Variation-Aware Behavioural Modelling Using Support Vector Machines and Affine Arithmetic

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades
Doktor-Ingenieurin
(abgekürzt Dr.-Ing.)
genehmigte Dissertation

von Frau

Dipl.-Ing. Anna Krause

geboren am 10. Juli 1983

in Salzgitter, Deutschland

2019

1. Referent: Prof. Dr.-Ing. Erich Barke
2. Referent: Prof. Dr.-Ing. Lars Hedrich
Tag der Promotion: 2. Oktober 2019

Acknowledgements

This thesis is the result of my research at the Institute of Microelectronic Systems (IMS) at Leibniz University of Hannover. At the IMS, I was a member of the Analogue Team which focussed on advancing simulation techniques for analogue and mixed-signal circuits with parameter variations.

I want to thank my doctoral advisor, Prof. Erich Barke, for giving me the opportunity to combine my own research interest in Machine Learning with Electronic Design Automation. I want to thank him for always asking the right questions and for his support. Prof. Lars Hedrich and Prof. Holger Blume formed my thesis committee together with Prof. Erich Barke, and I would like to thank them for that.

The IMS provided a wonderful research environment and companionship that extended beyond our research. I would like to extend special thanks to Dr.-Ing. Markus Olbrich for his support and co-supervision of my research work. Then I would like to thank the members of the Analogue Team, who patiently sat through algorithm descriptions and a LOT of maths. I would like to thank all members of the IMS for their input in our research talks, coffee table discussions, and a generally great time. I would like to thank my students, Joshua Hopp and Christian Henning for their contributions.

Last but not least, I want to thank my family for their support. My sisters Maike and Lena started the family tradition of supporting thesis work with chocolates and sweets. My parents Karin and Klaus supported me all the way through my studies and my research. My husband Michael always had my back and was always there for me - even if it meant research talks in the middle of the night. I couldn't have done this without him.

Anna Krause

Abstract

AGIAS Generalised Interval Arithmetic Simulator (AGIAS) is a specialised simulator which uses affine arithmetic to model parameter variations. It uses a specialised root-finding algorithm to simulate analogue circuits with parameter variations in one single simulation run. This is a significant speed-up compared to the multiple runs needed by industrialised solutions such as Monte-Carlo (MC) or Worst-Case Analysis (WCA). Currently, AGIAS can simulate analogue circuits only under very specific conditions. In many cases, circuits can only be simulated for certain operating points. If the circuits is to be evaluated in other operating points, the solver becomes numerically unstable and simulation fails. In these cases, interval widths approach infinity.

Behavioural modelling of analogue circuits was introduced by researchers working around limitations of simulators. Most early approaches require expert knowledge and insight into the circuit which is modelled. In recent years, Machine Learning techniques for automatic generation of behavioural models have made their way into the field. This thesis combines Machine Learning techniques with affine arithmetic to include the effects of parameter variations into models.

Support Vector Machines (SVMs) train two sets of parameters: one slope parameter and one offset parameter. These parameters are replaced by affine forms. Using these two parameters allows affine SVMs to model effects of parameter variations with varying widths. Training requires additional information about maximum and minimum values in addition to the nominal values in the data set. Based on these changes, affine ε Support Vector Machine ($\hat{\varepsilon}$SVR) and ν Support Vector Machine ($\hat{\nu}$SVR) algorithms for regression are presented. To train the affine parameters directly and profit from the Sequential Minimal Optimisation algorithm (SMO)'s selectivity, the SMO is extended to handle the new, larger optimisation problems.

The new affine SVMs are tested on analogue circuits that have been chosen based on whether they could be simulated with AGIAS and how strongly non-linear their characteristic function is.

Keywords: Support Vector Machines, Parameter Variations, Behavioural Modelling, Affine Arithmetic, Uncertainty Modelling

Kurzfassung

AGIAS Generalised Interval Arithmetic Simulator (AGIAS) ist ein spezieller Analogsimulator, der Parametervariationen mittels affiner Arithmetik darstellt. Er verfügt über eine spezielle Nullstellensuche, die die Simulation von analogen Schaltungen mit Parametervariationen in nur einem einzigen Simulationslauf ermöglicht. Dies ist eine erhebliche Beschleunigung gegenüber industriell etablierten Verfahren wie der Monte-Carlo (MC) oder Worst Case Analyse (WCA). Aktuell können analoge Schaltungen nur unter bestimmten Randbedingung mit AGIAS simuliert werden. In vielen Fällen beschränkt sich die Simulierbarkeit auf einen oder wenige Arbeitspunkte. Wird die Schaltung außerhalb dieser Arbeitspunkte betrieben, so ist die Nullstellensuch numerisch instabil. In diesem Fall wachsen die Intervallbreiten ins unendliche.

Verhaltensmodellierung wurde von Forschern entwickelt, um Einschränkungen von Simulatoren zu umgehen. Viele bekannte Ansätze lassen sich nur umsetzen, wenn der Entwickler des Modells über Expertenwissen über die zu modellierende Schaltung verfügt. Die neusten Verfahren verwenden Algorithmen des Machinellen Lernens, um Modelle automatisiert zu erzeugen. Diese Arbeit kombiniert diese Algorithmen mit Affiner Arithmethik, um die Auswirkungen von Parametervariationen im Modell abzubilden.

Support Vector Machines (SVMs) trainieren zwei Parameter: die Steigung und den Offset einer Hyperebene. Diese Parameter werden durch affine Formen ersetzt. Damit ist es möglich Ausgabeintervalle mit unterschiedlichen Breiten zu modellieren. Für das Training müssen zusätzlich zu den Nominalwerten Minimal- und Maximalwerte im Datensatz enthalten sein. Basierend auf diesen Änderungen werden affine ε Support Vector Machines ($\hat{\varepsilon}$SVR) und affine ν Support Vector Machines ($\hat{\nu}$SVR) vorgestellt. Um diese SVMs zu trainieren, wurde der Sequential Minimal Optimisation algorithm (SMO) angepasst. Der Erweiterte SMO kann auf den größeren Optimierungsproblemen arbeiten und ist selektiv.

Die neuen affinen SVMs werden verwendet, um verschiedene Analogschaltungen zu modellieren. Dabei wurden Schaltungen gewählt, die mit AGIAS nur eingeschränkt simuliert werden können. Die Schaltungen sind dabei unterschiedlich stark nichtlinear.

Schlüsselwörter: Support Vector Machines, Parametervariationen, Verhaltensmodellierung, Affine Arithmetik, Modellierung von Unsicherheiten

Contents

List of Abbreviations

εSVC εSVM for Classification

εSVM SVM using the ε-insensitive loss function

εSVR εSVM for Regression

ε̂SVR ε Support Vector Machine for regression with affine parameters

νSVM SVM using the ε-insensitive loss function and minimising the maximum permissible error ε during training

νSVR νSVM for Regression

ν̂SVR ν Support Vector Machine for regression with affine parameters

AAF Lib Affine Arithmetic Mathematic Library

AC Alternating Current, analysis for determining a circuit's small signal behaviour

AGIAS AGIAS Generalised Interval Arithmetic Simulator

AHDL Analogue Hardware Description Language

ARARMAX Autoregressive Autoregressive Moving Average with Exogenous Input

ARARX Autoregressive Autoregressive with Exogenous Input

ARMA Autoregressive Moving Average

ARMAX Autoregressive Moving Average with Exogenous Input

ARX Autoregressive with Exogenous Input

BJ Box-Jenkins

BJT Bipolar Junction Transistor

BSIM Berkeley Short-channel IGFET Model

CANCER Computer Analysis of Integrated Circuits Excluding Radiation

CC Corner Case

CT Continuous-Time

CVX Package for specifying and solving convex problems

DC Direct Current, analysis for determining a circuit's operating point

DT Discrete-Time

EPD Extended Partial Deviations

FIR Finite Impulse Response

HDL Hardware Description Language

i.i.d. independent and identically distributed

IC Integrated Circuit

IGFET Insulated-Gate Field-Effect Transistor

KKT Karush-Kuhn-Tucker condition

LS-SVM Least Square SVM

MC Monte-Carlo

MISO Multiple Input Single Output

MLP Multilayer Perceptron

MNA Modified Nodal Analysis

MOS Short form of MOSFET

MOS1 MOSFET model by Shichman and Hodges

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor

NN Artificial Neural Network

OE Output Error

RBF Radial Basis Function

RF Radio Frequency

RKHS Reproducing Kernel Hilbert Space

SAG Simplification After Generation

SBG Simplification Before Generation

SDG Simplification During Generation

SISO Single Input Single Output

SMO Sequential Minimal Optimisation algorithm

SPICE Simulation Program with Integrated Circuit Emphasis

SV Support Vector

SVM Support Vector Machine

WCA Worst-Case Analysis

List of Symbols

A	Area
b	Offset of a Support Vector Machine
$\hat{b}$	Affine arithmetic offset of a Support Vector Machine
b_0	Centre value of the affine offset parameter
b_1	Weight of the affine offset parameters
ε	Maximum permissible error for the ε-insensitive loss function
e	Modelling error calculated during validation of affine models
ε_b	Noise symbol of the affine offset parameter
ε_w	Noise symbol of the affine gradient parameter
ε_k	k-th noise symbol of an affine form $\hat{x}$
e_{Test}	Test error
e_{Train}	Training error
$f(\mathbf{x})$	Support Vector Machine estimate
$\hat{f}(\mathbf{x})$	Support Vector Machine estimate of a Support Vector Machine with affine arithmetic parameters
$\mathcal{H}$	Feature space for Support Vector Learning
I	Electric Current
k	Kernel function
L	Lagrange function
$\mathcal{L}$	Loss function
l	Number of data samples for Support Vector Machine learning
$\mathcal{L}_\varepsilon$	ε-insensitive loss function
m_{ov}	Measure for the overestimation
ν	Parameter for trading off the maximum permissible error versus the empirical risk in a SVM
P	Number of parameters
$\mathbb{R}$	Set of real numbers
R	Resistance
R_{emp}	Empirical Risk
Θ	Mapping function for mapping data from input into feature space

V	Voltage
V_{in}	Input voltage
V_{out}	Output voltage
V_{th}	Threshold voltage
$\mathbf{w}$	Gradient parameter vector of a Support Vector Machine
$\mathbf{w}_0$	Centre value vector of the affine gradient parameter vector
$\mathbf{w}_1$	Weight vector of the affine gradient parameter vector
$\hat{\mathbf{w}}$	Affine arithmetic gradient parameter vector of a SVM
$\mathcal{X}$	Input space for Support Vector learning
$\hat{x}$	Affine form: affine arithmetic interval
$\mathbf{x}$	Input vector
X	Classical interval arithmetic interval
x_0	Mean value of an affine form
ξ	Slack variable for estimates larger than the original target
ξ^*	Slack variable for estimates smaller than the original target
$\xi^{(*)}$	Shorthand referring to all ξ and ξ^*
x_k	Weight of the k-th noise symbol of an affine form
$\overline{x}$	Maximum value of the interval X
$\underline{x}$	Minimum value of the interval X
$\mathcal{Y}$	Output space
y	Nominal output value
$\overline{y}$	Maximum output value
$\underline{y}$	Minimum output value

List of Figures

List of Tables

1. Introduction

In 1970 CANCER (Computer Analysis of Integrated Circuits Excluding Radiation) was introduced as a programme for circuit simulation and just two years later it was renamed to SPICE (Simulation Program with Integrated Circuit Emphasis) [54]. Although highly specialised tools had been around before, SPICE was the first all-rounder which was of practical use [77].

The early versions of SPICE suffered from several problems. In their 1974 paper, Boyle, Cohn, Pederson and Solomon state that "[b]ecause of the large number of these [semiconductor] devices in large scale IC systems, the analysis can surpass the computer's memory capability, simulator circuit-size capability, or the inherent numerical accuracy of the computer" ([8], p. 1).

Starting in the early days of SPICE, developers worked around the simulator's limitations by creating behavioural models. Boyle, Cohn, Pederson and Solomon introduced macro modelling from digital into analogue design with their famous operational amplifier model [8]. As SPICE was not able to process differential equation systems directly, macro modelling was unrivalled [36].

Since 1974 a lot has changed. On the one hand, algorithms were continuously refined, on the other hand, computer hardware advanced greatly. The PC came with enough computing power to bring simulators to every circuit designer's desk. This boosted the development of commercial simulators which soon surpassed SPICE and overcame its internal problems. The most popular SPICE-like simulators today are Cadence's Spectre and Synopsys' Saber. Those simulators come with a wide variety of analysis types and can easily simulate analogue circuits with hundreds of transistors.

SPICE-like simulators depend on mathematical models of circuits elements. These models are mainly created based on the component's physical behaviour. Device models come in different flavours which depend on the intended usage: simple models are used for manual calculations, highly detailed models are used in simulators. Simple models are for example Ohm's Law for ohmic resistors or the Shichman-Hodges MOS1[1] model for Metal-Oxide-Semiconductor Field-Effect Transistors (MOSFETs) [68]. The latest releases of the Berkeley Short-channel IGFET Model (BSIM) family, BSIM3v3, BSIM4 and BSIM6, are among the most detailed MOSFET transistor models [1, 42, 59].

With each new technology node, the size of the physical structures on a chip decreases. With smaller structures more physical effects became relevant and new effects were discovered. These new effects were included into device models and models accuracy improved greatly

[1] In 1968 this model represented the understanding of semiconductors at that time.

with deeper understanding of device physics. At the same time, model complexity grew, too. Simultaneously, the influence of process variations on the circuit's performance grew. As production of only a very few sample chips for measurement is very expensive, methods to estimate and evaluate such variations are needed. Contemporary circuit simulators provide two classes of simulation methods for this: statistical methods and methods for Worst-Case Analysis (WCA). The most common techniques are the Monte-Carlo (MC) method for statistical analysis and Corner Case (CC) for worst case analysis.

Both MC and CC analysis require many simulation runs, and are therefore not feasible for large circuits. This inspired the development of a different simulation concept: circuit simulation using specialised interval arithmetic [26]. This concept replaces parameter distributions by intervals and introduces new simulation algorithms which operate on intervals as well as on real values. The resulting software suffers from some of the early SPICE problems: algorithmic problems and low numerical stability. Challenges are strongly non-linear operations and piecewise defined functions in models.

1.1. Analogue Design

Integrated Circuits (ICs) can be divided into three classes: analogue, digital and mixed-signal circuits. They are differentiated by the type(s) of signal(s) which they process. Analogue circuits process signals which are continuous in time and value. Digital designs process signals which are discrete in time and value. Mixed-signal designs are a combination of analogue and digital circuits and process both analogue and digital signals.

Analogue design describes the activity of creating analogue circuits. It can be viewed on different hierarchical levels. Graeb suggests four design levels [27, p. 7]: system level, architecture level, circuit level, and device or process level (from highest to lowest). The system level typically refers to a very complex design, for example a hi-fi amplifier. The architecture of a system consists of different building blocks, for example different audio amplifiers. On circuit level, the design for each building block is represented by a netlist which contains electrical components such as transistors and passives. On device level, compact models for transistors are derived from doping profiles and geometric structures. Sometimes, the process level is considered a separate level which is used to simulate the manufacturing process [27].

Simulation of analogue designs often takes place on circuit level. Although contemporary simulation software can simulate around 100,000 transistors, it often takes too much time to simulate analogue systems in their entirety. To speed up simulation of analogue systems, new forms of modelling using for example so-called Analogue Hardware Description Languages (AHDLs) have been developed.

1.2. Simulation of Analogue Circuits with Parameter Variations

Circuits have two types of parameters: operating parameters and component parameters. Operating parameters determine the operating conditions of the circuit and are defined in the circuit's specification. Component parameters are the parameters of the circuit's components, e.g. nominal values of resistors.

A circuit's specification describes its function formally and formulates both the requirements and the objectives for the circuit's performances. For example, a very rudimentary specification could be:

A single-ended amplifier with an amplifier gain of at least 20 dB. *The circuit has to work with operating voltages between* 4.8 V *and* 5.2 V.

In this example, the functionality is described by giving a circuit class ("amplifier"). The requirements describe the external conditions under which the circuit has to operate successfully. There is only one requirement in the given example: correct operation for a varying operating voltage. Requirements from the specification are called operating parameters for the circuit. Operating parameters are usually represented by bounded intervals which define a minimum and a maximum value. The objectives lay down the terms for evaluating whether the circuit operates correctly. Usually, the objectives are described by one-sided bounded intervals. The example objective is a gain larger than 20 dB.

Layouts of circuits consist of various geometrical structures which define regions for doping, oxide or metal layers. From the layout, the masks for manufacturing are created. Fluctuations in the manufacturing process translate to statistical variations of process parameters such as oxide thicknesses or channel widths. In device models, these quantities influence device parameters such as the transistor gain. For simulations, statistical variations are represented by probability distribution functions. Most commonly normal and uniform distributions are used. In addition, some simulators support user defined distribution functions. These distribution functions can only be calculated by testing and examining the manufacturing process. However in practice, designers often estimate parameters of the distribution functions.

Earlier, MC and CC simulation were introduced. This section presents these two methods and introduces a third approach: simulating circuits using interval arithmetic. The class A amplifier as shown in Fig. 1.1 is used to demonstrate how these simulation methods work. To keep the demonstrations simple, only two parameters are varied: the cross-sectional area factor A of the Bipolar Junction Transistor (BJT) and the resistance of the collector resistor R_C.

1.2.1. Monte Carlo Simulation

The modern MC method was developed during the Manhattan Project [51]. Generally, the MC experiment is a stochastic method based on the strong law of large numbers. It aims

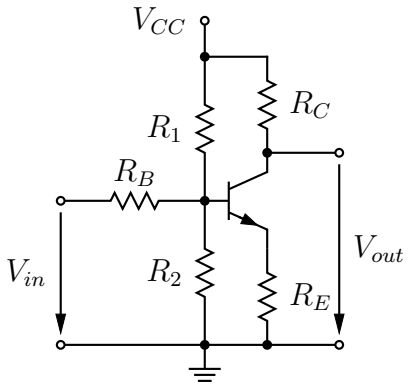


Figure 1.1.: Example circuit: class A amplifier.

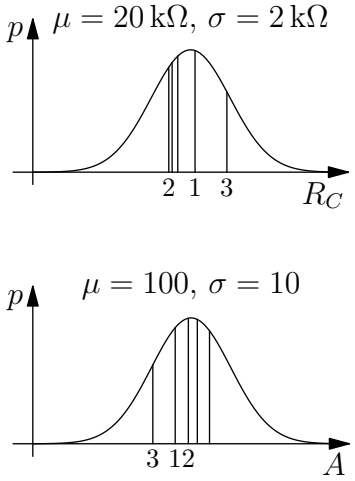
to determine numerical results for an experiment by randomly sampling the experiment's parameters and conducting the experiment a given number of times. The MC method's mathematical properties are detailed in [28]. MC experiments usually follow this pattern: first the parameters are randomly sampled following a distribution function, then the experiment is conducted and evaluated, and the results are gathered.

The MC method debuted in circuit simulations in the 1960s [37]. In circuit simulation the experiment is a circuit. The evaluation of the experiment is any analysis type, such as DC, AC or transient simulation. The experiment's parameters are component and operating parameters for which distribution functions have been specified. The designer configures the simulator and specifies the distribution functions and the number of simulation runs. For each simulation run a new parameter set is sampled from the distribution functions. Both Spectre and Saber offer MC analysis [9, 71].

In this demonstration the class A amplifier is the experiment. It is evaluated using the DC sweep analysis. The input voltage V_{in} is swept from $-0.8\,\mathrm{V}$ to $0.8\,\mathrm{V}$ in steps of $0.02\,\mathrm{V}$. The MC parameters are the resistor R_C and the area coefficient A. With only two MC parameters, the parameter set is sampled 1000 times.

Fig 1.2 shows the sequence of the MC simulations. To demonstrate the MC method a Gaussian distribution is specified for each parameter. The means and standard deviations are given in Fig. 1.2a: the mean value of the distribution function is the nominal value of the parameter, the standard deviation determines the width of the variations. Both the collector resistor R_C and the cross-sectional area factor A are varied by $10\,\%$. A new parameter sample is drawn for each simulation run. Tab. 1.2b displays the first five parameter sets. In addition, the first threes samples are marked by vertical lines in the distribution graphs and are indexed, according to the simulation run for which they were drawn. The resulting characteristic function $V_{out} = f(V_{in})$ and a histogram for the amplifier gain are shown in Fig. 1.2c.

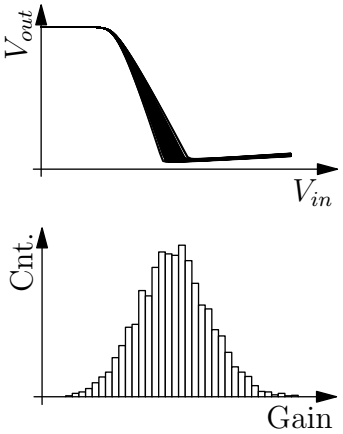
Usually, the MC analysis is implemented as a loop: after a set of parameters has been determined, the circuit is simulated, the simulation results are saved and the next MC experiment is started by sampling new parameters.



(a) Distribution functions with mean and standard deviation.

No.	R_C in kΩ	A
1	20.22	95.98
2	18.89	99.31
3	21.84	90.33
4	19.35	101.57
5	19.06	104.65

(b) Sampled parameter values for the first five MC runs.



(c) Characteristic function (top) and amplifier gain histogram (bottom).

Figure 1.2.: Monte Carlo simulations of the class A amplifier.

As the MC method is based on the strong law of large numbers: a large number of simulation runs is needed to obtain reliable information about a circuit's performance. From this data, the mean and quantiles can be calculated. In this example, the amplifier gain approaches a Gaussian distribution with $\mu = -16.04$ and $\sigma = 0.44$.

The disadvantage of this simulation type lies in the high number of simulation runs which are needed to obtain reliable data. This leads to a high overall simulation time and a high memory usage for saving the results. Furthermore, it is impossible to know beforehand how many simulation runs are needed. Although MC simulation is a powerful tool to evaluate a circuit's behaviour with parameter variations, its disadvantages prevent it from being used for large circuits and systems.

1.2.2. Worst Case Analysis

The specification of a circuit does not only define operating parameters but also performance objectives. These are usually given as upper or lower bounds or both, e.g. minimum amplifier gain or maximum power consumption. The aim of WCA is finding the worst-case performance if all component and operating parameters take an arbitrary value in their given tolerance regions. If a lower bound is specified for a performance feature, the worst case is the maximum deviation from the nominal value in negative direction. WCA calculates the minimum value of the performance feature. If an upper bound is specified, the worst case lies in direction of that bound and the maximum value of the performance feature is calculated [27].

WCA has three main types: classical, realistic and general. The three types are distin-

guished by the form of the tolerance region and the performance function. Tolerance regions can be boxes for ranged parameters and ellipsoids for statistical parameters with normal distribution, performance functions can either be linear or non-linear. Classical WCA starts from a tolerance box and a linear (or linearised) performance function. Realistic WCA uses an ellipsoid tolerance region and a linear performance function. The most general WCA uses an ellipsoid tolerance region and non-linear performance function.

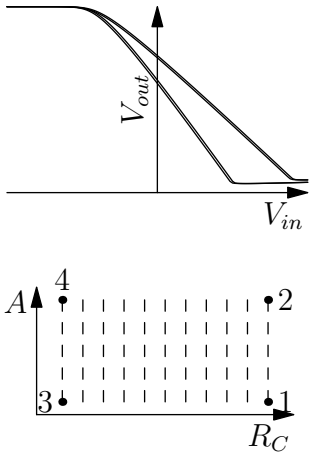
Classical WCA is the basis for CC simulation. The minima and maxima of a linear function over a rectangular parameter space are found in the corners of the parameter space. The CC simulation conducts one simulation for each corner. Given P parameters the parameter space has 2^P corners. Just like MC, CC simulation is a wrapper for standard analysis types. The designer configures the simulation and selects the parameters which are varied. The number of simulations is fixed at the number of corners of the parameter space. CC simulation is available in both Spectre and Saber [9, 71].

For this example, the two parameters are chosen to lie within an interval. These intervals are$R_C \in [16\,\mathrm{k\Omega}, 24\,\mathrm{k\Omega}]$ and $A \in [80, 120]$. The amplifier gain is evaluated for the four corners of the parameter space.

Fig. 1.3 shows the flow for a CC analysis. A parameter space spanned by two parameters has four corners. These corners are shown in Tab. 1.3a. For each of these corners, the simulation is run once, resulting in 4 simulation runs. Fig. 1.3b shows the DC transfer function for different parameter combinations at the top. The bottom shows the amplifier gain over the two parameters. In this example, the performance is linear in the parameters. Tab. 1.3c displays the gain for the four corners. The two extreme values are -13.78 as the worst performance and -17.55 as the best performance.

No.	R_E in kΩ	A
1	24.0	80
2	24.0	120
3	16.0	80
4	16.0	120

(a) Corners of the parameter space.



(b) DC transfer function (top) and parameter corners of the parameter space with the gain (dashed lines).

No.	Gain
1	-17.55
2	-17.46
3	-13.78
4	-13.81

(c) Amplifier gain for the corner cases.

Figure 1.3.: Corner Case simulations of the class A amplifier.

For circuits with only a small number of varying parameters, CC simulations run in very short time compared with MC simulations. However, the number of simulations is determined by 2^P. With the number of simulations increases exponentially with the number of parameters, the overall simulation time for the CC increases exponentially, too. Therefore, for circuits with many parameters, CC analysis is not feasible.

A second problem arises from the underlying assumptions: parameters' tolerance regions are bounded and performances are described by linear functions. In real world circuits, component parameters' variations are characterised by normal distributions which lead to ellipsoid tolerance regions. Therefore, boxed tolerance regions are at best much worse than the real worst cases. Furthermore, in many cases performance measures are not linear in the parameters. Using the CC simulation on non-linear performance measures therefore does not guarantee to return the worst cases.

1.2.3. Simulation with Intervals

In 2008, a new numerical simulation technique for analogue circuits with parameter variations was introduced [25, 26]. Instead of classical interval arithmetic it employs affine arithmetic [16], a simplification of Hansen's generalised interval arithmetic [34]. The root-finding algorithms have been adapted for calculations with affine arithmetic. Although specialised simulation tools for WCA have been developed before, this was the first all-rounder with affine arithmetic. The current simulator implementation is called AGIAS (AGIAS Generalised Interval Arithmetic Simulator).

Fig. 1.4 shows the programme flow of the simulator. It consists of a symbolic preprocessor and a numeric solver. The maple preprocessor takes the circuit's netlist and parameters and creates the equation system and the simulator configuration as well. The resulting files are parsed for numerical solving of the equation system. The core of the simulator is the so-called Extended Partial Deviations (EPD) solver [25]. Using the Affine Arithmetic Mathematic Library (AAF Lib), the EPD solver conducts the numerical simulation. The results can either be plotted using an external tool such as gnuplot, or written to text files.

The EPD solver calculates the interval-valued solution in two steps. For each time step, first the nominal solution is calculated for which all parameters take their nominal value, then the interval-valued solution is calculated. Therefore, AGIAS needs only one simulation run. Compared to MC and CC this is a huge gain in terms of simulation time and memory usage.

In AGIAS, parameters are written as affine forms, the interval representation of affine arithmetic. Affine arithmetic is described in greater detail in Sec. 2.1. Tab. 1.5a shows the two affine parameters used in the class A amplifier example. From these parameter variations the simulator calculates all possible output voltages for a given input voltage. Instead of a single DC transfer function, the solver calculates all possible transfer functions for the given parameter set. In doing so, AGIAS guarantees enclosing all possible DC transfer functions by overestimating the output intervals.

Fig. 1.5 illustrates simulation with AGIAS. Fig. 1.5b shows the affine DC transfer function.

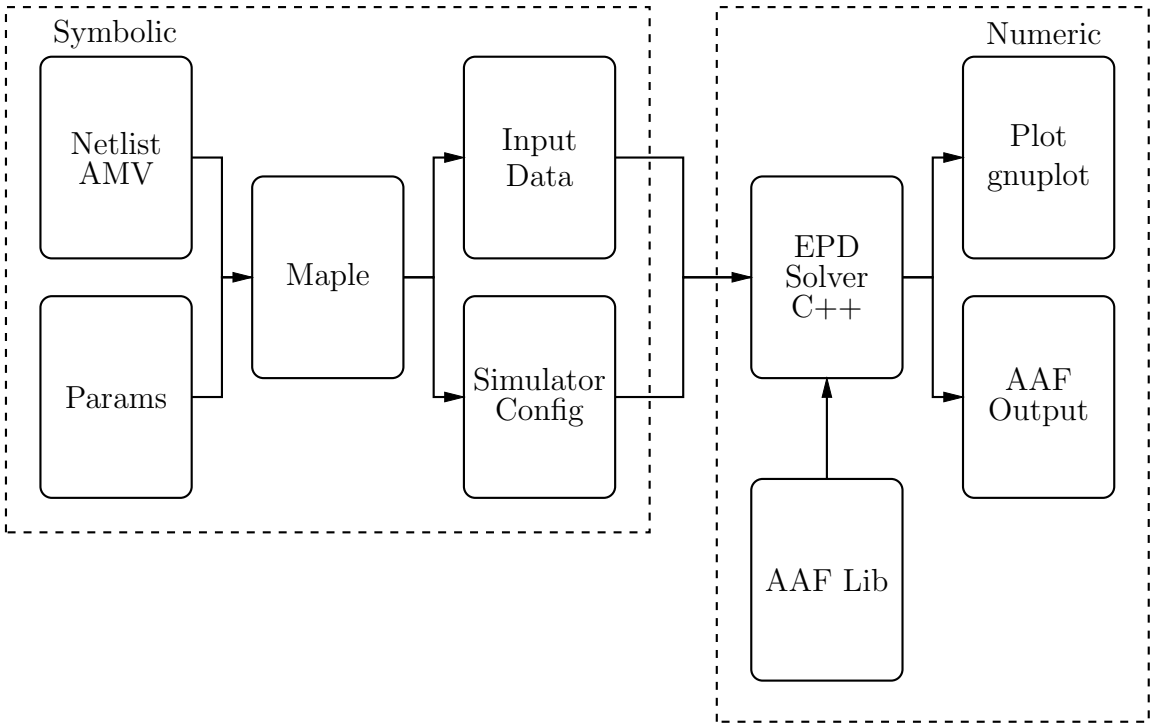


Figure 1.4.: Semi-symbolic simulation with AGIAS: the equation system for the numerical solver backend is created using Maple libraries.

AGIAS' standard for plotting affine forms is plotting the minimum and maximum values. Nominal values currently have to be obtained from an additional simulation run as they are not part of the output of an affine simulation. Nominal values are plotted as a dashed line. The class A amplifier can not be simulated with affine arithmetic over the full operation range. Simulation stops shortly before the knee and can only be resumed for much higher input voltages. Simulation is numerically unstable for settings in the area for which no minimum and maximum values are plotted.

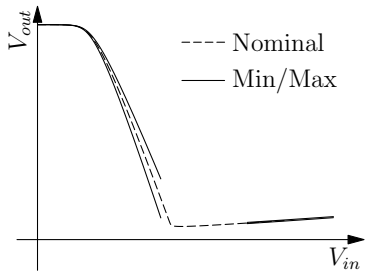
Some output values in affine form are given in Tab. 1.5c. The output voltage contains the noise symbols of both varying parameters and a third symbol ε_{err}. As common noise symbols represent correlation, the correlation between output voltage and R_C and transistor gain is given by the noise symbols' weights. The third symbol is introduced to enclose the linearisation error.

Simulating circuits with affine arithmetic is not universally possible. The root-finding algorithm is particularly sensitive to the widths of the intervals and to mathematical formulations used to describe the circuit components. Usually, semiconductor components are described by non-linear equation systems and — depending on the chosen model — piecewise functions. Both strong non-linearities and piecewise functions pose a problem for the EPD solver.

Calculations in affine arithmetic are carried out through linearisation. Therefore, large

$R_C = 20\,\mathrm{k\Omega} + 4\,\mathrm{k\Omega}\varepsilon_{RC}$
$A = 100 + 20\varepsilon_A$

(a) Interval definitions of the parameters.



(b) DC transfer function.

V_{out}	ε_{RC}	ε_A	ε_{err}
1.42	-0.56	0.0	0.07
1.38	-0.57	0.0	0.07
1.34	-0.57	0.0	0.07

(c) Affine output voltages.

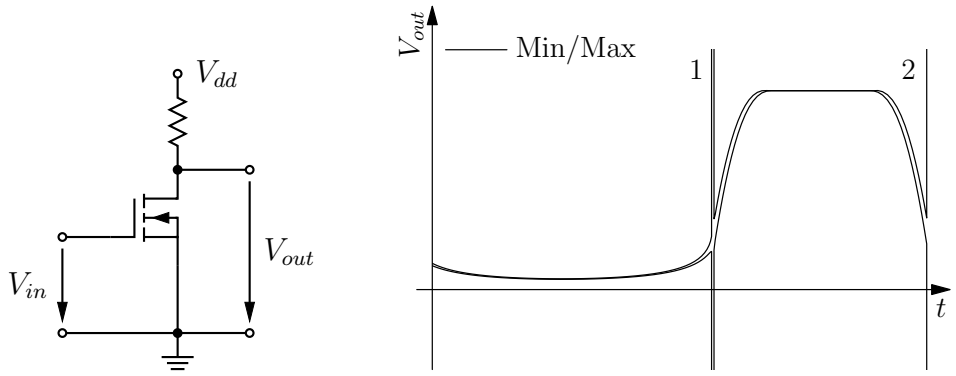
Figure 1.5.: Simulation with affine arithmetic using AGIAS.

intervals on strongly non-linear operations lead to very large overestimation. Ultimately the width of intervals approaches infinity.

Piecewise functions are implemented using Heaviside functions. Heaviside functions are evaluated using only the centre value of the affine form to determine which branch is to be used. This leads to incorrect results and very large overestimation. Ultimately, the EPD solver will fail to converge. Additional experiments revealed that the solver shows different behaviours for different branches of the Heaviside. This depends heavily on the model.

Fig. 1.6 illustrates the influence of piecewise functions. On the left another simple one-transistor circuit, an NMOS inverter, is shown in Fig. 1.6a. The input stimuli were chosen so that the NMOS transistor in the inverter operates in all regions. When switching from linear to saturation region the root-finding algorithm fails to calculate the interval correctly, which results in huge overestimation. This can be seen as two peaks in Fig. 1.6b. Using transient simulation, this example also shows that the direction of switching is important. While on the first occurrence (1), the simulator overestimates the intervals hugely switching from linear to saturation region, but converges to sensible values afterwards. When switching back (2), the simulator fails to converge and simulation terminates.

Unfortunately, AGIAS suffers from similar problems as the 1970's SPICE. It is sensitive to non-linearities and exceeds numerical capacities. In addition, the root-finding algorithm cannot handle piecewise defined functions. Strong non-linearities and piecewise defined functions are most commonly found in transistor models. Older models often contain piecewise functions, while their more contemporary counterparts are built around single non-linear equations. This leads to massive overestimation in the simulation of transistor circuits or the algorithm fails to converge. Therefore, AGIAS is not used to generate data for modelling, but only to test models in this thesis.



(a) Circuit diagram.

(b) Minimum and Maximum output voltage for sinusoidal excitation signal. (1) and (2) reference switching events in the transistor model which lead to numerical instability.

Figure 1.6.: NMOS inverter: convergence failure for piecewise functions.

1.3. Scope of this Thesis

Behavioural modelling has historically been coupled to the development of circuit simulation software. Every simulator has its limits: the number of transistors, the number of components, the numerical stability, the computing power of a PC or the robustness of the root-finding algorithms.

As circuit designers cannot improve the algorithms, behavioural modelling became their approach of choice to overcome a simulator's limitation. The introduction of the operational amplifier model was motivated by the shortcomings of the newly released SPICE [8].

Since the 1970s the meaning of "a large number of transistors" has changed drastically. In 1974, it meant "around ten", nowadays the numbers depend on the objective of the simulation. Simulating nominal circuit behaviour is generally considered an easy task and standard simulations can be performed on circuits with roughly 100,000 transistors [62]. For specialised simulation types, such as MC and CC simulation, the number of transistors can be much lower. Additionally, if the simulation takes too much time, certain analysis types are considered inapplicable to large circuits.

As AGIAS is still a very new implementation and its root-finding algorithms come with a unique set of challenges, "a large number of" means "about ten" again. Calculations in interval arithmetic can be a bit of a hassle. AGIAS' objective is to include all possible solutions in one simulation run. To achieve this, the EPD solver introduces a new affine symbol to cover additional intervals introduced by solving non-linear equations. The EPD solver will always overestimate the solution. This leads to convergence problems if the intervals are too wide.

To overcome these limitations, a new approach for behavioural modelling is proposed. Behavioural modelling is a wide field which ranges from macro modelling to high level

AHDLs. In general, creating behavioural models is a tedious task, which requires a lot of time and expert knowledge. In recent years, a lot of research has gone into automating the generation of behavioural models [53, 67]. At the core of this research were learning algorithms, that allowed automated black-box modelling of circuits.

Standard behavioural models are created for real-valued simulation methods. Therefore, most behavioural models are not variation-aware.

The focus of this thesis is to create behavioural models for AGIAS. These models have the following characteristics:

- The models are variation-aware. The key concept of AGIAS is simulating circuits with parameter variations in one simulation run. It uses affine forms as representation for parameter variations. Therefore, model parameters have to be affine forms.
- The models enclose the original circuit with high overall accuracy. AGIAS guarantees to enclose the correct results. Therefore, behavioural models have to enclose the original circuit. Enclosure can be obtained by different techniques. To preserve accuracy it is necessary to enclose the original circuit as closely as possible.
- The models do not contain piecewise defined functions and contain relatively few non-linear functions. Piecewise functions and strongly non-linear functions lead to overestimation and convergence failure.
- The models are generated at least semi-automatically.

2. Behavioural Modelling

Scientific models are theoretical or objective representations of real world features. Theoretical representations include, but are nor limited to, mathematical models, conceptual models and graphic models. Objective representations include to scale miniatures and constructions kits among others. Real world features can be anything from weather phenomena to human built machinery. Models are created to understand, simplify, visualise, or simulate a certain feature. Usually one model cannot achieve all these different goals. Therefore, for each of these tasks different types of models are used. Models can be hierarchical, meaning models can represent other models.

In electrical engineering, modelling usually refers to creating mathematical representations of real world features or other electrical models. Objective representations are quite uncommon and are only used for very specific subjects. Mathematical models in electrical engineering represent a wide variety of real world features ranging from circuit components, circuits and systems to physical effects and mechanical models. In this thesis, the focus lies on creating mathematical models of analogue circuits and systems and circuit components.

Sec. 1.1 introduced different abstraction levels which are used when discussing IC properties. These levels are not directly transferred to classify models for analogue circuits and components. In fact, there is no universal way to fulfil this task. Mantooth and Fiegenbaum suggest to classify models hierarchically according to Table 2.1.

Table 2.1.: Classification of models according to [46, p. 19].

Design Abstraction	Modelling characteristics
Primitive	Devices (MOS, BJT, diode, etc.) represented by: - Analytical equations - Tables
Functional	Macromodels derived by: - Circuit simplification - Circuit build-up - Symbolic Methods - Combinations of the above
Behavioural	High Level language descriptions - Linear and non-linear mathematical equations - Tables

They introduce three design abstraction levels: primitive, functional and behavioural. Primitive models are highly detailed models used in transistor level simulation. All models representing semiconductor devices are primitive models. In analogue designs, primitives are used on circuit level to create circuits. Models on functional level are called macro models. Macro models are used instead of the detailed circuit schematics. They are created from idealised electrical components and usually have some similarity to the circuit they represent, e.g. Boyle's operational amplifier model [8]. These models can be found on architecture level and system level. Models on the behavioural level are arbitrary mathematical or logical function blocks. These models are created using equations or procedural descriptions, e.g. Hardware Description Languages (HDLs). Behavioural models usually do not bear any similarity to the topology of the modelled circuit. They can also be found on both architecture and system level. Mantooth and Fiegenbaum's classification does not cover any models for the device/process level as given in Sec. 1.1, as these models are not used in circuit simulation.

Another approach to formalising and classifying are Y-charts. In 1983, Gajski and Kuhn introduced the Y-chart for characterising and formalising design methodologies for digital designs [21]. Y-charts aim at incorporating every aspect of digital design and offer a universal language to describe different design flows. Therefore, they can also be used to classify different modelling methodologies. However, they are not directly applicable to analogue design methodologies. The Y-chart was adapted for analogue design in 1994 by Hosticka et alii [35]. Y-charts distinguish three different views and several levels which in Y-chart terminology are called layers.

SPICE-like simulators are built to simulate conservative electrical systems which are constructed from lumped elements and which are described either as I-V or Q-V equation systems. Therefore, models used with these simulators have to be formulated as I-V or Q-V equation systems. These restrictions can be avoided to a certain extent by using dependent sources which allow arbitrary mathematical operations to model the relation between input and output signals. Modelling for AGIAS comes with additional constraints: AGIAS specialises in simulating circuits with parameter variations and uses affine arithmetic to represent these variations. If a model's parameters have physical meaning, they can directly be described using affine forms. If not, parameter variations have to be taken into account during model creation.

2.1. Affine Arithmetic

Affine arithmetic was introduced by de Figueiredo and Stolfi in the mid-1990s with a first full overview in [16]. Standard interval arithmetic describes intervals by their lower and upper bounds

$$X = [\underline{x}, \overline{x}] \qquad \underline{x}, \overline{x} \in \mathbb{R} \tag{2.1}$$

In classical interval arithmetic, the basic assumption for calculations is that all intervals are uncorrelated. This leads to huge widening of intervals, sometimes called interval explosion.

Intervals in affine arithmetic are called affine forms:

$$\hat{x} = x_0 + \sum_{k=1}^{N} \varepsilon_k x_k \tag{2.2}$$

Each affine form is denoted by a hat over the variable name and consists of a mean value x_0 and a sum of weighted noise symbols ε_k with weights x_k. Affine arithmetic does not assume that intervals are uncorrelated, instead affine forms can be linearly correlated. Each affine form can be constructed from an arbitrary number of noise symbols. Two affine forms are correlated, if they share the same noise symbol with non-zero weight. If the weights have the same sign, the affine forms are positively correlated, else they are negatively correlated. Affine forms can be converted to classical intervals by using the radius. The radius is defined as

$$\mathrm{rad}(\hat{x}) = \sum_{k=1}^{N} |x_k| \tag{2.3}$$

The classic interval to represent the affine form then is

$$X = [x_0 - \mathrm{rad}(\hat{x}), x_0 + \mathrm{rad}(\hat{x})]. \tag{2.4}$$

Mathematical operations on affine forms can be divided into two groups: affine operations and non-affine operations. Affine operations on affine forms return affine forms, non-affine operations do not. To calculate the result of non-affine operations, the operation has to be linearised. This linearisation introduces problems to numerical simulation: overestimation and loss of correlation information. Overestimation causes intervals to grow steadily. This can lead to violation of functions' domains and can therefore render a given problem non-solvable. The linearisation error is incorporated into the affine form as a new noise symbol at the expense of the correlation information. Two algorithms for calculating non-affine operations are presented in [16].

2.2. Physical Modelling

Physical modelling is the activity of creating models of electronic devices based on equations which accurately describe the complex physical mechanisms in the device. The resulting models are usually called device models. Progression of technology usually means scaling down the channel width of MOSFETs. This introduces new physical mechanisms into the models and increases the complexity of the model. Physical models can be put into three categories: analytical models, table look-up models and empirical models [2].

Analytical models are derived directly from device physics. This group can be sub-classified into charge sheet models and compact models. Charge sheet models are based on surface potential analysis. The basic assumption is that the inversion region is of infinitesimal thickness [2, p. 238-242][74, p. 131-140]. As the complete charge model is too complicated, a simplified charge sheet model has been introduced by Tsividis [74, p. 140-146].

