Implementation of a simulated annealing algorithm for Matlab

Training performed

Stephane Moins

LITH-ISY-3339-2002

23/08/2002

Implementation of a simulated annealing algorithm for Matlab

Training performed in Electronics systems
Linköping Institute of Technology

Stéphane MOINS

LiTH-ISY-EX-3339-2002

Supervisor: Emil Hajalmarson

Examiner: Kent Palmkvist

Linköping August 23, 2002

Avdelning, Institution
Division, Department

Institutionen för Systemteknik
581 83 LINKÖPING

Datum
Date
2002-08-23

Språk
Language
Svenska/Swedish
X Engelska/English

Rapporttyp
Report category
Licentiatavhandling
X Examensarbete
C-uppsats
D-uppsats

Övrig rapport

ISBN

ISRN LITH-ISY-EX-3339-2002

Serietitel och serienummer **ISSN**
Title of series, numbering

URL för elektronisk version
http://www.ep.liu.se/exjobb/isy/2002/3339/

Titel
Title Implementation of a Simulated Annealing algorithm for Matlab

Författare
Author Stephane Moins

Sammanfattning
Abstract
In this report we describe an adaptive simulated annealing method for sizing the devices in analog circuits. The motivation for use an adaptive simulated annealing method for analog circuit design are to increase the efficiency of the design circuit.
To demonstrate the functionality and the performance of the approach, an operational transconductance amplifier is simulated. The circuit is modeled with symbolic equations that are derived automatically by a simulator.

Nyckelord
Keyword
Simulated annealing, Matlab, circuit optimization, analog circuits,

Abstract

In this report we describe an adaptive simulated annealing method for sizing the devices in analog circuits. The motivation for use an adaptive simulated annealing method for analog circuit design are to increase the efficiency of the design circuit.

To demonstrate the functionality and the performance of the approach, an operational transconductance amplifier is simulated. The circuit is modeled with symbolic equations that are derived automatically by a simulator.

Acknowledgments

I would like to thanks several people at the university for their inspiration and their great effort to explain things clearly and in a simple way for welcoming and during my training period.

It is difficult to overstate my gratitude to my examiner Kent Palmkvist for welcoming me at the Department of Electrical Engineering.

I would like to gratefully acknowledge my supervisor Emil Hajalmarson and Robert Högglund who were always here when I need it and for providing resources and subjects, and offering direction in my work.

Special thank to Peter Johansson, Greger Karlströms for helping me with computer related problems and Thomas Johansson for helping me solve various Unix problems.

I would like also to thank all of my friends at the university for helping me improve my English through our number discussions.

Table of contents

1 Presentation of the assignment

Simulated annealing is an effective and commonly optimization algorithm used to solve non linear optimization problems. At our department a device sizing program for analog integrated circuits is being developed. The optimization routine used in the current version of the program is a gradient-based sequential quadratic programming (SQP). To improve the speed and efficiency of the optimization process the simulated annealing optimization method could be used instead or in conjunction with the existing method.

1.1 Simulated annealing

Simulated annealing (SA) is a Monte Carlo approach for minimizing multivariate functions.

SA is a numerical optimization technique based on the principles of thermodynamics. SA is motivated by an analogy to annealing in solids. The idea of SA comes from a paper published by Metropolis *et al*[1]. in 1953. The algorithm in this paper simulated the cooling of material in a heat bath. This is a process known as annealing.
If you heat a solid past melting point and then cool it, the structural properties of the solid depend on the rate of cooling. If the liquid is cooled slowly enough, large crystals will be formed. However, if the liquid is cooled quickly (quenched) the crystals will contain imperfections.
Metropolis's algorithm simulated the material as a system of particles. The algorithm simulates the cooling process by gradually lowering the temperature of the system until it converges to a steady, *frozen* state.

It helps to visualize the problems presented by a system as a geographical terrain. For example, consider a mountain range, with two "parameters," e.g., along the North-South and East-West directions. We wish to find the lowest valley in this terrain. SA approaches this problem similar to using a bouncing ball that can bounce over mountains from valley to valley. We start at a high "temperature," where the temperature is an SA parameter that mimics the effect of a fast moving particle in a hot object like a hot molten metal, thereby permitting the ball to make

[1] Metropolis, A.Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, *Equation of State Calculations by Fast Computing Machines*, J. of Chem. Physics, 21 (1953) 1087 - 1092

very high bounces and being able to bounce over any mountain to access any valley, given enough bounces. As the temperature is made relatively colder, the ball can not bounce so high, and it also can settle to become trapped in relatively smaller ranges of valleys.

We imagine that our mountain range is aptly described by a “cost function.” We define probability distributions of the two directional parameters, called generating distributions since they generate possible valleys or states we are to explore. We define another distribution, called the acceptance distribution, which depends on the difference of cost functions of the present generated valley we are to explore and the last saved lowest valley. The acceptance distribution decides probabilistically whether to stay in a new lower valley or to bounce out of it. All the generating and acceptance distributions depend on temperatures.

