



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER SYSTEMS

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

EVOLUTIONARY ANALOG AMPLIFIER OPTIMISATION

EVOLUČNÍ OPTIMALIZACE ANALOGOVÝCH ZESILOVAČŮ

BACHELOR'S THESIS

BAKALÁŘSKÁ PRÁCE

AUTHOR

AUTOR PRÁCE

MAREK BIELIK

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. MICHAL BIDLO, Ph.D.

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů

Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Bielik Marek**

Obor: Informační technologie

Téma: **Evoluční optimalizace analogových zesilovačů**
Evolutionary Analogue Amplifier Optimisation

Kategorie: Umělá inteligence

Pokyny:

1. Perform a study of selected types of analogue amplifiers.
2. Take up with possibilities of electronic circuit simulation using SPICE.
3. Take up with basic principles of optimisation using evolutionary algorithms.
4. Propose a solution for the optimisation of selected amplifier circuits by means of evolutionary algorithm.
5. Perform several sets of experiments in order to demonstrate performance and quality of the proposed method.
6. Evaluate the results obtained and discuss potential possibilities for the future work.

Literatura:

- According to the recommendation of the project supervisor.

Pro udělení zápočtu za první semestr je požadováno:

- Fulfilling steps 1 to 3 of the assignment, demonstration of a prototype system from step 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese

<http://www.fit.vutbr.cz/info/szz/>

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

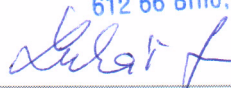
Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bidlo Michal, Ing., Ph.D., UPSY FIT VUT**

Datum zadání: 1. listopadu 2016

Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
602 00 Brno, Božetěchova 2



prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstract

This thesis demonstrates the capabilities of the evolutionary algorithms, namely the evolution strategies, in the domain of the analog amplifiers design. The ngSPICE simulator is integrated into the implementation and it is used for the evaluation of the optimized solutions. The thesis contains various evaluation methods of the amplifiers and it also contains experiments and their results which were used for the determination of the most optimal parameters for evolution strategies. The goal was to optimize the values of components of the single and two stage common emitter amplifiers. The result is a tool that provides the desing of amplifiers with an arbitrary gain, which is within the bounds of the circuit's possibilities, without using any mathematical apparatus.

Abstrakt

Táto práca demonštruje možnosti využitia evolučných algoritmov, konkrétne evolučných stratégií, v doméne dizajnu analógových zosilňovačov. Do implementácie je zahrnutý ngSPICE simulátor, ktorý je použitý na vyhodnotenie optimalizovaných riešení a v práci je navrhnutých niekoľko vyhodnocovacích metód. Práca tiež zahŕňa experimenty a ich výsledky, ktoré boli použité na určenie najvodnejších parametrov evolučných stratégií. Cieľom bolo optimalizovať hodnoty súčiastok jedno a dvoj stupňových zosilňovačov s bipolárnymi tranzistormi v zapojení so spoločným emitorom. Výsledkom je nástroj umožňujúci návrh zosilňovačov s ľubovoľným zosilnením v rámci možností daného obvodu bez použitia akéhokoľvek matematického aparátu.

Keywords

artificial intelligence, evolutionary computation, evolution strategies, ngSPICE, SPICE, RInside, analog amplifier, fitness function, optimization

Klíčová slova

umelá inteligencia, evolučný algoritmus, evolučné stratégie, ngSPICE, SPICE, RInside, analógový zosilňovač, fitness funkcia, optimalizácia

Reference

BIELIK, Marek. *Evolutionary Analog Amplifier Optimisation*. Brno, 2017. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Bidlo Michal.

Evolutionary Analog Amplifier Optimisation

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Ing. Michal Bidlo, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....

Marek Bielik

May 8, 2017

Acknowledgements

I want to thank my supervisor for providing valuable knowledge, my family for support and especially Georgina for giving it all sense.

Contents

1	Preface	2
2	Evolutionary algorithms	3
2.1	Optimization problem	3
2.2	Evolution strategies	4
2.3	Selection	5
2.4	Mutation	6
2.4.1	Mutation with one step size	6
2.4.2	Mutation with n step sizes	7
3	Analog amplifiers	8
3.1	Single stage common emitter amplifier	8
3.2	Two stage amplifier	9
3.3	ngSPICE	10
4	Evaluation of amplifiers	13
4.1	Best match with the reference solution	13
4.2	Ideal sine wave	15
4.3	Maximal amplitude	16
4.4	Waveform symmetry	16
5	Experiments	19
5.1	Initial stages of experiments	19
5.2	Evolution parameters	20
5.3	Population size, selection type and selective pressure	21
5.4	The typical course of the evolution	22
5.5	Single stage amplifier optimization	24
5.6	Two stage amplifier optimization	25
6	Conclusion	27
	Bibliography	29
	Appendices	30
A	User interface	31
B	CD content	34

Chapter 1

Preface

The standard way to design an amplifier is to calculate the values of its components by mathematical equations. The goal of this thesis is to automate this process and make the computer determine the values without the equations.

This task can be also seen as an optimization problem where we look for the most optimal solution among many others.

Chapter 2

Evolutionary algorithms

‘Owing to this struggle for life, variations, however slight and from whatever cause proceeding, if they be in any degree profitable to the individuals of a species, in their infinitely complex relations to other organic beings and to their physical conditions of life, will tend to the preservation of such individuals, and will generally be inherited by the offspring. The offspring, also, will thus have a better chance of surviving, for, of the many individuals of any species which are periodically born, but a small number can survive. I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection.’
Darwin, 1859 [3, p.40].

In biological evolution, species are selected depending on their relative success in surviving and reproducing in the environment and mutations play a crucial role in this process as they are the key to the adaptation of the species [1]. This concept has been used as a metaphor in computer science and it inspired the formation of a whole new area of artificial intelligence called Evolutionary Computation. This area mainly deals with an *optimization problem* which we discuss in the following section.

2.1 Optimization problem

Consider the function $f : A \rightarrow \mathbb{R}$ from a set A to the real numbers and we are looking for an element x_0 in A such that $f(x_0) \leq f(x)$ or $f(x_0) \geq f(x)$ for all x in A (we are looking for either the global minimum or maximum of the function f). The domain A of f is called the *search space* and the elements of A are called *candidate solutions*. The function f can be called variously (an *objective function*, a *loss function*), in this text we will call the function f a *fitness function*. A candidate solution that gives us the global maximum or minimum of the fitness function is called the *optimal solution*. The search space A is usually in the form of n -dimensional Euclidean space \mathbb{R}^n and the function f is also n -dimensional [2].

A simple approach to find the optimal solution would be to scour the whole search space and try every candidate solution. This approach will always give us the optimal solution, but the search space can be so vast, that we are not able to try every candidate solution in a reasonable time.

A more sophisticated approach would be to use the technique presented in algorithm 1. The *population* $P \subseteq A$ would be a subset of the search space A and every *individual* of the population would represent one element of A . In the context of evolutionary algorithms, the individuals can be also called *chromosomes*. Every loop in the algorithm represents

one *generation* of individuals. This approach helps us to avoid those elements in the search space which wouldn't provide any good solution, but on the other hand, finding the optimal solution is not guaranteed.

From the biological point of view, this algorithm can be seen as the driving force behind evolution and from the mathematical point of view, it can be seen as a stochastic, derivative-free numerical method for finding global extrema of functions that are too hard or impossible to find analytically.

Biology uses this approach to create well adapted beings and humans can use this method to find optimal solutions to problems that are difficult to solve analytically.

Algorithm 1 Evolutionary algorithm [1]

- 1: Create a *population* P of randomly selected candidate solutions;
 - 2: **repeat**
 - 3: *(Selection)* Select the appropriate individuals (parents) for breeding from P;
 - 4: *(Mutation)* Generate new individuals (offspring) from the parents;
 - 5: *(Reproduction)* Replace some or all individuals in P with the new individuals;
 - 6: **until** terminating condition
-

2.2 Evolution strategies

Evolution strategies (ES) is a family of stochastic optimization algorithms which belongs to the general class of evolutionary algorithms. It was created by Rechenberg and Schwefel in the early 60s and attracted a lot of attention due to its strong capability to solve real-valued engineering problems. ES typically uses a real-valued representation of chromosomes and the evolutionary process relies primarily on selection and mutation. ES also often implements *self-adaptation* of the parameters used for mutating the parents in the population during the evolution, which means that these parameters coevolve with the individuals. This feature is natural in evolution strategies and other evolutionary algorithms have adopted it as well over the last years [1, 5].

Algorithm 2 is a basic non-self-adaptive form of ES. Chromosomes in this algorithm are in the form of vectors of real numbers $x = (x_1, \dots, x_n) \in \mathbb{R}^n$. Every vector represents one solution of the search space in our problem domain and the goal of the algorithm is to find such vector that produces global extremum of our fitness function.

The terminating condition of the algorithm can have various forms which can be combined together, for example:

- Wait until the algorithm finds a chromosome with such a fitness function value that meets our requirements.
- Limit the total number of generations.
- Track the best chromosome in every generation and stop the evolution if the fitness doesn't change after a certain number of generations.

