UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Victor Hugo Bueno Preuss

A Software Package for the Steady-State Simulation of Autonomous Circuits using the Harmonic Balance Method

Florianópolis
2021

Victor Hugo Bueno Preuss

A Software Package for the Steady-State Simulation of Autonomous Circuits using the Harmonic Balance Method

Dissertation submitted to the Electrical Engineering Postgraduate Program of the Federal University of Santa Catarina in order to obtain the degree of Master of Electrical Engineering.
Supervisor: Prof. Fernando Rangel de Sousa, PhD.

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Preuss, Victor Hugo Bueno
A software package for the steady-state simulation of autonomous circuits using the Harmonic Balance method / Victor Hugo Bueno Preuss ; orientador, Fernando Rangel de Sousa, 2021.
124 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, 2021.

Inclui referências.

1. Engenharia Elétrica. 2. Nonlinear circuit simulation. 3. Harmonic balance. 4. Oscillator analysis. 5. Periodic steady-state. I. Sousa, Fernando Rangel de . II. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia Elétrica. III. Título.

Victor Hugo Bueno Preuss

A Software Package for the Steady-State Simulation of Autonomous Circuits using the Harmonic Balance Method

Florianópolis, 2021.

I dedicate this work to my brothers and sisters and to my beloved parents.

ACKNOWLEDGEMENTS

I would like to thank my family for their immense support throughout this journey. My mother Glória, who went through huge efforts so that me, my brothers and sisters could have the best education possible. My father, Leoberto, who greatly instigated in me the love for the hard sciences. My siblings Sara, Leoberto, Pedro and Luiza with their constant support and encouragement.

My most deep appreciation to my girlfriend, Paula, whose invaluable help and love during these difficult times were fundamental for the completion of this work.

Also, to my advisor, Prof. Fernando Rangel de Sousa, who accepted three years ago under his guidance an inexperienced firmware developer with little to no understanding of the vast and beautiful world of radiofrequency circuits. This period enjoying his great knowledge of electronics and also his friendship, have completely changed the course of my career and future, and for that I am truly grateful.

"People claiming that anything is impossible
are often interrupted by those doing it."
Quote from 'Radio Frequency Integrated Circuit Design' by J. Rogers and C. Plett

ABSTRACT

The main goal of this work is the development of an open-source software package for steady-state simulation of autonomous circuits using the Harmonic Balance method. Oscillators are autonomous circuits of great interest in radiofrequency applications and a main component of transceivers. The periodic steady-state response of an oscillator is very important to designers, to verify parameters such as oscillating frequency, output power and power consumption. Transient simulations are not efficient to evaluate the periodic steady-state of an oscillator, as a large amount of computation is wasted during the initial startup period. The alternative explored in this work is the usage of the Harmonic Balance method with the Auxiliary Generator Technique to solve directly for the steady-state response of oscillators. To achieve that, a full circuit simulation engine was implemented in the Python programming language, with support to DC, AC, transient and harmonic balance analysis. The circuit netlists are described in code using a simple API. Several device models are available for simulation, such as RLC elements, current and voltage sources, Diode, BJT and MOSFET. Multiple examples are presented and the simulation results are compared to commercial engines to validate the implementations. Advantages of simulating circuits inside a Python environment are presented, involving easiness of data-processing and integration with other libraries.

Keywords: Nonlinear circuit simulation. Harmonic balance. Oscillator analysis. Periodic steady-state.

RESUMO

O objetivo principal deste trabalho foi o desenvolvimento de um programa de código-aberto para simulação do regime permanente periódico de circuitos autônomos utilizando o método do Balanço Harmônico. Osciladores são circuitos autônomos de grande interesse em aplicações de radiofrequência e são um componente primordial em transceptores. A resposta de regime permanente periódico de um oscilador é de grande importância para projetistas, pois permite verificar parâmetros como frequência de oscilação, potência de saída e consumo. Simulações transiente não são eficientes para avaliar o regime permanente periódico de um oscilador, visto que grande parte da computação requerida é desperdiçada durante o período transitório inicial. A alternativa explorada neste trabalho é o uso do método do Balanço Harmônico em conjunto com a técnica do Gerador Auxiliar para resolver o conjunto de equações diretamente para a resposta em regime permanente do oscilador. Para alcançar o objetivo, um simulador de circuitos elétricos foi implementado na linguagem de programação Python, com suporte a análises DC, AC, transiente e de balanço harmônico. Diversos modelos de simulação para dispositivos estão disponíveis, como elementos RLC, fontes de tensão e corrente, Diodo, BJT e MOSFET. Múltiplos exemplos são apresentados e os resultados de simulação comparados a simuladores comerciais para validar as implementações. Vantagens em simular circuitos dentro de um ambiente Python são apresentadas, envolvendo facilidade no tratamento de dados e possibilidade de integração com outras ferramentas.

Palavras-chave: Simulação de circuitos não-lineares. Balanço harmônico. Osciladores. Regime permanente periódico.

RESUMO EXPANDIDO

Introdução

A simulação de circuitos elétricos é fundamental para o desenvolvimento de sistemas eletrônicos e está integrada com a maioria das ferramentas de design de circuitos. Diversos algoritmos numéricos existem para solução do conjunto de equações diferenciais de um circuito elétrico. Análises DC, AC, transiente e de regime permanente são aplicadas para averiguar o comportamento de um circuito sob diversas condições de excitação. Um problema de grande interesse em circuitos de radiofrequência é o regime permanente periódico de circuitos autônomos não-lineares.

Osciladores são circuitos autônomos não-lineares cuja resposta de regime permanente periódico pode ser determinante nas figuras de mérito de um transceptor. Análises transientes são ineficientes na obtenção do regime permanente periódico de osciladores pois grande parte da computação requerida é desperdiçada durante o período transitório inicial. O algoritmo de Balanço Harmônico pode ser utilizado em conjunto com a técnica do Gerador Auxiliar para obter diretamente a resposta do oscilador em regime permanente de maneira mais eficiente.

Objetivos

O objetivo geral deste trabalho é o desenvolvimento de um software chamado YalRF. O software de código-aberto escrito na linguagem Python deve conter suporte ao algoritmo de Balanço Harmônico estendido para suportar circuitos autônomos osciladores utilizando a técnica do Gerador Auxiliar.

Os objetivos específicos deste trabalho são:

- Desenvolvimento de uma API amigável para descrição de *netlists,* análises e processamento de dados;
- Implementação dos algoritmos de simulação DC, AC e transiente;
- Inclusão de um número significativo de dispositivos para simulação, visando testar circuitos relevantes;
- Uso de gerenciador de pacotes simples para que o código suporte múltiplos sistemas operacionais com mínimo esforço;
- Escrita de código simples e legível visando utilizar este projeto como ponto de partida para pesquisas futuras em técnicas de simulação do estado-da-arte;
- Utilizar bibliotecas de Python para realizar otimização de circuitos.

Metodologia

A metodologia para atingir os objetivos propostos iniciou por uma ampla revisão de técnicas de simulação de circuitos, equações diferenciais, métodos numéricos, física de semicondutores, programação e circuitos para RF. A partir da revisão foi possível determinar o que deveria ser buscado para a implementação do software, delimitando modelos de dispostivos não-lineares e técnicas de simulação. Foi então realizada a análise matemática mais detalhada de todos os algoritmos a serem implementados: AC, DC, transiente, balanço harmônico multi-tons e com suporte a osciladores.

A modelagem matemática de dispositivos lineares e não-lineares também foi abordada em detalhe. Após o período de estudos foi dado início ao desenvolvimento dos códigos em Python. Para fazer a verificação e validação do funcionamento dos modelos de dispositivos e algortimos de análise, diversos códigos de teste em Python foram escritos emulando circuitos diversos. Os resultados de simulação obtidos com o

software YalRF também foram validados contra os simuladores Xyce, QucsStudio e ADS.

Resultados e Discussão
Diversos circuitos foram simulados utilizando YalRF e comparados a simuladores comerciais. Os resultados validaram os algoritmos e modelos implementados. Osciladores utilizando transistores bipolares tiveram sua resposta de regime permanente periódico obtida corretamente pelo software YalRF. A performance em tempo de execução dos algoritmos é naturalmente baixa devido ao uso da linguagem Python. De qualquer forma, problemas de convergência também atrasam a simulação e podem ser melhorados em alguns casos observados. A possibilidade de poder realizar paralelamente simulação e modelagem matemática dentro de um ambiente Python é uma característica intrínsica ao uso do YalRF e pode ser explorada no futuro para análise de circuitos.

Considerações Finais
Os objetivos traçados para o projeto foram alcançados. O software YalRF foi implementado em Python com suporte a análise de Balanço Harmônico de circuitos osciladores. A sintaxe para uso do simulador é simples e diversos dispositivos estão disponíveis para a descrição de circuitos. O uso do gerenciador de pacotes Conda faz com que a interoperabilidade do software seja bastante simples, facilitando a entrada de novos usuários e desenvolvedores. Apesar do software funcionar da maneira esperada, diversas melhorias podem ser aplicadas aos algoritmos explorados visando performance e estabilidade.

Palavras-chave: Simulação de circuitos não-lineares. Balanço harmônico. Osciladores. Regime permanente periódico.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AFM	Artificial Frequency Mapping
APFT	Almost-Periodic Fourier Transform
API	Application Programming Interface
BJT	Bipolar Junction Transistor
CB	Common-Base
CC	Common-Collector
DFT	Discrete Fourier Transform
EDA	Eletronic Design Automation
FFT	Fast Fourier Transform
GPU	Graphics Processing Unit
HB	Harmonic Balance
IDFT	Inverse Discrete Fourier Transform
KCL	Kirchhoff's Current Law
LMS	Linear Multistep Method
MNA	Modified Nodal Analysis
PSS	Periodic Steady-State
RF	Radiofrequency
TAHB	Transient-Assisted Harmonic Balance
UI	User Interface
YalRF	Yet-Another Lazy RF Circuit Simulator

CONTENTS

1 INTRODUCTION

Circuit simulation is a fundamental step of electronic systems design and is ubiquitous in Electronic Design Automation (EDA) tools. A number of numerical methods exist to evaluate a circuit response under different circumstances, such as the circuit DC operating point, small-signal AC characteristic and time-domain transient and steady-state responses. A problem that is often of great interest to designers is the evaluation of periodic or quasiperiodic steady-state response for nonautonomous nonlinear circuits.

The simplest solution for that problem is to perform a transient simulation and wait enough periods until the circuit response no longer changes. This approach is often not feasible for circuits with very different time constants, since the differential equation solver needs to run for a very long time for transients to vanish, while using a fine time step on the integrator. For example, the case of an amplifier two-tone test excited by the close frequencies f_1 and f_2, the intermodulation product $2f_1 - f_2$ can only be measured using a transient analysis if the total simulated period is many times greater than the difference $f_1 - f_2$. For a narrowband amplifier, f_1 and f_2 must be close to each other, so that their intermodulation product is not attenuated by bandwidth limitation. This creates a large ratio between the total period that needs to be simulated with respect to the average time step used to represent the radiofrequency (RF) signals. This ratio is directly related to the number of points to be simulated. Therefore, if it is made too large, a lot of numerical calculations and memory storage must be deployed in order to reach the quasiperiodic solution. Mixers suffer from a similar problem since the local oscillator and RF frequencies may be widely separated, and the difference in time constants creates a difficult situation for traditional transient analysis.

Another issue to obtain the quasiperiodic response of nonlinear networks is that RF and microwave circuits also often employ distributed elements on their configurations and those are tricky for transient simulation. Time-domain models for those devices are usually very difficult to obtain algebraically and they are often replaced by lumped approximations, which need very high order models for fidelity, or by impulse responses, that need expensive convolution operations (KUNDERT; WHITE; SANGIOVANNI-VINCENTELLI, 1990). Lastly, network analyzers extract very accurate measurement models for devices in the frequency-domain, therefore it would be interesting for a simulation method to directly support frequency domain representation, such as admittance or scattering parameters.

The mentioned problems are mostly solved by the popular Harmonic Balance technique (KUNDERT; SANGIOVANNI-VINCENTELLI, 1986). Harmonic Balance (HB) calculates the periodic/quasiperiodic steady-state response operating mainly in the frequency domain and usually employs Newton's method to solve the circuit's nonlinear algebraic system of equations. Due to its frequency domain nature, the computational complexity of Harmonic Balance depends only on the number of frequencies being used, not on their actual values or the associated time constants. For this reason, HB is often a much better choice in speed to solve for the steady-state response of mildly nonlinear systems.

Harmonic Balance can also be used to find the steady-state response of autonomous circuits with periodic and quasiperiodic responses (SUÁREZ, 2009). Oscillators, which are autonomous circuits with periodic responses, do not have a frequency of excitation over which to write the Harmonic Balance equations. Therefore, the frequency must be also considered an unknown variable to be found in the solution iterations (KUNDERT; WHITE; SANGIOVANNI-VINCENTELLI, 1990). A method called Auxiliary Generator Technique was formalized by Quere et al. (1993) and later by Ngoya and Suárez et al. (1995), in order to take advantage of HB robustness and numerical efficiency for the nonautonomous case and add auxiliary probes to force the circuit solution to converge for its autonomous response. Since this artificial probe contains a generator with a defined frequency of excitation working as candidate solution, this approach removes the issue of cases like the free-running oscillator, where there is no generator on the circuit at the frequency of interest. Commercial engines, such as Cadence's AWR and Synopsys's HSPICE RF, use this technique to assist users into quickly reaching the steady-state solution of autonomous circuits.

That last remark presents a particular issue with using the Harmonic Balance algorithm, specially for the design of oscillators. Commercial engines used for circuit design, such as AWR, ADS, Spectre RF and HSPICE RF, provide large support for HB and its more advanced features, like oscillating frequency determination and phase noise calculations. Nevertheless, the cost to access those tools is often prohibitive and creates a barrier to further popularize the use of HB for this type of design. Although there are a number of free and also open-source Spice-like circuit simulation engines out there (SpiceOpus, NGSpice, LTSpice), the HB algorithm is usually not a present feature. To the author's knowledge, the only free tools with stable multitone HB support are QucsStudio, fREEDA and Xyce. Yet, none of them support HB for autonomous circuits.

QucsStudio (MARGRAF, n.d.) has great usability, from a circuit designer perspective, due to its dedicated user-interface (UI), which enables the user to quickly setup complex HB analyses using its schematic capture. It works out of the box and no difficult setups are required. Its main downside is that, running into convergence issues using QucsStudio, the UI does not offer many options into the inner workings of the HB algorithm, such as configuring solver parameters and continuation methods. Also, since the code is not open-sourced, it is unclear what are the available options. Meanwhile, Xyce (SANDIA, n.d.), which is an open-source engine developed by Sandia Laboratories, provides a lot more freedom for the user to control its powerful HB simulation and have advanced options such as transient-assisted HB (TAHB) to help convergence. Xyce's disadvantage is that it doesn't ship with an UI to create schematics and process the resulting data. Qucs-S is a project in which Xyce can be integrated to the Qucs UI and greatly simplifies its usability, but as of today the integration is still somewhat limited. Lastly, fREEDA (STEER et al., 2006) is an open-source circuit simulation tool with focus on very high dynamic range transient simulation, very useful for wireless communication.

It also contains a working multitone HB implementation. Nonetheless, the fREEDA project was last updated in November 2010, its website is no longer accessible and, most importantly, since the code is no longer maintained, and as is usually the case with C++ projects, the build system is no longer working out of the box on newer operating systems. Therefore, anyone who wants to use fREEDA must go through the process of fixing the GNU Autotools scripts and compiling old libraries from scratch.

Finally, as seen by a 2020 funding topic from the American Department of Defense[1], there is still great interest on further research and development of RF circuit simulation capabilities, specially on the context of further improving simulation speed and dynamic range for time-varying RF spectral content. Envelope methods can be employed for this kind of time-varying analysis using HB (NGOYA; LARCHEVEQUE, 1996). The work from Zhang et al. (2021) discuss in length a recent trend in microwave circuit simulation towards automated modelling of nonlinear and parametric electromagnetic devices with the aid of neural networks and support vector machines. This approach can be used to speedup RF/microwave system level simulations without loss of accuracy. There are developments being made on leveraging Graphic Processing Units (GPUs) for accelerated circuit simulation (LEE; ACHAR; NAKHLA, 2018) and also optimized direct factorization for solving HB equations (BANDALI; GAD; BOLIC, 2014). Lastly, there is still development to reduce the size of HB problems and find approximate solutions, such as the work of Bizzarri, Brambilla and Codecasa (2016). Concepts from model order reduction techniques are employed to circuit simulation to reduce the computational complexity of HB, as in Gad et al. (2000), and can be applied to large sized problems, where the Jacobian matrix might not fit the system memory, or performing its factorization becomes prohibitively slow.

1.1 SCOPE OF THE PROJECT

As shown, the free options to run HB analysis for circuit design possess some kind of limitation. None of the options support the HB method for autonomous circuits. Besides, there is still a lot of academic interest on the development of ever more efficient algorithms and models to be used within HB. Therefore, this dissertation falls into the contexts of free and open-source circuit simulation, with emphasis on research and educational purposes, nonlinear RF systems and the Harmonic Balance algorithm with support to autonomous circuits.

1.1.1 Main Goal

The main goal of this project is the development of YaIRF (Yet-Another Lazy RF Circuit Simulator), an easy-to-use free and open-source circuit simulation engine with Harmonic Balance for autonomous circuits, written in the Python programming language.

[1] Link: `https://www.sbir.gov/node/1696355`. Last Accessed: 06/26/2021

1.1.2 Specific Goals

To accomplish the main goal of this project, some groundwork coding is required. Also other goals regarding future development and usability are taken into account. Therefore, the specific goals of this project can be summarized as:

- Development of an user-friendly Python Application Programming Interface (API) to describe circuits, analyses and process/plot the resulting data;
- Implement DC, AC and transient analysis, all of which can be used to support the HB algorithm;
- Inclusion of a significant number of device models, linear and nonlinear, so that meaningful circuits can be tested within the engine;
- Usage of simplified Python packaging for the software, so that multiple operating systems are supported with minimum effort and to avoid issues with unmantained/deprecated code;
- Write simple and readable code to encourage users to open the source of YalRF and better understand the simulation engine inner workings. Also, to serve as startup point for future research into simulation algorithms;
- Allow the possibility to integrate YalRF with other Python tools such as Jupyter Notebooks, openEMS, SignalIntegrity and scikit-RF;
- Use of Python optimization libraries to perform circuit tuning;

1.1.3 Dissertation Organization

The next chapter presents a brief review of the simulation of circuit networks. Chapter 3 reviews the theory behind the Harmonic Balance algorithm, how it is expanded into multitone analysis and the usage of the oscillator probe to obtain the autonomous response. The Chapter 4 introduces the features currently implemented in YalRF and how they were implemented. Next, Chapter 5 shows simulation results obtained using YalRF and compares them to other engines. Finally, Chapter 6 presents the concluding remarks with a number of suggestions for future work and research into YalRF.

2 BASICS OF CIRCUIT SIMULATION

Simulation of electronic circuits is a fundamental method used to understand the physical behavior of circuits comprising several elements, like resistors, capacitors, transistors, voltage sources and others. There are several types of analysis that can be performed over a circuit, and different types of input excitation being applied.

This chapter presents a brief review on the fundamentals of circuit network simulation. Section 2.1 presents the concept of the Modified Nodal Analysis (MNA) used for the circuit mathematical formulation. The next three sections introduce the concepts behind the most basic analyses: DC, AC and Transient. Lastly, Section 2.5 comments on the Periodic Steady-State (PSS) analysis and the main two algorithms associated to it.

2.1 MODIFIED NODAL ANALYSIS

The MNA is the standard method to automatically formulate a circuit set of equations. It is based on nodal analysis (Kichhoff's Current Law / KCL) and branch constitutive equations. MNA is chosen over standard nodal analysis due to its ability to naturally support ideal voltage sources and current-dependent controlled sources (MCCALLA, 1988). Considering a linear circuit, the MNA system can be written as:

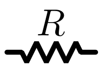
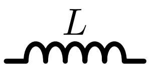
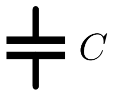
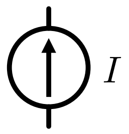
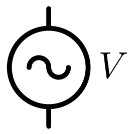
$$\begin{bmatrix} Y_R & B \\ C & D \end{bmatrix} \begin{bmatrix} V \\ I \end{bmatrix} = \begin{bmatrix} J \\ E \end{bmatrix} \tag{1}$$

where Y_R is a reduced admittance matrix determined by the elements interconnections, B is related to voltage source connections and output currents and C and D represent the constitutive equations of current-controlled devices, unknown current devices (such as independent voltage sources) and also output currents. J and E represent the values of independent current and voltages sources, respectively.

The process of filling the matrices from Equation 1 is performed by the concept of stamping. Every device has associated to it a matrix footprint, called a stamp, with its algebraic contribution to the system matrices. The stamping process consists of appropriately including the stamp of every device in the circuit to the MNA matrices, effectively formulating the circuit's complete system of equations. Figure 1 shows the basic circuit stamps of some linear devices for AC simulation with respect to their attached nodes.

To exemplify the stamping process, a simple DC linear example is shown in Figure 2.

Figure 1 – Example of stamps for basic elements in AC simulation.

	$\begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix}$
	$\begin{bmatrix} \frac{1}{j\omega L} & -\frac{1}{j\omega L} \\ -\frac{1}{j\omega L} & \frac{1}{j\omega L} \end{bmatrix}$
	$\begin{bmatrix} j\omega C & -j\omega C \\ -j\omega C & j\omega C \end{bmatrix}$
	$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I \\ -I \end{bmatrix}$
	$\begin{bmatrix} \cdot & \cdot & 1 \\ \cdot & \cdot & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ V \end{bmatrix}$

Source: Author.

Its MNA equations can be written as:

$$\begin{aligned} \text{Node 1:}\quad & I_1 + \frac{V_1 - V_2}{R_1} = 0 \\ \text{Node 2:}\quad & \frac{V_2 - V_1}{R_1} + I_2 + \frac{V_2 - V_3}{R_2} = 0 \\ \text{Node 3:}\quad & \frac{V_3 - V_2}{R_2} = I_{in} \\ & V_1 = V_{in} \\ & \frac{V_2}{R_3} - I_2 = 0 \end{aligned} \tag{2}$$

where the first three expressions are the circuit nodal analysis, the fourth equation is the independent voltage source, while the last equation represents the branch relation of current I_2, which was considered an unknown so that it can become an output variable of the analysis. The equations in 2 can be placed into the MNA matrix format:

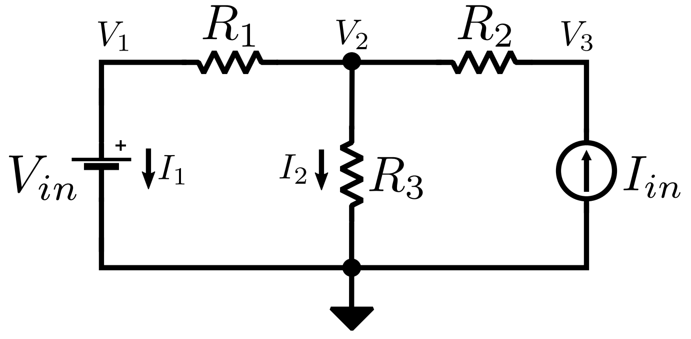
$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 & 1 & 0 \\ -\frac{1}{R_1} & \frac{1}{R_1} + \frac{1}{R_2} & -\frac{1}{R_2} & 0 & 1 \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{R_3} & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ I_{in} \\ V_{in} \\ 0 \end{bmatrix} \tag{3}$$

The stamps for the resistor, independent current and voltage sources from Figure 1 are the same for AC and DC analysis. Excluding the ground node, which adds no information to the system, and using the stamps from Figure 1, the matrix system in Equation 3 can be assembled automatically by a computer, simply by placing the stamps at the correct nodes. The only particularity was the stamp used for R_3. If the current through a resistor, I_R, is desired as a direct output of the MNA analysis, its stamping matrix needs to be modified:

$$\begin{bmatrix} . & . & 1 \\ . & . & -1 \\ \frac{1}{R} & -\frac{1}{R} & -1 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ I_R \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

Although the example presented is for a simple linear network, the stamping method for the MNA system can be expanded to support dynamic elements with time-varying nature and also nonlinear devices, by employing a linearized version of the system with an iterative solution.