1.1.1 Presentation of the algorithm

Initialization(Current_solution,Temperature)
Calculation of the Current_Cost
LOOP
    New_State
    Calculation of the new_Cost
    IF Δ(Current_cost - New_Cost) $\leq$ 0 THEN
        Current_State= New_State
    ELSE
        IF $\mathrm{Exp}\left(\frac{\text{Current\_cost - New\_Cost}}{\text{Temperature}}\right)$ > Random (0,1)
    THEN
        -- Accept
        Current_State = New_State
    ELSE
        -- Reject

    Decrease the temperature

EXIT When STOP_CRITERION
END LOOP

Figure 1: Simulated annealing algorithm

The algorithm starts from a valid solution and randomly generated new states, for the problem and calculates the associated cost function. Simulation of the annealing process starts at high fictitious temperature. A new state is randomly chosen and the difference in cost function is calculated. If Δ(Current_cost - New_Cost) $\leq$ 0, i.e., the cost is lower, then this new state is accepted. This forces the system toward a state corresponding to a local or a possibly a global minimum. However, most large optimization problems have many local minima and the optimization algorithm is therefore often trapped in a local minimum.

To get out of a local minimum, an increase of the cost function is accepted with a certain probability, i.e. ,if then the new state is accepted. The simulation starts with a high temperature. This makes the left hand side of Equation 1 close to 1. Hence, a new state with a larger cost has a high probability of being accepted.

$$\text{Exp}\left(\frac{\text{Current\_cost - New\_Cost}}{\text{Temperature}}\right) \qquad (1)$$

For example, starting in state *i*, as illustrated in Figure 2, the new state *k1* is accepted, but the new state *k2* is only accepted with a certain probability. The probability of accepting a worse state is high at the beginning and decreases at the temperature decreases. For each temperature, the system must reach an equilibrium i.e. ,a number of new states must be tried before the temperature is reduced typically by 10 %. It can be shown that the algorithm will find, under certain condition, the global minimum and not get stuck in a local minima.

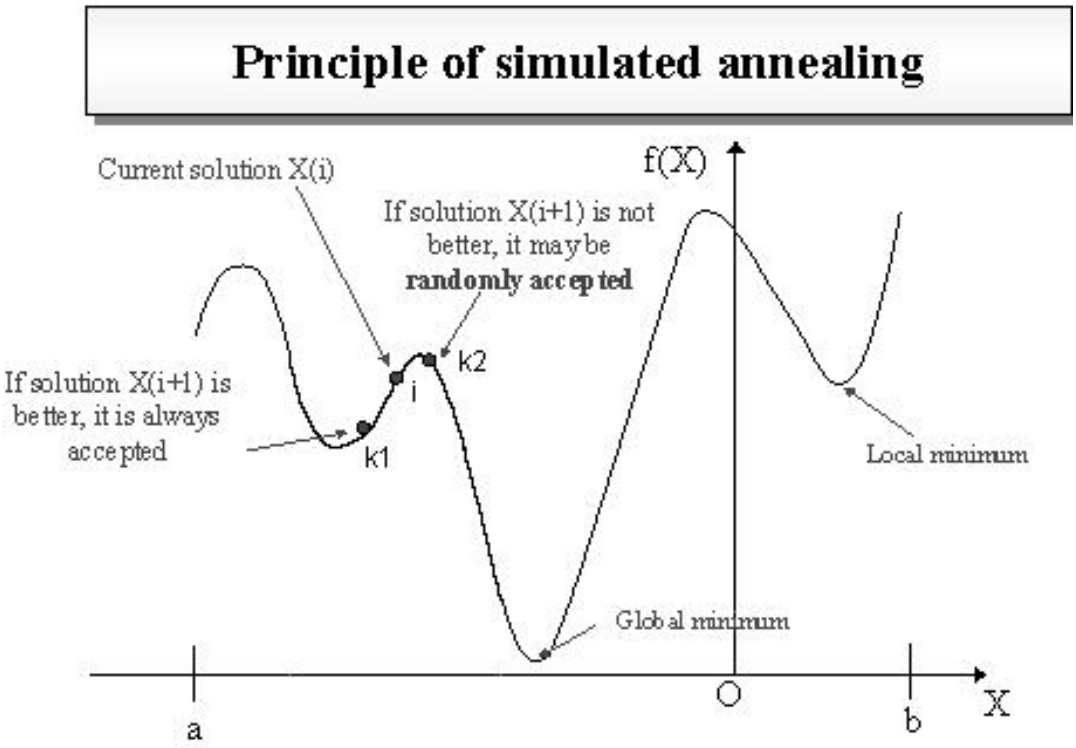

Figure 2: Selection of a new states in simulated annealing

2 ASA and ASAMIN

2.1 Adaptive Simulated Annealing

This method of Adaptive Simulated Annealing[2] (ASA), previously called Very Fast Simulated Annealing (VFSA) only named so in order to distinguish it from the previous method of Fast Annealing (FA).

ASA is a C-language code developed to statistically find the best global fit of a non linear constrained non-convex cost-function over a D-dimensional space. This algorithm permits an annealing schedule for a "temperature" T decreasing exponentially in annealing-time *k*, $T = T_0 \exp(-ck^{\frac{1}{D}})$. The introduction of re-annealing also permits adaptation to change sensitivities in the multi-dimensional parameter-space. This annealing schedule is faster than Cauchy annealing, where $T = \frac{T_0}{k}$, and much faster than Boltzmann annealing, where $T = \frac{T_0}{\ln k}$. ASA has over 100 options to provide robust tuning over many classes of non linear stochastic systems.