Usually, charge sheet models are not used in circuit simulation. Compact models are created from charge sheet models by using different equation systems for different operation regions and by removing physical phenomena of lesser significance [17, 47]. While in earlier models switching through operation regions is represented by piecewise defined functions, contemporary models are built using smoothing functions. Almost all models used for transistor level simulation are compact models.

For table look-up models, the device currents for different geometries and bias points are stored in tables [69]. There are different approaches for creating the data base for these models: using a dedicated device simulator, measurements of devices or approximating the currents by using splines. Table look-up models are used in so-called Fast-SPICE simulators [50]. Using table look-up models requires some interpolation to obtain values that are not explicitly part of the look-up table.

Empirical models are created based on experimental data. For these models, components have to be measured. Then, the model equations are created using curve fitting algorithms [78]. These models are very rarely used in simulators and are not discussed further.

The number of effects which are modelled is not only determined by the developer but also by the number of effects known at the time the model is created. Therefore, compact models represent the understanding of semiconductor physics at the time of their creation. This can be demonstrated by examining the development of MOSFET models [11]: The first compact MOS model was the MOS1 model by Shichman and Hodges [68]. It is based on 1970's device physics for long-channel and uniform-doping effects. The model equations are very simple and are still used for manual calculations. Over the following years, different models were published which advanced MOS 1 little by little. The next major step forward was the BSIM1 model which included improved short–channel effects and gives good results for channel lengths above 1 µm. The BSIM3 models include short-channel, narrow width and high field effects [42]. In the subsequent versions of this model, accuracy was improved and a single-equation approach was introduced. The latest members of the BSIM family are BSIM4 and BSIM6. BSIM4 extends the existing models into the sub-10 nm regime and introduces better RF handling [59]. BSIM6 changes the core structure from threshold voltage based to charge based [1].

For BJT the same is true for the Ebers-Moll model and the Gummel-Poon model. The Ebers-Moll model was published in 1954 [19] and was limited to DC behaviour with only rudimentary modelling of transient behaviour. The Gummel-Poon model was published in 1970 [31]. Simplifying and idealising the Gummel-Poon model results in the Ebers-Moll model.

Contemporary compact models are the most accurate models used in circuit simulation. They model dynamic, non–linear large signal behaviour. The models are formulated as differential equation systems which only include derivatives with respect to time. Older models use piecewise-defined functions, newer models use smoothing functions. Compact models always abide by Kirchhoff's and Ohm's laws. As the parameters of these models have physical meaning, the models can be used for statistical and worst case evaluation.

For the same reason it is also very easy to use intervals instead of distribution functions. However, as the equation system contains strongly non-linear and piecewise defined functions, numerical calculations with intervals often fail. Compact models can only be created with expert knowledge about semiconductor physics. Therefore, there is no automation for this approach.

2.3. Equivalent Circuit Models

Equivalent circuit models are models which reproduce the behaviour of a circuit or semiconductor device by combining idealised electrical components. These models can be divided into two classes: equivalent circuit device models and macro models.

Equivalent circuit device models are based on numerical data which describe device characteristics or on mathematical device equations. Although developed as physical models (see Section 2.2), the Ebers-Moll and Gummel-Poon models are prominent examples for equivalent circuit models [23]. Getreu describes different equivalent circuit representations of the models' equation systems showing that the equivalent circuit representation of an equation system is not unique [23, p. 10 et seq.]. Another commonly used example are small signal equivalent circuits. Approaches for the automatic generation of equivalent circuit models from device simulations are sparse. One approach has been presented by Pacelli et alii in 2000 [58]. This method first partitions a device into functional regions and then creates one circuit block per functional region.

Macro modelling is one of the oldest modelling approaches for modelling analogue circuits. It has been imported from modelling digital circuits by Boyle et alii with their famous model of an operational amplifier [8]. Macro modelling is closely related to circuit design. It even borrows its basic approaches from circuit design: the top-down and the bottom-up approach from circuit design become circuit simplification and circuit build-up in macro modelling respectively [46].

Circuit simplification starts with the original circuit and simplifies it by removing unnecessary characteristics. The most common simplifications are removing circuit elements and replacing complex device models with simpler models. Usually, a circuit component contributes to several characteristics. Therefore, removing components can damage characteristics which are important to the model. Removing components does not result in a high degree of simplification. Much better results can be obtained by replacing complex models with simpler models, e.g. replace BSIM3 by MOS1 models. Replacing models allows designers to remove characteristics from a component without removing the respective component from the circuit.

Circuit build-up produces a configuration from ideal elements. The objective of this approach is to reproduce the terminal characteristics. The resulting circuit configuration does not necessarily resemble the original circuit. Creating a model via circuit build-up, the designer first has to identify the circuit performances and has to select the electrical characteristics for the model. The second step is to partition the characteristics in function

blocks, e.g. amplification and low-pass behaviour. Creating the function blocks is the last step. Commonly, first the ideal behaviour is modelled and then the non-ideal behaviour is added on top of that.

Macro models consist of several stages: input, output and main function stage. The in- and output stage model the respective in- and output terminal characteristics. The main function stage represents the transfer function and all non-linear effects. This stage usually consists of several function blocks [12, p. 16]. It is also possible to have additional stages between main function stage and output stage, e.g. stages to model transient behaviour.

Both macro modelling techniques are not mutually exclusive, they can be combined. Boyle's operational amplifier model is an early example for a combined approach: the input stage was created using circuit simplification, the rest of the model was created using circuit build-up [8, 46].

Built mainly from idealised components, macro models adhere to Kirchhoff's and Ohm's laws and can easily be simulated by standard simulators. Equivalent circuit models therefore have similar properties as circuits: equivalent models can model dynamic, non-linear large signal behaviour. Depending on the components' models the resulting equation system can contain piecewise-defined functions. Creating macro models requires expert knowledge. Macro models can be used for statistical and worst case analysis under certain conditions: parameters have to retain physical meaning for the circuit. This is the case for models obtained from circuit simplification or models from circuit built-up which have the same topology as the original circuit. There is no universal automation approach for creating macro models, but there are approaches to semi-automatically generate models of certain circuit types, e.g. operational amplifiers.

2.4. Symbolic Modelling

Symbolic modelling is directly related to symbolic simulation and symbolic simplification. The objective of symbolic simulation is to obtain an analytic, closed-form expression for the system transfer function [24] or determine the relation between input and output of a system in time domain [6]. Symbolic simulation gives designers a different perspective from numerical simulation: it helps gaining better understanding of a circuit's behaviour. The resulting equations can be used as a behavioural model.

Usually, equations determined by symbolic simulation feature too many terms to obtain insights into the circuits behaviour [33]. Therefore, different simplification algorithms have been developed. These algorithms form a family of hybrid symbolic/numeric algorithms for expression simplification. The basic procedure is the same: the designer chooses a numerical reference simulation and an error bound. The algorithm then applies simplifications to the symbolic expression and simulates the expression numerically. If the error exceeds the error bound, the simplification is cancelled. This is repeated until no more simplifications are possible [33]. Although the result of a symbolic simulation can be used as a behavioural model, usually simplification algorithms are applied before the equations are used as model.

There are different simplification methods for linear and non-linear systems. For linear systems there are three types: simplification before, during and after generation. Simplification Before Generation (SBG) simplifies the circuit before the symbolic simulator is started. Symbolic simulators support different forms of circuit description and there are different SBG methods for each. SBG algorithms remove circuit components or replaces complex device models with simpler ones [18], simplified graphs [14, 15] or remove elements from matrices [55]. This approach is also associated with manual evaluation [20]. Simplification During Generation (SDG) is mainly done by simplifying graphs using the Two-Graph Method [48, 55]. Different methods based on sorting spanning trees have been published, e.g. in [79]. One alternative to sorting spanning trees is a method based on sensitivity analysis [81]. The transfer function is usually given in a sum-of-products formulation. Most Simplification After Generation (SAG) approaches evaluate the influence of single products on the complete transfer functions in a given nominal point. If a product contributes less than a predefined value ϵ_0, it is eliminated [81].

Simplification procedures for non-linear systems are sparse. Weakly non-linear systems are approximated using Volterra-series [4, 80]. In these approaches usually second and third order polynomials are used [55]. A second approach uses place holders to represent non-linear components. After the transfer function has been generated, the place holders can be replaced by arbitrary non-linear functions [49]. SBG methods for linear systems have also been applied to non-linear systems. The approach presented by Borchers [5] operates on the symbolic equation system. This approach simplifies iteratively. First one term is removed from the equation system, then numerical simulation is used to determine the impact of the removal. If the impact is smaller than an a priori defined error bound, the term is removed permanently. Symbolic simulation is then carried out after the simplification algorithm has terminated. Other approaches suggest using ideal circuits components as every piecewise linear formulation can be transformed into a linear equations system [45] or approximation with posynomial functions [15].

The results of symbolic simulations can be used as behavioural models. The transfer functions represent non-linear, large signal behaviour and include dynamics as well. Although based on conservative systems, the models are not conservative. Depending on the components' mathematical models, the resulting model function can contain piecewise defined functions. Creating symbolic models requires the same amount of knowledge as designing a circuit. The parameters of a symbolic model are the parameters of its components. The parameters have physical meaning and the model can directly be used for statistical and worst case analysis. Symbolic modelling can be considered a semi-automated approach to modelling as the circuit has to be designed by hand, but the resulting model is determined automatically by the simulator. Symbolic models are transfer function models which have to be wrapped before using them with a SPICE-like simulator.

2.5. Classical System Identification

System theory defines systems as objects which operate on signals. Signals are defined as the representation of information [75]. Systems produce an observable signal. These observable signals are usually called outputs. Systems can also be affected by external stimuli. If these stimuli can be manipulated by the observer, they are called inputs. Others are called disturbances. Some disturbances can be measured directly, others can only be observed by their influence on the system's output [43].

Although most identification approaches are designed for Discrete-Time (DT) systems, modelling analogue circuits requires Continuous-Time (CT) models. There are two basic approaches for creating CT models: the indirect approach which first determines a DT model and then transforms it to CT and the direct approach which creates the CT model directly from the data [22].

The prerequisites for identification of CT and DT models are the same: a data set, a model structure, a criterion of fit between data and model and a way to evaluate resulting models. The data set is determined experimentally. The model structure – that is the architecture, model order, the representation of dynamics, and the interface – are defined by the designer. The model is trained automatically. Training means the parameters of the model are determined by solving an optimisation problem. After training the model is validated using the fit criterion. If validation fails, one or more of the prior steps (data generation, model structure selection, training) have to be revised [82, p. 2 et seq.][56, p. 7 et seqq.]. A short remark on the interface: when creating data based models, the designer chooses the inputs and outputs for the models. Training models with multiple outputs is more difficult and therefore, often Multiple Input Single Output (MISO) models are used.

Classical DT system identification offers two main modelling architectures: transfer function models and state-space models. The most general transfer function model is represented by

$$y(k) = \mathbf{G}(q)\mathbf{u}(k) + \mathbf{H}(q)\mathbf{v}(k) = \frac{\mathbf{B}(q)}{\mathbf{F}(q)\mathbf{A}(q)}\mathbf{u}(k) + \frac{\mathbf{C}(q)}{\mathbf{D}(q)\mathbf{A}(q)}\mathbf{v}(k) \tag{2.5}$$

where y is the system output, $\mathbf{u}$ are the system inputs and $\mathbf{v}$ are the disturbances[43]. $\mathbf{G}(q)$ is the input transfer function, $\mathbf{H}(q)$ is the noise transfer function and q is the forward shift operator. The transfer functions are represented as rational functions which are created from five polynomials $\mathbf{A}(q)$, $\mathbf{B}(q)$, $\mathbf{C}(q)$, $\mathbf{D}(q)$ and $\mathbf{F}(q)$. Depending on which of these polynomials are used, different special models can be derived. Table 2.2 presents the most common special cases for dynamic linear Single Input Single Output (SISO) models. These can be expanded to non-linear problems by replacing the polynomials by arbitrary non-linear functions.

In state space models the relationships between input, noise and output signals are described by first order differential or difference equations [43]. State-space models are especially useful for continuous time models of physical systems. In general, it is much easier to incorporate prior knowledge about physical connections into state-space models than into

Table 2.2.: Some common black-box SISO models as special cases of Eq. (2.5.)[43, p. 88].

Polynomials Used in Eq. (2.5)	Name of Model Structure
B	FIR (Finite Impulse Response)
AB	Autoregressive with Exogenous Input (ARX)
ABC	Autoregressive Moving Average with Exogenous Input (ARMAX)
AC	Autoregressive Moving Average (ARMA)
ABD	Autoregressive Autoregressive with Exogenous Input (ARARX)
ABCD	Autoregressive Autoregressive Moving Average with Exogenous Input (ARARMAX)
BF	OE (Output Error)
BFCD	BJ (Box-Jenkins)

transfer function models. State space models are described by the following equation system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) \tag{2.6}$$

$$y(k) = \mathbf{C}^T\mathbf{x}(k) + \mathbf{D}\mathbf{v}(k) \tag{2.7}$$

with $\mathbf{x}(k)$ as the states and y, $\mathbf{u}$ and $\mathbf{v}$ as in transfer function models.

Both transfer functions and state-space models can be used to model non-linear systems. In transfer function models the two linear transfer functions $\mathbf{G}(q)$ and $\mathbf{H}(q)$ are replaced by non-linear functions. In state-space models the equations for calculating the change of the state and the output are also written as non-linear equations.

Training requires solving different optimisation problems for linear and non-linear models. For linear optimisation problems, different variants of the least squares method are used. For non-linear optimisation problems, a variety of methods is available. There are two types of methods: local and global optimisation methods. Local methods converge to a local optimum, while global methods converge to a global optimum. For linear dynamic system identification, recursive algorithms are used because they are computationally less demanding.

Dynamics can be modelled by including them directly into the model or by modelling them externally [56]. Models with internal dynamics are often non-linear state-space models. These models are trained using the backpropagation-through-time algorithm [56]. For external dynamics, the model is split into a linear dynamic and non-linear static block. If the linear dynamic filter is followed by the static block, the system is called Wiener system. If the blocks are connected the other way round, it's a Hammerstein system. Wiener and Hammerstein systems are created using non-linear optimisation algorithms.

One method to generate a CT model directly from the data is the state-variable filter method. This method applies a minimum-order filter to a transfer function model before

estimating the model parameters. The filter $K(q)$ is characterised by

$$K(q) = \frac{1}{(p-\lambda)^n} \tag{2.8}$$

where the order is determined by n and the bandwidth by λ. Both parameters are chosen a priori. Although system identification is usually done in discrete time, CT approaches bring some advantages. They operate on non-uniformly sampled data, it is easier to preserve a priori knowledge and models do not suffer from the underlying assumptions which have to be made when transforming a DT model to a CT model. Beside the state-variable filter, there are also linear filter methods, integration based methods and modulating function methods [82].

Models from classical system identification are black-box models. They are generated from data which have been obtained from simulation or measurement. Overall, black-box models require little a priori knowledge. Both transfer function models and state-space models are generally not conservative models. Classical system identification offers algorithms to identify non-linear dynamic systems. Non-linearities can be modelled internally and externally, which calls for different optimisation algorithms. The parameters in the models do not have physical meaning. This makes stochastic or worst-case analysis difficult. System identification approaches can be automated.

2.6. Artificial Neural Networks

The (human) brain is an extremely powerful structure for tasks such as information processing and pattern recognition. It is capable of learning and adapting. Artificial Neural Networks (NNs) are mathematical models which were inspired by the (human) brain.

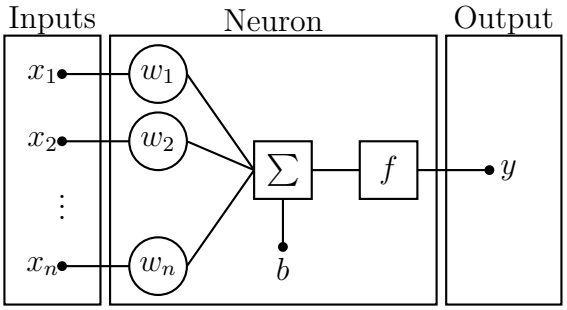


Figure 2.1.: General structure of a neuron with n inputs and 1 output.

Fig. 2.1 shows a neuron, the central calculation structure of a NN. Each neuron has a fixed number of inputs $\mathbf{x}_i, i \in 1 \ldots n$, a bias b, a transfer function f and an output y. The inputs are weighted by w_i, summed with the bias and then fed into the transfer function to calculate the output

$$y = f(\sum_i w_i x_i + b). \tag{2.9}$$

Common transfer functions are the linear function, the step function and the sigmoid function [32].

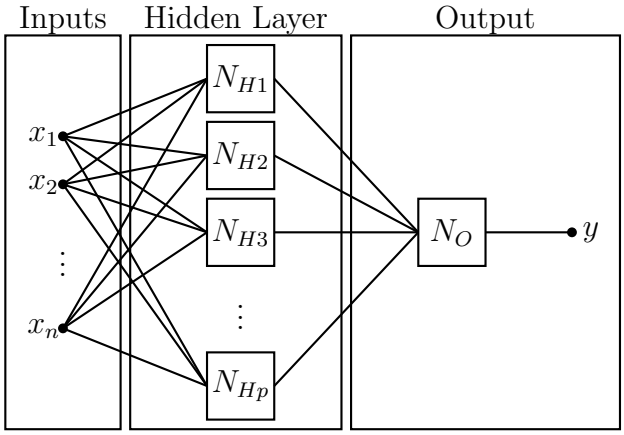


Figure 2.2.: Structure of a minimal Multilayer Perceptron type Artificial Neural Network.

Although neurons can be connected in arbitrary ways to form a NN, usually layered structures are preferred. They limit the number of possible connections between neurons. Multilayer Perceptrons (MLPs), the most common network architecture, use layers and directionality. MLPs consist of at least three layers: one input layer, one output layer N_O and at least one hidden layer N_H. The MLP in Fig. 2.2 has the minimum number of layers with one hidden layer. The signal flow direction is from input to output. Neurons can only be connected to neurons in the next layer and are usually connected to all neurons in the next layer: all inputs are connected to all neurons of the hidden layer and are not connected to neurons of the output layer. Neurons cannot be connected to other neurons in the same layer: the neurons of the hidden layer are not connected to each other.

Usually the input layer does not contain any neurons, its only task is to pass the inputs on to the first hidden layer. Neurons in the hidden layers usually use non-linear transfer functions for regression. The output layer contains one neuron per output with a linear transfer function. Each hidden layer of the MLP can be constructed from an arbitrary number of neurons. The layers do not need to have the same number of neurons and the neurons of one layer do not need to have the same transfer function. Usually, each output of one layer is connected to all neurons of the following layer. Before the advent of Deep Learning, MLPs with one hidden layer were the most common, while MLPs with more than two hidden layers were considered exotic [56].

MLPs are feedforward networks and can only model static behaviour as presented so far. For dynamic behaviour, both external and internal approaches can be used. Employing the external approach, NNs can be used as the static non-linear approximator in a Wiener or Hammerstein model.

A second option is using recurrent networks with tapped delay lines. Tapped delay lines are filter structures in NNs. Recurrent networks distinguish themselves from feedforward

networks by at least one feedback loop. Two types of feedback are commonly used: global feedback and local feedback [56]. Global feedback is an alternative approach for external dynamics: the output is fed back to the model as input via tapped delay lines. Local feedback is an approach for modelling dynamic behaviour internally. In local feedback models, the feedback is restricted to links or the neurons themselves. Nelles distinguishes three types of local feedback: local synapse[1] feedback, local activation feedback, and local output feedback [56, p. 648 et seq.].

When creating NNs, the designer defines the number of in- and outputs and the architecture by configuring the number of hidden layers and the number of neurons on the hidden layers. For each neuron, the transfer function must also be determined beforehand. Then, a backpropagation algorithm determines all weights and biases of the NN.

For creating NNs, little knowledge about the circuit which is to be modelled is needed. NNs can model non-linear dynamic systems applying either approaches which create external or internal dynamics. The designer has full control over the topology: the number of neurons and the configuration of the network as well as the transfer functions of the neurons. The parameters, the weights and biases, are determined by an optimisation algorithm. There are approaches that also apply optimisation algorithms for creating the topology. As the parameters do not have physical meaning, stochastic or worst case evaluations of the model are not possible.

2.7. Fuzzy Systems

Fuzzy logic was introduced as an extension of classical Boolean logic by Zadeh in 1965 [83]. While Boolean logic assigns each variable either the value 0 (false) or 1 (true), fuzzy logic allows one to assign every value in the interval $[0, 1]$. With this convention, fuzzy logic is very close to the vague formulations usually used in human communication.

Fuzzy systems take real-valued inputs and calculate real-valued outputs by using fuzzy sets. This requires three blocks: fuzzification, inference mechanism and defuzzification [41]. Fuzzification converts real values to fuzzy sets by calculating the degree of membership. The inference mechanism uses a set of rules to calculate the output fuzzy sets. Defuzzification combines these fuzzy sets and calculates the real-valued output [7, 56]. The two basic concepts of fuzzy systems are fuzzy sets and rules.

Fuzzy sets are created for linguistic values of linguistic variables. A linguistic variable is a word which describes a physical quantity and linguistic values characterise linguistic variables. For example *height* is a linguistic variable and *tall* and *short* are linguistic values. For each linguistic value, one fuzzy set is created. In contrast to conventional set theory, numbers can have partial membership in a fuzzy set. Membership functions are used to calculate the degree of membership of a fuzzy set. Formally, a fuzzy set M can be written as

$$M = \{(x, \mu^M(x)) : x \in \mathcal{X}\} \tag{2.10}$$

[1] A link between neurons.

where $\mathcal{X}$ is the universe of discourse, the set of numbers on which the variable is defined, x is a number and $\mu^M(x)$ is the membership function of the fuzzy set M. A membership function maps $\mathcal{X}$ to $[0, 1]$. Common membership functions are triangular functions, Gaussian functions and singletons [41]. Partial membership is expressed by the degree of membership which is determined by evaluating the membership function for a given input value x. The degree of membership is needed to evaluate the rules.

Rules for fuzzy systems are closely linked to if-then-clauses which are commonly used in human communication. The rules map a condition for the system's input to a consequence for the output. Conditions for different operators are combined using operators such as AND or OR. For these operators, several definitions exist [60]. The degree of firing μ_{R_i} describes the certainty with which a rule R_i is fired, this is the extent to which it is taken as true.

The most common fuzzy models are the Mamdani [44] and Takagi-Sueno [72] fuzzy systems. They share the fuzzification algorithm, but apply different rules for inference and different principles for defuzzification.

Mamdani fuzzy systems use rules of the form

$$R_i : \text{IF } u_1 = A_{i1} \text{ AND } u_2 = A_{i2} \ldots \text{ AND } u_K = a_{iK} \text{ THEN } y = B_i, \tag{2.11}$$

where u_k denotes the K inputs, y is the output, i indexes the rules and A_{iK} denotes the fuzzy set which is used. The degree of firing for each rule is used to define a recommendation for each output $\mu_{B_i,rec}$.

There are two algorithms for defuzzification: centre of gravity and centre average. These calculation rules accumulate the recommendations from all rules and determine the real-valued output. The centre of gravity method calculates the weighted average of the centres of gravity of all recommendations $\mu_{B_i,rec}$. The only difference for the centre average method are the weights: this method uses the degree of firing as the weight.

Singleton fuzzy systems simplify the defuzzification stage of Mamdani systems by simplifying the output fuzzy sets to singleton fuzzy sets. The rules have the following form:

$$R_i : \text{IF } u_1 = A_{i1} \text{ AND } u_2 = A_{i2} \ldots \text{ AND } u_P = a_{iP} \text{ THEN } y = s_i, \tag{2.12}$$

where s_i represents a single value.

Takagi-Sueno fuzzy systems use rules in the form

$$R_i : \text{IF } u_1 = A_{i1} \text{ AND } u_2 = A_{i2} \ldots \text{ AND } u_P = a_{iP} \text{ THEN } y = f_i(u_1, \ldots, u_p), \tag{2.13}$$

where f is an arbitrary function. Most commonly a linear function is used. The output for this system is calculated as

$$y = \frac{\sum_i f_i(\mathbf{u})\mu_{R_i}}{\sum_i \mu_{R_i}} \tag{2.14}$$

where $f_i(\mathbf{u})$ is a memoryless function, often a linear combination of the inputs, and μ_{R_i} is the degree of firing. As the rules of the Takagi-Sugeno fuzzy systems calculate a real value as a consequence, the defuzzification stage and inference stage overlap.

Fuzzy systems can be used as approximators for modelling dynamics externally. However, there is also the possibility to model dynamics internally: past outputs and inputs can be included into rules, too. Lilly [41] presents a method for creating dynamic models based on Taqkagi-Sugeno systems and plant fuzzy systems. The consequence of each rule has the form

$$\dot{\mathbf{x}}(t) = A_i\mathbf{x}(t) + b_i\mathbf{u}(t) \tag{2.15}$$

where $\mathbf{x}$ denotes the system states and $\mathbf{u}$ the inputs to the system

Although fuzzy systems are a black-box approach, the creation of a fuzzy system requires expert knowledge to infer the fuzzy sets and fuzzy rules. An automated form of creating fuzzy systems is obtained by adopting algorithms from NNs for optimising the system. The resulting systems are called neuro-fuzzy systems [56]. Dynamics are usually modelled externally. Fuzzy systems generate transfer function models which are not conservative. The parameters of a fuzzy system do not have physical meaning. Fuzzy systems can contain piecewise defined functions for fuzzyfication, but these can be avoided by the designer. Fuzzy systems are popular in modelling control systems.

2.8. Support Vector Machines

Support Vector Machines (SVMs) are a relatively young topic in machine learning and statistical learning theory. They were first published by Vapnik and Cortes in 1995 [13]. Although they were originally developed for classification and pattern recognition problems, algorithms for regression problems followed quickly. SVMs do not suffer from the curse of dimensionality. They have been successfully used in modelling circuits [52, 66].

In classification, the task for SVMs is to separate data into two classes by finding the optimal separating hyperplane. The optimal separating hyperplane is the one that separates the data with maximal margin [76]. Pattern recognition algorithms are separated into two classes: hard margin classifiers and soft margin classifiers. Hard margin classifiers attempt to find a hyperplane which classifies all members of the data set correctly. On real world data, a hard margin separator can rarely be found. In this case soft margin classifiers are used. Soft margin classifiers allow for classification errors and penalise misclassifications using a loss function. SVMs for regression are closely related to soft margin classifiers.

Vapnik's SVMs estimate functions

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{2.16}$$

on independent and identically distributed (i.i.d.) data $(\mathbf{x}_i, y_i)$ with $i = 1, 2, \ldots, l$ with $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in [-1, +1]$ for classification and $y_i \in \mathbb{R}$ for regression. $\mathbf{x}_i$ are the inputs and y_i are the outputs of the modelled system.

The optimisation problem for determining the parameters $\mathbf{w}$ and b is constructed by minimising the empirical risk

$$R_{emp} = \sum_{i=1}^{l} E_i \tag{2.17}$$

where E_i is the estimation error which is calculated by a loss function. Most commonly the linear ε-insensitive loss function is used. The SVM algorithms developed by Vapnik are called ε-SVMs for using this loss function to penalise errors. The ε-insensitive loss function is defined as

$$\mathcal{L}_\varepsilon = |y_i - f(\mathbf{x}_i)|_\varepsilon = \max\{0, |y_i - f(\mathbf{x}_i)| - \varepsilon\} \tag{2.18}$$

where y_i is an original value, $f(\mathbf{x}_i)$ is an estimate and ε is the maximum permissible error. The maximum permissible error is chosen a priori and controls the accuracy of the model [76].

In 2000 Schölkopf et alii published a variant of the ε-SVM which minimises the maximum permissible error additionally. They introduce a parameter ν which realises the trade-off between the maximum permissible error ε against the empirical risk R_{emp}. The resulting algorithm is called νSVM [65].

Least Square SVMs (LS-SVMs) follow a slightly different approach [70]. They introduce a small change to Vapnik's original algorithm: they change the function for calculating the empirical risk to

$$R_{emp,2} = \sum_{i=1}^{l} E_i^2 \tag{2.19}$$

and use equality constraints instead of inequality constraints. They also use the ε-insensitive loss function to determine the constraints.

SVMs are black-box models. No knowledge about the circuit is needed in modelling. The obtained model is a transfer function model. Inherently, SVMs can only model static non-linear behaviour. Dynamic behaviour can only be added to the SVM using the Wiener or Hammerstein approach. There is no internal approach for modelling dynamics. The resulting model equations do not describe a conservative system. SVMs training is a constraint optimisation problem which can be solved either by using quadratic programming or convex programming. As in other black-box models, the parameters do not describe physical properties and therefore cannot be used for statistical or worst case approximation. There are fully automated approaches for creating SVM models including data generation and selection [52, 66].

2.9. Comparison

This section compares the different modelling techniques which were introduced in this chapter. Table 2.3 compares general features: the level on which the models are used, their compatibility with SPICE-like simulators, their ability to model dynamic circuits, the amount of prior knowledge which is needed to obtain the model and their degree of automation.

Different levels in analogue design use different types of models. Lower levels require high accuracy in representing the physical effects while high levels don't require detailed information about the internal mechanisms of a certain block but only need a representation of its general functionality. Of the procedures presented in this chapter, only two are used

Table 2.3.: Comparison of different modelling approaches.

Modelling Technique	Level (Sec. 1.1)	Conservative	Dynamics	Expertise	Automation
Physical Modelling	Transistor	Yes	Internal	White Box	Low
Equivalent Circuit Modelling	Transistor	Yes	Internal	White Box	Medium
Symbolic Modelling	Circuit	No	Internal	White Box	Low
Classical System Identification	Circuit	No	Both	Grey Box	High
Artificial Neural Networks	Circuit	No	Both	Black Box	High
Fuzzy Systems	Circuit	No	Both	Black Box	Medium
Support Vector Machines	Circuit	No	External	Black Box	High

to create transistor level designs: physical modelling and equivalent circuit modelling. All other approaches are used to model analogue designs on circuit level or higher (Sec. 1.1).

SPICE-like simulators use the Modified Nodal Analysis (MNA) to obtain the equation system for a circuit. The MNA is based on Kirchhoff's laws. The MNA can only be applied to models which describe the voltage and current relations on all their ports. Physical and equivalent circuit models are directly compatible with the MNA. In general, transfer function models and state-space models do not describe voltages and currents at all inputs and outputs. Those models can be simulated by SPICE-like simulators by wrapping, e.g. with dependent sources. Whether arbitrary models can be wrapped by a dependent source depends on the capabilities of the simulation software.

All techniques in this chapter can be used to model non-linear static behaviour. However, not all can model dynamic behaviour inherently. For techniques which do not offer means to model dynamic behaviour directly, the Wiener or Hammerstein approach can be used to add dynamics to the model. This is called external dynamic modelling in contrast to internal dynamic modelling. Dynamics are modelled internally in physical, equivalent circuit and symbolic modelling. For these approaches external techniques are not used.

Classical system identification provides two main types of algorithms: transfer-function and state-space models. For the former mainly external dynamics are used, for the latter internal dynamics are more common. NNs and fuzzy models can be used as static non-linear approximator for a Wiener or Hammerstein approach, but for both methods there are also internal approaches, namely tapped delay lines for NNs and Neuro-Fuzzy Systems for fuzzy modelling. SVMs can inherently only model static behaviour and can only be extended to include dynamic behaviour using external methods.

Creating a model requires a certain amount of a priori knowledge. Whether modelling can be automated and the amount of time which it takes to create said model is directly

linked to the amount of expertise which is needed. Generally, the following is true: the more a priori knowledge is needed, the longer it takes to create the model and the lower is the grade of automation. Modelling approaches can be categorised by that into three groups: white box, black box and grey box models. White box models require full knowledge about the internal mechanisms of the circuit. Behavioural models that require a high level of expertise are usually created manually and creating a model requires a lot of time. Black box models are the exact opposite: no knowledge about the internal mechanisms of a circuit is required, the model is created using only data which are measured at the input and the output of the circuit. The creation of black box models can be automated for general settings. Everything in between is called a grey box model. Typical examples of white box models are compact models, equivalent circuit models and symbolic models. SVMs and NNs are black box models. Fuzzy systems are also black box models, but creating the fuzzy rules requires expert knowledge. In classical system identification, transfer function models are commonly black box models while state-space models can be considered grey box models as the designer can include a priori knowledge.

SPICE-like simulators can handle all models which were introduced in this chapter. For the AGIAS simulator, the mathematical formulation of the models is important. The root-finding algorithm can handle non-linear functions, but fails on piecewise-defined functions. Table 2.4 compares the complexity and different characteristics of the mathematical formulation of different models.

Table 2.4.: Comparison of the mathematical complexity of different models.

Model	Function		
	Piecewise	Domain-restricted	Overall
Compact Model	Yes	Yes	High
Equivalent Circuit Model	Yes	Yes	High
Symbolic Model	Yes	Yes	High
Transfer Function Model	No	Yes	Medium
State Space Model	No	No	Medium
NN Model	Yes	No	Low
Fuzzy Model	Yes	No	Medium
SVM Model	No	No	Medium

Compact models use different equation systems for different ranges of operations. Although in newer models smoothing functions are used for the transition between the different equation systems, older variants are defined piecewise. Therefore, all approaches which use compact models – such as equivalent circuit models and symbolic models – also can contain piecewise-defined functions. Fuzzy models and NN models can also contain piecewise-defined functions, depending on the choice of the designer. The membership functions in fuzzy models and the decision functions neurons of NN models can be piecewise-defined functions. Transfer function models, state space models and SVM do not contain piecewise functions.

Using any interval arithmetic for numerical calculations can lead to problems, if equation systems contain domain restricted functions, e.g. the root function. Models which contain these functions can lead to failure in solving the equation system. Compact models, equivalent circuit models, symbolic models and transfer function models contain domain restricted functions.

Chapter 1 introduced the MC analysis and WCA. These methods vary component parameters according to some predefined distribution function or interval. Therefore, these analyses are only used on models whose parameters have physical meaning. For simulating circuits with parameter variations with the AGIAS simulator, there is a trade off between parameters with physical meaning and the number of parameters. The more parameters are represented by affine forms, the larger the overestimation in each calculation step is and the higher the probability that the root-finding algorithm will not converge. Table 2.5 compares the parameters of different models.

Table 2.5.: Parameters in different models.

Model	Representation	Number
Physical Model	Physical	High
Equivalent Circuit Model	Physical	High
Macro Model	Both	Medium
Symbolic Model	Physical	High
Classical System Identification	Mathematical	Medium
Artificial Neural Networks	Mathematical	Medium
Fuzzy Systems	Mathematical	Medium
Support Vector Machines	Mathematical	Low

Parameters in physical models, equivalent circuit models and symbolic models have physical meaning. These models can directly be used for statistical and worst case analysis. In macro models only parameters which result from top-down modelling have physical meaning. In bottom-up models the parameters may have physical meaning. Macro models can only be used for statistical and worst case after careful consideration. Parameters in the other models are mathematical parameters.

Physical models, equivalent circuit models and symbolic models come with a high number of parameters. Macro models, transfer function models and state-space models from classical system identification, NN models and fuzzy models have a medium number of parameters which is dependent on the designer's choice. SVM models have a very low number of parameters.

Sec. 1.3 presented the conditions for creating models for affine simulations. The first and second point on the list do not exclude any algorithm which has been presented in this chapter. The third item on the list excludes every approach that returns models with piecewise defined functions and a high number of non-linear functions. Physical models tend to fall in either category: older models like the MOS1 model are constructed using

piecewise defined functions. Newer models are built using non-linear functions. Therefore, physical models, equivalent circuit models, symbolic models and macro models cannot be used. This leaves system identification, NNs, fuzzy models and SVMs. Of the four data based modelling procedures, fuzzy modelling is the most complicated as creating the rules for a fuzzy system and neuro-fuzzy systems cannot be automated. Finally, models with fewer parameters have a higher chance of simulating successfully. From the remaining three approaches SVMs comes with the smallest number of parameters. SVMs are chosen for creating behavioural models of analogue circuits with parameter variations even though parameters do not carry physical meaning.

3. Support Vector Machines

In 1998 Vladimir Vapnik published the comprehensive book on Support Vector (SV) learning [76]. Support Vector Machines (SVMs) are based on the following idea: "[The SVM] maps the input vector $\mathbf{x}$ into a high-dimensional feature space [$\mathcal{H}$] through some non-linear mapping, chosen a priori. In this space, an optimal separating hyperplane is constructed." [76, p. 421]

Vapnik's general description introduces two concepts which are essential for SVMs: the mapping into a possibly high-dimensional feature space and the optimal separating hyperplane. A hyperplane is written as

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{3.1}$$

An optimal separating hyperplane is constructed to separate two classes of data with maximum margin and minimum gradient. This is the basic formulation of an SVM. With the dot product in Eq. (3.1) the algorithm is limited to solving linear problems. This is overcome by mapping the data into a feature space. Solving a non-linear pattern recognition problem is difficult. SVMs work around this by using a non-linear mapping function Θ to map data into a feature space $\mathcal{H}$ in which the linear separating hyperplane is constructed. The mapping function Θ gives way to another important concept: the kernel function k.

Data for training SVMs are represented as

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l) \in \mathcal{X} \times \mathcal{Y}. \tag{3.2}$$

In statistical learning the $\mathbf{x}_i$ are called patterns and the y_i are called labels or targets [64]. More common in engineering are *inputs* and *outputs* respectively which will be used throughout this thesis. Although the input space can be an arbitrary construction, often the input space is constructed from vectors of real numbers

$$\mathcal{X} = \mathbb{R}^N. \tag{3.3}$$

The output space takes two different forms depending on the application of the SVM. There are two main applications: pattern recognition and regression. For pattern recognition the output space is a countable set of labels – with binary pattern recognition as the simplest case and

$$\mathcal{Y}_{bin} = \{-1, +1\}. \tag{3.4}$$

For regression the output space usually equals the space of real numbers

$$\mathcal{Y} = \mathbb{R}. \tag{3.5}$$

During training of an SVM a measure is needed to determine the quality of the estimate. This is done using the risk and loss functions. Loss functions are functions which assign a real value to measure misclassification or misprediction. This means they measure the training error e_{Train}. The risk function collects the single errors.

3.1. Statistical Learning Concepts

Fig. 3.1 shows the learning model of learning from examples which was introduced by Vapnik [76, p. 20]. It consists of a generator, a target operator and the learning machine. The generator generates input vectors $\mathbf{x} \in \mathcal{X}$ where $\mathcal{X}$ denotes the input space. The input vectors are i.i.d. and are generated from some unknown probability distribution function $F(\mathbf{x})$.

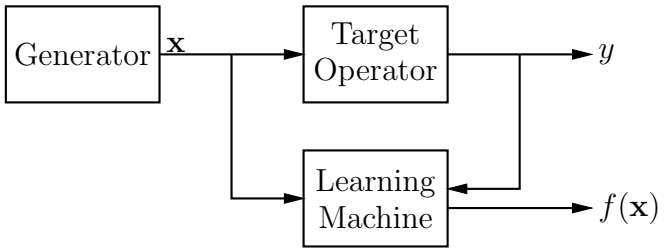


Figure 3.1.: Learning model according to Vapnik [76, p. 20].

The input vectors $\mathbf{x}$ are fed into the target operator which returns the output values y. The function which transforms the input vectors is unknown, but it is guaranteed to exist and does not change. Mathematically, the target operator represents an unknown conditional distribution function $F(y|\mathbf{x})$ which also includes using a function $y = g(\mathbf{x})$.

The pairs that form the training set are drawn according to the joint distribution function $F(\mathbf{x}, y) = F(\mathbf{x})F(y|\mathbf{x})$. The learning machine observes the training set and constructs an operator that approximates the target operator. This process is called the learning process. During learning an appropriate function is chosen from a given set of functions. The learning problem is based on the general statistical problem of minimising the risk functional on the basis of empirical data.

Choosing a function from a set of functions is an optimisation problem. In mathematics, optimisation means finding the minimum or maximum of an objective function. The objective function is a measure of quality for the optimisation problem. Optimisation problems can be unconstrained or constrained; and there are also two types of constraints: equality and inequality constraints [57]. Training an SVM means solving a constrained optimisation problem with inequality constraints.

3.1.1. Generalisation and Overfitting

Generalisation is a central point in learning. If a model generalises well, it can estimate the outputs for inputs that have not been available during training. Consider a learning model in the real world, e.g. a student. Generalising well means that the student is able to learn a rule from the given data and apply it to new data correctly. Not generalising well means the given data are learnt by heart and new data cannot be handled correctly.

It is difficult to measure the ability of an algorithm to generalise directly. One such measure is obtained by using two disjoint sets of data: the training set $\mathcal{X}_{Train}$ and the test set $\mathcal{X}_{Test}$. The training set is the data set on which the model is trained. Both its inputs and outputs are known during training. The test set is not available during training. For the finished SVM this set is completely unknown. The ability to generalise well can only be calculated after training is finished.

This measure is based on the training error e_{Train} and the test error e_{Test}. The training error is calculated for each data pair $\mathbf{x}_i, y_i$ in $\mathcal{X}_{Train}$. For pattern recognition, the error measure is often based on counting the number of classification errors. For regression, usually the difference between the original and estimated output or the norm of that difference is used.

$$e_{Train,i} = \|y_i - f(\mathbf{x}_i)\| \qquad \mathbf{x} \in \mathcal{X}_{Train} \tag{3.6}$$

where $f(\mathbf{x}_i)$ is the estimate for the input $\mathbf{x}_i$. The test error is calculated on the test set. Both the test set and the test error are defined in a similar way as the training set and training error. The data pairs are denoted by the index k in contrast to the index i of the training data.

$$e_{Test,k} = \|y_k - f(\mathbf{x}_k)\| \qquad \mathbf{x} \in \mathcal{X}_{Test}. \tag{3.7}$$

Usually, both measures are not given for each data sample separately, but the overall training error e_{Train} and test error e_{Test} are calculated by averaging or summing up the errors for each set.

Of course, the training error is a measure for the success of the training. Therefore, a small training error is desired. A large training error indicates that the model is inaccurate. The training error alone can not measure the performance of a model in a setting with previously unknown inputs. That means that a model with a very low training error can show very poor performance in a more general setting. This information gap is closed by combining the training and the test error. Fig. 3.2 illustrates the interpretation of these two measures. There are two cases that denote a poor model: first case, the training error is high, which was already discussed. This is called underfitting: the underlying model is too simple. The second case is when the training error is low, but the test error is high. This case is called overfitting and denotes a poor generalisation ability of the model. In this case the training data are estimated with very high accuracy, but the underlying model is overly complex and cannot estimate other data correctly [64].

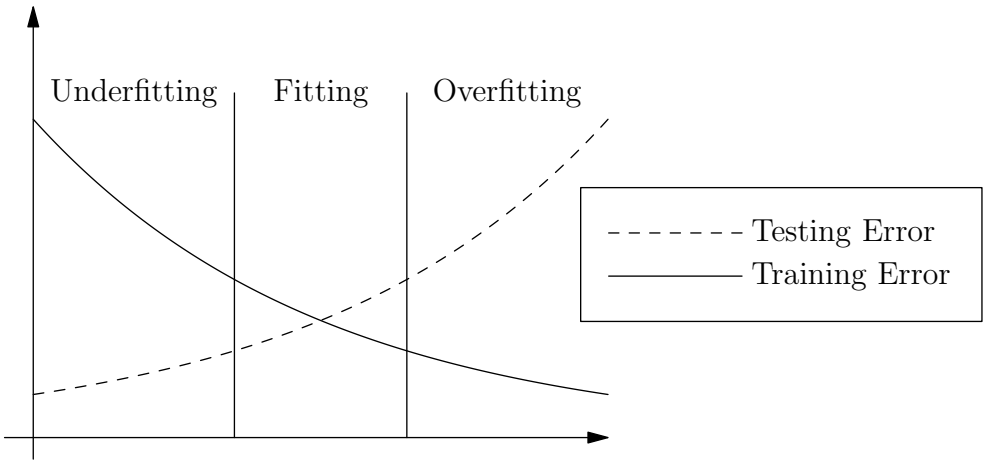


Figure 3.2.: Illustration of overfitting.