The last two approaches are suitable when the algorithm gets stuck in a local optimum and doesn't leave it after a significant amount of time.

This form of the algorithm is also implemented in this thesis with various extensions in

order to produce better results. These extensions are described and discussed in the rest of this chapter.

Algorithm 2 Non-self-adaptive Evolution Strategies $(\mu + \lambda)$ [1]

- 1: Randomly create an initial population $\{\vec{x}^1, \dots, \vec{x}^\mu\}$ of parent vectors, where each vector \vec{x}^i is of the form $\vec{x}^i = (x_1^i, \dots, x_n^i)$, $i = 1, \dots, \mu$;
 - 2: Evaluate the fitness function of each chromosome;
 - 3: **repeat**
 - 4: **repeat**
 - 5: Randomly select a parent from $\{\vec{x}^i : i = 1, \dots, \mu\}$;
 - 6: Create a child vector by applying a mutation operator to the parent;
 - 7: **until** λ children are generated
 - 8: Rank the $\mu + \lambda$ chromosomes (children and parents) from the best to worst;
 - 9: Select the best μ of these chromosomes to continue into the next generation;
 - 10: **until** terminating condition
-

2.3 Selection

There are two main parameters in evolutionary strategies that are used to describe the type of the selection process:

1. **Parameter μ** specifies the number of chromosomes (parents) that are selected for reproduction.
2. **Parameter λ** specifies the number of children that are produced in every generation.

There are also two types of selection:

1. **$(\mu + \lambda)$ -ES** selection scheme,
2. **(μ, λ) -ES** selection scheme.

In the first scheme, denoted by $(\mu + \lambda)$ -ES, both parents from the previous generation and newly produced children are mixed together and compete for survival in every generation where only the best μ chromosomes (parents for the next generation) are selected and get the chance to be reproduced. This scheme implements so called *elitism* because if there is a chromosome with a very good fitness function, it will survive for many generations and it can be replaced only if a better individual occurs. This approach has a higher natural tendency to get stuck in local optima than the second scheme.

In the second type of selection, denoted (μ, λ) -ES, the parents are selected only from the set of λ children, so every chromosome lives only in one generation. Even a chromosome with a very good fitness function is discarded and replaced by a child with potentially worse fitness, so the convergence is not as strict as in the previous scheme, but it is easier to move away from a local optimum for this method [1, 5].

Both schemes are implemented in this thesis with various values of μ and λ parameters ($\mu \ll \lambda$). Schemes such as $(1 + 1)$ -ES and $(1, \lambda)$ -ES are also possible ($(1 + 1)$ -ES was mainly studied when evolution strategies were invented), but they show worse convergence properties as shown in section 5.3.

2.4 Mutation

Chromosomes in evolution strategies are represented as vectors of real values, formula 2.1.

$$\vec{x} = (x_1, \dots, x_n) \quad (2.1)$$

These values correspond to the solution variables. Mutations can be performed in various ways, the simplest form is shown in formula 2.3 where m is a real number taken from a normal (Gaussian) distribution (formula 2.2) with the mean in 0 and the standard deviation σ is chosen by the user. The parameter σ represents the mutation step size and its value is crucial since it determines the effect of the mutation operator. The value should be kept relatively small to the problem domain in order not to perform large mutations too often. The parent chromosome is denoted by $\vec{x}(t)$ and $\vec{x}(t+1)$ denotes a child. It is also important to keep the values (x_1, \dots, x_n) of the candidate solution in their domain interval and return it back every time the mutation shifts it out of the interval.

$$m = N(0, \sigma) \quad (2.2)$$

$$\vec{x}(t+1) = \vec{x}(t) + m \quad (2.3)$$

This approach ignores two important facts:

1. When the algorithm is close to the global optimum, it is appropriate to perform only subtle changes to the chromosomes so that the algorithm will not leave the good region.
2. Every dimension in the solution space may require different scaling so one dimension may require large mutation steps while another dimension only small ones.

Both these facts proved to be crucial to the ability of the algorithm to find an optimal solution during the implementation, so they are discussed and examined in the following sections.

2.4.1 Mutation with one step size

In this version of mutation, we change the vector \vec{x} from the previous section, formula 2.4.

$$\vec{x} = ((x_1, \dots, x_n), \sigma) \quad (2.4)$$

(x_1, \dots, x_n) is the original vector and σ is a real-valued strategy parameter which controls the mutation process for every chromosome separately. The mutation process is described in formulas 2.6 and 2.7. The constant τ is set by the user and it is inversely proportional to the square root of the problem size. It represents the learning rate of the algorithm. The σ parameter is then mutated by multiplying with a variable with log-normal distribution, which ensures that σ is always positive. The reasons for this form of mutation of σ by a log-normal distribution are stated in [5]. The newly mutated σ is then used to mutate the solution values for the child where $i = 1, \dots, n$ are the n elements making up one chromosome. It is important to keep the right order of the mutation — to mutate the value of σ first and then mutate the vector's elements itself.

$$\tau \propto \frac{1}{\sqrt{n}} \quad (2.5)$$

$$\sigma(t+1) = \sigma(t) \cdot e^{\tau \cdot N(0,1)} \quad (2.6)$$

$$x_i(t+1) = x_i(t) + \sigma(t+1) \cdot N_i(0,1), \quad i = 1, \dots, n \quad (2.7)$$

This approach allows the mutation step size to coevolve with the chromosomes since the fitness function indirectly evaluates the suitability of the mutation. It is important to vary the step size during the evolution run as we want the algorithm to prefer longer steps when it is far away from the optimum and small steps when it is close to the optimum.

2.4.2 Mutation with n step sizes

The fitness function surface can have a different inclination in every dimension, so while one dimension requires large mutation steps, another one might require only subtle ones. We can solve this problem by adding a strategy parameter σ to every element of the chromosome's solution vector, formula 2.8.

$$\vec{x} = ((x_1, \dots, x_n), \sigma_1, \dots, \sigma_n) \quad (2.8)$$

The mutation process is then described in formulas 2.10 and 2.11.

$$\tau' \propto \frac{1}{\sqrt{2\sqrt{n}}} \quad (2.9)$$

$$\sigma_i(t+1) = \sigma_i(t) \cdot e^{\tau' \cdot N(0,1)' + \tau \cdot N_i(0,1)} \quad (2.10)$$

$$x_i(t+1) = x_i(t) + \sigma_i(t+1) \cdot N(0,1)_i, \quad i = 1, \dots, n \quad (2.11)$$

Notice that the mutation in equation 2.6 differs from 2.10, we added one more normally distributed variable (formula 2.9) to the exponent. A simple modification of 2.6 which we could do is shown in formula 2.12, where we did not add anything to the exponent. The reason why we add one more normally distributed variable in equation 2.10, which is used in the actual algorithm, is because we want to keep the overall change of the mutability but we also want a finer granularity on the coordinate level.

$$\sigma_i(t+1) = \sigma_i(t) \cdot e^{\tau \cdot N_i(0,1)} \quad (2.12)$$

The preceding equations were adopted from [1, 5] where also further details are stated.

Chapter 3

Analog amplifiers

One of the goals of the thesis is to automate the design of analog amplifiers which are electronic circuits that are used to increase the input signal with the minimum amount of distortion to the output signal. There are two amplifiers used in this thesis which are discussed below.

3.1 Single stage common emitter amplifier

The single stage common emitter amplifier was chosen because it belongs to the most commonly used examples of analog amplifiers in class A. The circuit diagram is shown in figure 3.1. We can find a thorough description of this circuit in [6].

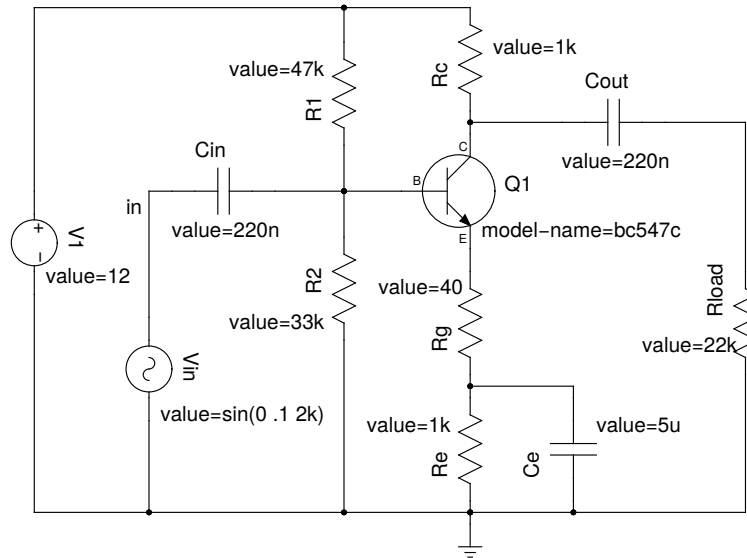


Figure 3.1: Circuit diagram for the common emitter amplifier

Resistors $R1$ and $R2$ are used to bias transistor $Q1$ in order to set the quiescent point of the transistor to the middle of its DC load line so that the collector of $Q1$ is put at $1/2$ of the supply voltage $V1$. This allows the maximum symmetrical swings of the output signal without clipping (flattening of the top or the bottom of the waveform). The collector

voltage depends on the collector current (quiescent current), equation 3.1. This current depends on the applied base bias and the values of R_c and R_e .