One important thing is the constraints. A general formulation for these is

$$\min f(x) \quad \text{subject to} \quad C_i(x) = 0 \quad i \in \varepsilon$$

where the functions *f* and C_i are all smooth and ε is a finite sets of indices. As before, we called f the objective function, or cost function, while C_i , $i \in \varepsilon$, are the equality constraints.

Analogous to the physical process, the temperature is slowly reduced causing the probability of accepting a poor move to decrease with time. The schedule by witch the temperature is reduced is called the cooling schedule and is critical to the success of ASA. Two important parameters governing the cooling schedule are the step size for perturbation and the temperature. The parameter setting suggested by most researchers are step sizes for that allow approximately 80% of the poor moves to be accepted.

[2] ASA package is available at http://ingber.com

2.2 ASAMIN

2.2.1 Introduction

This program is a Matlab gateway to the ASA program. ASAMIN[3] was developed by Shinichi Sakata. The current version of ASAMIN is written for Matlab 5.3 but it works well with the version 6.2 on Windows, Unix and Linux. The most common commands are described and an example is supplied.

ASAMIN provides an interface (mex file) for using ASA. It is possible with ASAMIN to use the ASA program in order to optimize a cost function coded in Matlab language. It also offers a way to change many of ASA's run-time options from within Matlab. It's therefore possible to use ASAMIN with any knowledge about the C language.

2.3 Installation of ASAMIN package

- ***Installation procedure for UNIX***

1) Store the three files provided in the ASA package (asa.c, asa.h, and asa_user.h) and move in the same directory as ASAMIN.

2) Before launching the makefile, open it, and add this line :
" *#CMEX=sw/matlab/6.1/bin/cmex* " below the line " *CC=gcc* ".

3) Now type make. A new file is created (asamin.mexsol). If this file is not create, type into Matlab *mex -setup* and choose the first option. This option is for building the mex file using the gcc compiler .

4) Use the Matlab command addpath in order to add the directory where ASAMIN is installed.
addpath ~/mame_of_your_directory

Otherwise it's possible to find the correct makefile on my account at *steph/asamin/Makefile* or find directly the mex file at *steph/asamin/Unix* for Unix.

[3] ASAMIN package is available at http://www.econ.lsa.edu/~ssakata/software

- ***Installation procedure for LINUX***

For using ASAMIN on Linux store the mex file at *steph/asamin/Linux*

Below is the complete Makefile:

```
#*****************************************************************************
# MATLAB Gateway Routine for Lester Ingber's Adaptive Simulated
#Annealing (ASA)
#
# Copyright (c) 1999-2001 Shinichi Sakata. All Rights Reserved.
#*****************************************************************************
# $Id: Makefile,v 1.23 2002/04/19 15:07:46 ssakata Exp ssakata $
#
CFLAGS = -O4
LDFLAGS = -O4
DEBUGFLAG = -DDEBUG
DEFINES = \
        -DUSER_ACCEPTANCE_TEST=TRUE \
        -DUSER_ASA_OUT=TRUE \
$(DEBUGFLAG)
CC=gcc
#CMEX=sw/matlab/6.1/bin/cmex
#
TARGET = asamin.mex
asamin.mex: asamin.c asamin.h asa.h asa.o
         cmex $(DEFINES) asamin.c asa.o -ll -lm
.c.o:
        $(CC) -c $(CFLAGS) $(DEFINES) -o $@ $<
asa.o: asa.c asa.h asa_user.h
clean:
        rm -rf *.o asatest[12].log
new: clean $(TARGET)
```

- ***Installation procedure for WINDOWS***

1) Store the three files provide by ASA package : asa.c, asa.h, and asa_user.h and move in the same directory as ASAMIN.

2) In Matlab, type :

mex asamin.c asa.c -DUSER_ACCEPTANCE_TEST#TRUE -DUSER_ASA_OUT#TRUE

If your compiler complains that DBL_MIN is undefined, try adding a switch :
"*-DDBL_MIN#2.2250738585072014e-308*".

3) Use the matlab command addpath in order to add the directory where ASAMIN is installed.
addpath ~/mame_of_your_directory

2.3.1 How used ASAMIN

Configuration commands

`asamin ('set', opt_name)`
shows the current value of the option given by a character string; e.g.
asamin ('set', 'seed')

`asamin ('set', opt_name, opt_value)`
set the value opt_value to the option opt_name; e.g.
asamin ('set', 'test_in_cost_func', 0)

`asamin ('reset')`
resets all option values to the hard-coded default values.

Calling ASAMIN

The valid option available with ASAMIN are described in section II.2.4. ASAMIN does not allow setting all options ASA. If necessary the options not accessible from ASAMIN can be attired by modifying the asa_user.h file.

Usage:

[fstar,xstar,grad,hessian,state] = …
asamin ('minimize', func,xinit, xmin, xmax, xtype)

Input argument:

func is the name of the m-file containing the cost function.

xinit is a vector specifying the initial value of the arguments in the cost function.

Each element of the vectors **xmin** and **xmax** specify the lower and the upper bound of the corresponding argument.

The vector **xtype** indicates the desired form of the output argument. If xtype is equal to -1, it means the result is a real; xtype is equal to 1 it means the result is an integer.

Output arguments:

fstar is the value of the objective function at xstar;

xstar is the argument vector at the exit from the ASA routine. If things go well, xstar should be the minimizer of "func";

grad is the gradient of "func" at xstar;

hessian is the Hessian of "func" at xstar;

state is the vector containing the information on the exit state. The first parameter is the exit_code, and the second is the cost flag. Exit_code contains the code for the reason ASA exited, a listing of the various exit codes are given in Table 1.