3.1.2. Risk

Optimisation problems require a function that serves as quality criterion. Section 3.1.1 introduced two possibilities: the training error and the test error. Minimising the expected test error is called a transduction problem which is very difficult to handle "both computationally and conceptually" [64, p. 66]. Instead, the training error is chosen to define the quality criterion. Sec. 2.8 introduced the empirical risk R_{emp} as a sum of estimation errors. Therefore, Eq. (2.17) can be rewritten in terms of the training error

$$R_{emp} = \sum_{i=1}^{l} e_{Train}. \tag{3.8}$$

More formally, SV learning problems are function selection problems. From a set of functions $\{f(\mathbf{x})\}$ one function $f^*(\mathbf{x})$ should be chosen so that it minimises

$$R = R(f(\mathbf{x})) = \int \mathcal{L}(\mathbf{x}, y, f(\mathbf{x}))dF(\mathbf{x}) \tag{3.9}$$

where R is the risk functional, $\mathcal{L}$ is a loss function and $F(\mathbf{x})$ an unknown probability distribution function according to which the inputs for training have been drawn. As $F(\mathbf{x})$ is unknown, Eq. (3.9) cannot be minimised directly. But as the inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l$ are known, the task is to minimise the risk functional based on empirical data. This is one of the main problems of mathematical statistics [76].

One common way to approximate integrals is to use a finite sum. In SV learning this leads to the empirical risk minimisation induction principle where the empirical risk R_{emp} is used instead of Eq. (3.9) [64].

Definition 3.1.1. *Given a loss function $\mathcal{L}(\mathbf{x}, y, f(\mathbf{x}))$ on a set of functions $\{f(\mathbf{x})\}$ in $\mathcal{X} \in \mathbb{R}^n$ and a sample $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2) \ldots, (\mathbf{x}_l, y_l) \in \mathcal{X}_{Train}$ drawn according to an unknown*

distribution function $F(\mathbf{x})$ on $\mathcal{X}$. Then the risk $R(f(\mathbf{x}))$ is approximated by the empirical risk

$$R_{emp} = \frac{1}{l}\sum_{i=1}^{l}\mathcal{L}(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) \tag{3.10}$$

3.1.3. Loss Functions

Loss functions are the basis to calculate both the risk R and the empirical risk R_{emp}. Section 3.1.1 introduced the difference between the estimated output $f(\mathbf{x})$ and the original output y or some norm of that difference as a measure for the training error for a regression problem. A more formal definition for loss functions is given by:

Definition 3.1.2. *(Loss function [64, p. 62]) Denote by $\{\mathbf{x}, y, f(\mathbf{x})\} \in \mathcal{X} \times \mathcal{Y} \times \mathcal{Y}$ the triplet consisting of [an input] $\mathbf{x}$, an [output] y and a prediction $f(\mathbf{x})$. Then the map $\mathcal{L} : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto [0, \infty)$ with the property $\mathcal{L}(\mathbf{x}, y, y) = 0$ for all $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ will be called a loss function.*

In other words, loss functions are strictly non-negative functions that return 0 if the estimate equals the original output value. Therefore, the minimum of any loss function is 0 and can be obtained for any given $\mathbf{x}$ and y.

For regression, loss functions use the amount of misprediction $y - f(\mathbf{x})$ and do not incorporate the input $\mathbf{x}$ separately to determine the quality of the prediction. Therefore, loss functions for regression are written as $\mathcal{L}(y, f(\mathbf{x}))$. Three loss functions were considered by Vapnik [76]:

$$\mathcal{L}_\varepsilon(y, f(\mathbf{x})) = |y - f(\mathbf{x})|_\varepsilon \tag{3.11}$$

$$\mathcal{L}(y, f(\mathbf{x})) = |y - f(\mathbf{x})|_\varepsilon^2 \tag{3.12}$$

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} c|y - f(\mathbf{x})| - \frac{c^2}{2} & \text{for } |y - f(\mathbf{x})| > c \\ \frac{1}{2}|y - f(\mathbf{x})|^2 & \text{for } |y - f(\mathbf{x})| \leq c \end{cases} \tag{3.13}$$

the linear ε-insensitive loss function (Eq. (3.11)), the quadratic ε-insensitive loss function (Eq. (3.12)) and the Huber loss function (Eq. (3.13)), with

$$|y - f(\mathbf{x})|_\varepsilon = \begin{cases} |y - f(\mathbf{x})| - \varepsilon & \text{if } |y - f(\mathbf{x})| > \varepsilon \\ 0 & \text{otherwise} \end{cases}. \tag{3.14}$$

Here ε represents the maximum permissible error: all mispredictions smaller than ε are admissible and the corresponding loss is set to zero. For $\varepsilon = 0$ the linear ε-insensitive loss function becomes an l_1 loss function and the quadratic ε-insensitive loss function equals an l_2 or squared loss function [64]. Similar to the maximum permissible error ε, the parameter c is chosen a priori and determines which mispredictions are penalised according to a squared function or a linear function.

Per definition, loss functions can only return non-negative values. Therefore, the optimisation problem cannot profit from negative errors. At the same time, loss functions cannot represent the direction of a misprediction, as there is no information about whether the estimate is larger or smaller than the original output. This information is incorporated into so-called slack variables ξ and ξ^* for estimates which are larger and estimates which are smaller than the output respectively. All mispredictions regardless of the direction are addressed by $\xi^{(*)}$.

From the loss functions, the distribution of y for a given $\mathbf{x}$ according to some underlying functionality $g(\mathbf{x})$ can be calculated by minimising the log-likelihood

$$\mathfrak{L}[g] = \sum_{i=1}^{l} -\ln p(y_i|\mathbf{x}_i, g). \tag{3.15}$$

The function $p(y_i|\mathbf{x}_i, g)$ gives the probability with which the loss for a certain pair $(y_i, \mathbf{x}_i)$ will occur. Table 3.1 presents the most common loss functions and their corresponding density models. For simplicity, the term $y - f(\mathbf{x})$ is replaced by ξ.

Table 3.1.: Common loss functions and corresponding density models [...] [64, p. 70]

	[L]oss [F]unction $\mathcal{L}(\xi)$	[D]ensity [M]odel $p(\xi)$
ε-insensitive	$\lvert\xi\rvert_\varepsilon$	$\frac{1}{2(1+\varepsilon)} \exp(-\lvert\xi\rvert_\varepsilon)$
Laplacian	$\lvert\xi\rvert$	$\frac{1}{2} \exp(-\lvert\xi\rvert)$
Gaussian	$\frac{1}{2}\xi^2$	$\frac{1}{\sqrt{2\pi}} \exp(-\frac{\xi^2}{2})$
Huber's robust loss[1]	$\begin{cases} \frac{1}{2\sigma}\xi^2 & \text{if } \lvert\xi\rvert \le \sigma \\ \lvert\xi\rvert - \frac{\sigma}{2} & \text{otherwise} \end{cases}$	$\propto \begin{cases} \exp(-\frac{\xi^2}{2\sigma}) & \text{if } \lvert\xi\rvert \le \sigma \\ \exp(\frac{\sigma}{2} - \lvert\xi\rvert) & \text{otherwise} \end{cases}$
Polynomial	$\frac{1}{d}\lvert\xi\rvert^d$	$\frac{d}{2\Gamma(1/d)} \exp(-\lvert\xi\rvert^d)$
Piecewise polynomial	$\begin{cases} \frac{1}{d\sigma^{d-1}}\lvert\xi\rvert^d & \text{if } \lvert\xi\rvert \le \sigma \\ \lvert\xi\rvert - \sigma\frac{d-1}{d} & \text{otherwise} \end{cases}$	$\propto \begin{cases} \exp(-\frac{\lvert\xi\rvert^d}{d\sigma^{d-1}}) & \text{if } \lvert\xi\rvert \le \sigma \\ \exp(\sigma\frac{d-1}{d} - \lvert\xi\rvert) & \text{otherwise} \end{cases}$

3.1.4. Regularisation

Minimising only the empirical risk R_{emp} can lead to numerical instabilities and can harm generalisation. The most common way to avoid these problems is to restrict the set of admissible functions. This technique is called regularisation and was introduced by Tikohonov and Arsenin [73]. For kernel methods – such as SVMs – two special methods are used: a

[1] This formulation and the formulation used in Eq. (3.13) describe the same principle and lead to an equal measure for the mispredictions for $c = \sigma = 1$.

coefficient space constraint on the expansion coefficients of the weight vector in feature space or a function space regularisation which penalises the weight vector directly [64].

The main purpose of regularisation is to stabilise the optimisation problem. This is done by adding a stabilisation or regularisation term to the original objective function. The original objective function for SVMs is the empirical risk R_{emp}. Together with a regularisation term, the most commonly used objective function Φ is obtained:

$$\Phi = \Omega[f] + CR_{emp} \tag{3.16}$$

where $\Omega(f)$ is the regularisation term and C is a non-negative constant. The regularisation term is a measure for the simplicity of a function. The constant C is the regularisation parameter which determines a trade-off between the empirical risk R_{emp} and the regularisation term $\Omega(f)$.

Considering the hyperplane given in Eq. (3.1), the standard choice for the regularisation term for SVMs is the quadratic regulariser

$$\Omega[f] = \frac{1}{2}\|\mathbf{w}\|^2. \tag{3.17}$$

The geometric interpretation of minimising this regularisation function is finding the function with the smallest gradient.

3.1.5. Optimal Separating Hyperplane and Soft-Margin Hyperplane

Separating hyperplanes are a basic concept in learning and have been used to create classical learning algorithms [76]. They are used to separate two classes of data in binary pattern recognition. If two classes are separable, the optimal separating hyperplane can be determined. As most problems are not separable, the soft-margin hyperplane is used which allows for misclassifications in the final hyperplane.

Hyperplanes have the general form

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{3.18}$$

The parameters $\mathbf{w} \in \mathcal{X}$ and $b \in \mathbb{R}$ define the hyperplane and are determined during training. There are numerous ways to describe one hyperplane by scaling one parameter set $(\mathbf{w}, b)$. To get rid of that degree of freedom, the canonical form of the hyperplane is used. The canonical form is obtained by scaling $(\mathbf{w}, b)$ such that

$$\min|\mathbf{w} \cdot \mathbf{x}_i - b| = 1. \tag{3.19}$$

The point closest to the hyperplane therefore has a distance of $\frac{1}{\|\mathbf{w}\|}$.

Creating an optimal margin hyperplane requires solving an optimisation problem:

$$\text{minimise } \Phi = \frac{1}{2}\|\mathbf{w}\|^2 \tag{3.20}$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 \tag{3.21}$$

The constraints formulate the fact that the product of the original output and the estimated output must be positive – equal to 1 in the ideal case.

The optimal margin hyperplane introduces one very severe limitation: data must be separable for the algorithm to converge [76]. An optimal separating hyperplane can rarely solve real world problems. To overcome this limitation, it is necessary to allow some error to remain in the resulting SVM. This possibility is created by the soft margin hyperplane.

Soft margin hyperplanes are designed to allow a certain amount of errors. Misclassifications are incorporated into so-called slack variables $\xi \geq 0$. These slack variables are used to relax the constraints which leads to the following optimisation problem

$$\text{minimise } \Phi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \tag{3.22}$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 + \xi_i \tag{3.23}$$

In this case the objective function is expanded by the sum over all slack variables. Creating the soft-margin hyperplane makes use of all concepts introduced in earlier sections: the empirical risk R_{emp}, the loss function $\mathcal{L}_\varepsilon$ and regularisation.

3.1.6. Kernel Functions and Feature Space

The previous section introduced separating hyperplanes. Hyperplanes are linear constructs and can therefore not solve non-linear problems. However, most real-world problems require the learning machine to construct some non-linear rule. Creating a non-linear rule is a rather complicated task in input space. Here, two more key concepts are employed: feature space $\mathcal{H}$ and kernel functions k. The key idea is that data can be mapped from input to feature space using some arbitrary function. As long as the dot product is defined in the feature space, a separating hyperplane can be constructed.

Commonly the feature space is of higher dimension than the input space which makes it subject to the curse of dimensionality. In approximation theory, the curse of dimensionality describes that with increasing dimension the rate of asymptotic convergence decreases drastically. This is avoided by using kernel functions k. Kernel functions allow to calculate the dot product in feature space without explicitly calculating the mapping into feature space [64].

Kernel functions have the general form

$$k(\mathbf{x}, \mathbf{x}') = \Theta(\mathbf{x}) \cdot \Theta(\mathbf{x}') \tag{3.24}$$

where $k(\mathbf{x}, \mathbf{x}')$ denotes the kernel function and $\Theta(\mathbf{x})$ is the mapping function $\Theta : \mathcal{X} \to \mathcal{H}$ from input into feature space.

For constructing kernel functions and feature spaces, there are two possibilities: constructing a kernel function for a given feature space or start with a kernel function and construct the associated feature space.

Constructing a kernel function from a given feature space requires formulating the dot product from the mapping function and obtain a formulation in input space – the kernel function. Schölkopf and Smola demonstrate this approach on product features and obtain the polynomial kernel [64, p. 27 et seq.]

$$k(\mathbf{x}, \mathbf{x}') = \Theta_d(\mathbf{x}) \cdot \Theta_d(\mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^d \tag{3.25}$$

where d is the order of the polynomial and $\Theta_d(x)$ is the mapping function which maps data x from the input space to a vector in feature space. The elements of this vector are all possible dth degree products of the elements of $\mathbf{x}$. The dimension of the feature space of a polynomial kernel can be calculated as

$$N_{\mathcal{H}} = \binom{d+N-1}{d} = \frac{(d+N-1)!}{d!(N-1)!} \tag{3.26}$$

where N is the dimension of $\mathbf{x}$ and d is the degree of the polynomial. For real world scenarios this leads to a very high dimensionality. Calculating the mapping explicitly would be very costly. Using the kernel function given by Eq. (3.25) makes use of the feature space without explicitly calculating the mapping.

The second approach is to construct a feature space from a given positive definite kernel function. This property is only defined for matrices, not for functions. To define positive definiteness for a kernel function, the kernel matrix is defined as

$$K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) \qquad \mathbf{x}_i, \mathbf{x}_j \in \mathcal{X}. \tag{3.27}$$

A positive definite kernel function is defined as follows [64]:

Definition 3.1.3. *Given a non-empty set $\mathcal{X}$ and a kernel function k on $\mathcal{X} \times \mathcal{X}$. If $(K_{i,j})$ is positive definite, the kernel function is positive definite.*

This implies positivity on the diagonal, this is

$$k(\mathbf{x}, \mathbf{x}) \geq 0 \quad \forall \mathbf{x} \in \mathcal{X}, \tag{3.28}$$

and symmetry

$$k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_j, \mathbf{x}_i). \tag{3.29}$$

From positive definitive kernels, first a map into a feature space is constructed. Next, this map has to be turned into a vector space. Finally, the dot product is defined on that vector space so that the kernel function satisfies Eq. (3.24). Feature spaces which are constructed like this are called Reproducing Kernel Hilbert Spaces (RKHSs) [64].

Beside the polynomial kernel (Eq. (3.25)) a variety of other kernel functions can be used. From the polynomial kernel, two other kernels can be obtained: the linear kernel for $d = 1$ (Eq. (3.30)) and the inhomogeneous polynomial kernel (Eq. (3.31)):

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}' \tag{3.30}$$

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d \tag{3.31}$$

where $c > 0$ is a positive real constant and d is the degree of the polynomial. Vapnik suggests using Gaussian Radial Basis Functions (RBFs) [76]

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}) \tag{3.32}$$

with $\sigma > 0$, and sigmoid kernels

$$k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa(\mathbf{x} \cdot \mathbf{x}') + \vartheta) \tag{3.33}$$

with $\kappa > 0$ and $\theta > 0$. Sigmoid kernels are not positive definite, but proved useful in practice nevertheless. Schölkopf and Smola suggest using B_n spline kernels of odd order

$$k(\mathbf{x}, \mathbf{x}') = B_{2p+1}(\|\mathbf{x} - \mathbf{x}'\|) \quad \text{with } B_n = \bigotimes_{n=1}^{n} I_{[-\frac{1}{2}, \frac{1}{2}]} \tag{3.34}$$

with $p \in \mathbb{N}$. The kernel calculates the $2p + 1$ fold convolution of the centred unit interval $[-\frac{1}{2}, \frac{1}{2}]$. A second suggestion are arbitrary RBF kernels

$$k(\mathbf{x}, \mathbf{x}') = f(d(\mathbf{x}, \mathbf{x}')) \tag{3.35}$$

where d is a metric on $\mathcal{X}$ and f a function on $\mathbb{R}_0^+$ [64].

The third important concept is the so-called kernel trick. This concept states that if an algorithm is formulated in terms of a positive definite kernel, an alternative algorithm can be obtained by replacing the kernel k by another positive definite kernel $\tilde{k}$ [64, p. 34].

3.1.7. Optimisation

In mathematics, optimisation means finding an optimum for a given problem by minimising or maximising a quality measure. Generally, constrained optimisation problems are written as

$$\begin{aligned} &\underset{x}{\text{minimise}}\ \Phi(x) \\ &\text{subject to } g_i(x) \leq 0 \quad \forall i \in [n] \\ &\qquad\qquad h_j(x) = 0 \quad \forall j \in [m] \end{aligned} \tag{3.36}$$

where $\Phi(x)$ is the target function, $g_i(x)$ are inequality constraints, $h_j(x)$ are equality constraints and $n, m \in \mathbb{N}_0$ with either n or m larger than zero. As every minimisation problem can be transformed into a maximisation problem by multiplying the target function by -1, the remainder of this subsection only deals with minimisation.

One common method to transform optimisation problems is the Lagrange multiplier rule. It was suggested in 1788 by Lagrange for minimising a function with equality constraints [76]. Consider the optimisation problem in Eq. (3.36) with $n = 0$. Both the target function $\Phi(x)$ and the constraints $h_j(x)$ and their derivatives are continuous. Assume that x^* is a local minimum in the optimisation problem and consider a function

$$L(x, \lambda_0, \lambda) = \lambda_0 \Phi(x) + \sum_{i=1}^{m} \lambda_i h_i(x). \tag{3.37}$$

L is called Lagrange function or Lagrangian and the coefficients $\lambda_k \quad k = 0, 1, \ldots, m$ are called Lagrange multipliers.

Theorem 3.1.1 (Langrange [76, p. 392]). *Let the functions $h_i(x)$, $i = 1, \ldots = m$, be continuous and differentiable in the vicinity of point x^*. If x^* is the point of a local extremum, then one can find Lagrange multipliers $\lambda^* = (\lambda_1, \ldots, \lambda_m)$ and λ_0 which are not equal to zero simultaneously such that the following conditions (the so-called stationary conditions)*

$$L'_x(x^*, \lambda_0^*, \lambda^*) = 0 \tag{3.38}$$

hold true. That is

$$L'_{x_i}(x^*, \lambda_0^*, \lambda^*) = 0 \quad i = 1, \ldots, n. \tag{3.39}$$

To guarantee that $\lambda_0 \neq 0$ it is sufficient that the vectors

$$h'_1(x^*), \ldots, h'_m(x^*) \tag{3.40}$$

are linearly independent.

The last important concept used to create SVM algorithms is duality. It is directly linked to the Lagrange multiplier method. The concept of duality means for every primal optimisation problem in x one can formulate a dual maximisation problem in terms of λ by computing the saddle point of the Lagrangian. The solution of the dual problem is guaranteed to exist following the Wolfe theorem which is based on the Karush-Kuhn-Tucker conditions (KKTs) [64].

The most important sufficient criterion for optimality is the Kuhn-Tucker saddle point condition. It is often referred to as KKT, as it is based on earlier work by Karush. While the inequality constraints $g_i(x)$ can be included directly, the equality conditions are included by splitting them into $h_j(x) \leq 0$ and $h_j(x) \geq 0$.

Theorem 3.1.2 (Kuhn-Tucker Saddle Point Conditions [64, p. 167]). *Assume an optimisation problem of the form [Eq. (3.36)], where $\Phi, g_i, h_j : \mathbb{R}^k \to \mathbb{R}$ for $i \in [n]$ and $j \in [m]$ are arbitrary functions, and a Lagrangian*

$$L(x, \alpha) := \Phi(x) + \sum_{i=1}^{n} \alpha_i g_i(x) + \sum_{j=1}^{m} \beta_j h_j(x) \text{ where } \alpha_i \geq 0 \text{ and } \beta_j \in \mathbb{R}. \tag{3.41}$$

If a set of variables $\bar{x}, \bar{\alpha}, \bar{\beta}$ with $\bar{x} \in \mathbb{R}^k$, $\bar{\alpha}_i \geq 0$ and $\bar{\beta} \in \mathbb{R}^m$ exists, such that for all $x \in \mathbb{R}^m$, $\alpha \in [0, \infty)^n$ and $\beta \in \mathbb{R}^m$,

$$L(\bar{x}, \alpha, \beta) \leq L(\bar{x}, \bar{\alpha}, \bar{\beta}) \leq L(x, \bar{\alpha}, \bar{\beta}), \tag{3.42}$$

then $\bar{x}$ is a solution to [Eq. (3.36)].

These sufficient conditions become necessary conditions with the additional assumptions that Φ and g_i are convex functions on the convex set $\mathcal{X} \subseteq \mathbb{R}^k$, if g_i fulfils one of the following conditions [64, p. 167]

1. Slater's condition: There exist an $x \in \mathcal{X}$ such that for all $i \in [n]$ $g_i(x) < 0$
2. Karlin's condition: For all non-zero $\alpha \in [0, \infty)^n$ there exists an $x \in \mathcal{X}$ such that $\sum_{i=1}^{n} \alpha_i g_i(x) \leq 0$.
3. Strict constraint qualification: The feasible region $\mathcal{X}$ contains at least two distinct elements, and there exists an $x \in \mathcal{X}$ such that all g_i are strictly convex at x with respect to $\mathcal{X}$.

The KKT conditions as described up to here are not particular useful for calculation. But as they formulate conditions for a saddle point, they can be rewritten in terms of derivatives of the Lagrangian: [64, p. 170]

1. Saddle point in $\bar{x}$: $\partial_x L(\bar{x}, \bar{\alpha}) = \partial_x \Phi(\bar{x}) + \sum_{i=1}^{n} \bar{\alpha}_i \partial_x g_i(\bar{x}) = 0$
2. Saddle point in $\bar{\alpha}$: $\partial_\alpha L(\bar{x}, \alpha) = g_i(\bar{x}) \leq 0$
3. Vanishing KKT gap: $\sum_{i=1}^{n} \bar{\alpha}_i g_i(\bar{x}) = 0$

3.1.8. Specialised Optimisation Algorithm: Sequential Minimal Optimisation

SVM parameters can be determined using standard quadratic programming. For larger problems, this is problematic as computing the kernel matrix can take very long or can be impossible due to memory restrictions. The basic idea of the Sequential Minimal Optimisation algorithm (SMO) is to reduce the optimisation problem to the smallest quadratic optimisation problem possible: an optimisation problem in only two variables. In case of SVMs, this means an optimisation problem in two Lagrange multipliers. The SMO was published in 1999 by Platt [61].

The SMO has three components: choosing a multiplier pair for optimisation, calculating the solution analytically and an algorithm to calculate the offset b. The SMO requires the SVM to have the following form:

$$\underset{\mathbf{a}}{\text{maximize}}\ W(\mathbf{a}) = -\frac{1}{2} \sum_{i,j=1}^{N} \mathbf{q}_i \mathbf{q}_j k(\mathbf{x}_i, \mathbf{x}_k) \mathbf{a}_i \cdot \mathbf{a}_j + \sum_{i=1}^{N} \mathbf{g}_i \mathbf{a}_i \tag{3.43}$$

$$\text{subject to } \sum_{i=1}^{N} \mathbf{q}_i \mathbf{a}_i = 0 \tag{3.44}$$

$$0 \leq \mathbf{a}_i \leq C \tag{3.45}$$

The vector $\mathbf{a}$ is the vector of Lagrange multipliers and the vector $\mathbf{q}$ is a sign vector. The elements of the single vectors are referenced by an index. The elements of the vector $\mathbf{q}$ can take the values $[1, -1]$. The vectorised problem is obtained by first formulating the Lagrange multiplier vector $\mathbf{a}$ and sign vector $\mathbf{q}$. These two vectors can be obtained from

the constraints of the optimisation problem. Then the target function is rewritten using the two vectors. In this process the vector $\mathbf{g}$ is determined. As a final step, all constraints are rewritten in terms of the vectors and their elements.

The algorithm uses two heuristics to chose the multipliers, one for the first and one for the second multiplier. For the first multiplier, all multipliers are collected in two groups: bounded and non-bounded multipliers. All multipliers that are not equal to C or 0 are called non-bounded multipliers. They are more likely than bounded multipliers to violate the KKT conditions. The second multiplier is chosen to maximize the step taken in the solution. As there might be solutions which don't make positive progress, this heuristic iterates over candidates for optimisation so that a pair with positive progress is guaranteed.

After two multipliers α_1 and α_2 have been chosen for optimisation, both multipliers are limited to the square between 0 and C by Eq. (3.45). Eq. (3.44) further limits the solution space to a straight line. The algorithm then calculates the minimum of Eq. (3.43) on the line given by Eq. (3.44). All equations used in this process can be derived analytically.

After the multipliers have been optimised, the offset b is calculated, so that the KKT conditions are fulfilled for the two multipliers. The algorithm iterates over all multipliers until is can't find any pairs for optimisation.

3.2. Training Algorithms for Support Vector Machines

The algorithms which are presented in the following subsections follow the same general method of construction:

1. First the primal optimisation problem is constructed from the risk and loss function. The empirical risk functional and a regularisation term form the target function while the loss function is used to construct the constraints.

2. The primal optimisation problem is then transformed into a single function using the Lagrange multiplier method.

3. The Lagrange function derivatives with respect to the primal variables are calculated. They are used to obtain a calculation rule for the gradient $\mathbf{w}$ and the dual formulation. The dual formulation is obtained by resubstituting the derivatives into the Lagrange function to remove the primal variables. The result is a quadratic optimisation problem which is formulated in terms of the Lagrange multipliers only.

The dual optimisation problem is solved using quadratic or convex programming or are solved by specialised algorithms such as the SMO. The dual variables are then used to calculate the primal parameters. Some parameters of an SVM have to be set before training the model. These parameters are called hyperparameters.

Several algorithms are based on this general approach. This section presents Vapnik's εSVM for Classification (εSVC) and εSVM for Regression (εSVR) and Schölkopf's νSVM for Regression (νSVR). Suykens's Least Square SVM (LS-SVM) is not described here, as it was not used as basis for the algorithms which are presented in Chapter 4.

3.2.1. Vapnik's Support Vector Machines for Classification

εSVM for Classifications (εSVCs) can be constructed from both the optimal margin hyperplane and the soft margin hyperplane. As the optimal margin hyperplane is of no practical significance, this section introduces the algorithm based on the soft margin hyperplane [76, p. 408 et seqq.].

The optimisation problem for soft margin hyperplane classifiers has already been constructed in Eq. (3.22):

$$\underset{\mathbf{w},\xi_i}{\text{minimise}}\ \Phi(\mathbf{w},\xi_i) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \tag{3.46}$$

$$\text{subject to } y_i(\mathbf{w}\ \cdot\ \mathbf{x}_i + b) \geq 1 - \xi_i \tag{3.47}$$

From this the Lagrangian is obtained as

$$L(\mathbf{w},b,\xi_i) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i - \sum_{i=1}\alpha_i(y_i(\mathbf{w}\ \cdot\ \mathbf{x}_i - b) - 1 + \xi_i) - \sum_{i=1}^{l}\beta_i \tag{3.48}$$

where α_i and β_i are the Lagrange multipliers. As SVMs optimisation problems are formulated with the inequality constraints larger than zero rather than smaller than zero, there is a sign change in the Lagrangian (minus instead of plus).

Deriving the Lagrangian for the primal variables results in

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l}\alpha_i y_i \mathbf{x}_i = 0 \tag{3.49}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l}\alpha_i y_i = 0 \tag{3.50}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0 \tag{3.51}$$

These derivatives are substituted into the Lagrangian to obtain the dual optimisation problem:

$$\begin{aligned} \underset{\alpha_i}{\text{maximise}}\ W(\alpha_i) &= -\frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\ \cdot\ \mathbf{x}_j + \sum_{i=1}^{l}\alpha_i \\ \text{subject to } \sum_{i=1}^{l}\alpha_i y_i &= 0 \\ 0 \leq \alpha_i &\leq C \end{aligned} \tag{3.52}$$

The first derivative is the calculation rule for the parameter $\mathbf{w}$

$$\mathbf{w} = \sum_{i=1}^{l}\alpha_i y_i \mathbf{x}_i. \tag{3.53}$$

This rule can be used to illustrate what exactly support vectors are: the Lagrange multipliers α_i are associated with a certain data pair $(\mathbf{x}_i, y_i)$. The multipliers are determined by solving the dual optimisation problem. Usually, only a portion of these is non-zero. The input vectors $\mathbf{x}_i$ associated with the non-zero multipliers α_i are called support vectors.

The parameter b is determined using the KKT conditions. According to the KKT, only the non-zero Lagrange multipliers α_i correspond to constraints which have been exactly met. Therefore for all $\alpha_i > 0$

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 = 0 \tag{3.54}$$

holds. For each constraint, the offset b_i can be calculated as

$$b_i = y_i - \sum_{j=1}^{l} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{3.55}$$

One method to calculate b is to average all b_i [64].

Substituting Eq. (3.53) into the hyperplane form results in the expansion of the hyperplane

$$f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b. \tag{3.56}$$

This algorithm is extended to non-linear problems by using the kernel trick. The algorithm has been developed using the linear kernel. Therefore, the general algorithm can be written as

$$f(\mathbf{x}) = \sum_{i=1}^{l} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b. \tag{3.57}$$

3.2.2. Vapnik's Support Vector Machines for Regression

In his 1998 book, Vapnik uses three different loss functions to construct SVMs for function estimation: the ε-insensitive loss function, the squared ε-insensitive loss function and Huber's loss function [76]. The most common variant is the SVM constructed using the ε-insensitive loss function, which was called εSVM by Schölkopf et alii [65].

A separating hyperplane

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{3.58}$$

is estimated on i.i.d. data

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l) \in \mathcal{X} \times \mathbb{R}. \tag{3.59}$$

As with the εSVC, the basis for this algorithm is formed by the soft margin separating hyperplane.

Using the regularisation term $\frac{1}{2}\|\mathbf{w}\|^2$ and the ε-insensitive loss function, the most common εSVR training algorithm is

$$\underset{\mathbf{w},\xi_i^{(*)}}{\text{minimise}}\ \Phi(\mathbf{w},\xi_i^{(*)}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{3.60}$$

$$\text{subject to } (\mathbf{w}\ \cdot\ \mathbf{x}_i + b) - y_i \leq \varepsilon + \xi_i \tag{3.61}$$

$$y_i - (\mathbf{w}\ \cdot\ \mathbf{x}_i + b) \leq \varepsilon + \xi_i^* \tag{3.62}$$

$$\xi_i, \xi_i^* \geq 0 \tag{3.63}$$

with two hyperparameters: the cost C and the maximum permissible error ε.

From this the Lagrange function is calculated as

$$\begin{aligned} L(\mathbf{w}, b, \xi_i^{(*)}) =& \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ &- \sum_{i=1}^{l}\alpha_i(\varepsilon + \xi_i + y_i - \mathbf{w}\ \cdot\ \mathbf{x}_i - b) \\ &- \sum_{i=1}^{l}\alpha_i^*(\varepsilon + \xi_i^* - y_i + \mathbf{w}\ \cdot\ \mathbf{x}_i + b), \end{aligned} \tag{3.64}$$

where $\mathbf{w}$, b and $\xi_i^{(*)}$are the primal variables and the Lagrange multipliers $\alpha^{(*)}$ and $\eta^{(*)}$ are the dual variables.

Differentiating the Lagrange function with respect to the primal variables results in

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)\mathbf{x}_i = 0 \tag{3.65}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0 \tag{3.66}$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \tag{3.67}$$

Eq. (3.65) is the calculation rule for the parameter $\mathbf{w}$. Substituting the derivatives into the Lagrangian results in the dual optimisation problem

$$\begin{aligned} \underset{\alpha_i^{(*)}}{\text{maximise}}\ & W(\alpha_i^{(*)}) = -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\mathbf{x}_i\ \cdot\ \mathbf{x}_j \\ & - \varepsilon\sum_{i=1}^{l}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{l} y_i(\alpha_i^* - \alpha_i) \end{aligned} \tag{3.68}$$

$$\text{subject to } \sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0 \tag{3.69}$$

$$\alpha_i^{(*)} \in [0, C] \tag{3.70}$$

Note that the dual variables $\eta_i^{(*)}$ in Eq. (3.64) have been eliminated through the conditions given in Eq. (3.67).

Different loss functions lead to different dual formulations. The general formulation of the optimisation problem is

$$\underset{\mathbf{w},\xi_i^{(*)}}{\text{minimise}}\ \Phi(\mathbf{w},\xi_i^{(*)}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\mathcal{L}(\xi_i)+\mathcal{L}(\xi_i^*)) \tag{3.71}$$

$$\text{subject to } \mathbf{w}\cdot\mathbf{x}_i + b - y_i \leq \varepsilon + \xi_i \tag{3.72}$$

$$\text{subject to } y_i - \mathbf{w}\cdot\mathbf{x}_i - b - \leq \varepsilon + \xi_i^* \tag{3.73}$$

$$\xi_i^{(*)} \geq 0 \tag{3.74}$$

After calculating the Lagrange function, the following general dual optimisation problem is obtained

$$\begin{aligned}\underset{\alpha_i^{(*)}}{\text{maximise}}\ & W(\alpha_i^{(*)}) = -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^*-\alpha_i)(\alpha_j^*-\alpha_j)\mathbf{x}_i\cdot\mathbf{x}_j + \sum_{i=1}^{l}y_i(\alpha_i^*-\alpha_i) \\ & -\sum_{i=1}^{l}\varepsilon(\alpha_i^*+\alpha_i) + C\sum_{i=1}^{l}(T(\xi_i)+T(\xi_i^*))\end{aligned} \tag{3.75}$$

$$\text{where } T(\xi_i^{(*)}) = \mathcal{L}(\xi_i^{(*)}) - \xi_i^{(*)}\frac{\partial\mathcal{L}(\xi_i^{(*)})}{\partial\xi_i^{(*)}} \tag{3.76}$$

$$\text{subject to } \sum_{i=1}^{l}(\alpha_i^*-\alpha_i) = 0 \tag{3.77}$$

$$0 \leq \alpha_i^{(*)} \leq C\frac{\partial\mathcal{L}(\xi_i^{(*)})}{\partial\xi_i^{(*)}} \tag{3.78}$$

$$0 \leq \xi_i^{(*)} \tag{3.79}$$

$$\xi_i^{(*)} = \inf\left\{\xi_i^{(*)} | C\frac{\partial\mathcal{L}(\xi_i^{(*)})}{\partial\xi_i^{(*)}}\right\} \tag{3.80}$$

Although the formulation of the optimisation problem is different depending on the loss function, all formulations result in the same calculation rule for the gradient $\mathbf{w}$ given in Eq. (3.65). The offset b is calculated using the KKT conditions. Solving Eq. (3.65) for $\mathbf{w}$, expanding Eq. (3.58), and applying the kernel trick leads to the well known εSVM formulation

$$f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_i^*-\alpha_i)k(\mathbf{x}_i,\mathbf{x}) + b. \tag{3.81}$$

3.2.3. Schölkopf's Support Vector Machines for Regression

νSVRs [65] are closely related to εSVRs. While in εSVRs the accuracy is subject to the designer's choice of the maximum permissible error ε, νSVRs aim to minimise the maximum

permissible error for highest possible accuracy. This additional optimisation goal is included into the objective function via the hyperparameter ν

$$\Phi(\mathbf{w}, \xi_i^{(*)}, \varepsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + C(\nu\varepsilon + R_{emp}(\xi_i^{(*)})). \tag{3.82}$$

The parameter ν realises a trade-off between the empirical risk R_{emp} and the maximum permissible error ε. ε is determined during training. This makes ε an additional primal variable. As the νSVM also uses the ε-insensitive loss function, the remainder of the primal optimisation problem changes only marginally: as ε has to be larger than 0, an additional inequality is added:

$$\underset{\mathbf{w},\xi_i^{(*)},\varepsilon}{\text{minimise}}\ \Phi(\mathbf{w}, \xi_i^{(*)}, \varepsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + C(\nu\varepsilon + \sum_{i=1}^{l}(\xi_i + \xi_i^*)) \tag{3.83}$$

$$\text{subject to } (\mathbf{w} \cdot \mathbf{x}_i + b) - y_i \leq \varepsilon + \xi_i \tag{3.84}$$

$$y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \leq \varepsilon + \xi_i^* \tag{3.85}$$

$$\xi_i, \xi_i^*, \varepsilon \geq 0 \tag{3.86}$$

This is the Lagrangian for the νSVR

$$\begin{aligned} L(\mathbf{w}, \xi_i^{(*)}, \varepsilon, \alpha_i^{(*)}, \eta_i^{(*)}, \mu) =& \frac{1}{2}\|\mathbf{w}\|^2 + C\nu\varepsilon + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ & - \mu\varepsilon - \sum_{i=1}^{l}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ & - \sum_{i=1}^{l}\alpha_i(\varepsilon + \xi_i + y_i - \mathbf{w} \cdot \mathbf{x}_i - b) \\ & - \sum_{i=1}^{l}\alpha_i^*(\varepsilon + \xi_i^* + \mathbf{w} \cdot \mathbf{x}_i + b - y_i) \end{aligned} \tag{3.87}$$

Differentiating for primal variables results in

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)\mathbf{x}_i = 0 \tag{3.88}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0 \tag{3.89}$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \tag{3.90}$$

$$\frac{\partial L}{\partial \varepsilon} = C\nu - \sum_{i=1}^{l}(\alpha_i + \alpha_i^*) - \mu = 0 \tag{3.91}$$

As in Vapnik's algorithm, the first of these equations is the calculation rule for the parameter $\mathbf{w}$. All four equations are used to construct the dual optimisation problem

$$\begin{aligned} \underset{\alpha_i^{(*)}}{\text{maximise}} \quad & W(\alpha_i^{(*)}) = -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)\mathbf{x}_i \cdot \mathbf{x}_j \\ & \qquad + \sum_{i=1}^{l} y_i(\alpha_i^* - \alpha_i) \end{aligned} \tag{3.92}$$

$$\text{subject to} \quad \sum_{i=1}^{l}(\alpha_i^* - \alpha_i) = 0 \tag{3.93}$$

$$\alpha_i^{(*)} \in [0, C] \tag{3.94}$$

$$\sum_{i=1}^{l}(\alpha_i^* + \alpha_i) \leq C\nu \tag{3.95}$$

With the calculation rule for $\mathbf{w}$ being the same as in Vapnik's algorithm, the resulting formulation is also the same. Both b and ε can be calculated from the constraints of the primal problem using the KKT-conditions.

3.3. Support Vector Machines in Behavioural Modelling

SVMs are applied in various scientific fields. Common examples are recognition of handwritten characters, image processing, e.g. cancer classification, face detection, brain scan classification, control systems, and financial time series prediction. εSVRs and νSVRs have also been used in automated approaches for behavioural modelling of circuits and systems. Mielenz and Senger presented an automated procedure in 2009 and 2011 respectively [52, 66].

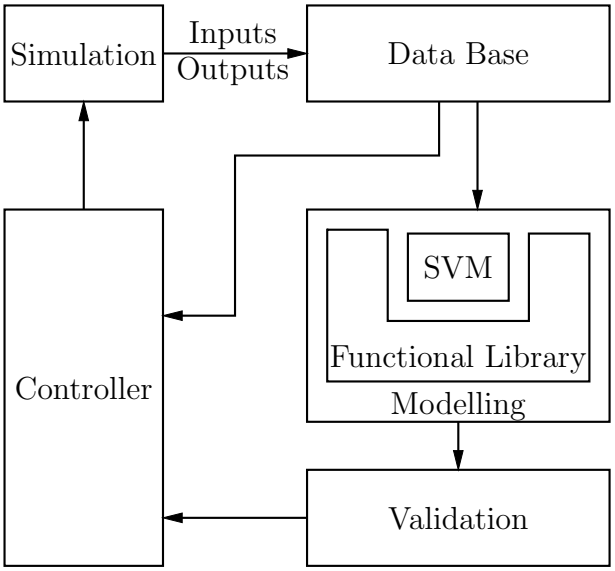


Figure 3.3.: Modelling procedure as presented by Mielenz [52].

Fig. 3.3 presents Mielenz' basic system for an automated iterative modelling procedure. It consists of four main blocks: simulation, controller, modelling and a data base. One iteration starts with simulating the original circuit. The simulation data is saved to the data base. From the data base, data are chosen for generating or adapting the model, an SVM. The controller observes two process variables and determines the setup for the next iteration. As SVMs cannot model dynamic behaviour, dynamics are modelled employing an external functional library [52]. This automated modelling system concentrates on creating transfer function models. It is expanded to provide electrical models, i.e. models which model the V-I relations at each port, and better automation by Senger [66].

4. Variation-Aware Behavioural Modelling Using Support Vector Machines

All standard modelling approaches are used to create models with real valued parameters which give one single value output estimate for each input vector. These models are called nominal models. The aim of this thesis is to create variation-aware behavioural models. Variation-aware behavioural models are models which include the effects of parameter variations. Sec. 1.2.3 introduced affine forms as means for parameters to model these effects. Therefore, this class of models is called affine models.

Algorithms for creating variation-aware behavioural models are based on the SVM algorithm. SVMs are data-based models. Data-based models can only represent information that has already been captured by the data. For nominal models, information is collected by DC, AC or transient simulation of the analogue circuit with different input stimuli. For affine models, the effects of parameter variations have to be included into the data. Three options to achieve this have been presented in Chap. 1: MC simulation, WCA and affine arithmetic simulation.

With additional information available, there are two basic approaches to create models which take parameter variations into account: external variation-aware modelling and internal variation-aware modelling. External variation-aware modelling procedures first train a nominal model using any standard approach presented in Chap. 2 and then expand the model to include the variations. The internal variation-aware modelling procedure modifies the training algorithm to train the parameters of the chosen algorithm to include the variations.

For internal variation-aware modelling, εSVR and νSVR algorithms are expanded to accommodate affine parameters. The new algorithms are denoted by a circumflex over the first letter and are called ε Support Vector Machine for regression with affine parameters ($\hat{\varepsilon}$SVR) and ν Support Vector Machine for regression with affine parameters ($\hat{\nu}$SVR) respectively. SVMs have two parameters which are estimated during training. These two parameters are represented by affine forms. Using the primal-dual approach, new optimisation algorithms are determined. Including information about variations requires additional constraints in the primal optimisation problem. Therefore, the dual optimisation algorithms have more constraints, too. These optimisation problems cannot be solved using the SMO algorithm. The SMO is adapted to solve optimisation problems with additional equality constraints.

4.1. Data Base Representing Effects of Parameter Variations

Black-box models can only represent information which is present in the data base. For modelling analogue circuits, this information is obtained by simulation. Nominal models are created using input-output data pairs which are acquired using DC, AC or transient simulation. The standard for black-box models are data pairs with multiple inputs and a single output (MISO). For affine models, this data base has to be expanded to include information about parameter variations. Affine black-box models are also based on MISO data and cannot model any parameter variations that are not reflected in the output data. This information is obtained using one of the three analysis types which were presented in Chap. 1: MC analysis, WCA, and simulation with affine arithmetic. These three algorithms return different forms of output data which represents the effects of parameter variations. This section details the different forms of data and presents one generalised form which is then used for creating affine black-box models.

