Macromodeling for Mixed Signal Circuits

by J. Chen
Advisor: L. Milor
Macromodeling for Mixed Signal Circuits

by

Chen Jie

Thesis submitted to the Faculty of the Graduate School of the University of Maryland in partial fulfillment of the requirements for the degree of Master of Science 1992

1Supported by - Department of Electrical Engineering, University of Maryland at College Park and Minta Martim Foundation
ABSTRACT

Title of Thesis: Macromodeling for Mixed Signal Circuits
Jie Chen, Master of Science, 1992

Thesis directed by:
Linda Milor
Assistant Professor
Department of Electrical Engineering
System Research Center
University of Maryland, College Park, MD 20742

Macromodels of functional blocks of VLSI circuits are used to reduce the computational effort involved in the detailed simulation of general-purpose circuits. However the macromodeling approaches in the literature have considerable limitations both in the type of circuits that can be modeled or the functions that can be simulated. In this paper we propose a method to generate macromodels for any kind of circuit and any kind of function that needs to be simulated, without restrictions. Furthermore our approach has superior convergence properties, compared to the previous results in the literature. Therefore our method can get optimal macromodels in reasonable time and the models are accurate enough to be used for general circuit simulation.
Acknowledgments

I would like to thank my research advisor, Assistant Professor Linda Milor for suggesting the topic of this thesis; her outstanding work in the area of Computer-Aided Simulating and Testing of Analog Circuits will always be an example for me to look up to. I would also like to thank Professor Andre L. Tits for suggesting the outline of unconstrained and constrained optimal theory. I am indebted to Professor Ray Chen for his kind support for my thesis work. I also appreciate the efforts of J. Zhou, Graduate Research Assistant of Professor Andre L. Tits for his helpful discussion about CONSOLE software tandem.

At the beginning of this research project, Professor Giorgio Casinovi of Georgia Institute of Technology released his previous software package developed in U.C. Berkeley to me and suggested the outline of his work. Here I would like to thank him for his valuable help.

Financial support received from Electrical Engineering Department, Univ. of Maryland at College Park and Minta Martin Foundation is also gratefully acknowledged.

The computer implementation of the macromodeling would not have been possible without the valuable advice of Jyh-Fong Lin, Po-Yang F. Lin, Shu-Sun Yu and other students working in VLSI Design and Testing Lab.

Finally, I would like to express my appreciation for the work of Ravi Kolagotla as system administrators of our computers.
Contents

1 Introduction 1

1.1 Introduction .................................................. 1

1.1.1 Previous Work in Macromodeling .......................... 1

2 The Modeling Methodology 6

2.1 The Modeling Methodology ................................. 6

3 Minimizing the Cost Function 9

3.1 Minimizing the Cost Function ......................... 9

3.1.1 Optimization Theory ..................................... 10

3.1.2 Saddle Point Algorithm ................................. 11

3.1.3 FSQP — Feasible Sequential Quadratic Programming 12

4 Implementation 13

4.1 Implementation ............................................. 13

4.1.1 Computing the cost function ......................... 15
4.1.2 Implementation of the Interface .................................... 15

5 Numerical Results .......................................................... 18

5.1 Numerical Results ......................................................... 18

5.1.1 Input Part of an Operational Amplifier ......................... 18

5.1.2 Operational Amplifier ................................................ 22

5.1.3 Filter ........................................................................ 25

6 Conclusions .................................................................. 30

6.1 Conclusions ................................................................. 30

A FSQP Algorithm .............................................................. 34

B CONSOLE .................................................................... 35

C Nonlinear Optimization ................................................... 37

C.1 Unconstrained Problems ............................................... 37

C.1.1 First order methods .................................................... 39

C.1.2 Newton's Method ....................................................... 39

C.1.3 Quasi-Newton Method ............................................... 39

C.2 Constrained Minimization ............................................. 40

C.2.1 Equality and Inequality Constraints ............................. 40

C.2.2 Feasible Sequential Quadratic Programming — FSQP ...... 41
List of Figures

1.1 Macromodel of an operational amplifier for its transient response after optimization ........................................ 3

1.2 Macromodel of an operational amplifier for its frequency response after optimization of the transient response. ........................................ 4

1.3 A macromodel with internal feedback. ........................................ 4

2.1 This is an functional block structure. ........................................ 7

2.2 Modeling Methodology ........................................ 7

4.1 This is overall structure of macromodeling ........................................ 14

5.1 This is a schematic diagram and input file for op amp to be modeled .... 19

5.2 This is a schematic and input file for macromodel ........................................ 20

5.3 This is the PDF file for macromodeling ........................................ 20

5.4 Macromodel for the operational amplifier of op amp ........................................ 21

5.5 Macromodel and operational amplifier outputs before optimization. .... 22

5.6 Macromodel and operational amplifier outputs after optimization. .... 23
List of Tables

5.1 The performance of macromodeling op amp ....................... 21
Chapter 1

Introduction

1.1 Introduction

The design of large scale mixed signal circuits uses a modular design style, in which multiple instances of cells are used repeatedly in different parts of the circuits. It is well known that as integrated circuits scale down to smaller sizes, the computational effort involved in the detailed simulation of such designs increases rapidly. Especially in the analog domain, circuits are usually simulated at the transistor level and this is very CPU intensive. In order to reduce computational cost, a number of alternate approaches to circuit simulation have been developed. These approaches provide a tradeoff between cost and accuracy. One such approach is macromodeling. A macromodel refers to an electrical network containing fewer devices and/or fewer nodes than the circuit it represents where at least one performance matches that of the circuit it models. In fact, the use of macromodel is premised on the idea that some detail in a circuit model can be eliminated if only one or two responses of that circuit are of interest.

1.1.1 Previous Work in Macromodeling

The task of circuit simulation is to solve differential equations accurately enough to predict the circuit's static and/or dynamic behavior. For digital circuits, since the transient response is of primary concern, macromodels attempt to predict delays and logic levels. The literature on macromodeling in this area include:
• logic [1], [2] phase simulation which computes the binary behavior of the circuit the transistor or gate level,

• switch level [3], timing [4], [5] simulation which attempts to analyze the various delays in the circuit's critical paths without solving the differential equation with great accuracy, and

• quasi-static timing simulation [7] for gates, where a cell is modeled by a set of delays and boolean equations.

Such models are very efficient for digital simulation, since they neglect much of the detailed electrical performance of the circuit. However this type of macromodel is inappropriate for many applications which need accurate circuit simulation, like for example when race conditions and charge sharing need to be predicted. A more accurate approach was proposed in [6] where the macromodel could take into account the effects of waveforms of the functional blocks with very stringent assumptions about the shape of the waveforms, which the author approximates with time-shifted ramps with exponential tails $\sum_i e^{-R_i}$, where $R_i$ is the function of time. Most digital integrated circuits can be approximated with good accuracy using this method, but the assumption about the shape of the waveform essentially limits the scope of those methods to the macromodeling of digital circuits. Another contribution was made in [8]. This method is applicable to most of circuits without restrictive assumptions about waveforms involved despite the fact that the example in that paper is for a NAND gate, but this approach is limited to macromodeling the dc response.