$$V_c = V_1 - I_c \cdot R_c \quad (3.1)$$

Instead of using a voltage divider for biasing, it is possible to use only resistor R_1 , but this approach would make the quiescent point highly dependent on the current gain β of the transistor. This parameter varies considerably in every transistor and the manufacturer usually specifies only a certain range of β . For this reason, we use a voltage divider constituted by R_1 and R_2 which impedance should be small compared with the impedance looking into the transistor base, formula 3.2. This will give us a divider which is stiff enough so that the quiescent point is insensitive to variations in transistor β . However, the current flowing in the divider should not be unnecessarily large as it affects the overall consumption of the amplifier.

$$R_1 \parallel R_2 \ll \beta R_e \quad (3.2)$$

Capacitors C_{in} and C_{out} form high pass filters and they are used as coupling capacitors to separate the AC signals from the DC voltage used to set up the amplifier. Their values are chosen so that the capacitors have low impedance for the desired input and output signal frequencies. This ensures that the signal's average is zero, as the capacitors pass only AC signals and block any DC component.

The input voltage V_{in} causes a wiggle in the base voltage. The emitter voltage follows the base voltage which causes a wiggle in the emitter and also collector current. The output voltage which is the collector voltage depends on the current flowing through R_c , equation 3.1. When this current rises, the collector voltage drops and vice versa. That means that the amplifier also inverts the input signal, figure 3.3. The output AC signal is then superimposed on the collector DC voltage and the DC component is subsequently filtered out by the capacitor C_{out} (high pass filter).

Capacitor C_e is used to reach the amplifier's maximum gain. The values of R_e and R_g set the emitter voltage which affects the amplifier's gain and we usually want these values to be as low as possible since this gives us the highest gain. However, if the emitter voltage is too low, it will vary significantly as the base-emitter drop varies with temperature. The solution is to bypass the R_e resistor so that the impedance for the emitter will vary according to the signal's frequency. The bypass capacitor C_e is an open circuit component for DC bias and the emitter voltage depends on both R_e and R_g which allows stable biasing. For higher frequency signals, the bypass capacitor short circuits R_e and the emitter voltage now depends mostly on R_e which value sets the maximum gain.

The analytical calculation for this circuit is beyond the scope of this thesis and it is left up to the evolutionary algorithms. The subject of the optimization are elements R_1 , R_2 , R_e , R_g , R_c , C_{in} , C_e and C_{out} . These elements represent the solution variables in the evolution strategies and make up the vector \vec{x} which is thoroughly discussed in section 2.4. In this case, we optimize 8 components in the circuit so the search space has 8 dimensions.

3.2 Two stage amplifier

We can see the circuit diagram for a two stage amplifier on figure 3.2. The functionality and structure of the circuit are similar to the previous one with the difference that now we use two transistors connected in a cascade where transistor Q_1 sends its output to the

base of transistor $Q2$. This circuit was chosen as a more difficult problem to optimize, since the structure is twice as complicated as in the previous amplifier. The subject of the optimization are all the resistors and capacitors in the diagram apart from resistor R_{load} , so we have 14 components to optimize which makes the search space 14-dimensional.

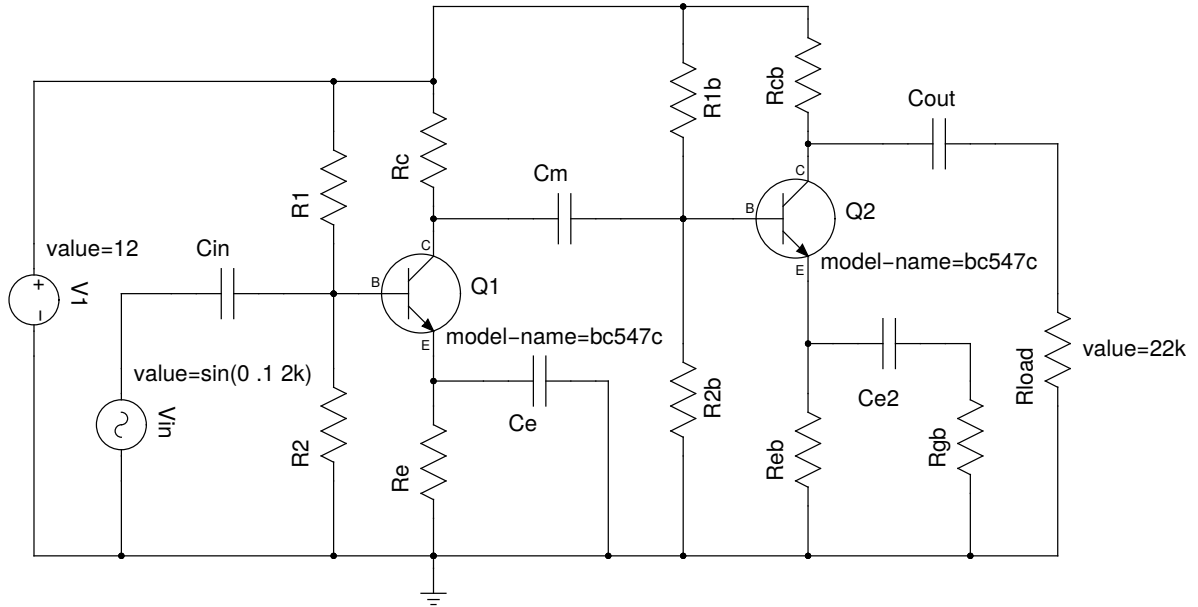


Figure 3.2: Circuit diagram for the two stage common emitter amplifier

3.3 ngSPICE

ngSPICE¹ is an open-source, general-purpose circuit simulation program based on SPICE (*Simulation Program with Integrated Circuit Emphasis*). It can simulate circuits with linear components and also nonlinear circuits which contain common semiconductor devices. Besides the simulation of analog circuits, it also offers simulation of digital circuits, in which case the simulator operates only with the logic states of the circuit, which has a great influence on the speed of the simulator. ngSPICE also offers a mixed-mode simulation where both the digital and analog simulations are combined together [7].

ngSPICE uses a language which syntax has been inherited from SPICE for describing circuits. An example of such description of the circuit from figure 3.1 is shown in listing 3.1. The first line contains the name of the circuit and on the next four lines, there is a description of the transistor used in the circuit. On the next lines, there is a list of the circuit's components. The name of the component is on the first position followed by the numbers or names of the nodes which the component is connected to. The last position contains the properties of the component. The second last line contains the description of the simulation. In this case, we do a transient analysis of the circuit for 1.22 ms with the sampling period 20 μ s which provides us the time course of the amplifier's output signal. Later on, we use this data to evaluate the amplifier's quality.

¹ngSPICE home page - <http://ngspice.sourceforge.net/>

Such netlist can be generated by using the gEDA ² (*Electronic Design Automation*) toolkit which is also an open-source project oriented towards electronic design.

```
1  amplifier
2  .model  bc547c  NPN  (BF=730  NE=1.4  ISE=29.5F  IKF=80M  IS=60F
3          +  VAF=25  ikr=12m  BR=10  NC=2  VAR=10  RB=280  RE=1  RC=40
4          +  VJE=.48  tr=.3u  tf=.5n  cje=12p  vje=.48  mje=.5
5          +  cjc=6p  vjc=.7  mjc=.33  isc=47.6p  kf=2f)
6  Vin  in  0  sin(0  .1  2k)
7  V1  4  0  9
8  Ce  0  5  5u
9  Cout  1  out  220n
10 Cin  in  3  220n
11 Rc  1  4  1k
12 Rload  0  out  22k
13 Re  0  5  1k
14 Rg  5  2  40
15 R2  0  3  33k
16 R1  3  4  47k
17 Q1  1  3  2  bc547c
18 .TRAN  20u  1.22m
19 .end
```

Listing 3.1: description of the common emitter amplifier using the SPICE syntax

In figure 3.3 we can see the simulation output of the circuit from listing 3.1 which is the same circuit as in figure 3.1. This circuit was also built using real electronic components in order to verify the simulation results. The measurement results of the real circuit proved that the simulator works correctly for this circuit as the measured values corresponded to the simulation.

The last version of ngSPICE offers a C API which makes the simulator the ideal choice for utilizing it with the evolutionary algorithms. Every chromosome (candidate solution) in the evolution represents one amplifier and the simulator is used to determine the quality of it.

The API is used to pass the description of the amplifier to the simulator, run the simulation and obtain the simulation results. The results are in the form of arrays in C which represent time and the corresponding voltage values. These arrays have an equal length which depends on the simulation duration and the sampling period.

