_7 &= \left\{ 7 \cdot 2^{e-3} \mid e = -1 \dots 3 \right\} = \{0.4325, 0.875, 1.75, 3.5, 7\} \end{aligned} \quad (2.5)$$

The entire set  $M$  of normalized nonnegative numbers consists of 21 floating point numbers. These numbers are:  $M = \{0, 0.25, 0.3125, 0.375, 0.4325, 0.5, 0.625, 0.75, 0.875, 1, 1.25, 1.5, 1.75, 2, 2.5, 3, 3.5, 4, 5, 6, 7\}$ . To point out the nonequidistant spacing in between the adjacent floating point numbers (members of  $M$  set), let's plot these numbers:

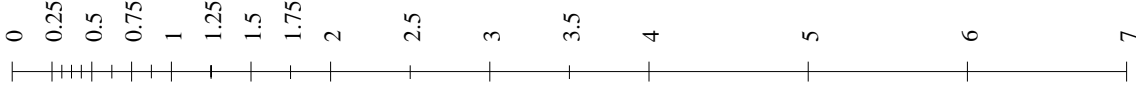


Figure 2.1: Normalized floating point number of  $M$  set

We can observe that gaps in between two adjacent floating point numbers grow twice each power of 2.

Numbers with mantissa  $0 < m < 2^{t-1}$  are called subnormal numbers. All subnormal numbers are denormalized. Subnormal numbers do not have full precision because there are less bits of precision capable of holding information since first  $i \geq 1$  digits of mantissa is always 0s. The entire floating point set  $\mathbb{F}$  is an union of normal and subnormal number sets and the following example lists the remaining portion of the  $\mathbb{F}$  set described by ex. 1.

**Example 2.** The set  $M$  from the previous example can be extended with subnormal numbers — elements of  $M_1$ ,  $M_2$ , and  $M_3$  sets and are defined as:

$$\begin{aligned} M_1 &= \left\{ 1 \cdot 2^{e-3} \mid e = -1 \dots 3 \right\} = \{0.0625, 0.125, 0.25, 0.5, 1\} \\ M_2 &= \left\{ 2 \cdot 2^{e-3} \mid e = -1 \dots 3 \right\} = \{0.125, 0.25, 0.5, 1, 2\} \\ M_3 &= \left\{ 3 \cdot 2^{e-3} \mid e = -1 \dots 3 \right\} = \{0.1875, 0.375, 0.75, 1.5, 3\} \end{aligned} \quad (2.6)$$

Note that numbers 0.25, 0.5, and 1 of set  $M_1$  are duplicate and should not be used to avoid unambiguity and a loss of accuracy. The same holds for the remaining two sets.

All nonnegative normal and subnormal numbers for given parameters  $t$ ,  $e_{min}$ , and  $e_{max}$  can be obtained with Mathematica software by using the following formulae:

```

normalnum[t_, emin_, emax_] :=
  Sort[Union[Flatten[Table[m*2^(e - t), {m, 2^(t - 1), 2^t - 1},
    {e, emin, emax}]]], {0}]]

subnormalnum[t_, emin_, emax_] :=
  Sort[Union[Flatten[Table[m*2^(e - t), {m, 1, 2^(t - 1) - 1},
    {e, emin, emax}]]]]]

```

### 2.3.1 Floating Point Calculations

Denote  $\text{fl } x$  the floating point representation of  $x \in \mathbb{R}$ . We know that  $x$  cannot be represented precisely in  $\mathbb{F}$  since this set is discrete in contrary to  $\mathbb{R}$ .  $\text{fl } x$  is a mapping  $\text{fl } x : \mathbb{R} \rightarrow \mathbb{F}$  which is surjective but not injective. Being this mapping non injective, a discretization error may occur since an interval of  $\mathbb{R}$  is mapped onto a single element of  $\mathbb{F}$ . Number  $x$  gets mapped (rounded) to the nearest element of  $\mathbb{F}$  with some relative error we define later.

IEEE 754 standard defines rounding modes for floating point arithmetic. Normally  $x$  is rounded to the nearest element (round to nearest) of  $\mathbb{F}$  but if it is closest to two adjacent floating point numbers, it gets rounded to the number of greater magnitude (round away from zero) or to the number with the last digit  $d_t = 0$  therefore the resulting mantissa is even (round to even). The rounding performed during the transformation of number  $x$  onto  $\mathbb{F}$  set can be understood as multiplying  $x$  by a constant representing some rounding error. Function  $\text{fl } x$  can then be written as:

$$\text{fl } x = x(1 + \delta), \quad |\delta| \leq u,$$

where number  $u = 2^{-t}$  and is called *unit roundoff* and stands for the largest relative error that can arise during the transformation  $\text{fl } x$ .

**Theorem 1** *All numbers of  $\mathbb{R}$  set can be mapped onto  $\mathbb{F}$  set with a relative error no greater than  $u$ .*

**Proof.** Prove theorem 1 for positive numbers. We can safely assume that  $x > 0$  because the proof is same for negative numbers. Let  $x = \mu \times 2^{e-t}$ , where  $\mu = 2^{t-1} \leq x < 2^t$ ,  $x \in \mathbb{R}$ .  $x$  can be rounded to either  $y_1 = \lfloor \mu \rfloor \cdot 2^{e-t}$  or  $y_2 = \lceil \mu \rceil \cdot 2^{e-t}$ . It is then obvious that  $x$  is bounded as follows:

$$y_1 \leq x \leq y_2$$

The absolute error of  $|\text{fl } x - x|$  is then no greater than a half of the distance of adjacent floating point numbers  $y_1$  and  $y_2$ . That gives the following inequality

$$|\text{fl } x - x| \leq \frac{|y_2 - y_1|}{2} \leq \frac{1}{2} 2^{e-t}$$

The relative error follows with the help of static error

$$\left| \frac{\text{fl } x - x}{x} \right| \leq \frac{\frac{1}{2} 2^{e-t}}{\mu \times 2^{e-t}} \leq 2^{-t} = u.$$

□

Since we know that any floating point number can be represented with relative error no greater than  $u$ , we can ask how much precise are the essential arithmetic operations such as addition, subtraction, multiplication and division. IEEE 754 standard arithmetic uses one level higher data type for evaluation and rounds the result back to the original desired single or double data type. Single precision evaluation uses double precision while double uses extended one. To relate IEEE 754 arithmetic with terms of this dissertation thesis proposal, present the exponent values for all three IEEE 754 data types:

type	$t$	$e_{min}$	$e_{max}$	$u$
single	24	-125	128	$2^{-24} \approx 5.96 \cdot 10^{-8}$
double	53	-1021	1024	$2^{-53} \approx 1.11 \cdot 10^{-16}$
extended	64	-16381	16384	$2^{-64} \approx 5.42 \cdot 10^{-20}$

Table 2.2: IEEE 754 arithmetic precision and exponent bounds

Note that  $t$  in tab. 2.2 is 1 higher than the number of bits. Since we require numbers be normalized ( $m > 2^{t-1}$ ), there is no necessity to write the first 1 and we can use one more bit of precision instead. This feature is called “hidden one”. Subnormal numbers can be differentiated from normal numbers by looking at exponent which is always minimal for subnormal numbers.

More information about the floating point arithmetic such as floating point exceptions are beyond the limits of this dissertation thesis proposal and can be found in IEEE 754 Standard [20] or [18].

## 2.4 The Arithmetic of Residual Number System

The arithmetic of residual number system (RNS) is a promising way of making the extraction algorithm to run in parallel. It stands atop of the Chinese residual theorem (CRT) [14, 15, 25, 35] which defines residual representation for each rational number and its back transformation onto a rational number. This decomposition can be particularly useful since all operations performed on rational numbers can also be performed on their remainder representation. The advantage of the remainder representation is that it presents  $n$  independent numbers rather than 1 rational number and all calculations are performed independently on the others. Such ability allows us to make the computation run in parallel and increase the computation speed significantly. Other advantage of the RNS is that there are no carry and no rounding during the computations which can potentially lead to better results. The only disadvantages of RNS are that there are no  $<$  and  $>$  operators and no division operator which is superseded with a multiplicative inverse instead.

### 2.4.1 Principles of RNS Computation

The principle of leveraging RNS arithmetic is to employ it to solve the core linear system in eq. 3.33 on page 18. Since each floating point number is a rational number, the forward transformation takes a floating point matrix and performs normalization on it. The normalization process of matrix  $\mathbf{A}$  is performed by extending each element of  $\mathbf{A}$  by  $10^{16}/a_{ij_{min}}$  factor, where  $a_{ij_{min}}$  is the smallest element of  $\mathbf{A}$ . Once the normalization process is done, the remainder representation is evaluated with  $M$  prime number modules. The number of modules  $M$  specifies the precision of the computation and the higher it is, the higher the precision of the computation is. After the remainder representation is obtained, a set of linear congruencies (SLC) is solved. Since the evaluation of particular SLCs is independent, it can be solved in parallel which is the major advantage of RNS. Having solved all  $M$  SLCs, the backward transformation is run and transforms results into a rational number again. A comparison of running time during SLE solving is presented in section 5.4.

### 2.4.2 Error Analysis in RNS

The fact that the RNS calculation is error free [14] (no error is generated) does not mean that there are no errors on the input. Due to this, we have to find a way how to propagate input errors through the RNS calculation and obtain them back during the backward transformation.



Once of the possible ways is to use running error analysis where we come from the fact that no error happens as a result of particular RNS operation. We can revise expressions in section 4.2.2 on page 38 by neglecting the first term (error generating term) of the final expression in each arithmetic operation. The only “new” error is the error of a multiplicative inverse which can be evaluated as an error of division, where  $a = 1$  and its error  $e = 0$ :

$$|g| \leq \left| \frac{f}{\widehat{b}(\widehat{b} - f)} \right| = \left| f(\widehat{b}^2 - \widehat{b}f)^{-1} \right| \quad (2.7)$$

## 2.5 Arbitrary Precision Arithmetics

The IEEE 754 arithmetic is limited with the length of the mantissa and exponent and needs to fit into a single computer word. To go around this problem, various arbitrary precision arithmetic have been developed and this section provides a brief overview of them. The most common is undoubtedly GNU MP [13] arithmetic and its floating point branch called MPFR [16]. There also exist other multi precision arithmetics such as SuperNumber library [40] but they are not covered by this dissertation thesis proposal.

### 2.5.1 GNU MP/MPFR

GNU MP arithmetic consists of several arithmetics that include integer (MPZ) arithmetic, rational number arithmetic (MPQ), floating point arithmetic (MPF) and extended floating point arithmetic with common functions (MPFR). Each of the mentioned arithmetics forms a special library and can be used independently of the other ones. All of these libraries have one thing in common and that is that they are based on MPN library which provides processor native version of the essential operations used by the libraries. MPN library is system optimized for almost all processors that include Alpha, AMD, Cray, Intel, Motorola 68K, Motorola 88K, PowerPC, Ultra Sparc and some more. Among the features supported there are AltiVec, MMX, SSE, SSE2, 3D Now and other SIMD type processor features. Being native allows MPN’s clients to get achieve evaluation speed. The following section described MPFR library in detail as it is one of the things considered.

#### 2.5.1.1 MPFR

MPFR C library is built atop of MPF which is a portion of GMP library. MPFR is of particular interest since it support reliable floating point arithmetic with extensible mantissa length. MPFR supports a variety of functions in contrary to its predecessor MPF and allows mantissa length be scaled at any time during the calculation. All of the functions are implemented by using appropriate series expansions and multiplication is done with FFT. There are also two disadvantages of this library and that is its sequentiality and the ability to make user responsible for mantissa length setting. If the mantissa length is very long, there is no possibility of splitting operation’s execution on more computers and the parallelism can only be applied at operation level. Moreover, user has to provide mantissa which is enough long or the computation can break and there is no easy way of doing so. When calculating with matrices, we can use eg. Hadamard’s estimation of determinant size to predict mantissa length but this mantissa tends to be unnecessarily long and computation time is therefore wasted. A comparison of running time during SLE solving is presented in section 5.4.

### 3 Previous Work

#### 3.1 MOSFET Overview

Transistors are used in a vast majority of electronic circuits both analog and digital. In analog circuits, transistors are used for current amplification which is controlled either with the input voltage for field effect transistors (FET)s, or with a an electric current for bipolar junction transistors (BJT)s. In digital circuits, transistors are used for switching purposes and almost arbitrary digital component contains transistors. Transistors are used to create gates — an essential building blocks of digital circuits, memories — static random access memory (SRAM) used commonly as L1 caches, for example, liquid crystal display (LCD)s that contain about 4 transistors per single pixel and much more components. Nowadays processors contain hundreds of thousands of transistors. Transistors superseded almost entirely vacuum tubes that are still used in few special applications such as audio amplification. The absolute pros for transistors are that they are significantly smaller than vacuum tubes and do not require a warm-up time as vacuum tubes do and have significantly longer lifetime. The warm-up time for a vacuum tube is an interval after the device is ready to use and may be even 1 minute for the most complex vacuum tubes. The only cons for transistors are that they are much less resistive against electromagnetic pulses.

The Metal Oxide Substrate Field Effect Transistor (MOSFET, IGFET<sup>1</sup>) is the first transistor type ever manufactured and is the most common FET transistor. Fig. 3.1 shows N-MOSFET transistor which consists of n-type semiconductor material and its channel is made of p-type material. N-type semiconductor materials are based on a doping process which adds certain types of atoms to the semiconductor to increase the number of free negative mobility carriers — electrons. The process of n-type doping produces an abundance of electrons in the material and therefore electrons carry the charge. The semiconductor is usually made of silicon (Si) which has 4 electrons in the VA layer and the other material which is put into the crystal lattice with the doping process may be phosphorus (P), arsenic (As) or antimony (Sb) or any other member of VA group of the periodic table.

Fig. 3.1 depicts n-type MOSFET where the four terminals: gate, drain, base, and source are depicted.  $V_{GS}$  (gate-source voltage) controls the value of  $I_{DS}$  (drain-source current) thus how much current flows in between drain and source by creating an inversion layer (electric field) which connects the circuit in between drain and source.  $I_{DS}$  and  $V_{GS}$  together with  $V_{DS}$  (drain-source voltage) and  $V_{BS}$  (base-source voltage) control the shape of the inversion layer and are the only variables we measure. These variables also form the shape of the drain-current (DC) characteristic which is depicted below at fig. 3.2:

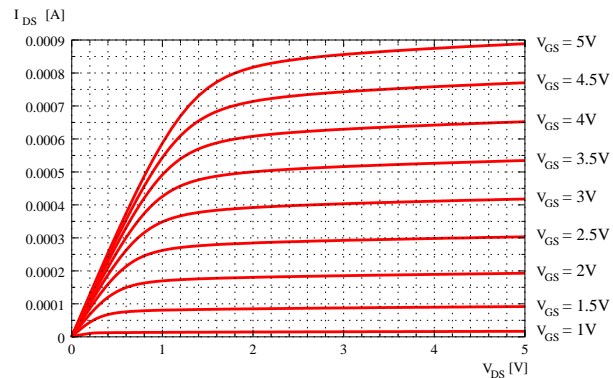
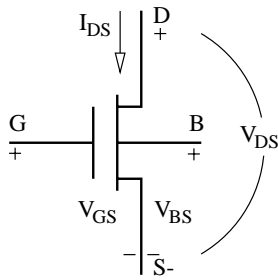


Figure 3.1: N-MOSFET Transistor

Figure 3.2: N-MOSFET DC Characteristic

<sup>1</sup>MOSFET may also be called IGFET because its gate is insulated with a gate oxide.