In a recent paper [9], the authors propose a general purpose algorithm that can be used to generate models for transient responses of MOS circuits. It uses a macromodel with a nonlinear devices (e.g. a nonlinear voltage controlled current source using a look-up table description). The theoretical results are impressive, since the authors consider the problem of finding the best macromodel in all possible transient input waveforms. However the authors did not consider the behavior of the output current and responses other than the transient response. It is sometimes important to model a response other than the transient response. For example, if an operational amplifier is to be used as a component in a filter, then its frequency response is of interest. Suppose for example that we build a macromodel for the transient response of this circuit [Fig. 1.1]. There is no reason to believe that the frequency response is also optimized and indeed it is not optimal [Fig. 1.2]. Obviously, there is a need to develop an algorithm that will macromodel any circuit performance. And sometimes it may be necessary to build a macromodel that will approximate more than one circuit performance, like both the voltage and current
Figure 1.1: Macromodel of an operational amplifier for its transient response after optimization.

Our main concern in this thesis is to develop a macromodeling system which can macromodel any kind of circuit and any kind of function or set of functions that need to be simulated without any restriction. Some of the main differences between our work and most recent papers on macromodeling are the following:

- **Our algorithm can macromodel any function that can be simulated by SPICE [11]. We can handle not only the transient analysis but also AC simulation, NOISE analysis, and even distortion analysis.**

- **The structure of the macromodel is limited only by the SPICE element set. We can therefore model analog circuits with internal feedback, by including a Miller capacitor (Ct) and resistor (Rt) [see Fig. 1.3].**

- **We define our cost function not only as the difference of voltages of the original circuit and macromodel but also as the difference of currents when necessary. Therefore we can macromodel a very wide class of circuits, especially the more challenging analog case. Our macromodeling technique can also be applied to digital circuits,**
Figure 1.2: Macromodel of an operational amplifier for its frequency response after optimization of the transient response.

Figure 1.3: A macromodel with internal feedback.
In this thesis, we propose a bottom-up approach to macromodeling, in order to handle large scale integrated circuits with a complex structure. In an example, we show that by starting with the macromodeling of small blocks, we can proceed by building macromodels of larger blocks. Another contribution of this thesis is the use of Feasible Sequential Quadratic Programming (FSQP) [14] to solve the constrained optimization problem of finding the best parameters for the given structure of the macromodel. The Feasible Sequential Quadratic Programming algorithm is a quasi-Newton optimization algorithm, and hence under some conditions that are satisfied most of the time for circuit optimization, the rate of convergence is superlinear [14], [13]. The algorithm also maintains feasibility of the solution. This optimization algorithm has some superior properties to the algorithm in [9]. In particular, the algorithm in [9] solves an unconstrained optimization problem, and hence unless the initial parameters are close to the optimal solution, the sequence of feasible solutions is likely to diverge, causing the simulator to eventually crash. To solve this problem, we allow constraints on macromodel parameters, so in the worst case, a "best" set can be found which is simulatable by SPICE. Hence, the initial set of macromodel parameters do not have to be close to optimal, in order to avoid divergence of the sequence of feasible solutions.

This thesis is organized as follows. In the next section, we define the cost function which is the weighted sum of the differences of output behavior of one or many responses between the original circuit and macromodel. This specifies that the goal of macromodeling of a circuit is to minimize the cost function and find the local minimizer (the parameters of the macromodel). In section 3, we introduce the FSQP algorithm and discuss its properties. We then describe in section 4 the implementation of our macromodeling system. In section 5, we will present some examples of macromodeling and their numerical results. And in the last section, we conclude and discuss future work on this topic.
Chapter 2

The Modeling Methodology

2.1 The Modeling Methodology

A circuit may be viewed as a collection of \( l \) subcircuits, some of which may consist of individual devices such as transistors, whose terminals are connected together at nodes. (see Fig. 2.1). Each subcircuit \( S_j \) can be viewed as a multiterminal element with its terminals attached to nodes; each terminal has an associated voltage and current, which is inherited from the node to which the terminal is connected. Let \( \mathbf{r}_a = (r_1, \ldots, r_m) \) be the vector of the current flowing into the input nodes of subcircuit \( X \), and let \( \mathbf{v}_a = (v_1, \ldots, v_k) \) be the vector of input node voltages. Suppose we wish to simulate subcircuit \( a \), then the input-output behavior of this original circuit is described by the following set of equations:

\[
\begin{align*}
\begin{cases}
\vec{I}_a = F_a(\vec{v}_a, \vec{r}_a) \\
\vec{V}_a = G_a(\vec{v}_a, \vec{r}_a)
\end{cases}
\end{align*}
\]  

(2.1)

where \( \vec{I}_a \) and \( \vec{V}_a \) are the output currents and voltages of subcircuit \( a \) respectively. Similarly for a macromodel the output currents and voltages, \( \vec{I}_b \) and \( \vec{V}_b \) are functions of voltages and currents on the input terminals, \( \vec{v}_b, \vec{r}_b \).

\[
\begin{align*}
\begin{cases}
\vec{I}_b = F_b(\vec{v}_b, \vec{r}_b) \\
\vec{V}_b = G_b(\vec{v}_b, \vec{r}_b)
\end{cases}
\end{align*}
\]  

(2.2)

A macromodel usually refers to a compact representation of a circuit, discarding redundant information when computing a particular performance or set of performances. In other words, a macromodel may only be able to model some of the circuit performances, but it does so with less computational cost compared to the original circuit. That is to say, the
The functional block to be modeled

(With k nodes)

\[ \text{(I}_1, \text{V}_1) \]

\[ \text{(I}_2, \text{V}_2) \]

\[ \text{(I}_n, \text{V}_n) \]

Figure 2.1: This is an functional block structure.

macromodel should require the solution of fewer differential equations to get the desired performance, since generally speaking, the macromodel contains fewer nodes and devices than the original circuit.

Our modeling methodology applies the same inputs to the original circuit and macromodel and matches the circuit output voltage and current waveforms which are obtained by dynamic and/or static simulation [see Fig. 2.2]. To do this, the user must propose a topology for the macromodel, and then the algorithm finds the parameters of the macromodel that minimize the differences in responses between the original and the macromodel.
In order to quantify the difference between responses, suppose we want to build a macromodel for response $w_a(\chi)$ of the original circuit. $w_a$ denotes any circuit performance, including voltage and current transient responses, voltage and current ac responses, etc. $\chi$ is in general a sweep variable, including time, frequency, or even voltage. The response of the macromodel, for the same inputs, $w_b(\chi, \xi)$, not only depends on the sweep variable, but also the parameters, $\xi$, of the macromodel.