WCA returns two output values per input: the minimum and maximum output value that can be obtained for the given parameters. The easiest algorithm for WCA is the CC analysis. More complex algorithms perform a line search on the boundaries of the parameter space or even search the complete parameter space. In any case, data sets obtained from WCA have the following form

$$(\mathbf{x}_1, \underline{y}_1, \overline{y}_1), (\mathbf{x}_2, \underline{y}_2, \overline{y}_2), \ldots, (\mathbf{x}_l, \underline{y}_l, \overline{y}_l) \tag{4.1}$$

where the underline denotes the minimum value and the overline denotes the maximum value of the output. The WCA does not return a nominal value. One way to obtain this information is to run a nominal simulation of the circuit. For symmetric variations of the output around the nominal value there is a second possibility: the mean value of the minimum and maximum value can be calculated.

The MC simulation returns one output value per sampled parameter configuration. The output values contain statistical information such as expected value and standard deviation. The data set takes the form

$$(\mathbf{x}_1, \mathrm{Y}_1), (\mathbf{x}_2, \mathrm{Y}_2), \ldots, (\mathbf{x}_l, \mathrm{Y}_l) \tag{4.2}$$

where Y_i denotes the set of MC output values

$$\mathrm{Y}_i = y_{i,1}, y_{i,2}, \ldots, y_{i,N_{MC}} \tag{4.3}$$

and N_{MC} the number of MC runs. For each input vector $\mathbf{x}_i$ there are N_{MC} output values. The whole data set for a system with m inputs and one output consists of

$$N_{set} = l(m + N_{MC}) \tag{4.4}$$

real values with l as the number of samples. For small data sets, it may be possible to operate on the complete data set. However, standard data sets for black-box modelling are

too big to be operated on: they require too much memory and too much time to be iterated. Therefore, data from MC simulation has to be processed. There are different options for processing: expected value and standard deviation can be extracted but there is also the possibility to extract minimum, maximum and mean value.

The third option for obtaining data is simulation with the AGIAS simulator. This simulator determines an affine form output value for each input value. These output values carry information about which variations contributed to the output value. Data obtained this way has the form

$$(\mathbf{x}_1, \hat{y}_1), (\mathbf{x}_2, \hat{y}_2), \ldots, (\mathbf{x}_l, \hat{y}_l) \tag{4.5}$$

where the affine form output is

$$\hat{y}_i = y_{0,i} + \sum_{k=1}^{l} \varepsilon_k y_{k,i} = y_{0,i} + \sum_{k,P} \varepsilon_{k,P} y_{k,P,i} + \sum_{k,E} \varepsilon_{k,E} y_{,E,i}. \tag{4.6}$$

The noise symbols in the output can be separated into two groups: the first group are parameters of circuit elements, the second group are parameters which cover the linearisation error. The sum which collects all these parameters in Eq. (4.6) is separated into these two groups. The first group is denoted by the index P, the second by the index E. Noise symbols of the first group carry correlation information based on the physical relations. The second group does not carry additional information on physical relations. These noise symbols are purely mathematical.

The form of the data and the information they incorporate are different for each of the simulation approaches. Data from MC and WCA can be obtained for every circuit as those algorithms are common in industrial simulation software. Obtaining data using AGIAS is quite difficult as simulation often fails.

Tailoring a modelling procedure to one of these data sets limits its applicability. This is especially the case if the procedure is tailored to operate on data from the AGIAS as the simulator can only simulate a very limited amount of circuits. To avoid that, all three data sets are transformed to one data representation: input value, and minimum, nominal, and maximum output value.

$$(\mathbf{x}_1, \underline{y}_1, y_1, \overline{y}_1), (\mathbf{x}_2, \underline{y}_2, y_2, \overline{y}_2), \ldots, (\mathbf{x}_l, \underline{y}_l, y_l, \overline{y}_l). \tag{4.7}$$

This form is called min-nominal-max form. From here on, all maximum values are marked by the overline, all minimum values are marked by the underline. Output values without either reference nominal values. If nominal values are not available, the mean value of the minimum and maximum value can be used as an alternative.

Tab. 4.1 lists the different methods which are employed to calculate the min-nominal-max form. This form can most easily obtained from data created by WCA or simulation with AGIAS as the output of the WCA is already very close to the desired form and the AGIAS output can be transformed with very easy calculations.

The missing value from WCA can either be determined by a nominal simulation or by calculating the mean value from the given data. The latter only provides valid information

about a circuit's nominal behaviour if the variations are symmetric to the nominal values. This assumption may not hold for strongly non-linear circuits.

The processing effort for data from AGIAS simulator is almost as simple. The nominal values are part of the affine form, they are the central values. The minimum and maximum values can easily be calculated using the radius of the affine form.

Table 4.1.: Calculation rules to determine data in the form $(\mathbf{x}, \underline{y}, y, \overline{y})$ from different simulation types.

	Minimum	Nominal	Maximum
WCA	$\underline{y}$	$\frac{1}{2}(\overline{y} + \underline{y})$ y from nominal simulation	$\overline{y}$
AGIAS	$y_0 - \mathrm{rad}(\hat{y})$	y_0	$y_0 + \mathrm{rad}(\hat{y})$
MC	min(Y) $E[Y] - n\sigma[\mathrm{Y}]$	$E[\mathrm{Y}]$	max(Y) $E[Y] + n\sigma[\mathrm{Y}]$
	min(Y) $y - n\sigma[\mathrm{Y}]$ $E[\mathrm{Y}] - n\sigma[\mathrm{Y}]$	y from nominal simulation	max(Y) $y + n\sigma[\mathrm{Y}]$ $E[\mathrm{Y}] + n\sigma[\mathrm{Y}]$

Converting data from MC simulation is more complex than for the other approaches. There are two possibilities to obtain a nominal value: nominal simulation or calculating the expected value of the output data $E[\mathrm{Y}]$ for each input. There are also different ways to calculate the minimum and maximum values. The easiest approach is to determine the absolute minimum and maximum from the data Y. However, this approach ignores the statistical nature of the data completely: for MC simulation, parameter deviations are usually modelled by normal distribution functions. The output data is distributed along a normal distribution function. A normal distribution function cannot be used to obtain an absolute minimum and maximum value of the distributed quantity. In MC simulation, the minimum and maximum values are different in each run due to different samples drawn from the parameter space. Minimum and maximum values can be obtained using integer multiples n of the standard deviation $\sigma[\mathrm{Y}]$. This approach determines the extent of validity of the model. In case the expected value and standard deviation are used, the minimum value is calculated as $E[\mathrm{Y}] - n\sigma[\mathrm{Y}]$ and the maximum value as $E[\mathrm{Y}] + n\sigma[\mathrm{Y}]$ respectively.

4.2. Concepts for Creating Variation-Aware Behavioural Models

Chapter 2 introduced and compared different methodologies for modelling. A comparison shows that models with physical parameters can directly be used for simulation with parameter variations, but are difficult to be used in simulation with affine arithmetic. Models

with mathematical parameters can not be used for standard simulation with parameter variations, but are easier to be used for simulation with affine arithmetic as they have often a lower number of parameters. This thesis focuses on SVMs for modelling. SVMs have two mathematical parameters which are calculated during training. Therefore, special concepts for creating variation-aware behavioural models based on SVMs are required.

The standard steps to generate nominal black box models were introduced in Sec. 2.5: the data set, the model structure, a quality criterion ("criterion of fit") and a measure to evaluate this criterion. Fig. 4.1 shows a minimal modelling flow based on these steps. The first step is data generation. From all data generated, two disjoint sets are formed: the training set and the test set. The training set is then used to create the model using a predefined model structure. The test set is used for the final step, validation: here the quality criterion is evaluated using data samples from the test set to determine whether model generation was successful.

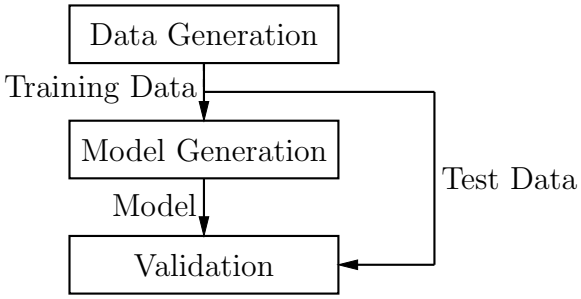


Figure 4.1.: Flow for creating nominal black-box models.

The general modelling flow for external variation-aware modelling concepts is presented in Fig. 4.2. After data generation, a nominal model is created and validated using the nominal values from the data set. For modelling, any existing algorithm for creating black-box models can be used. The nominal model is then expanded into an affine model. For this expansion, the distance between the nominal estimates and the minimum and maximum output values is measured. This information is then used to create an affine formulation. In the final step, the affine model is validated using the test and training data set.

Internal variation-aware modelling has the same flow as nominal modelling, see Fig. 4.3. Only the algorithms used for generating and validating the models are different. As of writing there are no known algorithms to directly infer models with affine parameters. One central problem in using affine arithmetic is that this arithmetic produces new noise symbols in every non-linear calculation. Therefore, there are two options: either use a formulation that is linearly in the affine parameters or use an algorithm that calculates the centres and weights of the affine parameters separately. Constructing an algorithm that is purely linear in the affine expressions is not possible with quadratic optimisation, which is the standard algorithm in training SVMs. Therefore, this work presents algorithms which fall into the second category. Algorithms for creating SVMs with affine parameters are introduced in Sec. 4.3.

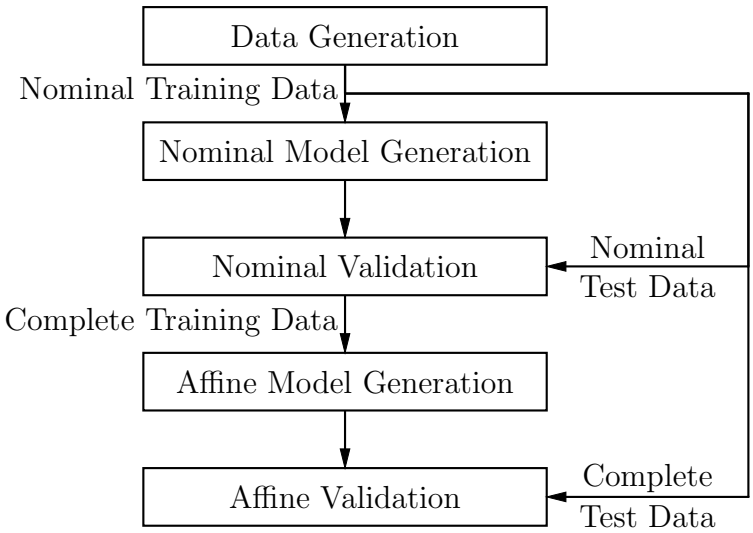


Figure 4.2.: External variation-aware modelling: Flow for creating affine black-box models with external extension.

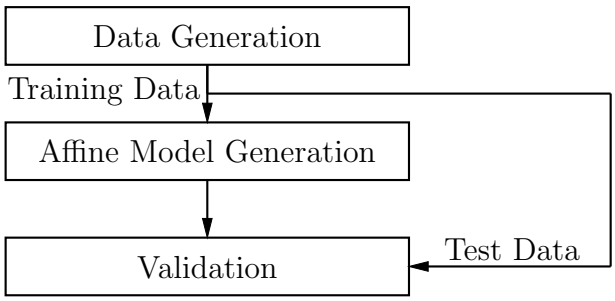


Figure 4.3.: Internal variation-aware modelling: Flow for creating affine black-box models using specialised algorithms.

4.2.1. External Variation-Aware Modelling using Parallel Translation

Sec. 4.2 introduced the basic concept of external variation-aware modelling for models which reproduce a circuit's behaviour with parameter variations. Models which are created following this concept are based on a nominal model. Although any modelling method can be used to create the nominal model, this thesis focuses on SVM models.

Sec. 4.1 introduced a data base in the min-nominal-max form. The nominal values are used to create the nominal model and the expansion to an affine model uses this nominal model and the minimum and maximum values. One method to obtain this expansion is to model the behaviour for minimum and maximum values with two separate models using the same model structure as in the nominal model. Converting these three models into one model with affine parameters is difficult and, in the general case, it is impossible altogether.

The easiest way to incorporate the parameter variations into the model and write the

parameters as affine arithmetic forms is parallel translation. It can be performed on arbitrary mathematical functions. Fig. 4.4 illustrates parallel translation of an arbitrary non-linear function by p_1 both upwards and downwards. In this example the mean value is shown by the dashed line and the solid lines mark the minimum and maximum values.

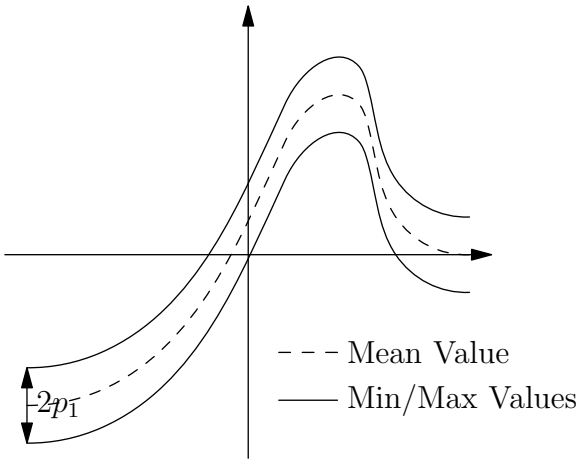


Figure 4.4.: Parallel translation by p_1 upwards and downwards.

Affine forms can be interpreted as a parallel transform of the centre value x_0 by a certain value $\sum_{k=1}^{N} \varepsilon_k x_k$. Therefore, affine forms can be used to represent the parallel translation:

$$\hat{f}(\mathbf{x}) = f(\mathbf{x}) + \varepsilon_1 p_1 \tag{4.8}$$

where the centre value is represented by a nominal estimate $f(\mathbf{x})$ and the variations are represented by the parameter p_1.

Fig. 4.5 shows the more detailed modelling flow for external variation-aware modelling with parallel translation. Data generation is split into two steps: circuit simulation and postprocessing. Simulating the circuit using one of the three methods – WCA, MC simulation and simulation with affine arithmetic – is followed up by a postprocessing step which ensures that data has the min-nominal-max form. After the nominal model has been validated, the distance for the parallel translation is determined. Sec. 1.3 postulated that the affine model encloses the data. This means, the minimum and maximum estimated output values have to lie outside the minimum and maximum output values. The distance parameter is calculated as the maximum distance between the nominal estimates and the minimum and maximum values of the training set.

$$p_1 = \max\{|f(x_i) - \underline{y}_i|, |f(x_i) - \overline{y}_i|\} \qquad \mathbf{x}_i \in \mathcal{X}_{Train}. \tag{4.9}$$

Here p_1 is the distance, $f(x_i)$ is the estimated nominal value and $\underline{y}$ and $\overline{y}$ are the minimum and maximum output values respectively.

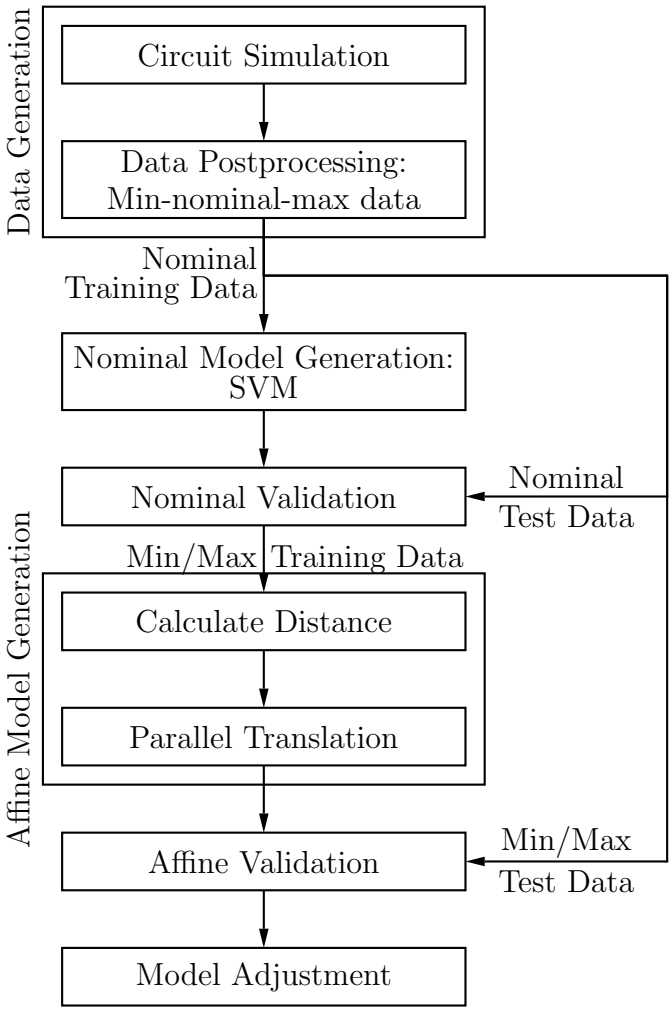


Figure 4.5.: External variation-aware modelling flow: SVMs as nominal model structure with parallel translation.

Chap. 3 introduced SVMs as the basic model structure, which gives the resulting model the mathematical form

$$\hat{f}(\mathbf{x}) = f(\mathbf{x}) + \varepsilon_p p_1 = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) k(\mathbf{x}_i, \mathbf{x}) + b + \varepsilon_p p_1 \tag{4.10}$$

where $\hat{f}(\mathbf{x})$ is the affine model estimate, $f(\mathbf{x})$ is the nominal SVM estimate, p_1 is the distance as per Eq. (4.9) and ε_p is the noise symbol of the affine model.

This affine model $\hat{f}(\mathbf{x})$ is then validated against the minimum and maximum values from the test data set. Validation is based on the same principle as calculating the distance for parallel translation: the model has to enclose the original data. The measure to determine the violation of the original data range is called modelling error. The affine model violates the enclosure condition if its minimum output values are larger than the original minimum

values or if its maximum output values are smaller than the original maximum values respectively. The model is simulated with AGIAS to obtain data for comparison with the test data set. The modelling error e is defined as

$$e = \max\{0, \overline{y}_i - \overline{f(\mathbf{x}_i)}, \underline{f(\mathbf{x}_i)} - \underline{y}_i\} \tag{4.11}$$

where overlined quantities represent maximum values and underlined quantities minimum values. The three arguments of the maximum function represent no violation, violation of the upper boundary, and violation of the lower boundary respectively. A model which encloses the original data returns a modelling error of zero. If the modelling error is not equal to zero, the model can be corrected by using the modelling error e as distance for another parallel translation. This gives the most general form of an affine behavioural model based on parallel translation:

$$\hat{f}(\mathbf{x}) = f(\mathbf{x}) + \varepsilon(p_1 + e). \tag{4.12}$$

Parallel translation has one big disadvantage: it can only generate tubes of constant width independent of the input value. In the example given above, this width is given as

$$2\mathrm{rad}(\hat{f}(\mathbf{x})) = 2(p_1 + e). \tag{4.13}$$

This does not sufficiently model real world data. In real world data the distance between the minimum and maximum varies over the input values. One such example is the class A amplifier which was used to introduce simulation of circuits wit parameter variations: the top graph in Fig. 1.2c shows a plot of the output values over the input values. While in the saturation region the output range is very small, it is quite large along the operating region.

4.2.2. Internal Variation-Aware Modelling with Support Vector Machines

External variation-aware modelling with parallel translation provides a fast, all-round algorithm for creating variation-aware behavioural models. The resulting models have a constant distance between minimum, mean and maximum values. Usually, the original data describes ranges of varying width. Therefore, the overall accuracy of models created by the external variation-aware modelling flow from Sec. 4.2.1 is rather low.

Internal variation-aware modelling can overcome the disadvantage of parallel translation by directly varying the parameters of the model. This offers the possibility to produce model output ranges with varying width for different input values. The higher overall accuracy of these models is traded off for the complexity of the training algorithm.

Using affine forms as parameters introduces a new difficulty to modelling. When training non-linear models, the optimisation algorithms are often non-linear in the model parameters. When trying to train affine parameters directly, non-affine operations introduce additional noise symbols to enclose the linearisation error. This makes it difficult to determine model parameters.

There are two ways to avoid additional noise symbols: constructing an algorithm with only affine operations or estimate centre values and weights separately. Only addition, subtraction and multiplication with a constant are affine operations. A training algorithm based only on these operations could not be formulated. In original SVMs, the gradient is calculated as an expansion of the Lagrange multipliers. These multipliers are determined by a quadratic optimisation problem. Determining an affine gradient directly means that the multipliers are represented by affine forms, too. The calculation rule for the affine gradient $\hat{\mathbf{w}}$ is derived from Eq. (3.65) by solving for $\mathbf{w}$ and replacing $\mathbf{w}$ and the Lagrange multipliers by affine forms:

$$\hat{\mathbf{w}} = \sum_{i=1}^{l} (\hat{\alpha}_i^* - \hat{\alpha}_i)\mathbf{x}_i \tag{4.14}$$

with the affine multipliers $\hat{\alpha}_i^{(*)}$. These multipliers are determined by the quadratic optimisation problem

$$\begin{aligned} \underset{\hat{\alpha}_i^{(*)}}{\text{maximise}} \quad & W(\hat{\alpha}_i^{(*)}) = -\frac{1}{2}\sum_{i,j=1}^{l}(\hat{\alpha}_i^* - \hat{\alpha}_i)(\hat{\alpha}_j^* - \hat{\alpha}_j)\mathbf{x}_i \cdot \mathbf{x}_j \\ & \qquad - \varepsilon\sum_{i=1}^{l}(\hat{\alpha}_i^* + \hat{\alpha}_i) + \sum_{i=1}^{l} y_i(\hat{\alpha}_i^* - \hat{\alpha}_i) \end{aligned} \tag{4.15}$$

$$\text{subject to } \sum_{i=1}^{l}(\hat{\alpha}_i^* - \hat{\alpha}_i) = 0 \tag{4.16}$$

$$\hat{\alpha}_i^{(*)} \in [0, C] \tag{4.17}$$

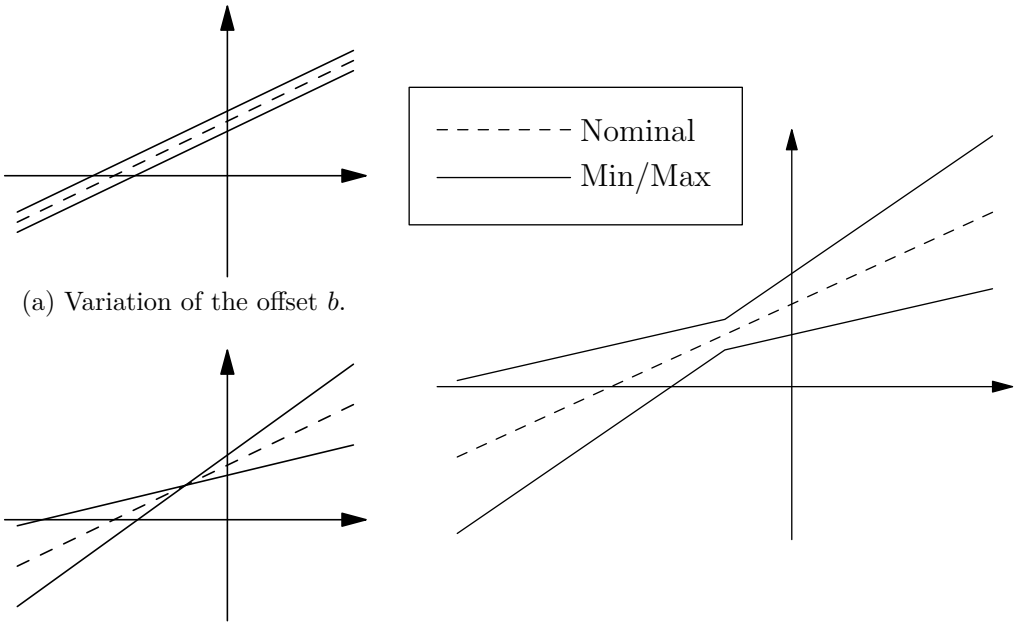
Assuming each multiplier $\hat{\alpha}_i^{(*)}$ has one correlated noise symbol, the optimisation problem introduces new noise symbols each time the target function is evaluated as the square function is a non-affine operation. Standard solvers for quadratic optimisation problems use non-affine operations and so does the SMO. None of the known algorithms could be written without non-affine operations.

The second option separates each affine parameter into multiple real-valued parameters: one for the centre value and one for each weight. This enlarges the algorithm significantly. The size of the algorithm translates directly into solver run-time. Furthermore, the data base has to contain enough information to train multiple weights simultaneously without constructing an additional optimisation problem for the weights.

SVMs have two parameters which are determined during training: the gradient and the offset. Using affine forms for both parameters allows to model different output ranges. As the data only contain information about the centre values and the envelope of the data – the minimum and maximum values – one independent noise symbol per parameter is introduced. To determine the centre values and weights additional constraints are included into the optimisation problem.

4.3. Extending the Support Vector Algorithm

Sec. 3.2 introduces training algorithms that determine a hyperplane. In this extension, the hyperplane's parameters are replaced by affine forms: the gradient $\hat{\mathbf{w}}$ and the offset $\hat{b}$. Fig. 4.6 shows the effects of varying these two parameters: varying the offset is a parallel translation (see Fig. 4.6a); varying the gradient causes the planes to swing open (see Fig. 4.6b). Combining these two results in a bow shape as shown in Fig. 4.6c. This allows the modelling of different widths of the output values and with the kernel trick this solution extends to non-linear problems.



(a) Variation of the offset b.

(b) Variation of the slope w.

(c) Variation of both offset b and slope w.

Figure 4.6.: Variations of the parameters of a straight line $g(x) = wx + b$.

The min-nominal-max form of the data base emphasises the envelope of the data. The hyperplane is extended to describe a range by the affine parameters. Affine forms can contain an arbitrary number of noise symbols. The number chosen for the new SVM algorithms is limited by the information available from the data base and the size of the resulting algorithm. As the weights are determined separately, each additional noise symbol increases the size of the algorithm. With the emphasis on the envelope of the data, one noise symbol per parameter is used. This leads to the affine hyperplane formulation

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{w}} \cdot \mathbf{x} + \hat{b} = (\mathbf{w}_0 + \varepsilon_w \mathbf{w}_1) \cdot \mathbf{x} + b_0 + \varepsilon_b b_1. \tag{4.18}$$

This formulation introduces the affine gradient parameter $\mathbf{w} = \mathbf{w}_0 + \mathbf{w}_1 \varepsilon_w$ and the affine offset $b = b_0 + b_1 \varepsilon_b$. The parameters have been expanded to affine forms using two uncorrelated

noise symbols ε_w and ε_b respectively. The weights of these noise symbols, $\mathbf{w}_1$ and b_1, are the two new parameters which have to be determined by the optimisation algorithm.

As illustrated in Fig. 4.6a, the parameter b_1 realises a parallel translation. The maximum permissible error ε can be interpreted similarly: by using the ε-insensitive loss function, a tube is constructed around the original values which marks the area for correct estimates. This tube is constructed by parallel translation by the maximum permissible error. As both ε and b_1 represent an area that is constructed by parallel translation, b_1 is chosen as

$$b_1 = \varepsilon. \tag{4.19}$$

At this point note should be taken that the letter ε is used for two purposes in the context of this thesis: it denotes the noise symbol in affine forms and the maximum permissible error in the ε-insensitive loss-function. To distinguish these two uses, the following convention is used: if the ε has an index, it represents a noise symbol in an affine form; else, it represents the maximum permissible error of the ε-insensitive loss function.

In the original algorithm, the regularisation term represents the geometric interpretation of searching the hyperplane with the lowest gradient. This basic idea from the original algorithm is still used to determine the centre values. Following Sec. 1, the model should enclose the data as closely as possible. This means the space enclosed by the minimum and maximum hyperplanes should be minimal. The minimum and maximum values for the hyperplane given in Eq. (4.18) are calculated as follows

$$\overline{\hat{f}(\mathbf{x})} = \mathbf{w}_0\mathbf{x} + |\mathbf{w}_1\mathbf{x}| + b_0 + |b_1| \tag{4.20}$$

$$\underline{\hat{f}(\mathbf{x})} = \mathbf{w}_0\mathbf{x} - |\mathbf{w}_1\mathbf{x}| + b_0 - |b_1| \tag{4.21}$$

In the one dimensional case, the area between $\overline{\hat{f}(x)}$ and $\underline{\hat{f}(x)}$ is calculates as

$$A = \int_{\underline{x}}^{\overline{x}} (\overline{\hat{f}(x)} - \underline{\hat{f}(x)})dx \tag{4.22}$$

$$= \int_{\underline{x}}^{\overline{x}} 2(|w_1 x| + |b_1|)dx \tag{4.23}$$

$$= \int_{\underline{x}}^{0} 2(-\frac{1}{2}|w_1|x + |b_1|)dx + \int_{0}^{\overline{x}} 2(\frac{1}{2}|w_1|x + |b_1|)dx \tag{4.24}$$

$$= |w_1|(\overline{x}^2 + \underline{x}^2) + 2|b_1|(\overline{x} - \underline{x}) \tag{4.25}$$

with $\overline{x}$ as the maximum and $\underline{x}$ as the minimum input values. The resulting formula is still dependent on the weights of both noise symbols. If this is used to expand the optimisation as is, the resulting optimisation problem cannot be used using quadratic programming. As b_1 is chosen equal to the maximum permissible error, only the first term of the sum is considered for optimisation. As data can always be scaled so that the factor $(\overline{x}^2 + \underline{x}^2)$ equals 1, only the term $\mathbf{w}_1$ is included into the optimisation problem. $\mathbf{w}_1$ is considered as a second regularisation term, analogously to $\frac{1}{2}\|\mathbf{w}_0\|^2$. Minimising $\mathbf{w}_1$ means minimising the variation of the gradient.

The next step is to apply a cost function to the distance between the minimum and maximum estimates and original minimum and maximum values. This cost function is the ε-insensitive loss function. And as for the nominal values, constraints are calculated from this quality measure. Instead of two constraints for the nominal values, six constraints with two for minimum, nominal and maximum values each are included into the optimisation problem.

$$(\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) - y_i \leq \varepsilon + \xi_i \tag{4.26}$$
$$y_i - (\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) \leq \varepsilon + \xi_i^* \tag{4.27}$$
$$(\mathbf{w}_0 \cdot \mathbf{x}_i + |\mathbf{w}_1 \cdot \mathbf{x}_i| + b_0 + |b_1|) - \overline{y}_i \leq \varepsilon + \zeta_i \tag{4.28}$$
$$\overline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i + |\mathbf{w}_1 \cdot \mathbf{x}_i| + b_0 + |b_1|) \leq \varepsilon + \zeta_i^* \tag{4.29}$$
$$(\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b_0 - |b_1|) - \overline{y}_i \leq \varepsilon + \delta_i \tag{4.30}$$
$$\overline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b_0 - |b_1|) \leq \varepsilon + \delta_i^* \tag{4.31}$$

where $\xi_i^{(*)}$are the slack variables to measure misestimation of the nominal values, and $\zeta_i^{(*)}$ and $\delta_i^{(*)}$ are the slack variables to collect misestimation of maximum and minimum values respectively.

With the choice of b_1, the new regularisation term based on $\mathbf{w}_1$, and the additional constraints for minimum and maximum values, new SVM algorithms can be constructed. The following sections present new SV algorithms which estimate hyperplanes with variations. One group is based on Vapnik's εSVR algorithm, the other on Schölkopf's νSVR algorithm. As these algorithms are used for modelling analogue circuits, only algorithms for regression are presented.

4.3.1. ε Support Vector Machines with Affine Parameters

The ε̂SVR is the affine expansion of the εSVR. The training algorithm estimates a hyperplane where both the gradient and the offset are represented as affine parameters

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{w}} \cdot \mathbf{x} + \hat{b} \tag{4.32}$$

using the ε-insensitive loss function. With an additional regularisation term based on $\mathbf{w}_1$, additional constraints for minimum and maximum values (given by Eq. (4.26) to Eq. (4.31)), and additional slack variables, the following optimisation problem is obtained

$$\begin{aligned} \underset{\mathbf{w}_0,\mathbf{w}_1,\xi_i^{(*)},\zeta_i^{(*)},\delta_i^{(*)}}{\text{minimise}} \quad & \Phi(\mathbf{w}_0,\mathbf{w}_1,\xi_i^{(*)},\zeta_i^{(*)},\delta_i^{(*)}) = \frac{1}{2}\|\mathbf{w}_0|^2 + \frac{1}{2}\|\mathbf{w}_1\|^2 \\ & \qquad + C\sum_{i=1}^{l}(\xi_i + \xi_i^* + \zeta_i + \zeta_i^* + \delta_i + \delta_i^*) \end{aligned} \tag{4.33}$$

$$\text{subject to } (\mathbf{w} \cdot \mathbf{x}_i + b) - y_i \leq \varepsilon + \xi_i \tag{4.34}$$
$$y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \leq \varepsilon + \xi_i^* \tag{4.35}$$

$$(\mathbf{w}_0 \cdot \mathbf{x}_i + |\mathbf{w}_1 \cdot \mathbf{x}_i| + b + |b_1|) - \overline{y}_i \leq \varepsilon + \zeta_i \tag{4.36}$$
$$\overline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i + |\mathbf{w}_1 \cdot \mathbf{x}_i| + b + |b_1|) \leq \varepsilon + \zeta_i^* \tag{4.37}$$
$$(\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) - \underline{y}_i \leq \varepsilon + \delta_i \tag{4.38}$$
$$\underline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) \leq \varepsilon + \delta_i^* \tag{4.39}$$
$$\xi_i^{(*)}, \zeta_i^{(*)}, \delta_i^{(*)} \geq 0 \tag{4.40}$$

For creating the Lagrangian, additional multipliers $\beta_i^{(*)}$, $\gamma_i^{(*)}$, $\kappa_i^{(*)}$, and $\lambda_i^{(*)}$ are introduced. This leads to the Lagrangian

$$\begin{aligned}
&L(\mathbf{w}_0, \mathbf{w}_1, b_0, b_1, \xi_i^{(*)}, \zeta_i^{(*)}, \delta_i^{(*)}; \alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)}, \eta_i^{(*)}, \kappa_i^{(*)}, \lambda_i^{(*)}) \\
&\quad = \frac{1}{2}\|w_0\|^2 + \frac{1}{2}\|w_1\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^* + \zeta_i + \zeta_i^* + \delta_i + \delta_i^*) \\
&\quad - \sum_{i=1}^{l} \alpha_i(\varepsilon - \xi_i + y_i - (\mathbf{w} \cdot \mathbf{x}_i + b)) \\
&\quad - \sum_{i=1^{l}} \alpha_i^*(\varepsilon - \xi_i^* + (\mathbf{w} \cdot \mathbf{x}_i + b) - y_i) \\
&\quad - \sum_{i=1}^{l} \beta_i(\varepsilon + \zeta_i + \overline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|)) \\
&\quad - \sum_{i=1}^{l} \beta_i^*(\varepsilon + \zeta_i^* + (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) - \overline{y}_i) \\
&\quad - \sum_{i=1}^{l} \gamma_i(\varepsilon + \delta_i + \underline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|)) \\
&\quad - \sum_{i=1}^{l} \gamma_i^*(\varepsilon + \delta_i^* + (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) - \underline{y}_i) \\
&\quad - \sum_{i=1}^{l} \eta_i \xi_i - \sum_{i=1}^{l} \eta_i^* \xi_i^* - \sum_{i=1}^{l} \kappa_i \zeta_i - \sum_{i=1}^{l} \kappa_i^* \zeta_i^* - \sum_{i=1}^{l} \lambda_i \delta_i - \sum_{i=1}^{l} \lambda_i^* \delta_i^*
\end{aligned} \tag{4.41}$$

with $\alpha^{(*)}, \beta^{(*)}, \gamma^{(*)}, \eta^{(*)}, \kappa^{(*)}, \lambda^{(*)} \geq 0$. Using affine forms results on additional primal variables. Deriving the Lagrangian for the primal variables results in

$$\frac{\partial L}{\partial \mathbf{w}_0} = \mathbf{w}_0 - \sum_{i=1}(\alpha_i^* - \alpha_i^* + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i)\mathbf{x}_i = 0 \tag{4.42}$$
$$\frac{\partial L}{\partial \mathbf{w}_1} = \mathbf{w}_1 - \sum_{i=1}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) \begin{pmatrix} \operatorname{sgn}(\mathbf{w}_{1,1})|\mathbf{x}_{i,1}| \\ \vdots \\ \operatorname{sgn}(\mathbf{w}_{1,N})|\mathbf{x}_{i,N}| \end{pmatrix} = 0 \tag{4.43}$$
$$\frac{\partial L}{\partial b_0} = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) = 0 \tag{4.44}$$

$$\frac{\partial L}{\partial b_1} = \sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) = 0 \tag{4.45}$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \tag{4.46}$$

$$\frac{\partial L}{\partial \zeta_i^{(*)}} = C - \beta_i^{(*)} - \kappa_i^{(*)} = 0 \tag{4.47}$$

$$\frac{\partial L}{\partial \delta_i^{(*)}} = C - \gamma_i^{(*)} - \lambda_i^{(*)} = 0. \tag{4.48}$$

In the original algorithm, the derivative for $\mathbf{w}$ forms the calculation rule for the gradient parameter. This holds for the derivative for $\mathbf{w}_0$ here, but not for the derivative for $\mathbf{w}_1$. With the sign function, Eq. (4.43) can neither be solved for $\mathbf{w}_1$ nor be used to create an optimisation problem that can be solved by quadratic or convex programming. The sign function can be removed from the equations by using the symmetry property of affine forms. Consider the standard notation of the affine form representing the gradient

$$\hat{\mathbf{w}} = \mathbf{w}_0 + \mathbf{w}_1\varepsilon_w \qquad \varepsilon_w \in [-1, +1]. \tag{4.49}$$

In the one dimensional case, solving Eq. (4.43) for $\mathbf{w}_1$ while ignoring the sign function results in

$$w_1 = \operatorname{sgn}(w_1)\sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)|x_i|. \tag{4.50}$$

When substituting this function into the affine form, the parameter w_1 is multiplied by a noise symbol. The sign function can return three values: -1 for negative values, 0 for zero and $+1$ for positive input values. The noise symbol represents the interval $[-1, +1]$. Multiplying these quantities,

$$\operatorname{sgn}(w1) \cdot [-1, +1] = [-1, +1] \cdot \begin{cases} +1 & \text{if } w_1 < 0 \\ 0 & \text{if } w_1 = 0 \\ -1 & \text{if } w_1 > 0 \end{cases} \tag{4.51}$$

$$= [-1, +1], \tag{4.52}$$

shows that the resulting interval is $[-1, +1]$. Therefore, the sign function can be dropped from Eq. (4.43). The resulting derivative is

$$\frac{\partial L}{\partial \mathbf{w}_1} = \mathbf{w}_1 - \sum_{i=1}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)|\mathbf{x}_i| = 0 \tag{4.53}$$

and the resulting calculation rule can be determined by solving Eq. (4.53) for $\mathbf{w}_1$.

Substituting the derivatives into the Lagrangian results in the following dual optimisation problem

$$\begin{aligned} W(\alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)}) = \\ & -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) \\ & \qquad (\alpha_j^* - \alpha_j + \beta_j^* - \beta_j + \gamma_j^* - \gamma_j)\mathbf{x}_i \cdot \mathbf{x}_j \\ \underset{\alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)}}{\text{maximise}} & -\frac{1}{2}\sum_{i,j=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)(\beta_j^* - \beta_j - \gamma_j^* + \gamma_j)|\mathbf{x}_i \cdot \mathbf{x}_j| \\ & -\varepsilon(\alpha_i^* + \alpha_i + \beta_i^* + \beta_i + \gamma_i^* + \gamma_i) \\ & +\sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i + \sum_{i=1}^{l}(\beta_i^* - \beta_i)\overline{y}_i + \sum_{i=1}^{l}(\gamma_i^* - \gamma_i)\underline{y}_i \end{aligned} \tag{4.54}$$

$$\text{subject to } \sum_{i=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) = 0 \tag{4.55}$$

$$\sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) = 0 \tag{4.56}$$

$$0 \leq \alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)} \leq C \tag{4.57}$$

The multipliers $\eta_i^{(*)}$, $\kappa_i^{(*)}$, and $\lambda_i^{(*)}$ have been eliminated from the optimisation problem by substituting the derivatives into the primal optimisation problem and by the fact that Lagrange multipliers are larger than or equal to zero by definition.

From the derivatives, two calculation rules are obtained: one for $\mathbf{w}_0$ and one for $\mathbf{w}_1$

$$\mathbf{w}_0 = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i)\mathbf{x}_i \tag{4.58}$$

$$\mathbf{w}_1 = \sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)|\mathbf{x}_i|. \tag{4.59}$$

With b_1 chosen to equal ε, only one parameter is left to be determined: b_0. Here the same approach as in the original algorithm is employed and b_0 is determined using the KKT conditions.

The SVM formulation from Eq. (4.32) is then expanded to

$$\begin{aligned} \hat{f}(\mathbf{x}) = & \sum_{i=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i)k(\mathbf{x}_i, \mathbf{x}) \\ & + \varepsilon_w \sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)|k(\mathbf{x}_i, \mathbf{x})| \\ & + b_0 + \varepsilon_b \varepsilon \end{aligned} \tag{4.60}$$

where k denotes the kernel function used to solve non-linear problems.

This algorithm uses one set of constraints for each minimum, nominal, and maximum values, resulting in six constraints in total. Solving the optimisation problem requires to calculate the kernel matrix. For smaller problems the kernel matrix can be calculated completely. With a kernel matrix at 9 times the size of the matrix of a nominal algorithm, the ŝSVR algorithm is almost guaranteed to run into memory problems. Additionally, this algorithm does not use the full potential of the symmetry property of affine forms.

The size of the algorithm can be reduced by removing the constraints associated with the nominal value from the primal optimisation problem. This also represents the underlying assumption that data is symmetrical to the nominal value and therefore the nominal value can be determined from the minimum and maximum values. Doing so removes the Lagrange multipliers $\alpha_i^{(*)}$ from the resulting dual optimisation problem and the solutions for the expansion of the hyperplane. To differentiate between the two forms, the algorithm with the full set is called ŝSVR6 and the reduced size algorithm is called ŝSVR4 with both numbers referencing the number of constraints in the primal optimisation problem.

The resulting dual optimisation problem for the ŝSVR4 is

$$\begin{aligned}
\underset{\beta_i^{(*)},\gamma_i^{(*)}}{\text{maximise}} \quad & W(\beta_i^{(*)},\gamma_i^{(*)}) = -\frac{1}{2}\sum_{i,j=1}^{l}(\beta_i^* - \beta_i + \gamma_i^* - \gamma_i)(\beta_j^* - \beta_j + \gamma_j^* - \gamma_j)\mathbf{x}_i \cdot \mathbf{x}_j \\
& \quad -\frac{1}{2}\sum_{i,j=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)(\beta_j^* - \beta_j - \gamma_j^* + \gamma_j)|\mathbf{x}_i \cdot \mathbf{x}_j| \\
& \quad -\varepsilon(\beta_i^* + \beta_i + \gamma_i^* + \gamma_i) \\
& \quad +\sum_{i=1}^{l}(\beta_i^* - \beta_i)\overline{y}_i + \sum_{i=1}^{l}(\gamma_i^* - \gamma_i)\underline{y}_i
\end{aligned} \tag{4.61}$$

$$\text{subject to} \quad \sum_{i=1}^{l}(\beta_i^* - \beta_i + \gamma_i^* - \gamma_i) = 0 \tag{4.62}$$

$$\sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) = 0 \tag{4.63}$$

$$0 \leq \beta_i^{(*)}, \gamma_i^{(*)} \leq C \tag{4.64}$$

Only the calculation rule for the gradient's centre value is changed:

$$\mathbf{w}_0 = \sum_{i=1}^{l}(\beta_i^* - \beta_i + \gamma_i^* - \gamma_i)\mathbf{x}_i \tag{4.65}$$

and the expansion of the SVM hyperplane is

$$\begin{aligned}\hat{f}(\mathbf{x}) = &\sum_{i=1}^{l}(\beta_i^* - \beta_i + \gamma_i^* - \gamma_i)k(\mathbf{x}_i, \mathbf{x}) \\ &+ \varepsilon_w \sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)|k(\mathbf{x}_i, \mathbf{x})| \\ &+ b_0 + \varepsilon_b \varepsilon\end{aligned} \tag{4.66}$$

where k denotes the kernel function used to solve non-linear problems [40].