Figure 2 – Simple resistive network for MNA example.



Source: Author.

2.2 DC ANALYSIS

The DC analysis is used to evaluate the operating point of a circuit. No time-varying input is considered, and the DC voltages and currents can be calculated at all nodes and branches. Nonlinear devices can be linearized at the operating point to obtain their models for small-signal variations. The solution to the system of equations of a linear DC circuit, as in (3), can be easily obtained using traditional direct methods such as Gaussian Elimination of LU Factorization (NAJM, 2010). To include support for nonlinear devices, such as diodes and transistors, a companion linear model is created for the device and included to the MNA system. Using an iterative solver, such as Newton-Raphson, the companion models are sequentially updated, using continuously improving solution guesses, until the MNA equations converge to a satisfying level. Rewriting the MNA system as a set of nonlinear equations:

$$g(V) = I \tag{5}$$

where V is the array of unknown variables to be obtained and I holds the independent current and voltage sources (the network stimulus vector). This equation can be transformed into an iterative root-finding problem by rewriting it as,

$$F(V^{(i)}) = g(V^{(i)}) - I^{(i)} = 0 \tag{6}$$

where i is the iteration number index and $F(V)$ is the function whose roots need to be determined. Finding the roots of $F(V)$ is equivalent to find the solution, V, of Equation 5. The most traditional method used for circuit simulation problems is the Newton-Raphson iteration.

2.2.1 Newton-Raphson Iteration

Assuming a candidate solution $V^{(i)}$ exists for the system in Equation 6, the Newton-Raphson iteration formula can be written as,

$$V^{(i+1)} = V^{(i)} - \left(J(V^{(i)})\right)^{-1} F(V^{(i)}) \tag{7}$$

where $V^{(i+1)}$ is and improved guess of the solution. J is the Jacobian matrix, defined as:

$$J(V^{(i)}) = \frac{\mathrm{d}F(V)}{\mathrm{d}V}\bigg|_{V=V^{(i)}} \tag{8}$$

Equation 7 can be rewritten as,

$$J(V^{(i)})V^{(i+1)} = J(V^{(i)})V^{(i)} - F(V^{(i)}) \tag{9}$$

Now the right-hand side of Equation 9 can be usefully interpreted. A more physical interpretation is that Newton-Raphson performs a linearization of the circuit equations over $V^{(i)}$ and improves the solution estimate by going on the direction of minimizing $F(V)$. Therefore, the Jacobian matrix will consist of the MNA matrix for the linear devices, along with the conductances obtained from linearization of the nonlinear models. The vector $F(V)$, is formed by the independent sources in the circuit, but also current sources arising from the equivalent companion models of nonlinear devices. That way, the MNA matrices can be automatically formed/updated at each iteration by recalculating those companion models, until a converged solution arises.

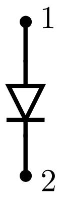
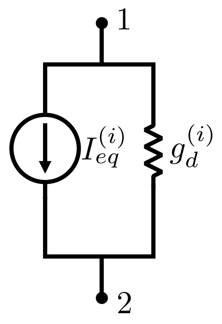
As an example, considering a diode described by the Shockley equation:

$$I_d = I_s\left(e^{\frac{V_d}{\phi_t}} - 1\right) \tag{10}$$

we can linearize this equation around $V^{(i)}$ as shown in Figure 3, and use those values to stamp the MNA matrices to perform the Newton-Raphson iteration.

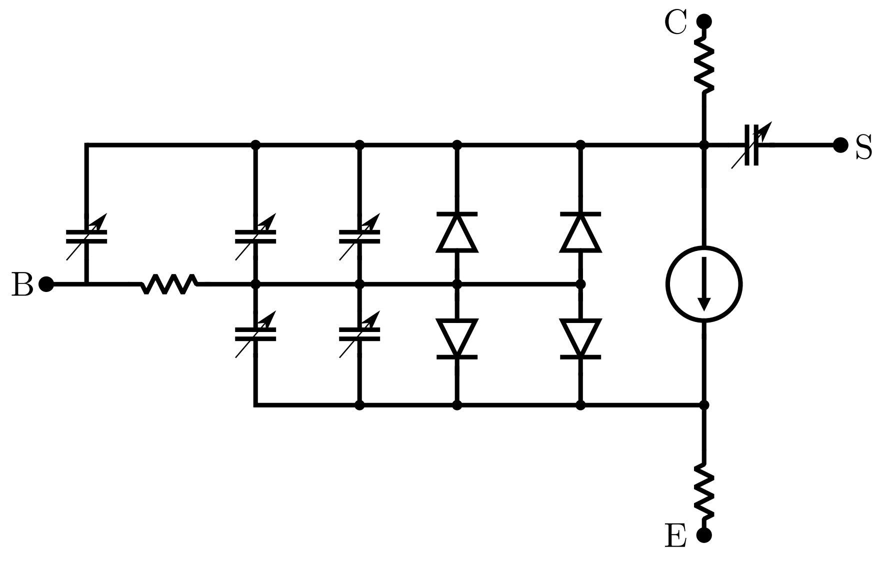
The procedure of creating companion models based on conductances and current sources done for the diode is employed on much more complex nonlinear models, such as the Gummel-Poon (GUMMEL; POON, 1970) for bipolar junction transistors (BJTs), shown in Figure 4 and implemented inside YalRF.

Figure 3 – Diode MNA stamps after linearization for Newton-Raphson.

Diode Schematic	Companion Model	MNA Stamps
		$\begin{bmatrix} g_d & -g_d \\ -g_d & g_d \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} -I_{eq} \\ I_{eq} \end{bmatrix}$ where, $g_d^{(i)} = \left.\frac{\partial I_d}{\partial V}\right\|_{V=V^{(i)}} = \frac{I_s}{\phi_t} e^{\frac{V^{(i)}}{\phi_t}}$ $I_{eq}^{(i)} = I_d^{(i)} - g_d^{(i)} V^{(i)}$

Source: Author.

Figure 4 – Bipolar Junction Transistor Gummel-Poon large-signal model.



Source: Author.

2.2.2 Limiting the Range of Device Models

The implementation of devices occasionally requires that limitations be applied to equations in a numerically acceptable manner. Functions like logarithm and exponential can quickly run into underflow/overflow issues, specially in HB, where large variations are expected to happen during the first iterations of the solution. Simply truncating is a poor choice, as it creates a discontinuity in the equation and brings convergence issues. The derivatives must also remain continuous.

There are a number of methods used to smoothly limit functions. For example, the voltage over an exponential junction, V, can be limited by using a hyperbolic tangent, given by (MCCALLA, 1988):

$$V_{lim} = V_{prev} + 10\phi_T \tanh \frac{V - V_{prev}}{10\phi_T} \tag{11}$$

where V_{prev} is the voltage over the exponential at the previous Newton-Raphson iteration and V_{lim} is the limited voltage to be used at the current iteration. This approach effectively limits the variation of the exponential function to $\pm 10\phi_T$ per iteration, greatly improving the stability of the exponential. Transient models are often less concerned with numerical stability, due to the incremental characteristic of transient simulations, where the per iteration variation in voltages/currents is small. That is not the case during a HB run, so models must be able to handle very large variations in voltage and current (MAAS, 2003).

2.2.3 Continuation Methods

During a Newton-Raphson run, sometimes the algorithm may fail to converge if the problem starts far from the solution or falls into some trap region. To improve convergence in such cases, a family of techniques, usually called continuation or homotopy methods are used. The core idea is to create a sequence of easier to solve incremental problems, such that the previous one serves as a good initial condition to the next, until the problem converges to the final circuit that must be solved (JAHN et al., 2007).

2.2.3.1 Source Stepping

Considering a circuit with an input vector I, the source stepping continuation method uses a multiplying factor $\alpha \in [0, 1]$, to solve for a problem of the type $I(\alpha) = \alpha I$, where the input of the circuit is being scaled down. Solving for continuously increasing values of α while reusing the previous solution as initial condition for the next has very good convergence. The final solution for the circuit of interest is reached when $\alpha = 1$.

2.2.3.2 g_{min} Stepping

Another technique of this family is the g_{min} stepping continuation method. The idea is to add a small conductance from every node of the circuit to ground. Starting, for example, from $g_{min} = 0.01$, a sequence of circuits is solved with gradually decreasing values of g_{min}. Again, the previous solution is used as a good initial condition to the next. The iteration ends when $g_{min} = 0$ and the original circuit is solved.

2.3 AC ANALYSIS

The AC analysis is a frequency-domain simulation used to evaluate the small-signal operation of a circuit around a determined operating point. No iterative process is required to reach the AC solution, since the nonlinear models are linearized at the DC bias point, and the resulting linear system is similar to the case of Equation 1. Therefore, all that is required to perform an AC analysis is to create the MNA matrices. Since there are frequency-dependent components of elements such as capacitors and inductors (see Figure 1) the AC matrices are filled by complex numbers. AC analyses are usually performed using frequency sweeps, therefore the complex part of the stamps dependent on $j\omega$, must be recalculated at each frequency value.

2.4 TRANSIENT ANALYSIS

Transient analysis is a time-domain nonlinear type of simulation. It is used to observe a circuit response to a particular stimulus over time. The algorithm of the transient analysis is very similar to what was presented for the DC case, with the difference that energy storing elements, such as inductors and capacitors, have also companion models obtained through the usage of integration methods.

Traditionally, linear multi-step (LMS) integration methods are used for circuit simulation due to good computational efficiency and their suitability to solve very stiff [1] differential equation problems (NAJM, 2010). The trapezoidal rule is a commonly used LMS method and can be written as (MCCALLA, 1988),

$$x_{k+1} = x_k + \frac{h_k}{2}(\dot{x}_{k+1} + \dot{x}_k) \tag{12}$$

where x_{k+1} represents the value of x at time t_{k+1} and h_k is the time-step of the simulation at the k-th time instant. The time-step is indexed in time since it generally varies throughout the simulation, being adapted as required by algorithms of local error estimation. Now, knowing that electrical current is the time-derivative of charge,

$$I(t) = \frac{\mathrm{d}Q(t)}{\mathrm{d}t} \tag{13}$$

the trapezoidal rule can be applied to find the charge through the numerical integration of current as,

$$Q_{k+1} = Q_k + \frac{h_k}{2}(I_{k+1} + I_k) \tag{14}$$

Considering a linear capacitor whose charge equation is $Q = CV$, Equation 14 can be rewritten to calculate the current flowing through a capacitor during a transient simulation,

$$I_{k+1} = \underbrace{\frac{2C}{h_k}}_{g_{eq}} V_{k+1} \underbrace{- \frac{2C}{h_k} V_k - I_k}_{I_{eq}} \tag{15}$$

This equation can finally be written in a format accepted by the MNA formulation,

$$I_{k+1} = g_{eq} V_{k+1} + I_{eq} \tag{16}$$

[1] stiffness being related to a system with strongly varying eigenvalues.

where the complete MNA entries are written like,

$$\begin{bmatrix} g_{eq} & -g_{eq} \\ -g_{eq} & g_{eq} \end{bmatrix} \begin{bmatrix} V_{1,k+1} \\ V_{2,k+1} \end{bmatrix} = \begin{bmatrix} -I_{eq} \\ I_{eq} \end{bmatrix} \tag{17}$$

This method can be generalized for other implicit LMS techniques of different orders, such as Gear, Adams-Bashforth and Adams-Moulton and also different energy storing elements. Since the problem of finding the transient solution V_{k+1} is implicit (I_{k+1} is required to predict V_{k+1}), the Newton-Raphson iteration is required to solve the MNA system, regardless of the presence of nonlinear devices. The textbook by McCalla (1988) and the Qucs documentation (JAHN et al., 2007) contain a lot more details into the features required for DC and transient analysis of nonlinear circuits employing MNA.

2.5 PERIODIC STEADY-STATE ANALYSIS

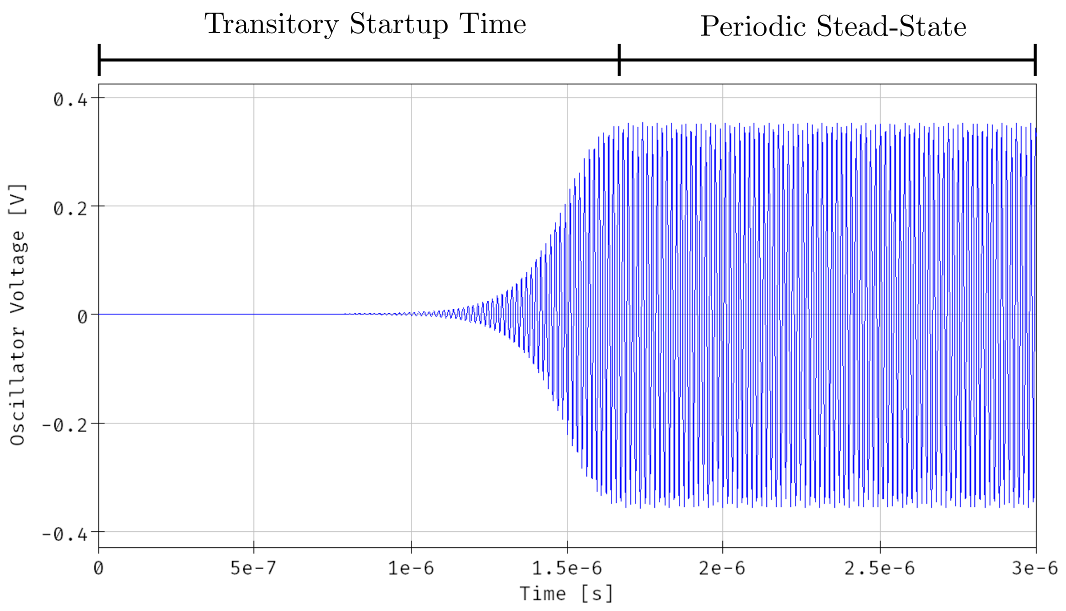
In the context of RF and microwave design, the steady-state response of a circuit is frequently of primary interest to the designer. Parameters such as distortion, gain and impedance can be much more well defined during the steady-state. If the steady-state response of a nonlinear circuit is comprised of a linear combination of DC solution and a number of harmonically related sinusoids, it is called a periodic steady-state response (PSS). The concept of quasiperiodic steady-state response also exists, in case the circuit is being driven by multiple non-harmonically related sinusoids, and the circuit response consists of a linear combination of the sum and differences of a set of finite frequencies and their harmonics.

It is often the case that a circuit present a long transient period before reaching steady-state. A simple example is a decoupling network created using very large capacitors. The charging time of those capacitors might be several orders of magnitude higher than the time constant of interest in the circuit. Performing a transient simulation in such a case can be very troublesome, since a lot of useless data is generated and time is consumed, before the circuit reaches the solution of interest. Figure 5 presents the transient simulation of a Colpitts oscillator, where the startup transitory is seen to consume a large portion of the simulated time window, although the startup might not be of any interest in case the designer is looking for the oscillating amplitude for example.

Specific algorithms were designed to perform directly a PSS analysis of nonlinear circuits with periodic excitations, removing the burden of transient simulations integrating through the transitory response. The two main techniques used for this type of analysis in circuit simulators are the time-domain Shooting method and the frequency-domain Harmonic Balance.

Shooting (APRILLE; TRICK, 1972) is a time-domain technique whose optimization problem aims to find a periodic solution for $x(t)$, submitted to the periodic constraint of $x(t_0 + T) - x(t_0) = 0$, where T is the period of the excitation. Starting from a solution candidate x_0, the circuit is integrated over an entire period and the constraint is verified. In

Figure 5 – Oscillator startup time before reaching periodic steady-state.



Source: Author.

case of failure, a root-finding algorithm, such as Newton-Raphson, can be used to update the value of x_0 and the process is repeated iteratively until convergence is reached.

The other technique used for PSS, and main focus of this dissertation, is the Harmonic Balance (KUNDERT; SANGIOVANNI-VINCENTELLI, 1986), which can be used to find the periodic and quasiperiodic response of nonautonomous (or driven) and autonomous (or non-driven) circuits. HB is a popular technique on modern EDA tools for RF and microwave circuits. The next chapter aims to provide an in-depth review of the HB technique.

3 HARMONIC BALANCE

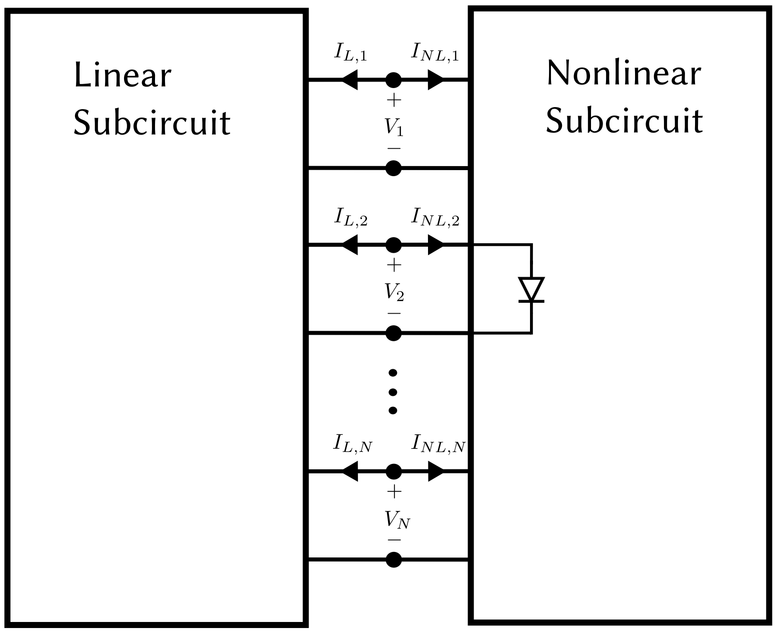
As mentioned previously, simulation of the PSS response of a circuit is often of major importance. Although transient analysis can be used to find the periodic steady-state solution, this approach is often prohibitively expensive. Circuits with long time-constants with respect to the period of the excitation signal, which is frequently the case in RF and microwave circuits, may have to integrate several periods before reaching steady-state. This was one of the main motivations cited by Nakhla and Vlach (1976), which introduced the Piecewise Harmonic Balance algorithm.

The Harmonic Balance differs from the classical transient approach in the sense that it uses Fourier series coefficients to represent signals, which allows it to solve circuits directly for their periodic or quasiperiodic steady-state solutions (SUÁREZ, 2009). The core idea is to use a frequency-domain representation of the KCL equations with a truncated amount of harmonics, and attempt to balance, at each chosen node, the sum of currents for every harmonic frequency. Linear elements can be treated directly in the frequency-domain using phasor analysis, while nonlinear devices use their time-domain models with the aid of the Discrete Fourier Transform (DFT). Since physical systems have an intrinsic low-pass behavior, it is possible to truncate its signals Fourier representation to a limited number of harmonics.

Harmonic Balance is heavily employed in the context of RF and microwave circuits, such as power amplifiers, mixers and oscillators. Besides being able to quickly handle circuits with large time-constants and fast inputs for their PSS solution, the frequency-domain nature of its formulation makes it easy to integrate Y or S-parameter models to the simulation, which can be a difficult task in the time-domain (MAAS, 2003).

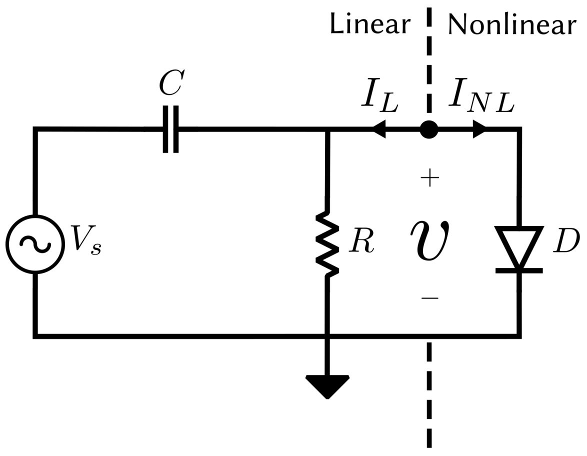
As mentioned by Kundert et al. (1990), the initial numerical formulation of Harmonic Balance was developed by Baily (1968). It used an optimization algorithm to adjust the Fourier coefficients of the solution until the least error was reached. In 1974, Egami (1974) showed for a resistive mixer, that the KCL equations in the frequency-domain could be solved iteratively by using the Newton-Raphson method. In 1976, Nakhla and Vlach (1976) presented a general formulation for the Piecewise Harmonic Balance method. In the piecewise approach, the current balancing is applied only to nodes with linear and nonlinear elements attached. The goal is to partition the circuit into linear and nonlinear subcircuits (see Figure 6). After a initial guess for the port voltages, which are each represented by a truncated Fourier series, the goal is to iteratively improve the port voltage guess until the currents that flow into the linear and nonlinear branches are balanced.

Figure 6 – A nonlinear circuit partitioned into linear and nonlinear subcircuits.



Source: Author.

Figure 7 – Capacitive coupling to a diode load.



Source: Author.