At any value of $\chi$, the sweep variable the difference in performance between the original circuit and the macromodel is the following:

$$C(\xi) = \|w_b(\chi, \xi) - w_a(\chi)\|$$  \hspace{1cm} (2.3)

Since $\chi$ is a continuous variable, the total difference between the macromodel and original performance must be integrated, where $\chi$ is the finite upper bound (a restriction).

$$C(\xi) = \int_{0}^{X} \|w_b(\chi, \xi) - w_a(\chi)\|^2 d\chi$$  \hspace{1cm} (2.4)

Finally if $n$ performances need to be matched then the cost function is the weighted sum of the integrated difference for each performance.

$$\text{Cost}(\xi) = \sum_{j=1}^{n} \nu_j \int_{0}^{X} \|w_b^{(j)}(\chi_j, \xi_j) - w_a^{(j)}(\chi_j)\|^2 d\chi_j$$  \hspace{1cm} (2.5)

Note that $w^{(j)}$ denotes the $j$th performance, and $\chi_j$ is the corresponding sweep variable. The weights, $\nu_j$, account for the different units of the performance, and can be used to give more importance to one response over another. For example, in eq. (2.5) if $w^{(1)}$ is a voltage transient response and $w^{(2)}$ is a current transient response, the units of current and voltage are different on the order of $10^{-6}$ to $10^{-3}$. So $\nu_1$ and $\nu_2$ will differ by the same order of magnitude. Note that the above cost function can be used for the transient response, as well as the at small-signal ac response, and/or any function that can be simulated. Except in the case where the input resistance of the circuitry connected to the output nodes is infinite, in both the transient and ac cases, it is important to include both voltage and current responses in the cost function, $\text{Cost}(\xi)$.

The cost function makes it possible to quantify the accuracy of a macromodel. In particular, the best macromodel is the one that minimizes the cost. Note that we are restricting our set of possible macromodels to a given topology, and therefore our problem is to find the optimal $\xi$ for that topology. If the optimal macromodel for the proposed topology is not acceptable, then the user may propose an alternative topology for the macromodel.
Chapter 3

Minimizing the Cost Function

3.1 Minimizing the Cost Function

In the previous section we introduced the cost function to measure the difference between the static and/or dynamic behavior of the original circuit and the macromodel. In order to achieve the closest macromodel, we should minimize such a difference as represented by the cost function.

\[
\min_{\xi} (\text{Cost})
\]

subject to

\[
\begin{pmatrix}
\tilde{I}_a \\
\tilde{I}_b
\end{pmatrix} =
\begin{pmatrix}
\mathcal{F}_a(\bar{v}_a, \bar{v}_a) \\
\mathcal{F}_b(\bar{v}_b, \bar{v}_b)
\end{pmatrix}
\]

\[
\begin{pmatrix}
\tilde{V}_a \\
\tilde{V}_b
\end{pmatrix} =
\begin{pmatrix}
\mathcal{G}_a(\bar{v}_a, \bar{v}_a) \\
\mathcal{G}_b(\bar{v}_b, \bar{v}_b)
\end{pmatrix}
\]

(3.2)

Here \(\xi\) refers to the set of macromodel parameters. Because the circuit simulator (SPICE) will solve the differential equations (3.2) to predict detailed electrical properties of the circuit, mathematically speaking, the best macromodel is a solution of the following problem:

\[
\begin{cases}
\min_{\xi} (\text{Cost}(\xi)) \\
\text{subject to } c_i(\xi) \leq 0, \quad i = 1, 2, \ldots, n.
\end{cases}
\]

(3.3)

where \(c_i(\xi)\) are the constraint functions on the values of parameters. The reasons to put constraint function are several. First, the constraints make the parameters in the macromodel have physical meaning, e.g. the value of resistance normally should be positive.
Second, the constraints on parameters will prevent the algorithm from crashing because of divergence. If a divergent sequence is generated, the optimal solution will be on the upper/lower bounds of the parameters.

Actually solving eq. (3.3) is equivalent to computing the global optimum of a general real-valued function which is almost impossible. The best that can be hoped for is to compute a local minimum of Cost(ξ), which means that the corresponding macromodel is the best among all those that can be obtained from it by small parameter variations.

### 3.1.1 Optimization Theory

The minimizing point or minimizer of (eq. 3.3) is referred to as ξ*. Note first that finding a global minimizer is not practical for our problem. On the other hand, a number of results are available in the literature that search for a local minimizer. Many early methods which were suggested for minimization were based on searching along the direction of the negative gradient, or in more general directions. The basic algorithm is the following:

- determine the direction of search \( d_j \),
- find step size \( \alpha_j \) to minimize Cost(\( \xi_j + \alpha_j d_j \)), and
- set \( \xi_{j+1} = \xi_j + \alpha_j d_j \).

Different methods correspond to different ways of choosing \( d_j \). Finding \( \alpha_j \) exactly is generally not computationally desirable. Usually the line search is stopped as soon as some adequate decrease in Cost is achieved.

A more computationally efficient approach is Newton’s method. Newton’s method relies on a local second order expansion of Cost. Newton’s method provides quadratic convergence whenever the step size \( \alpha_j \) is equal to one. A disadvantage of Newton’s method for circuit simulation problems is that the user must supply formulae from which the second derivative matrix \( G(\xi) = \nabla^2 \text{Cost}(\xi) \) can be evaluated. This is often a major disincentive, since standard circuit simulators don’t compute the second derivative when solving the differential equations. However methods closely related to Newton’s method, called quasi-Newton methods, can be used when only first derivatives are available, and first derivatives are generally easier to compute and are determined during circuit simulation. These methods don’t have quadratic rates of convergence like Newton’s method, but
have superlinear rate of convergence. These methods approximate $G_j^{-1}$ by a symmetric positive definite Hessian matrix $H_j$, which is updated from iteration to iteration. It is important that $H$ remains positive definite so that the corresponding search direction is a direction of decrease for the cost function.

The constrained optimization is similar, except the constraints in equation (3.3) are taken into account. That is to say, the search direction is determined not only by a second order approximation of Cost, but also a first order approximation of the constraints:

$$\begin{align*}
\min_{d_j} & \quad \text{Cost}(\xi_j) + \langle \nabla_\xi \text{Cost}(\xi_j), d_j \rangle + \frac{1}{2} \langle d_j, H_j d_j \rangle \\
& \quad c_i(\xi_j) + \langle \nabla_\xi c_i(\xi_j), d_j \rangle \leq 0, \quad i = 1, 2, \ldots n.
\end{align*}$$

(3.4)

If the Hessian, $H_j$, is equal to the Hessian of the Lagrange $\nabla_\xi L(\xi^*, \lambda^*)^{-1}$, where

$$L(\xi, \lambda) = \sum_i \lambda_i c_i(\xi) + \text{Cost}(\xi)$$

(3.5)

and $\lambda_i = 1, 2, \cdots n$, are Lagrange multipliers, and the step size $\alpha_j$ is equal to one, then the rate of convergence is quadratic. On the other hand, if $H_j$ is an approximation of $\nabla_\xi L(\xi^*, \lambda^*)^{-1}$ and $\alpha_j$ is equal to one, then we can get superlinear convergence.