4.3.2. ν Support Vector Machines with Affine Parameters

The $\hat{\varepsilon}$SVR is based on the εSVR and comes with the same basic advantages and disadvantages: it offers designers full control over the accuracy, but it is difficult to use when trying to obtain the best model in terms of the maximum permissible error. To create affine models with the smallest maximum permissible error ε, the νSVR algorithm is used as a basis. The new algorithm is called $\hat{\nu}$SVR accordingly.

Just as its parent algorithm, it extends the $\hat{\varepsilon}$SVR primal optimisation problem by the parameter ν and includes the parameter ε as an additional primal variable.

$$\begin{aligned}&\underset{\mathbf{w}_0, \mathbf{w}_1, \varepsilon; \xi_i^{(*)}, \zeta_i^{(*)}, \delta_i^{(*)}}{\text{minimise}} && \Phi(\mathbf{w}_0, \mathbf{w}_1, \varepsilon; \xi_i^{(*)}, \zeta_i^{(*)}, \delta_i^{(*)}) = \frac{1}{2}\|w_0\|^2 + \frac{1}{2}\|w_1\|^2 \\ &&& \quad + C(\nu\varepsilon + \sum_{i=1}^{l}(\xi_i + \xi_i^* + \zeta_i + \zeta_i^* + \delta_i + \delta_i^*)) \qquad (4.67) \\ &\text{subject to} && (\mathbf{w} \cdot \mathbf{x}_i + b) - y_i \leq \varepsilon + \xi_i \qquad (4.68) \\ &&& y_i - (\mathbf{w} \cdot \mathbf{x}_i + b) \leq \varepsilon + \xi_i^* \qquad (4.69) \\ &&& (\mathbf{w}_0 \cdot \mathbf{x}_i + |\mathbf{w}_1 \cdot \mathbf{x}_i| + b + |b_1|) - \overline{y}_i \leq \varepsilon + \zeta_i \qquad (4.70) \\ &&& \overline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i + |\mathbf{w}_1 \cdot \mathbf{x}_i| + b + |b_1|) \leq \varepsilon + \zeta_i^* \qquad (4.71) \\ &&& (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) - \underline{y}_i \leq \varepsilon + \delta_i \qquad (4.72) \\ &&& \underline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) \leq \varepsilon + \delta_i^* \qquad (4.73) \\ &&& \xi_i^{(*)}, \zeta_i^{(*)}, \delta_i^{(*)} \geq 0 \qquad (4.74)\end{aligned}$$

The parameter ν has to be chosen by the designer before training. The same boundaries apply to choosing ν as for the original νSVR algorithm.

With additional multipliers for the new primal variable, the following Lagrangian is obtained

$$\begin{aligned}
L(\mathbf{w}_0, \mathbf{w}_1, b_0, b_1, \varepsilon; \xi_i^{(*)}, \zeta_i^{(*)}, \delta_i^{(*)}) = {} & \frac{1}{2}\|w_0\|^2 + \frac{1}{2}\|w_1\|^2 + C\nu\varepsilon \\
& + C\sum_{i=1}^{l}(\xi_i + \xi_i^* + \zeta_i + \zeta_i^* + \delta_i + \delta_i^*) \\
& - \sum_{i=1}^{l}\alpha_i(\varepsilon - \xi_i + y_i - (\mathbf{w} \cdot \mathbf{x}_i + b)) \\
& - \sum_{i=1^l}\alpha_i^*(\varepsilon - \xi_i^* + (\mathbf{w} \cdot \mathbf{x}_i + b) - y_i) \\
& - \sum_{i=1}^{l}\beta_i(\varepsilon + \zeta_i + \overline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|)) \\
& - \sum_{i=1}^{l}\beta_i^*(\varepsilon + \zeta_i^* + (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) - \overline{y}_i) \\
& - \sum_{i=1}^{l}\gamma_i(\varepsilon + \delta_i + \underline{y}_i - (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|)) \\
& - \sum_{i=1}^{l}\gamma_i^*(\varepsilon + \delta_i^* + (\mathbf{w}_0 \cdot \mathbf{x}_i - |\mathbf{w}_1 \cdot \mathbf{x}_i| + b - |b_1|) - \underline{y}_i) \\
& - \sum_{i=1}^{l}\eta_i\xi_i - \sum_{i=1}^{l}\eta_i^*\xi_i^* - \sum_{i=1}^{l}\kappa_i\zeta_i - \sum_{i=1}^{l}\kappa_i^*\zeta_i^* - \sum_{i=1}^{l}\lambda_i\delta_i - \sum_{i=1}^{l}\lambda_i^*\delta_i^* - \mu\varepsilon
\end{aligned} \tag{4.75}$$

where $\alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)}, \eta_i^{(*)}, \kappa_i^{(*)}, \lambda_i^{(*)}, \mu \geq 0$ are the Lagrange multipliers. This Lagrange formulation differs from the formulation for the $\hat{\varepsilon}$SVR by only one term: $\mu\varepsilon$. Deriving this Lagrangian for the primal variables results in

$$\frac{\partial L}{\partial \mathbf{w}_0} = \mathbf{w}_0 - \sum_{i=1}(\alpha_i^* - \alpha_i^* + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i)\mathbf{x} = 0 \tag{4.76}$$

$$\frac{\partial L}{\partial \mathbf{w}_1} = \mathbf{w}_1 - \sum_{i=1}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)|\mathbf{x}_i| = 0 \tag{4.77}$$

$$\frac{\partial L}{\partial b_0} = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) = 0 \tag{4.78}$$

$$\frac{\partial L}{\partial b_1} = \sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) = 0 \tag{4.79}$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - \alpha_i^{(*)} - \eta_i^{(*)} = 0 \tag{4.80}$$

$$\frac{\partial L}{\partial \zeta_i^{(*)}} = C - \beta_i^{(*)} - \kappa_i^{(*)} = 0 \tag{4.81}$$

$$\frac{\partial L}{\partial \delta_i^{(*)}} = C - \gamma_i^{(*)} - \lambda_i^{(*)} = 0 \tag{4.82}$$

$$\frac{\partial L}{\partial \varepsilon} = C\nu - \mu - \sum_{i=1}^{l}(\alpha_i^* + \alpha_i + \beta_i^* + \beta_i + \gamma_i^* + \gamma_i) = 0. \tag{4.83}$$

The sign function in Eq. (4.77) has been eliminated following the reasoning given in Sec. 4.3.1.

Substituting these derivatives into the Lagrangian results in the dual optimisation problem for ŷSVRs

$$\begin{aligned} W(\alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)}) = \\ -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) \\ (\alpha_j^* - \alpha_j + \beta_j^* - \beta_j + \gamma_j^* - \gamma_j)\mathbf{x}_i \cdot \mathbf{x}_j \\ -\frac{1}{2}\sum_{i,j=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)(\beta_j^* - \beta_j - \gamma_j^* + \gamma_j)|\mathbf{x}_i \cdot \mathbf{x}_j| \\ +\sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i + \sum_{i=1}^{l}(\beta_i^* - \beta_i)\overline{y}_i + \sum_{i=1}^{l}(\gamma_i^* - \gamma_i)\underline{y}_i \end{aligned} \tag{4.84}$$

$$\text{subject to } \sum_{i=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) = 0 \tag{4.85}$$

$$\sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) = 0 \tag{4.86}$$

$$C\nu - \sum_{i=0}^{l}(\alpha_i^* + \alpha_i + \beta_i^* + \beta_i + \gamma_i^* + \gamma_i) \geq 0 \tag{4.87}$$

$$0 \leq \alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)} \leq C \tag{4.88}$$

The multipliers $\eta_i^{(*)}$, $\kappa_i^{(*)}$, $\lambda_i^{(*)}$, and μ are eliminated in the process of creating the dual optimisation problem.

The calculation rules for the gradient parameters $\mathbf{w}_0$ and $\mathbf{w}_1$ for the ŷSVR6 are the same as for the ε̂SVR. Therefore, the expansion for the original hyperplane is also the same. As for determining the offset parameters b_0 and b_1 are determined by the KKT conditions.

As for ε̂SVRs, a reduced algorithm with four constraints is obtained by removing the nominal value and its respctive constraints from the training algorithm. The resulting dual

optimisation problem for the ν̂SVR4 is

$$
\begin{aligned}
W(\beta_i^{(*)}, \gamma_i^{(*)}) = &-\frac{1}{2}\sum_{i,j=1}^{l}(\beta_i^* - \beta_i + \gamma_i^* - \gamma_i)(\beta_j^* - \beta_j + \gamma_j^* - \gamma_j)\mathbf{x}_i \cdot \mathbf{x}_j \\
\underset{\beta_i^{(*)},\gamma_i^{(*)}}{\text{maximise}} \quad &-\frac{1}{2}\sum_{i,j=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)(\beta_j^* - \beta_j - \gamma_j^* + \gamma_j)|\mathbf{x}_i \cdot \mathbf{x}_j| \qquad (4.89) \\
&+\sum_{i=1}^{l}(\beta_i^* - \beta_i)\overline{y}_i + \sum_{i=1}^{l}(\gamma_i^* - \gamma_i)\underline{y}_i \\
\text{subject to } &\sum_{i=1}^{l}(\beta_i^* - \beta_i + \gamma_i^* - \gamma_i) = 0 \qquad (4.90) \\
&\sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) = 0 \qquad (4.91) \\
&0 \le \beta_i^{(*)}, \gamma_i^{(*)} \le C \qquad (4.92)
\end{aligned}
$$

The changes to the algorithms have the same consequences as for the ε̂SVR4 algorithm.

4.4. Implementation

Sec. 1.3 introduced the basic conditions for generating variation-aware behavioural models: affine parameters, enclosure of the original data, no piecewise or strongly non-linear functions, and semi-automated generation of models. Furthermore, affine arithmetic should not be used for training and the resulting models have to be simulated successfully with the AGIAS simulator. The structure of the new SVM algorithms guarantees all but enclosure, semi-automated generation and models which can be simulated in AGIAS. These three conditions have to be guaranteed through the modelling flow. Fig. 4.7 details this modelling flow.

Data for modelling are generated using Saber. Stimuli are generated automatically. The basic idea for this automated data generation is available in [66]. Parameter variations are included by using either MC simulation or WCA. Then, data are processed to obtain the min-nominal-max form. The final step in data generation is splitting the data into training and test set.

Then the affine model is generated. ε̂SVR6, ε̂SVR4, ν̂SVR4, or ν̂SVR6 can be selected as training algorithm. The two original algorithms, εSVR and νSVR, are implemented for comparison. The optimisation problem is solved using CVX, a package for specifying and solving convex programmes, [29, 30] or an extended version of the SMO algorithm. The extended algorithm is described in Sec. 4.5. The CVX solver is very fast even though the complete kernel matrix has to be calculated. Its disadvantage is that it has a tendency to assign a very small non-zero value to each multiplier. This selects all data points as support vectors. The extended SMO algorithm is much slower, but selects only a subset of the data vectors as support vectors. For the CVX solver, an additional model reduction step is

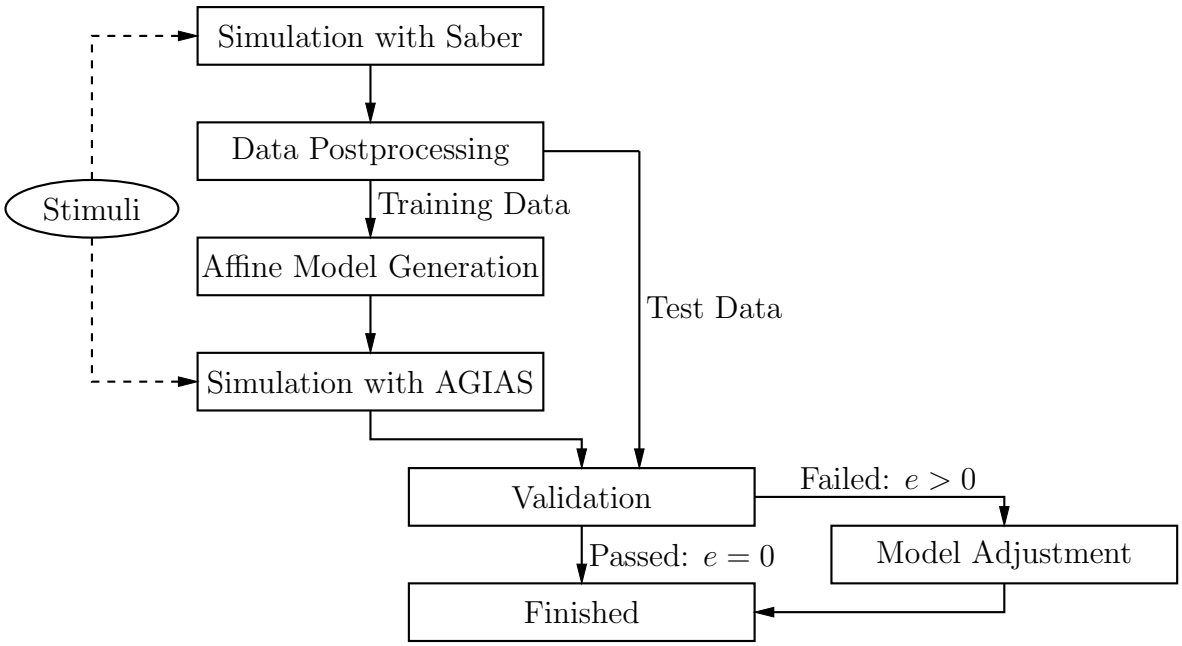


Figure 4.7.: Implemented flow for training SVMs with interval parameters.

included. This reduction step sets multipliers with values several dimensions smaller than the maximum multiplier value to zero.

The first validation step is simulating the model using AGIAS with the same stimuli used in the simulation with Saber. If a model cannot be simulated with AGIAS successfully, validation cannot be carried out. SVM algorithms cannot guarantee to find a model that encloses data as this information is not part of the optimisation problem. In the second validation step, the output estimates are compared with the complete original data set. This steps calculates the modelling error as measure for the model's quality. The modelling error e was introduced in Sec. 4.2.1 and is defined as

$$e = \max\{0, \overline{y}_i - \overline{f(x_i)}, \underline{f(x_i)} - \underline{y}_i, \}. \tag{4.93}$$

The modelling error e is positive, if the original maximum values are larger than the estimated maximum values or the original minimum value is smaller than the estimated minimum value respectively. If the modelling error is larger than zero, validation has failed and the model has to be adjusted.

Model adjustment is based on the same principle as external modelling. The model is adjusted by using the modelling error as distance for parallel translation. For SVM models, this results in the following model equation

$$\hat{f}(\mathbf{x}) = \hat{\mathbf{w}}\mathbf{x} + b_0 + \varepsilon_b(b_1 + e). \tag{4.94}$$

4.5. Extended SMO Algorithm for Training Affine SVMs

The new SVM algorithms cannot be solved using the standard SMO. First and foremost, because the SMO does not take the second equality constraint into account, and second, the new algorithms cannot be written in the vector form given in Sec. 3.1.8. This section introduces an extended vectorised form of the optimisation problem and an extended version of the SMO algorithm which is specifically tailored to train affine SVMs. The ε̂SVR is used to demonstrate the new algorithm. For ease of reading, the dual optimisation problem for the ε̂SVR is

$$\begin{aligned}
&W(\alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)}) = \\
&\qquad -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) \\
&\qquad\qquad (\alpha_j^* - \alpha_j + \beta_j^* - \beta_j + \gamma_j^* - \gamma_j)\mathbf{x}_i \cdot \mathbf{x}_j \\
\underset{\alpha_i^{(*)},\beta_i^{(*)},\gamma_i^{(*)}}{\text{maximise}} \quad &\qquad -\frac{1}{2}\sum_{i,j=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)(\beta_j^* - \beta_j - \gamma_j^* + \gamma_j)|\mathbf{x}_i \cdot \mathbf{x}_j| \\
&\qquad -\varepsilon(\alpha_i^* + \alpha_i + \beta_i^* + \beta_i + \gamma_i^* + \gamma_i) \\
&\qquad +\sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i + \sum_{i=1}^{l}(\beta_i^* - \beta_i)\overline{y}_i + \sum_{i=1}^{l}(\gamma_i^* - \gamma_i)\underline{y}_i
\end{aligned} \tag{4.95}$$

$$\text{subject to } \sum_{i=1}^{l}(\alpha_i^* - \alpha_i + \beta_i^* - \beta_i + \gamma_i^* - \gamma_i) = 0 \tag{4.96}$$

$$\sum_{i=1}^{l}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i) = 0 \tag{4.97}$$

$$0 \leq \alpha_i^{(*)}, \beta_i^{(*)}, \gamma_i^{(*)} \leq C \tag{4.98}$$

The vectors $\mathbf{a}$ and $\mathbf{q}$ are determined from Eq. (4.96). For the ε̂SVR these are

$$\mathbf{a} = [\alpha_i^*, \alpha_i, \beta_i^*, \beta_i, \gamma_i^*, \gamma_i]^T \tag{4.99}$$

$$\mathbf{q} = [1, -1, 1, -1, 1, -1]^T. \tag{4.100}$$

With these two vectors the target function can be rewritten:

$$\begin{aligned}
W(\mathbf{a}) = &-\frac{1}{2}\sum_{i,j=1}^{L}\mathbf{q}_i\mathbf{q}_j\mathbf{a}_i\mathbf{a}_j\mathbf{x}_i \cdot \mathbf{x}_j - \varepsilon\sum_{i=1}^{L}\mathbf{q}_i\mathbf{a}_i \\
&-\frac{1}{2}\sum_{i,j=1}^{L}(\beta_i^* - \beta_i - \gamma_i^* + \gamma_i)(\beta_j^* - \beta_j - \gamma_j^* + \gamma_j)|\mathbf{x}_i \cdot \mathbf{x}_j| \\
&-\varepsilon(\alpha_i^* + \alpha_i + \beta_i^* + \beta_i + \gamma_i^* + \gamma_i) \\
&+\sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i + \sum_{i=1}^{l}(\beta_i^* - \beta_i)\overline{y}_i + \sum_{i=1}^{l}(\gamma_i^* - \gamma_i)\underline{y}_i.
\end{aligned} \tag{4.101}$$

The last line of this equation can be reformulated by gathering the output values in one vector

$$\mathbf{L}_1 = [y_i, y_i, \overline{y}_i, \overline{y}_i, \underline{y}_i, \underline{y}_i]^T. \tag{4.102}$$

To reformulate the second line in terms of the vector $\mathbf{a}$, another vector is needed. This vector is constructed as

$$\mathbf{r} = [0, 0, 1, -1, -1, 1]^T \tag{4.103}$$

where the zeros mask the multipliers $\alpha_i^{(*)}$. Using these two vectors, the optimisation problem can be rewritten as

$$\begin{aligned} \underset{\mathbf{a}}{\text{maximise}} \quad & W(\mathbf{a}) = -\frac{1}{2}\sum_{i,j=1}^{L} \mathbf{q}_i\mathbf{q}_j k(\mathbf{x}_i, \mathbf{x}_j)\mathbf{a}_i\mathbf{a}_j \\ & \quad -\frac{1}{2}\sum_{i,j=1}^{L} \mathbf{r}_i\mathbf{r}_j k(\mathbf{x}_i, \mathbf{x}_j)\mathbf{a}_i\mathbf{a}_j + \sum_{i=1}^{L}\mathbf{L}_i\mathbf{q}_i\mathbf{a}_i \end{aligned} \tag{4.104}$$

$$\text{subject to} \quad \sum_{i=1}^{L} \mathbf{q}_i\mathbf{a}_i = 0 \tag{4.105}$$

$$\sum_{i=1}^{L} \mathbf{r}_i\mathbf{a}_i = 0 \tag{4.106}$$

$$0 \leq \mathbf{a}_i \leq Ci \tag{4.107}$$

where $\mathbf{L} = \mathbf{L}_1 - \varepsilon \cdot \mathbf{1}$ with $\mathbf{1}$ as the unit vector.

The original SMO chooses two multipliers for optimisation, one of which is substituted using the equality constraint of the optimisation problem. This approach can not solve optimisation problems with two equality constraints. As each equality constraint can be used to remove one variable from the optimisation problem, the optimisation problem can still be solved analytically when three multipliers are chosen.

The original algorithm employs a heuristic to select one multiplier which violates the KKT conditions and a second multiplier to maximize the increase in the objective function. Extending this to three multipliers requires new heuristics which fulfil these task and return triples for solving. Without loss of generality these three multipliers are denoted by the indexes 1, 2, and 3.

The second constraint does not apply to all multipliers in the algorithm. This introduces an additional restriction to the heuristics for choosing the multipliers and requires additional intelligence. If no valid triple can be chosen, the algorithm chooses a valid pair instead and optimises that pair.

Due to the formulation of the equality constraints not every triple can be optimised. The elements of the vector $\mathbf{q}$ can take the values $\{+1, -1\}$ and the elements of the vector $\mathbf{r}$ can take the values $\{+1, 0, -1\}$. There are different possibilities for a triple to be invalid for optimisation: two or three r_i $i \in \{1, 2, 3\}$ are 0 or the multiplier vectors $[q_1, q_2, q_3]$ and $[r_1, r_2, r_3]$ are linearly dependent.

To obtain a valid triple, the multipliers are sorted into different classes. The first two classes are obtained by collecting all multipliers in two classes C_{UP} and C_{LOW}. These two classes are already used in the original algorithm's heuristic [10, 61]. The second set of classes is based on the vector $\mathbf{q}$. The two resulting classes are called C_P and C_N for positive and negative q_i respectively. The third set of classes is based on the vector $\mathbf{r}$ and the relation between r_i and q_i. This set has three classes: C_{ZERO} for $r_i = 0$, C_{CIS} for $r_i = q_i$, and C_{TRANS} for $r_i \neq q_i$.

For different SVM algorithms, different combinations of these different classes are used for working set selection. Only the two classes C_{UP} and C_{LOW} are needed for the original εSVR algorithm, all twelve combinations of the different classes are needed to train the new ν̂SVR algorithms. Tab. 4.2 shows all class combinations for all algorithms which have been implemented. When all classes are evaluated, a set of possible candidates is chosen. From these candidates, the pair or triple which promises the largest increase of the objective function is chosen for optimisation.

Table 4.2.: Working set selection classes for original and affine SVM algorithms for regression.

Algorithm	Classes
εSVR	$\{C_{UP}, C_{LOW}\}$
νSVR	$\{C_{UP}, C_{LOW}\} \times \{C_P, C_N\}$
ε̂SVR4, ε̂SVR6	$\{C_{UP}, C_{LOW}\} \times \{C_{ZERO}, C_{CIS}, C_{TRANS}\}$
ν̂SVR4, ν̂SVR6	$\{C_{UP}, C_{LOW}\} \times \{C_P, C_N\} \times \{C_{ZERO}, C_{CIS}, C_{TRANS}\}$

For pairs, the analytic solution is the same as for the original SMO. For a valid set of three multipliers, the analytic solution is presented here. In this case the equality constraints can be written as

$$q_1 a_1 + q_2 a_2 + q_3 a_3 = -\sum_{i=4}^{L} q_4 a_4 = G_1 \tag{4.108}$$

$$r_1 a_1 + r_2 a_2 + r_3 a_3 = -\sum_{i=4}^{L} r_4 a_4 = G_2 \tag{4.109}$$

where G_1 and G_2 are constants. All variables are limited to values in the interval $[0, C]$. Solving Eq. (4.109) for a_3 and substituting into Eq. (4.108) returns a linear equation in a_1 and a_2

$$\underbrace{(q_1 - q_3 r_1 r_3)}_{m_1} a_1 + \underbrace{(q_2 - q_3 r_2 r_3)}_{m_2} a_2 = G_1 - q_3 r_3 G_2. \tag{4.110}$$

The optimisation problem has to be solved along this line. The last equation is solved for

a_2 and substituted into the target function along with Eq. (4.109).

$$\begin{aligned} W(a_1) = &-\frac{1}{2}[(K_{11} + r_1^2 r_3^3 K_{33} - 2r_1 r_3 K_{13})a_1^2 \\ &+ (K_{22} + r_2^2 r_3^2 K_{33} - 2r_2 r_3 K_{23}) \left(\frac{G_1 - q_3 G_2 - m_1 a_1}{m_2}\right)^2 \\ &+ 2(K_{12} - r_2 r_3 K_{13} - r_1 r_3 K_{23} + r_1 r_2 r_3^2 K_{33}) \frac{G_1 - q_3 G_2 - m_1 a_1}{m_2} a_1 \\ &+ 2(K_{13} G_2 - r_1 r_3 K_{33} G_2 + \sum_{i=4}^{l} (K_{1i} - r_1 r_3 K_{3i}) a_i) a_1 \\ &+ 2(K_{23} G_2 - r_1 r_3 K_{33} G_2 + \sum_{i=4}^{l} (K_{2i} - r_2 r_3 K_{3i}) a_i) \frac{G_1 - q_3 G_2 - m_1 a_1}{m_2}] \\ &+ (q_1 L_1 - q_3 r_1 r_3 L_3) a_1 + (q_2 L_2 - q_3 r_2 r_3 L_3) \frac{G_1 - G_2 q_3 r_3 - m_1 a_1}{m_2} + const \end{aligned} \tag{4.111}$$

The second derivative of this formula is needed to determine whether the maximum exists.

$$\begin{aligned} \frac{\partial^2 W}{\partial a_1^2} = &-(K_{11} + r_1^2 r_3^2 K_{33} - 2r_1 r_3 K_{13}) - \frac{m_1^2}{m_2^2}(K_{22} + r_2^2 r_3^2 K_{33} - 2r_2 r_3 K_{23}) \\ &+ 2\frac{m_1}{m_2}(K_{12} - r_2 r_3 K_{13} - r_1 r_3 K_{23} + r_1 r_2 r_3^2 K_{33}) = D \end{aligned} \tag{4.112}$$

As all entries in the kernel matrix $\mathbf{K}$ are strictly positive and $K_{ii} + K_{jj} > 2K_{ij}$ for $i \neq j$ [64], the second derivative of the target function with respect to a_1 is always smaller than zero and a_1 determined from the target function maximises the target function.

The new a_1 is determined by setting the first derivative of W to zero and solving for a_1.

$$\begin{aligned} a_1 = &\frac{1}{D}[\frac{m_1}{m_2}(G_1 - q_3 G_2)(K_{22} - r_2^2 r_3^2 - 2r_2 r_3 K_{23}) \\ &- \frac{1}{m_2}(G_1 - q_3 G_2)(K_{12} - r_2 r_3 K_{13} - r_1 r_3 K_{23} + r_1 r_2 r_3^2 K_{33}) \\ &+ \frac{m_1}{m_2}(K_{23} G_2 - r_1 r_3 K_{33} G_2 + \sum_{i=4}^{l} (K_{2i} - r_2 r_3 K_{3i}) a_i) \\ &- K_{13} G_2 + r_1 r_3 K_{33} G_2 - \sum_{i=4}^{l} (K_{1i} - r_1 r_3 K_{3i}) a_i) + q_1 L_1 - q_3 r_1 r_3 L_3 \\ &- \frac{m_1}{m_2}(q_2 L_2 - q_3 r_2 r_3 L_3)] \end{aligned} \tag{4.113}$$

All valid solutions have to lie in the interval $[0, C]$. After a_1 has been calculated, it is bounded to that interval, if necessary. Then a_2 is calculated and bounded, and finally a_3 is calculated and bounded, too. Then a_1 is recalculated and bounded based on a_2 and a_3 to make sure all three values are inside the valid interval.

5. Results

The capabilities of the new algorithms are presented by modelling the characteristic function of three example circuits. These example circuits are: an inverting amplifier, a class A amplifier and a single diode. These examples have been chosen to evaluate the new algorithms on data sets with different degrees on non-linearity: first is the inverting amplifier, which operates linearly over the largest part of its operation range, second is the class A amplifier with a smaller linear operation region, and third is the single diode, which is strongly non-linear. AGIAS can simulate all chosen examples only for parts of the chosen modelling range. The inverting amplifier can only be simulated for certain operating points. Therefore, data are generated using Saber and AGIAS is used to verify models at the end of the modelling process.

The example circuits showcase different SVMs and kernel functions. For each example, models are created with all available SVM algorithms. The original εSVM and νSVM algorithms serve as examples for external variation-aware modelling (Sec. 4.2.1), while the new $\hat{\varepsilon}$SVR and $\hat{\nu}$SVR algorithms are examples for internal variation-aware modelling (Sec. 4.2.2). Three kernel functions are available for all SVMs: linear kernel, Gaussian RBF kernel, and polynomial kernel (Sec. 3.1.6).

While every algorithm is suited for modelling every circuit, this does not hold for kernel functions. This is obvious for the linear kernel: if the circuit's characteristic curve is strongly non-linear, it cannot be modelled with a linear function with sufficient accuracy. It is not that obvious for non-linear kernels. The kernel functions and the their kernel function parameters have been determined experimentally for the examples presented here.

Data for modelling was generated with MC simulations using Saber. To save time and to have the same data set for all models of a given circuit data was generated once per circuit.

Models were evaluated for their size, modelling errors, overestimation and the areal overestimation. The size of a model is given as the number of SVs in the model. The modelling error is used to adjust models to fully enclose the original data. Overestimation is a measure to evaluate the average width of a model compared to the width of the original data and areal overestimation compares the area described by the model compared to the size of the original data set. Models are good, if they are sufficiently small, and the overestimation and areal overestimation are close to 1.

The extended SMO which was created to train affine SVMs is evaluated with respect to its runtime and selectivity.

5.1. Quality Measures for Affine Support Vector Machines

SVM models are trained with selective algorithms. These training algorithms can result in sparse models, e.g. models with a very low number of SVs compared to the number of data points in the training data set. In this work, models do not have to be sparse, they have to be sufficiently small. A model qualifies as sufficiently small under two conditions: first, the number of SVs is small compared to the number of elements in the training data set or second, it can be simulated with AGIAS.

The modelling error as defined in Sec. 4.2.1

$$e = \max\{0, \overline{y}_i - \overline{f(\mathbf{x}_i)}, \underline{f(\mathbf{x}_i)} - \underline{y}_i\}. \tag{5.1}$$

e cannot measure the quality of the final model. The modelling error is calculated during validation and is used to adjust the model. After validation has finished the modelling error equals zero on all models. Generally, models with a large modelling error before model adjustment have a lower overall accuracy. In this respect the modelling error can be used as a first estimate.

Overestimation was introduced by O. Scharf in [63]. AGIAS is designed to overestimate intervals during simulation. This guarantees that the real values are always enclosed in the simulation results. The overestimation is defined as the average ratio of width of the affine output values to the width of the original data. The overestimation is calculated as

$$m_{ov} = \frac{1}{l} \sum_{i=1}^{l} \frac{d_{i,\text{Model}}}{d_{i,\text{Org}}} \tag{5.2}$$

with $d_{i,\text{Model}}$ as the width of the AGIAS output and $d_{i,\text{Org}}$ as the width of the original data. The width of an affine form equals two times its radius. As data is given in the min-nominal-max form, the width of the original data is calculated as $d_{i,\text{Org}} = \overline{y}_i - \underline{y}_i$. Substituting these into Eq. (5.2), the overestimation is calculated as

$$m_{ov} = \frac{1}{l} \sum_{i=1}^{l} \frac{2\text{rad}(\hat{f}(\mathbf{x}_i))}{\overline{y}_i - \underline{y}_i}. \tag{5.3}$$

The overestimation is defined using the ratio of the widths of two intervals. If $d_{i,\text{Org}}$ equals 0 for one data point in the set, the resulting overestimation approaches infinity. As this is the case in all examples presented here, the median is used in Eq. 5.3 instead. Interpreting the overestimation stays difficult in models which have minute original widths for a large range of their input data.

The areal overestimation is defined as the ratio of the area covered by the model to the area covered by the original data. It is defined as

$$A_{ov} = \frac{A_{\text{Model}}}{A_{\text{Org}}} \tag{5.4}$$

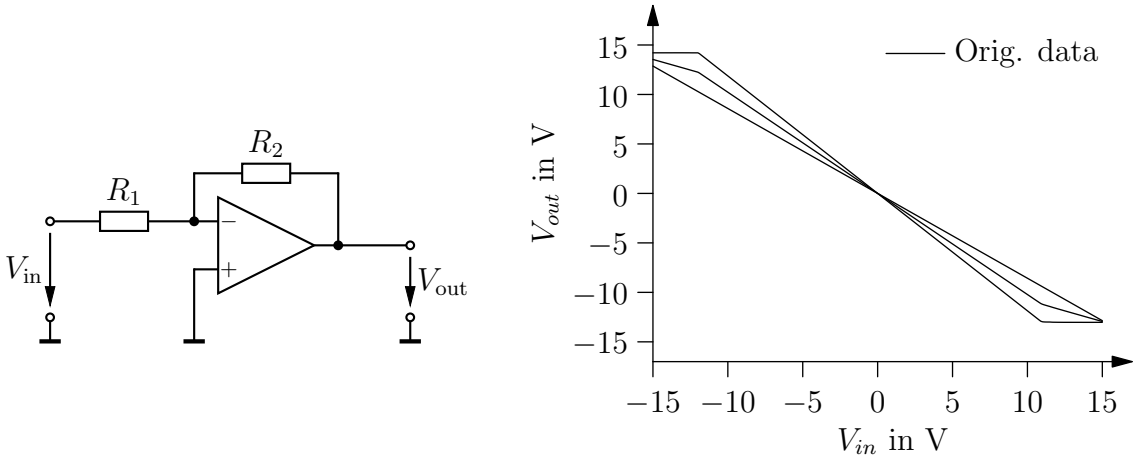
where A_{Model} is the area covered by the model and A_{Org} is the area covered by the original data. These two areas are calculated using the trapeze formula on its in- and output data pairs

$$A = \sum_{i=2}^{N} \frac{1}{2}(d_{y,i} + d_{y,i-1})(x_i - x_{i-1}) \tag{5.5}$$

where $d_{y,i}$ denotes the width of a given data interval.

5.2. Inverting Amplifier

The inverting amplifier consists of an operational amplifier and two resistors. The circuit diagram is given in Fig. 5.1a. In this circuit, the operational amplifier is a well-known µa741 which consists of 25 BJTs. The two top level resistors R_1 and R_2 are of the same size, which results in an amplification of 1. The op-amp is supplied with ±15 V. The inverting amplifier is simulated for inputs which are uniformly distributed between −15 V and 15 V. The model recreates the characteristic function $V_{out} = f(V_{in})$ as given in Fig. 5.1b. The circuit operates linearly over the largest part of its operating range (roughly between −11 V and 11 V). Although it is the most complex example in terms of the number of components and the circuitry involved, its data set has the simplest shape.



(a) Schematic diagram.

(b) Data for modelling.

Figure 5.1.: Inverting amplifier.

Data for modelling were obtained by running a MC simulation in Saber. For this simulation, the transistor gains and the top level resistances are varied by 10 %. Data were generated using the DC sweep analysis: the input voltage V_{in} is swept from −15 V to 15 V in steps of 0.01 V. 10000 MC runs were performed per input voltage step. The simulation returns 3001 data samples. 301 of these samples are used for training. Data

samples for training are chosen based on the gradient between data samples. Fig. 5.1b shows the characteristic curve of the inverting amplifier as well as the envelope of the MC results.

This circuit was modelled successfully using linear and Gaussian RBF kernels. The SVM cost hyperparameter C is set to 5. εSVRs use the maximum permissible error as hyperparameter. It is is varied between 0.00 V and 1.50 V in steps of 0.15 V. The trade-off parameter ν is varied between 0.1 and 1.0 in steps of 0.1 for νSVRs. Data is not scaled for training.

The linear kernel has no hyperparameters. The Gaussian RBF kernel has one hyperparameter: $\frac{1}{2\sigma^2}$. The LibSVM documentation [10] recommends using one over the number of data points as default. This gives good results in training nominal SVMs. For affine SVMs, bigger values give better results. As a rule of thumb $\frac{3}{N}$ is a good starting point. In this section $\frac{1}{120}$ was used for ε̂SVR models and $\frac{1}{70}$ was used for ν̂SVR models.

All in all, 264 models were created from the different algorithms and configuration options. An overview of quality measures and runtimes for all models is available in Appendix A.1.

For this section, inverting amplifier models with the following ε and ν parameters are presented: $\varepsilon = \{0.30\,\mathrm{V}, 1.05\,\mathrm{V}\}$ for both kernels and $\nu = \{0.2, 0.7\}$ for linear kernels and $\nu = \{0.1, 0.7\}$ for RBF kernels. For νSVMs with RBF kernels a different set was chosen as the difference for models with $\nu = 0.2$ and $\nu = 0.7$ is too small to illustrate.

SVM models with linear kernels have only two parameters, no matter how many SVs were selected during training as the model parameters w and b can be calculated directly from the SVs and their weights. The numbers of SVs in linear models are available from Tab. 5.1a for completeness. In SVMs with any other kernel, the number of SVs determines the number of non-linear elements in the model. Tab. 5.1b shows the SVs for the selected models with RBF kernels.

The CVX solver usually returns all data points as support vectors. Before calculating the number of SVs, the CVX models were reduced, by setting weights to zero that were more than three orders of magnitude smaller than the maximum weight. Models trained with the extended SMO profit from its selecting capabilities and are not reduced further. Models trained with the extended SMO are smaller than models trained wit the CVX.

The numbers show, that less SVs are used in models with larger permissible errors for εSVRs and models with lesser emphasis on the maximum permissible error in the cost function for νSVRs. This result is expected as the constraints for a successful training are more relaxed. Models that place more importance on the maximum permissible error through a higher trade-off parameter have more SVs.

Table 5.1.: Overview of the numbers of SVs for SVM models with linear and Gaussian RBF kernels generated with CVX and the extended SMO. The training set contains 301 samples.

(a) Linear kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	301	156	189	111	129	120
εSVR $\varepsilon = 1.05\,\mathrm{V}$	2	65	301	5	64	46
νSVR $\nu = 0.2$	301	301	301	139	280	244
νSVR $\nu = 0.7$	301	301	301	139	300	301

(b) RBF kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	38	118	116	27	25	30
εSVR $\varepsilon = 1.05\,\mathrm{V}$	6	31	79	6	7	4
νSVR $\nu = 0.2$	301	228	225	257	130	223
νSVR $\nu = 0.7$	301	301	301	257	219	298

Tab. 5.2 shows the modelling error for the selected inverting amplifier models. The modelling error is the remaining error between model and original data where the model does not fully enclose the original data. RBF kernels enclose the data better than linear kernels as the RBF models the regions of non-linear operation more closely than the linear kernel.

ϑSVRs4 with $\nu = 0.7$ and both selected ϑSVR6 models trained with the CVX solver, and ϑSVR6 with $\nu = 0.7$ trained with the extended SMO solver lead to models with a modelling error of $e = 0.0\,\mathrm{V}$. These models enclose the original data without further adjustment. All other models are adjusted before overestimation and areal overestimation are calculated.

Affine SVMs have a much lower modelling error than their nominal counterparts. Models with higher maximum permissible error ($\varepsilon = 1.05\,\mathrm{V}$, $\nu = 0.2$) have lower modelling errors than models with more restrictive settings.

Table 5.2.: Modelling errors for SVM models with linear and Gaussian RBF kernels.

(a) Linear kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	2.085	0.651	0.619	2.087	0.439	1.118
εSVR $\varepsilon = 1.05\,\mathrm{V}$	1.952	1.271	0.768	1.908	1.193	1.537
νSVR $\nu = 0.2$	2.537	0.441	0.615	2.824	0.335	1.541
νSVR $\nu = 0.7$	2.537	0.092	0.087	2.824	1.254	2.614

(b) RBF kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	2.082	0.074	0.074	1.919	0.372	0.440
εSVR $\varepsilon = 1.05\,\mathrm{V}$	1.816	0.534	0.501	1.603	1.262	1.258
νSVR $\nu = 0.2$	2.383	0.083	0.000	2.381	3.220	0.276
νSVR $\nu = 0.7$	2.383	0.000	0.000	2.381	0.046	0.000

Tab. 5.3 compares the overestimation for selected models. All models in this table have been adjusted to a modelling error of 0.0 V. Internal variation-aware modelling with affine SVMs leads to models with a lower overestimation than external variation-aware modelling. Models with larger maximum permissible errors have a higher overestimation. Reason for this is the larger width of the model even for small original widths. Models with RBF kernels perform better than models with linear kernels as the non-linear models fit the original data more closely.

$\hat{\varepsilon}$SVR6 and $\hat{\nu}$SVR6 models do not generally outperform $\hat{\varepsilon}$SVR4 and $\hat{\nu}$SVR4 models in terms of overestimation. On the given model selection, a larger permissible error leads to larger overestimates while a trade off putting stronger emphasis on the permissible error leads to smaller overestimates.

Table 5.3.: Overestimation for adjusted SVM models with linear and Gaussian RBF kernels.

(a) Linear kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	5.928	3.366	3.443	6.078	2.914	1.354
εSVR $\varepsilon = 1.05\,\mathrm{V}$	7.656	5.925	6.262	7.545	5.767	7.078
νSVR $\nu = 0.2$	6.470	2.412	2.784	6.471	2.135	1.102
νSVR $\nu = 0.7$	6.470	1.723	2.379	6.471	3.004	6.344

(b) RBF kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	6.075	2.264	2.264	5.660	2.993	1.583
εSVR $\varepsilon = 1.05\,\mathrm{V}$	7.309	5.517	5.500	6.414	7.995	4.753
νSVR $\nu = 0.2$	6.078	2.369	2.379	6.073	6.095	1.609
νSVR $\nu = 0.7$	6.078	1.637	2.779	6.073	2.126	2.421

Areal overestimation for the selected models is compared in Tab. 5.4. The areal overestimation is smaller than the point-wise overestimation discussed before.

The external variation-aware models overestimate the data area by a larger margin than internal variation-aware models as parallel translation does not handle the shape of the data well.

Table 5.4.: Areal overestimation for adjusted SVM models with linear and Gaussian RBF kernels.