3.1 AN INTRODUCTORY EXAMPLE TO HARMONIC BALANCE

Considering the circuit shown in Figure 7. Applying KCL, the current sum at the main node can be written as:

$$\frac{v(t)}{R} + C\frac{\mathrm{d}[v(t) - V_s \cos \omega t]}{\mathrm{d}t} + I_s e^{\frac{v(t)}{\phi_t}} = 0 \tag{18}$$

This is a nonlinear differential equation that cannot be solved algebraically. In a transient simulation, the solution would be to numerically integrate Equation 18 starting from an initial condition until periodic steady-state is reached. But a frequency-domain approach to this problem can also be taken. At steady-state, it is possible to decompose the currents flowing to the linear and nonlinear subcircuits into a sum of phasors at the harmonic frequencies of the fundamental excitation. This is analogous to employing the Fourier series. Now, truncating the number of harmonics being considered to K, the KCL equations for each harmonic can be written as:

$$I_L(k\omega) + I_{NL}(k\omega) = 0 \quad \text{for } k = 0, 1, ..., K \tag{19}$$

Assuming $V(k\omega)$ is the phasor of the node voltage at frequency $k\omega$, the current flowing into the linear circuit for each harmonic can be calculated,

$$\begin{aligned} I_L(0) &= \frac{V(0)}{R} \\ I_L(\omega) &= j\omega C[V(\omega) - V_s] + \frac{V(\omega)}{R} \\ I_L(k\omega) &= jk\omega CV(k\omega) + \frac{V(k\omega)}{R} \quad \text{for } k = 2, 3, ..., K \end{aligned} \tag{20}$$

For the nonlinear current flowing through a diode, the current phasor amplitude for each harmonic can be found by using the results by Clarke and Hess (1971). First, two strong considerations must be made: that the voltage harmonics above the fundamental have a negligible amplitude and that $V(\omega)$ is a real number. Both assumptions are fair if the capacitor is large enough to look like a short at the fundamental. Now, writing the diode current when submitted to a sinusoidal voltage as a trigonometric Fourier series:

$$\begin{aligned} I_{NL}(t) &= I_s \exp\left\{ \frac{(V(0) + V(\omega)\cos(\omega t))}{\phi_t} \right\} \\ &= \sum_{n=0}^{K} C_n \cos(\omega t) \end{aligned} \tag{21}$$

where

$$\begin{aligned} C_0 &= \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{\frac{V(\omega)\cos(\theta))}{\phi_t}} d\theta \\ C_k &= \frac{1}{\pi} \int_{-\pi}^{\pi} e^{\frac{V(\omega)\cos(\theta))}{\phi_t}} \cos k\theta d\theta \quad \text{for } k = 1, 2, ..., K \end{aligned} \tag{22}$$

The C_n coefficients can be found by using the modified Bessel function of the first kind, $I_k(x)$:

$$\begin{aligned} C_0 &= I_s e^{\frac{V(0)}{\phi_T}} I_0 \left(\frac{V(\omega)}{\phi_T} \right) \\ C_k &= 2I_s e^{\frac{V(0)}{\phi_T}} I_k \left(\frac{V(\omega)}{\phi_T} \right) \quad \text{for } k = 1, 2, ..., K \end{aligned} \tag{23}$$

Inspecting Equations 21 and 23 it can be seen that the Fourier coefficients, C_n, are in fact the amplitude of each current harmonic entering the nonlinear subcircuit, $I_{NL}(k\omega) = C_k$ for all k. Thus this result can be used with Equation 19, to calculate the balance between the linear and nonlinear currents.

From this point on two more things are needed to be able to reach a solution. First, a reasonable guess of the initial node voltage $V(k\omega)$ for all k. And second, a method to iteratively improve this voltage guess in case Equation 19 does not hold true.

But before that, it is worth discussing the treatment that was done for the nonlinear element in this circuit. For the exponential nonlinearity, it was possible to compute the necessary current harmonic amplitudes. A strong assumption about the circuit behavior was needed: it only works for large capacitors. Also, this approach might not be viable for other types of nonlinearities, whose algebraic treatment to obtain the Fourier coefficients could be much more complex. Lastly, the most standard numerical method used to find the node voltages at Harmonic Balance is the Newton-Raphson iteration (MAAS, 2003). Newton's method demand that the terms $\frac{\partial I_k}{\partial V_l}$ are calculated form the Jacobian, which is the derivative of the k-th current phasor with respect to the l-th voltage phasor. Obtaining this result algebraically can be very troublesome. Therefore, a more general approach to handle the problem of finding the Fourier series of nonlinear elements is required. That is the role played by the Discrete Fourier Transform.

Nonlinear devices, such as diodes, BJTs and MOSFETs usually have very complex time-domain models for their I/V characteristics and also its derivatives $\frac{\partial i}{\partial v}$. It is convenient to re-use those models in the Harmonic Balance simulation. For that, the most straightforward way to obtain the nonlinear device coefficients is to convert the circuit voltage, $V(k\omega)$, into a sampled time-domain representation, $v(t)$, using the Inverse Discrete Fourier Transform (IDFT). Then the nonlinear devices can be evaluated in the time-domain and the results are converted back into phasors using the forward DFT.

Before introducing the complete HB algorithm, to conclude the example from Figure 7, Appendix A presents how to model the Harmonic Balance problem to reach the circuit solution without using the Fourier Transform. The main idea is to note from the Fourier coefficients of Equation 23, that the elements of $I_{NL}(k\omega)$ are already functions of the coefficients of $V(k\omega)$. Therefore the derivative terms $\frac{\partial I_k}{\partial V_l}$ can be approximated using the Secant method to obtain the periodic steady-state response.

3.2 THE HARMONIC BALANCE ALGORITHM

In this section, the ideas previously presented are used to show a more general approach to HB following the derivations by Kundert et al. (1986) and (1988) and Maas (2003), using the nodal approach instead of the piecewise method. As mentioned, the Discrete Fourier Transform is a critical step to perform the time/frequency domain conversions necessary to solve the harmonic balance equations. Therefore, this section starts by introducing some basic notation and then the DFT definition.

3.2.1 Notation

- N is the number of nodes in the circuit;
- K is the number of harmonics used for harmonic balance;
- T_0, ω_0 are the period and natural frequency of the first harmonic;
- $x(t)$ is a time-domain signal;
- $\mathbf{X}$ is a vector with the complex spectrum of $x(t)$;
- $\overline{\mathbf{X}}$ is the $\mathbb{R}^2$ representation of $\mathbf{X}$, where every complex number in $\mathbf{X}$ is expanded to an $\mathbb{R}^2$ vector;
- $X(k)$ is the complex phasor at frequency $k\omega_0$;
- n, m subscripts are used for circuit node index, e.g., $x_n(t)$ represents the time-domain signal at node n and $\mathbf{X_n}$ is its frequency representation;

3.2.2 The Discrete Fourier Transform

Considering a time-domain signal, represented by lower-case letters, $x(t) \in \mathbb{R}$, is sampled at equally spaced $2K+1$ time samples $t_0, t_1, \ldots, t_{2K} \in [0, T_0)$. To determine the Fourier coefficients, also called phasors, the single-sided or trigonometric form of the Discrete Fourier Transform (DFT) is used:

$$\overline{X}(k) = \begin{bmatrix} X^C(k) \\ X^S(k) \end{bmatrix} = \frac{2-\delta(k)}{2K+1} \sum_{s=0}^{2K} \begin{bmatrix} \cos(k\omega_0 t_s) \\ \sin(k\omega_0 t_s) \end{bmatrix} x(t_s) \tag{24}$$

where $\delta(k)$ is the Kronecker delta function, defined as,

$$\delta(k) = \begin{cases} 1 & \text{if} \quad k = 0 \\ 0 & \text{if} \quad k \neq 0 \end{cases} \tag{25}$$

and $\overline{X}(k)$ is an $\mathbb{R}^2$ representation of the complex $X(k)$, which is the phasor of $x(t)$ at frequency $k\omega_0$. The set of $X(k)\,\forall\, k \in [0, K]$ is the spectrum of $x(t)$, or its frequency-domain representation. To convert back to the time-domain, the IDFT can be used:

$$x(t_s) = X(0) + \sum_{k=1}^{K} \begin{bmatrix} \cos(k\omega_0 t_s) & \sin(k\omega_0 t_s) \end{bmatrix} \begin{bmatrix} X^C(k) \\ X^S(k) \end{bmatrix} \tag{26}$$

and the IDFT can also be represented as a matrix operator,

$$x(t) = \underbrace{\begin{bmatrix} 1 & \cos(\omega_0 t_0) & \sin(\omega_0 t_0) & \cdots & \cos(K\omega_0 t_0) & \sin(K\omega_0 t_0) \\ 1 & \cos(\omega_0 t_1) & \sin(\omega_0 t_1) & \cdots & \cos(K\omega_0 t_1) & \sin(K\omega_0 t_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \cos(\omega_0 t_{2K}) & \sin(\omega_0 t_{2K}) & \cdots & \cos(K\omega_0 t_{2K}) & \sin(K\omega_0 t_{2K}) \end{bmatrix}}_{\mathbf{\Gamma^{-1}}} \underbrace{\begin{bmatrix} X(0) \\ X^C(1) \\ X^S(1) \\ \vdots \\ X^C(K) \\ X^S(K) \end{bmatrix}}_{\mathbf{X}} \tag{27}$$

Similarly the DFT can be represented in matrix format as $\mathbf{\Gamma}$, and the DFT/IDFT matrix pair is used to derive the Jacobian matrix required to solve the Newton-Raphson iteration of the HB algorithm.

3.2.3 Single-Tone Nodal Analysis

Considering the contributions from nonlinear and voltage-controlled resistors and capacitors, linear devices and independent current sources, the KCL can be written at all nodes as,

$$f(v,t) = i(v(t)) + \frac{\mathrm{d}q(v(t))}{\mathrm{d}t} + \int_{-\infty}^{t} y(t-\tau)v(\tau)d\tau + i_s(t) = 0 \tag{28}$$

where v is the vector of all node voltage waveforms, i is the current contribution from nonlinear conductors, q is the charge contribution by nonlinear capacitors and y is the matrix-valued impulse response of the circuit with the nonlinear devices removed.

Now the terms from Equation 28 can be examined and converted to the frequency-domain. First the term $i(v(t))$, which represents the current contribution of every nonlinear device at every node of the circuit. Writing the current at a single node as,

$$i_n(v(t)) = f_i(v_1(t), v_2(t), ..., v_N(t)), \qquad n \in \{1, 2, \ldots, N\} \tag{29}$$

where n is the circuit node index. Applying the Fourier transform (operator $\mathcal{F}\{\}$) to this signal and truncating to K harmonics this current writes as,

$$\mathcal{F}\{i_n(v(t))\} = \mathbf{I_n} = \begin{bmatrix} I_n(0) \\ I_n(1) \\ I_n(2) \\ \cdots \\ I_n(K) \end{bmatrix}, \qquad n \in \{1, 2, \ldots, N\} \tag{30}$$

where $\mathbf{I_n} \in \mathbb{C}^{K+1}$ and each element $I_n(k) \in \mathbb{C}$, for $k = 0, 1, ..., K$, is the current phasor at frequency $k\omega_0$. Then the frequency-domain current vector for all nodes can be written as,

$$\mathbf{I}(\mathbf{V}) = \begin{bmatrix} \mathbf{I_0} \\ \mathbf{I_1} \\ \mathbf{I_2} \\ ... \\ \mathbf{I_N} \end{bmatrix} = \begin{bmatrix} I_1(0) \\ I_1(1) \\ ... \\ I_1(K) \\ I_2(0) \\ I_2(1) \\ ... \\ I_2(K) \\ ... \\ I_N(0) \\ I_N(1) \\ ... \\ I_N(K) \end{bmatrix} \tag{31}$$

where $\mathbf{I}(\mathbf{V}) \in \mathbb{C}^{N(K+1)}$ is the complete array with the spectrum of all node currents created by nonlinear resistors.

Using the same analysis for the nonlinear capacitor term, the charge for a single node is given by,

$$q_n(v(t)) = f_q(v_1(t), v_2(t), ..., v_N(t)), \qquad n \in \{1, 2, \dots, N\} \tag{32}$$

where q_n denotes the charge contribution of the nonlinear capacitances at node n. Applying the Fourier transform to this signal,

$$\mathcal{F}\{q_n(t)\} = \mathbf{Q_n} = \begin{bmatrix} Q_n(0) \\ Q_n(1) \\ Q_n(2) \\ ... \\ Q_n(K) \end{bmatrix}, \qquad n \in \{1, 2, \dots, N\} \tag{33}$$

where $Q_n \in \mathbb{C}^{K+1}$ is the spectrum of the charge for a single node and $Q_n(k)$ represent the charge phasor at frequency $k\omega_0$. The charge vector for all nodes, $\mathbf{Q}(\mathbf{V})$, can be written as,

$$\mathbf{Q}(\mathbf{V}) = \begin{bmatrix} \mathbf{Q_0} \\ \mathbf{Q_1} \\ \mathbf{Q_2} \\ \vdots \\ \mathbf{Q_N} \end{bmatrix} = \begin{bmatrix} Q_1(0) \\ Q_1(1) \\ \vdots \\ Q_1(K) \\ Q_2(0) \\ Q_2(1) \\ \vdots \\ Q_2(K) \\ \vdots \\ Q_N(0) \\ Q_N(1) \\ \vdots \\ Q_N(K) \end{bmatrix} \tag{34}$$

where $\mathbf{Q}(\mathbf{V}) \in \mathbb{C}^{N(K+1)}$.

The charge derivative term is required to calculate the current contribution of the nonlinear capacitances. The time-derivative corresponds to multiplying by $j\omega$ in the frequency-domain, therefore,

$$\mathcal{F}\left\{\frac{\mathrm{d}q_n(v(t))}{\mathrm{d}t}\right\} = jk\omega_0 Q_n(k), \qquad k \in \{0, 1, \ldots, K\} \tag{35}$$

Rewriting this as a matrix expression,

$$\mathcal{F}\left\{\frac{\mathrm{d}q(v(t))}{\mathrm{d}t}\right\} = j\mathbf{\Omega}\mathbf{Q}(\mathbf{V}) \tag{36}$$

where $\mathbf{\Omega}$ is a block diagonal matrix,

$$\mathbf{\Omega_{m,n}} = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & \omega_0 & 0 & \ldots & 0 \\ 0 & 0 & 2\omega_0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & 0 \\ 0 & 0 & 0 & \ldots & K\omega_0 \end{bmatrix} \tag{37}$$

$$\mathbf{\Omega} = \begin{bmatrix} \mathbf{\Omega_{1,1}} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{\Omega_{2,2}} & \ldots & \mathbf{0} \\ \ldots & \ldots & \ldots & \ldots \\ \mathbf{0} & \mathbf{0} & \ldots & \mathbf{\Omega_{N,N}} \end{bmatrix} \tag{38}$$

where $m, n = 1, 2, ..., N$ are node indexes.

The next term to be examined is the convolution integral responsible for the current contribution of the linear elements. A time-domain convolution represents a multiplication in the frequency-domain, so the following can be written:

$$\int_{-\infty}^{t} y(t-\tau)v(\tau)d\tau \xrightarrow{\mathcal{F}} \mathbf{YV} \tag{39}$$

where $\mathbf{V} \in \mathbb{C}^{N(K+1)}$ is the voltage spectrum for all nodes. $\mathbf{Y} \in \mathbb{C}^{N(K+1)\times N(K+1)}$ is the block node admittance matrix of the linear subcircuit. The matrix $\mathbf{Y}$ contains the admittances connected at all nodes and evaluated at all harmonic frequencies. The current contribution of the linear elements, $\mathbf{I_L}$, can be calculated as,

$$\begin{bmatrix} \mathbf{I_{L,1}} \\ \mathbf{I_{L,2}} \\ \vdots \\ \mathbf{I_{L,N}} \end{bmatrix} = \begin{bmatrix} \mathbf{Y_{1,1}} & \mathbf{Y_{1,2}} & \dots & \mathbf{Y_{1,N}} \\ \mathbf{Y_{2,1}} & \mathbf{Y_{2,2}} & \dots & \mathbf{Y_{2,N}} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{Y_{N,1}} & \mathbf{Y_{N,2}} & \dots & \mathbf{Y_{N,N}} \end{bmatrix} \begin{bmatrix} \mathbf{V_1} \\ \mathbf{V_2} \\ \vdots \\ \mathbf{V_N} \end{bmatrix} \tag{40}$$

where,

$$\mathbf{Y_{m,n}} = \begin{bmatrix} Y_{m,n}(0) & 0 & \dots & 0 \\ 0 & Y_{m,n}(1) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & Y_{m,n}(K) \end{bmatrix} \tag{41}$$

and $Y_{m,n}(k) \in \mathbb{C}$ is the total admittance connected between nodes m and n, evaluated at frequency $k\omega_0$.

Lastly, there is the term corresponding to the independent current sources of the circuit, $i_s(t)$. Converting this vector to the frequency-domain,

$$\mathcal{F}\{i_s(t)\} = \mathbf{I_s} \tag{42}$$

where $I_s \in \mathbb{C}^{N(K+1)}$ and can be written as,

$$\mathbf{I_s} = \begin{bmatrix} \mathbf{I_{s,1}} \\ \mathbf{I_{s,2}} \\ \vdots \\ \mathbf{I_{s,N}} \end{bmatrix} = \begin{bmatrix} I_{s,1}(0) \\ I_{s,1}(1) \\ \vdots \\ I_{s,1}(K) \\ I_{s,2}(0) \\ I_{s,2}(1) \\ \vdots \\ I_{s,2}(K) \\ \vdots \\ I_{s,N}(0) \\ I_{s,N}(1) \\ \vdots \\ I_{s,N}(K) \end{bmatrix} \tag{43}$$

where, similarly to the previous cases, $\mathbf{I_{s,n}} \in \mathbb{C}^{(K+1)}$ is the current spectrum at node n and $I_{s,n}(k) \in \mathbb{C}$ is the current phasor at harmonic $k\omega_0$ for node n.

With all the terms at hand, the frequency-domain version of Equation 28 can be written:

$$\mathbf{F}(\mathbf{V}) = \mathbf{I}(\mathbf{V}) + j\mathbf{\Omega}\mathbf{Q}(\mathbf{V}) + \mathbf{YV} + \mathbf{I_s} = \mathbf{0} \tag{44}$$

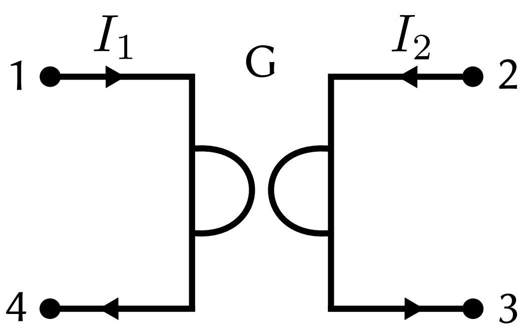
This equation is the frequency-domain equivalent of KCL and the foundation of Harmonic Balance. Basically it shows that, for each node of the circuit, and for each harmonic separately, summing the current phasor contributions of the circuit elements should balance to zero.

The term $\mathbf{F}(\mathbf{V})$ was added to Equation 44, because it will serve as an error function for the numerical problem. After determining what is the guess solution, $\mathbf{V}$, all the current contributions above are calculated and summed, to check if they reached balance. Ideally, if the exactly correct answer for $\mathbf{V}$ is found, Equation 44 will return zero. But since the problem generally starts with very rough guesses for the solution, $\mathbf{F}(\mathbf{V})$ will serve as a metric to determine how far the trial vector $\mathbf{V}$ is from the ideal solution.

3.2.3.1 Including Voltage Sources

Inspecting Equation 44 it can be noted that there is no straightforward way to include independent voltage sources (DC or sinusoids at frequency $k\omega_0$) to the KCL formulation. The simplest way to cope with this problem is to use a gyrator. The gyrator shown in Figure 8, can be modeled as,

Figure 8 – Ideal gyrator.



Source: Author.

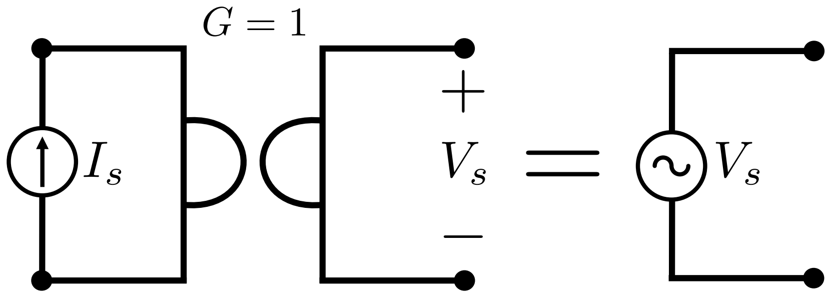
$$
\begin{aligned}
I_1 &= G \cdot (V_2 - V_3) \\
&= G \cdot V_2 - G \cdot V_3 \\
I_2 &= -G \cdot (V_1 - V_4) \\
&= -G \cdot V_1 + G \cdot V_4 \\
I_3 &= -I_2 \\
I_4 &= -I_1
\end{aligned}
\tag{45}
$$

Now its nodal analysis can be written in matrix format,

$$
\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & G & -G & 0 \\ -G & 0 & 0 & G \\ G & 0 & 0 & -G \\ 0 & -G & G & 0 \end{bmatrix}}_{\mathbf{Y_{gyrator}}} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}
\tag{46}
$$

Looking at the expressions in Equation 45, it can be seen that by making $G = 1$ and placing a current source between terminals 1 and 4, a voltage of same amplitude appears between terminals 2 and 3. Therefore, if an ideal independent voltage source is needed in the Harmonic Balance analysis, all that is required is to place a gyrator in series with a current source, making the current magnitude the same as the wanted voltage and the gyrator conductance equal to unity, as shown in Figure 9.

Figure 9 – Current source in series with a gyrator to model a voltage source.



Source: Author.

3.2.4 Numerical Solution of the HB Equations

Equation 44 provides a way to evaluate if a certain trial solution $\mathbf{V}$ is correct. But there is still the question of how to update $\mathbf{V}$ in case the solution does not satisfy the convergence criteria.

Optimization and relaxation methods can be used to find a solution for $\mathbf{V}$ (KUNDERT; SANGIOVANNI-VINCENTELLI, 1986). The optimization approach uses a cost function of type $\mathbf{F}^*(\mathbf{V})\mathbf{F}(\mathbf{V})$, where * represents the transposed conjugate. There is a large number of algorithms that can be used to minimize this cost function and optimize the Fourier coefficients of $\mathbf{V}$ to reach a solution. However, the use of a cost function eliminates the information regarding the individual contributions of currents to the error function. This limits the approaches that can be used to handle issues with convergence. Besides that, the large number of variables that need to be optimized makes optimization algorithms a less attractive approach for HB problems (KUNDERT; SANGIOVANNI-VINCENTELLI, 1986). Relaxation methods on the other hand, although simple, may suffer from unpredictable and possibly slow convergence characteristics. The consensus method to numerically solve the HB equations is the Newton-Raphson iteration (MAAS, 2003).

3.2.4.1 The Newton-Raphson Approach

Finding the roots of the error function $\mathbf{F}(\mathbf{V})$ results in finding the solution vector $\mathbf{V}$ of the HB set of equations. Using the Newton-Raphson iteration to Equation 44, the improved solution vector, $\mathbf{V}^{(i+1)}$, is written as:

$$\mathbf{V}^{(i+1)} = \mathbf{V}^{(i)} - \mathbf{J}(\mathbf{V}^{(i)})^{-1}\mathbf{F}(\mathbf{V}^{(i)}) \tag{47}$$

where $\mathbf{J}$ is the Jacobian matrix of $\mathbf{F}$, defined as:

$$\mathbf{J}(\mathbf{V}^{(i)}) = \frac{\mathrm{d}\mathbf{F}(\mathbf{V})}{\mathrm{d}\mathbf{V}}\bigg|_{\mathbf{V}=\mathbf{V}^{(i)}} \tag{48}$$

and the i superscript in parenthesis indicates the estimate solution at the ith iteration.