### 3.2 How MOSFETs Work

It is useful for this dissertation thesis proposal if we describe how MOSFET works. This helps us to understand the meaning of some of the parameters that appear in its mathematical model. These parameters will be listed later in this dissertation thesis proposal. The

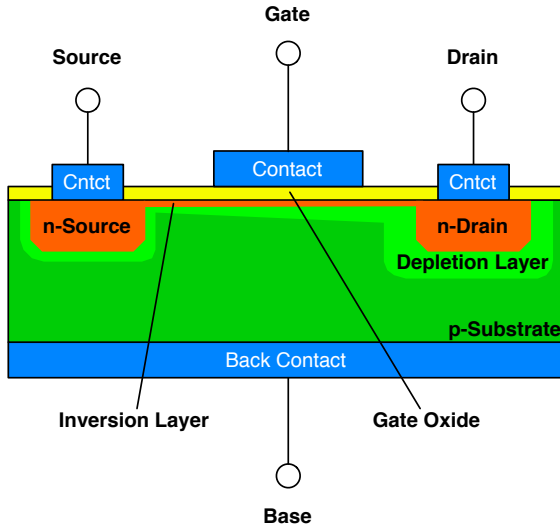


Figure 3.3: MOSFET Internals

age that is less than some threshold and form an electric field known as the “Inversion Layer” which connects the drain and source and closes the electric circuit. Electric current can now flow from in between source and drain and moreover drain (or source) can supply more electrons. Depending on the values of  $V_{DS}$  and  $V_{BS}$  the inversion layer changes its shape since electrons tend to stick to a positive source. Let us concentrate on drain. The same would apply for source in the exactly reverse order. Once the voltage on drain is smaller than some defined threshold voltage, the shape of the inversion layer is rectangular and the device is called to be in the linear (cf. fig. 3.3) mode. If we continue increasing the drain voltage above the threshold voltage, the shape of the inversion layer changes and becomes triangular. The device with a triangular inversion layer is called to be in a saturation (also called pinchoff) operation mode.

The current gain capability of a Field-Effect-Transistor (FET) is easily explained by the fact that no gate current is required to maintain the inversion layer and the resulting current between drain and source. The device has therefore an infinite current gain in DC. The current gain is inversely proportional to the signal frequency, reaching unity current gain at the transit frequency. The voltage gain of the MOSFET is caused by the fact that the current saturates at higher drain-source voltages, so that a small drain current variation can cause a large drain voltage variation [45].

The DC characteristic has already been presented at page 12. We are interested how the value of drain current  $I_{DS}$  depends on the values of  $V_{GS}$ ,  $V_{DS}$ , and  $V_{BS}$  and other parameters of the transistor that correspond to the physical reality such as substrate doping, oxide thickness, gate dimensions, threshold voltages, etc. The value of  $I_{DS}$  is transformed by means of a mathematical model where take all information that we know and with the use of mathematical modeling construct a function which presents the dependency. Such function is called a mathematical model and besides the physical parameters often introduces nonphysical parameters that do not necessarily need to correspond to some physical parameter. Such nonphysical parameters often combine the effects of one or more physical effects.

typical MOSFET is depicted at fig. 3.3 [45]: The central region of the picture 3.3 is called channel and is made of p-type substrate for N-MOSFET device. The other two portions, that are laid symmetrically in most cases, are source and drain that are made of n-type semiconductors. Since they are equivalent, it makes no difference which one we denote as drain or source. The gate oxide which is usually made of  $\text{SiO}_2$  insulates the gate terminal from the substrate and that is why the MOSFET is also called IGFET.

If we start with all voltages grounded and apply a positive voltage at gate ( $V_{GS}$ ), an electric field is created. This forces electrons move towards the gate oxide pushing out holes. Since the gate oxide is an insulator, electrons cannot pass through for a volt-

### 3.3 MOSFET Mathematical Model

This section presents the MOSFET mathematical model we have decided to use for our parameter extraction to demonstrate its complexity. Since this dissertation thesis proposal does not deal with technological description, more information about the model can be found in [1, 3, 21, 45].

$$C_{ox} = \frac{\varepsilon_{ox}}{t_{ox}} \quad (3.1)$$

$$W_{eff} = W \cdot scale + (xw - 2wd)scalem \quad (3.2)$$

$$L_{eff} = L \cdot scale + (xl - 2ld)scalem \quad (3.3)$$

$$\phi = 2\phi_F \quad (3.4)$$

$$x_{def} = \sqrt{\frac{2\varepsilon_{Si}}{qN_{eff}N_{sub}}} \quad (3.5)$$

$$\eta = 1 + \frac{\text{delta}\pi\varepsilon_{Si}}{4C_{ox}W_{eff}} \quad (3.6)$$

$$V_{crit} = \frac{\varepsilon_{Si}u_{crit}}{C_{ox}} \quad (3.7)$$

$$V_{BX} = \phi - v_{BS} \quad (3.8)$$

$$W_S = x_{def}\sqrt{V_{BX}} \quad (3.9)$$

$$\alpha_S = \frac{x_j}{2L_{eff}} \left[ \sqrt{1 + \frac{2W_S}{x_j}} - 1 \right] \quad (3.10)$$

$$V_{BIN} = V_{T0} - \gamma_0\sqrt{\phi} + (\eta - 1)V_{BX} \quad (3.11)$$

$$W_D = x_{def}\sqrt{V_{BX} + v_{DS}} \quad (3.12)$$

$$\alpha_D = \frac{x_j}{2L_{eff}} \left[ \sqrt{1 + \frac{2W_D}{x_j}} - 1 \right] \quad (3.13)$$

$$\gamma_{eff} = \gamma_0(1 - \alpha_S - \alpha_D) \quad (3.14)$$

$$V_{TH} = V_{BIN} + \gamma_{eff}\sqrt{V_{BX}} \quad (3.15)$$

$$\frac{\partial\gamma_{eff}}{\partial v_{BS}} = \gamma_0 \frac{x_{def}}{4L_{eff}} \left[ \frac{1}{\sqrt{\left(1 + \frac{2W_S}{x_j}\right) V_{BX}}} + \frac{1}{\sqrt{\left(1 + \frac{2W_D}{x_j}\right) (V_{BX} + v_{DS})}} \right] \quad (3.16)$$

$$C_D = \left[ \frac{\gamma_{eff}}{2\sqrt{V_{BX}}} - \frac{\partial \gamma_{eff}}{\partial v_{BS}} \sqrt{V_{BX}} + (\eta - 1) \right] C_{ox} \quad (3.17)$$

$$V_A = \frac{kT}{q} \left[ 1 + \frac{qN_{fs} + C_D}{C_{ox}} \right] \quad (3.18)$$

$$V_{GX} = V_A \ln \left[ 1 + \exp \left( \frac{v_{GS} - V_{TH}}{V_A} \right) \right] + V_{TH} \quad (3.19)$$

$$Fac = \sqrt{1 + 4 \left( \frac{\eta}{\gamma_{eff}} \right)^2 \left( \frac{V_{GX} - V_{BIN}}{\eta} \right) + V_{BX}} \quad (3.20)$$

$$\mu_{eff} = \frac{\mu_0}{1 + \frac{V_{GX} - V_{TH}}{V_{crit}} + u_{exp} \left( \sqrt{V_{BX}} - \sqrt{\phi} \right)} \quad (3.21)$$

$$V_{Clm} = \left( \frac{V_{max} x d_{eff}}{2\mu_{eff}} \right)^2 \quad (3.22)$$

$$V_{Lim} = \frac{V_{max} L_{eff}}{\mu_{eff}} \quad (3.23)$$

$$V_{Sat} = \frac{V_{GX} - V_{BIN}}{\eta} + \frac{1}{2} \left( \frac{\gamma_{eff}}{\eta} \right)^2 (1 - Fac) \quad (3.24)$$

$$V_{DSAT} = V_{Sat} + V_{Lim} - \sqrt{V_{Sat}^2 + V_{Lim}^2} \quad (3.25)$$

$$V_{DX} = \frac{v_{DS}}{\left[ 1 + \left( \frac{v_{DS}}{V_{DSAT}} \right)^{\frac{1}{UTRA}} \right]^{UTRA}} \quad (3.26)$$

$$Body = \frac{2}{3} \gamma_{eff} \left[ (V_{BX} + V_{DX})^{\frac{3}{2}} - V_{BX}^{\frac{3}{2}} \right] \quad (3.27)$$

$$\Delta L = x d_{eff} \left[ (v_{DS} - V_{DX} + V_{Clm})^{\frac{1}{2}} - V_{Clm}^{\frac{1}{2}} \right] \quad (3.28)$$

$$i_{DS} = \beta \left[ \left( V_{GX} - V_{BIN} - \frac{\eta}{2} V_{DX} \right) V_{DX} - Body \right] \quad (3.29)$$

$$\beta = \mu_{eff} C_{ox} \frac{W_{eff}}{L_{eff} - \Delta L} \quad (3.30)$$

### 3.4 MOSFET Mathematical Model Parameters

The mathematical model presented in section 3.3 has 34 parameters and, as mentioned before, some of these parameters have direct physical meaning such as the permeability of Silicon ( $\varepsilon_{Si}$ ) or channel dimensions ( $wd$ ,  $ld$ ), other parameters such as transverse field coefficient ( $UTRA$ ) combine several effects together and have only indirect physical influence and effect things like knee curvature in the case of ( $UTRA$ ). All model parameters are wrapped in table 3.1 together with their default values:

Parameter	Default Value	Unit	Parameter	Default Value	Unit
$V_{T0}$	$8.48 \cdot 10^{-1}$	$V$	$N_{fs}$	1.764	$cm^{-2}$
$N_{eff}$	$4.298 \cdot 10^1$	[1]	$\mu_0$	$4.146 \cdot 10^{-2}$	$cm^2V^{-1}s^{-1}$
$u_{crit}$	$5.037 \cdot 10^2$	$Vcm^{-1}$	$UTRA$	$3.69 \cdot 10^{-1}$	[1]
$V_{max}$	$5.956 \cdot 10^4$	$ms^{-1}$	$delta$	$1.0 \cdot 10^{-5}$	[1]
$N_{sub}$	$8.135 \cdot 10^6$	$cm^{-3}$	$wd$	$3.843 \cdot 10^{-7}$	$m$
$ld$	$2.243 \cdot 10^{-7}$	$m$	$u_{exp}$	$1.0 \cdot 10^{-5}$	[1]
$xj$	$1.617 \cdot 10^{-8}$	$m$	$scale$	1.0	[1]
$scalem$	1.0	[1]	$Temp$	$2.7 \cdot 10^1$	$K$
$nrd$	$5.0 \cdot 10^{-1}$	[1]	$nrs$	$5.0 \cdot 10^{-1}$	[1]
$W$	$2.6 \cdot 10^{-6}$	$m$	$L$	$8.0 \cdot 10^{-7}$	$m$
$xw$	0.0	$m$	$xl$	$5.0 \cdot 10^{-8}$	$m$
$tox$	$1.552 \cdot 10^{-8}$	$m$	$rsh$	$2.32 \cdot 10^1$	$\Omega$
$rd$	$3.026 \cdot 10^3$	$\Omega$	$rs$	$3.026 \cdot 10^3$	$\Omega$
$ldif$	$2.0 \cdot 10^{-7}$	$m$	$hdif$	$1.3 \cdot 10^{-6}$	$m$
$\varepsilon_{Si}$	$1.0359 \cdot 10^{-10}$	$Fm^{-1}$	$\varepsilon_{ox}$	$3.4531332486 \cdot 10^{-11}$	$Fm^{-1}$
$gelement$	$1.60217733 \cdot 10^{-3}$	$C$	$xk$	$1.380658 \cdot 10^{-7}$	[1]
$Tabs$	$2.7315 \cdot 10^2$	$K$	$xni$	1.45	[1]

Table 3.1: MOSFET Model Parameters

<b>VT0</b>	<b>scalem</b>
VT0 parameter presents the zero-bias threshold voltage <sup>2</sup> .	Geometry dependent scaling parameter.
<b>Nfs</b>	<b>nrd</b>
Fast surface state density.	Number of squares of drain diffusion.
<b>Neff</b>	<b>nrs</b>
Total channel-charge (fixed and mobile) coefficient.	Number of squares of source diffusion.
<b>ucrit</b>	<b>W</b>
Critical field for mobility degradation.	The physical channel width.
<b>UTRA</b>	<b>L</b>
Transverse field coefficient <sup>3</sup> .	The physical channel length.
<b>Vmax</b>	<b>xw</b>
Maximum drift velocity of carriers (electrons for N-MOSFET).	Mask/etch effect on W.
<b>delta</b>	<b>xl</b>
Width effect on threshold voltage.	Mask/etch effect on L.
<b>Nsub</b>	<b>tox</b>
The amount of substrate doping with class VA elements.	The thickness of the oxide layer.
<b>wd</b>	<b>rsh</b>
Lateral bulk diffusion along width.	Drain and source diffusion sheet resistance.
<b>ld</b>	<b>rd</b>
The width of lateral diffusion.	Drain ohmic resistance.
<b>uexp</b>	<b>rs</b>
Critical field exponent in mobility degradation.	Source ohmic resistance.
<b>xj</b>	<b>ldif</b>
Metallurgical junction depth.	Length of lightly doped diffusion adjacent to gate.
<b>scale</b>	<b>hdif</b>
Geometry dependent property scaling parameter.	Length of heavily doped diffusion, from contact to lightly doped region.
	<b>epsilonSi</b>
	Permeability of Si.
	<b>epsilonox</b>
	Permeability of oxide material.
	<b>qelement</b>
	Elementary charge.
	<b>TabS</b>
	Zero °C temperature in K.

---

<sup>2</sup>VT0 parameter is positive for n-type (enhancement) devices and negative for p-type (depletion) devices.

<sup>3</sup>UTRA is a nonphysical parameter which affects the shape of DC characteristic's knee region. The higher UTRA is, the sharper the knee region is.

### 3.5 Parameter Extraction Method

The purpose of MOSFET parameter extraction is to find out values of unknown parameters of a mathematical model such that their error satisfies some defined criterions. The algorithm which is depicted at fig. 3.4 is a result of a linearization process of the minimizing function

$$\Phi = \mathbf{f}(\mathbf{p})^T \mathbf{f}(\mathbf{p}), \text{ where}$$

$$f_i(\mathbf{p}) = \frac{i_{DS}(\mathbf{x}_i, \mathbf{p}) - IDS_i}{\max \{IDS_i, IDS_{min}\}}. \quad (3.31)$$

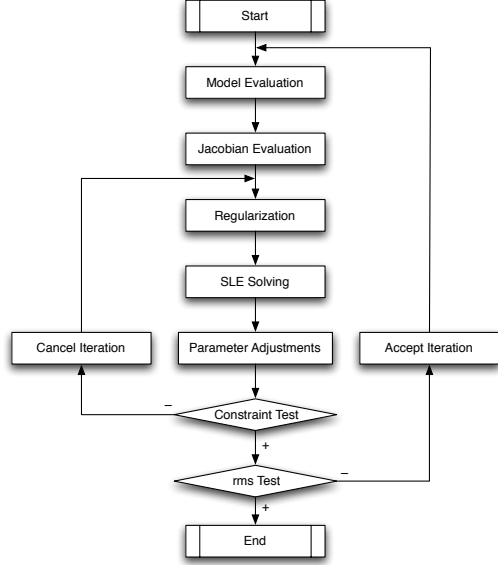


Figure 3.4: Extraction Algorithm

The  $i_{DS}(\mathbf{x}_i, \mathbf{p})$  symbol represents the value of MOSFET mathematical model at data point  $\mathbf{x}_i$  ( $i = 1 \dots N$ , where  $N$  stands for the number of data points measured) with parameters  $\mathbf{p} = (p_1, \dots, p_r)$ ;  $IDS_i$  is a measured value of a drain current at a given data point and  $IDS_{min}$  is the smallest value of  $IDS_i$ . After linearization of the least squares problem, the extraction process for a value of  $\mathbf{p}$  can be expressed with the following recurrent expression:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \Delta \mathbf{p}^{(k)}, \quad (3.32)$$

$$\Delta \mathbf{p}^{(k)} = \left( \mathbf{J}^T \mathbf{J} + \lambda^{(k)} (\mathbf{J}^T \mathbf{J})_{diag} \right)^{-1} \mathbf{J}^T \mathbf{f}(\mathbf{p}^{(k)}) \quad (3.33)$$

where Jacobi's matrix  $\mathbf{J} = (\nabla_{\mathbf{p}} \mathbf{f}(\mathbf{p})^T)^T$ . The task we solve is ill-posed and therefore we can expect ill-conditioned matrices in a set of linear equations (SLE) (3.33). Being  $\mathbf{J}^T \mathbf{J}$  matrix ill-conditioned with condition number  $\kappa(\mathbf{J}^T \mathbf{J}) \gg 1$ , the process requires some form of a regularization before the SLE is solved to effectively lower its condition number. Once the SLE is solved, parameters  $\mathbf{p}$  being extracted get adjusted and are passed to the constraint test which decides whether the iteration is valid. If the iteration is found to be invalid, the entire iteration is rejected, parameters are rolled back to their previous values and the regularization parameter  $\lambda$  is adjusted. Once the iteration is found valid, the global root mean square (*rms*) error is calculated (cf. Fig. 1) and iteration continues as we have not reached the requested accuracy. The *rms* criterion defines how accurately the mathematical model with estimated parameters fits the measured data set.