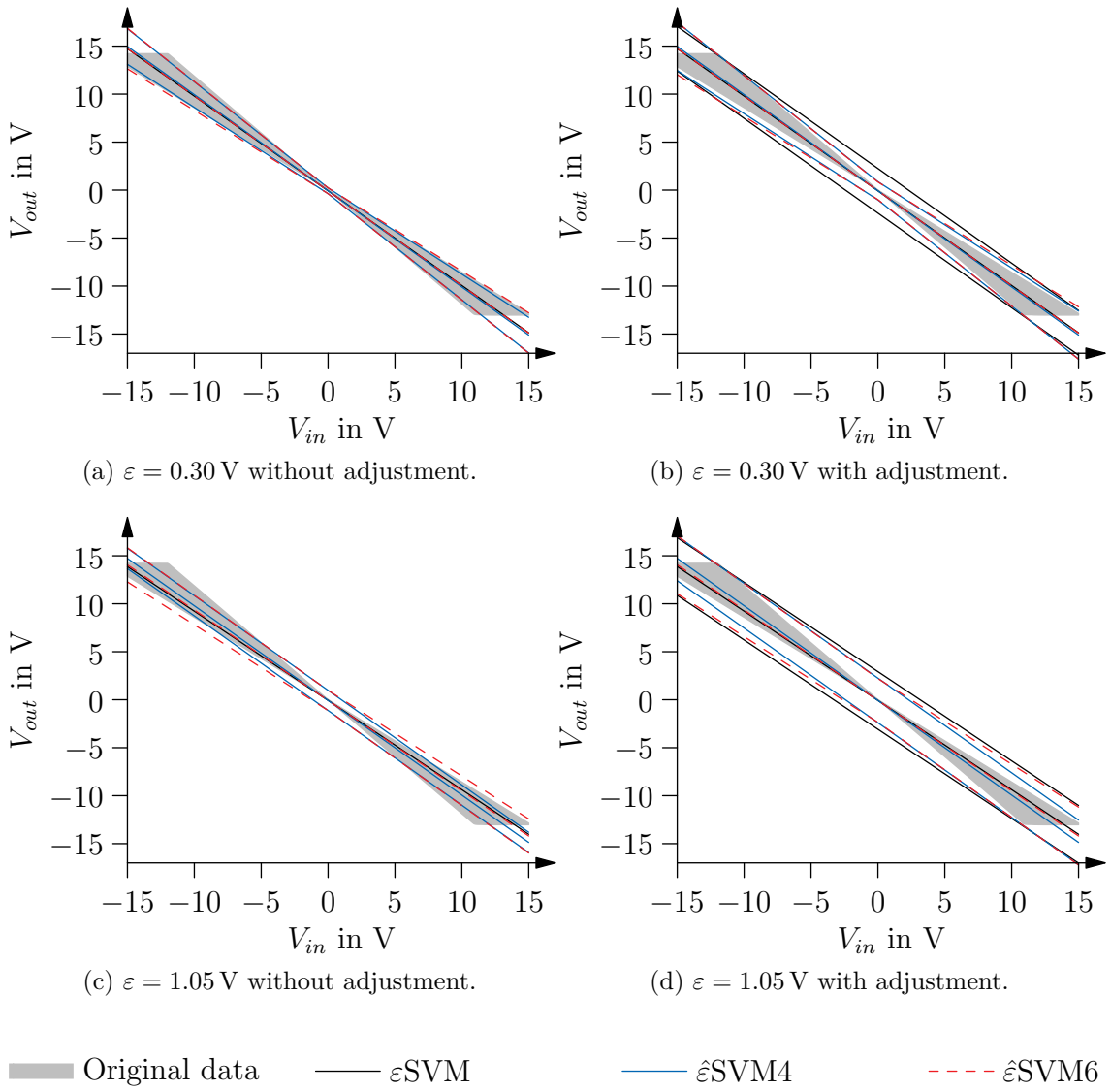
(a) Linear kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	2.36	1.75	1.85	6.66	1.69	2.53
εSVR $\varepsilon = 1.05\,\mathrm{V}$	3.05	2.36	2.69	6.86	4.74	6.15
νSVR $\nu = 0.2$	2.58	1.35	1.64	2.58	1.44	0.81
νSVR $\nu = 0.7$	2.58	1.34	1.34	2.58	1.65	2.57

(b) RBF kernel.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.30\,\mathrm{V}$	2.42	1.62	1.46	6.14	2.48	1.60
εSVR $\varepsilon = 1.05\,\mathrm{V}$	2.91	2.21	2.20	5.68	5.61	3.01
νSVR $\nu = 0.2$	2.42	1.60	1.49	7.24	8.32	1.73
νSVR $\nu = 0.7$	2.42	1.66	1.65	7.24	1.51	1.51

The SVR models with linear kernels are shown in Fig. 5.2 and Fig. 5.3 and for Gaussian RBF kernels in Fig. 5.4 and Fig. 5.5. Each figure presents models without adjustment on the left and adjusted models on the right. All models had to be adjusted after training as the affine SVMs did not enclose the data originally. This is normal behaviour for the algorithms as the optimization problem does not include a condition to force enclosure of the data. Adjusted models have been enhanced to enclose the original data. Fig. 5.5 shows models that do not need to be adjusted: ŏSVR models with $\nu = 0.7$ and the RBF kernel have a modelling error of 0.0 V. Even the linear ŏSVR models with the same trade-off parameter seem to enclose the data, although these models have modelling errors of 0.092 V and 0.087 V.



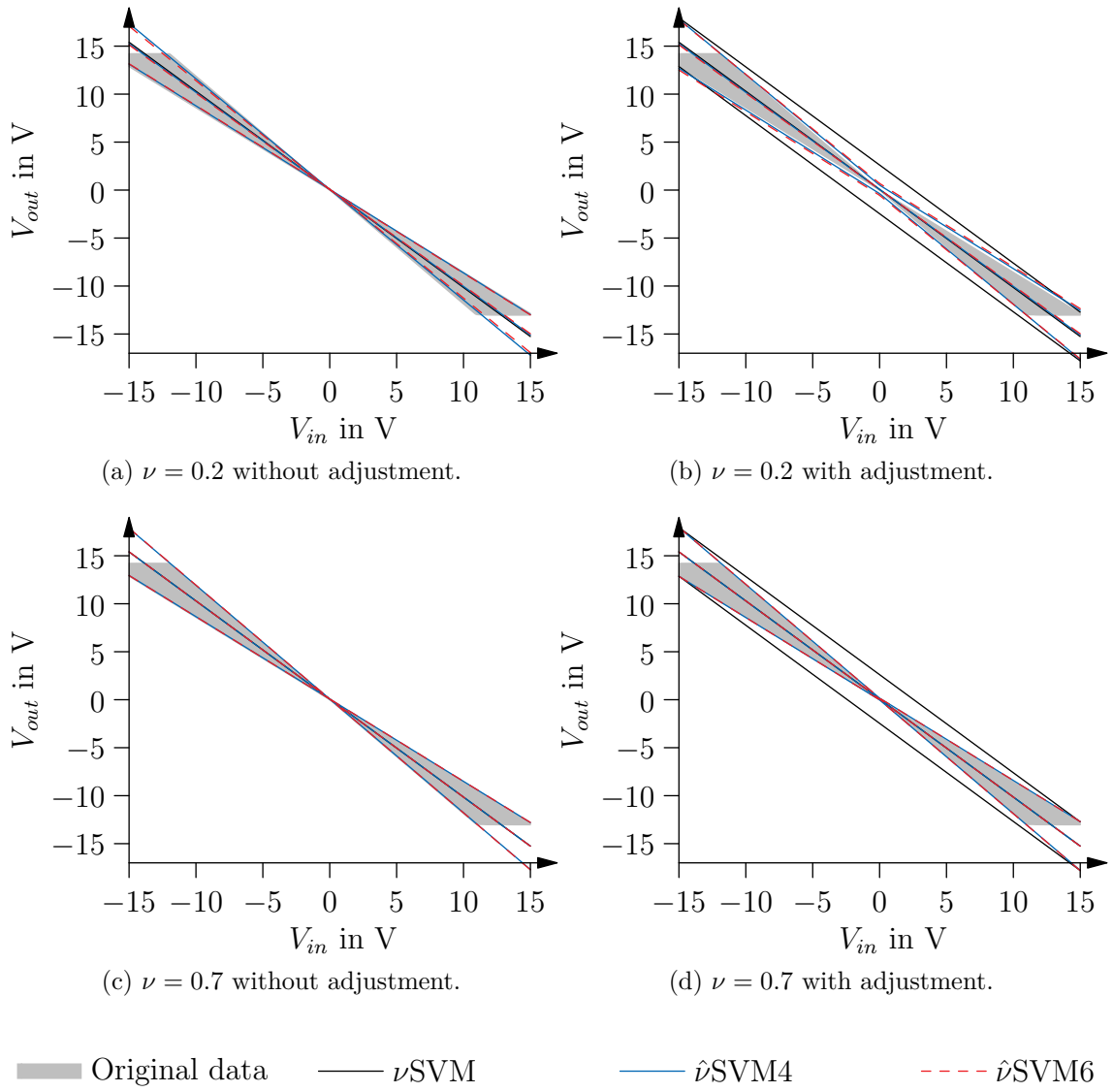
(a) $\varepsilon = 0.30\,\mathrm{V}$ without adjustment.

(b) $\varepsilon = 0.30\,\mathrm{V}$ with adjustment.

(c) $\varepsilon = 1.05\,\mathrm{V}$ without adjustment.

(d) $\varepsilon = 1.05\,\mathrm{V}$ with adjustment.

Figure 5.2.: Characteristic functions of εSVM models with linear kernels. Models are plotted without and with adjustment.



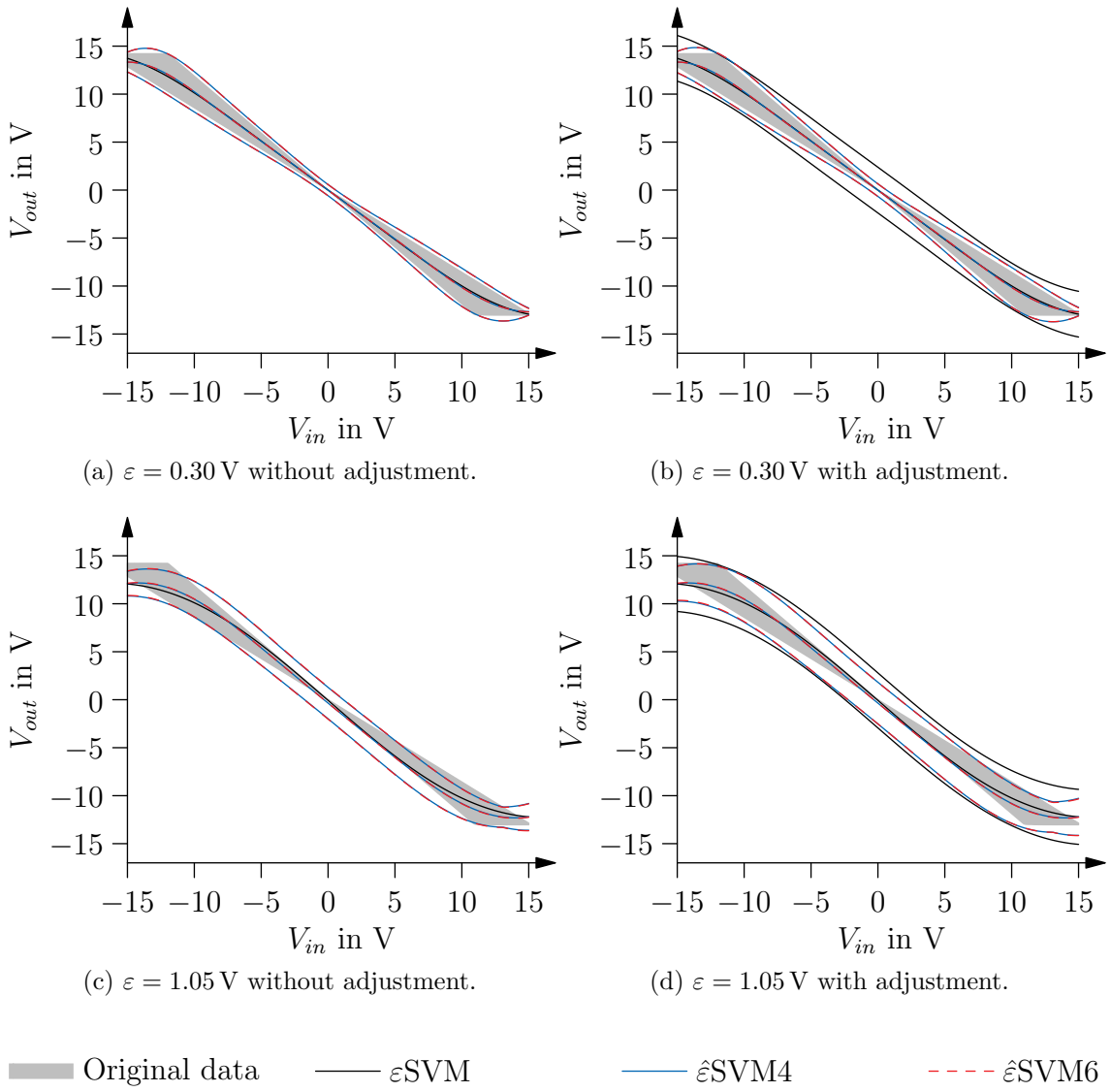
(a) $\nu = 0.2$ without adjustment.

(b) $\nu = 0.2$ with adjustment.

(c) $\nu = 0.7$ without adjustment.

(d) $\nu = 0.7$ with adjustment.

Figure 5.3.: Characteristic functions of νSVM models with linear kernels. Models are plotted without and with adjustment.



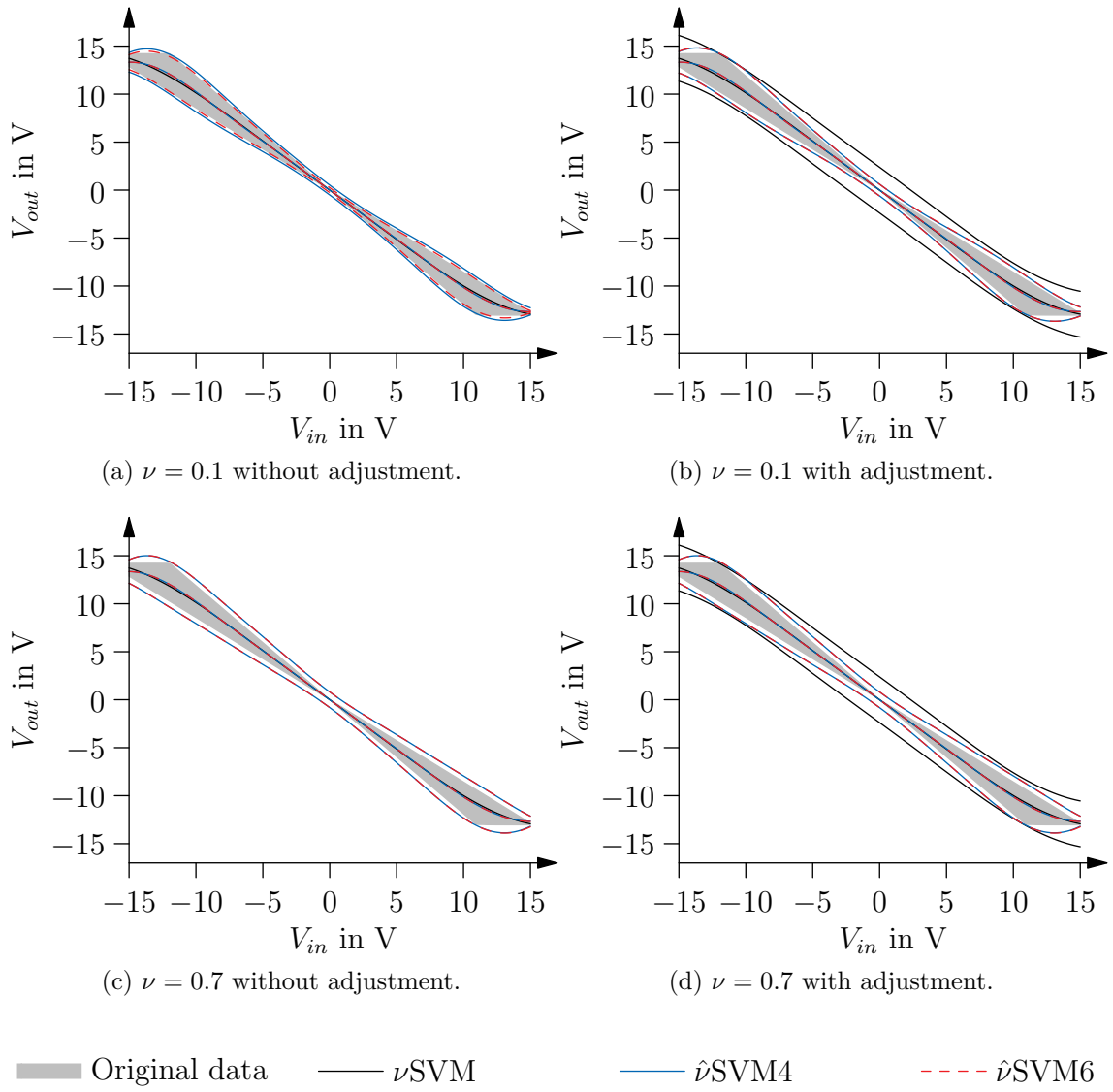
(a) $\varepsilon = 0.30\,\mathrm{V}$ without adjustment.

(b) $\varepsilon = 0.30\,\mathrm{V}$ with adjustment.

(c) $\varepsilon = 1.05\,\mathrm{V}$ without adjustment.

(d) $\varepsilon = 1.05\,\mathrm{V}$ with adjustment.

Figure 5.4.: Characteristic functions of εSVM models with RBF kernels. Models are plotted without and with adjustment.



(a) $\nu = 0.1$ without adjustment.

(b) $\nu = 0.1$ with adjustment.

(c) $\nu = 0.7$ without adjustment.

(d) $\nu = 0.7$ with adjustment.

Figure 5.5.: Characteristic functions of νSVM models with RBF kernels. Models are plotted without and with adjustment.

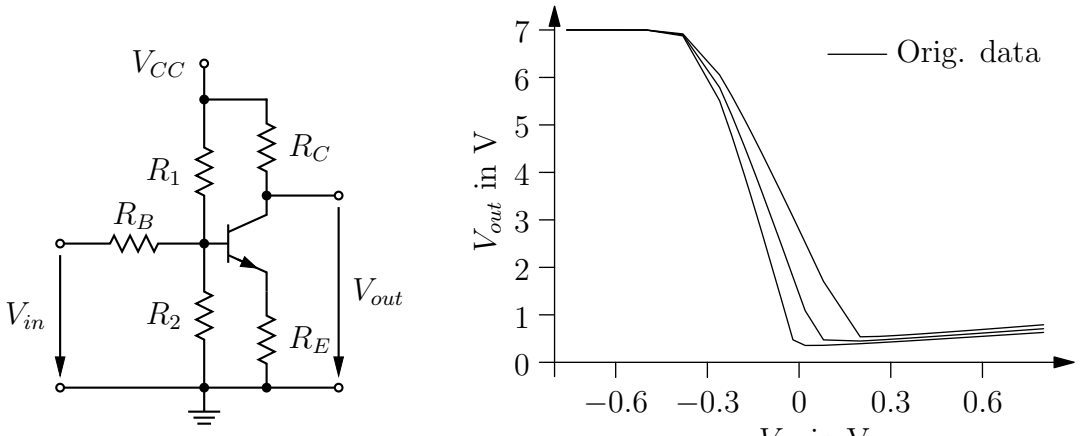
Inverting amplifiers are designed to operate linearly over the largest portion of the input values. Variations are strictly symmetric to the nominal values in the linear operation region. Only in the saturation regions, the circuits behaviour is non-linear and output variations are not symmetric to the nominal values. For this reason, linear $\hat{\varepsilon}$SVR4 and $\hat{\nu}$SVR4 models are expected to outperform non-linear models and $\hat{\varepsilon}$SVR6 and $\hat{\nu}$SVR6 models. Against expectations, $\hat{\varepsilon}$SVR4 and $\hat{\nu}$SVR4 models did not yield significantly better models than $\hat{\varepsilon}$SVR6 and $\hat{\nu}$SVR6 models. The basic idea here is that nominal values do not carry much additional information in a predominantly symmetrical setup. The results given here show that affine SVM4 and affine SVM6 algorithms select similar numbers of SVs in the model.

The model selection in this section shows different configurations that all lead to good models. The models come with relatively low modelling errors and the models fit the data closely even before adjusting the models. Models show small areal overestimation and relatively small overestimation.

5.3. Class A Amplifier

The class A amplifier was introduced in Sec. 1.2 as a rather simple circuit that cannot be simulated by AGIAS. The circuit diagram is given in Fig. 5.6a: the circuit consists of one BJT transistor and five resistors. The circuit is supplied with $V_{cc} = 7\,\mathrm{V}$. The resistors R_1 and R_2 bias the circuit at 0.9 V. The input voltage V_{in} was varied between $-0.8\,\mathrm{V}$ and 0.8 V.

This range includes the amplifying range for input voltages between approximately $-0.3\,\mathrm{V}$ and 0.1 V. The rest of the input voltage range includes a saturation region at V_{CC} for input voltages below $-0.3\,\mathrm{V}$ and an approximately linear operation range for input voltages above 0.1 V with an amplification of approximately 1. The input range for modelling was chosen deliberately to include more than just the amplification region. This leads to a data set which includes more non-linear behaviour than the data of the inverting amplifier in Sec. 5.2.



(a) Schematic diagramme. (b) Data for modelling.

Figure 5.6.: Class A amplifier.

For modelling, the cross-sectional area factor A of the BJT, and the resistance of the collector resistor R_C are varied. The cross-sectional area factor is varied by 20 %, the collector resistance is varied by 10 %.

Saber's MC simulation and a DC sweep were used to obtain data for modelling. The input voltage was swept from $-0.8\,\mathrm{V}$ to 0.8 V in 242 steps. 1000 MC samples were drawn for each input voltage step. This number was chosen as only two parameters in the circuit were varied. The simulation setup returns 243 data points. 31 of those are chosen as training data set. Fig. 5.6b shows the outline of the data area and the nominal values based on the training data set.

Each region in this data set has a different data distribution. In the saturation region, almost no variations exist. In the amplification region, the variations are not symmetric to the nominal values. Variations start at 0 and grow to around 2.5 V. In the linear operation

region, the variations form a symmetric almost constant-width tube around the nominal values.

This circuit is modelled using Gaussian RBF kernels. The cost hyperparameter C is set to 5. The maximum permissible error for εSVRs is varied between 0.0 V and 0.35 V in steps of 0.035 V. The trade-off parameter for νSVRs is varied between 0.1 and 1.0 in steps of 0.1. Data is not scaled for training. The kernel hyperparameter $\frac{1}{2\sigma^2}$ is set to 6.

For the class A amplifier, 126 models have been created. All model configurations were run with the CVX and extended SMO. An overview of the quality measures and run times for all models is available from Appendix A.2.

For a more detailed overview, class A amplifier models with $\varepsilon = \{0.070\,\mathrm{V}, 0.280\,\mathrm{V}\}$ and $\nu = \{0.2, 0.7\}$ are selected.

Tab. 5.5 compares the numbers of SVs for the selected class A amplifier models. Models trained with the CVX have been reduced before determining the number of SVs.

With just 31 samples in the training data set training algorithms were still selective. Only three ν̂SVR models with $\nu = 0.7$ use all data samples as SVs. Most models require less than half of the data points as SVs. This reduces the overall number of non-linear terms, that have to be evaluated by the simulator. Models with relaxed conditions on the modelling error — high maximum permissible error in εSVR and low trade-off parameter in νSVR models — have less SVs.

Table 5.5.: Class A Amplifier: Overview of the number of SVs for SVM models with Gaussian RBF kernels generated with CVX and the extended SMO. The training data set contains 31 samples.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.070\,\mathrm{V}$	15	20	21	16	23	22
εSVR $\varepsilon = 0.280\,\mathrm{V}$	6	10	9	6	8	11
νSVR $\nu = 0.2$	5	23	27	6	17	22
νSVR $\nu = 0.7$	5	31	31	6	26	31

Tab. 5.6 shows the modelling errors for the selected class A amplifier models. The maximum error in all models occurs around $V_{in} = 0.0\,\mathrm{V}$. This area has the largest width in the original data and the shape of the original data has a very sharp bend. This bend can not be represented as sharply in the models.

The modelling error is smaller in all models for larger permissible errors in εSVR models and νSVR models which give the maximum permissible error a greater weight in the cost function.

Table 5.6.: Modelling error for selected class A amplifier models in volts.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.070\,\mathrm{V}$	1.57	0.81	0.65	1.46	0.86	0.54
εSVR $\varepsilon = 0.280\,\mathrm{V}$	1.21	0.52	0.35	1.17	0.41	0.45
νSVR $\nu = 0.2$	1.22	0.92	0.78	1.08	1.96	0.53
νSVR $\nu = 0.7$	1.22	0.93	0.79	1.08	0.88	0.72

All models display relatively large overestimation. A summary is given in Tab. 5.7. With modelling errors as large as 20 % of the output range, models overestimate the regions above 0.1 V and below −0.3 V by a large margin. Overestimation for input values below 0.1 V grows to infinity as the width of the original data is 0. By using the median, the influence of these data samples on the overestimation is lowered.

Models trained with the CVX solver have lower overestimation than models trained with the extended SMO solver.

Table 5.7.: Overestimation for selected class A amplifier models.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.070\,\mathrm{V}$	3.73	3.70	3.30	5.34	4.10	3.01
εSVR $\varepsilon = 0.280\,\mathrm{V}$	3.65	3.48	3.15	5.14	3.19	3.29
νSVR $\nu = 0.2$	3.85	3.70	3.35	5.09	6.51	3.22
νSVR $\nu = 0.7$	3.85	3.71	3.32	5.09	3.70	3.36

Areal overestimation is given in Tab. 5.8. It indicated a larger overestimation of the data than the overestimation. The impact of regions for which the overestimation cannot be calculated directly is higher in this measure. Zero sized areas in the original data do not add to the area of the original data, while the respective areas of the model increase the model area significantly.

Table 5.8.: Areal overestimation for selected class A amplifier models.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.070\,\mathrm{V}$	6.30	4.54	3.95	7.61	4.97	3.54
εSVR $\varepsilon = 0.280\,\mathrm{V}$	6.16	4.48	3.58	7.26	3.88	4.06
νSVR $\nu = 0.2$	6.51	4.78	4.29	7.16	9.64	3.90
νSVR $\nu = 0.7$	6.51	4.84	4.32	7.16	4.74	4.15

Fig. 5.7 shows results for εSVR models of the class A amplifier, Fig. 5.8 shows the results for νSVR models of the class A amplifier. Models on the left have not been adjusted for enclosing all data, models on the right have been adjusted for full enclosure.

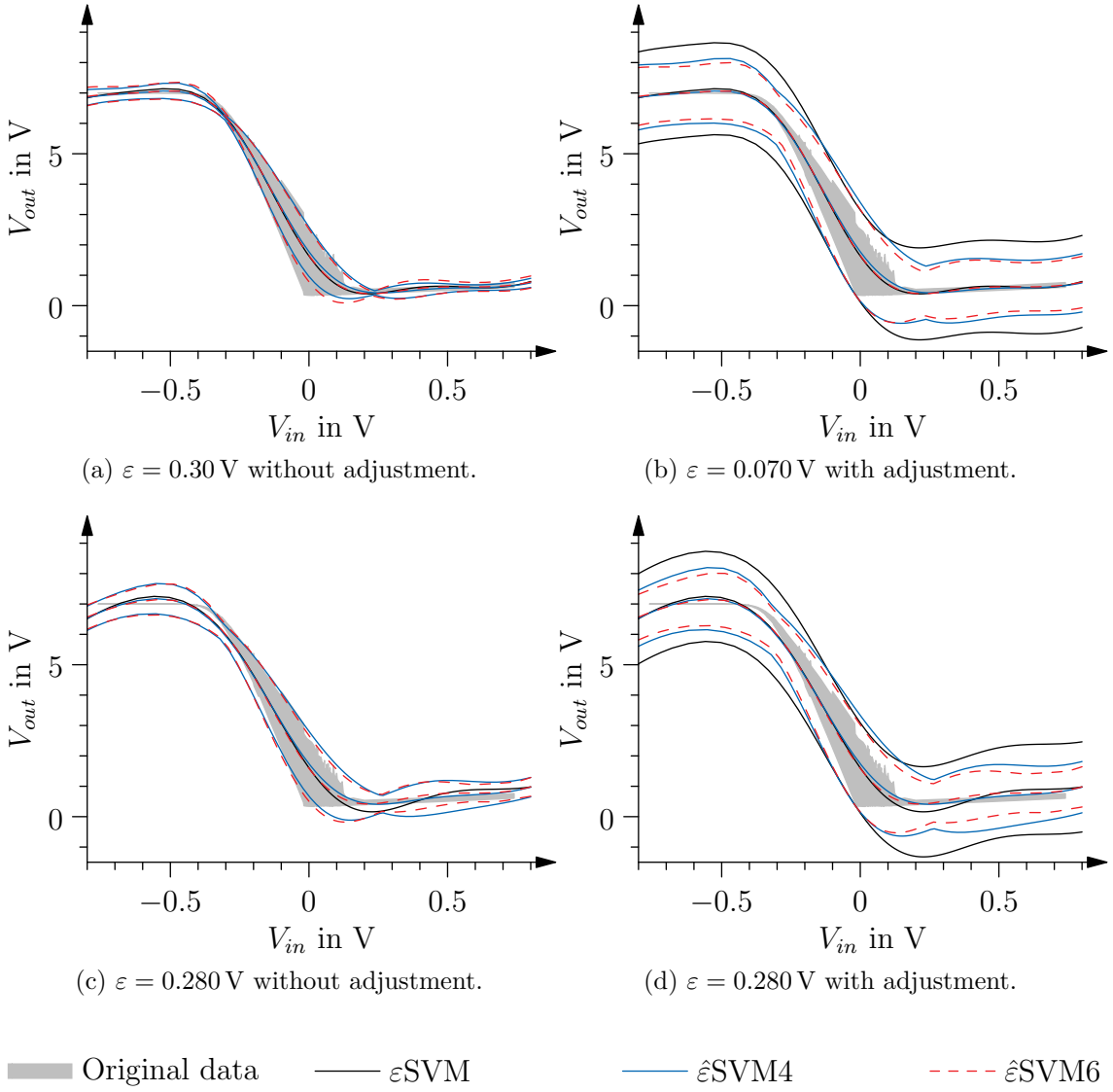
Both sets of figures illustrate the large overestimation introduced by adjustment. Models without adjustment hug the original data tightly outside the amplification region. The modelling error in these models is determined where the minimum output voltage hits approximately 0.7 V. In these models, adjustment is the main reason for the large overestimation.

The plots show that the full original data set is much more ragged around the edges than the training data set suggests. This is not a problem for algorithm adjustment as the maximum deviation from the original data is not observed around the rough patches.

Class A amplifiers operate linearly in their amplification region. This model deliberately includes more than just the amplification region. These additional operation regions pose the greatest challenge for the modelling algorithms.

The original data has zero width for $V_{in} < -0.3\,\mathrm{V}$. The region $V_{in} > 0.1\,\mathrm{V}$ has very small constant width. The affine SVM algorithms model the given data set fine, but with sometimes very large modelling errors. Adjusting the model to enclose the original data leads to large overestimates.

From the overestimation standpoint, models which are more restrictive with regards to the maximum permissible error should be preferred. As these models are trained on a very small training data set, more data points should be used for training.



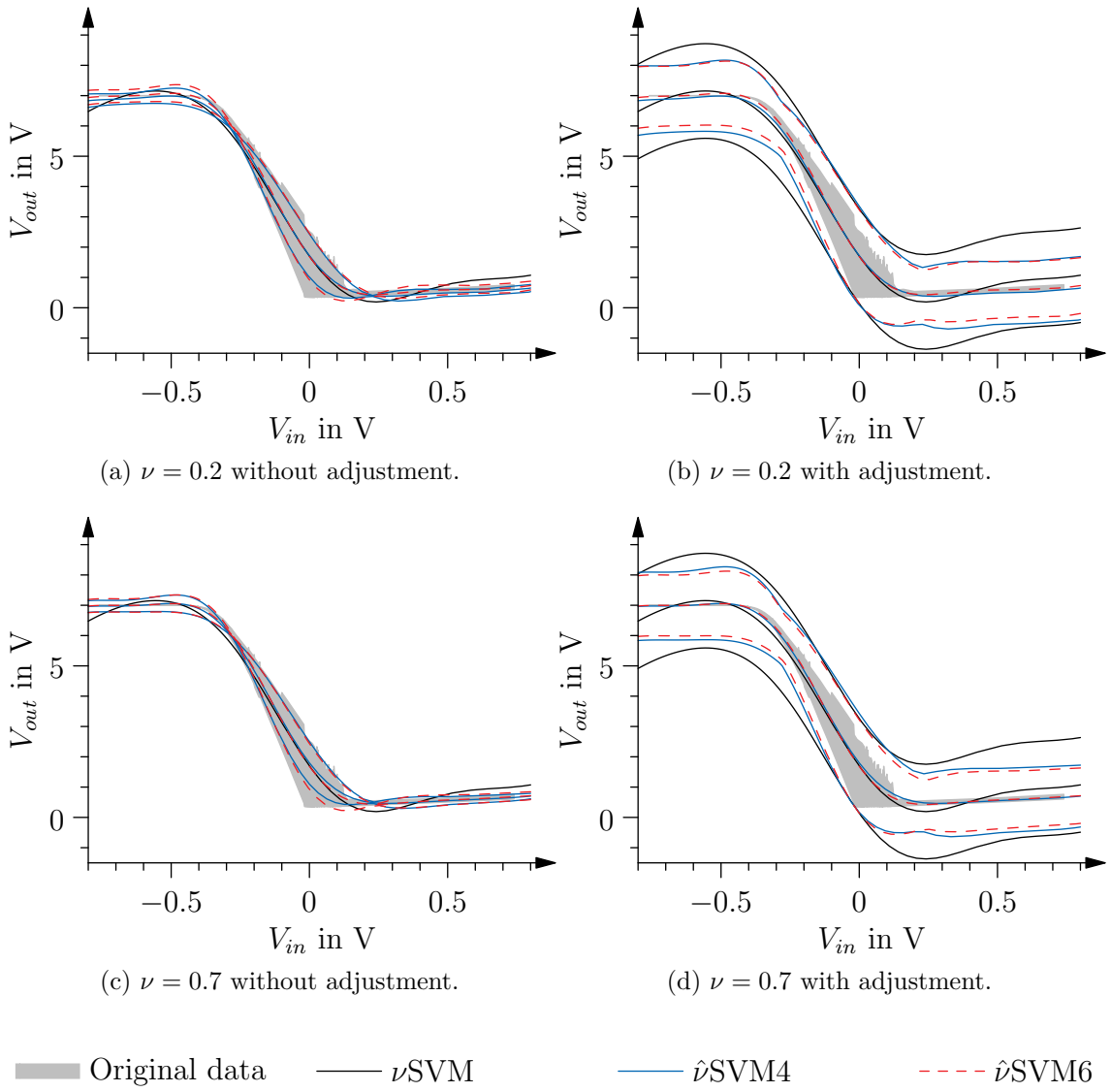
(a) $\varepsilon = 0.30\,\mathrm{V}$ without adjustment.

(b) $\varepsilon = 0.070\,\mathrm{V}$ with adjustment.

(c) $\varepsilon = 0.280\,\mathrm{V}$ without adjustment.

(d) $\varepsilon = 0.280\,\mathrm{V}$ with adjustment.

Figure 5.7.: Characteristic functions of εSVM models with RBF kernels. Models are plotted with and without adjustment.



(a) $\nu = 0.2$ without adjustment.

(b) $\nu = 0.2$ with adjustment.

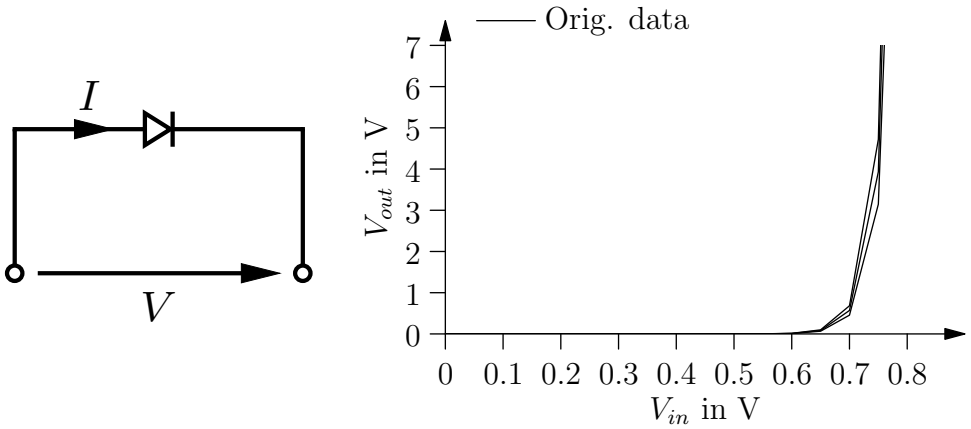
(c) $\nu = 0.7$ without adjustment.

(d) $\nu = 0.7$ with adjustment.

Figure 5.8.: Characteristic functions of νSVM models with RBF kernels. Models are plotted with and without adjustment.

5.4. P-N Junction Diode

The third example is a model of the characteristic function of a p-n junction diode. In this case, $I = f(V)$ is modelled, recreating the Shockley diode equation, also known as the diode law. The characteristic function is strongly non-linear. For voltage smaller than the thermal voltage V_T the diode conducts the saturation current which is very small. For voltages larger than the thermal voltage, the current increases exponentially. Fig. 5.9 shows the component (Fig. 5.9a) and the data used for modelling (Fig. 5.9b).



(a) Schematic diagramme. (b) Data for modelling.

Figure 5.9.: P-N junction diode.

For modelling, the thermal voltage V_{th} is varied by 1 % and the saturation current is varied by 20 %. Modelling data was generated using Saber's MC simulation and a DC sweep. The voltage across the diode was varied from 0.0 V to 0.8 V in steps of 1 mV. 1000 MC samples are drawn for each step in the DC sweep. This number was selected as only two parameters are varied. This simulation setup returns 802 data points. 18 of these data points were used for training. Fig. 5.9b shows the outline of the training data.

This data set is strongly non-linear. For $V < V_{\text{th}}$ the current is close to 0 A and the width of the tube is only a few pA. For $V > V_{\text{th}}$ the current rises quickly and so does the width of the tube. This poses a challenge in modelling as adjustment will decrease the model performance significantly. The diode current was scaled to the range $[0.0\,\text{A}, 1.0\,\text{A}]$ to improve convergence.

Polynomials are the only kernels that work on this data set. To model the quickly rising current after the knee, a relatively high, odd degree has to be chosen. All models presented here are modelled with 9th order polynomials. Ninth order polynomials describe a middle ground between modelling accuracy and simulating the model with AGIAS.

Polynomial kernels have three hyperparameters. The first is the degree, which is set to 9 as discussed above. The second is an offset coefficient which was set to 0.4. The third

hyperparameter scales the scalar product, which is set to 3. This results in the following kernel

$$k(x, x') = (3* < x, x' > +0.4)^9. \tag{5.6}$$

The cost hyperparameter C is set to 10. For εSVR models the maximum permissible error was varied between 0.000 A and 0.050 A in steps of 0.005 A. The trade-off parameter of νSVRs is varied between 0.1 and 1.0 in steps of 0.1.

For the diode, 126 models were created. All model configurations were run with both the CVX solver and the SMO solver. All models with their quality measures and runtimes are documented in Appendix A.3.

Models with the maximum permissible error $\varepsilon = \{0.015\,\text{A}, 0.035\,\text{A}\}$ were selected from εSVR models and with the tradeoff parameter set to $\nu = \{0.2, 0.7\}$ are selected from νSVR models for a more detailed inspection.

Tab. 5.9 gives an overview of the numbers of SVs for selected models. Models generated with the CVX solver cannot be reduced in size by the chosen model order reduction approach. Models trained with the extended SMO solver profit from the solver's selectivity fully. Depending on the setting for the maximum permissible error for εSVRs, only half of the training samples are used as SVs in the model. As νSVR models are optimised for smallest error, only three of the selected models profit from the selectivity of the extended SMO solver. Nominal νSVR models only use 9 while ν̂SVR models require all training samples as SVs.

Table 5.9.: Diode: Overview of the numbers of SVs for SVM models with polynomial kernels generated with CVX and the extended SMO. The training data set contains 18 samples.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.015\,\text{A}$	18	18	18	13	15	16
εSVR $\varepsilon = 0.035\,\text{A}$	18	18	18	10	13	16
νSVR $\nu = 0.2$	18	18	18	9	17	18
νSVR $\nu = 0.7$	18	18	18	9	18	18

The diode data set is strongly non-linear. The region with $V < V_{\mathrm{th}}$ is modelled closely by most models. νSVR models exhibit some oscillation which stems from the ninth-order polynomial. This effect is more subdued in the other selected models. It also does not dominate to the modelling error. For the selected models, the modelling error is calculated somewhere along the diode's forward operation region. Vertical distances grow rapidly in this region. This leads to large modelling errors. Modelling errors for the selected models are available from Tab. 5.10.

Table 5.10.: Modelling error in A for selected diode models.

Modelling error in A	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.015\,\mathrm{A}$	6.72	2.73	2.33	7.05	2.15	4.73
εSVR $\varepsilon = 0.035\,\mathrm{A}$	5.42	1.81	1.62	6.65	0.85	1.06
νSVR $\nu = 0.2$	4.99	3.55	3.49	5.08	0.12	2.39
νSVR $\nu = 0.7$	4.99	3.67	3.58	5.08	5.41	4.08

Overestimation for diode models is huge. The original data has a width in the range of $10^{-12}\,\mathrm{A}$ for the largest part of the input range. The original data has a noticeable width only above the knee. Even without adjusting the models for the modelling errors, overestimation for most data points is very large. Adjusting the models adds to this effect with the model width becoming even larger. Tab. 5.11 gives an overview of the overestimation for adjusted models.

Table 5.11.: Overestimation for selected diode models.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.015\,\mathrm{A}$	$6.97e6$	$3.11e6$	$2.89e6$	$7.29e6$	$2.55e6$	$5.69e6$
εSVR $\varepsilon = 0.035\,\mathrm{A}$	$6.23e6$	$2.73e6$	$2.63e6$	$7.43e6$	$1.95e6$	$2.49e6$
νSVR $\nu = 0.2$	$6.21e6$	$3.54e6$	$3.44e6$	$6.30e6$	$1.22e6$	$3.69e6$
νSVR $\nu = 0.7$	$6.21e6$	$3.63e6$	$3.58e6$	$6.30e6$	$5.86e6$	$4.43e6$

Areal overestimation for the selected diode models is given in Tab. 5.12. These values illustrate what has been observed with the other circuit examples, too: internal variation-aware modelling with affine SVMs leads to smaller overestimation than external variation-aware modelling. Models allowing large maximum permissible errors perform better than models with smaller maximal permissible errors. As with the overestimation, the areal overestimation is dominated by the models behaviour for $V < V_{\mathrm{th}}$.

Table 5.12.: Areal overestimation for selected diode models.