The Jacobian matrix contains the derivative of each error vector with respect to each harmonic component of $\mathbf{V}$, and can be interpreted as a measure of the sensitivity of all error components to the voltage. Now, derivating Equation 44 with respect to $\mathbf{V}$ to obtain the frequency-domain Jacobian:

$$\mathbf{J}(\mathbf{V}) = \frac{\mathrm{d}\mathbf{F}(\mathbf{V})}{\mathrm{d}\mathbf{V}} = \frac{\partial\mathbf{I}(\mathbf{V})}{\partial\mathbf{V}} + j\mathbf{\Omega}\frac{\partial\mathbf{Q}(\mathbf{V})}{\partial\mathbf{V}} + \mathbf{Y} = \mathbf{G} + j\mathbf{\Omega}\mathbf{C} + \mathbf{Y} \tag{49}$$

The phasor vectors and matrices so far were ordered in a harmonic-minor node-major format, since it first orders the harmonic indexes and subsequently the nodes. This ordering is convenient for the next step of NR, which involves the use of the DFT/IDFT transforms. Another step for implementation is to expand all complex phasors on vectors and matrices to an $\mathbb{R}^2$ representation. For that, each phasor, excluding the DC value which is a real number, is split into its trigonometric representation, i.e., for $X(k) \in \mathbb{C}$, $\overline{X}(k) = [X^C(k)\ X^S(k)]^T$. A topbar notation is used to represent the $\mathbb{R}^2$ converted vectors and matrices. Expanding the voltage and nonlinear current vectors one obtains:

$$\overline{\mathbf{V}} = \begin{bmatrix} V_1(0) & V_1^C(1) & V_1^S(1) & \ldots & V_N^C(K) & V_N^S(K) \end{bmatrix}^T \tag{50}$$

$$\bar{\mathbf{I}} = \begin{bmatrix} I_1(0) & I_1^C(1) & I_1^S(1) & \ldots & I_N^C(K) & I_N^S(K) \end{bmatrix}^T \tag{51}$$

where $\overline{\mathbf{V}}$ and $\overline{\mathbf{I}}$ are $N(2K+1)$ vectors of Fourier coefficients. $\mathbf{F}$, $\mathbf{Q}$ and $\mathbf{I_S}$ are similarly converted into $\overline{\mathbf{F}}$, $\overline{\mathbf{Q}}$ and $\overline{\mathbf{I_S}}$. Matrix $\mathbf{Y}$ expansion into $\mathbb{R}^2$ is achieved by making each complex element $Y_{m,n}(k)$ into a 2×2 submatrix of the format,

$$Y_{m,n}(k) \rightarrow \begin{bmatrix} Y^R_{m,n}(k) & -Y^I_{m,n}(k) \\ Y^I_{m,n}(k) & Y^R_{m,n}(k) \end{bmatrix} \tag{52}$$

and the complete $\overline{\mathbf{Y}}$ is an $N(2K+1)\times N(2K+1)$ matrix. The term $j\mathbf{\Omega}$ from Equation 49 is expanded into $\overline{\mathbf{\Omega}}$ by rewriting each individual frequency from the main diagonal in Equation 37, except from DC, as the 2×2 submatrix,

$$\Omega_{m,n}(k) \rightarrow \begin{bmatrix} 0 & -k\omega_0 \\ k\omega_0 & 0 \end{bmatrix} \tag{53}$$

where matrix $\overline{\mathbf{\Omega}} \in \mathbb{R}^{N(2K+1)\times N(2K+1)}$. Expanding also the complex phasors in $\mathbf{G}$ and $\mathbf{C}$ into their $\mathbb{R}^2$ representations, Equation 49 can be rewritten as,

$$\overline{\mathbf{J}} = \frac{\mathrm{d}\overline{\mathbf{F}}(\overline{\mathbf{V}})}{\mathrm{d}\overline{\mathbf{V}}} = \overline{\mathbf{G}} + \overline{\mathbf{\Omega}}\,\overline{\mathbf{C}} + \overline{\mathbf{Y}} \tag{54}$$

where $\overline{\mathbf{G}}$ and $\overline{\mathbf{C}}$ are $N(2K+1)\times N(2K+1)$ matrices defined as,

$$\overline{\mathbf{G}} = \frac{\partial\overline{\mathbf{I}}(\overline{\mathbf{V}})}{\partial\overline{\mathbf{V}}} \tag{55}$$

$$\overline{\mathbf{C}} = \frac{\partial\overline{\mathbf{Q}}(\overline{\mathbf{V}})}{\partial\overline{\mathbf{V}}} \tag{56}$$

and $\overline{\mathbf{J}}$, being the Jacobian, can be written as,

$$\overline{\mathbf{J}} = \begin{bmatrix} \frac{\partial\overline{\mathbf{F}}_\mathbf{1}}{\partial\overline{\mathbf{V}}_\mathbf{1}} & \cdots & \frac{\partial\overline{\mathbf{F}}_\mathbf{1}}{\partial\overline{\mathbf{V}}_\mathbf{N}} \\ \vdots & \ddots & \vdots \\ \frac{\partial\overline{\mathbf{F}}_\mathbf{N}}{\partial\overline{\mathbf{V}}_\mathbf{1}} & \cdots & \frac{\partial\overline{\mathbf{F}}_\mathbf{N}}{\partial\overline{\mathbf{V}}_\mathbf{N}} \end{bmatrix} \tag{57}$$

From the nonlinear models of transistors and diodes, the time-domain equations for $\frac{\partial i}{\partial v}$ and $\frac{\partial q}{\partial v}$ are usually known. To reuse these models into Equations 55 and 56, some manipulation using the DFT and the IDFT is required. First, to calculate $\overline{\mathbf{G}}$, voltage and current arrays can be converted from time to frequency and vice-versa using the DFT matrix transform,

$$\overline{\mathbf{V}}(\omega) = \overline{\mathbf{\Gamma}}\,v(t) \tag{58}$$

$$\overline{\mathbf{I}}(\omega) = \overline{\mathbf{\Gamma}}\,i(t) \tag{59}$$

$$v(t) = \overline{\mathbf{\Gamma}}^{-1}\,\overline{\mathbf{V}}(\omega) \tag{60}$$

$$i(t) = \overline{\mathbf{\Gamma}}^{-1}\,\overline{\mathbf{I}}(\omega) \tag{61}$$

where $\overline{\mathbf{\Gamma}}$ and $\overline{\mathbf{\Gamma}}^{-1}$ are $N(2K+1)\times N(2K+1)$ block-diagonal matrices formed, respectively, by $\mathbf{\Gamma}$ and $\mathbf{\Gamma}^{-1}$ along the main diagonal, and are responsible to apply the DFT and IDFT transforms nodewise. The time-domain signals, $v(t)$ and $i(t)$, have $N(2K+1)$ discrete time samples. The time-domain signals can be written at a single node n, for example the voltage $v_n(t) \in \mathbb{R}^{2K+1}$, is given by:

$$v_n(t) = \begin{bmatrix} v_n(t_0) & v_n(t_1) & \dots & v_n(t_{2K}) \end{bmatrix}^T \tag{62}$$

Omitting the time and frequency dependencies, the nonlinear current, which is voltage dependent, can be rewritten as,

$$\overline{\mathbf{I}}(\overline{\mathbf{V}}) = \overline{\mathbf{\Gamma}} \times i(v) = \overline{\mathbf{\Gamma}} \times i(\overline{\mathbf{\Gamma}}^{-1}\overline{\mathbf{V}}) \tag{63}$$

Applying the chain rule of differentiation to Equation 63, $\overline{\mathbf{G}}$ can be expressed in terms of the time-domain current derivative,

$$\overline{\mathbf{G}} = \frac{\partial \overline{\mathbf{I}}(\overline{\mathbf{V}})}{\partial \overline{\mathbf{V}}} = \overline{\mathbf{\Gamma}} \frac{\partial i}{\partial v} \overline{\mathbf{\Gamma}}^{-1} \tag{64}$$

where $\overline{\mathbf{G}}$ can be expanded as,

$$\overline{\mathbf{G}} = \begin{bmatrix} \mathbf{\Gamma}\left[\frac{\partial i_1}{\partial v_1}\right]\mathbf{\Gamma}^{-1} & \dots & \mathbf{\Gamma}\left[\frac{\partial i_1}{\partial v_N}\right]\mathbf{\Gamma}^{-1} \\ \vdots & \ddots & \vdots \\ \mathbf{\Gamma}\left[\frac{\partial i_N}{\partial v_1}\right]\mathbf{\Gamma}^{-1} & \dots & \mathbf{\Gamma}\left[\frac{\partial i_N}{\partial v_N}\right]\mathbf{\Gamma}^{-1} \end{bmatrix} \tag{65}$$

and and each submatrix $\left[\frac{\partial i_n}{\partial v_m}\right] \in \mathbb{R}^{2K+1}$, where n and m are node indexes, is written as,

$$\left[\frac{\partial i_n}{\partial v_m}\right] = \begin{bmatrix} \frac{\partial i_n(v_m(t_0))}{\partial v_m(t_0)} & \dots & \frac{\partial i_n(v_m(t_0))}{\partial v_m(t_{2K})} \\ \vdots & \ddots & \vdots \\ \frac{\partial i_n(v_m(t_{2K}))}{\partial v_m(t_0)} & \dots & \frac{\partial i_n(v_m(t_{2K}))}{\partial v_m(t_{2K})} \end{bmatrix} \tag{66}$$

Since the nonlinearities of both $i(v(t))$ and $q(v(t))$, are algebraic, i and q depend only on the voltage at a specific time. In other words, $i(v(t))$ and $q(v(t))$ at time t_k depend exclusively on $v(t_k)$. Due to this property, the $\left[\frac{\partial i_n}{\partial v_m}\right]$ matrix becomes diagonal,

$$\left[\frac{\partial i_n}{\partial v_m}\right] = \begin{bmatrix} \frac{\partial i_n(v_m(t_0))}{\partial v_m(t_0)} & 0 & \dots & 0 \\ 0 & \frac{\partial i_n(v_m(t_1))}{\partial v_m(t_1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\partial i_n(v_m(t_{2K}))}{\partial v_m(t_{2K})} \end{bmatrix} \tag{67}$$

The exact same analysis done for $\overline{\mathbf{G}}$ can be applied for the $\overline{\mathbf{C}}$ matrix, replacing current by charge. Therefore $\overline{\mathbf{C}}$ is given by:

$$\overline{\mathbf{C}} = \begin{bmatrix} \mathbf{\Gamma}\left[\frac{\partial q_1}{\partial v_1}\right]\mathbf{\Gamma}^{-1} & \dots & \mathbf{\Gamma}\left[\frac{\partial q_1}{\partial v_N}\right]\mathbf{\Gamma}^{-1} \\ \vdots & \ddots & \vdots \\ \mathbf{\Gamma}\left[\frac{\partial q_N}{\partial v_1}\right]\mathbf{\Gamma}^{-1} & \dots & \mathbf{\Gamma}\left[\frac{\partial q_N}{\partial v_N}\right]\mathbf{\Gamma}^{-1} \end{bmatrix} \tag{68}$$

where,

$$\left[\frac{\partial q_n}{\partial v_m}\right] = \begin{bmatrix} \frac{\partial q_n(v_m(t_0))}{\partial v_m(t_0)} & 0 & \dots & 0 \\ 0 & \frac{\partial q_n(v_m(t_1))}{\partial v_m(t_1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \frac{\partial q_n(v_m(t_{2K}))}{\partial v_m(t_{2K})} \end{bmatrix} \tag{69}$$

All the matrices required to calculate the expanded Jacobian, $\overline{\mathbf{J}}$, can be obtained as long as the voltage is known at all nodes. Converting Equation 47 from $\mathbb{C}$ to $\mathbb{R}^2$,

$$\overline{\mathbf{V}}^{(i+1)} = \overline{\mathbf{V}}^{(i)} - \overline{\mathbf{J}}(\overline{\mathbf{V}}^{(i)})^{-1}\overline{\mathbf{F}}(\overline{\mathbf{V}}^{(i)}) \tag{70}$$

Therefore, starting with a guess for $\overline{\mathbf{V}}^{(0)}$, the Jacobian $\overline{\mathbf{J}}^{(0)}$ is calculated using Equation 54 and the NR iteration can be applied to continuously improve the voltage guess until a solution is reached.

3.2.4.2 Termination Criteria

Convergence of the harmonic balance analysis is reached when the error obtained in $\mathbf{F}(\mathbf{V})$ can be neglected. Since current harmonics throughout the circuit and the spectrum can have vastly different magnitudes, an absolute and a relative stopping criteria are used for HB. If either of the criterias are satisfied for a harmonic component, this component is considered to have converged.

Defining the linear and nonlinear current contributions of HB based on Equation 44 as,

$$\mathbf{I_L} = \mathbf{YV} + \mathbf{I_s} \tag{71}$$
$$\mathbf{I_{NL}} = \mathbf{I}(\mathbf{V}) + j\mathbf{\Omega}\mathbf{Q}(\mathbf{V}) \tag{72}$$

the absolute and relative termination criterias are, respectively, given by:

$$\left|I_{L,n}(k) + I_{NL,n}(k)\right| < \epsilon_{abs} \qquad \forall \quad n,k \tag{73}$$
$$2 \cdot \left|\frac{I_{L,n}(k) + I_{NL,n}(k)}{I_{L,n}(k) - I_{NL,n}(k)}\right| < \epsilon_{rel} \qquad \forall \quad n,k \tag{74}$$

where $I_{L,n}(k), I_{NL,n}(k) \in \mathbb{C}$ are the linear and nonlinear current phasors for node n at frequency harmonic k.

Examining the termination equations, the absolute criteria alone may create unrealistic requirements for the convergence of large magnitude currents if a small value of ϵ_{abs} is required. Meanwhile, the relative criteria has the opposite effect, as it can create too strict convergence conditions for very small currents. Also, the relative criteria alone provides very little information about the convergence progress during the initial steps, since for large errors it converges to ± 2 (MAAS, 2003).

3.2.5 HB for Multi-Tone Excitation

The derivation of HB considered a periodic solution and a set of frequencies consisting only of harmonics of the single-tone excitation at ω_0. To expand the power of HB, it is necessary to add support for multi-tone analysis and quasiperiodic solutions. Multi-tone HB analysis is used to obtain the intermodulation distortion (IP2 and IP3) in power amplifiers and the intermodulation products of mixers in the presence of two different large-signal excitations.

Although only the frequency set $\lambda : \lambda = k\omega_0,\ k \in \mathbb{Z}$ and $0 \leq k \leq K$ was considered in the formulation of HB presented, nothing limited its use to commensurate frequencies. This set is convenient since the direct approach to the DFT can be used with equally spaced time samples. But as long as a time-frequency transformation exists, the frequency set used to calculate the harmonic balance matrices may have other formats.

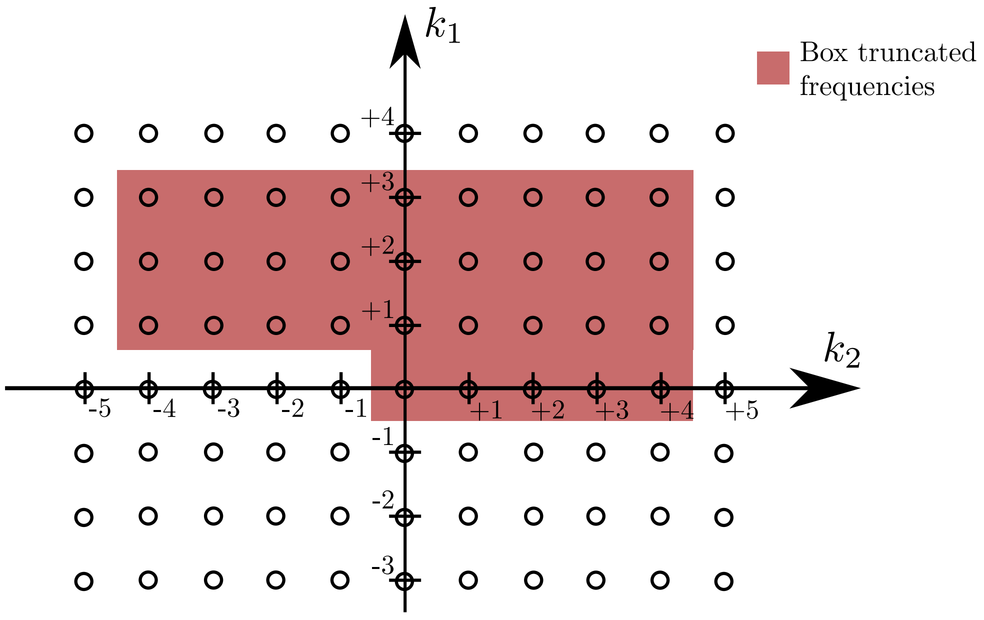
A circuit that is excited by multiple frequencies generates an infinite amount of intermodulation products with increasingly smaller amplitudes. Therefore, the first step for the generalization of the HB algorithm to multiple input tones, is to create the set of frequencies of interest, by truncating the number of intermodulation products that are generated. A common approach to generate this frequency set, and the one chosen for implementation, is the box truncation scheme.

The second step for multi-tone support is to determine a variation of the Fourier Transform that can be used in this, now unevenly spaced, grid of frequencies. Evenly spaced time samples create DFT/IDFT matrices, $\mathbf{\Gamma}$ and $\mathbf{\Gamma}^{-1}$, that have perfectly orthogonal rows, making the matrix well-conditioned with almost no error introduced due to the time-frequency transformations. Without harmonically related frequencies, a different method to choose the time samples is required, to avoid aliasing and other issues. A paper by Kundert et al. (1988) introduces the concept of the Almost-Periodic Fourier Transform (APFT), which is an elegant method that takes an oversampled set of time points, and via an orthogonalization procedure, selects a $2K+1$ set of points, that results in the most well-conditioned $\mathbf{\Gamma}$ matrix. This method described in the paper gives accurate results and reviews the issues of Fourier transforming quasiperiodic signals. Nonetheless, the method chosen for this implementation is the so called Artificial Frequency Mapping (AFM), which uses the simple one-dimensional DFT with a distorted frequency axes, and has overall better performance. The main drawback is that the intermediate time-domain waveforms of this technique have no physical meaning, but that is a small problem since the time-domain signal can be reconstructed from the spectrum at the end of the simulation.

3.2.5.1 Box Truncation Scheme

Considering a two-tones response, where the non commensurate excitation frequencies are ω_1 and ω_2, the intermodulation products $k_1\omega_1 + k_2\omega_2$ for $k_1, k_2 \in \mathbb{Z}$, can be represented in a two-dimensional plan, as shown in Figure 10.

Figure 10 – Example of box truncation for $K_1 = 3$ and $K_2 = 4$.



Source: Author.

The definition of the set of box truncated frequencies is then given by:

$$\Lambda_K = \{\lambda : \lambda = k_1\omega_1 + k_2\omega_2;\ 0 \le k_1 \le K_1,\ |k_2| \le K_2,\ k_1 \ne 0 \text{ if } k_2 < 0\} \tag{75}$$

where K is the number of non-zero frequencies,

$$K = \frac{1}{2}((2K_1 + 1)(2K_2 + 1) - 1) \tag{76}$$

The parameters K_1 and K_2 are truncation indexes for each frequency and the other restrictions in Equation 75 remove redundant frequency components from the set Λ_K (see Figure 10 for an example). It is straightforward to expand the box truncation for more than two tones.

The truncated Fourier series over the new frequency set Λ_K can be generalized as:

$$x(t_s) = X(0) + \sum_{\substack{\lambda_k \in \Lambda_K \\ \lambda_k \ne 0}} X^C(k)\cos(\lambda_k t_s) + X^S(k)\sin(\lambda_k t_s) \tag{77}$$

3.2.5.2 Artificial Frequency Mapping

The most important observation to understand the AFM technique, is the fact that the Fourier coefficients of a memoryless device (algebraic nonlinearity) do not depend on the frequency (KUNDERT; WHITE; SANGIOVANNI-VINCENTELLI, 1990). Considering a device described by:

$$i(v) = v + v^2 \tag{78}$$

which contains a very simple algebraic nonlinearity. Exciting this device with a two-tones input such as,

$$v(t) = V_1 \cos(\omega_1 t) + V_2 \cos(\omega_2 t) \tag{79}$$

the resulting current is written as,

$$\begin{aligned} i(v) = &\frac{(V_1^2 + V_2^2)}{2} + V_1 \cos(\omega_1 t) + V_2 \cos(\omega_2 t) \\ &+ \frac{V_1^2}{2} \cos(2\omega_1 t) + \frac{V_2^2}{2} \cos(2\omega_2 t) \\ &+ \frac{V_1 V_2}{2} [\cos((\omega_1 + \omega_2)t) + \cos((\omega_1 - \omega_2)t)] \end{aligned} \tag{80}$$

and the presence of intermodulation products can be seen at the output. But what is important for the AFM, is the observation that the Fourier coefficients in Equation 80 are all independent of ω_1 and ω_2. This is always true for algebraic nonlinearities. Therefore, the main insight of the AFM technique is that, if the Fourier coefficients, do not depend on the actual frequency, one can choose a convenient frequency value, λ_0, to perform the DFT, over which the set of frequencies Λ_K is actually dense and periodic. That way equally spaced time samples can be chosen and the traditional one-dimensional DFT can be used. This will have the effect that during HB transformations into time-domain to evaluate the nonlinear devices will have a distorted time axis and no actual physical meaning. Nonetheless, at the end of a simulation, the trigonometric form of the Fourier coefficients obtained can be used to calculate the correct time-domain response.

For a box-truncated set, applying the scaling factors α_1 and α_2 to the terms $k_1\omega_1$ and $k_2\omega_2$, respectively, creates a new frequency set which is uniform, where,

$$\alpha_1 = 1 \tag{81}$$

$$\alpha_2 = \frac{\omega_1}{\omega_2(2K_2 + 1)} \tag{82}$$

and,

$$k\lambda_0 = k_1\alpha_1\omega_1 + k_2\alpha_2\omega_2 \tag{83}$$

It is worth noting that the artificial frequency set is only required for the transformations between the time and frequency domains. The remaining of the HB algorithm, which is performed in the frequency-domain, is entirely done over the set Λ_K, including the assemble of the admittance matrix $\overline{\mathbf{Y}}$. Another important implementation detail is the fact that some intermodulation products inside Λ_K may have negative frequency values. This case is treated by taking always the absolute value of the frequency to perform the calculations, and after the DFT is applied, the complex conjugate Fourier coefficient of those terms is used. This approach works since the time-domain signal is real and therefore, $X(-\lambda_k) = X^*(\lambda_k),\ 1 \leq k \leq K$. The current implementation was restrained to the two-tone case presented, but a generalized form of this technique for a larger number of tones is shown by Rodrigues (1997) for the box-truncation case.

3.2.6 Including Autonomous Circuit Support

To perform a HB simulation the fundamental frequency must be provided by the user. In the case of autonomous circuits, the exact frequency of oscillation is not known, since only estimates can be made from hand calculations. Besides, even if the frequency basis is known exactly and set on the simulation, autonomous circuits frequently display another mathematical solution which contains no oscillatory behavior: the DC solution. Therefore, the initial condition provided to Newton-Raphson must be attracted to the oscillatory regime, otherwise the algorithm might converge to the degenerate DC solution. Providing this sufficiently-close-to-the-desired-solution initial condition is not trivial.