Since we are interested in a real-time parameter extraction, our intention is to complete each step as fast as possible. We will show later that headless chase for a speed does not always lead to the overall extraction speed improvement. The most important fact, which is in our opinion too often underestimated, is that the problem is ill-conditioned and therefore we can expect that even a very small change can dramatically influence the result. Since such change can even be a rounding error committed during the evaluation in either of steps, we need to be always aware of the error of the result or else we cannot guarantee that the result is meaningful. Regularization of the problem helps as it effectively lowers the condition number of (3.32) and therefore makes evaluation less sensitive to small perturbations that can be consequences of rounding and truncation errors committed during floating point operations.

Several next subsections will treat each step of the parameter extraction depicted at Fig. 3.4 in detail and discusses the improvements that were suggested and how they affected the overall speed and accuracy of the solution.



### 3.6 Optimization Methods

In order to be able to classify parameter extraction methods, a brief information about the methods and heuristics that are used to obtain mathematical model parameters and/or control the progress of the parameter extraction algorithm are necessary. This overview is also necessary because the following methods either already have been used for parameter extraction or could be used in future. This includes the essentials of evolutionary computing, simulated annealing [23], genetic algorithms [32], multi objective optimization [46], tabu search [11], ant-colony optimization [9], and a detailed description of a general curve fitting algorithm known as Levenberg-Marquardt's algorithm [28] which plays significant role in the parameter extraction methodology proposed by this dissertation thesis proposal. Various existing parameter extraction methods are compared in 3.7 on page 24.

#### 3.6.1 Simulated Annealing

Simulated annealing comes from metallurgy when the annealing process is controlled to obtain an optimal quartz granularity during metal annealing and a system with the lowest internal energy. Having system the lowest energy ensures that quartzes of the annealed metal (may be steel, for example) are big and metal's properties fulfill requirements such as their solidity, elasticity, etc.

In a computer world, simulated annealing [23] presents an another global optimization method which can be used in a case when there exists a large search space. Its purpose is to provide a good approximation to a global minimum. The essential task of simulated annealing type algorithms is to minimize the internal energy of the system which is represented by an energy function  $E(state)$  which is a function of the current state. Such energy function can be the *rms* error, for example. The lower the value of the energy function is, the closer is the state to a the state with the lowest energy and thus a global solution. Heuristic is used to find neighbors to the current state. After the neighbors are found, energy differences  $\Delta E = E(state_1) - E(state_2)$  are calculated for each of proposed new states and are passed to a probability function which determines the succeeding state. There is also a probability that no transition occurs and system stays in the current state. The probability of transitioning from one state to the another is a function of energy function difference together with a global parameter called temperature. The temperature parameter  $T(t)$  is a function of system annealing time  $t$  and controls annealing capabilities of the algorithm and once it reaches zero, the stopping criterion is met and algorithm ends. Temperature of the system is often represented with an exponential function  $\exp(-\Delta E/T)$  which decreases rapidly in early stages of the computation and slowly in final ones.

One of the specifics of simulated annealing type algorithms is that they allow transitions to states where the energy function delta would be positive. Positive energy function difference means that a proposed succeeding state is farther from the solution (worse) than the current one. This is often realized by assigning nonzero probabilities of transitioning from the current state to the "worse" state. Such a specific prevents annealing type algorithm from being stuck in a local minimum.

Algorithms using simulated annealing also strongly rely on parameters that control algorithm's progress. Each problem that wants to be applied simulated annealing on requires user to define neighbor selection heuristic, transition probability function and annealing control parameters. Of all of the mentioned prerequisites significantly influence the progress of the algorithm and the choice of parameters and functions is unfortunately unique for each problem which makes simulated annealing difficult to adapt to a given problem. Simulated annealing appears that it has not used in parameter extraction algorithm yet.

### 3.6.2 Evolutionary Computing

Evolutionary algorithms [46] are used to perform a stochastic optimization in potentially a very large search space. Generally, evolutionary algorithms operate upon a set of solution candidates on which they simulate the process of natural evolution. Such a candidate set is populated with the two basic operations known as selection and variation. Selection is represented with the process of natural selection, where only the strongest individuals tend to survive and variation duplicates the process of mutation and recombination of set's elements. Genetic algorithms or multiobjective optimization are all examples of evolutionary algorithms since the solution “evolves” as it is being approached. Evolutionary algorithms are considered to be a good search techniques because they are able to handle opposing criteria and search large and complex spaces.

### 3.6.3 Genetic Algorithms

Genetic algorithms are inspired with Darwin's theory of evolution. Nucleic acids bear an inherited information and their most important ability is their replication. Nucleic acids transfer the information they bear onto the population and this principle presents evolution and are building blocks of chromosomes that contain genetic information.

In the computer terminology, each state (member of a population) is encoded as a string (genotype) which consists of individual chromosomes (variables) and evolution is represented by the two essential operators that are crossover and mutation that map genotypes onto new genotypes. There is also a necessity to judge the vitality of the produced descendant with the means of some fitness function. With the help of these operators, we are able to simulate the evolution and fitness function performs the natural selection so only the strongest individuals survive.

Genetic algorithms can be used to solve all kinds of tasks. They are often used in a case where there is great space to search or a possibility of being stuck in a local minimum since they are good of getting out of it. Moreover, it is not difficult to construct a genetic algorithm, create a fitness function which may be

eg. an *rms* error and define appropriate mutation and crossover operators. While mutation operator randomly changes some portion(s) of a genotype it is passed to it, crossover operator combines both genotypes (eg. randomly swaps bits of two genotypes) and produces two entirely new genotypes. This produces an entirely new population and fitness function is used to consider the fitness of each member. The better a member fitness has, the more probable will it remain in the population.

A typical scenario [32] is depicted at figure 3.5. At the beginning of a genetic algorithm in the “Start” block, a random population of chromosomes is generated and genotypes are created. A set of genotypes present the population and it is passed to the “Fitness” block which evaluates fitness for each member of the population. The entire population is passed to the “New Population” block which starts with a selection. Only members that achieve a specified level of fitness are selected as parents for next generation in the “Selection” sub-block. Once the selection pro-

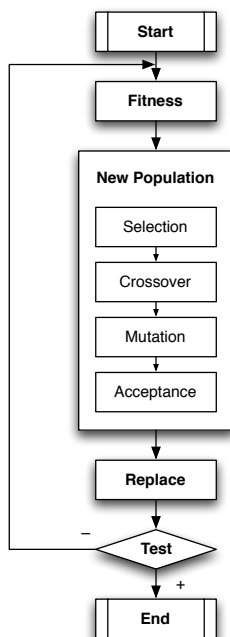


Figure 3.5: A typical genetic algorithm

cess is complete, “Crossover” starts to take its role. Each two members are combined together and some percentage of their chromosomes is swapped and two new members are produced. After crossing members over, they are passed to the “Mutation” block which randomly changes another portions of their chromosomes to the opposite values. New population is created in the “Acceptance” block and the old population is replaced with the new one during the “Replace” process. After all this is done, the “Test” block performs the ending condition and if not met, the algorithm starts over in the “Fitness” step otherwise we have found a satisfying solution of the problem. One of the obvious advantages of genetic algorithms is that they can easily be applied to almost any problem where a possibility of being stuck in a local minimum is high. There are also cons that each of its step depends on a setting of a control parameter. This parameter specifies, for example, how much an individual gets mutated, how many chromosomes are combined together, the level of fitness, etc. and therefore an experienced user is necessary to set these parameters. Despite this disadvantage, genetic algorithms are able to find satisfying solution in usually several hours. The mentioned cons did not mean that genetic algorithms were bad! They can be very useful in a case where we do not know an algorithm which would solve a problem and/or the problem has a large search space. GA have been used in parameter extraction and are compared to our approach in section 3.7 on page 24.

### 3.6.4 Multiobjective Optimization

Multiobjective optimization (also multicriteria optimization or vector optimization) [46] as its name suggests is able to combine several criterions at once making no assumptions about them. Such criterions may contradict one to the another, there is no global unique solution to the problem. The process of multiobjective optimization can help us to find the best compromise solution(s). Problems solution can be defined as a solution vector  $x = (x_1, x_2, \dots, x_n)$ ,  $x \in X$ , where  $X$  is the solution space. Then we can define an objectives vector  $y = (y_1, y_2, \dots, m_n)$  and a mapping function  $f : X \rightarrow Y$ , where  $Y$  is the objective space. A set of optimal solutions is called a Pareto set and  $X^* \subseteq X$  and we look for a good approximation to it. The image to a Pareto set in the objective space is Pareto front and  $Y^* = f(X^*) \subseteq Y$ . Any information about  $Y^*$  is useful and helps to identify the best compromise solution. Few multiobjective optimization methods combine all objectives into one by aggregating them and thus making a single scalar  $y$  that they try minimize but most of the methods is trying to find a good approximation to a Pareto set. Attempts to generate Pareto set may be very resource expensive and often unfeasible and therefore stochastic search methods like evolutionary algorithms, tabu search (3.6.6), simulated annealing (3.6.1) and ant colony optimization (3.6.5) have been developed.

It has not been mentioned yet, how one solution  $\mathbf{y}_1$  is compared to another one  $\mathbf{y}_2$ . Solution  $\mathbf{y}_1$  dominates over solution  $\mathbf{y}_2$  if it fulfills the following relation:

$$\mathbf{y}_1 \succ \mathbf{y}_2 \iff \mathbf{f}(\mathbf{x}_1) \succ \mathbf{f}(\mathbf{x}_2) \iff \forall_{i=1}^n y_{1i} \geq y_{2i}, \text{ where } \mathbf{y}_1 \neq \mathbf{y}_2. \quad (3.34)$$

This also means that the optimal solution is such solution which is not dominated by any other solution but can still be mapped to different objective vectors. There can be multiple solutions that differ in various elements of objectives and represent different trade-offs. A typical progress of a multiobjective optimization algorithm is depicted at fig. 3.6



Figure 3.6: Typical multiobjective optimization algorithm

First, a random population is created and fitness is evaluated for each of its members. Fitness function judges the vitality of a population member. After that, a defined number of members is randomly chosen from the population depending on their fitness. This process is called mating selection. Once the mating selection is done, variation (crossover) happens in the recombination block and new members are created. New members are then mutated, where some portion of them gets randomly changed and finally go through the selection block which determines the individuals that will continue in the process. Algorithm starts again until some stopping criterion (such as value of a fitness function) is met. Multiobjective optimization has been used in parameter extraction and is mostly combined with genetic algorithms. Section 3.7 on page 24 compares such algorithm with our approach.

### 3.6.5 Ant Colony Optimization

Ant colony optimization (ACO) [9, 27] is an optimization method that can be used to solve complex combinatorial problems. The method itself is inspired with the behavior of ants searching for a food. When ants are looking for a food, they wander randomly from their anthill and if they find some food, they return back while leaving a path of pheromones behind them. If another ant finds such path, it likely sticks to that path and soon there are many ants at the food location.

In a computer terminology, ACO is performed as a parallel search. A set of usually threads or processes (agents) that represent a colony of ants walk through the states of the problem. Their stochastic movement is controlled with two parameters called trails and attractiveness. Since agents work in parallel, each of them constructs its own solution to the problem and evaluates the quality of the solution by setting its trail value which acts as a pheromone information for the other ants. Since pheromones typically evaporate after some time, ACO includes two more mechanisms that control trail evaporation and perform global actions. Trail evaporation mechanism is used to avoid unlimited accumulation of trails and decreases the values of all trails from time to time and the daemon actions mechanism offers a way how to perform actions that single ants cannot do such as global information update. Each ant's behavior is influenced with the two mentioned variables — attractiveness and trail level. Attractiveness controls an a priori desirability of a move from the current state to a succeeding state while the trail level presents move history information and represents a posteriori desirability of the proposed move.

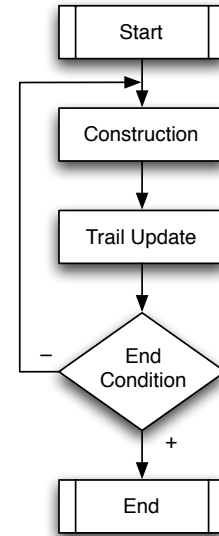


Figure 3.7: A typical ACO algorithm

The typical progress of the ACO type algorithm is depicted at fig. 3.7. After the optimization starts,  $k$  independent ants (threads/processes) are created. They start to construct their path in the “Construct” block by examining their neighborhood and deciding, based on the values of attractiveness and trails, where to continue. After all ants have completed their path construction, the value of their trails in the “Trail” block is adjusted and others are let known about their current situation. Finally, the algorithm performs the ending condition in the “End Condition” block which may be eg. rms error of the solution and if not met, search continues with construction again. Ant colony algorithms seem to be a promising searching way as it is naturally parallel and can be used to search a large and complex solution spaces. Ant colony

optimization has not been observed to be used in parameter extraction yet but this method could be used in the future because of its natural parallelism.

### 3.6.6 Tabu Search

Tabu search (TS) [11, 17] is a metaheuristic<sup>4</sup> algorithm and its purpose is to increase the efficiency of a searching function it is used with by employing appropriate supervision on it. The efficiency increase is accomplished by using an additional memory to keep track of some status information about the search progress, neighborhood being searched and solutions already found. Maintaining an extra state information is the key aspect of TS which makes it so successful. This history information is called tabu list and is used mostly during decisions search algorithm needs to do. Having a tabu list of length  $x$  prevents algorithm of entering a cycle of length  $x$ . If tabu lists were too restrictive and did not allow already visited state be reentered, the algorithm would change into a simple descent which would fall into a local minimum and stay there indefinitely. That is a reason why tabooed states or states that are further from the solution we are searching for are allowed to be entered after some time (or number of moves) so the algorithm can get out of the local minimum. Tabu search has been used in parameter extraction algorithms and our algorithm could use its features to lower the number oscillations and it is briefly outlined in section 3.7 on page 24.