### 3.1.2 Saddle Point Algorithm

In a recent paper [9], the authors present a saddle point algorithm for the determination of macromodel parameters. It is a second order method (quasi-Newton) which uses a modified Broyden's update to approximate the second derivative, and solve the min-max problem:

$$\min_{\xi} \left( \max_{i_w} C(\xi, i_w) \right)$$

(3.6)

where $i_w$ is the set of all transient input waveforms of finite length in time. Note that this is an unconstrained problem, since constraints $c_i(\xi) \leq 0, i = 1, 2, \cdots, n$ are not included. The saddle point algorithm will converge if the initial point is close to a saddle point. On the other hand, care must be taken as to the choice of parameters in the macromodel, because if one of them diverges (i.e. $R \to \infty$) or is set to a value that is difficult to simulate (i.e. $Gm = 0$), then the program will crash. Furthermore conditions are not discussed in [9], that guarantee convergence to a local minimizer.
3.1.3 FSQP — Feasible Sequential Quadratic Programming

The FSQP algorithm for constrained optimization can overcome the difficulties with [9] and is very efficient. The algorithm has excellent convergence properties because it is forced to generate iterates that satisfy all the constraints, and that yield monotonically decreasing objective function values,

\[
\begin{cases}
    c_i(\xi_j) \leq 0, & i = 1, 2, \ldots, n \\
    \text{Cost}(\xi_{j+1}) \leq \text{Cost}(\xi_j)
\end{cases}
\]  

(3.7)

To maintain feasibility, the algorithm uses a convex combination \( d \), of the usual quasi-Newton search direction and an the first order search direction, in such a way that \( d \) converges to the quasi-Newton search direction close to the accumulation point, \( \xi^* \). Hence the algorithm preserves the quasi-Newton character of iterations.

A superlinear rate of convergence to a local minimizer is proven provided that the approximation \( H_j \) of the Hessian of the Lagrangian at \( \xi^* \) satisfies the property

\[
\frac{\|P_j(H_j - \nabla \xi^* \mathcal{L}(\xi^*, \lambda^*))P_j d_j\|}{\|d_j\|} \rightarrow 0, \quad \text{as} \quad j \rightarrow \infty
\]  

(3.8)

where \( \lambda^* \) is the Lagrange multiplier associated with \( \xi^* \) and where

\[
P_j = [I - R_j(R_j^T R_j)^{-1} R_j^T]
\]  

(3.9)

with a full rank \( R_j = [\nabla c_i(\xi_j) | i \in I(\xi^*)] \) where \( I(\xi^*) \) is the set of active constraints at \( \xi^* \). This condition on the approximations \( H_j \) of the Hessian matrix is met for most problems. Furthermore, since the step size converges to one close to the minimizer, \( \xi^* \), two step superlinear convergence can be proven, i.e.

\[
\lim_{j \rightarrow \infty} \frac{||\xi_{j+2} - \xi^*||}{||\xi_j - \xi^*||} = 0.
\]

For further details see [13], [14] and the appendix.
Chapter 4

Implementation

4.1 Implementation

The macromodeling methodology and algorithm outlined in the previous sections has been implemented in a computer program that can handle VLSI circuits. Except in the simplest case, achieving anything close to optimal macromodel parameters would be impossible without the support of numerical optimization. Therefore, in this thesis, we use CONSOLE [10] and SPICE [11] to determine the best macromodel for VLSI circuits.

CONSOLE implements Feasible Sequential Quadratic Programming as an interactive, optimization-based design package to minimize the cost function. Here the cost function is the difference of the input-output behavior between the original circuit and the macromodel circuit for specific performances. SPICE is a powerful tool for evaluating time and frequency domain properties of a broad class of circuit models, and the CONSOLE-SPICE macromodeling system will automatically determine optimal values of macromodel parameters. The computer implementation of the algorithm uses the same interface as SPICE and CONSOLE. The user interface is through two SPICE input files containing a description of the circuit to be modeled and a description of a template for macromodeling. In addition, the user specifies what macromodel parameters the program is allowed to modify in the PDF file (Problem Description File). The PDF file also contains a description of the optimization problem to be solved (i.e., the cost function, upper bound/lower bound constraints on parameters, see the appendix B for more detail). CONSOLE is composed of two main programs: CONVERT and SOLVE. The main framework of our implementation can be seen in Fig. 4.1. The PDF is the input to CONVERT. CONVERT
checks the description for all possible syntax errors and some logic errors and then generates two files. One file is a data file which contains, among other things, the values and names of design parameters, the name of specifications, etc. The other file is an object file. It contains a compiled version of the various specifications (objectives, constraints etc.). Both files are input to SOLVE, together with the object files specified in SPICE and the SPICE-CONSOLE interface. To invoke CONVERT, the user can type

```
convert pdf
```

Some messages appear on the terminal indicating various operations CONVERT is performing. If any error occurs, the user should correct it and run CONVERT again. The next step is to invoke SOLVE and SPICE by typing

```
solved - spice pdf
```

solved-spice is a file containing command scripts. We will discuss how to construct such file in Section 4.3. Again, some messages appear on the terminal indicating various operations SOLVE is performing. If no error occurs, SOLVE prompts

```
< 0 >
```

showing that SOLVE and SPICE are ready to receive commands from the user to perform optimization. SOLVE and SPICE will iterate with the user to obtain an acceptable
solution (using the command sim to enter SPICE and the command back to return to SOLVE).

4.1.1 Computing the cost function

As we mentioned earlier, the cost function is the weighted sum of the integrated difference for each performance. If the user wants to do time simulation and/or ac analysis, he should specify the control card .tran and/or .ac in the circuit description files. Also the user should specify the performances like current and/or voltage in the .Plot or .Print cards. SPICE will first read the files, parse each sentence and then simulate the circuits according to the input control cards.

The CONSOLE-SPICE system will first simulate the original circuit once and save the results in the array called orgi_simouts. During the optimizing iterations, SPICE will just simulate the macromodel circuit and overwrite the array called opti_simouts during each iteration. In the CONSOLE-SPICE interface file conspice.c, we have the subroutine to calculate the cost function as following:

\[
\text{Cost}(\xi) = \sum_{j=1}^{n} \nu_j \int_{0}^{X} \| w^{(j)}(x_j, \xi_j) - w^{(j)}(x_j) \| ^2 \, dx_j
\]  

(4.1)

\(x_j\) is the corresponding sweep variable. The weights, \(\nu_j\), account for the different units of the performance, and can be used to give more importance to one response over another. If we consider both the current and voltage response, \(\nu_j\) can be computed by voltages divided by currents. Here \(\nu_j\) can be either hardcoded by default or set by the user.

4.1.2 Implementation of the Interface

As mentioned above, the file solved-spice contains command scripts. Its content is as follows:

```
#!/bin/csh -f
# Shell script to invoke SOLVE.

# Set the CONSOLE home directory.
set plain = "$home.dir/bin/$mach/solve-$mach$sfp $home.dir/running/ default-$mach.a $argv"
```
if ( $#argv < 1 ) then    ## check number of argument
    echo 'usage: solved-spice pdf'
    exit(1)
else if ( $sim == 'spice' ) then
    echo $argv
    # Dynamically link CONSOLE and SPICE.
    $plain "$home_dir1/conspice.o" "$home_dir1/example/demo/spice-arch"
endif

exit(1)

where we assume that the on-line facility of SPICE resides in the directory
set home_dir1 = /vim/chenjie/bspice
set home_dir = '/vim/chenjie/optimal/console'    ## CONSOLE's home directory

The files conspice.o and spice-sun4 contain the interface between CONSOLE and SPICE
plus a modified version of SPICE respectively. The spice-sun4 can be constructed by the
procedure below.