A few methods to solve this problem can be found in textbooks such as Kundert et al. (1990) and Suárez (2009). The mixed harmonic balance formulation, includes the frequency as a state variable to be determined during the Newton-Raphson iteration. First, to solve for free-running oscillators, the HB formulation in Equation 44 needs to be rewritten considering the fundamental frequency, ω, as also a variable:

$$\mathbf{F}(\mathbf{V},\omega) = \mathbf{I}(\mathbf{V}) + \mathbf{\Omega}(\omega)\mathbf{Q}(\mathbf{V}) + \mathbf{Y}(\omega)\mathbf{V} + \mathbf{I_S} = \mathbf{0} \tag{84}$$

Or more compactly:

$$\mathbf{F}(\mathbf{V},\omega) = \mathbf{0} \tag{85}$$

This formulation has the issue that there are a number of possible solutions for $\mathbf{V}$, since any time-shifted version of the solution of an oscillator is correct. Since there is no isolated solution, Newton-Raphson fails to converge. A simple method to constraint this problem and isolate the solutions, is to force some signal in the circuit to have its sinusoidal part of the fundamental equal to zero. That translates to,

$$\mathbf{V}_n^S(1) = 0 \tag{86}$$

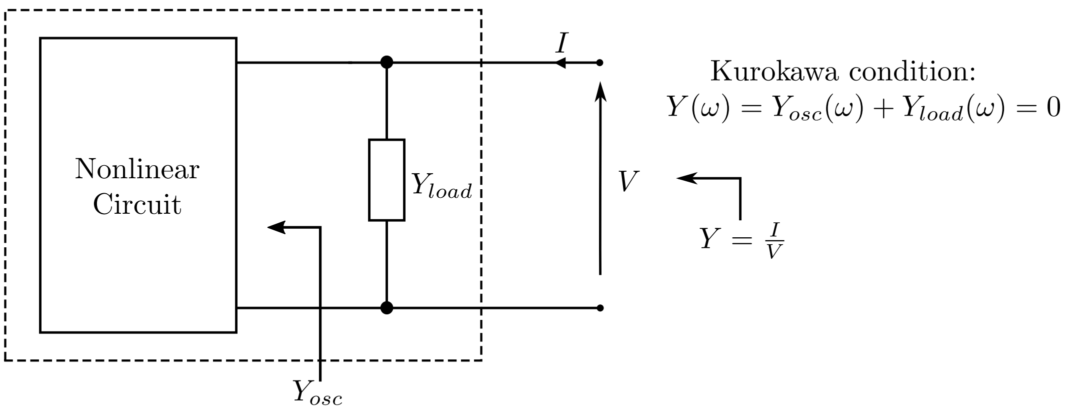
where n is the node number and S indicates the sinusoidal part only of $\mathbf{V}_n$. Applying Newton-Raphson to this system of equations (KUNDERT; WHITE; SANGIOVANNI-VINCENTELLI, 1990):

$$\begin{bmatrix} \mathbf{J}(\mathbf{V}^{(i)},\omega^{(i)}) & \frac{\partial \mathbf{F}(\mathbf{V}^{(i)},\omega^{(i)})}{\partial \omega} \\ \mathbf{e}_n^S(1) & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{V}^{(i+1)} \\ \Delta\omega^{(i+1)} \end{bmatrix} = -\begin{bmatrix} \mathbf{F}(\mathbf{V}^{(i)},\omega^{(i)}) \\ \mathbf{V}_n^S(1) \end{bmatrix} \tag{87}$$

where $\mathbf{e}_n^S(1)$ is a vector to select the first harmonic sinusoidal part of the voltage in node n.

This approach suffers from stability issues and is strongly dependent on initial condition as it tends to converge often to the degenerate solution. This can be improved with techniques such as the ones proposed in Xuan and Snowden (1987) and Elad et al. (1989). The method proposed by Chang et al. (1991) partially integrates Kurokawa's oscillation criteria to the formulation above, to ensure that the Newton iteration doesn't converge to the trivial DC solution. Figure 11 shows the one-port equivalent circuit of an oscillator used to check for

Figure 11 – One-port oscillator analysis



Source: Author.

the oscillation criteria. The idea is to include the Kurokawa condition for the fundamental oscillating frequency only, as a constraint to the Newton-Raphson.

Although the introduced methods work, the method chosen for YalRF to perform the HB analysis of oscillators is presented by Quere et al. (1993) and Ngoya and Suárez et al. (NGOYA; SUÁREZ, et al., 1995). Suárez (SUÁREZ, 2009) calls it the Auxiliary Generator Technique. Its working principle is simple and presented good results for YalRF so far. The fact that is used in commercial engines was also a strong motive for this choice. The technique is centered around the inclusion of an auxiliary generator to the circuit, used to force harmonic balance to converge towards a desired solution. That way, the issue with the absence of an excitation frequency in autonomous circuits is resolved, and the traditional HB algorithm can be employed. The artificial generator, also called an oscillator probe, will force the convergence towards an oscillatory regime, excluding the DC solution. The question remains how to find the correct frequency of oscillation and excitation magnitude that must be used, so that this auxiliary generator does not perturb the natural response of the oscillator.

Figure 12 shows the oscillator probe, consisting of a voltage generator, with excitation frequency ω_{osc}, and voltage amplitude V_{osc}, in series with an ideal impedance filter, which has zero impedance for the excitation frequency and infinite impedance for all other values. The use of an analogous current probe is also possible.

The auxiliary generator signal can be written as,

$$v_{osc}(t) = \mathrm{Re}\Big\{|V_{osc}|e^{j(\omega_{osc}t+\phi_{osc})}\Big\} \tag{88}$$

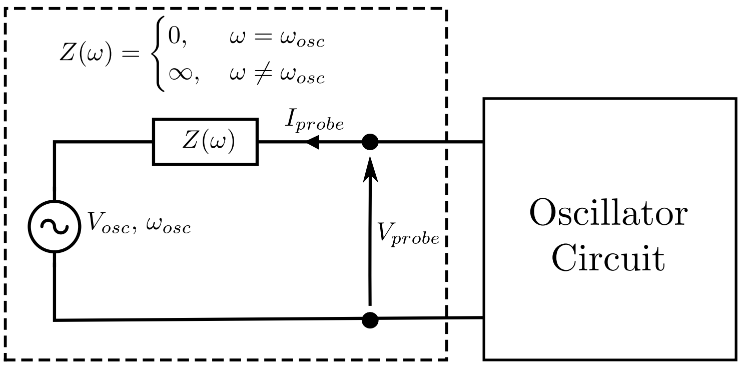
or in phasor representation,

$$V_{osc}(\omega_{osc}) = |V_{osc}|e^{j\phi_{osc}} \tag{89}$$

As there is a generator with a defined frequency ω_{osc}, the HB formulation of the oscillator circuit with the attached probe can be written in the short format,

$$\mathbf{F}(\mathbf{V}) = \mathbf{0} \tag{90}$$

Figure 12 – Auxiliary Generator Voltage Probe


$Z(\omega) = \begin{cases} 0, & \omega = \omega_{osc} \\ \infty, & \omega \neq \omega_{osc} \end{cases}$
$Z(\omega)$
I_{probe}
V_{osc}, ω_{osc}
V_{probe}
Oscillator Circuit

Source: Author.

without the frequency as a variable. Due to the existence of the ideal harmonic filter on the probe, the first harmonic of V_{probe} is equal to $V_{osc}(\omega_{osc})$, while the other harmonics are generated by the nonlinear circuit, since the probe appears like an open circuit.

The core idea of the method relies on the fact that the probe existence must not disturb the steady-state regime of the oscillator. That can be ensured by making the admittance of the fundamental frequency of the probe equal to zero, as the probe already does not disturb the other harmonics. Therefore:

$$Y_{probe}(|V_{osc}|, \phi_{osc}, \omega_{osc}) = \frac{I_{probe}(\omega_{osc})}{V_{probe}(\omega_{osc})} = 0 \tag{91}$$

where the value of I_{probe} is a product of the HB simulation, while the auxiliary generator $|V_{osc}|$, ϕ_{osc} and ω_{osc} variables must be found in order to satisfy Equations 91. This problem has 3 unknowns and only 2 equations, considering the real and imaginary parts of Y_{probe}. As already discussed, the phase is not important to an oscillator response, as the time origin does not influence the spectrum response. Therefore, this problem can be reduced to 2 unknowns by making $\phi_{osc} = 0$.

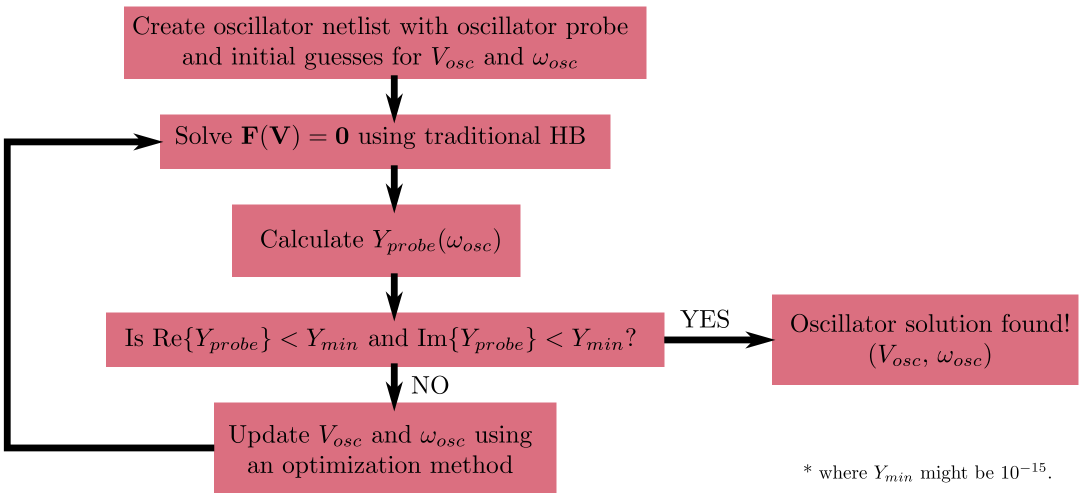
The final nonlinear system writes,

$$\begin{cases} Y_{probe}(|V_{osc}|, \omega_{osc}) = \mathbf{0} \\ \mathbf{F}(\mathbf{V}) = \mathbf{0} \end{cases} \tag{92}$$

with a well-conditioned problem of $N + 2$ equations and $N + 2$ variables, where N here is the size of the HB problem. As this is a root-finding problem, Newton's method can be applied to solve simultaneously for both equations in 92. A different approach is to use a two-tier optimization procedure, where the inner-tier solves the HB problem given the currently known values for $|V_{osc}|$ and ω_{osc}, while the outer-tier is responsible to improve the guesses of $|V_{osc}|$ and ω_{osc}, until Equation 91 is satisfied.

To evaluate different optimization methods and their performance at the outer-tier level, YalRF uses a two-tier optimization to obtain the values of $|V_{osc}|$ and ω_{osc}. The main idea of the method is shown in Figure 13.

Figure 13 – Fluxogram of the oscillator


Create oscillator netlist with oscillator probe and initial guesses for V_{osc} and ω_{osc}
Solve $\mathbf{F}(\mathbf{V}) = \mathbf{0}$ using traditional HB
Calculate $Y_{probe}(\omega_{osc})$
Is $\mathrm{Re}\{Y_{probe}\} < Y_{min}$ and $\mathrm{Im}\{Y_{probe}\} < Y_{min}$?
YES
Oscillator solution found! (V_{osc}, ω_{osc})
NO
Update V_{osc} and ω_{osc} using an optimization method
* where Y_{min} might be 10^{-15}.

Source: Author.

3.2.7 Improvements in Convergence

3.2.7.1 Initial Guess

The first problem to reach a HB solution, is to provide a good initial condition. Since the problem of HB is nonlinear, having a good initial condition is sometimes critical for the Newton-Raphson iteration to converge for the desired response. Fortunately, simple methods like starting from the circuit DC solution, very often provide good enough guesses. Other options are to use all zeroes or the AC response as the initial condition. Yet another approach used for slightly more complex cases is to run a few cycles of transient simulation to use as the starting point. This is called transient-assisted HB. Currently YalRF uses the DC solution as initial condition for HB, but the all zeroes and AC response start guesses were also explored during simulations.

3.2.7.2 Source Stepping

For cases where the input tone has a large magnitude and/or strong nonlinearities are in place, HB may present trouble to converge. Therefore, the source stepping continuation method can be once again applied. The main idea of a continuation method was already presented in Chapter 2, and consists of solving a sequence of problems which work as good initial conditions for the next one, until they solve the desired circuit. For source stepping, this is done by slowly increasing the vector of inputs until their magnitudes reach the expected level. Solving increasingly harder problems with also increasingly better initial estimates works very well for HB and is used inside YalRF.

4 IMPLEMENTING A CIRCUIT SIMULATION SOFTWARE: YALRF

This chapter introduces YalRF, a circuit simulation engine developed in Python. As previously explained, its main goal is to be able to simulate autonomous circuits using the Harmonic Balance method. Also, it work as necessary groundwork for future study and research into nonlinear simulation methods and device modelling for EDA.

4.1 PROGRAMMING TOOLS

The programming language of choice was Python mostly due to its enormous popularity. Because of its large employment in areas such as machine learning and data science, an abundant amount of resources are available for Python developers/learners which makes it more approachable for other people to eventually contribute to this project. Also, Python enables the writing of very readable and clean code, which invites new users to understand the inner workings of a circuit simulation engine.

The scripting characteristic makes it easy to test/debug the Python code and also makes it much simpler to create a running environment which works in multiple operating systems. If this engine was written in C/C++, a significant amount of effort would go to manage dependencies, libraries and creating deploy scripts. Meanwhile, YalRF was developed inside a Conda environment using Miniconda (ANACONDA INC., n.d.). So any user which installs Conda can easily create a Python environment compatible with the one YalRF was developed and quickly start running code using YalRF or edit its source code.

At the current time, the only dependencies of YalRF are Numpy and Scipy, libraries for scientific computing which contains linear algebra and optimization algorithms, and also Matplotlib, a library for plotting data.

Finally, the existence of three other tools: scikit-rf (ARSENOVIC, 2009), SignalIntegrity (PUPALAIKIS, 2018) and openEMS (LIEBIG et al., 2013) was also relevant for the use of Python in this project. scikit-rf and SignalIntegrity are written in Python, while openEMS has a Python API available. scikit-rf provides an easy interface to operate with S-Parameters, calibration algorithms and deembeding. SignalIntegrity is more focused on, as the name suggests, signal integrity problems and has powerful tools to perform time-domain reflectometry analysis, which is very important in high-speed digital communications. openEMS is a powerful finite-difference time-domain 3D electromagnetic solver and can export S-Parameters of the simulated structures. Those tools combined create a very powerful framework for analysis of RF systems and lack only on the ability to evaluate nonlinear devices. That is a role that could be filled by YalRF.

The major penalty for using Python is, of course, the execution time. Many benchmarks are available comparing Python to other programming languages more naturally suitable for scientific computing and Python usually lags by large margins. Due to the research nature of this project the performance was not a top requirement for the current implementation, although with the stabilization of the engine, there is a desire to port its core into a faster programming language.

4.2 OVERVIEW OF IMPLEMENTED ALGORITHMS AND DEVICES

YalRF has the following features currently implemented:

- Nonlinear DC analysis with g_{min} and source stepping continuations methods;
- AC analysis using the DC calculated operating point;
- Transient simulations using Euler and Trapezoid integration methods;
- Linear devices: resistor, capacitor, inductor, VCVS, VCCS, CCVS, CCCS, DC block, DC feed, gyrator;
- Supplies: DC and AC voltage and current sources, time-domain pulsed voltages for transient simulation;
- Nonlinear devices: Gummel-Poon BJT model (GUMMEL; POON, 1970), Diode, Opamp with finite gain and smooth saturation, Schichman-Hodges MOSFET (SHICHMAN; HODGES, 1968);
- Harmonic Balance analysis with support to two-tones using artificial frequency mapping and oscillator circuits using the auxiliary generator technique;

4.2.1 DC Analysis

Algorithm 1 describes the main function of the implemented DC analysis. Currently there is a sequence of attempts to solve the DC problem. The algorithm starts by checking if the netlist contains only linear elements. If that is the case, a simple linear equation problem needs to be solved and a standard LU decomposition is used. Otherwise, the Newton-Raphson iteration must be applied to deal with the nonlinear elements. If a simple Newton-Raphson run fails to reach the DC solution, the source and g_{min} stepping continuation methods are employed in sequence to increase chances of convergence.

In Algorithm 2, the methods used for the nonlinear DC analysis are shown. A lot of effort was placed to improve the convergence stability of the DC analysis. The DC problem can be highly nonlinear and often starts with very poor initial guesses for X_0, usually zeroes. Therefore, the inclusion of the source and g_{min} continuation methods along with limiting algorithms for the nonlinear devices was tremendously important to improve the convergence of this analysis.

Algorithm 1 DC Analysis

```
Input: netlist, $X_0$
Output: $X$ // Calculated node voltages and selected currents
 1: $A$, $Z \leftarrow$ add stamps from linear elements $\in$ netlist
 2: if $n \in [1, N]$ is a critical node then
 3:     $A[n,n] \leftarrow A[n,n] + g_{min}$
 4: if netlist is linear then
 5:     $X \leftarrow A^{-1}Z$
 6:     return $X$
 7: else
 8:     $X$, issolved $\leftarrow$ solve_dc_nonlinear($A$, $Z$, $X_0$)
 9:     if issolved then
10:         return $X$
11:     $X$, issolved $\leftarrow$ solve_with_source_stepping($A$, $Z$, $X_0$)
12:     if issolved then
13:         return $X$
14:     $X$, issolved $\leftarrow$ solve_with_gmin_stepping($A$, $Z$, $X_0$)
15:     if issolved then
16:         return $X$
17:     return failed to converge
```

4.2.2 AC Analysis

The AC analysis pseudocode is presented in Algorithm 3. The solution for the AC problem is very similar to the linear DC problem, the most troublesome part being creating the linearized models for the nonlinear devices. For a simple diode, the AC model has to add at least the junction capacitance, parasitic resistance and linearized conductance, while for more complex transistor models, several nonlinear capacitances and conductances must be linearized at the desired operating point before the model can be used. After all models are properly linearized and their values used to stamp the complex admittance matrix, a linear system of equations is solved using standard LU factorization.

4.2.3 Transient Analysis

The transient analysis was implemented as shown in Algorithm 4. Currently it is missing some more sophisticated features such as Gear and Adams integration methods, local truncation error estimation for adaptive time step control and predictor-corrector methods. The dynamic elements stamps are calculated using implicit Euler or the Trapezoidal method. Regardless, the algorithm works fine for well behaved circuits and the time step is actually controlled by the number of iterations that Newton-Raphson takes to converge. Too many iterations or failure of convergence is an indication that the time step is too large and should be reduced. On the other hand, too fast convergence indicates that the time step can probably be relaxed without losing stability or accuracy.

Algorithm 2 DC Analysis complementing methods

```
solve_dc_nonlinear($A$, $Z$, $X_0$):
while not converged do
  $A$, $Z$ ← add NR stamps calculated for nonlinear elements using $X_{prev}$
  $X \leftarrow A^{-1}Z$
  if $X - X_{prev}$ has converged then
    return $X$, true
  else
    $X_{prev} \leftarrow X$

solve_with_source_stepping($A$, $Z$, $X_0$):
$\alpha \leftarrow 0.01$
while not converged do
  $Z \leftarrow \alpha * Z$ // scaling of input vector
  $X$, issolved ← solve_dc_nonlinear($A$, $Z$, $X$)
  if issolved then
    $X_{prev} \leftarrow X$ // backup good solution
    if $\alpha \geq 1$ then
      return $X$, true
    else
      increase $\alpha$
  else
    $X \leftarrow X_{prev}$ // restore good solution
    decrease $\alpha$

solve_with_gmin_stepping($A$, $Z$, $X_0$):
$g_{min} \leftarrow 0.01$
while not converged do
  $A$ ← add shunt admittance gmin from every node to ground
  $X$, issolved ← solve_dc_nonlinear($A$, $Z$, $X$)
  if issolved then
    $X_{prev} \leftarrow X$ // backup good solution
    if $g_{min} < 1e-12$ then
      return $X$, true
    else
      reduce $g_{min}$
  else
    $X \leftarrow X_{prev}$ // restore good solution
    increase $g_{min}$
```

Algorithm 3 AC Analysis

```
Input: netlist, frequencies, $X_0$
Output: $X$ // Calculated node voltages and selected currents at all frequencies
 1: $X_{DC} \leftarrow$ run DC simulation to obtain the operating point
 2: for all nonlinear devices $\in$ netlist do
 3:     calculate AC model of the device at $X_{DC}$ operating point
 4: for all $f \in$ frequencies do
 5:     $A$, $Z \leftarrow$ add ac stamps for all devices calculated at frequency $f$
 6:     $X[f] \leftarrow A^{-1} * Z$
 7: return $X$
```

Algorithm 4 Transient Analysis

```
Input: netlist, $t_{stop}$, $X_0$
Output: $X$ // Calculated node voltages and selected currents at all time points
 1: if $X_0$ then
 2:     $X[t_0] \leftarrow X_0$
 3: else
 4:     $X[t_0] \leftarrow$ run DC simulation to obtain the initial operating point
 5: for all nonlinear devices $\in$ netlist do
 6:     initialize nonlinear devices at $X[t_0]$ operating point
 7: while $t \leq t_{stop}$ do
 8:     $t \leftarrow t + t_{step}$
 9:     $X_k \leftarrow X[t_k - 1]$ // start next step from previous time point
10:     // NR iteration to solve implicit integration
11:     while not converged do
12:         $A$, $Z \leftarrow$ add transient stamps for all devices using $X_k$ as attempted solution
13:         $X[t_k] \leftarrow A^{-1} * Z$
14:         if $X[t_k] - X_k$ has converged then
15:             converged $\leftarrow$ true
16:             break
17:         else
18:             $X_k \leftarrow X[t_k]$
19:     if converged then
20:         $X[t_k] \leftarrow$ save solution vector at time $t_k$
21:         $t_{step} \leftarrow$ adjust time step according to number of NR iterations
22:     else
23:         $t \leftarrow t - t_{step}$ // return in time before decreasing time step
24:         $t_{step} \leftarrow$ reduce time step due to convergence failure
25: return $X$
```

4.2.4 Harmonic Balance Analysis

Algorithm 5 present the overall idea behind the implementation of the Harmonic Balance algorithm. It starts by generating the frequency basis over which the simulation is performed. If it is a single tone analysis, the basis is simply formed by the multiples of the fundamental frequency set by the user, limited by the chosen number of harmonics. In a two-tones case, the frequency basis is generated using box-truncation, which contains the harmonics of both fundamentals, along with the intermodulation products generated by their combinations. The creation of the DFT/IDFT and $\overline{\Omega}$ matrices is straightforward after the frequency basis is known. The $\overline{I_s}$ array is created taking the DC and AC current sources from the circuit and properly stamping the array. The frequency of the AC currents must be checked against the fundamental tones set on the simulation, since there can be no excitation frequency outside the frequency basis. The admittance matrix, $\overline{Y}$, is filled with all the linear devices stamps, similarly to what is performed for the AC simulation, with the admittances being calculated at all frequencies. Lastly for the setup phase, an initial condition given by the user or based on the DC solution of the circuit, is used to initialize the voltage array.