### 3.6.7 Levenberg-Marquardt's algorithm

Levenberg-Marquardt's algorithm [2, 4, 28] (LMA) is one of the most widely used curve fitting algorithms. This algorithm solves weighted least squares (WLS) [2, 19] by interpolating in between Newton's iteration [2] with gradient descent methods [2] and is able to find solution even if started far off the minimum. The basic equation for a single iteration of LMA is:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \left[ \left( \mathbf{J}^{(k)} \right)^T \mathbf{W} \mathbf{J}^{(k)} + \lambda^{(k)} \mathbf{D}^{(k)} \right]^{-1} \left( \mathbf{J}^{(k)} \right)^T \mathbf{W} \mathbf{f}(\mathbf{p})^{(k)}, \quad (3.35)$$

where  $\mathbf{p}^{(k)}$  is some parameter vector in  $k$ -th state,  $\mathbf{J} \in \mathbb{R}^{m \times n}$  is Jacobi's matrix,  $m \geq n$ ,  $m, n \in \mathbb{R}$ , and  $\mathbf{W}$  is a weighting matrix,  $\lambda$  is a scalar damping factor and  $\mathbf{D}$  is a diagonal matrix,  $\mathbf{f}(\mathbf{p})$  presents an error function.

The presence of  $\mathbf{D}$  matrix is absolutely essential for the algorithm. This matrix reduces the size and changes the direction of the gradient descent step and also reduces oscillations and instabilities as diagonal elements of  $\mathbf{J}^T \mathbf{W} \mathbf{J}$  grow. LMA can alternatively be derived by linearizing the nonlinear functional  $F$  within the nonlinear Tikhonov functional  $\|F(x) - y^\delta\|^2 + \alpha \|x - x^*\|^2 \rightarrow \min$  [10]. LMA performs a regularization of the problem and therefore suppresses the influence of noise and condition number of the inverted portion of (3.35).

This dissertation thesis proposal uses LMA mostly in the following form, where  $\mathbf{W} = \mathbf{I}$  and we obtain the following ordinary least squares form:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \left[ \left( \mathbf{J}^{(k)} \right)^T \mathbf{J}^{(k)} + \lambda^{(k)} \mathbf{D}^{(k)} \right]^{-1} \left( \mathbf{J}^{(k)} \right)^T \mathbf{f}(\mathbf{p})^{(k)}. \quad (3.36)$$

The choice of  $\lambda$  damping factor is important for the progress of the algorithm. Having put  $\lambda = 0$ , (3.36) reduces to a simple Newton iteration:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \left[ \left( \mathbf{J}^{(k)} \right)^T \mathbf{J}^{(k)} \right]^{-1} \left( \mathbf{J}^{(k)} \right)^T \mathbf{f}(\mathbf{p})^{(k)}, \quad (3.37)$$

---

<sup>4</sup>Heuristic which controls the progress of some search function.

while for a very large  $\lambda$  and  $\mathbf{D} = \mathbf{I}$ , the second portion of (3.36)  $\lambda^{(k)}$  outweighs the first portion  $\mathbf{J}^{T(k)}\mathbf{J}^{(k)}$  and becomes a gradient (steepest) descent method:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \left(\lambda^{(k)}\right)^{-1} \left(\mathbf{J}^{(k)}\right)^T \mathbf{f}(\mathbf{p})^{(k)}. \quad (3.38)$$

The steepest descent method (3.38) could be very inefficient as it gives only the direction but not the step size. An obvious advantage of the steepest descent is that it does not require an inverse matrix at all.

Marquardt suggests the lambda parameter of (3.35) be set to:

$$\lambda^{(k)} = \frac{\mathbf{f}(\mathbf{p})^{T(k)} \mathbf{W} \mathbf{J}^{(k)} \left(\mathbf{J}^{(k)}\right)^T \mathbf{W} \mathbf{f}(\mathbf{p})^{(k)}}{S^{(k)}}, \quad (3.39)$$

where  $S^{(k)}$  is the least squares function. We have decided to use a different algorithm for setting  $\lambda$  and this algorithm is discussed later in the dissertation thesis proposal. Algorithm is usually stopped if either the parameter vector  $\mathbf{p}^{(k)}$  or the *rms* error change are lower than a defined value.

### 3.7 Other Parameter Extraction Methods

This section compares genetic algorithms that are often combined with multiobjective optimization with extraction algorithms based on Levenberg-Marquardt's method.

#### 3.7.1 Genetic algorithms & Multiobjective optimization

Genetic algorithms (GA)s described briefly in section 3.6.3 on page 20 are appropriate wherever there is a danger that we could get stuck in a local minimum of the problem being optimized. They have been used with parameter extraction [22, 29] and this section compares it to the method we have developed.

Traditional GA needs fitness function — a scalar we use to judge the vitality of given population member and uses 2 operators. These operators are mutation and crossover. Algorithm presented in [22] adds two more operators that are niching and nondominated sorting [38] that allow multiple objectives be simultaneously pursued.

##### 3.7.1.1 Principles of GA Parameter Extraction

GA maps the entire parameter set onto a single binary string of defined length, while each parameter has defined both upper and lower bound. Having available the bound, we can reduce the space we have to search. Once the string (member) is ready, it is passed to the fitness function which evaluates member's vitality. New population is created based on the fitness value which is also used for parents selection so only vital enough parents are chosen. After there is enough new parents, the crossover operator is applied on the set which interchanges random segments of two strings. Consequently, the mutation occurs which flips, at some probability, a random bits of a string to their opposite value. Both amount of mutation and crossover are parameters of this algorithm and if they are poor, the algorithm does not work.

The purpose of niching operator is to keep rather a set of solutions instead of one and guarantees that there are always more than one solution available at a time. The niching operator modifies the value of a computed fitness function in a way that it compares  $n$  bits (niche size) of each string and splits the entire population into  $n$  groups and reduces the value of fitness function so members of each group have a chance to appear in the chosen population. Nondominated sorting

modifies selection and replacement of standard genetic algorithms to enable the discovery of a wide-spread, dense Pareto-optimal front so the solution is better approached.

Once the new population is done, it is converted back to a parameter set and parameters are examined if it fulfills the criterions we have defined. If the fitness function does not go better for several iterations, algorithm ends as it ends in the case when the fitness function reaches pre-defined level.

### 3.7.1.2 Comparing GAs to Our Solution

GA papers often criticize Levenberg-Marquardt's algorithm since it needs a good initial guess which is close to some solution. That is true but we are able to obtain such guess since some of the parameters are at least roughly known before the extraction starts and provide us a good initial guess which is the requirement. The main problem of GA parameter extractions is that they are nondeterministic and strongly depend on the method of fitness function evaluation and constants that specify amounts of mutation, crossover and niche size. Setting either of these constants to a not suitable value breaks the entire algorithm. GA usually takes a very long time to perform and we have absolutely no idea what may be the error of the result we obtain. For this reason we rather abandon GAs since error bound is a very necessary information for an ill-posed problem as is parameter extraction.

### 3.7.2 Other LMA Algorithms

There exist other algorithms that employ Levenberg-Marquardt's algorithm (LMA) [1, 41] and their progress is similar to our algorithm with few exceptions. Our algorithm performs error analysis during the evaluation of its mathematical model so we are able to tell the error of the mathematical model. We also plan to extend the error analysis on the entire algorithm so we are likely to know when some unstable evaluation happens and plan to make evaluation run in parallel by means of the residual number system arithmetic which is discussed on the last chapter.

### 3.7.3 Miscellaneous Extraction Algorithms

There are no signs of ant colony optimization (ACO), simulated annealing (SA) or tabu search (TS) optimization methods in parameter extraction algorithms. ACO is a very new method and may provide fast, parallel parameter extraction if used with proper algorithm control and error analysis. On the other hand, SA is too dependent on the constants so there will probably never be a parameter extraction algorithm that employed SA technique. While TS as a meta heuristic, it could be used in either of the mentioned algorithms since tabu lists could be used to make a history and to possibly reduce oscillations.

## 4 Main Results

### 4.1 Jacobi's Matrix & Numerical Derivative Evaluation

This section describes the ways we have tried and results we have obtained during one of the most important steps of the extraction algorithm, where Jacobi's matrix is evaluated. Jacobi's matrix  $\mathbf{J}$  is an outer product of function  $\mathbf{f}(\mathbf{p})$  of parameter vector  $\mathbf{p}$  of dimension  $n$  with its first order derivative operator  $\nabla_{\mathbf{p}}$  with respect to parameter  $\mathbf{p}$  and is defined as follows:

$$\mathbf{J} = [\nabla_{\mathbf{p}}\mathbf{f}(\mathbf{p})^T]^T, \text{ where} \quad (4.1)$$

$$\nabla_{\mathbf{p}} = \begin{bmatrix} \frac{\partial}{\partial p_1} \\ \vdots \\ \frac{\partial}{\partial p_n} \end{bmatrix}. \quad (4.2)$$

There are essentially two ways how to evaluate  $\mathbf{J}$ . The first one is based on derivative definition (4.3), where each element of  $\mathbf{J}$  gets approximated with one or more elements of Taylor series expansion of (4.3) in a neighborhood of 0 and the second approach uses symbolic computation.

#### 4.1.1 Numeric Derivative Approximation

The purpose of numeric derivative is to approximate as close as possible the value of derivative  $f'(x)$  of function  $f(x)$  based on several values of  $f(x)$  for some step size  $h$ . The approximated  $f'(x)$  will be written as  $g(x)$ . The opt for a numeric derivative is because of the complexity of the mathematical model we have and the evaluation of analytical derivative with symbolic computing takes significantly longer time.

If we look at the derivative definition

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}, \quad (4.3)$$

we see that only two evaluations of function  $f(x)$  are necessary for the simplest numerical derivative evaluation and it is undoubtedly a very fast way. That would give us the initial approximation:

$$g(x, h) = \frac{f(x+h) - f(x)}{h} \quad (4.4)$$

Unfortunately if the we have approximations of function  $f(x)$  and  $f(x+h)$  with errors  $\varepsilon$  and  $\varepsilon_h$  it does not say anything about  $f'(x)$  and its error  $\varepsilon'$  can be *arbitrarily* big [30] depending on function  $f(x)$ . The error of the approximation can be, assuming that  $f(x)$  is defined in neighborhood of 0, deduced from Taylor series expansion of  $f(x)$  by the step size  $h$  in the neighborhood of 0 as follows:

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + \frac{1}{6}h^3f'''(x) + \dots + \frac{1}{n!}h^n f^{(n)}(x) \quad (4.5)$$

Applying (4.5) to approximation (4.4) gives:

$$g(x, h) = f'(x) + \frac{1}{2}hf''(x) + \frac{1}{6}h^2f'''(x) + \dots + \frac{1}{n!}h^{n-1}f^{(n)}(x) \quad (4.6)$$

The other possibility of a numerical derivative evaluation is to take the reverse direction of step  $h$  which yields:

$$g(x, h) = \frac{f(x) - f(x-h)}{h} \quad (4.7)$$



Finding out Taylor series expansion for (4.7) gives:

$$f(x+h) = f(x) + hf'(x) - \frac{1}{2}f''(x)h + \frac{1}{6}f^{(3)}(x)h^2 + O(h^3) \cdots + \frac{1}{n!}h^n f^{(n)}(x) \quad (4.8)$$

Applying (4.8) to approximation (4.7) we obtain:

$$g(x, h) = f'(x) - \frac{1}{2}hf''(x) + \frac{1}{6}h^2f'''(x) + \cdots + \frac{1}{n!}h^{n-1}f^{(n)}(x) \quad (4.9)$$

We see that both approximations differ in the minus sign and therefore if we combine them together, we obtain an even better approximation:

$$g(x, h) = f'(x) + \frac{1}{6}h^2f^{(3)}(x) + \frac{1}{120}h^4f^{(5)}(x) + \cdots + \frac{1}{(2n+1)!}h^{2n}f^{(2n+1)}(x) \quad (4.10)$$

We see that the first element of (4.4), (4.7), and (4.10) is the derivative and the remaining elements form an error of the method. There are not only errors of the method but also errors we perform as evaluating on the computer and the severe cancellation in derivative definition magnified with the step size is absolutely non-negligible error.

#### 4.1.1.1 Recommended Step Size

The accuracy of numerical derivative approximation depends strongly on a proper choice of the step size  $h$  and errors we commit during its evaluation. A discussion about the step size is beyond the limits of this dissertation thesis proposal and can be found eg. in [30, 34].

#### 4.1.1.2 Derivative Free Jacobi's Matrix

There were attempts [5] to remove the critical cancellation in a form of division by step size  $h$  from (3.32) by defining uniform step size for the entire Jacobi's matrix  $\mathbf{J}$  and collecting it out as follows:

$$\mathbf{J} \approx h^{-1}\Delta\mathbf{F}(\mathbf{p}, h) = h^{-1} \begin{bmatrix} f_1(p_1+h) - f_1(p_1) & \cdots & f_1(p_n+h) - f_1(p_n) \\ \vdots & \ddots & \vdots \\ f_m(p_1+h) - f_m(p_1) & \cdots & f_m(p_n+h) - f_m(p_n) \end{bmatrix} \quad (4.11)$$

By combining (3.32) and (4.11) we obtain:

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} - \frac{h \cdot \Delta\mathbf{F}(\mathbf{p}^{(k)}, h)^T \mathbf{f}(\mathbf{p}^{(k)}, h)}{\Delta\mathbf{F}(\mathbf{p}^{(k)}, h)^T \Delta\mathbf{F}(\mathbf{p}^{(k)}, h) + \lambda^{(k)} (\Delta\mathbf{F}(\mathbf{p}^{(k)}, h)^T \Delta\mathbf{F}(\mathbf{p}^{(k)}, h))_{diag}} \quad (4.12)$$

As we see (4.12) has completely removed the division by step size and has a multiplication by step size instead. This approach works with a restriction of an uniform step size for all differences which may be troublesome because the same step size for all parameters does not need to be desired. We cannot deal with this problem by using step size matrix instead of a scalar as such matrix would be singular. The best approach would be to leave out the numerical derivative in favor of analytical one which is described in the following section.

### 4.1.2 Analytical Derivative Evaluation

As stated before, the process of evaluating analytical derivative requires symbolic calculations. Mathematica software [44] can easily perform such symbolic calculations but is unfortunately too slow to be employed in the real-time parameter extraction where usually thousands of evaluations are necessary per single iteration. We have the following options how to evaluate derivative of the expression in the analytical way:

- **Mathlink** — an API provided by Wolfram Research, Inc., <http://www.wolfram.com>, which provides Mathematica’s kernel access and symbolic calculations via RPC<sup>1</sup> to other applications. This approach is possible but fully complies with the above statement about the speed.
- **Custom** — a completely custom approach which requires some form of a symbolic expression parser and analytical deriver. This approach is fully discussed within the next paragraphs.

We have decided to use the custom alternative for analytical derivative evaluation as it allows us to have access to all portions of the source code and perform system related optimization of numerical code where necessary. The process of obtaining a value of analytical derivative requires the following 4 steps:

1. **Parser** — a portion of analytical derivative which reads C source code of the mathematical model into an internal form of a tree which can be evaluated with evaluator and/or deriver.
2. **Evaluator** — the internal form created by **parser** can be evaluated at will with the **evaluator** which walks the internal tree form and instructs each of its nodes to evaluate its value. **Evaluator** also supports symbolic manipulation and common subexpression elimination.
3. **Deriver** — the purpose of **deriver** is to transform the entire internal form of the C expression obtained from **parser** into an another internal form which presents derived expression’s symbolic derivative with using common derivative rules such as chain rule, derivative of a product rule, etc.
4. **Code generator** — since it takes a lot of memory accesses and a lot of cache misses to walk the entire internal form in order to obtain result, code generator takes the passed-in internal form and generates C code in IEEE 754 double precision arithmetic, mdouble arithmetic which is discussed later in this dissertation thesis proposal or GNU MP/MPFR arithmetic.