The default input frequency for the amplifier is 2 kHz, the duration of the simulation is set to 1.22 ms and it contains approximately two and a half period of the output signal. The duration does not need to be longer because the shape of the waveform does not change after the circuit gets to a stable state. The sampling period is 20 μ s so that the waveform is smooth enough on one hand and the simulation does not take too long on the other hand. The array containing the output voltage is used for the evaluation of the chromosomes which is thoroughly discussed in chapter 4.

The simulator is launched several thousand times during a typical evolution and it takes most of the computation time. During the implementation of the algorithms discussed in section 2 and the integration of the simulator, it emerged that there are memory leaks on

²gEDA home page - <http://geda-project.org/>

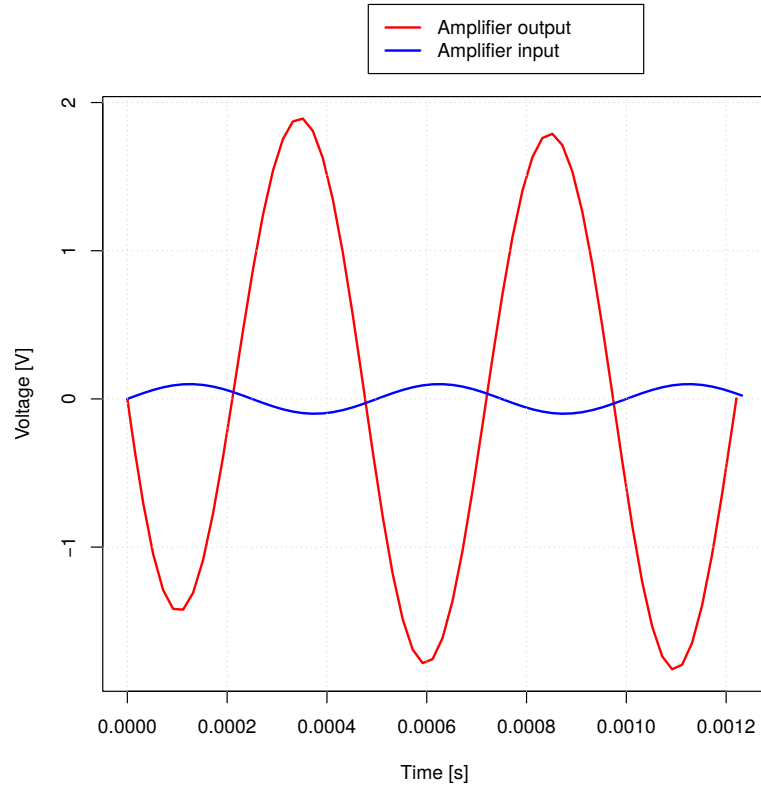


Figure 3.3: Simulation of the single stage common emitter amplifier

several places in the source code of the simulator. This was a problem as the simulator leaked memory in every iteration. A significant amount of time was spent on fixing these bugs and the result is that the memory leaks have been removed. These improvements were proposed as a patch to the development team and they will be incorporated into the next versions of ngSPICE.

Chapter 4

Evaluation of amplifiers

A fitness function provides us means by which we can determine how good the evolved solution is so that we can distinguish the more appropriate candidates from the less appropriate ones. It is important to mention that the quality of this function is essential because the ability of the algorithm to find the most suitable solution is highly dependent on the ability of the fitness function to distinguish between the individuals. The value of the function is a positive real number in our case and the algorithm uses it only for comparing the individuals between each other. We consider those individuals with the lower value as the better ones.

The implementation required some methods of trial and error and in the end, there are three fitness functions implemented in this thesis. All of them use the ngSPICE simulator in order to obtain the output of the amplifier and assess it afterwards. The simulation is done for every candidate solution and it takes most of the computation time of the overall evolutionary algorithm.

4.1 Best match with the reference solution

This fitness function has been implemented as the first one in order to find out if the evolution has at least the ability to get close to the analytical solution. It is based on the method of least squares which is used in regression analysis.

The fitness function uses the output voltage vector from the simulator for assessing the chromosomes. The vector represents the waveform of the output signal from the circuit and the shape of the waveform gets similar to an inverted sine wave as the solution evolves. The length of the vector is set to 69 (the sampling period is approximately 20 μ s) elements during the whole evolution and the vector contains approximately 2.5 periods of the signal (the signal's frequency is 20 kHz). The fitness function uses two types of these vectors, one serves as the reference vector (the output of the amplifier which elements' values were calculated analytically) and all the candidate vectors are compared to it. As we can see in figure 3.3, the first period of the signal is unstable, so the fitness function skips it and picks only the second period for comparison with the reference signal. It is sufficient to compare only the second period because the shape of the signal doesn't change anymore after the circuit gets to a stable state.

The functions in algorithm 3 are used to extract the second period from the waveform. Since the vector has a constant length and the values represent an inverted sine wave, we

can walk through the first two and the last half-period to get to the desired part of the signal.

The first function starts at the start of the vector and it skips the first half-period which values are less than zero and it also skips the second half-period in a similar way. The second function starts from the end of the vector and it skips the second last period of the signal in the vector. The resulting *start* and *end* indices point to the start and the end of the second period of the signal.

When the evolution starts, the waveform of some candidate solutions is not in the shape of an inverted sine wave and therefore it crosses zero sooner than the desired waveform. In this case, the algorithm ends with indices close to the starting and ending index and such candidate solution is assessed with a high fitness value.

Algorithm 3 Find the first and last index of the second period

```

1: function GETSTART(vector)
2:   start  $\leftarrow$  0;
3:   while vector[start] < 0 do
4:     start  $\leftarrow$  start + 1;
5:   end while
6:   while vector[start] > 0 do
7:     start  $\leftarrow$  start + 1;
8:   end while
9:   return start;
10: end function
11:
12: function GETEND(vector)
13:   end  $\leftarrow$  vector.length();
14:   while vector[end] < 0 do
15:     end  $\leftarrow$  end - 1;
16:   end while
17:   return end;
18: end function

```

The function in algorithm 4 is used to evaluate the candidate solutions. It iterates over the second period in both reference and candidate vectors and it calculates the sum of squares which sides are defined by the difference between the values in the reference vector and the candidate vector. The overall sum is returned as the result of the fitness function and the lower the value, the better the candidate solution is.

The reason why we add up squares and not only the absolute values of the differences is important. Consider two imaginary vectors of two values which differ from the reference vector by (1, 5) and by (3, 3). The sum of the absolute values is 6 in both cases, so this method doesn't distinguish between these vectors. However, the sum of the squares is 26 for the first vector and 18 for the second one. This difference is important as we want all the values to be close to the reference and not only some of them and therefore the second vector is assessed as the better one.

We can see the result of the evolution using this fitness function in figure 4.1.

Algorithm 4 Fitness evaluation using the analytical solution

```
1: function RATECHROMOSOME(candidateVector, referenceVector)
2:   fitness  $\leftarrow$  0;
3:   start  $\leftarrow$  GETSTART(candidateVector);
4:   end  $\leftarrow$  GETEND(candidateVector);
5:   for i  $\leftarrow$  start : end do
6:     fitness  $\leftarrow$  fitness + (referenceVector[i] - candidateVector[i])2;
7:   end for
8:   return fitness;
9: end function
```

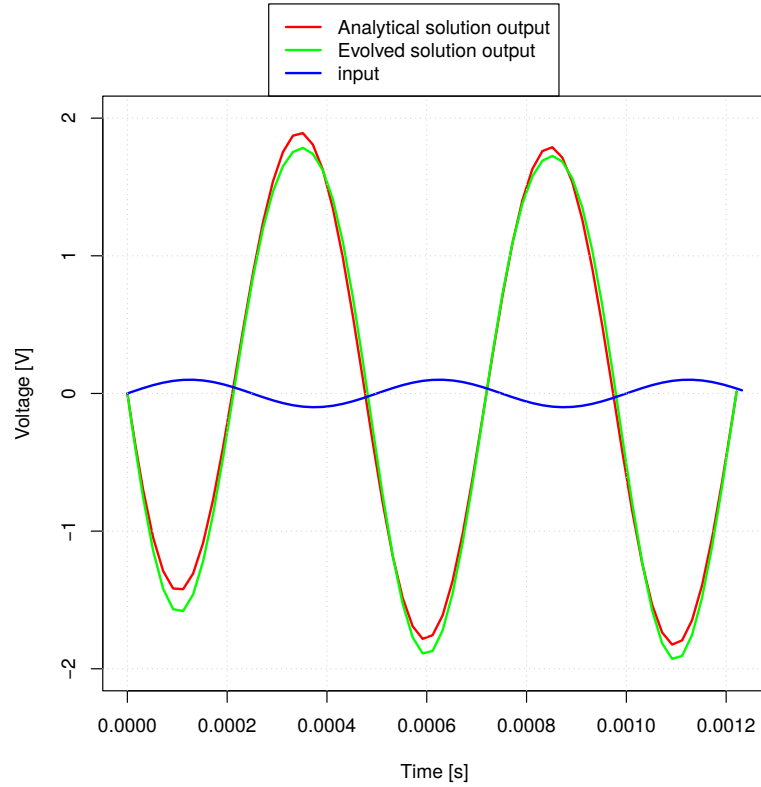


Figure 4.1: The result of evolution towards the analytical solution

4.2 Ideal sine wave

This method is similar to the previous one but instead of using the pre-simulated output as a reference, it uses an analytically calculated sine wave. An analytical sine wave is used in order to force the evolution towards such solution that won't produce a distorted output signal since the input is in the form of a sine wave as well. The function also uses the method of least squares explained in the previous section and it iterates over the second period of the signal which is represented by the *candidateVector*.