1. Extract the file RESsetup.o from the library archive file of DEV.a (part of SPICE):
ar x DEV.a RESsetup.o

2. Construct the file spice-sun4.a:
ar r spice-sun4.a RESsetup.o
ranlib spice-sun4.a

3. Delete the file RESsetup.o from the library archive file of DEV.a:
ar d DEV.a RESsetup.o
ranlib DEV.a

4. Construct the file spice-sun4:
ar r spice-sun4.a $(all.o)
ranlib spice-sun4.a
ld -r -S -x -o spice-sun4 -u _i_len -u _xargc -u _xargv
$(SPICE) $(GRAPH) $(MATH) $spice-sun4.a

where all.o $(SPICE) $(GRAPH) $(MATH) are also parts of SPICE, the file conspice.o is
the main interface routine (between CONSOLE and SPICE), and spice-sun4.a is a revised SPICE library.

We use the same input waveform for the macromodel and the original circuit and obtain the difference of their outputs as cost function [see Fig. 4.1]. Multiple objectives can be dealt with, and specifications involving DC, AC and transient simulation can be handled concurrently. To minimize the difference, macromodel parameters are then modified by the program. Our program which uses CONSOLE, is interactive, with short and clearly defined commands providing efficient communication between the program and the user. And SPICE, implemented with X window graphics, provides the user with easy-to-interpret information on the current design. All the functions which SPICE provides can be used and user can use the Plot command to see the difference between the outputs of the circuits being simulated. The initial values of macromodel parameters are templated by the user, without restriction.
Chapter 5

Numerical Results

5.1 Numerical Results

To illustrate the performance of our macromodeling system, we report numerical results for macromodeling the following circuits:

- the input part of an one stage CMOS operational amplifier,
- a full CMOS operational amplifier, and
- a two order lowpass sallen-key filter.

5.1.1 Input Part of an Operational Amplifier

We now report the results of macromodeling the input part of an one stage CMOS operational amplifier containing nine transistors. It is used as a comparator in an A/D converter. A schematic diagram for the input part of a CMOS operational amplifier and its circuit description are shown in Fig. 5.1. A schematic diagram for the macromodel and its circuit description are shown in Fig. 5.2. The PDF file (Problem Description File) is shown in Fig. 5.3. The design parameters of the macromodel which can be changed during optimization iterations are $R_{out}$, $R_{in}$, $C_{in}$, $C_{out}$, $C_t$, $G_m$, and $v_{off}$. This example was also used in [9], where the initial values of the macromodel depended on the static and
Figure 5.1: This is a schematic diagram and input file for op amp to be modeled

* op amp model test
*op amp description
m1 5 1 7 9 nm2 w=22u l=1.6u ad=88p
m2 6 0 7 9 nm2 w=22u l=1.6u ad=88p
m3 3 3 8 8 pm2 w=1.6u l=1.6u ad=16p
m4 4 4 8 8 pm2 w=1.6u l=1.6u ad=16p
m5 3 10 8 8 pm2 w=17u l=1.6u ad=68p
m6 4 10 8 8 pm2 w=17u l=1.6u ad=68p
m7 3 11 5 9 nm2 w=22u l=1.6u ad=88p
m8 4 11 6 9 nm2 w=22u l=1.6u ad=88p
m9 7 12 9 9 nm3 w=38u l=3u ad=152p
ri1 1 20 1.0e5
ci1 1 0 4f
v1 12 0 dc -1.1
v2 10 0 dc 0.75
v3 11 0 dc 1.75
v4 2 0 dc 0
vdd 8 0 dc 2.5
vss 9 0 dc -2.5
vinc 20 0 dc 0 pulse(0 1 0ns 0ns 0ns 10ns 20ns)
.model nm2 nmos vto=0.8 kp=60u cj=0.5e-3
.model nm3 nmos vto=0.8 kp=60u cj=0.5e-3
.model pm2 pmos vto=-0.8 kp=14u cj=0.5e-3
.tran 1ns 40ns
.end
*op amp macro modeling description

Gm 3 0 1 0 2.06e-04
Rt 1 3 1.0e+05
Ct 1 3 4.0e-14
Cin 1 0 4.5e-13
Rin 1 0 1.0e+05
Rout 3 2 6.55e+04
Cout 3 0 6.5e-13
voff 2 0 1.0677
vinc 1 0 dc 0 pulse(0 1 0ns 0ns 10ns 20ns)
.tran 1ns 40ns
.end

Figure 5.2: This is a schematic and input file for macromodel

design_parameter Gm init=2.06e-04 variation=1.0e-5
design_parameter Rin init=1.0e+05 variation=1.0e+03 min=0.0
design_parameter Rout init=6.55e+04 variation=1.0e+03 min=0.0
design_parameter Cin init=4.5e-13 variation=1.0e-15 min=0.0
design_parameter Cout init=6.5e-13 variation=1.0e-15 min=0.0
design_parameter Ct init=4.0e-14 variation=1.0e-16 min=0.0
design_parameter Rt init=1.0e+05 variation=1.0e+03 min=0.0

objective "quadratic"
minimize {
    import Gm,Rin,Rout,Cin,Cout,Rt,Ct
double cost();
    return(AC,Transient)
}

    good_curve = 0.0
    bad_curve = 1.0

Figure 5.3: This is the PDF file for macromodeling

20
<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gm</td>
<td>2.06e-04</td>
<td>3.01e-05</td>
<td>85.4%</td>
</tr>
<tr>
<td>Rin</td>
<td>1.0e+05</td>
<td>1.00e+05</td>
<td>0%</td>
</tr>
<tr>
<td>Rout</td>
<td>6.55e+04</td>
<td>6.47e+04</td>
<td>1.22%</td>
</tr>
<tr>
<td>Cin</td>
<td>4.5e-13</td>
<td>5.0e-14</td>
<td>-88.9%</td>
</tr>
<tr>
<td>Cout</td>
<td>6.5e-13</td>
<td>3.0e-13</td>
<td>53.8%</td>
</tr>
<tr>
<td>Ct</td>
<td>4.0e-14</td>
<td>1.0e-15</td>
<td>-97.5%</td>
</tr>
<tr>
<td>Rt</td>
<td>1.0e+05</td>
<td>1.07e+11</td>
<td>1.07e+8%</td>
</tr>
<tr>
<td>Time</td>
<td>4.8 sec</td>
<td>1.1 sec</td>
<td>-77.1%</td>
</tr>
<tr>
<td>Cost</td>
<td>2.51e+03</td>
<td>3.38e+00</td>
<td>-99.9%</td>
</tr>
<tr>
<td>CPU Time</td>
<td>183 sec</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.1: The performance of macromodeling op amp

![Diagram of op amp](image)

Figure 5.4: Macromodel for the operational amplifier of op amp

dynamic output of the original circuit computed by SPICE. For example, the initial transfer characteristic of the controlled source was linear, with a slope equal to the small-signal transconductance. For our algorithm, we allow the user, who may not have any knowledge of the original circuit, to guess the initial values of macromodel parameters. Macromodel parameters are then modified by the program according to the FSQP algorithm described in section 3. Their final values are printed out upon completion of optimization, so that the macromodel can be used by any simulator understanding SPICE syntax.