#### 4.1.2.1 Parser

**Parser** which transforms the C source code of the mathematical model into an internal form which we work with consists of two portions that are lexical analyzer and context free LALR(1) (single token look ahead bottom-up) parser. The purpose of lexical analyzer (sometimes also called token scanner) is to break the incoming text into tokens and feed with them the above layer parser which performs a syntax analysis of the text. Lexical analyzer was created with a help of Fast Lexical analyzer [33] UNIX command line tool. The flex file which contains the definition of the lexical analyzer used in **parser** with intercalated comments follows:

---

<sup>1</sup>Remote Procedure Call

The first section of the flex file will get directly copied into the output and consists of operator definitions which are in `operators.h` file, generated bison token table in `parser.tab.h` and UNIX `fcntl.h` header file.

```
%{
    #include "operators.h"
    #include "parser.tab.h"
    #include <fcntl.h>
}%
```

The second section defines regular expressions of basic token types such as common operators, identifiers which consist of alphanumeric characters and underscores, newline characters, separators and digits.

```
PLUS      "+"
MINUS     "-"
MUL       "*"
DIV       "/"
LPAR      "("
RPAR      ")"
POWER     "^"
DIGIT     "[0-9]"
IDENTIFIER [_a-zA-Z][_a-zA-Z0-9]*
ASSIGN    "="
NEWLINE   "\n"
SEMICOLON ";"
COMMA     ","
LT        "<"
LE        "<="
GT        ">"
GE        ">="
EQ        "=="
NE        "!="
COLON     ":"
QUESTION  "?"
AND       "&&"
OR        "||"
```

Before the third section of the flex file starts, scanner mode types need to be defined. A mode can be thought of as a sub-scanner and the basic scanner starts in the implicit mode called `INITIAL`. Since C source code can have comments, it is useful to define a comment processing mode which drops comments out. We define such mode and it is called `COMMENT` and switch to this mode with a `BEGIN(MODE_NAME)` command.

```
%x COMMENT
%%

[ \t]+                /* remove space */

"/*"                  BEGIN(COMMENT);
<COMMENT>[~*\n]*      /* eat anything that's not a '*' */
<COMMENT>"*" + [~*\n]* /* eat up '*'s not followed by '/'s */
```

```
<COMMENT>\n
<COMMENT> "*" + "/"          BEGIN(INITIAL);
```

The purpose of the following lines is to just pass found operators directly to the upper level, where bison parser resides.

```
{PLUS}          return TOK_PLUS;
{MINUS}         return TOK_MINUS;
{MUL}           return TOK_MUL;
{DIV}           return TOK_DIV;
{LPAR}          return TOK_LPAR;
{RPAR}          return TOK_RPAR;
{POWER}         return TOK_POWER;
{NEWLINE}       return TOK_CR;
{SEMICOLON}     return TOK_SEMICOLON;
{COMMA}         return TOK_COMMA;
{LE}            return TOK_LE;
{LT}            return TOK_LT;
{GE}            return TOK_GE;
{GT}            return TOK_GT;
{EQ}            return TOK_EQ;
{NE}            return TOK_NE;
{QUESTION}      return TOK_QUESTION;
{COLON}         return TOK_COLON;
{AND}           return TOK_AND;
{OR}            return TOK_OR;
```

One of the token type is a number. Since numbers can be written in several ways, we have to cover all of them. Numbers essentially consists of DIGIT tokens defined above and can be integral or floating point. Floating point numbers can have fractional part or can be written in a form with an exponent.

```
{DIGIT}+        yylval.d=atof(yytext); return TOK_NUMBER;
{DIGIT}+"." + {DIGIT}*
{DIGIT}*"." + {DIGIT}+
{DIGIT}*"." + {DIGIT}* [eE] [-+] {DIGIT}+
{DIGIT}+ [eE] [-+] {DIGIT}+  yylval.d=atof(yytext); return TOK_NUMBER;
```

Common functions also need to be defined. These include goniometric, cyclometric, power, logarithm and exponential functions. Most of the functions are unary and power is a binary function (takes two arguments). The result of either of the function is its name contained within the `yylval.s` union.

```
sin|cos|tan      strcpy(yylval.s,yytext); return TOK_UNFUNCTION;
asin|acos|atan   strcpy(yylval.s,yytext); return TOK_UNFUNCTION;
sqrt|log|exp|log10|exp10
pow              strcpy(yylval.s,yytext); return TOK_BIFUNCTION;
```

Branching keywords, function return keyword and identifiers need also to be defined.

```
return           strcpy(yylval.s,yytext); return TOK_RETURN;
if               strcpy(yylval.s,yytext); return TOK_IF;
else              strcpy(yylval.s,yytext); return TOK_ELSE;
```

```

{IDENTIFIER}                strcpy(yylval.s,yytext); return TOK_IDENTIFIER;

{ASSIGN}                     return TOK_ASSIGN;

```

Whatever else is found generates an error

```

.                             perror("unknown token %s\n",yytext);

```

The last portion of the file starts after %% identifier which specifies that the rest of the parser definition file gets directly forwarded into the destination file. It is mandatory to define `yyerror` function which is called in the case of an error.

```

%%
void yyerror(const char* a)
{
    perror(a);
    exit(1);
}

extern BaseObj*      gYYParse;
extern int           yydebug;

extern double        Eval(BaseObj* b);
extern BaseObj*      Differentiate(BaseObj* a, const char* by);
extern unsigned long readline( const char**  ioBuffer,
                               const char*   ioBufferEnd,
                               char*         vptr,
                               unsigned long maxlen);

        int          result = 0;

int main(int argc, const char** argv)
{
    char          buffer[4096];
    double        cr;
    FILE*         fd = fopen("model.eq","rb");
    char*         input = NULL, *inputBegin;
    ssize_t       bufferLen;

    inputBegin = input = (char*)calloc( 65536, 1 );
    if( !input )
        exit(1);

    bufferLen = fread( input, 1, 65536, fd );

    fclose( fd );

    /*yydebug = 1;*/

    while(readline((const char**)&input,inputBegin+bufferLen,buffer,4096))

```

```

    {
        yy_scan_string(buffer);
        result = yyparse();
    }

    /* PROGRAM CONTINUES HERE */

    return cr;
}

```

Once we have the lexical analyzer, we are able to produce tokens that can be passed to bison [7] parser generator which generates LALR(1) parser from a grammar file. We use this parser to generate internal form from the expression being parsed. The internal form consists of a symbol table and trees that represent symbol meanings. The first portion of the grammar file gets directly copied into the destination parser and defines operator construction rules.

```

%{
    #include "operators.h"

    BaseObj*    gYYParse;
}%

```

Bison return data type needs to be an union if one needs to return various kinds of data. In our case it is a tree form pointer (`BaseObj*`), number in a double precision arithmetic, or symbol name.

```

%union {
    BaseObj*    obj;
    double      d;
    char        s[128];
}

```

All kinds of possible tokens need to be defined next.

```

%token TOK_COMMA
%token TOK_NUMBER
%token TOK_PLUS
%token TOK_MINUS
%token TOK_MUL
%token TOK_DIV
%token TOK_LPAR
%token TOK_RPAR
%token TOK_POWER
%token TOK_IDENTIFIER
%token TOK_UNFUNCTION
%token TOK_BIFUNCTION
%token TOK_ASSIGN
%token TOK_SEMICOLON
%token TOK_RETURN
%token TOK_CR
%token TOK_LE
%token TOK_LT

```

```

%token TOK_GE
%token TOK_GT
%token TOK_EQ
%token TOK_NE
%token TOK_IF
%token TOK_ELSE
%token TOK_QUESTION
%token TOK_COLON
%token TOK_AND
%token TOK_OR

```

Once the definitions are done, the grammar rules are defined as follows:

```

%%

s:
    | TOK_IDENTIFIER TOK_ASSIGN e tern
    | TOK_RETURN e TOK_SEMICOLON
      { gYYParse = $<obj>2; }
;

```

In order to define conditioned assignments etc., ternary operator (? in C) is required and is defined as follows:

```

tern: TOK_SEMICOLON
      { $<obj>$ = Construct_Symbol($<s>-2,$<obj>0); }
    | TOK_QUESTION e TOK_COLON e TOK_SEMICOLON
      { $<obj>$ = Construct_TernaryOperator($<s>-2,$<obj>0,$<obj>2,$<obj>4); }
;

```

The lowest priority operators && and ||.

```

e:      cmp
      { $<obj>$ = $<obj>1; }
    | e TOK_AND cmp
      { $<obj>$ = Construct_And( $<obj>1, $<obj>3 ); }
    | e TOK_OR cmp
      { $<obj>$ = Construct_Or( $<obj>1, $<obj>3 ); }
;

```

Logical and and or operators are followed with <, <=, >, >=, == and != operators.

```

cmp:    1
    | cmp TOK_LT 1
      { $<obj>$ = Construct_LT( $<obj>1, $<obj>3 ); }
    | cmp TOK_LE 1
      { $<obj>$ = Construct_LE( $<obj>1, $<obj>3 ); }
    | cmp TOK_GT 1
      { $<obj>$ = Construct_GT( $<obj>1, $<obj>3 ); }
    | cmp TOK_GE 1
      { $<obj>$ = Construct_GE( $<obj>1, $<obj>3 ); }
    | cmp TOK_EQ 1
      { $<obj>$ = Construct_EQ( $<obj>1, $<obj>3 ); }

```

```

|   cmp TOK_NE 1
    { $<obj>$ = Construct_NE( $<obj>1, $<obj>3 ); }
;

```

The next level of priority is used by addition and subtraction operators,

```

1:   t
    |   1 TOK_PLUS t
        { $<obj>$ = Construct_Add($<obj>1,$<obj>3); }
    |   1 TOK_MINUS t
        { $<obj>$ = Construct_Sub($<obj>1,$<obj>3); }
;

```

followed by multiplication and division operators.

```

t:   f
    |   t TOK_MUL f
        { $<obj>$ = Construct_Multiply($<obj>1,$<obj>3); }
    |   t TOK_DIV f
        { $<obj>$ = Construct_Divide($<obj>1,$<obj>3); }
;

```

Unary minus and plus operators have very high priority and need to be defined after the binary ones.

```

f:   TOK_MINUS h
    |   { $<obj>$ = Construct_UnMinus($<obj>2); }
    |   TOK_PLUS h
        { $<obj>$ = Construct_UnPlus($<obj>2); }
    |   h
        { $<obj>$ = $<obj>1; }
;

```

This is the highest priority, where number, parentheses, identifiers and functions reside. They also conclude the grammar file.

```

h:   TOK_NUMBER
    |   { $<obj>$ = Construct_Constant($<d>1); }
    |   TOK_LPAR e TOK_RPAR
        { $<obj>$ = Construct_UnPlus($<obj>2); }
    |   TOK_UNFUNCTION TOK_LPAR e TOK_RPAR
        { $<obj>$ = Construct_UnFunction($<obj>3,$<s>1); }
    |   TOK_BIFUNCTION TOK_LPAR e TOK_COMMA e TOK_RPAR
        { $<obj>$ = Construct_BiFunction($<obj>3,$<obj>5,$<s>1); }
    |   TOK_IDENTIFIER
        { $<obj>$ = Construct_SymbolRef($<s>1); }
;
%%

```

#### 4.1.2.2 Evaluator & Deriver

The last rule of the file being parsed should always be **return expression** rule which defines default symbol which is evaluated when performing **Evaluate** method on **BaseObj** object instance. This method instructs each object in the internal form constructed with the parser to



evaluate its value and cache results that do not rely on any symbol. Further evaluation of the same symbol is faster as the common subexpressions get eliminated.

Another feature of the `BaseObj` object is its `Differentiate` method which causes object to duplicate itself as its own derivative. The differentiation process starts by walking the entire internal form and making symbol dependency lists. With that we are able to tell whether symbol  $a$  depends on symbol  $b$  or  $c$ , etc. and can perform chain rule if necessary or set the value of given branch straight away to 0. Once the dependency search is done, chosen symbol is found in the symbol table and derivative is applied on it. Each node by node by leaf by leaf are walked and transformed to their derivatives with the use of common derivative rules and chain rule if given symbol is a function of the symbol with respect to the symbol which we perform the derivative by. In order to lower algorithm's space complexity, all symbols are reference counted and further references to the same symbol do not consume more memory but bump just the reference count. Once the derivative is finished, a new symbol is created which contains the internal form of the derivative wrt. a symbol we derived by.

#### 4.1.2.3 Code Generator

The value of the any symbol in the symbol table can be obtained with `evaluator`. Since the internal form can depend on many sub-symbols which each of them presents a node, a potentially large number of nodes can be hidden beneath a symbol name. There are  $\approx 800000$  nodes for the most complex derivative of MOSFET mathematical model. Because of the tree structure of the symbol, many memory references are necessary to traverse the entire tree before its value is obtained. Needless to say, this needs to be done just once and could certainly be done faster and in a more efficient way. One of the ways of optimizing internal form evaluation is to spit the entire internal form in optimized form back into a C readable source code. This is the reason why the **code generator** was implemented and can generate C source code in IEEE 754 arithmetic, GNU MP/MPFR arithmetic if we require higher precision, or `mdouble` arithmetic which is essentially an IEEE 754 arithmetic enriched with a running error analysis defined later in this dissertation thesis proposal.

#### 4.1.2.4 Results

Besides the working derivative code, we have obtained an interesting compiler benchmark that can test the quality of the compiler and its optimization unit. The code generated with the **code generator** especially with the running error analysis built-in which employs the `mdouble` arithmetic was very tough to compile for certain compilers. This difficulty is because the most complex derivative has around 800000 nodes and each node has to be created and often copied from one place to the another. The copying process involves a copy constructor being called and that causes stack pointer to move to allocate a space for a temporary object and a function call being performed. Optimizer unit has to create a table of such expressions in hope that it can find objects that are the same to have them omitted from the evaluation. Such needs lead to excessive memory requirements and often failed due to a lack of system resources like computing time, maximum data segment size or insufficient RAM. See tab. 4.1 for observations obtained while trying to compile cca 800K C++ file generated with **code generator** in `mdouble` arithmetic with gcc on Mac OS X, HP-UX, and Linux and Microsoft Visual C++ 2003.Net run on Microsoft Windows XP Professional Edition. For more information about `mdouble` arithmetic refer to section 4.2 on page 4.2.

Compiler	Platform	Processor	Freq.	Compile Time	Opt. level
gcc	HP-UX	IA-64	1 GHz	N/A	-O0
cc1: out of memory allocating 2853206 bytes after a total of 253867048 bytes					
gcc	Mac OS X	PowerPC 7400	500 MHz	3:12:00	-O0
Success					
gcc	Mac OS X	PowerPC 7400	500 MHz	N/A	-O3
cc1plus(1322) malloc: *** vm_allocate(size=3888427008) failed (error code=3)					
cc1plus(1322) malloc: *** error: can't allocate region					
cc1plus(1322) malloc: *** set a breakpoint in szone_error to debug					
MSVC C++	Windows XP Pro	2x AMD Opteron 246	2 GHz	5:00	/O0
Success					
MSVC C++	Windows XP Pro	2x AMD Opteron 246	2 GHz	6:00	/O2
Success					

Table 4.1: Compile time observations

## 4.2 Mdouble Arithmetic

Mdouble arithmetic is essentially an IEEE 754 arithmetic with a-posteriori error estimates. As each arithmetic operation is performed, an error of that operation is added to an accumulating error bound giving out an a-posteriori error bound of the result. The biggest advantage of mdouble arithmetic is that it is not necessary to rewrite the code which already uses double precision arithmetic and the only thing which is necessary is to replace each occurrence of `double` type with `mdouble` struct name.

### 4.2.1 Mdouble Structure

This section comments `mdouble.h` header file used with `mdouble` arithmetic. This file has to be included from all files that need this arithmetic and this is one of the two changes that are necessary when transitioning from `double` to `mdouble`. The other change is to replace all `double` keyword occurrences with `mdouble`. The `mdouble.cpp` file related to this header file will not be commented here.