	CVX			Extended SMO		
	SVM	SVM4	SVM6	SVM	SVM4	SVM6
εSVR $\varepsilon = 0.015\,\mathrm{A}$	40.53	19.57	17.57	42.40	17.28	32.89
εSVR $\varepsilon = 0.035\,\mathrm{A}$	36.25	17.44	16.53	43.21	14.56	14.17
νSVR $\nu = 0.2$	36.12	21.81	21.43	36.62	8.94	21.04
νSVR $\nu = 0.7$	36.12	22.49	22.01	36.62	35.47	26.60

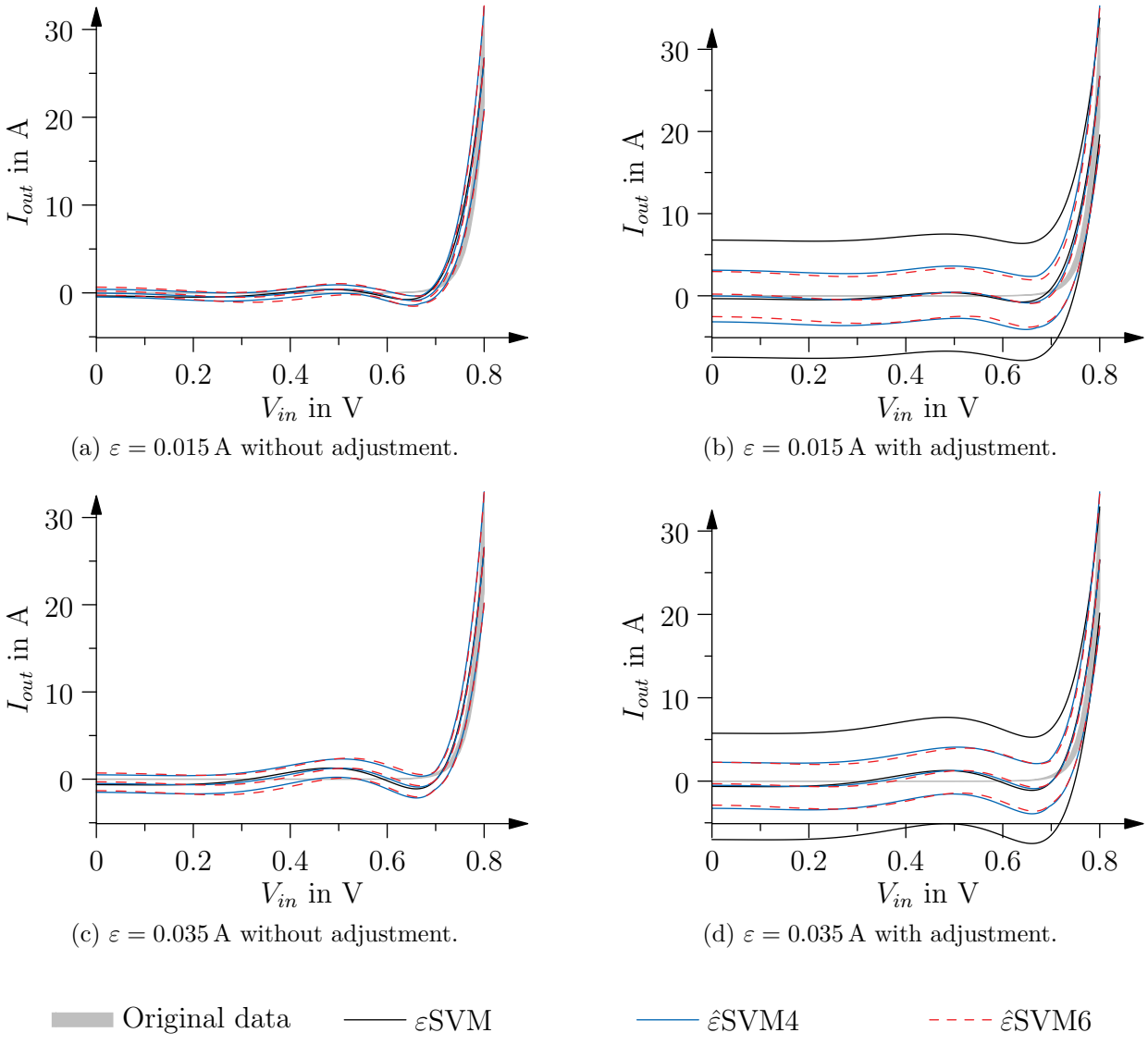
Fig. 5.10 and Fig. 5.11 show the characteristic curves for εSVR and νSVR diode models with and without model adjustment. On the right are models with adjustment. These figures illustrate the huge overestimation for $V < V_{\mathrm{th}}$. In forward operation the distance between the minimum and maximum values is still very close to the original data.

The diode models are different from the two examples presented earlier. These models model the relationship between current and voltage at one component. Therefore, input and output of the models cannot be separated. Currently, the diode models cannot be simulated with AGIAS as the solver cannot find a starting solution for the EPD, the newly introduced weight that captures errors when solving an affine equation system. The EPD that is determined during the evaluation is very large and indicates numerical instability in the solver. The results presented in this chapter were obtained using Matlab to evaluate the model.

The diode model is the only model for which data has been scaled for modelling. Originally output values fall in the range $[0.0\,\mathrm{A}, 30\,\mathrm{A}]$. This data is scaled to the range $[0.0\,\mathrm{A}, 1.0\,\mathrm{A}]$. The maximum permissible errors used in training εSVR models have to be interpreted with respect to the range after scaling. The largest maximum permissible error is set to $\varepsilon = 0.05\,\mathrm{A}$. This means models allow up to 5 % of the output range as error.

Overestimation is a measure that cannot be used for diode models. Area overestimation offers a glimpse into how much the model actually overestimates the original data. Both overestimation properties are dominated by the diode's non-conductive region. In this region the original data has a width of only a few pico amperes while the models all have a significant width.

Polynomials often come with oscillation. In the models selected here, oscillations did occur for large maximum permissible errors, but they did not impact the models negatively. The oscillations could be a cause for large modelling errors. This is not the case with the models shown here.



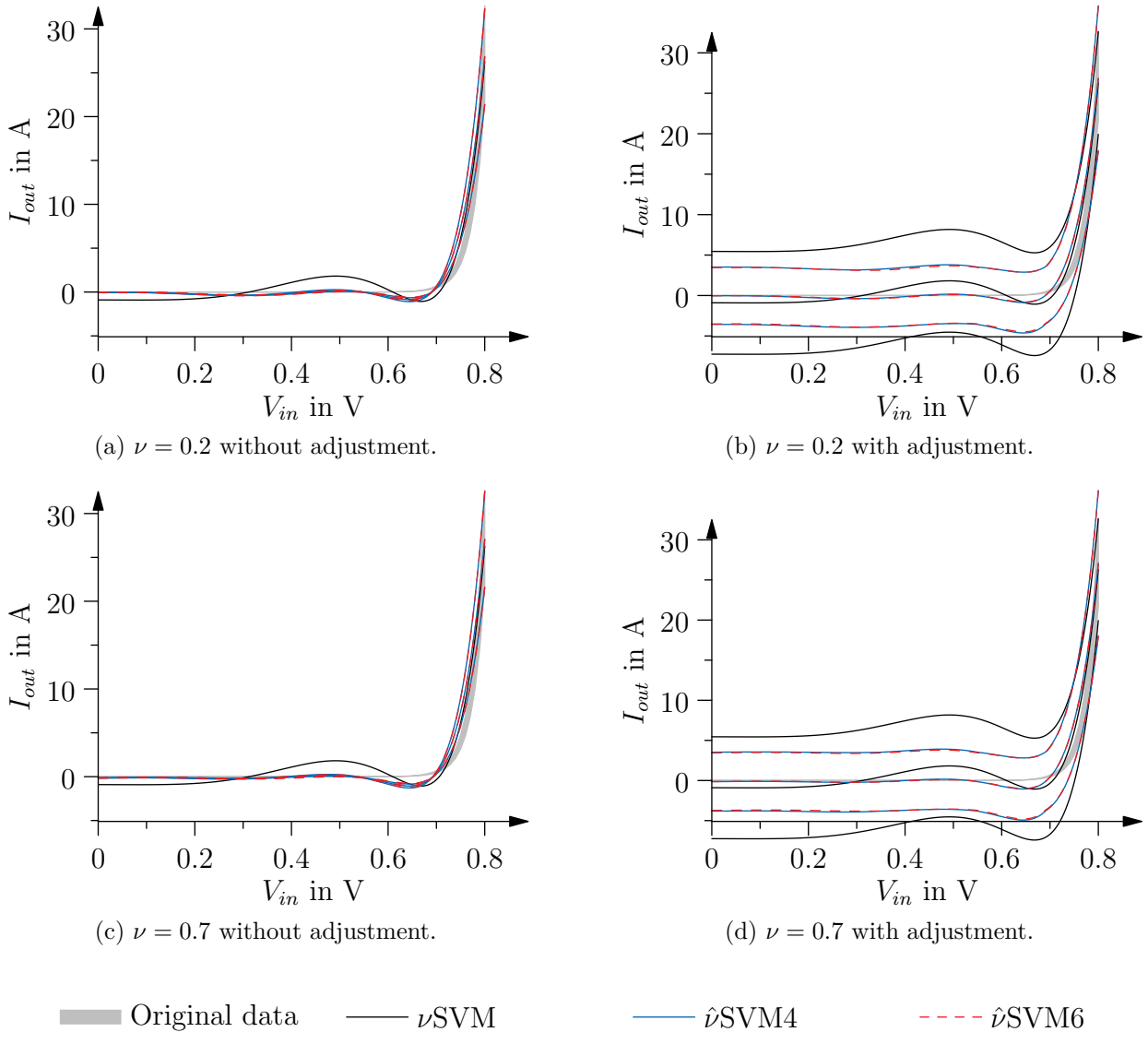
(a) $\varepsilon = 0.015\,\mathrm{A}$ without adjustment.

(b) $\varepsilon = 0.015\,\mathrm{A}$ with adjustment.

(c) $\varepsilon = 0.035\,\mathrm{A}$ without adjustment.

(d) $\varepsilon = 0.035\,\mathrm{A}$ with adjustment.

Figure 5.10.: Characteristic functions of εSVM models with polynomial kernels.



(a) $\nu = 0.2$ without adjustment.

(b) $\nu = 0.2$ with adjustment.

(c) $\nu = 0.7$ without adjustment.

(d) $\nu = 0.7$ with adjustment.

Figure 5.11.: Characteristic functions of νSVM models with polynomial kernels.

5.5. Notes on the Extended SMO

All models were trained with two solvers: the CVX solver and the extended SMO solver. The CVX solver is available as Matlab toolbox. Its optimizers are written in C. The CVX is not selective. It will always return all data points as SVs. The extended SMO extends Platt's SMO to work on optimization problems with two equality constraints. It is constructed to be selective and is implemented in Matlab.

Tab. 5.13 compares the numbers of SVs for εSVR models of the inverting amplifier, Tab. 5.14 contains the numbers of SVs for νSVM models of the inverting amplfier. The inverting amplifier models are trained on the largest data set. In εSVR models the differences in size between the two solvers are on average more pronounced than with νSVR models. The SMO solver rarely returns all data points as support vectors for linear εSVR models.

Table 5.13.: Numbers of SVs for εSVR models of the inverting amplifier for the CVX and SMO solver. Models trained with the CVX solver have been reduced before evaluating the number of SVs. The training data set contains 301 samples.

(a) Linear kernel.

ε in V	0.00	0.15	0.30	0.45	0.60	0.75	0.90	1.05	1.20	1.35	1.50
εSVR, SMO	185	140	111	77	51	24	13	3	5	5	5
εSVR, CVX	301	301	301	84	56	30	12	2	2	301	
ε̂SVR4, SMO	175	162	129	128	115	98	81	64	47	31	21
ε̂SVR4, CVX	301	301	156	183	115	301	81	65	301	96	301
ε̂SVR6, SMO	137	131	120	111	86	74	81	46	51	7	8
ε̂SVR6, CVX	301	280	189	223	139	301	301	301	301	63	22

(b) RBF kernel.

ε in V	0.00	0.15	0.30	0.45	0.60	0.75	0.90	1.05	1.20	1.35	1.50
εSVR, SMO	289	77	27	18	10	13	6	6	6	6	6
εSVR, CVX	301	80	38	27	11	8	7	6	6	6	6
ε̂SVR4, SMO	301	63	35	22	18	15	7	7	7	7	7
ε̂SVR4, CVX	301	114	118	45	22	42	68	31	8	21	11
ε̂SVR6, SMO	301	96	30	25	19	16	4	4	7	4	4
ε̂SVR6, CVX	301	204	116	52	34	25	13	79	6	28	7

νSVR models do not profit from the extended SMO's selectivity as much as εSVR models. If the trade-off parameter is larger than 0.5, the number of SVs does not change anymore. As νSVR models minimize the maximum permissible error, they tend to have more SVs than εSVR models.

Tab. 5.15 compares the number of SVs of εSVR class A amplifier models. νSVR models have been ommited as they show the same effect as νSVR inverting amplifier models. Data

Table 5.14.: Numbers of SVs for νSVR models of the inverting amplifier for the CVX and SMO solver. Models trained with the CVX solver have been reduced before evaluating the number of SVs. The training data set contains 301 samples.

(a) Linear kernel.

ν	0.1	0.2	0.3	≥ 0.4
νSVR, SMO	139	139	139	139
νSVR, CVX	301	301	301	301
ν̂SVR4, SMO	51	208	280	300
ν̂SVR4, CVX	301	301	301	301
ν̂SVR6, SMO	146	244	297	301
ν̂SVR6, CVX	301	301	301	301

(b) RBF kernel.

ν	0.1	0.2	0.3	0.4	≥ 0.5
νSVR, SMO	257	257	257	257	257
νSVR, CVX	301	301	301	301	301
ν̂SVR4, SMO	58	130	179	197	219
ν̂SVR4, CVX	236	228	297	301	301
ν̂SVR6, SMO	158	223	270	291	298
ν̂SVR6, CVX	152	225	279	292	301

for the class A amplifier show that the extended SMO is selective on a small data set with an non-linear kernel.

Table 5.15.: Number of SVs for class A amplifier εSVR models for the CVX and SMO solver. Models trained with the CVX solver have been reduced before evaluating the number of SVs. The training set contains 31 data points.

ε in mV	0.0	35	70	105	140	175	210	245	280	315	350
εSVR, SMO	28	19	16	14	14	12	12	12	6	12	6
εSVR, CVX	31	18	15	12	10	10	9	8	6	5	5
ε̂SVR4, SMO	31	26	23	17	17	14	12	8	8	10	8
ε̂SVR4, CVX	31	25	20	17	17	13	12	10	10	8	8
ε̂SVR6, SMO	31	27	22	19	18	12	12	13	11	9	11
ε̂SVR6, CVX	31	24	21	20	15	12	10	11	9	9	10

Tab. 5.16 compares the numbers of SVs for εSVR diode models. All models are trained on 18 training samples. The table compares only models trained with the extended SMO as models trained with the CVX could not be reduced. νSVR models are ommitted as the algorithm does not select on these models. The extended SMO returns models with only half the training samples as SVs provided the configuration allows a large enough error.

Table 5.16.: Numbers of SVs for εSVR models of the diode for the SMO solver. The training data set contains 18 samples. Models trained with the CVX solver could not be reduced and are not included here.

ε in mA	0.0	5	10	15	20	25	30	35	40	45	50
εSVR	16	13	13	13	13	12	12	10	10	9	9
ε̂SVR4	16	15	12	15	15	14	13	13	13	14	12
ε̂SVR6	15	17	12	16	12	11	14	16	13	12	11

Tab. 5.17 gives an overview of all runtimes for εSVR models for the inverting amplifer. All νSVR models' runtimes are available from Tab. 5.18. Runtimes for models trained with the CVX solver evolve as expected: nominal models train in around 48 s, ε̂SVR4 and ν̂SVR4 models take approximately three times as long at approximately 140 s, and ε̂SVR6 and ν̂SVR6 models take the longest at around 305 s — around twice as long as ε̂SVR4 and ν̂SVR4 models.

Runtimes of the extended SMO solver vary wildly. Linear εSVR models, ε̂SVR6 models with RBF kernels, and all νSVR models take longer to train than with the CVX solver. νSVR models and ν̂SVR4 with $\varepsilon \geq 0.9\,\mathrm{V}$ train significantly faster than with the CVX solver.

The runtimes for class A amplifiers are available from Tab. 5.19. Runtimes for εSVR models trained with the extended SMO still vary wildly. For the νSVR models, runtimes are much more streamlined. Training takes longer than with the CVX solver, but times are more consistent.

Runtimes for the diode models are compared in Tab. 5.20. Training diode models with the SMO solver usually takes much longer than with the CVX solver, but runtimes are consistent within a model class.

The solvers running within the CVX toolbox profit from being written in C. This explains at least in part, why these run so much faster than the extended SMO. Although no data is available here, SVMs trained with LibSVM were used in first experiments with external variation-aware modelling. LibSVM's C implementation was used for training and performed just as fast as the CVX solver.

Blaming runtime differences solely on the choice of programming language and non-optimized code is too simple-minded. The extended SMO solver solves the optimization problem iteratively. In every iteration, it calculates a heuristic to determine the pair or tripe of data points for which the optimization function is minimized. It then calculates new weights and updates the error terms for the next iteration. Optimization ends if the improvement of the model sinks below a given threshold. There is no limit on the number of iterations to reach the improvement threshold.

Additionally, SVMs are known to show unpredictable convergence behaviour, if output data is not scaled before training. As the two examples which show wildly varying runtimes were trained on unscaled data, this observation holds true for the extended SMO.

Table 5.17.: Runtimes in s for εSVR models of the inverting amplifier for the CVX and SMO solver.

(a) Linear kernel.

	εSVR		ε̂SVR4		ε̂SVR6	
ε	SMO	CVX	SMO	CVX	SMO	CVX
0.00	10943.890	52.740	2313.080	139.126	3356.961	300.823
0.15	3992.644	46.138	2122.242	140.241	3382.514	301.555
0.30	5107.015	45.588	1723.244	140.338	2982.000	298.950
0.45	10228.137	46.103	3693.044	140.249	3362.397	301.065
0.60	24875.594	45.809	5543.656	141.054	3157.383	296.443
0.75	31857.805	46.644	8858.616	141.043	3367.764	298.268
0.90	255.930	48.258	2544.609	139.417	3477.080	299.782
1.05	515.150	47.798	6797.498	140.381	1407.597	297.836
1.20	975.370	46.646	4427.342	140.839	4230.705	301.815
1.35	974.907	46.197	2601.076	140.680	2832.872	301.180
1.50	953.319	45.347	632.906	140.213	2703.262	301.103

(b) RBF kernel.

	εSVR		ε̂SVR4		ε̂SVR6	
ε	SMO	CVX	SMO	CVX	SMO	CVX
0.00	1331.636	45.535	2819.798	142.744	10748.438	307.921
0.15	164.888	44.703	47212.329	144.626	35806.283	301.446
0.30	40.500	45.430	108.280	140.869	3897.379	303.091
0.45	33.595	45.425	48381.961	141.330	3611.629	300.877
0.60	29.002	44.924	48204.843	140.809	23922.556	305.129
0.75	32.848	45.432	131.732	141.888	3301.648	302.350
0.90	25.048	44.957	45.821	142.504	2767.277	304.918
1.05	25.018	45.324	44.996	141.165	2792.945	304.138
1.20	24.952	45.286	45.837	141.949	2807.070	303.211
1.35	24.921	44.848	45.509	143.754	2543.629	306.952
1.50	24.697	44.862	45.894	141.232	2501.438	305.801

Table 5.18.: Runtimes in s for νSVR models of the inverting amplifier for the CVX and SMO solver.

(a) Linear kernel.

	νSVR		ν̂SVR4		ν̂SVR6	
ν	SMO	CVX	SMO	CVX	SMO	CVX
0.1	1149.853	49.220	11320.919	143.042	16923.135	306.490
0.2	1149.150	49.806	25865.832	143.316	42890.786	300.061
0.3	1138.419	50.104	33962.535	141.768	69740.263	302.159
0.4	1157.486	49.860	45515.602	144.115	84527.863	303.677
0.5	1143.836	49.484	66676.825	143.068	107442.328	305.584
0.6	1140.483	49.956	66908.327	142.649	109225.551	300.392
0.7	1149.941	49.456	66109.814	142.010	109727.976	300.519
0.8	1147.962	49.861	67669.059	141.470	110580.835	305.379
0.9	1144.473	49.818	67229.627	142.527	109780.658	303.009
1.0	1146.399	49.856	67701.318	141.748	111487.404	301.625

(b) RBF kernel.

	νSVR		ν̂SVR4		ν̂SVR6	
ν	SMO	CVX	SMO	CVX	SMO	CVX
0.1	341.101	45.110	1395.444	144.640	3381.126	310.922
0.2	338.187	45.406	4043.718	144.343	7711.609	309.153
0.3	341.330	45.031	6495.287	145.646	17981.850	304.674
0.4	339.480	44.979	10168.727	145.199	26134.396	304.560
0.5	341.123	44.498	14680.257	144.346	51194.106	307.333
0.6	338.339	45.217	14931.892	145.226	52291.515	310.621
0.7	345.023	44.970	14811.167	144.790	52581.182	309.855
0.8	340.830	44.773	14715.273	144.295	51703.803	309.866
0.9	341.378	44.582	14922.307	146.955	51851.606	310.697
1.0	340.431	44.772	14884.491	146.188	52111.031	310.447

Table 5.19.: Runtimes in s for all class A amplifier models for the CVX and SMO solver.

(a) εSVR models.

	εSVR		ε̂SVR4		ε̂SVR6	
ε	SMO	CVX	SMO	CVX	SMO	CVX
0.000	28.400	25.133	130.183	11.269	516.292	13.128
0.035	18621.356	10.168	36558.218	10.949	42459.570	13.064
0.070	18882.784	9.804	35824.397	10.962	9190.917	13.045
0.105	19199.261	9.892	35629.246	11.197	8870.806	13.146
0.140	18890.565	9.757	35591.034	10.943	9178.347	13.218
0.175	18630.890	9.902	34623.524	11.075	7418.083	12.993
0.210	19047.435	9.786	13.316	11.331	7634.555	13.311
0.245	42.286	9.681	13.005	11.013	8094.898	13.304
0.280	10.267	9.881	12.804	10.953	7325.498	13.110
0.315	19250.386	9.735	12.499	11.093	6623.903	13.379
0.350	30.572	9.863	12.390	11.207	7191.846	13.235

(b) νSVR models.

	νSVR		ν̂SVR4		ν̂SVR6	
ν	SMO	CVX	SMO	CVX	SMO	CVX
0.1	30.739	9.810	36.697	11.372	49.104	13.784
0.2	30.346	9.899	88.969	11.456	98.859	13.802
0.3	30.630	10.009	127.838	11.400	219.355	13.779
0.4	30.327	9.808	180.600	11.503	316.319	13.681
0.5	30.730	9.780	231.588	11.405	508.159	13.561
0.6	30.540	9.788	235.259	11.231	489.524	13.592
0.7	30.921	9.809	230.707	11.403	503.521	13.671
0.8	30.225	9.875	233.738	11.469	493.892	13.582
0.9	30.776	9.766	232.205	11.420	496.640	13.570
1.0	30.975	9.804	232.962	11.485	505.001	13.640

Table 5.20.: Runtimes in s for all diode models with for the CVX and SMO solver.

(a) εSVR models.

	εSVR		ε̂SVR4		ε̂SVR6	
ε	SMO	CVX	SMO	CVX	SMO	CVX
0.000	1742.677	10.890	2826.215	9.889	4467.333	10.788
0.005	1384.832	9.444	2351.300	10.259	4152.247	10.840
0.010	1441.301	9.386	2085.790	9.858	3746.267	11.393
0.015	2081.033	9.847	2052.813	10.243	4134.379	10.894
0.020	2774.992	9.452	2114.147	10.340	4341.665	10.773
0.025	2423.213	9.528	2260.814	9.992	4176.730	11.331
0.030	2554.148	9.874	2095.042	10.315	4388.949	10.926
0.035	2234.318	9.544	2258.589	10.355	4670.135	10.850
0.040	2116.340	9.407	2033.453	10.383	3626.911	11.365
0.045	2002.916	9.763	2606.741	10.021	285.038	10.846
0.050	1419.553	9.287	3860.840	10.194	2882.571	11.027

(b) νSVR models.

	νSVR		ν̂SVR4		ν̂SVR6	
ν	SMO	CVX	SMO	CVX	SMO	CVX
0.1	1439.645	9.701	2773.102	10.648	5196.824	11.229
0.2	1423.210	9.473	4172.945	10.290	8673.961	11.546
0.3	1433.214	9.712	6015.294	10.337	10541.161	11.216
0.4	1442.023	9.450	7801.963	10.175	13676.213	11.079
0.5	1436.027	9.446	9229.548	10.157	14656.770	11.403
0.6	1440.176	9.750	9294.388	10.266	14366.720	10.944
0.7	1443.778	10.018	9176.258	10.470	14476.337	11.167
0.8	1432.628	9.585	9147.337	10.313	14400.468	11.582
0.9	1441.119	10.020	9151.134	10.272	14570.998	11.079
1.0	1424.439	9.384	9218.937	10.694	14319.017	11.176

6. Summary

Target of this work was the creation of behavioural models of analogue circuits that describe the effects of parameter variations and can be simulated with AGIAS. First, for simulation with AGIAS parameter variations have to be represented as affine parameters. Second, models have to enclose the original data fully. Third, models have to be written in mathematical forms that are numerically stable even when calculated with interval arithmetic.

Sec. 4.2 explored two concepts for creating behavioural models of analogue circuits which describe the effects of parameter variations: external and internal variation-aware modelling. For external variation-aware modelling, first a model on the nominal behaviour of the circuit was created. Then parallel translation was used to create one affine parameter that extends the model's offset to enclose data fully. The distance for the parallel translation is determined by calculating the largest distance between the nominal model and the minimum and maximum of the original data respectively.

Internal variation-aware modelling requires the modelling algorithm to be adapted to determine affine parameters during training. This work extends Support Vector Machines with two affine parameters. These parameters were introduced to linear SVMs. Training a linear SVM means determining its slope and offset. Affine SVMs determine affine intervals for these two parameters. The offset parameter describes a parallel translation, an affine slope describes a family of lines. Extending affine SVMs with the kernel trick to non-linear SVMs does not change the interpretation of the offset parameter. The affine slope in non-linear SVMs is a function of the input value.

Sec. 4.3 provided affine εSVR and affine νSVR algorithms. Like their nominal counterparts, the $\hat{\varepsilon}$SVR algorithms allow the user to determine a maximum permissible error. $\hat{\nu}$SVR algorithms include the maximum permissible error into the cost function and optimize this parameter. Like their nominal counterparts they allow a trade-off between the overall model error and the maximum permissible error. Both models use the maximum permissible error as initial width of the affine offset.

Two algorithms have been developed for each algorithm class: algorithms which use minimum, nominal, and maximum values of the original data are called $\hat{\varepsilon}$SVR6 and $\hat{\nu}$SVR6; algorithms that only use minimum, and maximum values are called $\hat{\varepsilon}$SVR4 and $\hat{\nu}$SVR4. The number appended to the algorithm name refers to the number of error constraints. For comparison: the original SVM algorithms have two error constraints.

LibSVM, one standard implementation for training SVMs, implements the Sequential Optimal Minimization algorithm. This algorithm solves the quadratic optimization problem by sequentially selecting two data points for optimization. The resulting optimization

problem in two variables can be solved analytically. The SMO depends heavily on the fact that the original optimization problem has one equality constraint that links the selected variables. The optimization problem for the affine SVMs has two equality constraints. Sec. 4.5 extended the SMO to operate on optimization problems with two equality constraints. It also introduces a new heuristic for selecting data pairs or triples as not all variables of the optimization problem are linked by the second equality constraints. Sec. 5.5 shows that the extended SMO is selective, but requires additional work to lower its runtime.

Three circuits were modelled using the new SVM algorithms: the inverting amplifier, a class A amplifier and a diode. For the first two circuits classical behavioural models are created by modelling $V_{\text{out}} = f(V_{\text{in}})$. The diode model is closer to a physical component model as it models $I = f(V)$. The inverting amplifier is the most linear circuit, the class A amplifier is non-linear in the range chosen for modelling and the diode is strongly non-linear by nature.

Sec. 5.2 presents the results for modelling the inverting amplifier. Behavioural models are created using linear and RBF kernels. All models that have been created have been simulated with AGIAS. Models with the RBF kernel create slightly better models as these models trace the non-linearities of the circuit more closely than linear kernels. Against expectations, the difference between affine models with 4 and 6 error constraints is quite small.

The class A amplifier was modelled using RBF kernels in Sec. 5.3. All models that have been created have been simulated with AGIAS. Class A amplifier models without adjustment enclose the original data very closely. $\hat{\varepsilon}$SVR models with smaller maximum permissible error model the data outside the operation region more closely. Even with adjustment, this leads to lower overestimation and mismatch. For $\hat{\nu}$SVR the differences between high and low trade-off parameters is not as pronounced as the maximum permissible error is one result of the training.

Diodes are modelled in Sec. 5.4. Working models could only be obtained with ninth order polynomial kernels. With their different nature, diode models offer a different result. While diodes can be used in nominal simulation with AGIAS, affine simulations fail. Therefore, diode models were evaluated in Matlab. Due to the nature of the original data, diode models have large modelling errors and due to the parallel translation, very large overestimation and mismatch. Therefore, diode models require more work on the solver in AGIAS.

All models presented in Chap. 5 have only one input, they only consider static behaviour, and two of three models do not model their ports electrical. In future work, these points should be addressed. Moving to more than one input was already considered in the algorithms presented in Sec. 4.3, but has so far not been tested. One challenge is data generation. Dynamic models require information about dynamic behaviour which also turns this into a data generation problem at first glance. However, SVMs do not offer direct capabilities for modelling dynamics. Modelling electric behaviour of all ports requires more data again, but it also requires more than one model as SVMs do not support multi-input-multi-output models in one SVM.

A. Generated Models

A.1. Inverting Amplifier

Table A.1.: Inverting amplifier εSVR models trained with the CVX solver.

(a) Linear kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	52.740	301	2.537	2.537	2.546	2.580	0.000	0.000
0.15	46.138	301	2.085	2.085	2.244	2.273	0.267	0.133
0.30	45.588	301	2.013	2.024	2.333	2.364	0.534	0.266
0.45	46.103	84	2.017	2.029	2.489	2.521	0.801	0.399
0.60	45.809	56	2.002	2.015	2.625	2.659	1.068	0.532
0.75	46.644	30	1.972	1.985	2.746	2.782	1.334	0.665
0.90	48.258	12	1.944	1.958	2.869	2.907	1.601	0.798
1.05	47.798	2	1.937	1.952	3.013	3.053	1.868	0.931
1.20	46.646	2	1.944	1.959	3.171	3.213	2.135	1.064
1.35	46.197	301	1.950	1.967	3.329	3.373	2.402	1.197
1.50	45.347	301	1.957	1.974	3.487	3.533	2.669	1.330

(b) RBF kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	45.535	301	2.361	2.383	2.392	2.424	0.000	0.000
0.15	44.703	80	2.176	2.197	2.356	2.387	0.267	0.133
0.30	45.430	38	2.062	2.082	2.391	2.422	0.534	0.266
0.45	45.425	27	2.028	2.045	2.504	2.537	0.801	0.399
0.60	44.924	11	1.933	1.948	2.558	2.591	1.068	0.532
0.75	45.432	8	1.766	1.780	2.539	2.573	1.334	0.665
0.90	44.957	7	1.754	1.766	2.676	2.711	1.601	0.798
1.05	45.324	6	1.804	1.816	2.877	2.914	1.868	0.931
1.20	45.286	6	1.794	1.806	3.017	3.056	2.135	1.064
1.35	44.848	6	1.784	1.795	3.157	3.199	2.402	1.197
1.50	44.862	6	1.774	1.785	3.298	3.341	2.669	1.330

Table A.2.: Inverting amplifier ε̂SVR4 models trained with the CVX solver.

(a) Linear kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	139.126	301	0.087	0.087	1.124	1.341	1.781	1.092
0.15	140.241	301	0.412	0.413	1.491	1.574	1.737	1.006
0.30	140.338	156	0.648	0.651	1.720	1.755	1.731	0.953
0.45	140.249	183	0.823	0.827	1.881	1.896	1.736	0.920
0.60	141.054	115	0.958	0.966	2.013	2.023	1.759	0.907
0.75	141.043	301	1.079	1.088	2.138	2.150	1.797	0.910
0.90	139.417	81	1.189	1.199	2.249	2.269	1.827	0.915
1.05	140.381	65	1.260	1.271	2.335	2.364	1.874	0.934
1.20	140.839	301	1.085	1.097	2.386	2.410	2.262	1.129
1.35	140.680	96	0.586	0.596	2.221	2.235	2.832	1.421
1.50	140.213	301	0.645	0.656	2.445	2.461	3.120	1.565

(b) RBF kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	142.744	301	0.000	0.000	2.192	2.197	3.890	1.915
0.15	144.626	114	0.000	0.000	1.792	1.757	3.163	1.532
0.30	140.869	118	0.057	0.074	1.542	1.457	2.606	1.205
0.45	141.330	45	0.161	0.175	1.736	1.624	2.767	1.260
0.60	140.809	22	0.241	0.253	1.853	1.754	2.859	1.305
0.75	141.888	42	0.346	0.356	1.974	1.901	2.931	1.342
0.90	142.504	68	0.463	0.472	2.111	2.053	2.972	1.371
1.05	141.165	31	0.525	0.534	2.261	2.214	3.126	1.457
1.20	141.949	8	0.577	0.585	2.481	2.441	3.438	1.610
1.35	143.754	21	0.627	0.635	2.703	2.672	3.754	1.767
1.50	141.232	11	0.677	0.685	2.927	2.903	4.069	1.925

Table A.3.: Inverting amplifier ε̂SVR6 models trained with the CVX solver.

(a) Linear kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	300.823	301	0.087	0.087	1.124	1.341	1.781	1.092
0.15	301.555	280	0.415	0.416	1.538	1.627	1.809	1.050
0.30	298.950	189	0.617	0.619	1.805	1.854	1.921	1.067
0.45	301.065	223	0.764	0.768	1.996	2.027	2.053	1.106
0.60	296.443	139	0.892	0.899	2.207	2.226	2.180	1.148
0.75	298.268	301	1.027	1.035	2.386	2.399	2.269	1.173
0.90	299.782	301	1.154	1.165	2.580	2.592	2.402	1.227
1.05	297.836	301	1.221	1.232	2.675	2.690	2.476	1.253
1.20	301.815	301	1.316	1.328	2.672	2.697	2.349	1.174
1.35	301.180	63	1.061	1.073	2.542	2.567	2.575	1.286
1.50	301.103	22	0.653	0.663	2.451	2.467	3.116	1.563

(b) RBF kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	307.921	301	0.000	0.000	2.216	2.228	3.933	1.942
0.15	301.446	204	0.000	0.000	1.834	1.810	3.238	1.578
0.30	303.091	116	0.057	0.074	1.542	1.457	2.606	1.205
0.45	300.877	52	0.174	0.188	1.732	1.624	2.739	1.249
0.60	305.129	34	0.263	0.275	1.851	1.756	2.817	1.287
0.75	302.350	25	0.362	0.372	1.979	1.903	2.915	1.330
0.90	304.918	13	0.428	0.437	2.111	2.049	3.041	1.399
1.05	304.138	79	0.493	0.501	2.242	2.197	3.157	1.471
1.20	303.211	6	0.545	0.553	2.465	2.424	3.464	1.623
1.35	306.952	28	0.596	0.603	2.686	2.655	3.782	1.780
1.50	305.801	7	0.646	0.653	2.911	2.887	4.095	1.938

Table A.4.: Inverting amplifier νSVR models trained with the CVX solver.

(a) Linear kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	49.220	301	2.537	2.537	2.546	2.58	0.0	0.0
0.2	49.806	301	2.537	2.537	2.546	2.58	0.0	0.0
0.3	50.104	301	2.537	2.537	2.546	2.58	0.0	0.0
0.4	49.860	301	2.537	2.537	2.546	2.58	0.0	0.0
0.5	49.484	301	2.537	2.537	2.546	2.58	0.0	0.0
0.6	49.956	301	2.537	2.537	2.546	2.58	0.0	0.0
0.7	49.456	301	2.537	2.537	2.546	2.58	0.0	0.0
0.8	49.861	301	2.537	2.537	2.546	2.58	0.0	0.0
0.9	49.818	301	2.537	2.537	2.546	2.58	0.0	0.0
1.0	49.856	301	2.537	2.537	2.546	2.58	0.0	0.0

(b) RBF kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	45.110	301	2.361	2.383	2.392	2.424	0.0	0.0
0.2	45.406	301	2.361	2.383	2.392	2.424	0.0	0.0
0.3	45.031	301	2.361	2.383	2.392	2.424	0.0	0.0
0.4	44.979	301	2.361	2.383	2.392	2.424	0.0	0.0
0.5	44.498	301	2.361	2.383	2.392	2.424	0.0	0.0
0.6	45.217	301	2.361	2.383	2.392	2.424	0.0	0.0
0.7	44.970	301	2.361	2.383	2.392	2.424	0.0	0.0
0.8	44.773	301	2.361	2.383	2.392	2.424	0.0	0.0
0.9	44.582	301	2.361	2.383	2.392	2.424	0.0	0.0
1.0	44.772	301	2.361	2.383	2.392	2.424	0.0	0.0

Table A.5.: Inverting amplifier ŷSVR4 models trained with the CVX solver.

(a) Linear kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	143.042	301	1.658	1.666	2.119	2.129	0.619	0.380
0.2	143.316	301	0.441	0.441	1.418	1.525	1.531	0.939
0.3	141.768	301	0.110	0.110	1.143	1.349	1.759	1.078
0.4	144.115	301	0.087	0.087	1.122	1.338	1.777	1.089
0.5	143.068	301	0.087	0.087	1.125	1.342	1.782	1.092
0.6	142.649	301	0.087	0.087	1.124	1.341	1.781	1.092
0.7	142.010	301	0.092	0.092	1.125	1.339	1.772	1.086
0.8	141.470	301	0.089	0.089	1.122	1.337	1.773	1.087
0.9	142.527	301	0.087	0.087	1.121	1.337	1.775	1.088
1.0	141.748	301	0.087	0.087	1.126	1.343	1.783	1.093

(b) RBF kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	144.640	236	0.394	0.395	1.462	1.443	1.838	0.908
0.2	144.343	228	0.082	0.083	1.514	1.490	2.511	1.226
0.3	145.646	297	0.000	0.000	1.625	1.604	2.835	1.398
0.4	145.199	301	0.000	0.000	1.667	1.647	2.908	1.436
0.5	144.346	301	0.000	0.000	1.661	1.640	2.897	1.430
0.6	145.226	301	0.000	0.000	1.661	1.640	2.897	1.430
0.7	144.790	301	0.000	0.000	1.661	1.640	2.897	1.430
0.8	144.295	301	0.000	0.000	1.661	1.640	2.897	1.430
0.9	146.955	301	0.000	0.000	1.660	1.640	2.897	1.430
1.0	146.188	301	0.000	0.000	1.661	1.640	2.897	1.430

Table A.6.: Inverting amplifier ν̂SVR6 models trained with the CVX solver.

(a) Linear kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	306.490	301	1.496	1.503	2.197	2.215	0.976	0.598
0.2	300.061	301	0.614	0.615	1.562	1.642	1.446	0.886
0.3	302.159	301	0.266	0.267	1.287	1.439	1.661	1.018
0.4	303.677	301	0.087	0.087	1.122	1.338	1.777	1.089
0.5	305.584	301	0.087	0.087	1.124	1.341	1.781	1.092
0.6	300.392	301	0.087	0.087	1.124	1.341	1.781	1.092
0.7	300.519	301	0.087	0.087	1.126	1.343	1.783	1.093
0.8	305.379	301	0.094	0.094	1.127	1.340	1.770	1.085
0.9	303.009	301	0.101	0.101	1.135	1.345	1.766	1.082
1.0	301.625	301	0.087	0.087	1.126	1.342	1.783	1.093

(b) RBF kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	310.922	152	0.351	0.353	1.478	1.456	1.951	0.956
0.2	309.153	225	0.000	0.000	1.520	1.492	2.660	1.301
0.3	304.674	279	0.000	0.000	1.646	1.626	2.872	1.418
0.4	304.560	292	0.000	0.000	1.667	1.650	2.908	1.439
0.5	307.333	301	0.000	0.000	1.664	1.646	2.901	1.435
0.6	310.621	301	0.000	0.000	1.663	1.646	2.901	1.435
0.7	309.855	301	0.000	0.000	1.664	1.646	2.901	1.435
0.8	309.866	301	0.000	0.000	1.664	1.646	2.901	1.435
0.9	310.697	301	0.000	0.000	1.664	1.646	2.901	1.435
1.0	310.447	301	0.000	0.000	1.663	1.646	2.901	1.435

Table A.7.: Inverting amplifier εSVR models trained with the SMO solver.

(a) Linear kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	10943.890	185	2.537	2.537	7.641	7.741	0.000	0.000
0.15	3992.644	140	2.097	2.097	8.090	8.197	0.267	0.133
0.30	5107.015	111	2.075	2.087	6.573	6.659	0.534	0.266
0.45	10228.137	77	2.474	2.489	2.935	2.974	0.801	0.399
0.60	24875.594	51	2.561	2.578	3.173	3.215	1.068	0.532
0.75	31857.805	24	2.630	2.647	3.393	3.437	1.334	0.665
0.90	255.930	13	1.929	1.943	6.741	6.830	1.601	0.798
1.05	515.150	3	1.893	1.908	6.786	6.875	1.868	0.931
1.20	975.370	5	1.517	1.531	5.802	5.878	2.135	1.064
1.35	974.907	5	1.367	1.381	5.473	5.545	2.402	1.197
1.50	953.319	5	1.217	1.231	5.200	5.268	2.669	1.330

(b) RBF kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	1331.636	289	2.229	2.251	6.756	6.845	0.000	0.000
0.15	164.888	77	2.314	2.335	7.160	7.254	0.267	0.133
0.30	40.500	27	1.902	1.919	6.063	6.143	0.534	0.266
0.45	33.595	18	1.726	1.726	5.385	5.456	0.801	0.399
0.60	29.002	10	1.854	1.854	5.851	5.927	1.068	0.532
0.75	32.848	13	1.951	1.955	6.534	6.620	1.334	0.665
0.90	25.048	6	1.753	1.753	5.959	6.037	1.601	0.798
1.05	25.018	6	1.603	1.603	5.604	5.677	1.868	0.931
1.20	24.952	6	1.453	1.453	5.303	5.372	2.135	1.064
1.35	24.921	6	1.303	1.303	5.002	5.067	2.402	1.197
1.50	24.697	6	1.153	1.153	4.700	4.762	2.669	1.330

Table A.8.: Inverting amplifier ε̂SVR4 models trained with the SMO solver.

(a) Linear kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	2313.080	175	0.362	0.363	1.837	1.889	1.356	0.831
0.15	2122.242	162	2.907	2.922	1.486	1.593	1.887	1.097
0.30	1723.244	129	0.437	0.439	1.617	1.692	2.056	1.149
0.45	3693.044	128	0.432	0.435	2.600	2.630	2.278	1.239
0.60	5543.656	115	0.894	0.902	1.980	1.990	1.817	0.940
0.75	8858.616	98	0.936	0.945	3.963	3.991	1.912	0.974
0.90	2544.609	81	1.009	1.019	4.202	4.240	1.980	0.996
1.05	6797.498	64	1.181	1.193	4.679	4.735	1.935	0.966
1.20	4427.342	47	0.940	0.951	3.183	3.213	2.334	1.167
1.35	2601.076	31	0.584	0.594	3.228	3.251	2.834	1.422
1.50	632.906	21	0.685	0.696	3.625	3.668	2.722	1.357

(b) RBF kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	2819.798	301	0.000	0.000	2.189	2.194	3.885	1.912
0.15	47212.329	63	0.000	0.000	1.785	1.743	3.160	1.519
0.30	108.280	35	0.359	0.372	2.558	2.476	2.477	1.180
0.45	48381.961	22	0.455	0.461	1.929	1.805	2.625	1.170
0.60	48204.843	18	0.618	0.624	1.971	1.936	2.376	1.140
0.75	131.732	15	0.762	0.762	3.689	3.642	2.456	1.148
0.90	45.821	7	1.412	1.412	5.897	5.912	3.005	1.400
1.05	44.996	7	1.262	1.262	5.595	5.607	3.273	1.533
1.20	45.837	7	1.112	1.112	5.293	5.302	3.545	1.666
1.35	45.509	7	0.962	0.962	4.992	4.997	3.811	1.799
1.50	45.894	7	0.812	0.812	4.690	4.692	4.078	1.932

Table A.9.: Inverting amplifier ε̂SVR6 models trained with the SMO solver.