The performance of the program is listed in [Table. 5.1]. Macromodeling required 183 seconds on a Sun sparc station IPC. It is also interesting to compare our results to the transient characteristic of the op amp input stage which is quoted in [9] with a piecewise linear voltage control current source \( gm \) before and after optimization. The model used in [9] is shown in Fig. 5.4. The transient voltage response before and after optimization
Figure 5.5: Macromodel and operational amplifier outputs before optimization.

are shown in Fig. 5.5 and Fig. 5.6 respectively. Our macromodel [Fig. 5.2] appears to be more complex, but in fact fewer parameters had to be determined, since the macromodel in [9] uses a nonlinear voltage controlled current source described by 9 parameters. Our results are shown in Figure 5.7 and 5.8. Note that the algorithm in [9] is more successful in matching the transient response of the original circuit. This is because of the use of a nonlinear circuit element. On the other hand, the initial parameters for the macromodel are very close to the optimal solution in [9], while our initial parameters are far away from the optimal solution. This indicates that our algorithm is less sensitive to the choice of initial parameter values, while clearly improvements can be made by including nonlinear devices in our set of macromodeling components.

5.1.2 Operational Amplifier

The second example is a three stage CMOS op amp containing twelve transistors, and which is shown in Fig. 5.9. The macromodel is similar to the one used in the previous case (see Fig. 5.10), but in this case we wished to model the small signal ac response of the op amp, because this component will be used in the filter. The macromodel parameters were again templated, and the program was allowed to change the value of all the elements
Figure 5.6: Macromodel and operational amplifier outputs after optimization.

Figure 5.7: Macromodel an operational amplifier outputs before optimization
Figure 5.8: Macromodel an operational amplifier outputs after optimization

Figure 5.9: Operational Amplifier
in the macromodel. Modeling took approximately 123 seconds of CPU time on SUN sparc station IPC. Comparisons of the macromodel output and the op amp output before and after optimization are shown in Fig. 5.11 and 5.12 respectively; as in the previous case, the improvement in the accuracy of the macromodel is clearly noticeable.

5.1.3 Filter

The third example is a second order lowpass sallen-key filter, containing the operational amplifier in the previous case (see Fig 5.13). This example is more complex than those currently in the literature. Here the macromodel is similar to the one used in the previous case: the input resistor connects the input capacitance in series instead of in parallel, because the low pass filter requires that kind of connection (Fig. 5.14). The same connection is also suitable for a high pass filter, but it is necessary to exchange the position of the input resistor and the input capacitance compared to the low pass case.

In this example the original circuit now includes the macromodel which we obtained for the op amp in previous example. That is to say, we use this macromodel to substitute for the real op amp in the low pass filter. Thus a compact representation of the op amp circuit captures those features that are useful for the AC response, which is what we need to model for the low pass circuit.

The macromodel parameters for the filter were again arbitrarily initialized by the
Figure 5.11: Three Stage Operational Amplifier Output before Optimization

Figure 5.12: Three Stage Operational Amplifier Output after Optimization
Figure 5.13: Two Order Lowpass Sallen-key Filter and Its Operational Amplifier
Figure 5.14: Macromodel for Two Order Lowpass Sallen-key Filter

user. The program was allowed to change all the values of the elements of macromodel. Modeling took approximately 256 seconds of CPU time on a SUN sparc station IPC. The improvement in the accuracy of the macromodel can be evaluated by comparing Fig. 5.15 and Fig. 5.16. The first shows the outputs of the op amp and of the macromodel before optimization, where the input is chosen arbitrarily as an initial guess of the worst case situation by user. The second graph comes from the optimized macromodel, where its output is compared to the low pass filter's output when both are driven by the same input.
Figure 5.15: Two Order Lowpass Sallen-key Filter Output before Optimization

Figure 5.16: Two Order Lowpass Sallen-key Filter Output after Optimization
Chapter 6

Conclusions

6.1 Conclusions

The goal of this thesis was to present a general purpose algorithm for the generation of macromodels suitable for circuit simulation. The algorithm is based exclusively on a comparison of the input-output behavior of the macromodel with that of the circuit to be modeled. Because no restriction is put on the circuits to be modeled, the algorithm can be used to macromodel a very wide class of circuits, especially the more challenging analog case. The algorithm also exceeds those in the literature for the following reasons:

- Not only the transient response of the macromodel, but also other responses, like the DC transfer characteristic or the small signal AC transfer characteristic can be matched.
- Not only the voltage response of the macromodel but also the current response can be matched.
- Not one performance but also other performances, like a voltage and current transient responses and noise can be macromodeled concurrently.
- Not just simple circuits but also complex circuits like a second order low pass filter can be macromodeled. In addition, the user can chose a flexible macromodel topology to approach the behavior of the original functional block.
- The FSQP optimization routine is a reliable and efficient approach to finding the optimal macromodel parameters.
For future work, since we have used SPICE as a general purpose circuit simulation program for nonlinear DC, nonlinear transient and linear AC analyses, macromodel circuits may contain only linear resistors, capacitors, inductors etc. which SPICE can accept. Since macromodeling for large scale circuits provides a tradeoff between cost and accuracy, to reduce the number of devices needed to build macromodels of a large system and to improve the accuracy of our macromodel, we will need nonlinear devices. Specifically, in our further work:

- we also plan to study the construction of macromodels for classes of analog circuits,
- we will extend the set of macromodeling components beyond those in SPICE to be able to macromodel more complex analog circuits with minimal components, and
- we plan to extend our macromodeling technique to build macromodels that can not only match performances, but also performance statistics. Hence it will be possible to determine the worst case performance of a system built with a set of macromodels.
Bibliography


Appendix A

FSQP Algorithm

Algorithm [14]
Data: $\xi_0 \in \mathbb{R}^n$, $H_0 \in \mathbb{R}^{n \times n}$ — symmetric positive definite Hessian matrix
step 0: set $j = 0$.
step 1: Compute search direction
1) solving the quadratic program ,
\[
\begin{cases}
\min_{d_0} \frac{1}{2} \langle d_0, H_0 d_0 \rangle + \langle \nabla \text{Cost}(\xi_j), d_0 \rangle \\
\quad + \langle \nabla c_i(\xi_j), d_0 \rangle \leq 0, \quad i = 1, \ldots, n
\end{cases}
\]
(A.1)
if $d_j^0 = 0$ stop.
2) compute $d_j = (1 - \rho_j)d_k^0 + \rho_k d_k^1$ such that $d_j^1 = d^1(\xi_j), \rho_k = \rho(d_k^0)$.
3) compute $\bar{d}_j$ by solving the quadratic program similar to (eq. A.1)
\[
\begin{cases}
\min_{\bar{d}_j} \frac{1}{2} < d_j + \bar{d}_j, H_j(d_j + \bar{d}_j) > + \langle \nabla \text{Cost}(\xi_j), d_j + \bar{d}_j \rangle \\
\quad + \langle \nabla c_i(\xi_j), \bar{d}_j \rangle + ||d_j||^2 = 0, \quad i = 1, \ldots, n
\end{cases}
\]
(A.2)
step 3: arc search. compute $t_j$, the first number $t$ in the sequence \{1, $\beta, \beta^2, \ldots$\} satisfying:
\[
\begin{cases}
\text{Cost}(\xi_j + td_j + t^2 \bar{d}_j) \leq \text{Cost}(\xi_j) + \alpha t < \nabla \text{Cost}(\xi_j), d_j > \quad \alpha \in (0, \frac{1}{2}) \\
c_i(x_i + td_j + t^2 \bar{d}_j) = 0, \quad i = 1, \ldots, n.
\end{cases}
\]
(A.3)
step 4: Update
1) $\xi_{j+1} = \xi_j + t_j d_j + t^2 \bar{d}_k$
2) update $H_j$ giving $H_{j+1}$ using the BFGS update formula (see [15])
3) $j = j + 1$ and go to step 1.
Appendix B