The header file starts with a lock which prevents C++ preprocessor from including it several times into the file being processed.

```
#ifndef __MDOUBLE_H__
#define __MDOUBLE_H__
```

The definition of `mdouble` structure follows and we can see that it contains two double precision numbers `v` and `e` that hold value and error.

```
struct mdouble {
    double v;
    double e;
```

To successfully replace the most common operations, several constructors need to be defined together with assignment operators that have to be overloaded.

```
mdouble(int a);
mdouble(double a);
mdouble(double a, double e_);
mdouble();
```

```
mdouble operator=(mdouble a);
mdouble operator=(double a);
```

In order to be able to write conditional expressions and compare `mdouble` data types as normal `double` data types, almost all operators need to be overloaded too including unary minus.

```
bool operator==(mdouble a);
bool operator!=(mdouble a);
bool operator!();
bool operator>=(mdouble a);
bool operator>(mdouble a);
bool operator<=(mdouble a);
bool operator<(mdouble a);

mdouble operator-();

mdouble operator+=(mdouble a);
mdouble operator-=(mdouble a);
mdouble operator*=(mdouble a);
mdouble operator/=(mdouble a);
};
```

The four basic operators are also overloaded in the global scope.

```
mdouble operator+(mdouble a,mdouble b);
mdouble operator-(mdouble a,mdouble b);
mdouble operator*(mdouble a,mdouble c);
mdouble operator/(mdouble a,mdouble d);
```

Basic functions need to be overloaded too to work with `mdouble` structure. We have not derived an error bound for `fabs` which stands for absolute value since this function just propagates the input error on its output.

```
mdouble pow(mdouble a, mdouble b);
mdouble sqrt(mdouble a);
mdouble log(mdouble a);
mdouble exp(mdouble a);
mdouble fabs(mdouble a);

#endif
```

#### 4.2.2 A-posteriori Error Estimates

The essentials of a-posteriori error estimates [18, 42, 43] are discussed in section 2.2.2.3 on page 7 and this section presents the bounds for basic operations i.e. addition, subtraction, multiplication and division whereas the following section does the same for essential functions as are square root, natural logarithm, exponential function and power function.

#### 4.2.3 A-posteriori Error Estimates of Basic Operators

Exact variables will be denoted with  $a, b, s$ , their errors as  $e, f, g$ . Computed variables are denoted with a hat. Also we assume that  $|\varepsilon| \leq u$ . The following helper equations are used thorough the succeeding analyses:

$$\hat{a} = a + e \quad (4.13)$$

$$\hat{b} = b + f \quad (4.14)$$

$$\hat{s} = s + g \quad (4.15)$$

$$\frac{a + e}{b + f} = \frac{a}{b} - \frac{af - be}{b(b + f)} \quad (4.16)$$

#### 4.2.4 A-posteriori Error Estimates of Addition & Subtraction

$$s = a \pm b \quad (4.17)$$

$$s + g = (\hat{a} \pm \hat{b}) / (1 + \varepsilon) \quad (4.18)$$

$$s + g + \varepsilon \hat{s} = a + e \pm b \pm f \quad (4.19)$$

$$g + \varepsilon \hat{s} = e \pm f \quad (4.20)$$

$$|g| \leq u|\hat{a} \pm \hat{b}| + e + f. \quad (4.21)$$

#### 4.2.5 A-posteriori Error Estimate of Multiplication

$$s = a \cdot b \quad (4.22)$$

$$s + g = \hat{a}\hat{b} / (1 + \varepsilon) \quad (4.23)$$

$$s + g + \varepsilon \hat{s} = (a + e)(b + f) \quad (4.24)$$

$$s + g + \varepsilon \hat{s} = ab + af + eb + ef \quad (4.25)$$

$$g + \varepsilon \hat{s} = af + eb + ef \quad (4.26)$$

Assuming that  $e \ll a$  &  $f \ll b$ , the term  $ef$  is neglected, the error of multiplication is

$$|g| \leq u|\hat{a}\hat{b}| + e|\hat{b} - f| + f|\hat{a} - e|. \quad (4.27)$$

#### 4.2.6 A-posteriori Error Estimate of Division

$$s = \frac{a}{b}, \quad b \neq 0 \quad (4.28)$$

$$s + g = \frac{\hat{a}}{\hat{b}(1 + \varepsilon)}, \quad \hat{b} \neq 0 \quad (4.29)$$

$$s + g + \varepsilon \hat{s} = \frac{a + e}{b + f} \quad (4.30)$$

$$s + g + \varepsilon \hat{s} = \frac{a}{b} - \frac{ae - bf}{b(b + f)} \quad (4.31)$$

$$g + \varepsilon \hat{s} = -\frac{ae - bf}{b(b + f)} \quad (4.32)$$

$$|g| = \left| -\varepsilon \hat{s} - \frac{ae - bf}{b\hat{b}} \right| \quad (4.33)$$

$$|g| \leq u \left| \frac{\hat{a}}{\hat{b}} \right| + \left| \frac{(\hat{a} - e)f - (\hat{b} - f)e}{\hat{b}(\hat{b} - f)} \right|. \quad (4.34)$$

### 4.3 A-posteriori Error Estimates of Basic Functions

#### 4.3.1 A-posteriori Error Estimate of Square Root

Error analysis of `__ieee754_sqrt` function [39] proved that the error of square root operation is always smaller than  $u$ . We can therefore write:

$$s = \sqrt{a}, \quad a \geq 0 \quad (4.35)$$

$$s + g = \sqrt{\hat{a}}/(1 + \varepsilon), \quad \hat{a} \geq 0 \quad (4.36)$$

$$s + g + \varepsilon \hat{s} = \sqrt{a + e} \quad (4.37)$$

$$s + g + \varepsilon \hat{s} = \sqrt{a + e} - \sqrt{a} + \sqrt{a} \quad (4.38)$$

$$g + \varepsilon \hat{s} = (\sqrt{a + e} - \sqrt{a}) \frac{\sqrt{a + e} + \sqrt{a}}{\sqrt{a + e} + \sqrt{a}} \quad (4.39)$$

$$g + \varepsilon \hat{s} = \frac{e}{\sqrt{\hat{a}} + \sqrt{\hat{a} - e}} \quad (4.40)$$

$$|g| = \left| -\varepsilon \hat{s} + \frac{e}{\sqrt{\hat{a}} + \sqrt{\hat{a} - e}} \right| \quad (4.41)$$

$$|g| \leq u \left| \sqrt{\hat{a}} \right| + \left| \frac{e}{\sqrt{\hat{a}} + \sqrt{\hat{a} - e}} \right|. \quad (4.42)$$

#### 4.3.2 A-posteriori Error Estimate of Logarithm

Error analysis of `__ieee754_log` function [39] proved that the error of logarithm function is always smaller than  $u$ . We can therefore write:

$$s = \log a, \quad a > 0 \quad (4.43)$$

$$s + g = (\log \hat{a})/(1 + \varepsilon), \quad \hat{a} > 0 \quad (4.44)$$

$$s + g + \varepsilon \hat{s} = \log(a + e) \quad (4.45)$$

$$s + g + \varepsilon \hat{s} = \log(a + e) - \log a + \log a \quad (4.46)$$

$$g + \varepsilon \hat{s} = \log(a + e) - \log a \quad (4.47)$$

Since

$$f(a + e) - f(a) = ef'(a) + \frac{e^2}{2}f''(a) + \frac{e^3}{6}f'''(a) + \cdots + \frac{e^n}{n!}f^{(n)}(a) \quad (4.48)$$

Then, after taking the first order term neglecting the others since  $f'(a) \gg e$  we obtain:

$$g + \varepsilon \hat{s} \approx \frac{e}{a} \quad (4.49)$$

$$|g| \approx \left| -\varepsilon \hat{s} + \frac{e}{\hat{a} - e} \right| \quad (4.50)$$

$$|g| \lesssim u |\log \hat{a}| + \left| \frac{e}{\hat{a} - e} \right|. \quad (4.51)$$

### 4.3.3 A-posteriori Error Estimate of Exponential Function

Error analysis of `__ieee754_exp` function [39] proved that the error of exponential function is always smaller than  $u$ . We can therefore write:

$$s = \exp a \quad (4.52)$$

$$s + g = (\exp \hat{a}) / (1 + \varepsilon) \quad (4.53)$$

$$s + g + \varepsilon \hat{s} = \exp(a + e) \quad (4.54)$$

$$s + g + \varepsilon \hat{s} = \exp(a + e) - \exp a + \exp a \quad (4.55)$$

$$g + \varepsilon \hat{s} = \exp(a + e) - \exp a \quad (4.56)$$

Using (4.48) and taking the first order term neglecting the others since  $f'(a) \gg e$  we obtain:

$$g + \varepsilon \hat{s} \approx e \exp a \quad (4.57)$$

$$|g| \approx |-\varepsilon \hat{s} + e \exp(\hat{a} - e)| \quad (4.58)$$

$$|g| \lesssim u |\exp \hat{a}| + e \exp(\hat{a} - e). \quad (4.59)$$

### 4.3.4 A-posteriori Error Estimate of Generic Power Function

Error analysis of `__ieee754_pow` function [39] proved that the error of power function is always smaller than  $u$ . We can therefore write:

$$s = a^b \quad (4.60)$$

$$s + g = \hat{a}^{\hat{b}} / (1 + \varepsilon) \quad (4.61)$$

$$s + g + \varepsilon \hat{s} = (a + e)^{b+f} \quad (4.62)$$

$$s + g + \varepsilon \hat{s} = (a + e)^{b+f} - a^b + a^b \quad (4.63)$$

$$g + \varepsilon \hat{s} = (a + e)^{b+f} - a^b \quad (4.64)$$

Using (4.48) and taking the first order term neglecting the others we obtain:

$$g + \varepsilon \hat{s} \approx (a^{b+f} - a^b) + (ba^{b+f-1} + fa^{b+f-1})e + O(e^2) \quad (4.65)$$

$$|g| \approx \left| -\varepsilon \hat{s} + a^{b+f} \left( 1 + \frac{e(b+f)}{a} \right) - a^b \right| \quad (4.66)$$

$$|g| \approx \left| -\varepsilon \hat{s} + \hat{a}^{\hat{b}} \left( 1 + \frac{\hat{b}e}{(\hat{a} - e)} \right) - (\hat{a} - e)^{\hat{b}-f} \right| \quad (4.67)$$

$$|g| \lesssim u |\hat{a}^{\hat{b}}| + \left| \hat{a}^{\hat{b}} \left( 1 + \frac{\hat{b}e}{(\hat{a} - e)} \right) - (\hat{a} - e)^{\hat{b}-f} \right|. \quad (4.68)$$

## 5 Experiments and/or Evaluation of Results

### 5.1 Mathematical Model Approximation

Another way that could lead to the extraction algorithm speed-up was to approximate the entire MOSFET mathematical model with a neural network. We have chosen GMDH type neural network [26] because it is well known for its approximation capabilities and its special version GAME-GMDH [24, A.5] seems to be very promising.

There were tried two MOSFET model approximation approaches that could lead to a potential speedup. The data that was modeled was artificial and came from mathematical model where the input data was perturbed by 5 per cent to add some noise. The former approximation used traditional GMDH neural network with polynomial transfer function with polynomials of degree 2. Needless to say GMDH polynomial neural network was unable to approximate the DC characteristic and produced 5 layers at approximately 4 neurons per layer. The mismodeled DC characteristic is shown at the following figure:

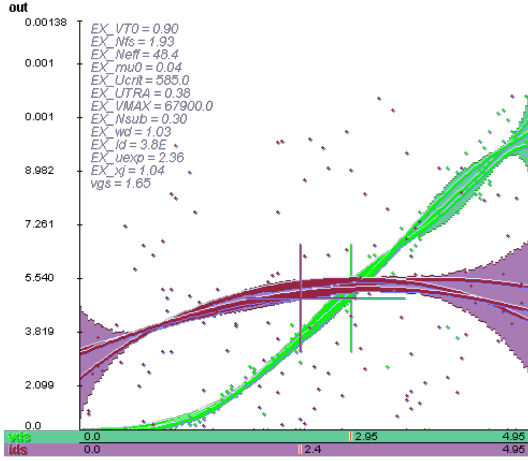


Figure 5.1: Mismodelled DC characteristic

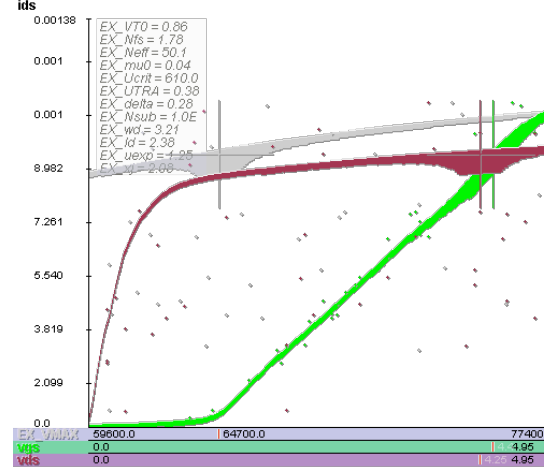


Figure 5.2: Modelled DC characteristic

The violet (dark) portion of the left figure presents the DC characteristic that came from 1500 artificial data samples with the former approach. We see that it is absolutely wrong and can conclude that polynomial neurons of the second degree at 5 layers are not able to approximate the characteristic at all. On the other hand, if GAME modeling is used, we get a completely different result. GAME is special not only in a way that it uses genetic algorithms to build neural network structure but its neurons are not just polynomial. GAME neurons can have various transfer functions such as polynomial, rational function, linear transfer function and can contain perceptron type subneural network. That is powerful enough to be able to model arbitrary data and the only cost for this advantage is that the learning phase can take several hours instead of few minutes as traditional learning usually does. The right portion of the above figure shows that GAME was able to model the DC characteristic from artificial data while the experimental precision was acceptable. GAME algorithm modeled GMDH network consisting of 9 layers and 40 complex neurons and modeling phase took over 7 hours. The network consisted of 2 3-degree rational functions neurons, 8 sigmoidal neurons, and 30 perceptron subnetworks. The resulting GAME neural network was able to model the DC characteristic while discarding few of the inputs as “not important” during the sensitivity analysis which it automatically performed. The evolution of network generation and importance of parameters can be seen at the following picture,

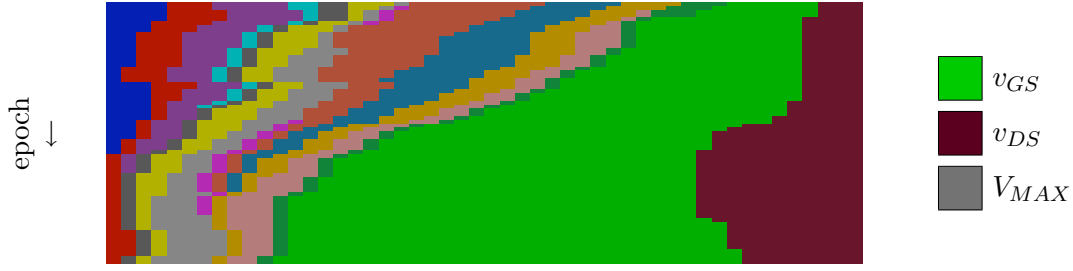


Figure 5.3: Sensitivity analysis of mathematical model

where we can see 70 epochs (top to down) together with the importance of parameters. The wider a single color area at the bottom of the figure, the higher the sensitivity is and that parameter is considered being more important.  $v_{GS}$  (gate voltage) is surely the most important parameter for the DC characteristic shape along with  $v_{DS}$  (drain voltage) and  $V_{MAX}$  (maximal voltage). Other parameters are not as sensitive for the DC characteristic and will therefore affect its value only a little. The problem with modelled network was that it was unable to adapt to greater changes of input values since it discarded them during the sensitivity analysis. If the values were not discarded, the network was too complex and did not speed up the evaluation at all. GMDH approach did not lead to the expected speedup and the durations of mathematical model evaluations are shown in table 5.1. The error was estimated by comparing GMP/MPFR 512 bit evaluation with given method.