We can see the technique in algorithm 5. In every iteration, it calculates the value of the sine wave and compares it with the simulated signal. The sum of the comparisons is then returned as the fitness value and the lower the result is, the closer the candidate solution is

to the reference. The amplitude of the sine wave is set by the user, so this method allows users to design amplifiers with an arbitrary amplification which doesn't exceed the circuit's capabilities.

Algorithm 5 Fitness evaluation using the ideal sine wave

```

1: function RATECHROMOSOME(candidateVector, amplitude)
2:   fitness  $\leftarrow$  0;
3:   start  $\leftarrow$  GETSTART(candidateVector);
4:   end  $\leftarrow$  GETEND(candidateVector);
5:   refSineSize  $\leftarrow$  end - start;
6:   for i  $\leftarrow$  0 : refSineSize do
7:     refSine  $\leftarrow$  -amplitude  $\cdot$   $\sin(\frac{2\pi i}{refSineSize-1})$ ;
8:     fitness  $\leftarrow$  fitness + (refSine - candidateVector[start + i])2;
9:   end for
10:  return fitness;
11: end function

```

4.3 Maximal amplitude

This fitness function is designed to rate the candidate solutions only according to the amplitude of the output regardless of the waveform's shape. It can be used for finding the highest amplification capabilities of the circuit. The function finds the trough and the peak in the second amplitude and returns the multiplicative inverse of their difference.

Algorithm 6 Rating the chromosomes according to the amplitude

```

1: function RATECHROMOSOME(candidateVector)
2:   start  $\leftarrow$  GETSTART(candidateVector);
3:   end  $\leftarrow$  GETEND(candidateVector);
4:   trough  $\leftarrow$  min(candidateVector[start], candidateVector[end]);
5:   peak  $\leftarrow$  max(candidateVector[start], candidateVector[end]);
6:   return  $\frac{1}{peak-trough}$ ;
7: end function

```

4.4 Waveform symmetry

All of the fitness functions discussed above have difficulties in finding symmetrical waveforms. An example is shown in figure 4.2.

The problem is that at the start, the output signal has zero power and as the evolution continues, only one half of the signal rises and the fitness function value decreases even though the evolution doesn't go in the right direction. At the end, the evolution ends up in the state shown in figure 4.2. The first solution to this problem is described in formula 4.1.

$$result = (|peak + trough| + 1) \cdot fitness \quad (4.1)$$

The *peak* and *trough* values are obtained in the same way as in algorithm 6. If the signal is symmetrical, the absolute value of the sum of these values is close to zero and

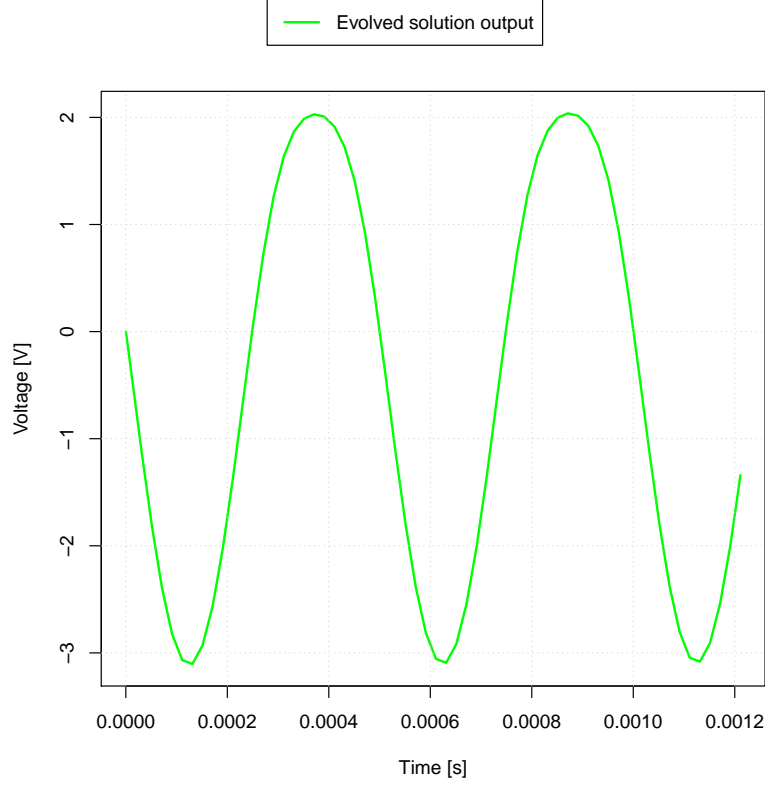


Figure 4.2: An asymmetrical result of the evolution using the 'ideal sine wave' evaluation

therefore the *result* will also be lower. We add 1 to the sum because we need to distinguish between perfectly symmetrical signals with different fitness. However, it emerged that this approach is too restrictive as it promotes only signals which are perfectly symmetrical, but often distorted. The result is shown in figure 4.3.

The ability of the evolution to find the desired solution was also decreased because some asymmetrical candidate solutions head towards a good solution. These consequences led to another solution to this problem.

Algorithm 7 Rating the chromosomes with regard to the symmetry of the signal

```

1: function RATECHROMOSOME(candidateVector, maxDifference)
2:    $min \leftarrow \min(|trough|, peak);$ 
3:    $max \leftarrow \max(|trough|, peak);$ 
4:   if  $\frac{min}{max} < (1 - \frac{maxDifference}{100})$  then
5:     return DOUBLE_MAX;
6:   end if
7:   ▷ continue evaluating the chromosome
8: end function

```

Condition on line 4 in algorithm 7 is used to separate the symmetrical and asymmetrical chromosomes. The values of the variables *trough* and *peak* are obtained in the same way as in algorithm 6. The value of *maxDifference* is in the interval $]0, 100]$ and it is set by the user. It represents the maximal percentage difference between the peak and trough in

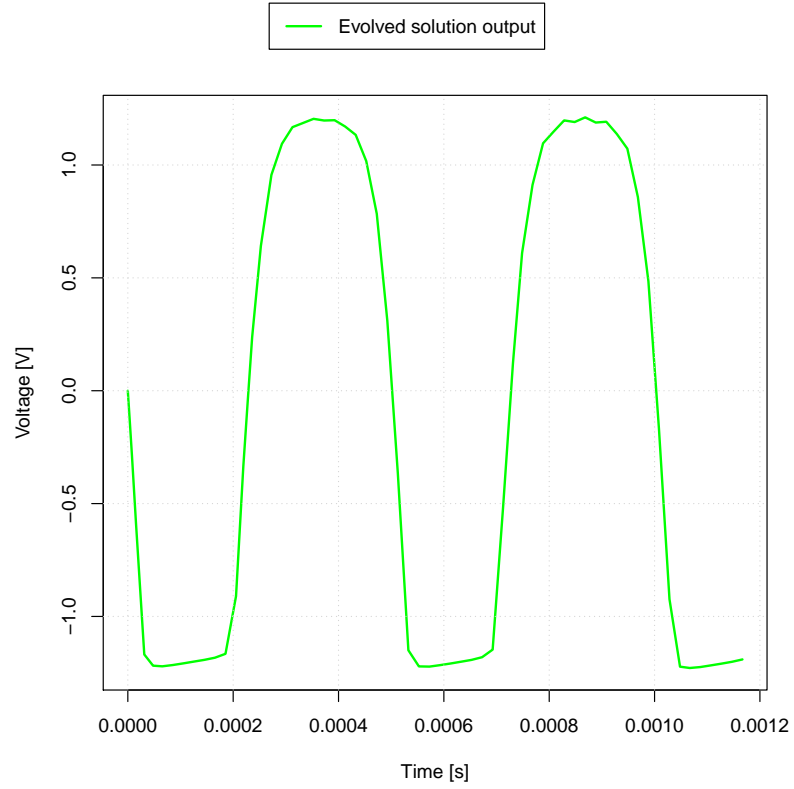


Figure 4.3: A distorted result using the 'ideal sine wave' evaluation with perfect symmetry

the second period of the signal. The algorithm calculates the ratio between the peak and trough and if the result is less than the threshold value, the chromosome is evaluated with the maximal fitness and the function is terminated. Otherwise, the evaluation continues. This allows the user to control the results of the evolution with respect to the symmetry of the output signal.

Chapter 5

Experiments

The evolutionary algorithms described in section 2 were implemented in C++. This language was chosen due to its flexibility, performance and the possibility to make use of the ngSPICE simulator. The implementation also utilizes the RInside¹ library which integrates R² into C++ [4]. The R language is used to plot graphs dynamically during the evolution either to the screen or to a file so the user can see and record how the solution evolves.