The Newton-Raphson iteration then follows what was presented in Chapter 3. First, the voltage candidate solution is transformed to the time-domain to evaluate the nonlinear devices. The currents, charges and their derivatives obtained are converted back to frequency-domain to assemble the Jacobian matrix. Using only the currents and charges the algorithm checks for convergence and the final step is to update the voltage spectrum using the calculated Jacobian in case a new iteration is required. An outer step that is not shown in Algorithm 5 for simplicity, is that the HB implementation of YalRF also employs source-stepping to improve convergence if the first Newton-Raphson attempt fails. The idea implemented is the same as the one shown in Algorithm 2.

Finally, Algorithm 6 is used to obtain the steady-state response of an oscillator using HB. The first step is to insert the oscillator probe, consisting of an ideal harmonic filter and an AC voltage source, at the node given by the user. To employ solvers from the Scipy optimization library an objective function must be defined. The goal of this objective function is to take a guess for the frequency and magnitude of the oscillation on the probe node, apply it to a harmonic balance simulation, and return the total probe admittance. The optimization algorithm of choice will have to successively improve its guesses until the probe admittance is minimized below an acceptable value (it currently defaults to 10^{-12}). As discussed in Chapter 3, when the admittance of the probe converges, the probe presents virtually no influence to the oscillator response, meaning that the voltage being applied by the probe actually matches the natural voltage at which the node oscillates. Currently the oscillator analysis defaults to the **fmin** function from Scipy's optimization library, which employs the Nelder-Mead simplex algorithm. The minimization is being performed over the magnitude of the admittance and is unconstrained. This method has proved enough for multiple oscillator structures tested, but a wide variety of optimization methods are available in Scipy and shall be explored.

Algorithm 5 Harmonic Balance Analysis

```
Input: netlist, $f_1$, $f_2$, $K_1$, $K_2$, $V_0$
Output: $V$ // Calculated voltage phasors at all nodes for all frequencies
 1: $\lambda \leftarrow$ create frequency grid using box-truncation
 2: $\overline{\Gamma}$, $\overline{\Gamma}^{-1} \leftarrow$ create DFT/IDFT matrices based on $\lambda$
 3: $\overline{\Omega} \leftarrow$ create frequencies matrix based on $\lambda$
 4: $\overline{I_s} \leftarrow$ create independent current sources vector
 5: $\overline{Y} \leftarrow$ create linear admittance matrix
 6: $\overline{V} \leftarrow$ create initial guess using DC solution
 7: while true do
 8:     $v \leftarrow \overline{\Gamma}^{-1} * \overline{V}$
 9:     for all nonlinear devices $\in$ netlist do
10:         $i$, $q$, $\frac{\partial i}{\partial v}$, $\frac{\partial q}{\partial v} \leftarrow$ calculate using nonlinear model equations and $v$ for all time samples
11:         $\frac{\partial I}{\partial V} \leftarrow \Gamma * \text{diag}(\frac{\partial i}{\partial v}) * \Gamma^{-1}$
12:         $\frac{\partial Q}{\partial V} \leftarrow \Gamma * \text{diag}(\frac{\partial q}{\partial v}) * \Gamma^{-1}$
13:     $\overline{J} \leftarrow \overline{Y} + \frac{\partial \overline{I}}{\partial \overline{V}} + \overline{\Omega}\frac{\partial \overline{Q}}{\partial \overline{V}}$ // Jacobian matrix
14:     $\overline{I}_L \leftarrow \overline{YV} - \overline{I_s}$ // current from linear elements
15:     $\overline{I}_{NL} \leftarrow \overline{\Gamma} * i + \overline{\Omega} * \overline{\Gamma} * q$ // current from nonlinear elements
16:     $\overline{F} \leftarrow \overline{I}_L + \overline{I}_{NL}$ // error function
17:     if $\overline{F}$ has converged then
18:         $V \leftarrow \overline{V}$ // convert from $\mathbf{R}^2$ to phasor representation
19:         return $V$, true
20:     $\Delta\overline{V} \leftarrow \overline{J}^{-1}\overline{F}$ // calculate NR step
21:     $\overline{V} \leftarrow \overline{V} - \Delta\overline{V}$
```

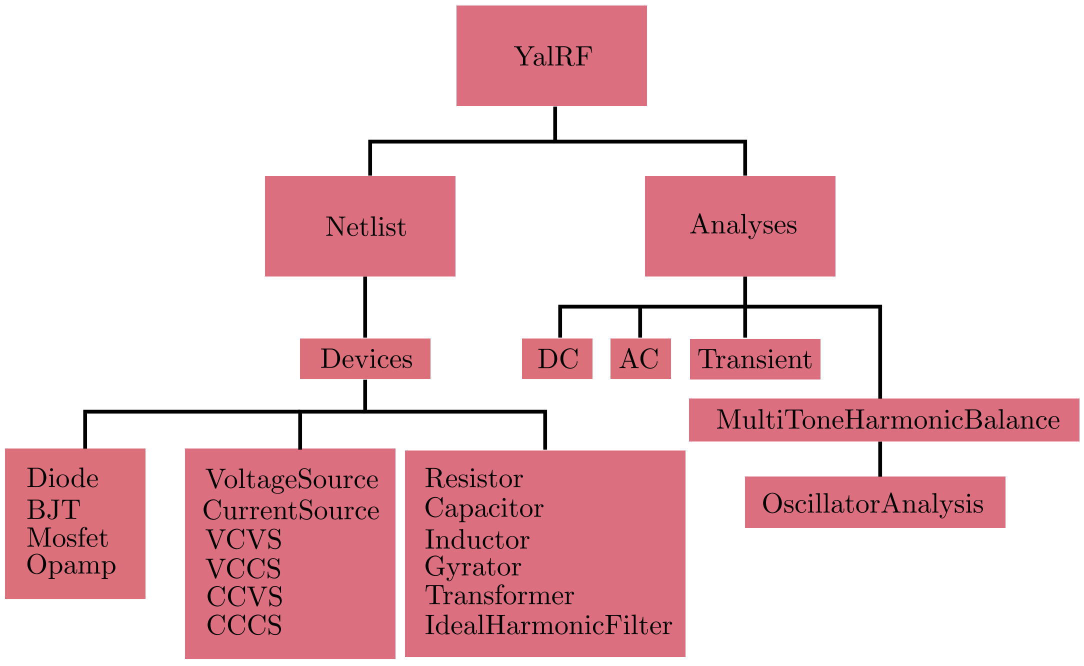
Algorithm 6 HB Oscillator Analysis

```
Input: netlist, osc_node, $f_{guess}$, $V_{guess}$, $K$
Output: $f_{osc}$, $V_{osc}$, $V$
 1: netlist $\leftarrow$ insert oscillator probe to the osc_node
 2: $f_{osc}$, $V_{osc} \leftarrow$ optimize($f_{guess}$, $V_{guess}$, objective_function)
 3:
 4: objective_function($f_{osc}$, $V_{osc}$):
 5: probe filter $\leftarrow f_{osc}$ // set frequency of the probe ideal harmonic filter
 6: probe source $\leftarrow f_{osc}$, $V_{osc}$ // set frequency and amplitude of the probe input tone
 7: $V \leftarrow$ run_HB($f_{osc}$)
 8: $Y \leftarrow$ calculate oscillator probe admittance using $V$
 9: return $Y$
```

4.3 YALRF CODE ORGANIZATION

The repository structure of YalRF[1] is quite straightforward. There are three main directories: one for future documentation, one for test scripts and one for the engine source code. Inside the tests directory there are Jupyter Notebooks with examples of circuits in YalRF, along with netlists used for simulations inside Xyce. The results from Xyce are used for comparison and to verify the proper working of YalRF's algorithms.

Figure 14 – YalRF Code Organizational Structure



Source: Author.

The main source code is organized in a hierarchical manner according to whats is depicted in Figure 14. Only very simple object orientation practices were employed, since the code base needed to remain flexible to the perspective of a lot of experimental features still being integrated. There is a top-level class named **YalRF**, which inherits the API used to create netlists and has methods to declare and run circuit analyses. It also contains helper functions used to access data, plot graphs and manage the netlist.

The **Netlist** class is responsible for holding the graph data structure describing the circuit topology. It holds information regarding what devices are connected to which nodes and also associates node names to node indexes used during MNA matrix formulation. Once a device is included to the netlist, an object representing this device is returned to the user and may have its parameters modified dynamically so that, for example, parameter sweeps can be performed.

[1] Please find YalRF at: `https://github.com/victorpreuss/YalRF`. All the testbenches used for the results section are also in the repository.

The **Analyses** directory on the diagram contains the implemented algorithms for each analysis currently supported. The creation of an abstract class for the analyses may be considered later on to standardize the way the user interacts with the algorithms. The **Devices** directory encapsulates all the implemented components so far. For the **Devices** case, the creation of a parent abstract class, **Device**, establishing a standardized API to interface with all the individual components will be an improvement to the source code in the future. This will allow the simulation algorithms to treat devices in a more agnostic fashion than it is currently possible.

The components placed below the **Devices** directory have a shared API used basically for stamping the MNA matrices using the appropriate model equations. For example, the BJT Gummel-Poon equations (JAHN et al., 2007) are placed inside the **BJT.py** file and have very different stamping depending on the analysis. This approach makes it fairly easy to include new components to YalRF, since all that is required is to create a model and define the stamp matrix for all the analyses of interest. The stamp matrices implemented were mostly obtained from the Qucs documentation (JAHN et al., 2007) and McCalla (1988).

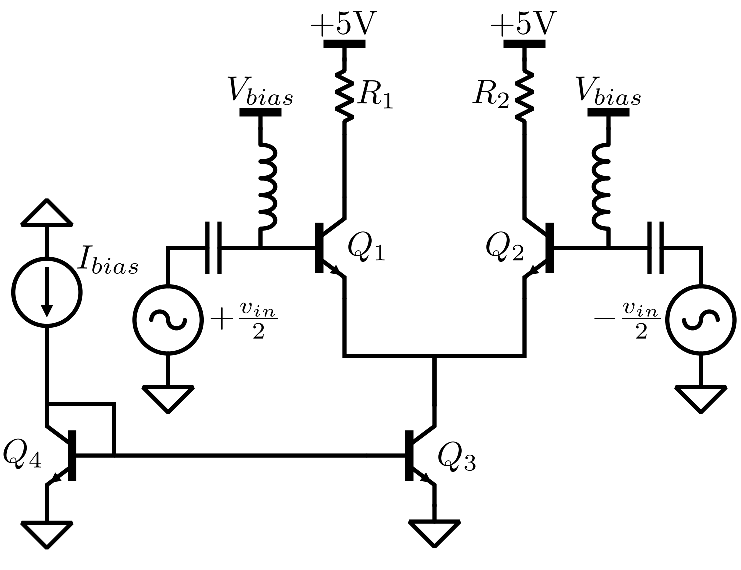
5 SIMULATION RESULTS

This chapter presents simulation results of a few topologies using YalRF and some comparisons are made to Keysight's ADS and QucsStudio. The first section presents the analysis of a simple differential amplifier using DC, AC, and HB simulation. The following section presents the design of multiple oscillator circuits with their steady-state response obtained using YalRF's implementation of the Auxiliary Generator Technique. Lastly, a diode demodulator example presents the two tones feature of the HB simulation, employing the Artificial Frequency Mapping method for the Fourier Transform of quasiperiodic signals. Appendix B shows the testbenches written in Python for YalRF.

5.1 CASE STUDY: DIFFERENTIAL AMPLIFIER

A bipolar differential amplifier with resistive loads and an active current-mirror for biasing is presented in Figure 15. The base of the differential pair transistors, Q_1 and Q_2, is biased using a bias-T structure. The circuit parameters used for this circuit are presented in Table 2. The NPN BJTs all have the same model, displayed in Table 3.

Figure 15 – Differential Amplifier Schematic



Source: Author.

5.1.1 DC Operating Point

The first step to be observed when designing an amplifier is evaluating the operating point. This will determine the amplifier low-frequency gain and the maximum signal excursion that can be achieved without reaching nonlinearities such as the supply rail or removing the BJTs from their forward-active region. Table 4 presents the results obtained by YalRF, ADS

Table 2 – Differential Amplifier Simulation Parameters

Parameter	Value	Unit
R_1, R_2	50	Ω
I_{bias}	50	mA
V_{bias}	1.2	V

Source: Author.

Table 3 – BJT model parameters

Parameter	Value	Unit
I_s	81.1	fA
N_f	1	-
N_r	1	-
I_{kf}	0.5	A
I_{kr}	0.225	A
V_{af}	113	V
V_{ar}	24	V
I_{se}	10.6	pA
N_e	2	
I_{sc}	0	A
N_c	2	-
β_f	205	-
β_r	4	-
C_{je}	29.5	pF
C_{jc}	15.2	pF
C_{js}	0	pF

Source: Author.

and QucsStudio across the circuit nodes. The results agree very well, which works as a good validation of the DC portion of the BJT model.

Table 4 – Operating Point Comparison of the Differential Amplifier

Measured Variable	**YalRF**	**ADS**	**QucsStudio**
Collector voltage of Q_1 and Q_2	3.773 V	3.773	3.773
Emitter voltage of Q_1 and Q_2	0.515 V	0.515	0.515
Base voltage of Q_3	0.705 V	0.705	0.705
Collector current on Q_3	49.35 mA	49.35 mA	49.35 mA

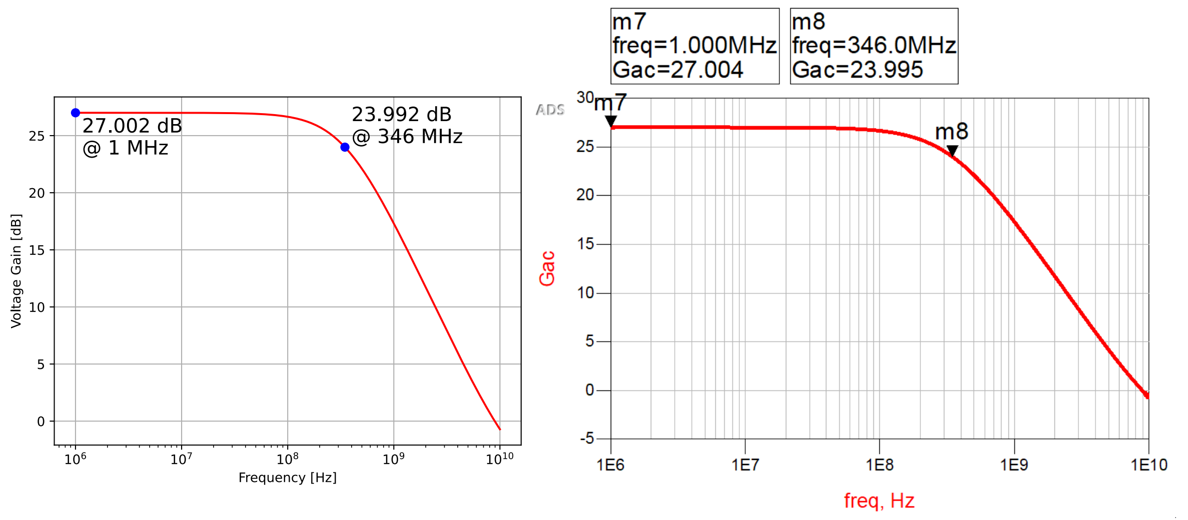
Source: Author.

5.1.2 AC Gain and Gain Compression

With the calculated operating point, the AC gain and bandwidth of the amplifier can be evaluated with an AC simulation. The low-frequency gain of the differential amplifier taking a single-ended output is given by half the product of the load resistance and the bipolar transconductance. This is calculated as,

$$A_{v0} = \frac{g_m R_1}{2} = \frac{I_c}{\phi_t}\frac{R_1}{2} \approx 23.7\,\mathrm{V/V} \text{ or } 27.5\,\mathrm{dB}. \tag{93}$$

Figure 16 – Differential Amplifier AC Gain



a) YalRF b) ADS

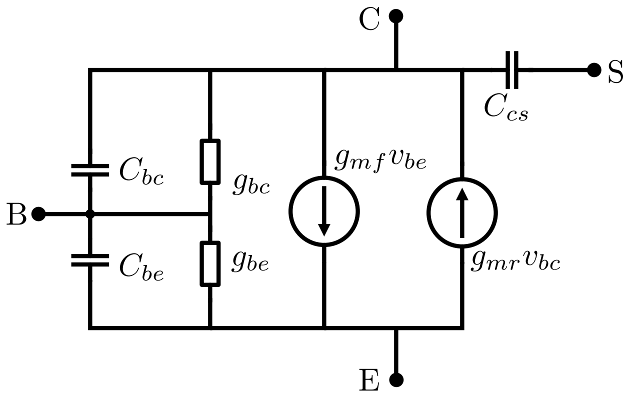
Source: Author.

Figure 16 shows the magnitude of the AC gain and the measured 3dB bandwidth of the amplifier. The results from YalRF and ADS are very coherent. The low-pass behavior and cutoff frequency seen on the AC gain is caused by the linearized parasitic junction capacitances calculated for the Gummel-Poon model. The base-emitter and base-collector capacitances have diffusion and depletion contributions, with the depletion capacitance dominating at reverse bias and the diffusion capacitance dominating at forward bias. The Figure 17 presents the equivalent small-signal model used during AC simulation.

Figure 18 shows the first harmonic gain compression of the amplifier. To obtain this simulation result, an input voltage sweep is performed using harmonic balance simulation and the large-signal gain of the amplifier is calculated taking only the first harmonic magnitude of input and output. As the input signal starts to grow, nonlinearities begin to take place and effectively distort the output signal of the amplifier. If the base-emitter voltage swing becomes large enough, the small-signal assumption used in the gain calculation no longer holds true, as the exponential characteristic of the BJT starts to generate multiple harmonics. There is also the voltage rail saturation, which limits signal excursion and amplification, and the possibility that the transistor leaves the forward active region, in case the base-collector junction becomes forward biased. Those effects generate gain compression at the fundamental, as seen on Figure 18.

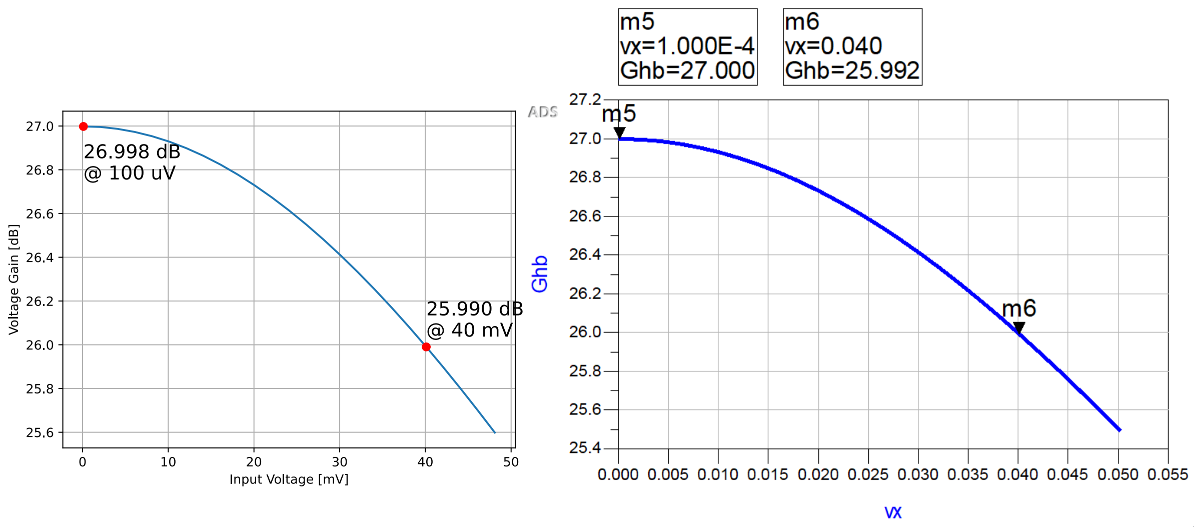
To illustrate the distortion in the output voltage, Figures 19 and 20 present the spectrum and time-domain waveform of a highly distorted amplified signal for YalRF and ADS. For this

Figure 17 – Small-Signal model of the BJT



Source: Author.

Figure 18 – Differential Amplifier Gain Compression



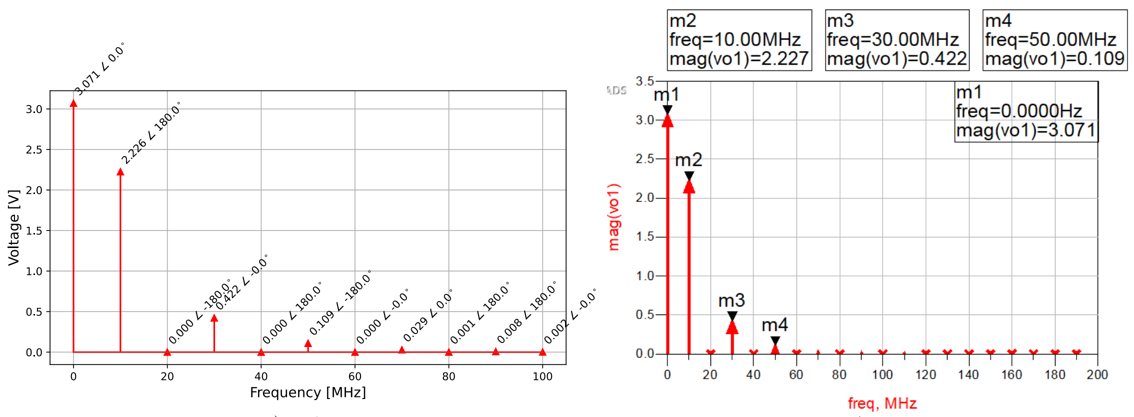
a) YalRF b) ADS

Source: Author.

simulation R_1 and R_2 were changed to 80 Ω and the input v_{in} was set to 60 mV. Again, a comparison between YalRF and ADS waveforms show nearly identical results.

Although no rigorous evaluation of the running time of YalRF was performed so far, to give perspective, the testbench containing the AC and HB sweeps along with the plotting function calls took 6.2 seconds to run on a 2016 laptop with 8GB of RAM and a 7th generation Intel i7 processor. Table 5 shows the execution time of only the harmonic balance voltage sweep, which contains 25 steps, for a different number of harmonics alongside the size required for the Jacobian matrix. For the parameter sweep, the solution of the previous iteration was reused as initial condition for the next one. This can greatly speedup the Newton-Raphson convergence, specially if the sweep values are not too far apart, which means starting the HB simulation with an excellent initial condition.

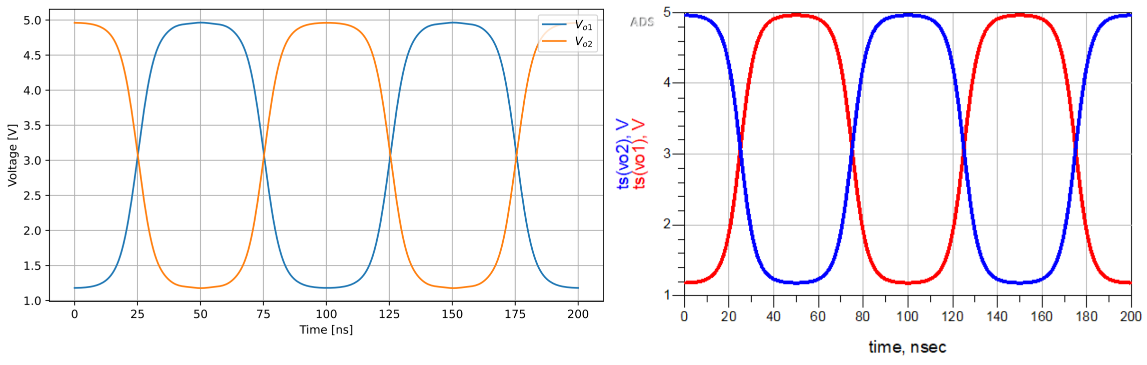
Figure 19 – Differential Amplifier Output Spectrum



a) YalRF b) ADS

Source: Author.