Model evaluation method	Time [s]	Abs. prec.
GMP/MPFR 128bit	$2.219845 \cdot 10^{-3}$	$10^{-41}$
GMDH	$1.421641 \cdot 10^{-4}$	$10^{-6}$
IEEE 754	$1.062413 \cdot 10^{-5}$	$10^{-18}$

Table 5.1: Various mathematical model evaluation times

### 5.1.1 Neural Networks Conclusion

The GMDH-GAME approach was able to model the characteristic at some experimental level with an error which we found good. That, however, does not say anything about the stability as we are completely unable to tell what is the error of the modeling process and how stable the algorithm computing the transfer functions of a particular neuron is.

Supposing that each neuron takes inputs that present outputs from the previous level and there can even be links that lead from higher levels to lower ones, the rounding error may grow rapidly. It may be worth trying to apply some kind of rounding error analysis to see how stable the neural network computation was. Anyway, there is not enough control over the algorithm when approximated with a neural network and that is a reason why we will not use neural networks for approximation purposes since removing an instability in the case of neural network would mean replacing the neuron with something else and therefore reevaluating all weight coefficients in the entire network.



## 5.2 Model Speedup

This section describes an optimization attempt we have performed to increase the overall parameter extraction speed. Mathematical model's evaluation process is optimized it by means of dynamic programming and therefore heavy caching of immediate values is employed. The purpose of the mathematical model described in section 3.3 on page 14 is to evaluate a value of drain current  $i_{DS}$  and is evaluated with a C function of approximately 120 lines long. We have sampled the entire extraction process with Shark 4.0 profiler. Shark is Apple's performance tool application distributed as a portion of Apple's Xcode Developer Tools. The following table wraps the results obtained from Shark:

Percentage	Module	Function
68.2%	extrakce	mos_15
12.1%	extrakce	der_fun
5.9%	extrakce	elim
4.6%	extrakce	invx
2.8%	extrakce	main_der
2.3%	extrakce	main
1.5%	libm	dyld_stub_sqrt
1.1%	libm	dyld_stub_pow
0.5%	extrakce	negx
0.3%	libm	dyld_stub_log
0.2%	extrakce	jacob
0.2%	libm	dyld_stub_exp
0.1%	libc	dyld_stub_strcmp
0.1%	extrakce	sig
0.1%	extrakce	meas_val
0.0%	libm	dyld_stub_fmod
0.0%	extrakce	ex_print
0.0%	extrakce	int_test
0.0%	extrakce	_start
0.0%	extrakce	_dyld_init_check

Table 5.2: Profile data of parameter extraction

The analysis has shown that almost 70 per cent of the entire parameter extraction was spent in the `mos_15` function. We can expect that a significant speedup defined by Amdahl's law could be achieved if the percentage of `mos_15` function would get reduced. The speedup we could expect would be:

$$s = \frac{1}{1 - F_E + \frac{F_E}{S_E}} = \frac{1}{0.318 + \frac{0.682}{S_E}}, \quad (5.1)$$

where  $F_E$  is the fraction we are enhancing,  $S_E$  is the speedup of that fraction, and  $s$  is the overall speedup. If we improved `mos_15` function so it evaluated twice as fast as now, the expected overall speedup would be  $s \doteq 1.51745$ .

Since we do not extract all parameters of the mathematical model, a subset of them remains constant and therefore some form of caching and pre-calculation can be used at a cost of some extra memory. This will surely speed the extraction process up hence about a half of `mos_15` function can be improved this way. An analysis of `mos_15` function pointed out its critical

places. The testing program evaluated 1,000,000 times the value of mathematical model in the entire MOSFET operating area and took 11 seconds to perform. The detailed sampling analysis of the most interesting functions is shown in the following table:

Percentage	Time [ms]	Expression
11.7%	1324.4	$VDX = VDS / \text{pow}(1.0 + \text{pow}(ve, 1.0 / p \rightarrow EX\_UTRA), p \rightarrow EX\_UTRA);$
9.2%	1003.7	$Body = (2.0 / 3.0) * \text{gammaeff} * (\text{pow}(VBX + VDX, 1.5) - \text{pow}(VBX, 1.5));$
7.2%	852.3	$VGX = VA * \log(1.0 + \exp(ve)) + VTH;$
6.9%	788.8	$VA = vtemp * (1.0 + p \rightarrow PH\_qelement * p \rightarrow EX\_Nfs / Cox + \text{beta} + \text{gammaeff} / 2.0 / \text{sqrt}(VBX));$
5.1%	592.6	$\text{mueff} = \mu_0 / (1.0 + (VGX - VTH) / V_{crit} + p \rightarrow EX\_uexp * (\text{sqrt}(VBX) - \text{sqrt}(\phi)));$
5.1%	576.5	$\mu_0 = p \rightarrow EX\_mu_0 * \text{pow}(\text{tempdc} / (t_{nom} + p \rightarrow PH\_Tabs), (-1.66));$
3.4%	396.5	$\phi = 2.0 * vtemp * \log(p \rightarrow EX\_Nsub / n_i);$
3.2%	378.4	$n_i = p \rightarrow PH\_xni * \text{pow}(\text{tempdc} / (t_{nom} + p \rightarrow PH\_Tabs), 1.5) * \exp(\text{EnergT0} / 2.0 / vtempT0 - \text{Energ} / 2.0 / vtemp);$
3.2%	369.3	$\alpha_D = \text{sqrt}(1.0 + 2.0 * WD / (p \rightarrow EX\_xj)) - 1.0;$
3.0%	335.1	$VSat = v_1 - VBX + \text{gammae} * (1.0 - \text{sqrt}(1.0 + 2.0 / \text{gammae} * v_1));$
2.8%	331.1	$x_{deff} = \text{sqrt}(XD_{squ} / p \rightarrow EX\_Neff);$
2.8%	328.0	$\text{gamma}_0 = \text{sqrt}(2.0 * p \rightarrow PH\_epsilonSi * p \rightarrow EX\_Nsub * p \rightarrow PH\_qelement) / Cox;$
2.4%	279.7	$\alpha_S = \text{sqrt}(1.0 + 2.0 * WS / (p \rightarrow EX\_xj)) - 1.0;$
2.4%	278.7	$VBIN = p \rightarrow EX\_VT0 - 1.42e-3 * (p \rightarrow IA\_Temp - t_{nom}) - \text{gamma}_0 * \text{sqrt}(\phi) + \text{beta} * VBX;$
2.3%	278.7	$VDSAT = VSat + VLim - \text{sqrt}(VSat * VSat + VLim * VLim);$
2.3%	272.6	$VTH = VBIN + \text{gammaeff} * \text{sqrt}(VBX);$
2.2%	261.6	$WD = \text{sqrt}(XD_{squ} * (VBX + VDS));$
Total	8648.0	

Table 5.3: Profile data of original MOSFET mathematical model function

The function contains a lot of floating point division operation and since division is disastrously slow and is considered the lengthiest floating point operation<sup>1</sup>, elimination of division would speed up the problem. Another problem is that there is a lot of calls to `libm` functions such as `pow` and `sqrt`. These calls also take a huge amount of time and their reduction such as precaching would speed the overall extraction time. The improved version of the mathematical model ran for 7.1s, therefore there's 3.9s improvement and its profile is wrapped up in Table 5.3.

The speedup of the `mos_15` function is  $\approx 35.4$  per cent and from Amdahl's law, we could expect that the overall speedup will be  $\approx 21.7$  per cent. We could be satisfied with the result for that time and had improved model implemented into the extraction algorithm.

<sup>1</sup>This applies for all processors whose FPU does not have square root operation.

Percentage	Time [ms]	Expression
22.3%	1611.4	Body = (2.0/3.0)*gammaeff*(pow(VBX + VDX,1.5) - pow(VBX,1.5));
7.1%	543.4	VGX = VA*log(1.0 + exp(ve)) + VTH;
4.4%	321.0	VSat = v1 - VBX + gammae*(1.0 - sqrt(1.0 + 2.0/gammae*v1));
4.2%	317.0	xdeff = sqrt(XDsqu/p->EX_Neff);
4.0%	309.9	VDSAT = VSat + VLim - sqrt(VSat*VSat + VLim*VLim);
3.8%	285.7	alphaS = sqrt(1.0 + 2.0*WS*localCacheOneOverEX_xj) - 1.0;
3.7%	277.6	alphaD = sqrt(1.0 + 2.0*WD*localCacheOneOverEX_xj) - 1.0;
3.7%	276.7	phi = p->cachePhi2 + p->cachePhi1*log(p->EX_Nsub);
3.6%	266.7	deltaL = xdeff*(sqrt(VDS - VDX + VClm*VClm) - VClm);
3.2%	243.4	WS = sqrt(XDsqu*VBX);
3.0%	237.5	localCacheSqrtPhi = sqrt(phi);
3.0%	224.4	localCacheSqrtVBX = sqrt(VBX);
2.9%	223.3	WD = sqrt(XDsqu*(VBX + VDS));
2.9%	217.4	gamma0 = sqrt(p->EX_Nsub)*p->cacheGamma0;
2.4%	173.1	mueff = mu0/(1.0 + (VGX - VTH)/Vcrit + p->EX_uexp*(localCacheSqrtVBX - localCacheSqrtPhi));
Total	5528.5	

Table 5.4: Profile data of improved MOSFET mathematical model function

### 5.2.1 Model Speedup Conclusions

Surprisingly the extraction algorithm with improved mathematical model ran about 4 **more** minutes. To understand what happened, we need to realize that caching of intermediate calculations tends to keep values with higher errors and almost avoids using of initial values that have lesser errors. An example of this statement is a division to multiplication conversion. We convert a number into its reciprocal and commit therefore a possible rounding error. Another error may be committed when we multiply with this reciprocal. Speedup is achieved with this optimization since we cache the reciprocal but in the end, we can do 2 errors instead of 1 and get possibly worse result. If the original number  $x$  was small ( $0 < x \ll 1$ ), we obtain a result of high magnitude and that is very undesired [18] since it may swamp the data resulting in a loss of information. To conclude, we leave model's optimization still open and will use it with `mdouble` arithmetic and with principles sketched in the next chapter to find out optimized version of the mathematical model which still produces results at acceptable error.

## 5.3 Root Mean Square Evaluation

*Rms* error is used to find out how far the fit is from the extracted data and is evaluated as follows:

$$rms = \sqrt{\frac{1}{N} \sum_{i=1}^N \left( \frac{i_{DS}(\mathbf{x}_i, \mathbf{p}) - IDS_i}{\max\{IDS_i, IDS_{min}\}} \right)^2}, \quad (5.2)$$

where  $i_{DS}(\mathbf{x}_i, \mathbf{p})$  presents the value of drain current evaluated from the mathematical model in data point  $\mathbf{x}_i$  for parameter set  $\mathbf{p}$  and  $IDS_i$  presents  $i$ -th measured value of a drain current. The following figure shows the progress of the *rms* error during individual iterations of our extraction algorithm:

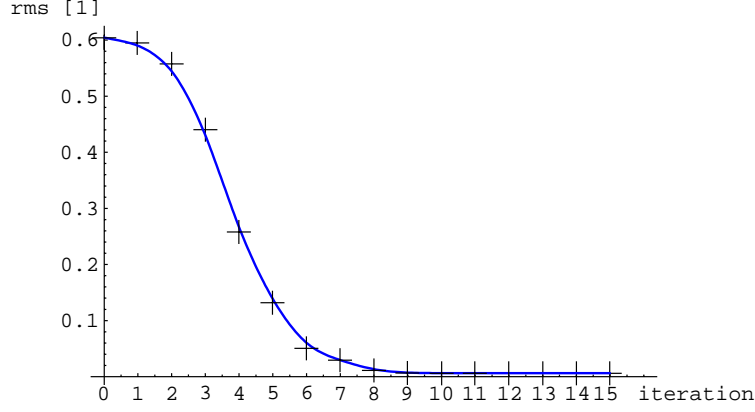


Figure 5.4: The progress of *rms* error

As the algorithm converges to a minimum (not guaranteed to be the global one) and the approximation gets better from one step to the another. We can observe that *rms* error drops and the final error at iteration 15 is  $(9.27953 \pm 0.0648948) 10^{-4}$ . The error is up to about 0.7 per cent of the value.

Since we are using running error analysis, the a posteriori error bound is calculated concurrently with the *rms* error and its relative value RREB to the *rms* error for each iteration is depicted at the figure 5.5:

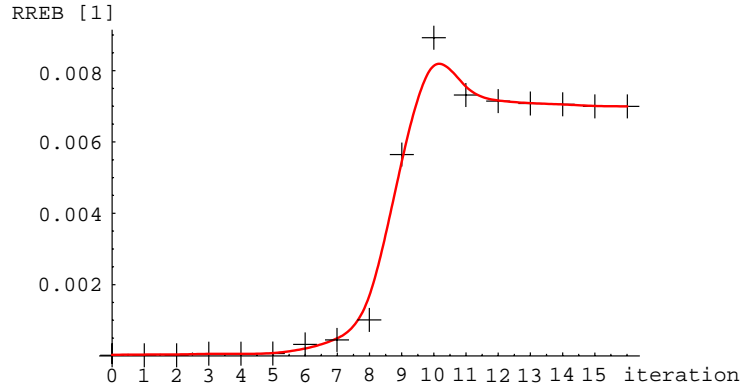


Figure 5.5: Relative running error bound for *rms* error

Hence the error bound is sharp and the running error analysis treats the error as if it happened even if it eventually did not, the value of 0.7 per cent is not bad and the result is acceptable.

#### 5.4 Comparing Various Arithmetics

This section presents experimental results obtained when employed arithmetics mentioned in sections 2.4 and 2.5 to solve a set of linear equations with Hilbert's matrix of order  $N$  [12, 25, 36]

on the left side and vector of powers of 2 going from 0 to  $N - 1$  on the right side, where  $N$  stands for SLE's size. We have used this matrix intentionally as it requires many correct significant digits. Despite such computation would probably have no sense in the real world, it provides a good example where the mantissa length needs to be long. We have evaluated the SLE twice with MPFR and once with RNS. MPFR evaluations were evaluated at precision specified by Hadamard's estimation of determinant size [31] and the second was empirically assigned. The amount of RNS prime number modules used for the evaluation of SLE was initially unknown and the calculation was stopped once the stopping criterion was achieved. The results obtained are presented at the following figure:

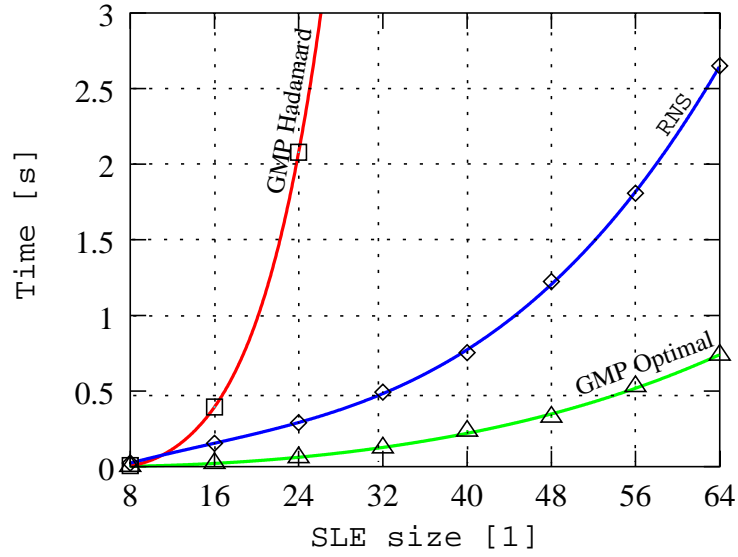


Figure 5.6: SLE Solve Time for various arithmetics

## 6 Conclusions

This chapter concludes the observations we have obtained from the parameter extraction algorithm we have developed and provides topics of our further research concerning MOSFET mathematical model parameter extraction. Observations start off with a sketch of the parameter extraction algorithm and continue with a-posteriori error estimates observations, `mdouble` arithmetic and derivative evaluation. Chapter ends by providing further research topics.