Exit code	Description
0	Normal exit
1	Parameter temperature too small
2	Cost temperature too small
3	Cost repeating
4	Too many invalid states
5	Immediate exit
7	Invalid user input
8	Invalid cost function
9	Invalid cost function derive
-1	Calloc failed

Table 1. Exit codes

Normal exit means the search has run its normal course.

Parameter temperature too small: A parameter temperature was too small using the set criteria. Often this is an acceptable status code.

Cost temperature too small: The cost temperature was too small using the set criteria. Often this is an acceptable status code.

Cost repeating: The cost function value repeated a number of times using the set criteria. Often this is an acceptable status code.

Too many invalid states: Too many repetitive generated states were invalid using the set criteria. This is helpful when using cost_flag, as discussed above, to include constraints.

Immediate exit: The user has set OPTIONS ->Immediate_Exit to TRUE. valid user input: The user has introduced invalid input.

Invalid cost function: The user returned a value of the cost function to ASA which is not a valid number, or the user returned a value of a parameter no longer within its proper range (excluding cases where the user has set the lower bound equal to the upper bound to remove a parameter from consideration).

Invalid cost function derive: While calculating numerical cost derivatives, a value of the cost function was returned which is not a valid number, or, while calculating numerical cost derivatives, a value of a parameter no longer within its proper range.

Calloc failed: Calloc memory allocation failed in asa.c.

Description :

In cost_function , cost_flag should be set to FALSE (0) if the parameters violate a set of user defined constraints (e.g., as defined by a set of boundary conditions), or TRUE (1) if the parameters represent a valid state. If cost_flag is returned to ASA as FALSE, no acceptance test will be attempted, and a new set of trial parameters will be generated.

- If test_in_cost_func is set to zero, ASAMIN calls the "cost function" with one argument, say x (the real cost function is evaluated at this point). cost_func is expected to return the value of the objective function and cost_flag, the latter of which must be zero if any constraint (if any) is violated; otherwise one.

- When test_in_cost_func is equal to one, ASAMIN calls the "cost function" with three arguments, say, x ,critical_cost_value, and no_test_flag, and expects cost_func to return three scalar values, say, cost_value, cost_flag, and user_acceptance_flag in the following manner:

  1) The function cost_func first checks if x meet the constraints of the minimization problem. If any of the constraints is not met, cost_func is set to zero cost_flag and return it. (user_acceptance_flag and cost_value will not be used by ASAMIN in this case). If all constraints are staisfied, cost_flag is set to one.

2) If ASAMIN calls cost_func with no_test_flag equal to one, cost_func must compute the value of the cost function, set the minimize of the cost function in the cost_value and return it. When no_test_flag is equal to zero, cost_func is expected to judge if the value of the cost function is greater than critical_cost_value. If the value of the cost function is greater than critical_cost_value, user must set the user_acceptance_flag to zero (ASAMIN will not use cost_value in this case). On the other hand, if the value of the cost function is lower than critical_cost_value, cost_func computes the cost function at x, compute cost_value to the minimum of the cost function, and the user_acceptance_flag set to one.

For use this option, change the file, test_cost_func1. The first line was remove by :

function [cost_value, cost_flag, user_acceptance_flag] = ….
test_cost_func2 (x, critical_cost_value, no_test_flag);

2.3.2 Example

To use the example provided by Leaster Inger, two Matlab files are needed. This files should be placed in the same directory as ASAMIN. It is possible to find the two files at *~steph/asamin/*

The file asatest.m is given in appendix 1 and the file cost_function.m is given in appendix 2.

Note that Asatest1.log is the file witch contains the result of the optimization. This file also contains the valid options during the optimization process.

2.3.3 Viewing the optimization results

To see a plot of the number of generated points versus, the value of the cost function, ASA_PIPE, is set to TRUE in the asa_user.h file. A file, named asa_pipe, will be created during the optimization. To use this file in Matlab, edit the file in a text editor and remove all the characters. Save the file and use the following commands from within Matlab:

Load asa_pipe
x = asa_pipe(:,1) ;
y = asa_pipe(:,3) ;
semilogx(x,y)