Figure 20 – Differential Amplifier Output Time-Domain Waveform



a) YalRF b) ADS

Source: Author.

Table 5 – Execution Time of Harmonic Balance Voltage Sweep

Number of Harmonics	Jacobian Size	Execution Time [seconds]
10	210×210	1.85
20	410×410	3.96
40	810×810	9.63
80	1610×1610	31.42

Source: Author.

5.2 CASE STUDY: OSCILLATOR ANALYSIS USING HARMONIC BALANCE

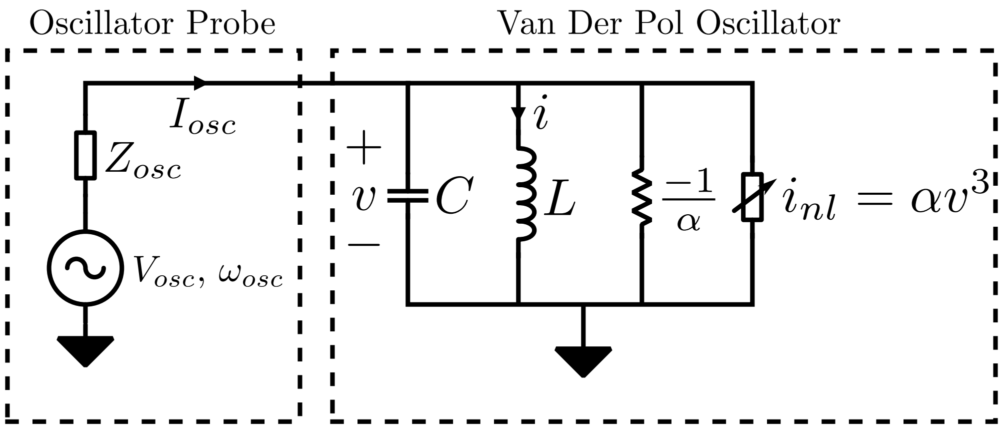
The main interest of this dissertation was the development of a Harmonic Balance implementation with support to oscillator analysis. This section presents some examples of the oscillator analysis feature along with some algebraic modelling of the proposed circuits.

5.2.1 Van Der Pol Oscillator

A classic example in nonlinear stability analysis is the Van Der Pol oscillator, which arose in connection to electrical circuits used in the first radios (STROGATZ, 2015). Figure 21 shows an schematic implementation of a Van Der Pol oscillator, along with the oscillator probe used to achieve the harmonic balance response. This circuit can be described by a second order differential equation, that can be written in its two-dimensional form as:

$$\begin{cases} C\frac{\mathrm{d}v}{\mathrm{d}t} + i + \alpha(v^3 - v) = 0 \\ L\frac{\mathrm{d}i}{\mathrm{d}t} - v = 0 \end{cases} \tag{94}$$

Figure 21 – Van Der Pol Oscillator Schematic



Source: Author.

In Figure 21, a behavioral element is used to represent the cubic nonlinearity of the Van Der Pol equation. The term behavioral is used since this element do not necessarily correspond to a physical electronic device. The behavioral device must be included to the MNA equations in order to be used in simulation. To do that, its I/V and $\frac{\partial I}{\partial V}$ characteristic curves must be known. For a cubic nonlinearity this can be modelled by equations:

$$I(V) = \alpha V^3 \tag{95}$$

$$\frac{\partial I(V)}{\partial V} = 3\alpha V^2 \tag{96}$$

The Van Der Pol oscillator can be analyzed under two different conditions. For large values of α ($\alpha > 1$), the circuit starts to behave in a strongly nonlinear fashion, with slow buildups, followed by fast discharge periods. This is called a relaxation oscillation, and is typical of some oscillator circuit topologies based on comparators. On the other hand, for small values of α, Equation 94 has a very weakly nonlinear behavior, with an autonomous almost purely sinusoidal response. Let α be very small ($\alpha << 1$), so that the the Van Der Pol oscillator response can be described by a single harmonic, then its voltage $v(t)$ is written as,

$$v(t) = a_1 \cos \omega t + a_2 \sin \omega t \tag{97}$$

Applying this solution to Equation 94, the expansion writes:

$$\begin{aligned} &\left[-\omega C a_1 + \frac{a_1}{\omega L} - \alpha a_2\right] \sin \omega t + \left[\omega C a_2 - \frac{a_2}{\omega L} - \alpha a_1\right] \cos \omega t + \\ &\alpha \left[(a_1 \cos \omega t + a_2 \sin \omega t)^3 - (a_1 \cos \omega t + a_2 \sin \omega t)\right] = 0 \end{aligned} \tag{98}$$

Throwing away the third harmonic terms coming from the cubic nonlinearity this expression can be simplified to:

$$\begin{aligned} &\left[-\omega C a_1 + \frac{a_1}{\omega L} - \alpha a_2 + \frac{3}{4}\alpha a_2^3 + \frac{3}{4}\alpha a_1^2 a_2\right] \sin \omega t + \\ &\left[-\omega C a_2 - \frac{a_2}{\omega L} - \alpha a_1 + \frac{3}{4}\alpha a_1^3 + \frac{3}{4}\alpha a_1 a_2^2\right] \cos \omega t = 0 \end{aligned} \tag{99}$$

Further simplifying, the values of the inductor and capacitor are made $L = C = 1$. Now to find the values of a_1 and a_2, the following system is written:

$$\begin{cases} a_1 \left(\frac{1}{\omega} - \omega\right) + \alpha a_2 \left[\frac{3}{4}\left(a_1^2 + a_2^2\right) - 1\right] = 0 \\ a_2 \left(\frac{1}{\omega} - \omega\right) + \alpha a_1 \left[\frac{3}{4}\left(a_2^2 + a_1^2\right) - 1\right] = 0 \end{cases} \tag{100}$$

A solution for this system is the classical degenerate DC, with $a_1 = a_2 = 0$. A nontrivial solution can be found by making $\omega = 1$ and,

$$A^2 = a_1^2 + a_2^2 = \frac{4}{3} \tag{101}$$

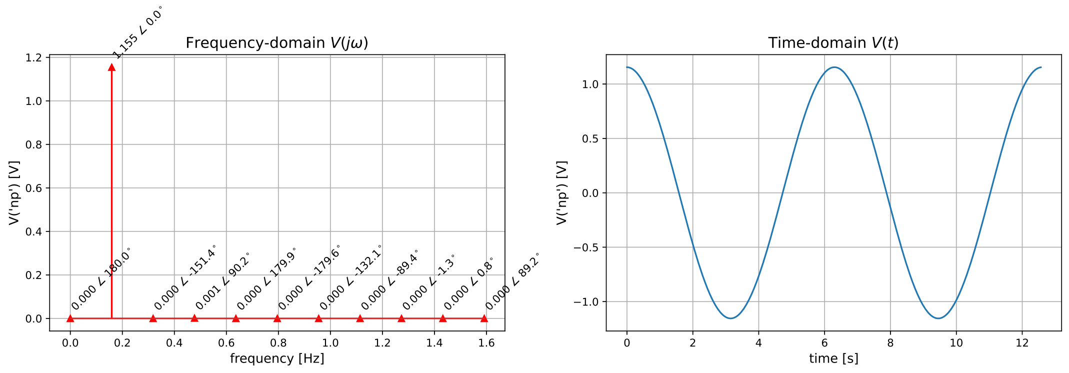
where A is the magnitude of the voltage $v(t)$. Therefore, finally, the oscillating voltage magnitude of the described Van Der Pol oscillator is calculated to be,

$$A = \frac{2}{\sqrt{3}} \approx 1.154701V \tag{102}$$

for small values of α, where the equations behave in a weakly nonlinear regime.

With the expected result at hand, the circuit from Figure 21 was simulated using YalRF to find the autonomous response of the Van Der Pol equations using the harmonic balance method. For $\alpha = 0.01$ and a response comprising 10 harmonics, the result obtained for YalRF is shown in Figure 22. The oscillation is indeed almost purely sinusoidal, validating the assumption used for the algebraic solution. The simulated frequency of oscillation obtained

Figure 22 – Van Der Pol Oscillator voltage response for $\alpha = 0.01$

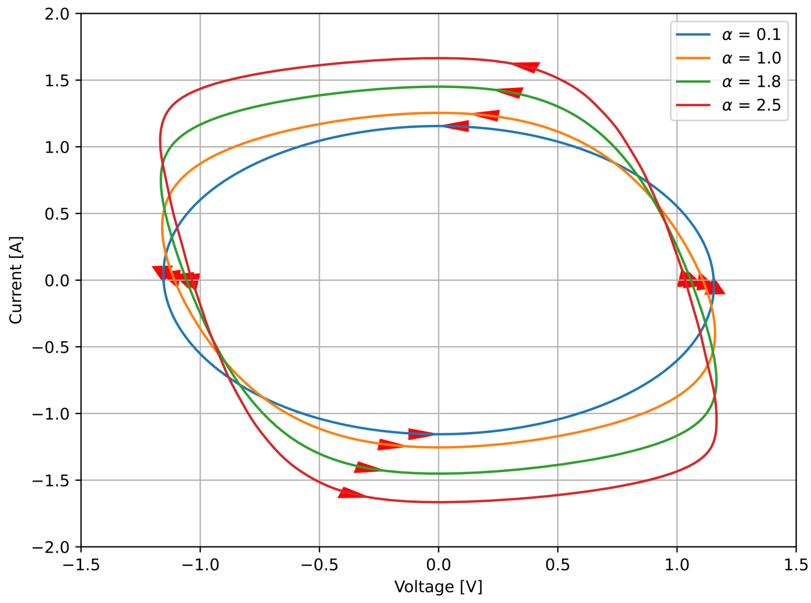


Source: Author.

was $\omega = 0.99999375$ rad/s, and the magnitude of the voltage was $V(\omega) = 1.154703$ V, both agreeing excellently with the calculated values.

An important feature inherent to YalRF is that the resulting data from the simulation is readily available to be post-processed and plotted within the Python environment, without the need to export/convert data or learn UI-specific commands. To display the power of this functionality, with about 10 extra lines of Python code added to the script that generated Figure 22, the limit cycle of the Van der Pol oscillator for multiple simulated values of α can be plotted as shown in Figure 23.

Figure 23 – Limit Cycles of the Van Der Pol Oscillator for multiple values of α



Source: Author.

The limit cycle is a closed trajectory plotted in a two-dimensional phase space, used to analyze time-varying and steady-state characteristics of oscillating systems. For this Van Der Pol oscillator, $v(t)$ and $i(t)$ are the state variables of the system composing the phase space, i.e., they are the x and y axis of the plot, respectively. In Figure 23, four limit cycles are plotted for different values of α, showing how the circuit states evolve over time in a cyclical fashion, characteristic of an oscillator. It can be seen that as alpha increases, the trajectory shape of the current versus voltage deviates from the original ellipse, indicating a stronger nonlinear characteristic.

Although not a complex curve to be displayed, to generate Figure 23 using a Spice-like engine without support to oscillators steady-state response and a simple data analysis interface is a lot more troublesome. The user must setup a parameter sweep over α, a long enough transient simulation to reach steady-state (with an appropriate initial condition to ensure oscillation), clip the resulting waveforms to a point where the oscillator has stabilized, and create an XY plot of voltage versus current. Not to mention that for this particular case, we have the cubic nonlinearity as a device on YalRF, which can be tricky to add in other simulators using behavioral modelling functionalities, such as Verilog-A. Describing this circuit in YalRF takes 5 lines of code, while looping through the harmonic balance simulations, incrementing α and plotting the results took another 15 lines of simple Python scripting.

5.2.2 Common-Base Colpitts Oscillator

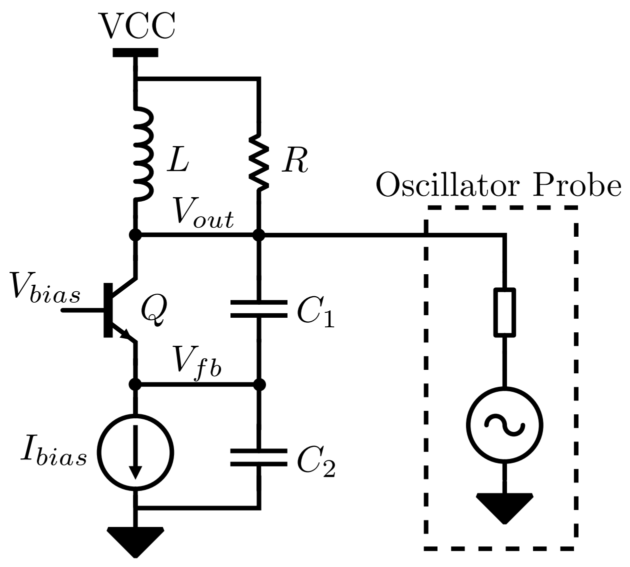
The first practical example will be the classic common-base (CB) Colpitts oscillator, presented in Figure 24, with a harmonic balance oscillator probe attached to the output. The BJT transistor Q is the active element responsible for amplifying the feedback voltage, V_{fb}, which is scaled by the capacitive divider formed by C_1 and C_2. The inductor L serves as a DC path to bias the transistor, but most importantly, it forms a resonating tank circuit with C_1 and C_2. Resistor R encompasses all the tank losses and is directly related to its quality factor. Finally, I_{bias} is the DC tail current responsible for biasing Q and determining the small-signal transconductance g_m.

5.2.2.1 Small-Signal Analysis

An important step while designing an oscillator is determining its ability to oscillate. That is traditionally achieved by viewing an oscillator as a closed-loop feedback system and checking if the Barkhausen criteria is satisfied for a circuit topology.

RF oscillators have usually at its core an LC resonator, and can also be analyzed using the negative resistance approach (ROGERS; PLETT, 2003). A practical LC resonator if excited by a step will oscillate, but its oscillations will eventually fade with time due to the inherent losses of any real physical system. To overcome the losses due to the finite quality factor of the resonator, an active element must be used to supply the lost energy, and keep the oscillations going. This can be modelled as shown in Figure 25, where the losses of the

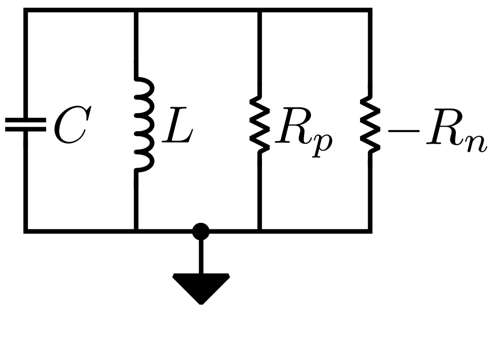
Figure 24 – Common-Base Colpitts Oscillator Schematic



Source: Author.

resonator are encompassed in R_p and the active element compensates R_p by synthesizing a negative resistance $-R_n$, which effectively eliminates the system losses.

Figure 25 – Negative resistance compensates for the resonator inherent losses.



Source: Author.

The negative resistance of the Colpitts oscillator from Figure 24 is generated by transistor Q and the feedback capacitive divider. The circuit is redrawn in Figure 26 replacing the BJT for its small-signal model. Considering $r_e \approx 1/g_m$, one can write for the circuit currents,

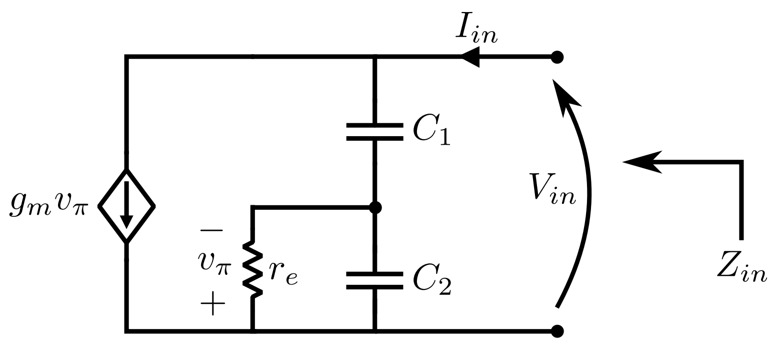
$$I_{in} = g_m v_\pi + j\omega C_1 \left(V_{in} + v_\pi\right) \tag{103}$$

$$j\omega C_1 \left(V_{in} + v_\pi\right) = -\left(g_m + j\omega C_2\right) v_\pi \tag{104}$$

This system of equations can be solved for the input impedance, Z_{in}:

$$Z_{in} = \frac{V_{in}}{I_{in}} = \frac{-g_m}{\omega^2 C_1 C_2} + \frac{1}{j\omega C_1} + \frac{1}{j\omega C_2} \tag{105}$$

Figure 26 – Negative resistance of the CB Colpitts Oscillator.



Source: Author.

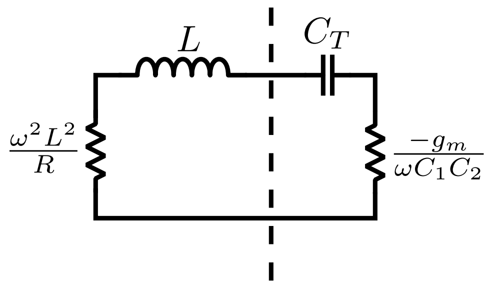
The input impedance obtained on Equation 105 is a negative resistance in series with an equivalent capacitor, C_T, written as:

$$C_T = \frac{C_1 C_2}{C_1 + C_2} \tag{106}$$

Since Z_{in} presents a series equivalent, the RL circuit from the Colpitts oscillator must be transformed into a series representation as well, as shown in Figure 27, in order to check if there is enough negative resistance to compensate for the losses and to calculate the oscillating frequency. According to the parallel to series impedance conversion equations, considering a high quality factor for the RL circuit ($Q_L >> 1$), the inductance L does not change due to parallel-series transformation and the series resistance is

$$R_s = \frac{R}{Q_L^2} = \frac{R}{\left(\frac{R}{\omega L}\right)^2} = \frac{\omega^2 L^2}{R} \tag{107}$$

Figure 27 – CB Colpitts equivalent circuit using negative resistance representation.



Source: Author.

The condition for oscillation is that there is more negative resistance in order to fully eliminate the losses, which transcribes to

$$\left|\frac{-g_m}{\omega^2 C_1 C_2}\right| > \frac{\omega^2 L^2}{R} \tag{108}$$

Defining the oscillating frequency from the equivalent circuit as,

$$\omega = \frac{1}{\sqrt{LC_T}} \tag{109}$$

and the ratio of the Colpitts capacitive divider as,

$$n = \frac{C_1}{C_1 + C_2} \tag{110}$$

Substituting into Equation 108, finally, an expression for the minimum required transconductance of the Colpitts oscillator can be found:

$$g_m > \frac{1}{n\,(1-n)\,R} \tag{111}$$

This equation is fundamental to understand the requirements for the Colpitts oscillator to start-up. It shows a strong the dependency on n, which is the amount of feedback signal provided by the capacitive divider back to the common-base amplifier input. It also shows the effect that the resonator losses, R, have on preventing continuous oscillation to happen.

5.2.2.2 Large-Signal Analysis

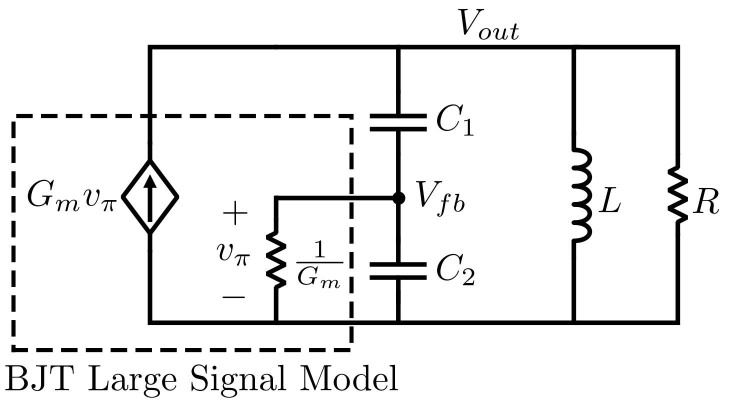
Another parameter of interest to the designer is an estimation of the output amplitude to be expected of an oscillator. This is determined by the nonlinearities present on the system, which limit the amplitude growth. The nonlinear mechanism which limits the amplitude of oscillation for the Colpitts from Figure 24 is the reduction of the transistor transconductance for large-signals.

The large-signal transconductance of a transistor, G_m, can be defined through the use of describing functions, a method employed in nonlinear analysis. As presented in Lee (2004), for large periodic inputs, the transistor can be considered to conduct strongly for a brief period of time, when the input is at its maximum, while being cutoff for most of the rest of the period. Under this assumption, the current through the transistor can be approximated as a series of pulses, whose first harmonic, I_1, converges to the amplitude of $2I_{bias}$. Therefore, the describing function of the transistor, G_m, can be written as,

$$G_m = \frac{I_1}{V_1} \approx \frac{2I_{bias}}{V_1} \tag{112}$$

where I_1, for the BJT case, is the first harmonic of the collector current, while V_1 is the sinusoidal base-emitter voltage applied. Knowing the large-signal I/V relationship of the first harmonic of the BJT is enough to estimate the voltage amplitude of the Colpitts oscillator, since the voltage signals are nearly purely sinusoidal. Figure 28 presents an equivalent circuit for the Colpitts schematic, where the BJT was replaced by its large-signal equivalent (or its describing function representation), and the noted voltages are pure sinusoids.

Figure 28 – CB Colpitts large-signal equivalent circuit.



Source: Author.

Considering that C_1 and C_2 form an ideal impedance transformer, i.e. $1/G_m$ does not significantly load the capacitive divider, the emitter impedance can be transformed to an equivalent resistor, R_{eq}, which appears in parallel to R, and has a magnitude of,

$$R_{eq} \approx \frac{1}{n^2 G_m} \tag{113}$$

where n is the capacitive divider ratio previously defined. Also the dependent current source $G_m v_\pi$ actually produces a constant sinusoid current of magnitude $2I_{bias}$. From those observations, at resonance, the total output voltage can be written as,

$$V_{out} = 2I_{bias}\left(R \parallel R_{eq}\right) = 2I_{bias}\left(R \parallel \frac{1}{n^2 G_m}\right) \tag{114}$$

From the capacitive divider,

$$v_\pi = nV_{out} \tag{115}$$

and substituting into the large-signal transconductance:

$$G_m = \frac{2I_{bias}}{nV_{out}} \tag{116}$$

Now replacing G_m into Equation 114, finally the estimated output voltage for the Colpitts oscillator can be written:

$$V_{out} \approx 2I_{bias}R(1-n) \tag{117}$$

This result is limited to cases where the feedback signal is large, meaning the transistor operates more closely to a switch, with a pulsed current behavior. Another assumption used is that the $1/G_m$ emitter impedance does not load significantly the capacitive divider. Nonetheless, those assumptions can be easily met and this is a powerful result to understand how oscillating amplitude exchanges with other design factors, such as bias current and the capacitive divider ratio.