CONSOLE

CONSOLE is a software tandem including two parts — CONVERT and SOLVE. The following are the main features of the CONSOLE tandem [18]:

- PDF (Problem Description File) is C like file to describe objectives, functional objectives, constraints and functional constraints.
  
  \[
  \begin{align*}
  \text{design parameter :} & \quad \text{initial value of elements to be modified in macromodel} \quad \forall \xi \\
  \text{objective "quadratic" :} & \quad \min_{M \in M}(\text{Cost}(M)) \\
  \text{constraint :} & \quad c_i(M) \\
  \text{good value \& bad value}
  \end{align*}
  \]  
  \text{(B.1)}

- CONVERT check for all possible syntax errors and some logic errors of PDF.
- SOLVE is linear and nonlinear optimization-based design based on FSQP

SOLVE takes an arbitrary number of arguments, but at least one. The first argument must be the name of a PDF File that has been successfully processed by CONVERT. The remaining arguments can be anything that is a valid argument for the UNIX loader \textit{ld}. Before SOLVE is ready to receive commands from the user, it performs four successive operations.

- SOLVE invokes the UNIX loader \textit{ld} to load the compiled version of the PDF together with the simulators SPICE and interface routines.
• SOLVE then reads in the binary data file associated with the PDF.

• SOLVE calls the routine named sim_init to perform an initialization of the simulator SPICE.

• SOLVE calls the routine named sim_pupd to update the current design parameters declared in PDF.
Appendix C

Nonlinear Optimization

C.1 Unconstrained Problems

Let us consider optimization problems of the form

\[
\begin{align*}
\min f(x) \\
\text{subject to } x \in \mathbb{R}^n
\end{align*}
\]  

(C.1)

where \( f \) is a real-valued function.

**Theorem [12]** Assume \( f \in C^2 \) is a function on \( \Omega \). If \( x^* \) is a local minimizer of \( f \) over \( \Omega \), then for all \( d \in \mathbb{R}^n \), we have \( \nabla f(x^*) = 0 \) and \( d^T \nabla^2 f(x^*) d \geq 0 \).

Here we define that \( \nabla^2 f(x^*) \) is the Hessian of function. A necessary condition to have local minimum point is the gradient of the function vanishes and the Hessian is positive semidefinite; and conversely, if at a point the gradient vanishes and the Hessian is positive definite, that point is a local minimizer. If the function is convex, that minimizer is global. Saddle point is one in which the eigenvalues of Hessian matrix have both positive and negative sign.

Actually computing the global minimizer is very computationally involved. So here we only consider iterative algorithms for finding local minimum points, there are two distinct issues: global convergence properties and local convergence properties. The first is concerned with whether starting at an arbitrary point the sequence generated will
converge to a solution. Local convergence properties are a measure of the ultimate speed of convergence, one of such measures is defined as following

**Definition.** Assume the sequence \( \{x_k\}_{k=0}^{\infty} \) converges to \( x^* \). The *order* of convergence of \( \{x_k\} \) is defined as the supremum of the nonnegative numbers \( p \) satisfying

\[
0 \leq \lim_{k \to \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} < \infty
\]

if the sequence \( \{x_k\} \) converges to \( x^* \) in such a way that

\[
\lim_{k \to \infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = \beta < 1
\]

the sequence is said to converge *linearly* to \( x^* \) with *convergence ratio* \( \beta \). The smaller the ratio the faster the convergence rate. The case where \( \beta = 0 \) is referred to as *superlinear convergence*.

Suppose the minimizing point or minimizer of eq. (C.1) is \( x^* \). Many methods which were suggested for minimization were based on searching along the opposite direction of the first derivative, or in more general directions. The basic algorithm is the following :

- determine a descent search direction \( d_k \),
- find step size \( \alpha_k \) to minimize \( f(x_k + \alpha_k d_k) \), and
- set \( x_{k+1} = x_k + \alpha_k d_k \).

However finding \( \alpha_k \) exactly is generally not computationally desirable. The practical and popular criterion for determining the step size \( \alpha_k \) and terminating a line search is the *Armijo’s rule*. The essential idea is that the rule should first guarantee that the selected \( \alpha_k \) is not too large, and next it should not be too small. Normally we begin with the step size \( \alpha_k \) equal to one. Then we decrease the step size according to *the Armijo’s rule* if necessary.

On the other hand, there are several different ways to choose search direction \( d_k = -H_k^{-1} \nabla f(x) \) like first order method, Newton’s method and quasi-Newton method.
C.1.1 First order methods

The first order method, where the Hessian matrix is identity, is called *steepest descent method*. The method is extremely important from a theoretical viewpoint, since it is one of the simplest for which a satisfactory analysis exists. More advanced algorithms are often motivated by an attempt to modify the basic steepest descent technique in such a way that the new algorithm will have superior convergence properties. In first order method, the sequence converges linearly to the minimizer.

C.1.2 Newton's Method

The idea behind Newton's method is that the function $f$ being minimized is approximated locally by a quadratic function, and this approximate function is minimized exactly. Thus near $x_k$ we can approximate $f$ by the truncated Taylor series,

$$f(x) = f(x_k) + \nabla f(x_k)(x - x_k) + \frac{1}{2}(x - x_k)^T H(x_k)(x - x_k).$$  \hspace{1cm} (C.4)

The right-hand side is minimized if $H(x_k) > 0$

$$x_{k+1} = x_k - H(x_k)^{-1} \nabla f(x_k)$$  \hspace{1cm} (C.5)

this equation is the pure form of Newton's method.

**Theorem** Let $f \in C^3$ on $E^n$, and assume that at the local minimum point $x^*$, the Hessian $H(x_k)$ is positive definite. Then if the initial point starts sufficiently close to $x^*$, the sequence generated by Newton's method converges to $x^*$. The order of convergence is at least two.

Although Newton's method has order two convergence, it is rarely used in practice on large problem, since Newton method must be modified to insure global convergence, and evaluation of the Hessian at every point is usually not worth the trouble.

C.1.3 Quasi-Newton Method

The basic motivation behind quasi-Newton methods is to try to obtain the rapid convergence associated with Newton's method without explicitly evaluating the Hessian at every step. This can be accomplished by constructing approximations to the inverse Hessian
or the Hessian based on information gathered during the optimization process. It can be used when only the first derivatives are available, and the first derivatives are generally easier to compute. There are different constructing methods:

- Rank One Correction,
- DFP (Davidon-Fletcher-Powell) Method, and
- BFGS (Broyden-Fletcher-Goldfarb-Shanno) Method.