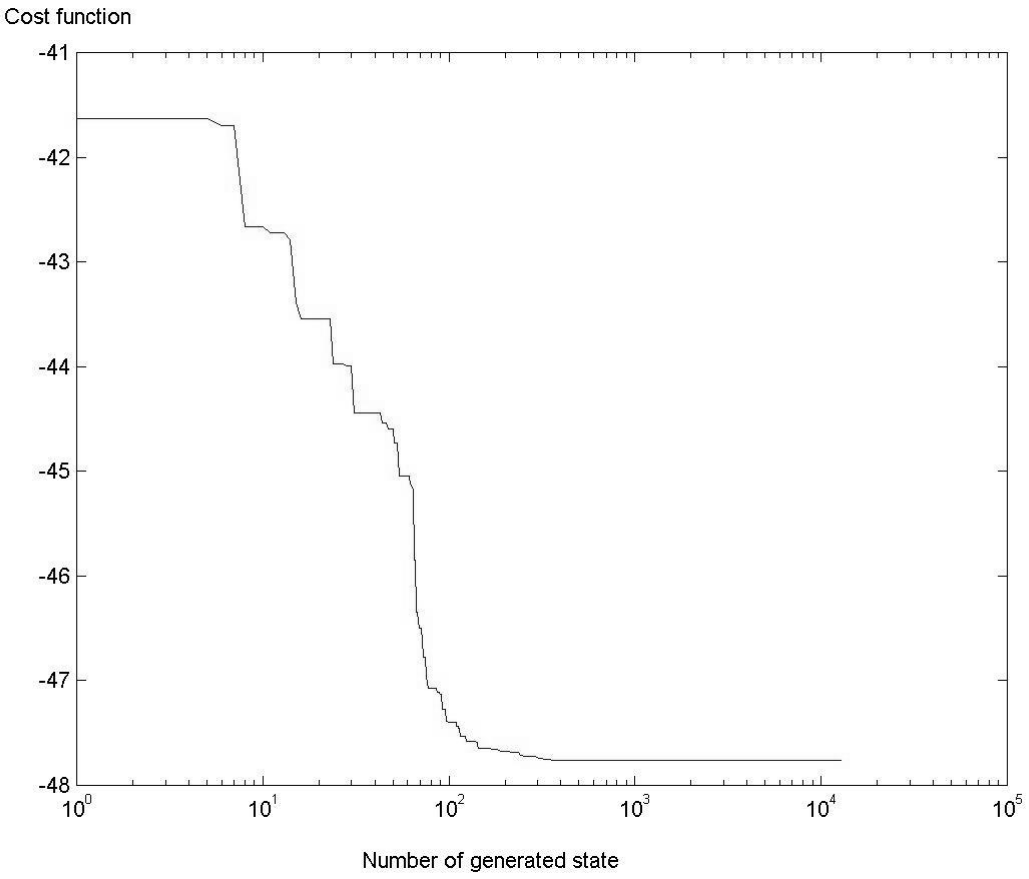

Figure 3: Value of the cost function versus the number of evaluated points.

2.3.4 Description of the ASAMIN's option

In this chapter, a more detailed description of the various options that can be used with ASAMIN, is given. The valid options for ASAMIN are:

- rand_seed
- use_rejected_cost
- cost_parameter_scale
- temperature_anneal_scale
- include_integer_parameters
- user_initial_parameters
- limit_acceptances
- sequential_parameters
- limit_generated
- initial_parameter_temperature
- limit_invalid
- acceptance_frequency_modulus
- accepted_to_generated_ratio
- generated_frequency_modulus
- cost_precision
- reanneal_cost
- maximum_cost_repeat
- reanneal_parameters
- number_cost_samples
- delta_x
- temperature_ratio_scale

Rand_seed is the seed of the random number generation in ASA.

Use_rejected_cost The default value of option use_rejected_cost is zero. If use_rejected_cost is set to one, the ASA routine uses the current cost value to compute certain indices, even if the current state is rejected by the user cost function, provided that the current cost value is lower than the cost value of the past best state.

Limit_acceptances[10000] is the maximum number of states accepted before quitting. If Limit_Acceptances is set to zero, then no limit is observed.

Limit_Generated[99999] is the maximum number of states generated before quitting . If Limit_Generated is set to 0, then no limit is observed.

Limit_invalid[1000] sets limit of repetitive invalid generated states, e.g., when using this method to include constraints. This also can be useful to quickly exit ASA if this is requested by your cost function: Setting the value of Limit_Invalid_Generated_States to 0, will exit at the next calculation of the cost function. For example, to exit ASA at a specific number of generated points.

Accepted_to_generated_ratio[1.0E-6]. It' is the least ratio of accepted to generated states. All the templates in ASA have been set equal to 1.0E-4 to use Accepted_To_Generated_Ratio and to illustrate the way, these options can be changed.

Cost_precision[1.0E-18] sets the precision required of the cost function if exiting because of reaching Maximum_Cost_Repeat, which is effective as long as Maximum_Cost_Repeat > 0.

Maximum_cost_repeat[5] is the maximum number of times that the cost function repeats itself, within limits set by Cost_Precision, before quitting. This test is performed only when Acceptance_Frequency_Modulus or Generated_Frequency_Modulus is invoked, or when the ratio of accepted to generated points is less than Accepted_To_Generated_Ratio, in order to help prevent exiting prematurely in a local minimum. If Maximum_Cost_Repeat is 0, this test is bypassed.

Number_cost_samples[5], when Number_Cost_Samples > 0, the initial cost temperature is calculated as the average of the absolute values of Number_Cost_Samples sampled cost functions. When Number_Cost_Samples < -1, the initial cost temperature is calculated as the deviation over a sample of Number_Cost_Samples number of cost functions, i.e., the square-root of the difference of the second moment and the square of the first moment, normalized by the ratio of Number_Cost_Samples to Number_Cost_Samples - 1.

Temperature_ratio_scale[1.0E-5] this scale is a guide to the expected cost temperature of convergence within a small range of the global minimum.

Cost_parameter_scale[1.0] this is the ratio of cost_parameter temperature annealing scales. Cost_Parameter_Scale_Ratio is a very influential Program Option to determine the scale of annealing of the cost function.

Temperature_anneal_scale[100.0] this scale is a guide to achieve the expected cost temperature sought by Temperature_Ratio_Scale within the limits expected by Limit_Acceptances.