(a) Linear kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	3356.961	137	1.911	1.921	1.593	1.710	1.694	1.038
0.15	3382.514	131	1.264	1.264	2.057	2.085	1.464	0.839
0.30	2982.000	120	1.118	1.118	2.521	2.533	1.235	0.659
0.45	3362.397	111	3.400	3.400	2.532	2.546	1.355	0.700
0.60	3157.383	86	1.811	1.823	2.863	2.878	1.791	0.925
0.75	3367.764	74	2.723	2.741	2.929	2.943	2.096	1.075
0.90	3477.080	81	0.709	0.717	2.512	2.529	2.763	1.431
1.05	1407.597	46	1.524	1.537	6.094	6.146	2.529	1.281
1.20	4230.705	51	0.303	0.311	2.255	2.276	3.344	1.715
1.35	2832.872	7	0.698	0.708	2.643	2.656	3.342	1.697
1.50	2703.262	8	0.094	0.102	2.323	2.341	3.846	1.958

(b) RBF kernels.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.00	10748.438	301	0.000	0.000	2.197	2.208	3.899	1.925
0.15	35806.283	96	0.000	0.000	1.864	1.831	3.299	1.596
0.30	3897.379	30	0.439	0.440	1.700	1.599	2.090	1.005
0.45	3611.629	25	0.610	0.626	1.875	1.843	2.122	1.066
0.60	23922.556	19	0.973	0.973	2.192	2.164	2.144	1.024
0.75	3301.648	16	1.103	1.103	2.380	2.384	2.243	1.101
0.90	2767.277	4	1.408	1.408	3.024	3.010	2.946	1.376
1.05	2792.945	4	1.258	1.258	3.024	3.010	3.212	1.509
1.20	2807.070	7	0.966	0.968	2.595	2.559	2.842	1.375
1.35	2543.629	4	0.958	0.958	3.024	3.010	3.750	1.775
1.50	2501.438	4	0.808	0.808	3.024	3.010	4.018	1.908

Table A.10.: Inverting amplifier νSVR models trained with the SMO solver.

(a) Linear kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	1149.853	139	2.81	2.824	2.547	2.58	0.0	0.0
0.2	1149.150	139	2.81	2.824	2.547	2.58	0.0	0.0
0.3	1138.419	139	2.81	2.824	2.547	2.58	0.0	0.0
0.4	1157.486	139	2.81	2.824	2.547	2.58	0.0	0.0
0.5	1143.836	139	2.81	2.824	2.547	2.58	0.0	0.0
0.6	1140.483	139	2.81	2.824	2.547	2.58	0.0	0.0
0.7	1149.941	139	2.81	2.824	2.547	2.58	0.0	0.0
0.8	1147.962	139	2.81	2.824	2.547	2.58	0.0	0.0
0.9	1144.473	139	2.81	2.824	2.547	2.58	0.0	0.0
1.0	1146.399	139	2.81	2.824	2.547	2.58	0.0	0.0

(b) RBF kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	341.101	257	2.359	2.381	7.148	7.242	0.0	0.0
0.2	338.187	257	2.359	2.381	7.148	7.242	0.0	0.0
0.3	341.330	257	2.359	2.381	7.148	7.242	0.0	0.0
0.4	339.480	257	2.359	2.381	7.148	7.242	0.0	0.0
0.5	341.123	257	2.359	2.381	7.148	7.242	0.0	0.0
0.6	338.339	257	2.359	2.381	7.148	7.242	0.0	0.0
0.7	345.023	257	2.359	2.381	7.148	7.242	0.0	0.0
0.8	340.830	257	2.359	2.381	7.148	7.242	0.0	0.0
0.9	341.378	257	2.359	2.381	7.148	7.242	0.0	0.0
1.0	340.431	257	2.359	2.381	7.148	7.242	0.0	0.0

Table A.11.: Inverting amplifier ν̂SVR4 models trained with the SMO solver.

(a) Linear kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	11320.919	51	1.409	1.417	0.709	0.765	1.821	0.953
0.2	25865.832	208	0.335	0.335	1.307	1.444	1.648	1.006
0.3	33962.535	280	0.086	0.088	1.922	2.004	3.285	1.747
0.4	45515.602	300	1.067	1.072	1.392	1.473	1.641	0.952
0.5	66676.825	300	1.254	1.254	1.597	1.647	1.384	0.817
0.6	66908.327	300	1.254	1.254	1.597	1.647	1.384	0.817
0.7	66109.814	300	1.254	1.254	1.597	1.647	1.384	0.817
0.8	67669.059	300	1.254	1.254	1.597	1.647	1.384	0.817
0.9	67229.627	300	1.254	1.254	1.597	1.647	1.384	0.817
1.0	67701.318	300	1.254	1.254	1.597	1.647	1.384	0.817

(b) RBF kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	1395.444	58	3.009	3.010	8.085	8.110	6.637	3.424
0.2	4043.718	130	3.220	3.220	8.294	8.319	6.701	3.369
0.3	6495.287	178	0.148	0.166	1.531	1.530	2.293	1.127
0.4	10168.727	198	0.132	0.140	1.612	1.610	2.366	1.152
0.5	14680.257	219	0.044	0.046	1.505	1.506	2.571	1.255
0.6	14931.892	219	0.044	0.046	1.505	1.506	2.571	1.255
0.7	14811.167	219	0.044	0.046	1.505	1.506	2.571	1.255
0.8	14715.273	219	0.044	0.046	1.505	1.506	2.571	1.255
0.9	14922.307	219	0.044	0.046	1.505	1.506	2.571	1.255
1.0	14884.491	219	0.044	0.046	1.505	1.506	2.571	1.255

Table A.12.: Inverting amplifier ŷSVR6 models trained with the SMO solver.

(a) Linear kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	16923.135	146	1.177	1.188	0.483	0.516	2.375	1.202
0.2	42890.786	244	1.541	1.541	0.700	0.801	2.246	1.178
0.3	69740.263	297	1.096	1.105	0.739	0.903	2.354	1.258
0.4	84527.863	301	1.368	1.376	1.322	1.409	1.877	1.054
0.5	107442.328	301	2.600	2.614	2.542	2.569	0.386	0.197
0.6	109225.551	301	2.600	2.614	2.542	2.569	0.386	0.197
0.7	109727.976	301	2.600	2.614	2.542	2.569	0.386	0.197
0.8	110580.835	301	2.600	2.614	2.542	2.569	0.386	0.197
0.9	109780.658	301	2.600	2.614	2.542	2.569	0.386	0.197
1.0	111487.404	301	2.600	2.614	2.542	2.569	0.386	0.197

(b) RBF kernels.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	3381.126	158	0.345	0.347	1.763	1.767	1.927	0.932
0.2	7711.609	223	0.259	0.276	1.742	1.734	2.067	1.009
0.3	17981.850	270	0.136	0.154	1.578	1.565	2.329	1.144
0.4	26134.396	291	0.072	0.075	1.495	1.476	2.462	1.202
0.5	51194.106	298	0.000	0.000	1.535	1.509	2.688	1.315
0.6	52291.515	298	0.000	0.000	1.535	1.509	2.688	1.315
0.7	52581.182	298	0.000	0.000	1.535	1.509	2.688	1.315
0.8	51703.803	298	0.000	0.000	1.535	1.509	2.688	1.315
0.9	51851.606	298	0.000	0.000	1.535	1.509	2.688	1.315
1.0	52111.031	298	0.000	0.000	1.535	1.509	2.688	1.315

A.2. Class A Amplifier

Table A.13.: Class A amplifier εSVR models created with the CVX solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	25.133	31	1.481	1.571	3.869	6.535	NaN	0.000
0.035	10.168	18	1.421	1.511	3.804	6.425	0.091	0.077
0.070	9.804	15	1.357	1.446	3.727	6.295	0.183	0.154
0.105	9.892	12	1.299	1.388	3.784	6.390	0.274	0.232
0.140	9.757	10	1.238	1.327	3.495	5.902	0.366	0.309
0.175	9.902	10	1.163	1.252	3.505	5.920	0.457	0.386
0.210	9.786	9	1.131	1.221	3.504	5.918	0.549	0.463
0.245	9.681	8	1.123	1.212	3.572	6.033	0.640	0.540
0.280	9.881	6	1.117	1.207	3.649	6.163	0.731	0.618
0.315	9.735	5	1.129	1.219	3.772	6.371	0.823	0.695
0.350	9.863	5	1.125	1.215	3.854	6.508	0.914	0.772

Table A.14.: Class A amplifier ε̂SVR4 models created with the CVX solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	11.269	31	0.838	0.931	3.040	4.845	0.928	0.572
0.035	10.949	25	0.803	0.895	3.043	4.818	1.014	0.637
0.070	10.962	20	0.718	0.810	3.090	4.541	1.117	0.684
0.105	11.197	17	0.593	0.684	2.977	4.122	1.314	0.754
0.140	10.943	17	0.515	0.608	2.881	3.933	1.507	0.833
0.175	11.075	13	0.431	0.524	2.807	3.759	1.630	0.930
0.210	11.331	12	0.436	0.527	2.912	4.029	1.753	1.056
0.245	11.013	10	0.433	0.525	2.974	4.256	1.870	1.174
0.280	10.953	10	0.432	0.524	3.036	4.478	1.906	1.287
0.315	11.093	8	0.388	0.480	2.969	4.479	2.001	1.387
0.350	11.207	8	0.283	0.375	2.788	4.253	2.133	1.507

Table A.15.: Class A amplifier ε̂SVR6 models created with the CVX solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	13.128	31	0.695	0.785	2.703	4.324	1.098	0.630
0.035	13.064	24	0.659	0.749	2.705	4.321	1.166	0.707
0.070	13.045	21	0.557	0.647	2.642	3.947	1.222	0.745
0.105	13.146	20	0.479	0.570	2.933	4.110	1.316	0.775
0.140	13.218	15	0.446	0.537	2.759	3.711	1.518	0.863
0.175	12.993	12	0.422	0.514	2.787	3.747	1.623	0.939
0.210	13.311	10	0.424	0.516	2.836	3.921	1.711	1.028
0.245	13.304	11	0.340	0.431	2.708	3.743	1.796	1.126
0.280	13.110	9	0.261	0.353	2.558	3.584	1.885	1.220
0.315	13.379	9	0.242	0.334	2.543	3.680	1.924	1.308
0.350	13.235	10	0.187	0.232	2.468	3.661	2.042	1.420

Table A.16.: Class A amplifier νSVR models created with the CVX solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	9.810	5	1.125	1.215	3.854	6.508	0.914	0.772
0.2	9.899	5	1.125	1.215	3.854	6.508	0.914	0.772
0.3	10.009	5	1.125	1.215	3.854	6.508	0.914	0.772
0.4	9.808	5	1.125	1.215	3.854	6.508	0.914	0.772
0.5	9.780	5	1.125	1.215	3.854	6.508	0.914	0.772
0.6	9.788	5	1.125	1.215	3.854	6.508	0.914	0.772
0.7	9.809	5	1.125	1.215	3.854	6.508	0.914	0.772
0.8	9.875	5	1.125	1.215	3.854	6.508	0.914	0.772
0.9	9.766	5	1.125	1.215	3.854	6.508	0.914	0.772
1.0	9.804	5	1.125	1.215	3.854	6.508	0.914	0.772

Table A.17.: Class A amplifier ν̂SVR4 models created with the CVX solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	11.372	17	0.686	0.778	2.993	4.058	0.987	0.516
0.2	11.456	23	0.830	0.922	3.035	4.784	0.939	0.562
0.3	11.400	29	0.831	0.924	3.055	4.790	0.944	0.563
0.4	11.503	31	0.836	0.929	3.038	4.828	0.925	0.571
0.5	11.405	31	0.838	0.931	3.041	4.840	0.928	0.572
0.6	11.231	31	0.838	0.931	3.041	4.840	0.928	0.572
0.7	11.403	31	0.838	0.931	3.041	4.840	0.928	0.572
0.8	11.469	31	0.838	0.931	3.041	4.840	0.928	0.572
0.9	11.420	31	0.838	0.931	3.041	4.840	0.928	0.572
1.0	11.485	31	0.838	0.931	3.041	4.840	0.928	0.572

Table A.18.: Class A amplifier ν̂SVR6 models created with the CVX solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	13.784	22	0.584	0.675	2.681	3.662	1.022	0.543
0.2	13.802	27	0.688	0.778	2.713	4.289	1.105	0.628
0.3	13.779	31	0.678	0.768	2.712	4.229	1.079	0.620
0.4	13.681	31	0.689	0.780	2.691	4.297	1.097	0.629
0.5	13.561	31	0.695	0.785	2.704	4.323	1.098	0.630
0.6	13.592	31	0.695	0.785	2.704	4.323	1.098	0.630
0.7	13.671	31	0.695	0.785	2.704	4.323	1.098	0.630
0.8	13.582	31	0.695	0.785	2.704	4.323	1.098	0.630
0.9	13.570	31	0.695	0.785	2.704	4.323	1.098	0.630
1.0	13.640	31	0.695	0.785	2.704	4.323	1.098	0.630

Table A.19.: Class A amplifier εSVR models created with the SMO solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	28.400	28	1.485	1.575	4.913	7.796	NaN	0.000
0.035	18621.356	19	1.446	1.536	4.903	7.782	0.091	0.077
0.070	18882.784	16	1.373	1.462	4.794	7.614	0.183	0.154
0.105	19199.261	14	1.373	1.462	4.895	7.770	0.274	0.232
0.140	18890.565	14	1.341	1.431	4.902	7.780	0.366	0.309
0.175	18630.890	12	1.312	1.400	4.976	7.894	0.457	0.386
0.210	19047.435	12	1.248	1.336	4.836	7.678	0.549	0.463
0.245	42.286	12	1.295	1.383	5.071	8.035	0.640	0.540
0.280	10.267	6	1.082	1.172	4.562	7.259	0.731	0.618
0.315	19250.386	12	1.190	1.280	4.972	7.887	0.823	0.695
0.350	30.572	6	0.991	1.080	4.501	7.163	0.914	0.772

Table A.20.: Class A amplifier ε̂SVR4 models created with the SMO solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	130.183	31	0.846	0.938	3.210	4.978	0.931	0.577
0.035	36558.218	26	0.796	0.888	3.165	4.914	1.026	0.634
0.070	35824.397	23	0.773	0.864	3.200	4.966	1.109	0.688
0.105	35629.246	17	0.611	0.701	2.842	4.403	1.263	0.774
0.140	35591.034	17	0.509	0.601	2.663	4.109	1.622	0.898
0.175	34623.524	14	0.430	0.522	2.525	3.916	1.707	0.886
0.210	13.316	12	0.376	0.469	2.464	3.835	1.848	1.183
0.245	13.005	8	0.385	0.478	2.605	4.029	1.896	1.271
0.280	12.804	8	0.317	0.408	2.499	3.882	2.008	1.447
0.315	12.499	10	0.337	0.430	2.676	4.125	2.056	1.493
0.350	12.390	8	0.278	0.368	2.596	4.014	2.166	1.656

Table A.21.: Class A amplifier ε̂SVR6 models created with the SMO solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	516.292	31	0.690	0.780	2.767	4.287	1.093	0.629
0.035	42459.570	27	0.635	0.725	2.720	4.204	1.251	0.709
0.070	9190.917	22	0.451	0.542	2.313	3.544	1.319	0.737
0.105	8870.806	19	0.364	0.454	2.207	3.315	1.587	0.832
0.140	9178.347	18	0.355	0.445	2.261	3.429	1.693	0.943
0.175	7418.083	12	0.526	0.617	2.801	4.339	1.679	0.972
0.210	7634.555	12	0.353	0.444	2.397	3.729	1.583	1.102
0.245	8094.898	13	0.211	0.303	2.182	3.259	2.130	1.324
0.280	7325.498	11	0.356	0.447	2.624	4.056	2.009	1.423
0.315	6623.903	9	0.348	0.348	2.705	4.172	1.915	1.371
0.350	7191.846	11	0.193	0.193	2.354	3.643	2.286	1.551

Table A.22.: Class A amplifier νSVR models created with the SMO solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	30.739	6	0.991	1.08	4.504	7.157	0.914	0.772
0.2	30.346	6	0.991	1.08	4.504	7.157	0.914	0.772
0.3	30.630	6	0.991	1.08	4.504	7.157	0.914	0.772
0.4	30.327	6	0.991	1.08	4.504	7.157	0.914	0.772
0.5	30.730	6	0.991	1.08	4.504	7.157	0.914	0.772
0.6	30.540	6	0.991	1.08	4.504	7.157	0.914	0.772
0.7	30.921	6	0.991	1.08	4.504	7.157	0.914	0.772
0.8	30.225	6	0.991	1.08	4.504	7.157	0.914	0.772
0.9	30.776	6	0.991	1.08	4.504	7.157	0.914	0.772
1.0	30.975	6	0.991	1.08	4.504	7.157	0.914	0.772

Table A.23.: Class A amplifier ν̂SVR4 models created with the SMO solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	36.697	11	1.832	1.922	6.031	9.667	3.123	3.034
0.2	88.969	17	1.868	1.957	6.016	9.644	2.973	2.935
0.3	127.838	21	0.401	0.493	2.598	4.009	1.766	1.345
0.4	180.600	23	0.566	0.658	2.739	4.231	1.296	0.855
0.5	231.588	26	0.785	0.877	3.017	4.693	1.021	0.657
0.6	235.259	26	0.785	0.877	3.048	4.737	1.021	0.657
0.7	230.707	26	0.785	0.877	3.048	4.737	1.021	0.657
0.8	233.738	26	0.785	0.877	3.048	4.737	1.021	0.657
0.9	232.205	26	0.785	0.877	3.048	4.737	1.021	0.657
1.0	232.962	26	0.785	0.877	3.048	4.737	1.021	0.657

Table A.24.: Class A amplifier ŷSVR6 models created with the SMO solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in V	e_{Test} in V	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	49.104	18	0.245	0.337	2.285	3.474	1.960	1.312
0.2	98.859	22	0.441	0.533	2.518	3.900	1.557	1.056
0.3	219.355	26	0.454	0.545	2.521	3.904	1.481	1.026
0.4	316.319	30	0.509	0.600	2.447	3.803	1.313	0.751
0.5	508.159	31	0.625	0.715	2.698	4.151	1.220	0.699
0.6	489.524	31	0.625	0.715	2.698	4.151	1.220	0.699
0.7	503.521	31	0.625	0.715	2.698	4.151	1.220	0.699
0.8	493.892	31	0.625	0.715	2.698	4.151	1.220	0.699
0.9	496.640	31	0.625	0.715	2.698	4.151	1.220	0.699
1.0	505.001	31	0.625	0.715	2.698	4.151	1.220	0.699

A.3. Diode

All models have been created using polynomial kernels.

Table A.25.: Diode εSVR models created with the CVX solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	10.890	18	5.578	7.083	6924859	40.277	NaN	0.000
0.005	9.444	18	5.581	6.992	6968906	40.534	132945	0.773
0.010	9.386	18	5.580	6.747	6862666	39.916	265890	1.547
0.015	9.847	18	5.515	6.719	6968168	40.529	398836	2.320
0.020	9.452	18	5.374	6.365	6754740	39.288	531781	3.093
0.025	9.528	18	5.253	5.983	6514174	37.889	664727	3.866
0.030	9.874	18	5.199	5.710	6379815	37.107	797672	4.640
0.035	9.544	18	5.147	5.424	6233055	36.254	930618	5.413
0.040	9.407	18	5.095	5.140	6088530	35.413	1063563	6.186
0.045	9.763	18	5.043	5.043	6127236	35.638	1196509	6.959
0.050	9.287	18	4.991	4.991	6209455	36.116	1329454	7.733

Table A.26.: Diode ε̂SVR4 models created with the CVX solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	9.889	18	2.704	3.671	3626963	22.487	28516	1.613
0.005	10.259	18	2.524	3.422	3550835	21.879	205008	2.418
0.010	9.858	18	2.038	3.019	3310575	20.485	358743	3.316
0.015	10.243	18	1.748	2.734	3108585	19.567	435161	4.017
0.020	10.340	18	1.446	2.440	3033586	18.739	647830	4.862
0.025	9.992	18	1.154	2.157	2884440	17.987	775764	5.723
0.030	10.315	18	0.885	1.897	2746605	17.253	892432	6.468
0.035	10.355	18	0.791	1.809	2732468	17.444	963974	7.158
0.040	10.383	18	0.610	1.634	2775894	17.311	1178377	8.020
0.045	10.021	18	0.466	1.493	2874773	17.363	1415371	8.875
0.050	10.194	18	0.098	1.084	2536357	16.185	1476137	10.018

Table A.27.: Diode ε̂SVR6 models created with the CVX solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	10.788	18	2.619	3.583	3576756	22.013	66648	1.639
0.005	10.840	18	2.474	3.370	3464093	21.527	169791	2.366
0.010	11.393	18	1.862	2.839	3085551	19.385	309871	3.240
0.015	10.894	18	1.348	2.336	2894174	17.569	610603	4.287
0.020	10.773	18	1.148	2.145	2745039	17.120	648312	4.925
0.025	11.331	18	1.034	2.038	2681032	17.207	688079	5.615
0.030	10.926	18	0.956	1.968	2875984	17.809	951957	6.618
0.035	10.850	18	0.598	1.621	2632948	16.533	1047812	7.313
0.040	11.365	18	0.483	1.511	2715221	16.745	1237746	8.151
0.045	10.846	18	0.544	1.572	3172444	18.156	1635149	9.214
0.050	11.027	18	0.349	1.267	2920846	17.016	1682075	9.811

Table A.28.: Diode νSVR models created with the CVX solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	9.701	18	4.991	4.991	6209455	36.116	1329454	7.733
0.2	9.473	18	4.991	4.991	6209455	36.116	1329454	7.733
0.3	9.712	18	4.991	4.991	6209455	36.116	1329454	7.733
0.4	9.450	18	4.991	4.991	6209455	36.116	1329454	7.733
0.5	9.446	18	4.991	4.991	6209455	36.116	1329454	7.733
0.6	9.750	18	4.991	4.991	6209455	36.116	1329454	7.733
0.7	10.018	18	4.991	4.991	6209455	36.116	1329454	7.733
0.8	9.585	18	4.991	4.991	6209455	36.116	1329454	7.733
0.9	10.020	18	4.991	4.991	6209455	36.116	1329454	7.733
1.0	9.384	18	4.991	4.991	6209455	36.116	1329454	7.733

Table A.29.: Diode ŷSVR4 models created with the CVX solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	10.648	18	1.815	2.819	2866314	17.887	106078	1.856
0.2	10.290	18	2.661	3.545	3537874	21.806	62014	1.646
0.3	10.337	18	2.691	3.631	3578126	22.289	21314	1.641
0.4	10.175	18	2.704	3.671	3626968	22.487	28512	1.613
0.5	10.157	18	2.704	3.671	3626964	22.487	28517	1.613
0.6	10.266	18	2.704	3.671	3626973	22.487	28518	1.613
0.7	10.470	18	2.704	3.671	3626963	22.487	28516	1.613
0.8	10.313	18	2.704	3.671	3626962	22.487	28516	1.613
0.9	10.272	18	2.704	3.671	3626962	22.487	28516	1.613
1.0	10.694	18	2.704	3.671	3626963	22.487	28516	1.613

Table A.30.: Diode ŷSVR6 models created with the CVX solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	11.229	18	1.701	2.705	2648893	17.112	4341	1.730
0.2	11.546	18	2.606	3.486	3443386	21.433	26944	1.613
0.3	11.216	18	2.655	3.601	3534078	22.064	12922	1.589
0.4	11.079	18	2.632	3.590	3575905	22.041	58517	1.624
0.5	11.403	18	2.619	3.583	3576756	22.013	66658	1.639
0.6	10.944	18	2.619	3.583	3576761	22.013	67001	1.639
0.7	11.167	18	2.619	3.583	3576758	22.013	66854	1.639
0.8	11.582	18	2.619	3.583	3576762	22.013	67039	1.639
0.9	11.079	18	2.619	3.583	3576758	22.013	66826	1.639
1.0	11.176	18	2.619	3.583	3576761	22.013	66976	1.639

Table A.31.: Diode εSVR models created with the SMO solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	1742.677	16	7.802	7.802	7627966	44.367	NaN	0.000
0.005	1384.832	13	13.591	14.434	14244815	82.853	132945	0.773
0.010	1441.301	13	5.707	6.779	6893899	40.097	265890	1.547
0.015	2081.033	13	5.222	7.048	7289747	42.400	398836	2.320
0.020	2774.992	13	5.157	6.890	7267893	42.273	531781	3.093
0.025	2423.213	12	5.113	6.812	7324269	42.600	664727	3.866
0.030	2554.148	12	5.060	6.700	7347817	42.737	797672	4.640
0.035	2234.318	10	5.021	6.646	7428519	43.207	930618	5.413
0.040	2116.340	10	4.911	6.514	7432525	43.230	1063563	6.186
0.045	2002.916	9	5.176	6.880	7922969	46.083	1196509	6.959
0.050	1419.553	9	5.080	5.080	6295792	36.618	1329454	7.733

Table A.32.: Diode ĉSVR4 models created with the SMO solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	2826.215	16	2.003	4.003	4024281	24.128	NaN	1.366
0.005	2351.300	15	2.025	2.123	2402674	16.981	327107	4.909
0.010	2085.790	12	1.636	2.201	2682255	16.044	530266	3.527
0.015	2052.813	15	1.130	2.153	2551579	17.279	446685	5.036
0.020	2114.147	15	0.824	0.875	1397976	11.673	542165	6.695
0.025	2260.814	14	7.486	7.486	8054476	48.468	736089	5.902
0.030	2095.042	13	1.780	3.124	3943375	24.135	888954	6.369
0.035	2258.589	13	0.000	0.845	1947554	14.559	1121564	9.754
0.040	2033.453	13	0.000	0.270	1866776	13.458	1602347	11.920
0.045	2606.741	14	0.000	0.000	1945257	15.550	1945257	15.550
0.050	3860.840	12	0.720	1.996	4378530	24.396	2427413	13.047

Table A.33.: Diode ĉSVR6 models created with the SMO solver.

					adjusted		not adjusted	
ε	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.000	4467.333	15	8.605	8.605	8956977	51.297	NaN	2.366
0.005	4152.247	17	5.744	5.744	6411727	37.528	795973	4.865
0.010	3746.267	12	5.429	5.429	6136187	35.474	828558	4.603
0.015	4134.379	16	4.727	4.727	5689375	32.885	1067706	6.003
0.020	4341.665	12	1.353	2.836	3980492	22.508	1207917	6.382
0.025	4176.730	11	2.242	3.912	5142790	28.785	1318600	6.542
0.030	4388.949	14	0.850	1.137	2405251	14.289	1293493	7.822
0.035	4670.135	16	0.311	1.056	2478278	14.166	1446203	8.163
0.040	3626.911	13	3.456	5.675	7007009	40.387	1458650	8.116
0.045	285.038	12	3.237	5.385	6515768	39.546	1250901	8.924
0.050	2882.571	11	4.783	7.017	8669217	50.169	1808448	10.264

Table A.34.: Diode νSVR models created with the SMO solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	1439.645	9	5.08	5.08	6295792	36.618	1329454	7.733
0.2	1423.210	9	5.08	5.08	6295792	36.618	1329454	7.733
0.3	1433.214	9	5.08	5.08	6295792	36.618	1329454	7.733
0.4	1442.023	9	5.08	5.08	6295792	36.618	1329454	7.733
0.5	1436.027	9	5.08	5.08	6295792	36.618	1329454	7.733
0.6	1440.176	9	5.08	5.08	6295792	36.618	1329454	7.733
0.7	1443.778	9	5.08	5.08	6295792	36.618	1329454	7.733
0.8	1432.628	9	5.08	5.08	6295792	36.618	1329454	7.733
0.9	1441.119	9	5.08	5.08	6295792	36.618	1329454	7.733
1.0	1424.439	9	5.08	5.08	6295792	36.618	1329454	7.733

Table A.35.: Diode ν̂SVR4 models created with the SMO solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	2773.102	12	2.007	2.007	3968336	23.369	2006079	11.956
0.2	4172.945	17	0.116	0.116	1222499	8.944	1109555	8.287
0.3	6015.294	18	2.410	3.709	4616163	26.972	990274	5.883
0.4	7801.963	18	3.741	3.741	4614762	26.716	957445	5.444
0.5	9229.548	18	5.405	5.405	5855132	35.473	570390	4.735
0.6	9294.388	18	5.405	5.405	5855132	35.473	570390	4.735
0.7	9176.258	18	5.405	5.405	5855132	35.473	570390	4.735
0.8	9147.337	18	5.405	5.405	5855132	35.473	570390	4.735
0.9	9151.134	18	5.405	5.405	5855132	35.473	570390	4.735
1.0	9218.937	18	5.405	5.405	5855132	35.473	570390	4.735

Table A.36.: Diode ŷSVR6 models created with the SMO solver.

					adjusted		not adjusted	
ν	Runtime in s	SV	e_{Train} in A	e_{Test} in A	m_{ov}	A_{ov}	m_{ov}	A_{ov}
0.1	5196.824	15	0.424	1.885	3078518	19.207	1235877	8.490
0.2	8673.961	18	2.390	2.390	3689008	21.038	1352269	7.447
0.3	10541.161	18	0.319	1.092	2317742	14.808	1249990	8.597
0.4	13676.213	18	4.044	4.044	4384226	26.115	430340	3.117
0.5	14656.770	18	2.027	4.077	4428558	26.599	442197	3.413
0.6	14366.720	18	2.027	4.077	4428558	26.599	442197	3.413
0.7	14476.337	18	2.027	4.077	4428558	26.599	442197	3.413
0.8	14400.468	18	2.027	4.077	4428558	26.599	442197	3.413
0.9	14570.998	18	2.027	4.077	4428558	26.599	442197	3.413
1.0	14319.017	18	2.027	4.077	4428558	26.599	442197	3.413

B. Bibliography

[1] H. Agarwal, S. Khandelwal, J. P. Duarte, Y. S. Chauhan, A. Niknejad, and C. Hu. *BSIM6.1.0 MOSFET Compact Model Technical Manual*. Department of Electrical Engineering and Computer Sciences University of California, Berkeley, CA 94720, 2014.

[2] N. Arora. *MOSFET Modeling for VLSI Simulation Theory and Practice*. World Scientific, 2007.

[3] A. Barghouti, A. Krause, C. Carta, J. C. Scheytt, and F. Ellinger. Design and Characterization of a V-Band Quadrature VCO Based on a Common-Collector SiGe Colpitts VCO. In *Compound Semiconductor Integrated Circuit Symposium (CSICS), 2010 IEEE*, pages 1–3, 10 2010.

[4] A. Bauer and W. Schwarz. Circuit analysis and optimization with automatically derived volterra kernels. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, Volume 1, pages 491–494, 2000.

[5] C. Borchers. *Automatische Generierung von Verhaltensmodellen für nichtlineare Analogschaltungen*. PhD thesis, Universität Hannover, January 1997. in German.

[6] C. Borchers, L. Hedrich, and E. Barke. Equation-based behavioral model generation for nonlinear analog circuits. In *Design Automation Conference Proceedings 1996, 33rd*, pages 236–239, Jun 1996.

[7] C. Borgelt, F. Klawonn, R. Kruse, and D. Nauck. *Neuro-Fuzzy-Systeme*. Vieweg, 2003.

[8] G. R. Boyle, D. O. Pederson, B. M. Cohn, and J. E. Solomon. Macromodeling of integrated circuit operational amplifiers. *IEEE Journal of Solid State Circuits*, 9(6):353–364, 1974.

[9] Cadence Design Systems. *Virtuoso Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*, June 2011.

[10] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[11] Y. Cheng and C. Hu. *MOSFET Modeling & BSIM3 User's Guide*. Kluwer Academic Publishers, 1999.

[12] J. A. Connelly and P. Choi. *Macromodeling with SPICE*. Prentice Hall, 1992.

[13] C. Cortes and V. N. Vapnik. Support vector networks. *Machine Learning*, 20(3):273–297, 1995.

[14] W. Daems, G. Gielen, and W. Sansen. Circuit complexity reduction for symbolic analysis of analog integrated circuits. In *Proceedings 36th Design Automation Conference*, pages 958–963, 1999.

[15] W. Daems, G. Gielen, and W. Sansen. An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits. In *Design Automation Conference, 2002. Proceedings. 39th*, pages 431–436, 2002.

[16] L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA/CNPq, Rio de Janeiro, Brazil, 1997.

[17] D. H. C. de Graaff and D. F. M. Klaassen. *Compact Transistor Modelling for Circuit Design*. Springer Verlag, 1990.

[18] F. Dorel and M. Declercq. A prototype tool for the design oriented symbolic analysis of analog circuits. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 12.5.1–12.5.4, May 1992.

[19] J. J. Ebers and J. L. Moll. Large-signal behavior of junction transistors. *Proceedings of the IRE*, 42(12):1761–1772, Dec 1954.

[20] F. Fernandez and A. Rodriguez-Vazquez. Symbolic analysis tools - the state of the art. In *Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE Symposium on*, Volume 4, pages 798–801, May 1996.

[21] D. Gajski and R. Kuhn. Guest Editors' Introduction: New VLSI Tools. *Computer*, 16(12):11–14, Dec 1983.

[22] H. Garnier, L. Wang, and P. C. Young. Direct identification of continuous-time models from sampled data: Issues, basic solutions and relevance. In H. Garnier and L. Wang, editors, *Identification of Continuous-time Models from Sampled Data*, Advances in Industrial Control, Chapter 1. Springer, 2008.

[23] I. E. Getreu. *Modeling the Bipolar Transistor*, Volume 1 of *Computer-Aided Design of Electronic Circuits*. Elsevier Scientific Publishing Company, 1978.

[24] G. Gielen and W. Sansen. *Symbolic Analysis for Automated Design of Analog Integrated Circuits*. Kluwer Academic Publishers, 1991.

[25] D. Grabowski. *Gebietsarithmetische Verfahren zur Simulation analoger Schaltungen mit Parameterunsicherheiten*. PhD thesis, Gottfried Wilhelm Leibniz Universität Hannover, 2009.

[26] D. Grabowski, M. Olbrich, and E. Barke. Analog circuit simulation using range arithmetics. In *Proceedings of the ASP-DAC 2008*, pages 762–767. ASP DAC, 2008.

[27] H. E. Graeb. *Analog Design Centering and Sizing*. Springer, 2007.

[28] C. Graham and D. Talay. *Stochastic Simulation and Monte Carlo Methods*. Stochastic Modelling and Applied Probability. Springer, 2013.

[29] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. http://stanford.edu/ boyd/graph_dcp.html.

[30] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. http://cvxr.com/cvx, Sept. 2013.

[31] H. K. Gummel and H. Poon. An integral charge control model of bipolar transistors. *Bell System Technical Journal*, 49:827–852, May 1970.

[32] M. T. Hagan, H. B. Demuth, and M. Beale. *Neural Network Design*. PWS Publishing Company, 1996.

[33] T. Halfmann and T. Wichmann. Overview of symbolic methods in industrial analog circuit design. Technical report, Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM, 2003.

[34] E. R. Hansen. A generalized interval arithmetic. In K. Nickel, editor, *Interval Mathematics*, Volume 29, pages 7 – 18. Springer, 1975.

[35] B. Hosticka, W. Brockherde, R. Klinke, and R. Kokozinski. Design methodology for analog monolithic circuits. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 41(5):387–394, May 1994.

[36] S. A. Huss. *Model Engineering in Mixed-Signal Circuit Design: A Guide to Generating Accurate Behavioural Models in VHDL-AMS*. Kluwer Academic Publishers, 2001.

[37] C. Jacoboni and P. Lugli. *The Monte Carlo Method for Semiconductor Device Simulation*. Springer-Verlag, 1989.

[38] A. Krause, M. Olbrich, and E. Barke. Enclosing the Modeling Error in Analog Behavioral Models Using Neural Networks and Affine Arithmetic. In *Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD), 2012 International Conference on*, pages 5–8, 09 2012.

[39] A. Krause, M. Olbrich, and E. Barke. Intervallwertige Support Vector Machines zur Verhaltensmodellierung analoger Schaltungen mit Parametervariationen. In *ANALOG 2014 (GMM-FB 80), Analogschaltungen im Systemkontext, Beiträge der 14. GMM/ITG-Fachtagung*, 09 2014.

[40] A. Krause, M. Olbrich, and E. Barke. Variation-aware Behavioural Models of Analog Circuits Using Support Vector Machines with Interval Parameters. In *Computer Science and Electronic Engineering Conference (CEEC), 2014 6th*, pages 121–126, 09 2014.

[41] J. H. Lilly. *Fuzzy Control and Identification*. John Wiley & Sons, Inc., 2010.

[42] W. Liu, X. Jin, X. Xi, J. Chen, M.-C. Jeng, Z. Liu, Y. Cheng, K. Chen, M. Chan, K. Hui, J. Huang, R. Tu, P. K. Ko, and C. Hu. *BSIM3v3.3 MOSFET Model Users' Manual*. Department of Electrical Engineering and Computer Sciences University of California, Berkeley, CA 94720, 2005.

[43] L. Ljung. *System Identification — Theory for the User*. Prentice Hall, 1999.

[44] E. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Human-Computer Studies*, 51(2):135–147, 1999.

[45] A. Manthe, Z. Li, C.-J. Shi, and K. Mayaram. Symbolic analysis of nonlinear analog circuits. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 1108–1109, 2003.

[46] H. A. Mantooth and M. Fiegenbaum. *Modeling with an Analog Hardware Description Language*. Analog Circuits and Signal Processing. Kluwer Academic Publishers, 1995.

[47] G. Massobrio and P. Antognetti. *Semiconductor Device Modeling with SPICE*. McGraw-Hill, 1993.

[48] W. Mayeda and S. Seshu. Topological formulas for network functions. *University of Illinois Bulletin*, 55(23), Nov. 1957.

[49] K. McCarville and M. Hassoun. An element stamp implementation of active device models is for symbolic simulation. In *Circuits and Systems, 1994., Proceedings of the 37th Midwest Symposium on*, Volume 1, pages 712–715, Aug 1994.

[50] Mentor Graphics Corporation. *ADiT^{TM} User's and Reference Manual*, Release 13.2a edition, 2014.

[51] N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, September 1949.

[52] H. Mielenz. *Verhaltensmodellierung in der Kraftfahrzeugtechnik mittels datenbasierter Methoden*. PhD thesis, Eberhard-Karls-Universität Tübingen, 2009.

[53] H. Mielenz, R. Doelling, and W. Rosenstiel. A method for semi-automated modeling of analog-mixed signal systems in automotive applications based on transient simulation data. In *Technical Proceedings of the 2007 NSTI Nanotechnology Conference and Trade Show*, Volume 3, pages 89–92, 2007.

[54] L. W. Nagel and D. Pederson. Spice (simulation program with integrated circuit emphasis). Technical Report UCB/ERL M382, EECS Department, University of California, Berkeley, Apr 1973.

[55] L. Näthke. *Ansätze zur automatischen Generierung hierarchischer Verhaltensmodelle von nichtlinearen integrierten Analogschaltungen*. PhD thesis, Universität Hannover, 2004.

[56] O. Nelles. *Nonlinear System Identification*. Springer, 2001.

[57] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.

[58] A. Pacelli, M. Mastrapasqua, and S. Luryi. Generation of equivalent circuit models from physics based device simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(11):1241–1250, November 2000.

[59] N. Paydavosi, T. H. Morshed, D. D. Lu, W. M. Yang, M. V. Dunga, X. J. Xi, J. He, W. Liu, Kanyu, M. Cao, X. Jin, J. J. Ou, M. Chan, A. M. Niknejad, and C. Hu. *BSIM4v4.8.0 MOSFET Model - User's Manual*. Department of Electrical Engineering and Computer Sciences University of California, Berkeley, CA 94720, 2013.

[60] W. Pedrycz. *Fuzzy Control and Fuzzy Systems*. Control Theory and Applications Series. John Wiley & Sons, Inc., 1993.

[61] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, Chapter 12, pages 185–208. MIT Press, 1999.

[62] M. Rewieński. A perspective on fast-spice simulation technology. In P. Li, P. Feldmann, and L. M. Sliveira, editors, *Simulation and Verification of Electronic and Biological Systems*, pages 23–39. Springer, 2011.

[63] O. Scharf, M. Olbrich, and E. Barke. Anwendung der affinen Arithmetik auf das BSIMSOI-Modell zur Simulation von Parameterschwankungen. In *ANALOG '11*, pages 49–54, 2011.

[64] B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, 2002.

[65] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. Bartlett. New support vector algorithms. *Neural Computation*, 12(5):1207–1245, May 2000.

[66] P. Senger. *Datenbasierte Methode zur automatischen Generierung elektrischer Verhaltensmodelle in der Automobilindustrie*. PhD thesis, Eberhard-Karls-Universität Tübingen, 2011.

[67] P. Senger, R. Doelling, and W. Rosenstiel. Automated electrical behavioural modelling of analogue and mixed signal circuits using data-based methods. In R. Alhajj, V. Leung, and R. Thring, editors, *Proceeding of IASTED Technology Conferences: Modelling and Simulation (696)*, July 2010.

[68] H. Shichman and D. Hodges. Modeling and simulation of insulated-gate field-effect transistor switching circuits. *IEEE Journal of Solid-State Circuits*, 3(3):285–289, Sep 1968.

[69] T. Shima, H. a. R. L. Tamada, and M. Dang. Table look-up MOSFET modeling system using a 2-D device simulator and monotonic piecewise cubic interpolation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(2):121–126, April 1983.

[70] J. A. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines*. World Scientific Publishing, 2002.

[71] Synopsys. *Saber User Guide*, h-2012.12 edition, December 2012.

[72] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modelling and control. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-15(1):116–132, Jan 1985.

[73] A. Tikhonov and V. Arsenin. *Solutions of ill-posed problems*. Scripta series in mathematics. Winston, 1977.

[74] Y. Tsividis. *Operation and Modeling of the MOS Transistor*. McGraw Hill, 1999.

[75] R. Unbehauen. *Systemtheorie 1 Allgemeine Grundlagen, Signale und lineare Systeme im Zeit- und Frequenzbereich*. R. Oldenbourg Verlag, 1997.

[76] V. N. Vapnik. *Statistical Learning Theory*. Wiley Series on Adaptive and Learning Systems for Signal Processing, Communications and Control. John Wiley & Sons, Inc., 1998.

[77] A. Vladimirescu. *The SPICE Book*. John Wiley & Sons, Inc., 1994.

[78] R. F. Vogel. Analytical MOSFET model with easily extracted parameters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(2):127–134, April 1985.

[79] P. Wambacq, V. Fernandez, G. Gielen, W. Sansen, and A. Rodriguez-Vazquez. Algorithm for efficient symbolic network analysis of large analogue circuits. *Electronics Letters*, 30:1108–1109, 1994.

[80] P. Wambacq, G. Gielen, and W. Sansen. Interactive symbolic distortion analysis of analogue integrated circuits. In *Design Automation. EDAC., Proceedings of the European Conference on*, pages 484–488, Feb 1991.

[81] P. Wambacq, G. Gielen, and W. Sansen. A new reliable approximation method for expanded symbolic network functions. In *Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on*, Volume 4, pages 584–587, May 1996.

[82] L. Wang. *Identification of Continuous-Time Models from Sampled Date*. Springer, 2008.

[83] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Curriculum Vitae

Personal Information

Name	Anna Krause
Date of Birth	10th July 1983
Place of Birth	Salzgitter
Nationality	German

Education

1990 – 1994	Primary School, Beilstein
1994 – 2003	Secondary School, Albert-Schweitzer-Gymnasium, Neckarsulm
2003 – 2009	University Education, Electrical Engineering, Technische Universität Dresden

Professional Experience

2009 – 2015	Research Engineer, Leibniz Universität Hannover
since 2016	Research Engineer, Robert Bosch GmbH