These methods don’t have quadratic rates of convergence like Newton’s method. But under mild assumption, they have superlinear rate of convergence when step size is one.

### C.2 Constrained Minimization

#### C.2.1 Equality and Inequality Constraints

The constrained optimization is similar to the unconstrained problem (eq. C.1), except the equality constraints $h(x)$ and the inequality constraints $g(x)$ are taken into account.

\[
\begin{align*}
\text{min } & f(x) \\
\text{subject to } & x \in \Omega \\
\Omega & = \{ x \in \mathbb{R}^n : h(x) = 0, \quad g(x) \leq 0 \} 
\end{align*}
\]  

(C.6)

**Theorem [12]** Suppose that $x^*$ is a local minimum of $f$ subject to $h(x) = 0$ and that $x^*$ is a regular point of these constraints. Then there is a $\lambda \in \mathbb{R}^m$ such that

\[
\nabla f(x^*) + \lambda^T \nabla h(x^*) = 0.
\]

where $\lambda$ is Lagrange multiplier. If we denote the tangent plane $M$ as

\[
M = \{ y : \nabla h(x^*) y = 0 \},
\]

then the matrix

\[
\mathcal{L}(x^*) = \nabla^2 f(x^*) + \lambda^T \nabla^2 h(x^*)
\]

is positive semidefinite on $M$, that is, $y^T \mathcal{L}(x^*) y \geq 0$ for all $y \in M$. 

40
Given a minimization problem subject to equality constraints in which all functions are smooth, a necessary condition satisfied at a minimum point is that the gradient of the objective function is orthogonal to the tangent plane of the constraint surface. If the point is regular, then the tangent plane has a simple representation in terms of the gradients of the constraint functions, and the above condition can be expressed in terms of Lagrange multipliers. If the functions have continuous second partial derivatives and Lagrange multipliers exist, then the Hessian of the Lagrangian restricted to the tangent plane plays a role in second-order conditions analogous to that played by the Hessian of the objective function in unconstrained problems. Specifically, the restricted Hessian must be positive semidefinite at a relative minimum point and, conversely, if it is positive definite at a point satisfying the first-order conditions, that point is a strict local minimum point. Inequalities are treated by determining which of them are active at a solution. An active inequality then acts just like an equality, except that its associated Lagrange multiplier can never be negative because of the sensitivity interpretation of the multipliers.

The extension of Newton and quasi-Newton techniques from unconstrained to constrained optimization is called Sequential Quadratic Programming (SQP). Given estimates $x_k$ to the solution of eq. (C.6) and $H_k$ to the Hessian of the corresponding Lagrangian function, the Sequential Quadratic Programming (SQP) iteration consists in first computing a search direction $d_k$ by solving the quadratic program

$$
\begin{aligned}
\min_{d_k} & \frac{1}{2}\langle d_k, H_k d_k \rangle + \langle \nabla f(x_k), d_k \rangle \\
\text{subject to } & g_i(x_k) + \langle \nabla g_i(x_k), d_i \rangle \leq 0, \quad i = 1, 2, \ldots, n. \\
& h_j(x_k) + \langle \nabla h_j(x_k), d_j \rangle = 0, \quad j = 1, 2, \ldots, m.
\end{aligned}
$$

Equation (C.7) has an unique solution if $H_k > 0$, where the Hessian, $H_k$ is equal to the Hessian of the Lagrange $\nabla_{xx}\mathcal{L}(x, \lambda, \mu)$,

$$
\mathcal{L}(x, \lambda, \mu) = f(x) + \sum_i \lambda_i g_i(x) + \sum_j \mu_j h_j(x)
$$

where $\mu_j, \lambda_i$ are the Lagrange multipliers corresponding to equality and inequality respectively, and if the step size $\alpha_k$ is equal to one, then the rate of convergence is quadratic. On the other hand, if $H_k$ is certain approximation to the Hessian of the Lagrange and $\alpha_k$ is equal to one, then we can get superlinear convergence.

### C.2.2 Feasible Sequential Quadratic Programming — FSQP

Extension of quasi-Newton method via SQP may be hampered by infeasibility of the quadratic programs. Feasibility of the successive iterates is very important in real-time
applications, i.e. in circuit simulation, the values of resistors and capacitors should be positive. Also to generate feasible iterates becomes much important, when the objective function is not well defined outside the feasible set.

In order to overcome this difficulty, let us only consider the inequality constrained optimization in eq. (C.7). If the algorithm is forced to generate iterates that all satisfy the constraints, and that yield decreasing objective function values

$$f(x_{k+1}) \leq f(x_k) \quad x_k \in X = \{x \text{ s.t. } g(x_k) \leq 0\}$$

. Then the inequality constrained optimization will always be feasible.

In FSQP [13] algorithm, search direction $d_k = (1 - \rho)d_k^0 + \rho d_k^1$, where $d_k$ consists of two part: $d_k^0$, the search direction used in SQP and $d_k^1$, arbitrary feasible descent direction. To ensure that $d_k$ inherits the quasi-Newton character of $d_k^0$, $\rho$ is forced to go to 0 so that the algorithm can approach the solution $x^*$ fast enough. Also the search direction $d_k$ will be bent i.e., performing a search along an arc $x_k + td_k + t^2 \tilde{d}_k$. This is because, even close to minimizer $x^*$, $x + d_k$ may violate both feasibility and decrease the objective function. $\tilde{d}_k$ will be chosen in such a way that

- in a point close to $x^*$, $x_k + d_k + \tilde{d}_k$ will be feasible and descent,
- $d_k + \tilde{d}_k$ will converge to $d_k$, so as to preserve the SQP character of the iteration.

In FSQP algorithm, in order to determine the step size $\alpha_k$ based on the Armijo's rule,

$$\begin{cases}
  f(x_k + td_k + t^2 \tilde{d}_k) \leq f(x_k) + \alpha t \nabla f(x_k), d_k > \\
  g_i(x_k + td_k + t^2 \tilde{d}_k) \leq 0, \quad i = 1, \ldots, n.
\end{cases} \quad (C.9)$$

t_k is selected as the first number $t$ in the sequence $\{1, \beta, \beta^2, \ldots\}$ during the arc searching.

A superlinear rate of convergence to a local minimizer is proven provided that the approximation $H_k$ of the Hessian of the Lagrangian at $x^*$ satisfies the property

$$\frac{\|P_k(H_k - \nabla^2 \mathcal{L}(x^*, \lambda^*))P_k d_k\|}{\|d_k\|} \to 0, \quad \text{as} \quad k \to \infty \quad (C.10)$$

where $\lambda^*$ is the Lagrange multiplier associated with $x^*$ and where

$$P_k = [I - R_k (R_k^T R_k)^{-1} R_k^T] \quad (C.11)$$
with a full rank $R_k = [\nabla g_i(x_k) \mid i \in I(x^*)]$ where $I(x^*)$ is the set of active constraints at $x^*$. Here $H_k$ is constructed using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update formula with Powell's modification [15]. Furthermore, since the step size converges to one close to the minimizer, $x^*$, two step superlinear convergence can be proven, [13] i.e.

$$\lim_{k \to \infty} \frac{||x_{k+2} - x^*||}{||x_k - x^*||} = 0.$$