Include_integer_parameters[FALSE] if Include_Integer_Parameters is TRUE, include integer parameters in derivative and reannealing calculations, except those with INTEGER_TYPE (2).

Initial_parameter_temperature[1.0] the initial temperature for all parameters

Acceptance_frequency_modulus[100] the frequency of testing for periodic testing and reannealing, dependent on the number of accepted states. If Acceptance_Frequency_Modulus is set to 0, then this test is not performed.

Generated_frequency_modulus[10000]. The frequency of testing for periodic testing and reannealing, dependent on the number of generated states. If Generated_Frequency_Modulus is set to 0, then this test is not performed.

Reanneal_cost [1] If Reanneal_Cost > 1, then the reannealed initial cost temperature is calculated as the deviation over a sample of Reanneal_Cost number of cost functions. If Reanneal_Cost < -1, then the cost index is reset to 1, and the initial and current cost temperatures are calculated as the deviation over a sample of Reanneal_Cost number of cost functions. A value of Reanneal_Cost set to FALSE=0 bypasses reannealing of the cost temperature. This might be done for systems where such reannealing is not useful.

Reanneal_parameters[TRUE] this permits reannealing of the parameter temperatures to be part of the fitting process. This might have to be set to FALSE for systems with very large numbers of parameters just to decrease the number of function calls.

Delta_x[0.001] is the fractional increment of parameters used to take numerical derivatives when calculation tangent for reannealing.

3 Optimization process

3.1 Introduction

In this program an optimization based approach is used to find the component values in an analog integrated circuit. This can be done by formulating the process of finding device sizes as a mathematical optimization problem with bounding conditions, constraints and a cost function. The specification and topology of the circuit is used to formulate a set of constraints and perform metrics. These metrics can in turn be used to form a cost function.

In this case, the aim is to compare the currently used optimization method with the adaptive simulated annealing.

3.2 Cost function and constraints

The form of the cost function have a major impact on the optimization problem. In the gradient based optimization method currently used, the cost function and the constraints are in two separated files, independent of each other. In the simulated annealing case, it's only a cost function available and thus the constraints must be included in the cost function.

To find a suitable DC operation point each transistor is modeled as a black box. Given the node voltage V_G, V_S, V_D and V_B along with the width and length of the transistor the drain current (I_D), the threshold voltage (V_{TH}) and the saturation voltage (V_{DSAT}) can be compute. Furthermore, Kirchhoff's current law has to be fulfilled in every node.

Starting from an initial guess of the voltage in all node, the optimization routine is used to compute each current in all circuits branches. Looking at figure 4, we see that the current going out of M1 must be equal to the current going into M2. The cost function is therefore to minimize the difference between those currents by adjusting the width of the transistors, the node voltages, and the available bias currents. Applying this to all transistors in a circuit will result in a DC solution with all transistors working in their desired operation region.

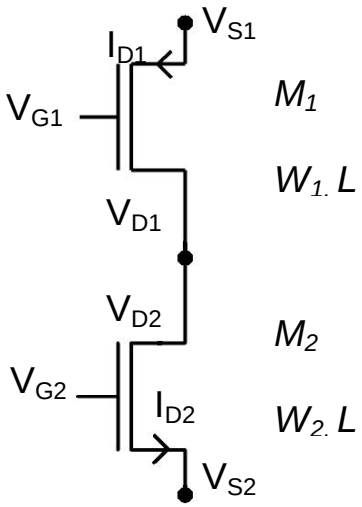

Figure 4: Example

If the DC constraints are not met, a penalty is generated. This penalty is added to the cost function. The magnitude of the penalty for violating constraints is much larger then the performance metric to optimize. In this way, the optimizer will try to satisfy the constraints before trying any of the performance metrics. The name of these function is the weight function, where the weight function G is given by

$$G(X)=\begin{cases} e^X \text{ if } X>1 & \text{penalty at } e^X \\ 1 \text{ if } X<1 & \text{No penalty} \end{cases}$$

where X is defined as the current value.

The DC constraints are that all transistors must operate in the saturation region. To guarantee this, the following well-known relationship must be met for each transistor.

$V_{GS}>V_T$
$V_{GS} - V_T = V_{eff}$

For each node current, if the, Kirchhoff's current law is not met, for example, on the figure 4, if the both current of M1 and M2 are not the same, we compute a penalty too. The best solution is when for all current nodes, the sum of the penalty is equal to zero.

3.3 Presentation of the circuit design

The circuit used to test the simulated annealing optimization is an operation transconductance amplifier.

The SQP gradient-based optimization, used many different Matlab files for find each computation performance metrics. In order to reduce the computation time, all parameters are remove. In the first step, the aim is to improve the power dissipation given by the performance metric.