The program is written as a console application and the user can specify the properties of the evolution via command-line arguments. This approach also allows the user to easily run the application multiple times and collect the results of the evolution.

The results are in the form of graphs and a text output which contains the details about the evolution run. The graphs can be either dynamically generated to the screen during the program run or saved as files to the local drive as well as the text output. The user can specify how often the graphs and text description are generated, for example, each time when the fitness function of the best chromosome in the population decreases by 1%.

The results of the experiments discussed below are related to the evolutionary algorithm itself. The results of the optimization of the amplifiers are analyzed in sections 5.5 and 5.6.

5.1 Initial stages of experiments

The first experiments were performed only in order to optimize the value of resistor $R1$ in the single stage amplifier. We used the 'best match' evaluation method and mutations with one step size. The evolution always found a value close to 50 k Ω which was the desired solution. These experiments were always successful as this was a very simple task for the evolution.

In the next stage, the task was to optimize the voltage divider constituted by $R1$ and $R2$. The 'best match' method and mutations with one step size were used again and it emerged, that the evolution was able to find the solution even faster than in the previous case. This was because we were looking only for the right ratio between the resistors and not for specific values so the set of the desired solutions was much larger in this case even though the search space was larger as well as it was two-dimensional.

The next task for the evolution was to find the right values for resistors $R1$, $R2$ and Rg . The evolution parameters were set in the same way as in the previous experiments. This task showed the weak spot of mutations with one step size as the evolution was able to find

¹RInside home page - <http://dirk.eddelbuettel.com/code/rinside.html>

²R home page - <https://www.r-project.org/>

an acceptable solution very rarely. The reason was that the desired value of Rg is very close to the lower bound of the search space (the analytical value is only 40Ω). Since mutations with one step size use the same mutation value in every dimension, it was not possible to mutate Rg only by a few ohms (not to move away from the lower bound of this dimension) and at the same time look for the right values in other dimensions which required steps in kilohms. The value of Rg heavily influences the overall fitness of the chromosome because high values significantly decrease the gain of the amplifier. The usual course of the evolution was that the value of Rg converged quickly to the lower boundary and since this parameter has a great influence on the fitness, the evolution preferred chromosomes with this property. This led to a quick decrease of the mutation step size σ (the fitness function also indirectly evaluates the value of σ) and the algorithm got stuck in a local optimum since other dimensions (values of other resistors) in the search space could not be searched properly. The solution to this problem was to implement mutations with n step sizes discussed in section 2.4.2. This approach allows us to treat every dimension separately and therefore search the search space more effectively. After the implementation, the evolution was able to find a satisfactory solution in most of its runs.

The previous experiments proved that the evolutionary algorithms are able to provide acceptable solutions in the domain of analog amplifiers. The next section describes the process of determining the ideal parameters for the evolution.

5.2 Evolution parameters

In order to make the evolution strategies work efficiently, we need to find and set the correct parameters for the evolution.

The size of the search space is defined by the number of dimensions and by the size of every dimension. The number of dimensions depends on the number of electronic components that we want to optimize and the sizes of the dimensions depend on the range of the values that every component can have. These ranges were set from 0 to 200 k Ω for resistors and from 0 to 500 μ F for capacitors. The upper bounds are based on the values of real electronic components so, for example, we do not expect to have a higher resistance than 200 k Ω in the circuit. The values of some particular components could have a smaller range, for example, resistor Rg which value does not need to be higher than a few kilohms, but the assumption is that we have no knowledge about the circuit at all.

The initial mutation step size σ is set to 100. This value is not very important because this parameter is adapted during the evolution. Experiments showed that there is no difference between setting the initial σ to 10 or 500 for example. However, if we set the initial value too high (10000 and above), it is more likely that the evolution will get stuck in a local optimum because the algorithm converges too quickly at the start.

The experiments also showed that the values of the learning parameters τ and τ' have a significant impact on the speed at which the evolution converges to the optimum. The higher the values were, the faster the evolution converged, nevertheless this also means that the probability of finding only a local optimum was rapidly increased. For this reason, the values were set according to formulas 5.1 and 5.2 to their lowest possible values. The parameter n is the number of dimensions of the search space.

$$\tau = \frac{1}{\sqrt{n}} \quad (5.1)$$

$$\tau' = \frac{1}{\sqrt{2\sqrt{n}}} \quad (5.2)$$

The maximum difference between the peak and trough of the amplifier's output waveform was empirically set to 25. This means that the values of the peak and trough of the waveform can differ at most by 25%.

5.3 Population size, selection type and selective pressure

Important parameters of the evolution are numbers μ and λ which define the population size (the number of parents and children in every generation) and the selective pressure which is defined by the ratio between μ and λ . The next important parameter is the type of the selection that we choose for the evolution ((μ, λ) -ES or $(\mu + \lambda)$ -ES). In order to determine the most suitable values of the parameters, there were various experiments carried out which are summarized in table 5.1.

Every row of the table contains results for a different combination of μ and λ . The values of μ are set to 1, 5, 10 and 15 and the selective pressure is also 1, 5, 10 and 15 for every value of μ , so there are four different values of λ for every μ . For every such combination, there are results for both selection types and under every type, there are three columns representing all the three evaluation methods that are discussed in chapter 4. One cell of the table corresponds to the result of one experiment where the evolutionary algorithm was run 10 times with the same parameters and the fitness values of the resulting chromosomes were added up and divided by 10. The task was to optimize the single stage amplifier.

We can see that there is no significant difference between the selection schemes, provided the selective pressure is higher than one. On lines 5, 9 and 13 (where the selection pressure is 1) the (μ, λ) -ES scheme performs significantly worse than the $(\mu + \lambda)$ -ES scheme and even increasing the population size has no effect (line 5 compared to line 9 and 13). However, when we sum up all the values in both schemes (excluding lines 1, 5, 9 and 13), the latter one gives us better results, so the $(\mu + \lambda)$ -ES selection scheme was chosen as the more effective one.

We can also see that when the value of μ is only 1, the algorithm's results are substantially worse than in the rest of the table and on the other hand when we increase it from 5 to 15, there is only a small change in the overall performance.

The value of λ is set according to the value of μ multiplied by the selection pressure and it highly influences the overall computation time of the algorithm because it defines the number of chromosomes that have to be evaluated in every generation (the evaluation includes the amplifier's simulation which takes most of the computation time). When the selective pressure and the value of μ are higher than one, the overall efficiency of the algorithm does not increase rapidly but there is a noticeable increase in the computation time. For this reason, the values of μ and λ were set to $\mu = 15$ and $\lambda = 150$ as a compromise between the algorithm's time demands and the optimization efficiency. Higher values were also examined but they did not provide any significant improvements.

	μ	λ	(μ, λ) -ES			$(\mu + \lambda)$ -ES		
			best match	ideal sin	max. ampl.	best match	ideal sin	max. ampl.
1.	1	1	N/A	N/A	N/A	21.82	99.00	2.90
2.	1	5	20.82	113.75	1.98	13.44	57.91	2.69
3.	1	10	18.84	76.81	1.40	24.70	75.01	2.84
4.	1	15	18.43	128.75	2.26	19.52	34.11	1.02
5.	5	5	49.62	169.02	36.50	9.81	56.61	3.18
6.	5	25	7.28	55.00	3.12	12.12	34.88	2.72
7.	5	50	16.18	64.22	3.21	9.75	43.73	3.57
8.	5	75	3.70	58.82	3.10	4.20	38.84	3.52
9.	10	10	49.41	154.66	26.40	8.94	53.77	3.18
10.	10	50	9.09	60.93	2.68	7.02	45.04	3.58
11.	10	100	6.97	38.02	3.16	4.91	26.06	3.10
12.	10	150	8.06	25.54	3.10	2.52	23.07	3.07
13.	15	15	44.48	165.76	34.99	2.54	47.69	3.18
14.	15	75	4.64	56.18	2.25	4.96	51.74	3.56
15.	15	150	6.63	47.32	3.52	6.36	12.19	1.60
16.	15	225	7.56	31.17	2.65	0.70	19.31	1.08

Table 5.1: Results of experiments, cells contain results where the evolution was run 10 times and the fitness values of the resulting chromosomes were added up and divided by 10

All the evolution parameters discussed above are summarized in table 5.2. These parameters were chosen as the most appropriate ones.

Parameter	Value
Resistance range	0 to 200 k Ω
Capacitance range	0 to 500 μ F
initial σ	100
max. peak-trough difference	25 %
μ	15
λ	150
selection type	$(\mu + \lambda)$ -ES
τ	equation 5.1
τ'	equation 5.2

Table 5.2: Optimal evolution parameters

5.4 The typical course of the evolution

Figure 5.1 shows us the typical course of the evolution. The population size parameters were set to $\mu = 10$ and $\lambda = 100$ and the terminating condition of the algorithm was that the evolution ends when the value of the fitness function of the best chromosome does not decrease by more than 1% after 300 generations.