### 6.1 Parameter Extraction

The parameter extraction algorithm described in 3.5 on page 18 consists of several parts and there are many problems such as model evaluation, its accuracy, and we have to deal with them in order to provide a numerically stable, system level optimized algorithm, which is able to obtain values of MOSFET parameter in real time.

### 6.2 A-posteriori Error Estimates

Since backward/forward error analyses of MOSFET mathematic model described in section 3.3 on page 14 tend to be very difficult, we have decided to use a-posteriori error estimates described in 4.2.2 on page 38. A-posteriori error estimates proved to be a good way how to calculate a sharper a posteriori error bound simultaneously with the value of drain current. We are able to obtain a tuple of drain current value and its error so we are able to tell if we have made some operation which would magnify error in a large way and also see how the input error projects to the output. `mdouble` arithmetic described in section 4.2 on page 36, an IEEE 754 arithmetic enriched with a-posteriori error estimates works with tuples  $\{value, error\}$  rather than with *value* only have been developed and error bounds have been derived for basic operators in section 4.2.4 and common function in section 4.3 on pages 38–39. This arithmetic can be easily used to replace already existing IEEE 754 double precision arithmetic code already written in a matter of minutes.

#### 6.2.1 Derivative evaluation

Our algorithm also needs derivative of the mathematical model and numerical derivative proved to be too inaccurate and therefore we have decided to implement analytical derivative which is evaluated by symbolic manipulation rather than approximation. Such algorithm leads to write a parser which reads C source code (see section 4.2) and transforms it into an internal tree form. This internal form can be evaluated at will and can be also written back into C in IEEE 754 arithmetic or GNU MP/MPFR (see 2.5 on page 11) by means of the code generator module. The internal expression form has a symbol table which keeps a tree representation of an expression represented with a particular symbol name. Deriver module has been developed which takes a symbol and transforms the entire tree onto that symbol's derivative by constructing symbol dependencies and walking the tree while performing derivative rules such as chain rule, etc. This way we transform the internal tree into its analytical derivative and have it written back into C with the code generator module.

### 6.3 Further Research Possibilities

The following sections describe our further research topics. These include a-posteriori error estimates, which are planned to be extended on the entire parameter extraction, system level optimization, parallelization of the algorithm and self stability checks.

#### 6.3.1 Extend A-posteriori Estimates

Our intention is to extend a-posteriori error estimates we have performed on the mathematical model, derivative evaluation and *rms* error evaluation on the entire extraction algorithm. That includes core SLE solving as required by (3.33) on page 18 and statistic quantifiers which are evaluated on the final parameter set once the parameter extraction is done.

#### 6.3.2 Perform System Level Optimization of Numeric Codes

We would like to perform optimization of the numeric codes at the system level so it is able to use processor features such as SIMD type instructions to achieve better speed while still keep algorithm stable.

#### 6.3.3 Make Parallel Extraction System

In order to extract more parameter in a real time, a need to run in parallel arises. With the help of the arithmetic of the residual number system, we are able to easily scale the entire parameter extraction algorithm to run on several computers at once.

#### 6.3.4 Perform A-posteriori Error Estimates of RNS

It may be interesting to know how input errors propagate through RNS calculation and this topic will be covered by our further research.

#### 6.3.5 Perform Self Stability Checks

Another thing that will be a topic of our further research as the self stability checks. Since the evaluator module which reads the source code transforms the expression into a tree form and each operator presents a node, we can watch how a-posteriori error bound grows and if necessary reorganize the tree to try to remove critical cancelations or other possibly harmful operations.

## 7 Bibliography

- [1] B. Ankele, W. Hölzl, and P. O’Leary. Enhanced mos parameter extraction and spice modelling for mixed analogue and digital circuit simulation. *IEEE Microelectronic Test Structures*, pages 73–78, 3 1989.
- [2] J. V. Beck and K. J. Arnold. *Parameter Estimation in Engineering and Science*. John Wiley & Sons, 1977.
- [3] Beigebag software, inc., b2 spice v5. Software, 2005. <http://www.beigebag.com>.
- [4] P. Bevington and D. K. Robinson. *Data Reduction and Error Analysis for the Physical Sciences*. McGraw-Hill Science/Engineering/Math, 2002.
- [5] M. K. Brown. Derivative free analogues of the levenberg-marquardt and gauss algorithms for nonlinear least squares approximation. *J. Numer. Math*, 18:289–297, 1972.
- [6] G. Dahlquist and Å. Björck. *Numerical Mathematics and Scientific Computation*. To be published with SIAM, PA, USA, 2004.
- [7] C. Donnelly and R. M. Stallman. *The YACC-Compatible Parser Generator*. Free Software Foundation, 2003.
- [8] M. Dont. *Elementy numerické lineární algebry*. ČVUT, 2004.
- [9] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Università Politecnica di Milano, 1992. DT.01-POLIMI92.
- [10] H. W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Kluwer Academic Publishers, 2000.
- [11] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Jul 1997.
- [12] G. H. Golub and C. F. V. Loan. *Matrix Computations 3rd Ed*. The Johns Hopkins University Press Baltimore, 1996.
- [13] T. Granlund. *GNU Multiple Precision Arithmetic Library*. Free Software Foundation, 2005.
- [14] R. T. Gregory. *Error-free computation: Why It Is Needed and Methods For Doing It*. Robert E. Krieger Publishing Company, 1980.
- [15] R. T. Gregory and E. V. Krishnamurthy. *Methods and Application of Error-free Computation*. Springer Verlag, 1984.
- [16] G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. *Multiple Precision Floating-Point Reliable Library*. Free Software Foundation, 2004.
- [17] A. Hertz, E. Taillard, and D. de Werra. A tutorial on tabu search. In *Proc. of Giornate di Lavoro AIRO’95 (Enterprise Systems: Management of Technological and Organizational Changes)*, pages 13–24, Italy, 1995.
- [18] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996.
- [19] S. V. Huffel and J. Vandewalle. *The total least-squares problem*. SIAM Philadelphia, 1991.



- [20] IEEE Computer Society Standards Committee. Working group of the Microprocessor Standards Subcommittee and American National Standards Institute. *IEEE standard for binary floating-point arithmetic*. ANSI/IEEE Std 754-1985. IEEE Computer Society Press, 1985.
- [21] S. H.-M. Jen, C. C. Enz, D. R. Pehlke, M. Schröter, and B. J. Scheu. Accurate modelling and parameter extraction for mos transistors valid up to 10 ghz. *IEEE Transactions on Electron Devices*, 46(11), 1999.
- [22] M. Keser and K. Joardar. Genetic algorithm based mosfet model parameter extraction. *Technical Proceedings of the 2000 International Conference on Modeling and Simulation of Microsystems*, pages 341 – 344, 2000.
- [23] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.
- [24] P. Kordík, M. Náplava, M. Šnorek, and M. Genyk-Berezovskyj. The modified gmdh method applied to model complex systems. In *Proceedings of International Conference on Inductive Modelling 2002*, pages 150–155, 2002.
- [25] R. Lórencz. *Aplikovaná numerická matematika a kryptografie*. Vydavatelství ČVUT, 2004.
- [26] A. Madala and H. Ivakhnenko. *Inductive Learning Algorithm for Complex System Modelling*. CRC Press, 1994.
- [27] V. Maniezzo, L. M. Gambardella, and F. D. Luigi. *Ant Colony Optimization, New Optimization Techniques in Engineering*, volume 1, chapter 5. Springer-Verlag, 2004. <http://www.idsia.ch/~luca/aco2004.pdf>.
- [28] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of Applied Mathematics*, 11(2):431–441, 1963.
- [29] M. Murakawa, M. Miura, and T. Higuchi. Towards automatic parameter extraction for surface-potential-based mosfet models with the genetic algorithm. *Proceedings of the 2005 Asia South Pacific Design Automation Conference (ASP-DAC 2005)*, 1:204–207, 2005.
- [30] M. Navara and A. Němeček. *Numerické metody*. Vydavatelství ČVUT, 2003.
- [31] M. Newman. Solving equations exactly. *J. Res.*, 71B:171–179, 1967.
- [32] M. Obitko and P. Slavík. Visualization of genetic algorithms in a learning environment. In *Proceedings of Spring Conference on Computer Graphics*, pages 101–106. Comenius University, 1999.
- [33] V. Paxson. Fast lexical analyzer generator, 1988. <http://www.gnu.org/software/flex/>.
- [34] W. H. Press, W. T. Vetterling, S. A. Teukolsky, and B. P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1999.
- [35] K. Rektorys and kol. *Přehled užití matematiky I*, volume 1. Nakladatelství Prometheus, 1995.
- [36] K. Rektorys and kol. *Přehled užití matematiky II*, volume 2. Nakladatelství Prometheus, 1995.
- [37] M. R. Spencer. *Polynomial Real Root Finding in Bernstein Form*. PhD thesis, Brigham Young University, 1994.

- [38] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.
- [39] Sun microsystems error analysis of `ieee754_` class functions. Not published (internal documentation), 1996. <http://www.sun.com>.
- [40] K. Tsuru. *SuperNumber Library*. Freeware, 2002.
- [41] D. E. Ward and K. Doganis. Optimized extraction of mos model parameters. *IEEE Transactions on Computer-Aided Design*, CAD-1:163–168, 8 1982.
- [42] J. H. Wilkinson. State of the art in error analysis. *NAG Newsletter*, 2/85, 1985.
- [43] J. H. Wilkinson. Error analysis revisited. *IMA Bulletin*, 22(11/12):192–200, 1986.
- [44] Wolfram research inc., mathematica 5.2. Software, Jul 2005. <http://www.wolfram.com>.
- [45] B. V. Zeghbroeck. *Principles of semiconductor devices*. Electrical and Computer Engineering Department, University of Colorado at Boulder, 2004. <http://ece-www.colorado.edu/~bart/book/>.
- [46] E. Zitzler, M. Laumanns, and S. Bleuler. A tutorial on evolutionary multiobjective optimization. In *Workshop on Multiple Objective Metaheuristics (MOMH 2002)*. Springer Verlag, 2003.

## 8 Relevant refereed publications of the author

- [A.1] T. Zahradnický and R. Lórencz. MOSFET Model Parameter Extraction Experience, 2005, Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic, Proceedings of 27th International Autumn Colloquium on Advanced Simulation of Systems, Přerov, Czech Republic Accepted for presentation at ASIS 2005 in September 2005.
- [A.2] T. Zahradnický. Mathematical Aspects of MOSFET Parameter Extraction, 2005, 80-01-03298-1, pp. 165–169, Department of Computer Science and Engineering, Faculty of Electrical Engineering, The Czech Technical University, Prague, Czech Republic, Počítačové architektury a diagnostika PAD 2005.
- [A.3] T. Zahradnický and R. Lórencz. Numerically Stable SPICE MOSFET Model Parameter Extraction, 2004, 80-8073-150-0, pp. 170–175, Department of Computers and Informatics of FEI, Technical University Košice, Slovak Republic, Proceedings of the Sixth International Scientific Conference on Electronic Computers and Informatics ECI 2004.
- [A.4] T. Zahradnický. MOSFET Mathematical Model Approximation Using GMDH Neural Network, 2004, 80-969202-0-0, pp. 100–105, Ústav informatiky SAV, Bratislava, Slovak Republic, Počítačové architektúry a diagnostika PAD 2004.
- [A.5] T. Zahradnický and R. Lórencz. MOSFET Mathematical Model Parameter Extraction, 2004, 980-6560-17-5, pp. 400–404, Austin, Texas, United States, Proceeding of International Conference on Computing, Communications and Control Technologies CCCT 2004, International Institute of Informatics and Systemics.
- [A.6] T. Zahradnický. Lossless real time parameter extraction of MOS-FET DC characteristic. 2003, 80-214-2471-0, pp. 51–53, Liberec, Czech Republic, Proceeding of Počítačové Architektury & Diagnostika PAD 2003, Technical University of Brno.

- [A.7] T. Zahradnický. Výpočetní jádro pro modelování dat pomocí extrakce parametrů. Master Thesis, 2003, FEL ČVUT.

## 9 Remaining refereed publications of the author

- [A.8] T. Zahradnický. FileMaker Pro Plug-Ins. 2003, In: The Book of FileMaker 6, No Starch Press, pp. 506–555, 1-886411-81-6, San Francisco, CA, USA

## 10 Unrefereed publications of the author

- [A.9] T. Zahradnický and R. Lórencz. Modelling a MOSFET: Speed vs. Accuracy, 2005, Proceedings of Workshop 2005 (CDROM), Prague, Czech Republic
- [A.10] T. Zahradnický and R. Lórencz. MOS FET Mathematical Model Parameter Extraction Sensitivity, 2004, 80-01-02945-X, pp. 40–41, Proceedings of Workshop 2004 (CDROM), Prague, Czech Republic
- [A.11] R. Lórencz and T. Zahradnický. Error-Free Parameter Extraction Using GMP Library and Maple. 2003, 80-01-02708-2, pp. 340–341, Proceedings of Workshop 2003 (CDROM), Prague, Czech Republic
- [A.12] T. Zahradnický and R. Lórencz. Error-Free Parameter Extraction of MOS FET Using Waterloo Maple and GMP Library. 2003. 80-7083-708-X, pp. 139–143, Liberec, Czech Republic, Proceeding of conference on Electronical Computing and Measurement Systems, ECMS 2003, Technical University of Liberec.

## 11 Dissertation Thesis

**Title:** MOSFET Parameter Extraction

### Abstract

Parameter extraction is a widely used method of obtaining parameters and there exist several approaches. Some of them use Levenberg-Marquardt's algorithm, other use Genetic Algorithms combined with Multiobjective Optimization. Vast majority of methods are able to extract values but their approach to the error analysis and knowledge of the error of the result they obtain is poor. This is because such methods do not deal with ill-conditioness of the parameter extraction problems and are limited with the number of parameters. Our parameter extraction methodology cares about errors to and due to this is able to go beyond the limits of these methods. The method is able to extract many parameters all-at-once and also keep the same environment conditions for all parameters. The self-monitoring and validation of the evaluation process methods proposed with this dissertation thesis proposal are used to keep the numerical stability of the extraction algorithm but increase the run time. To be able to perform the parameter extraction in the real time, this work performs system dependent system optimization of special numeric codes with respect to hardware aspects such as scalability and parallelism.

### Keywords

Error Analysis, MOSFET, Numerical Stability, Parallel Computing, Parameter Extraction, Self-Validation, System Optimization

### 11.1 Parallel, Numerically Stable MOSFET Parameter Extraction

Since our intention is to run in a real time and validation methods proposed in 11.2 increase extraction run time, the idea is to extend the already existing parameter extraction process to run in a parallel environment. This can be achieved by using system dependent optimization with respect to scalability and appropriate arithmetic such as the arithmetic of residual number system. The benefits of the arithmetic of the residual number system relevant to parallel computing are described in section 2.4 on page 10.

### 11.2 Self-Validated MOSFET Parameter Extraction

Our approach to an analytical derivative provides a way how to transform the entire mathematical model into an internal tree form. Each operator in the mathematical model is represented with a tree node and each symbol in the model has an underlying tree attached to it. Evaluator and Deriver tools described in section 4.1.2 on page 28 provide a way how to easily evaluate the tree form or transform it onto an analytical derivative of itself. Because of the fact, that each operator is represented by a node and each node has its own evaluation function, we are able to intercept the calculation at will and with the help of `mdouble` described in section 4.2 on page 36 we are able to evaluate an a-posteriori error bound simultaneously with the calculation. The idea of self-validating could extend this process by checking numerical stability of the code by detecting unstabilities and possibly transforming the tree form into a numerically stable form or advising user that such problem occurred.