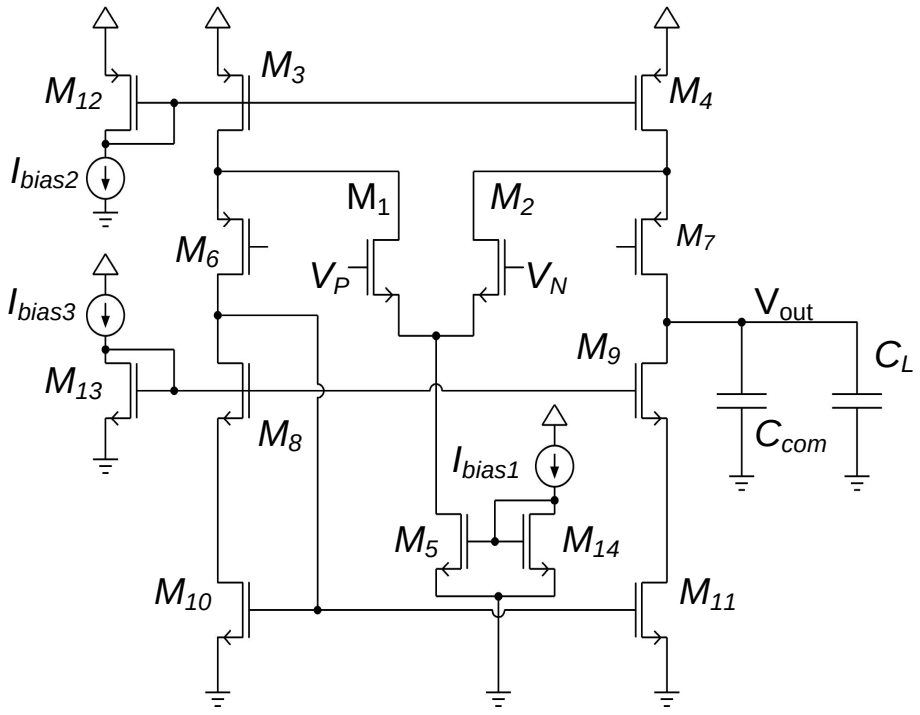

Figure 5: Design of the operation transconductance amplifier.

3.4 Algorithms

In this section, the functionality of the program is explained in a more detailed way.

3.4.1 Optimization files

The flow graph of the main program is show in figure X. It contains all system specifications and the process initialisation to call ASAMIN.

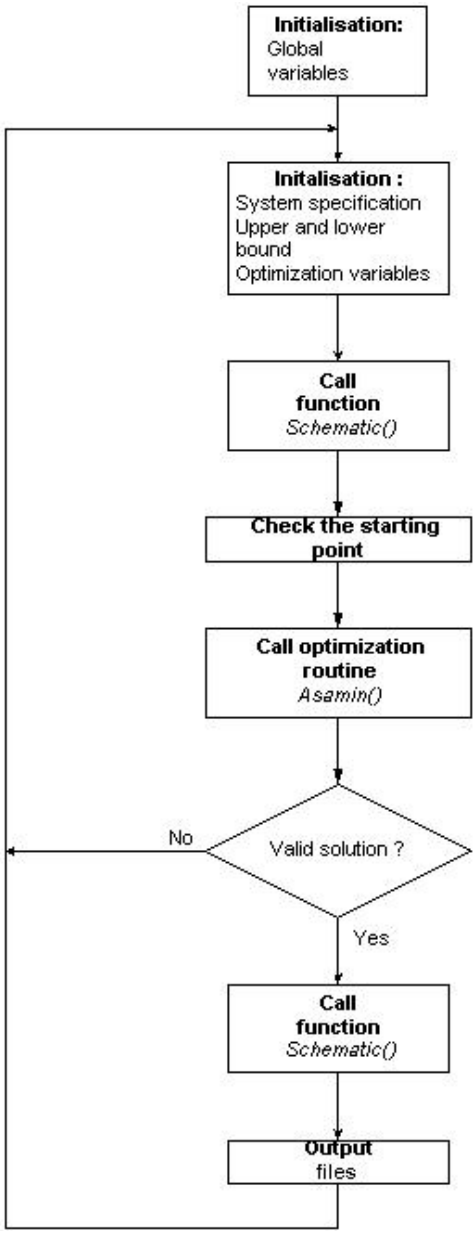

Figure 6: Algorithm of twostage.m

Initialisation :

There are three different types of initializations:

- The first one are the parameters of system specification of the OTA.
- The second is the upper and lower bound on optimization variables.
- The third one are the initial value of the optimization variables such as the size of the transistors.

Schematic () :

This routine contains information on the connections between different components, the transistor width and the transistor length and also the transistor type. The transistor length is set to 0,7 μm. This routine initialise each node voltage too.

Check the starting point :

This block calls the routine *dcop_start()* and calculates the drain current (I_D), threshold voltage (V_T) and the value of the saturation voltage (V_{DSAT}). The value of each optimization variables is randomly chosen. After the values have been computed, a function checks the result in order to verify that the current trough each transistor is positive (i.e., coming from grain to source for a NMOS). It's important to find a good starting point because the result of the optimization process depends it.

Call optimization routine :

This part of the program initializes the ASAMIN routine and calls the ASAMIN function. Many different starting points could be used in order to find a good solution. The number of invalid state generated by the ASAMIN, the maximum cost repeat and the acceptance limit are set-up at 500 times. The cost precision is equal to 1E-6.

For more information about the parameters, see the section II.2.6

Valid solution :

This is a test to see if the result is correct, i.e., the optimization was successful. In fact the program tests if the result of *state()* (Output function given by ASAMIN after each optimisation) is correct. If the test is correct then it continues else it returns at the initialisation to try a different starting point.
For the explanation about the *state()* output, see the section II.2.3, output arguments.

Schematic () :

This function is called again to calculate the resulting performance of the optimization process in order to prepare the output files.