The green line represents the fitness of the best chromosome (the one with the lowest fitness) in every generation, the blue line is the fitness of the worst chromosome and the

red line is the average fitness of every generation. In this case, we pick the chromosomes only from the set of μ parents. We can see that the red line is very close to the green line so the average fitness of every generation is close to the fitness of the best chromosome and that the blue line is scattered above the other two lines.

The evolution was stuck in a local optimum for approximately 20 generations between the 80th and 100th generation and the solution was evolved approximately after 300 generations. When the blue line is close the other two lines, it means that the sizes of the mutation steps σ are low and the algorithm searches only a small area of the search space. When the values of σ are low (close to zero), we can stop the algorithm after a few hundreds of generations because it will not provide any better solution than the current one and the resulting fitness is either in a local or global optimum.

The amount of generations needed for the optimization is usually a few hundred, as also shown in figure 5.1. When the size of the population is low (for example $\mu = 1$ and $\lambda = 5$), the optimization takes a few thousands of generations. In any case, there is approximately up to 30000 simulations in every evolution run which takes a few minutes (usually between two to four minutes) on the Intel Core i5-5200U processor.

In this case, the evaluation method was the 'best match' method and this evolution run was successful since the resulting fitness was very close to zero. With this method, we can consider those solutions which fitness is under 0.5 as sufficient because the amplifier's output waveform is very close to the desired one. The resulting value of the fitness of the best chromosome was 0.19.

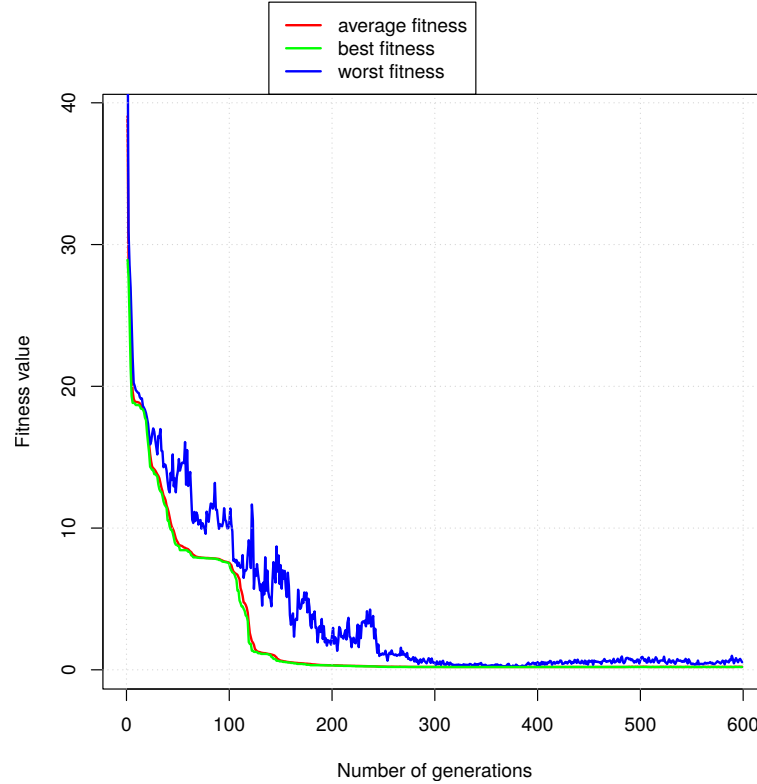


Figure 5.1: The typical course of the evolution

5.5 Single stage amplifier optimization

Every evaluation method implemented in this thesis provides us different means by which we can optimize the circuit.

The 'best match' method can be used for finding such properties that provide us the most similar output compared to the analytical solution. We optimize the values of 8 electronic components and experiments with this method proved that there are many other possible combinations which provide almost identical behaviour of the amplifier. Five examples are shown in table 5.3. The simulation output of the amplifier with these values of components was visually identical with the analytical solution.

$R1$ [k Ω]	$R2$ [k Ω]	Re [Ω]	Rg [Ω]	Rc [k Ω]	Ce [nF]	Cin [nF]	$Cout$ [nF]
200	76.2	7510	471	21.7	460	29	1010
78.5	16	3370	429	23	228 000	17	396 000
141	11.5	180	483	21.1	285 000	21	280 000
82.3	44.3	8590	400	16	499	202	95
118	41.2	11500	559	35.4	419	95	19

Table 5.3: Values of components that provide visually identical outputs with the analytical solution

The 'maximal amplitude' method can be used for finding the best amplification capabilities of the circuit as it does not take into consideration the shape of the output signal. This method is useful for the 'ideal sine' method because it allows us to find the appropriate maximal amplitude and then we can find the right waveform by using the latter method. By this approach, we can find even better solutions than the analytical one. One of such solutions is shown in figure 5.2 and the values of the components are summarized in table 5.4. Since the amplitude of the input signal was $V_{in} = 100$ mV and the amplitude of the output signal is approximately 4.2 V, the gain of this amplifier is approximately 42 (the gain of the analytically solved amplifier is approximately 18). The gain also depends on the voltage $V1$ that supplies the amplifier which is set to $V1 = 12$ V in the experiments.

$R1$ [k Ω]	$R2$ [k Ω]	Re [Ω]	Rg [Ω]	Rc [k Ω]	Ce [μ F]	Cin [μ F]	$Cout$ [nF]
168	15.1	169	49	4.21	16	274	86

Table 5.4: The best solution for the single stage amplifier found by the evolution

The 'ideal sine' evaluation method also provides us a universal tool for finding the appropriate values of components which produce an arbitrary amplification of the amplifier up to the maximal amplitude.

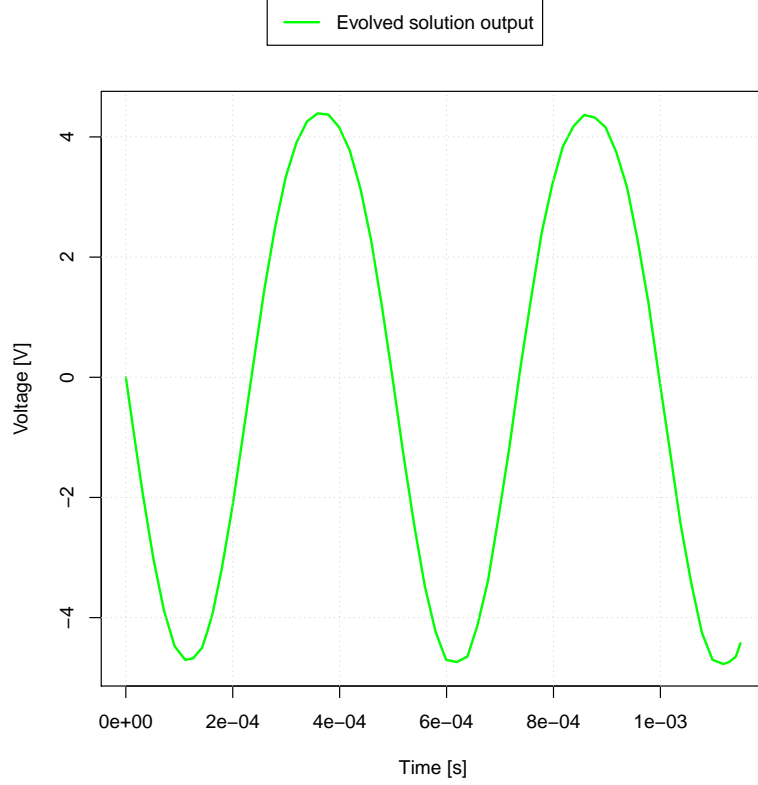


Figure 5.2: The best solution for the single stage amplifier found by the evolution

5.6 Two stage amplifier optimization

This amplifier was chosen as the more complicated task to optimize. There was no analytical solution for this circuit so it was not possible to use the 'best match' evaluation method. Experimenting with this type of amplifier showed that this amplifier is much more difficult to optimize and that it does not provide any better amplification properties than the previous single stage amplifier.

The same set of experiments was carried out as for the previous amplifier and one of the best solutions is shown in figure 5.3.

We can see that the resulting gain is approximately 50 (the amplitude of the input voltage is $V_{in} = 100\text{ mV}$, the amplitude of the output voltage is approximately 5 V, the supply voltage is $V_1 = 12\text{ V}$) but the output waveform is significantly distorted — the circuit generates a square wave. The experiments did not provide any solution that would generate a sine wave. The components values for this solution are stated in table 5.5.