Output files :

Two files are created for each optimization. The first one is name values#.txt and it contains the parameters of each transistor like the transistor width, bias currents, bias voltage and other component values. A new value-file is created for each new solution found.

The second file is named result.txt and contains the value of the performance metrics. The file also contains information on how the optimization went i.e., the *state()* flag.

This file contains the cost function.

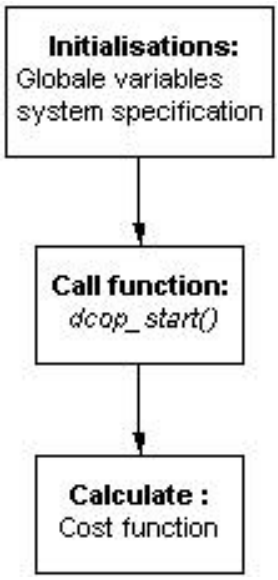

Figure 7: Algorithm of twostage_functions.m

Initialisation :

Initialisation of the system specification for the OTA.

Dcop_start() :

This routine calculates the new values of the drain current (I_D), the threshold voltage (V_T) and the saturation voltage(V_{DSAT}).

Cost function :

Equation (2) is an example of a partial cost function.

$$C(4) = log(1+abs(+T(3,ID)-T(6,ID)-T(1,ID))) \quad (2)$$

After have computing all partial cost functions, the program computes and returns the sum of all of them. This result (cost function) indicates the performance of the optimization. The closer to zero the value of the cost function is the better solution.

3.5 Cadence

When the simulation is finish using Matlab, and the best solution is chosen (the best set of parameters), a post verification can be made using the Spectre simulator in Cadence.

4 Future work

The way to continue this project is to improve the quality of the solutions found by the adaptive simulated annealing. The best result by simulated annealing method is an error close to 10^{-4} and to get a good solution, the error must be reduced to about 10^{-7}

Secondly, all performance metrics must be included during the optimization phase. At present only a few of the possible performance metrics have been used during computation to add all computation parameters like CMRR, DC gain.

5 Conclusion

This training period was very interesting from all point of view. The application that I have worked is useful in the analog circuit design and it's very interesting for me to have gained some knowledge in this field. All staff at the department were very accommodating and so it was very easy to work together. To finished my subject it miss me more knowledge on the circuit design. Moreover at the beginning of my training period the English was hard to cope with and I didn't understand the details given by my supervisor. This foreigner training period gave me a desire for remaking this experiment in the future.

References

“Simulated Annealing: Theory and Application” by P.J.M. van Laarhoven and E.H.L. Aarts, (1987) ISBN 90-277-2513-6

“A Computer-Aided design and Synthesis Environment” by Geert Van der Plas, Georges Gielen, Willy Sansen (2002), ISBN 0-7923-7697-8

“Numerical Optimization” by Jorcedal and Stephen J.Wright, (1999) ISBN 0-387-98793-2

“Optimization based device sizing in analog circuit design” by R. Hägglund, E.Hjalmarson and L. Wanhammar

“*Symbolic analysis for automated design of analog integrated circuits*” by Georges Gielsen (1991), ISBN 0-7923-9161-6

“*A synthesis environment for CMOS analog integrated circuits*” IEEE transaction on computer aided design of integrated circuits and systems, vol. 20 No 9 Sep. 2001

ASA package by Lester Ingber at http://ingber.com

ASAMIN package by Shinichi Sakata at
http://www.econ.lsa.edu/~ssakata/software

6 Appendix 1 : asatest.m

```
% Initiating the optimization parameters.
xinit = [0.040687;0.1481;0.7825;0.00134;0.48528;0.00068;0.18202];
xl=0.00005 * ones(7,1); //lower bound matrix 7*1
xu=1 * ones(7,1); //upper bound matrix of 7*1
xt=-1 * ones(7,1); //real argument
% Set the random seed to use during the optimization...
asamin('set','rand_seed',696969);
asamin('set','asa_out_file','asatest1.log'); //output file
asamin('set','test_in_cost_func',0);
[fstar, xstar, grad, hessian, state] = ...
asamin('minimize', 'cost_function', xinit, xl, xu, xt);
```

7 Appendix 2 : Cost_function.m

```
function [cost_value, cost_flag] = test_cost_func1 (x)
%initial conditions
c(1)= - 6.089;
c(2)= - 17.164;
c(3)= - 34.054;
c(4)= - 5.914;
c(5)= - 24.721;
c(6)= - 14.986;
c(7)= - 24.1;
c(8)= - 10.708;
c(9)= - 26.662;
c(10)= - 22.179 ;
y(1)= x(1);
y(2)= x(2);
y(3)= x(3);
y(4)= x(4);
y(5)= x(5);
y(6)= x(6);
y(8)= x(7);
y(10)= -(y(1)+ 2*y(2)+2*y(3)+ y(6)-2);
y(7)= -(y(4)+ 2*y(5)+y(6)-1);
y(9)= -0.5 *(y(3)+ y(7)+y(8)+ y(10)-1);
if(y(7)<0.000001 |y(8)<0.000001|y(10)<0.000001)
    cost_flag=0;
    cost_value=0;
    break
end
res=0;
sum=0;
for i=1:10
    sum = sum + y(i);
end
for i=1:10
    res = res + y(i)*(c(i)+log(y(i)/sum));
end
cost_value = res;
cost_flag=1;
```

Copyright :

På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

Linköping, 23 th August 2002