$R1$ [k Ω]	$R2$ [k Ω]	Re [k Ω]	Rc [k Ω]	Ce [μF]	Cin [μF]	$Cout$ [μF]
Rgb [Ω]	Reb [Ω]	Rcb [k Ω]	$R2b$ [k Ω]	$R1b$ [k Ω]	Cm [μF]	$Ce2$ [μF]
114	57	51.6	81.7	98	215	479
22	298	3.37	9.24	80.5	351	139

Table 5.5: The best solution for the two stage amplifier found by the evolution

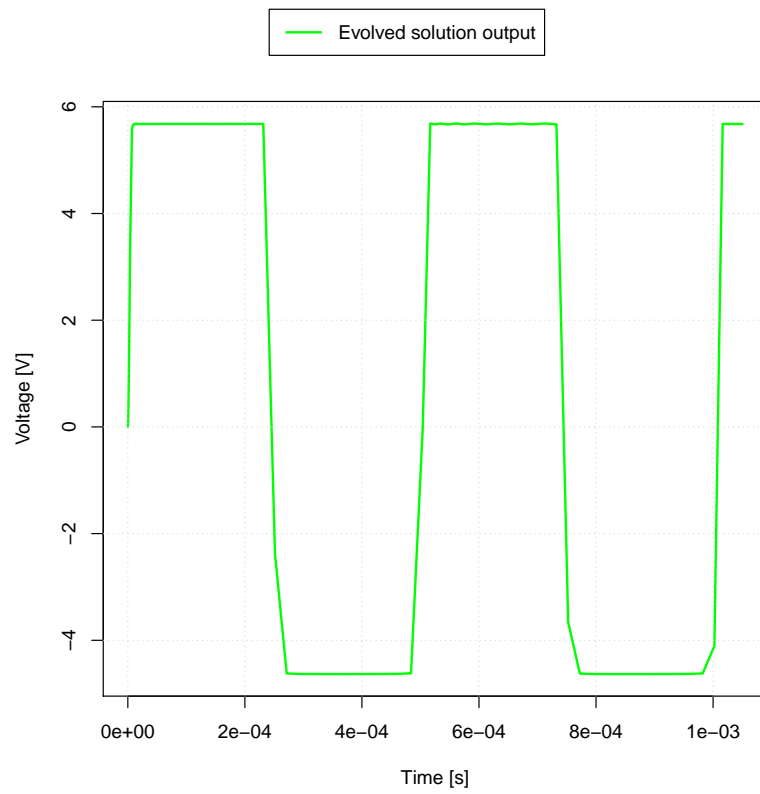


Figure 5.3: The best solution for the two stage amplifier found by the evolution

Chapter 6

Conclusion

This thesis demonstrated the capabilities of the evolutionary algorithms, namely the evolution strategies, in the domain of the analog amplifiers design. In the first phase, the task was to implement the concept of evolution strategies and integrate it with the ngSPICE simulator. An inherent part of the evolutionary algorithms is an appropriate fitness function so the next step was to develop methods for evaluating the quality of the amplifiers. In the last stage, there were various experiments carried out in order to demonstrate the performance of the proposed solution.

There were two types of amplifiers chosen for the optimization and it emerged that the evolution has the capability to find the desired solution and that it can even provide different variations of a circuit that has the same amplification properties. However, the results for the second amplifier were considerably limited.

There are various types of the fitness function developed and evaluated during the implementation and users can choose the most appropriate one according to their needs. The results of experiments were also used for the determination of the most optimal parameters for the evolution. During the experiments, it also emerged that various extensions to the basic version of evolutionary strategies had a great influence on the overall performance of the algorithm.

The resulting application provides a tool for designing amplifiers with arbitrary gain up to the maximum limits of the circuit without using any mathematical apparatus.

This thesis also contributed to the development of the ngSPICE simulator, which had memory leaks on various places in the source code. The simulator is run multiple times in a row during the optimization so there was a need to fix the memory leaks and the result is that they were successfully removed. The solution was proposed to the development team and the improvements will help in the next stages of development.

The future extension to this project could be an interface for entering various electronic circuits described in the SPICE syntax, so that users could utilize the current implementation of evolution strategies with different fitness functions and they would not be limited only to the embedded circuits.

Bibliography

- [1] Brabazon, A.; O'Neill, M.; McGarraghy, S.: *Natural Computing Algorithms*. Springer-Verlag Berlin Heidelberg. 2015. ISBN 978-3-662-43630-1.
- [2] Bäck, T.: *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press. 1996. ISBN 0-19-509971-0.
- [3] Darwin, C.: *On the Origin of the Species: by Means of Natural Selection Or the Preservation of Favoured Races in the Struggle for Life*. Cosimo, Inc.. 2007. ISBN 9781602061453.
- [4] Eddelbuettel, D.: *Seamless R and C++ Integration with Rcpp*. 2013. ISBN 978-1-4614-6867-7.
- [5] Eiben, A.; Smith, J. E.: *Introduction to Evolutionary Computing*. Springer-Verlag Berlin Heidelberg. 2003. ISBN 978-3-642-07285-7.
- [6] Horowitz, P.; Hill, W.: *The Art of Electronics*. Cambridge University Press. 2015. ISBN 978-0-521-80926-9.
- [7] Nenzi, P.; Vogt, H.: *Ngspice Users Manual Version 26plus (Describes actual ngspice source code at git)*. [Online; Accessed: 2017-05-04].
Retrieved from: <http://ngspice.sourceforge.net/docs/ngspice-manual.pdf>

Appendices

Appendix A

User interface

The program is implemented as a console application. The user can run the program in the following way:

```
bt [OPTIONS]...
```

The OPTIONS are as follows:

-h, --help

Prints the help message.

-o <directory_name>

Specifies the output directory. By default, **stdout** is used for the text output and the graphs are displayed on the screen.

--mu <number_of_ancestors>

Sets the cardinality of the population of ancestors.
The default value is 10 chromosomes.

--lambda <number_of_descendants>

Sets the cardinality of the population of descendants.
The default value is 150 chromosomes.

--max-gen <number_of_generations>

Sets the maximum number of generations in the evolution.
The default value is 3000 generations.

--stop-gen <number_of_generations>

Sets the number of generations after which the terminating condition will be checked.
This option is related to the **stop-change** option.
The default value is 500 generations.

--stop-change <value>

The evolution terminates when the fitness of the best chromosome in the population does not decrease by a certain percentage after a certain number of generations. The percentage is set by this option. The value is in the interval $]0, 1]$.
The default value is 0.99 meaning that the evolution terminates when the fitness does not change by more than 1%.

--print-gen <number_of_generations>
 Sets the number of generations after which the print condition will be checked. This option is related to the **print-change** option.
 The default value is 10 generations.

--print-change <value>
 The status of the evolution may be printed when the fitness of the best chromosome in the population decreases by a certain percentage after a certain number of generations. The percentage is set by this option. The value is in the interval]0,1].
 The default value is 0.9 meaning that the status will be printed every time the fitness decreases by 10%.

--ES (<'plus'> | <'comma'>)
 Specifies the selection scheme of the evolution strategies algorithm.
 The default value is 'plus'.

--max-res <maximum_resistance>
 Sets the maximum resistance of the circuit's resistors in ohms.
 The default value is 200 k Ω .

--max-cap <maximum_capacitance>
 Sets the maximum capacitance of the circuit's capacitors in nanofarads.
 The default value is 500 nF.

--sigma-init <initial_value>
 Sets the initial value of the mutation step.
 The default value is 100.

--fitness (<'bestMatch'> | <'idealSine'> | <'maxAmp'>)
 Specifies the evaluation method of the chromosomes' fitness.
 The default value is 'bestMatch'.

--amplitude <voltage>
 Sets the amplitude of the amplifier's output waveform in volts. This option only applies if the 'idealSine' evaluation method is used.
 The default value is 1 V.

--Rload <resistance>
 Sets the resistance of the load resistor for the amplifier in ohms.
 The default value is 22 k Ω .

--max-diff <difference>
 Sets the maximal percentage difference by which the trough and peak of the amplifier's output waveform may differ. The difference is in the interval]0,100].
 The default value is 100%.

--two-stage-amp
 Instead of the single stage amplifier, the subject of the optimization will be the two stage amplifier.

The output of the application contains graphs which were presented in the thesis and a text description which form is presented in listing [A.1](#). The description contains the values

of all the optimized components of the two stage amplifier and their last mutation steps in the 270th generation. The print frequency of this description may be set via `OPTIONS`.

```
1  Generation: 270
2  objective function: 22.7797
3  R1: 151 K, sigma: 1417.68
4  R2: 27.2 K, sigma: 49.2543
5  Re: 17.0 K, sigma: 0.920929
6  Rc: 40.2 K, sigma: 20.7433
7  Ce: 415 uF, sigma: 0.161956
8  Cin: 274 uF, sigma: 43.1271
9  Cout: 162 uF, sigma: 0.00136042
10 Rgb: 5.90 K, sigma: 75.8349
11 Reb: 17.8 K, sigma: 97.2993
12 Rcb: 10.4 K, sigma: 127.525
13 R2b: 185 K, sigma: 392.402
14 R1b: 200 K, sigma: 307.757
15 Cm: 159 uF, sigma: 261.11
16 Ce2: 184 uF, sigma: 73.005
```

Listing A.1: Evolution text output example

The running application may be terminated by sending the `SIGQUIT` (`Ctrl-\`) signal. The `SIGINT` signal does not work because the ngSPICE library uses it for its own purposes.

Appendix B

CD content

The attached CD contains:

- the modified source code of ngSPICE,
- the implementation of the evolutionary algorithms in C++,
- shell scripts for performing experiments with both the single and two stage amplifiers,
- the electronic version of this document along with its source code in L^AT_EX,
- file `README.txt` which describes the compilation of the application and some other auxiliary files.