5.2.2.3 Simulation Results

With the algebraic results at hand, the original schematic for the Common-Base Colpitts oscillator was described in YalRF for simulation with the parameters from Table 6. The desired oscillating frequency is f_{osc}, while I_s and β describe the BJT transistor model used.

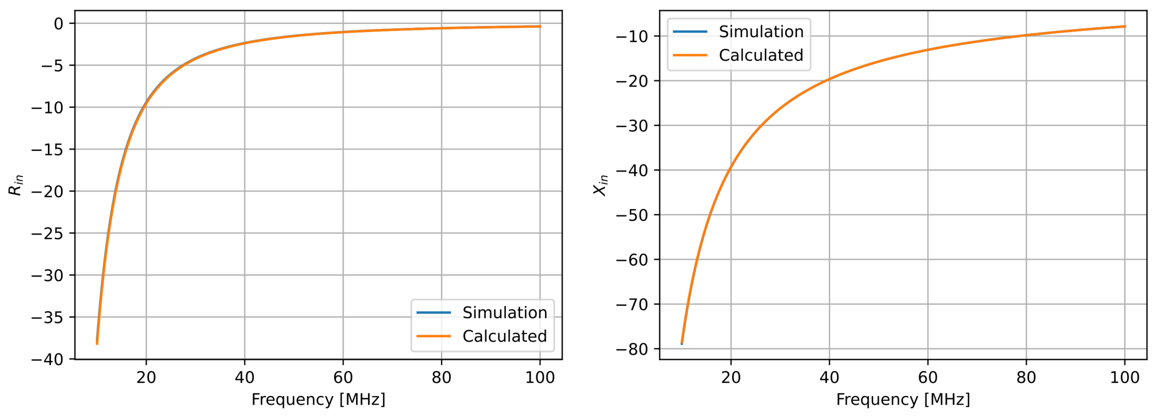
Table 6 – Nominal values of Common-Base Colpitts Parameters

Parameter	Value	Unit
f_{osc}	50	MHz
I_{bias}	1	mA
V_{cc}	5	V
R	850	Ω
L	50	nH
C_T	202.6	pF
n	0.3	-
I_s of Q	1	fA
β of Q	100	A/A

Source: Author.

The first analysis conducted is an AC simulation sweep from 10 to 100 MHz to verify the amount of negative resistance obtained from C_1, C_2 and Q. Converting R to a series resistance using Equation 117 gives around 290 mΩ, so that is the absolute minimum of negative resistance required for oscillation to start, although a larger value is desirable. Figure 29 presents the perfectly overlapping curves obtained for the input impedance of the Colpitts negative resistance generator, using simulation with YalRF, and algebraically employing Equation 105. At 50 MHz, the negative resistance is about $-2\,\Omega$, enough to compensate for the losses.

Figure 29 – Comparison of CB Colpitts negative resistance and series reactance obtained using YalRF AC analysis and the algebraic small-signal model.

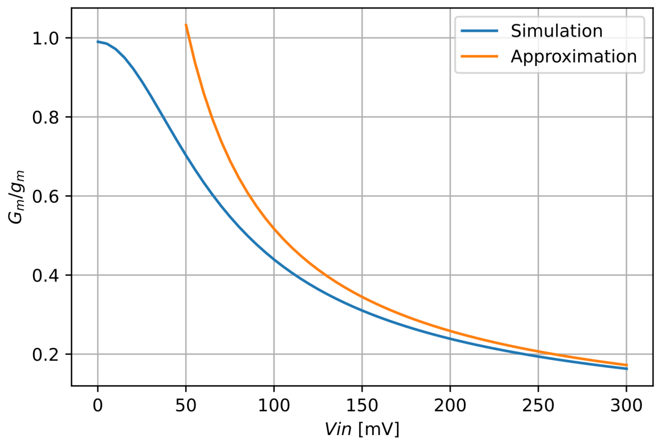


Source: Author.

Another analysis of interest is to verify how appropriate the approximation used for the large-signal transconductance (Equation 112) is for the BJT. To extract G_m, a Harmonic

Balance analysis must be conducted, since the reduction of the first-harmonic transconductance with respect to the level of input signal is a nonlinear characteristic of the transistor. Sweeping the amplitude of the input signal at the base of Q while polarized by I_{bias}, the curve from Figure 30 can be easily extracted using YalRF. The curve was normalized by the small-signal transconductance, g_m, and the approximation using Equation 112 is included to the plot to verify its accuracy. It is clear that the approximation starts to work very well for input values above around 150 mV.

Figure 30 – Normalized BJT large-signal transconductance waveform obtained with YalRF and its approximation for a pulsed current profile.



Source: Author.

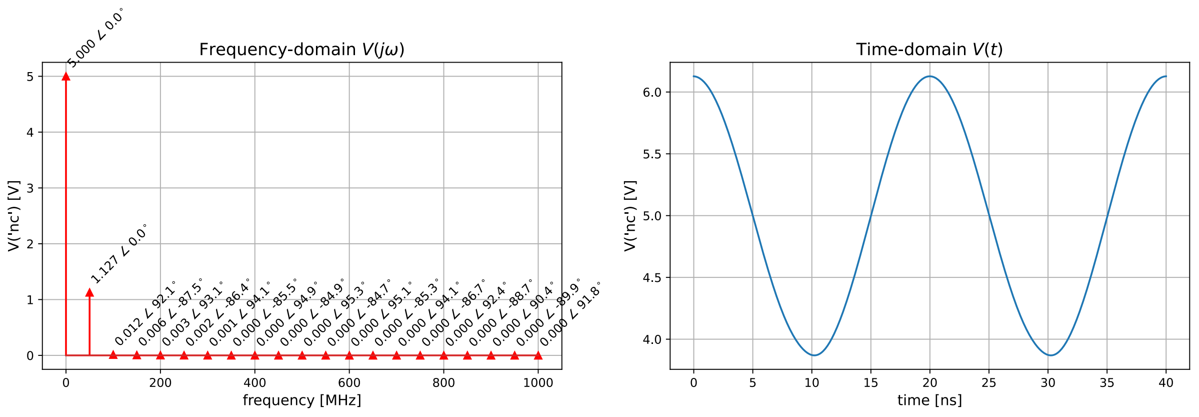
Again, it is worth noting the simplicity and usefulness of conducting, simultaneously, algebraic and simulated analysis of circuits using YalRF inside a Python environment. Both Figures 29 and 30 are created using a single script, where design equations and circuit simulation share design parameters and other variable definitions. It becomes very simple to exchange information between equations and simulation. This feature can be very desirable while studying and modeling new circuit topologies, a regular task for electronics students and researchers.

Finally, an HB Oscillator Analysis is performed for the complete Colpitts oscillator of Figure 24, using the probe inserted at the output node to find the periodic oscillating regime. In total 20 harmonics were used for simulation and the steady-state response of the designed common-base Colpitts oscillator is shown in Figure 31. The output voltage is calculated as,

$$V_{out} \approx 2 \times 1\,\text{mA} \times 850\,\Omega \times (1 - 0.3) = 1.19\,\text{V} \tag{118}$$

which is confirmed as a reasonable estimate if compared to the actual output obtained in simulation of 1.127 V for the fundamental frequency. Due to the high quality factor of the tank circuit, very little distortion is seen at the output spectrum. The frequency of oscillation found was 50 005 832.6 Hz, very close to the 50 MHz originally defined.

Figure 31 – Output voltage waveform of the CB Colpitts oscillator obtained with YalRF.



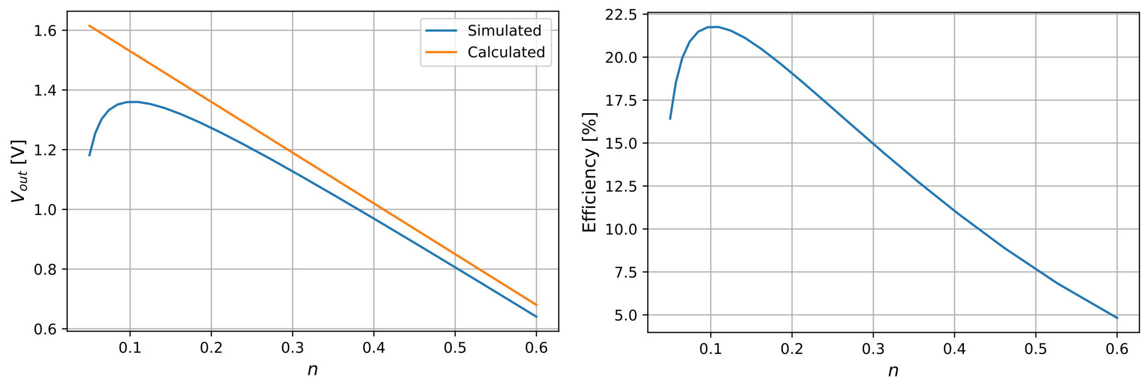
Source: Author.

During design, sweeps can be useful to understand how a circuit behaves according to some design parameter. An interesting analysis for the Colpitts oscillator, is to see how the output voltage and the efficiency of the oscillator vary with the capacitive divider ratio, n. Defining the efficiency η_{eff} as,

$$\eta_{eff} = \frac{P_{ac}}{P_{dc}} \times 100\% = 100 \times \frac{V_{out}^2}{2R\left(V_{cc}I_{bias}\right)} \tag{119}$$

an HB Oscillator Analysis sweep can be performed over n. The results are plotted in Figure 32, for 20 steps ranging from $n = 0.05$ to $n = 0.6$. The calculated estimative of the output voltage for multiple values of n was included to the plot, so that its accuracy could be verified against simulation.

Figure 32 – Output voltage and efficiency of the CB Colpitts oscillator as a function of the capacitive divider ratio.



Source: Author.

5.2.2.4 Automatic Design Optimization

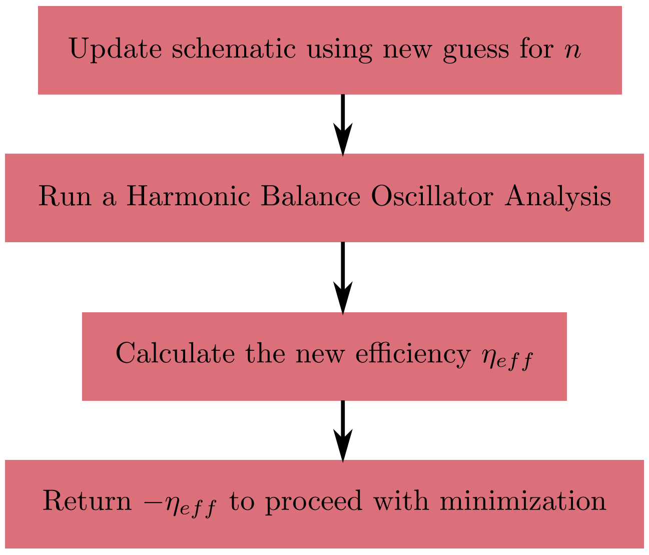
As seen in Figure 32, the efficiency of the Colpitts oscillator has an optimal value for n around 0.1. Finding the optimal point for a particular design is not often that straightforward. Using the broad class of optimizers present at the Scipy package, it is simple to define design optimization problems in a testbench with YalRF. The main concern of the designer is to properly define the objective function, which carries the goal of the optimization.

As an example, to obtain the optimal n for largest efficiency, the following minimization problem can be defined:

$$\min_{n} \left(-\eta_{eff}\right) \tag{120}$$

where $-\eta_{eff}$ is the objective function to be minimized. Minimizing the negative of the efficiency is equivalent to maximizing the efficiency itself. The objective function will take as input the current candidate solution for n and then it should follow the fluxogram described in Figure 33.

Figure 33 – Objective function fluxogram to optimize the Colpitts oscillator efficiency.


Update schematic using new guess for n
Run a Harmonic Balance Oscillator Analysis
Calculate the new efficiency η_{eff}
Return $-\eta_{eff}$ to proceed with minimization

Source: Author.

With the objective function defined, the optimizer can be called within a single line to start the optimization procedure. For this example, the **minimize_scalar** with the **bounded** method is employed, since this is a one-dimensional problem of minimizing a scalar function and boundaries for the value of n must be used. For more complex design problems, there are multidimensional constrained algorithms available and also global optimization methods, such as differential evolution (VIRTANEN et al., 2020).

The results obtained for the optimization of the Colpitts oscillator efficiency are presented in Table 7. The optimal values of n and η_{eff} perfectly agree with the peak observed in Figure 32.

Table 7 – Results of the efficiency optimization for the Colpitts oscillator.

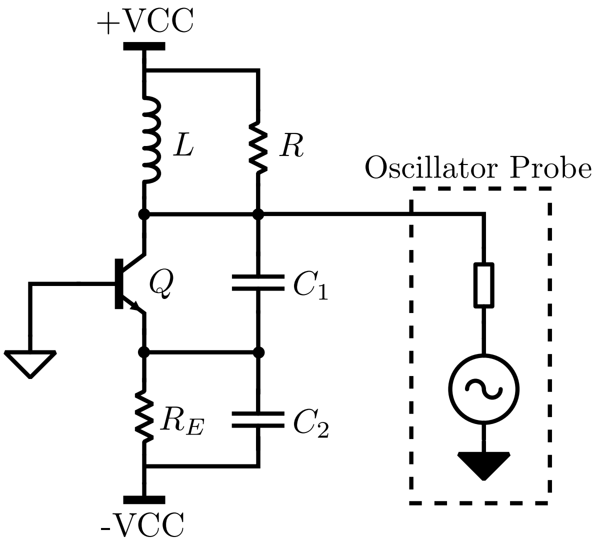
Parameter	Value	Unit
Optimal Efficiency (η_{eff})	21.8	%
Optimal Capacitive Divider Ratio (n)	0.104	-
Total Execution Time	62.4	seconds
Number of function evaluations	11	-

Source: Author.

5.2.2.5 Oscillator Results using a BJT Model with Nonlinear Parasitics

The schematic of Figure 34 presents yet another common-base Colpitts oscillator, with the ideal current source replaced by a resistor, R_E, whose purpose is also to set the bias current. Table 8 presents the parameters used in simulation.

Figure 34 – CB Colpitts Oscillator Schematic



Source: Author.

The main difference for this simulation was the model used for the BJT, which is shown in Figure 35. It emulates a BC847 transistor and contains parameters for many features of the Gummel-Poon model, such as forward and reverse current gain, Early voltage nonlinearity, parasitic junction capacitances and charge transit times. The ability to use this model with HB means that both current and charge equations are properly implemented into the BJT source code. The picture presents the model included in ADS but the exact same parameters were used with simulation in YalRF and also QucsStudio. The presence of modelled parasitic capacitances is very important in the design of practical oscillators, due to the effect they have in the actual oscillating frequency.

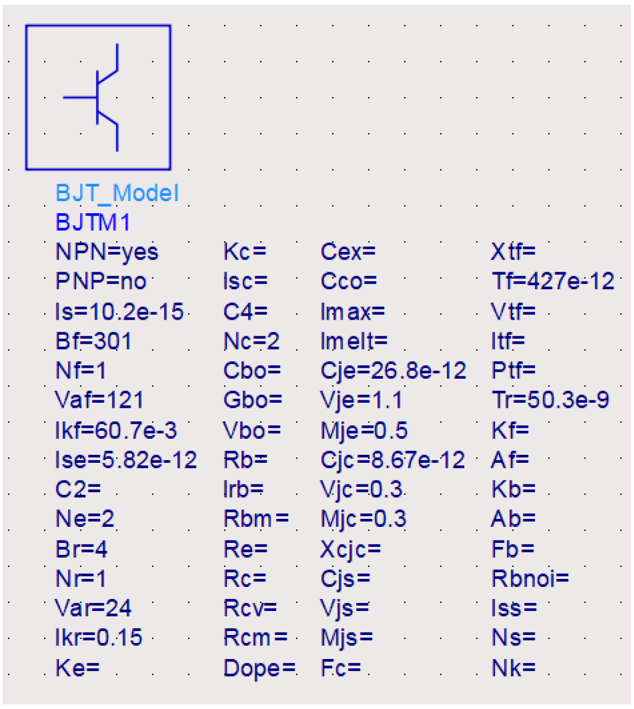
Figures 36 and 37 present the output waveform of the common-base Colpitts, taken at the collector of the bipolar. In ADS, the oscillatory response was found using HB and the aid

Table 8 – CB Colpitts Simulation Parameters

Parameter	Value	Unit
Harmonics	10	-
f_{guess}	8	MHz
V_{guess}	1	V
VCC	3	V
R_E	2.2	kΩ
R	8.73	kΩ
L	5	μH
C_1	200	pF
C_2	50	pF

Source: Author.

Figure 35 – BC847 adapted model used in ADS and YalRF for the oscillator



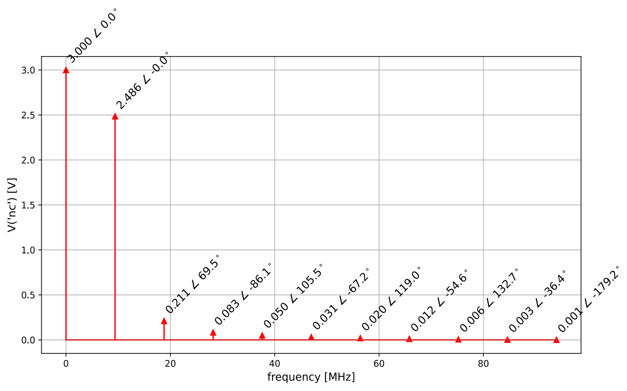
```
BJT_Model
BJTM1
NPN=yes         Kc=      Cex=            Xtf=
PNP=no          Isc=     Cco=            Tf=427e-12
Is=10.2e-15     C4=      Imax=           Vtf=
Bf=301          Nc=2     Imelt=          Itf=
Nf=1            Cbo=     Cje=26.8e-12    Ptf=
Vaf=121         Gbo=     Vje=1.1         Tr=50.3e-9
Ikf=60.7e-3     Vbo=     Mje=0.5         Kf=
Ise=5.82e-12    Rb=      Cjc=8.67e-12    Af=
C2=             Irb=     Vjc=0.3         Kb=
Ne=2            Rbm=     Mjc=0.3         Ab=
Br=4            Re=      Xcjc=           Fb=
Nr=1            Rc=      Cjs=            Rbnoi=
Var=24          Rcv=     Vjs=            Iss=
Ikr=0.15        Rcm=     Mjs=            Ns=
Ke=             Dope=    Fc=             Nk=
```

Source: Author.

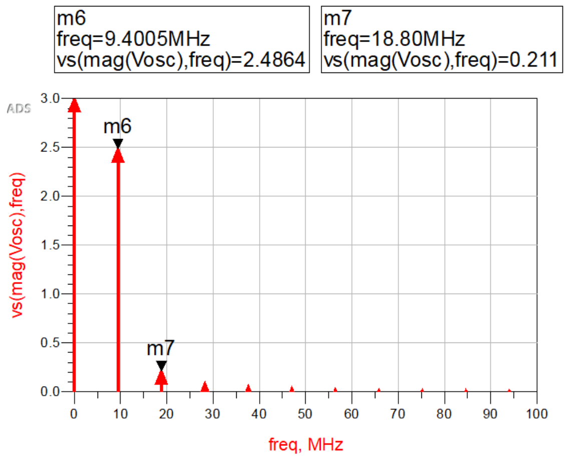
of an artificial device called **OscPort**. YalRF also succesfully found the oscillatory response using HB and the Artificial Generator Technique.

As can be seen by visual inspection, the outputs of ADS and YalRF are very similar in shape and absolute magnitudes, which works as good validation for YalRF algorithms and models. The initial voltage and frequency guesses, used to start the optimization procedure in search of the oscillatory response, were presented in Table 8. They are considerably distant from the obtained oscillating frequency and magnitude shown in Table 9, which demonstrates a decent robustness to poor initial guesses provided by the user. For this testbench, YalRF

Figure 36 – CB Colpitts with BC847 Output spectrum



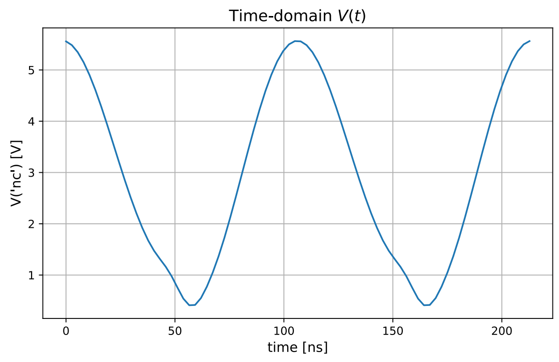
a) YalRF



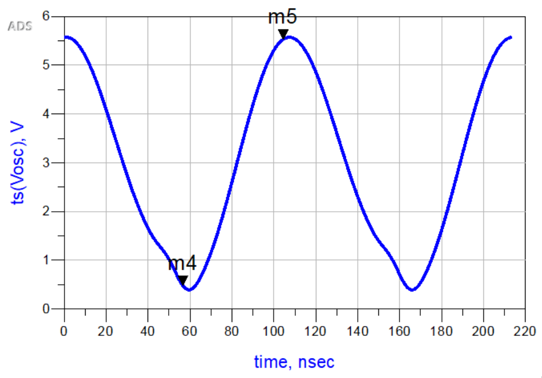
b) ADS

Source: Author.

Figure 37 – CB Colpitts with BC847 Output waveform



a) YalRF



b) ADS

Source: Author.

converged correctly with frequency guesses ranging from 5.8 up to 20 MHz.

The large magnitude of the oscillation is another positive aspect of this simulation results. The larger the voltages appearing over the bipolar transistor, the easier it is for the engine to suffer from convergence issues due to exponential overflow or lack of numerical precision. HB voltages can vary vastly during the first iterations of Newton-Raphson. Limiting and continuation techniques help alleviate this issue and increase robustness.

Finally, Table 9 shows a comparison of the HB oscillator analysis of ADS and YalRF, along with the Shooting method used by QucsStudio. Although the results overall agree very well, the phase mismatches observed must still be studied in greater depth.

Table 9 – CB Colpitts Results comparison: ADS x YalRF

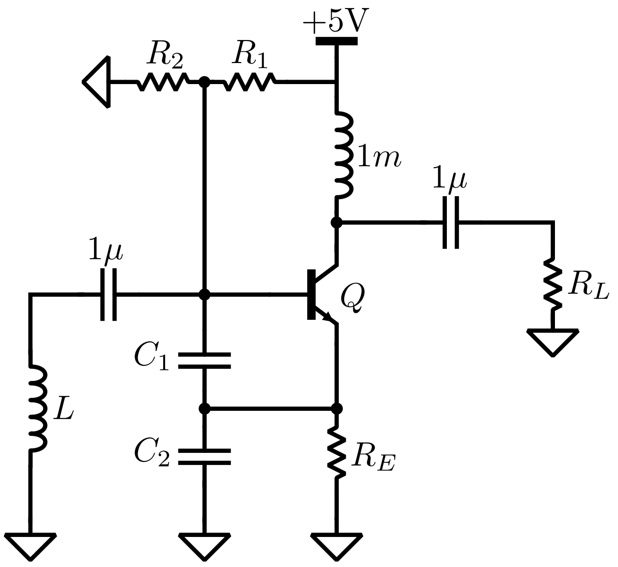
	f_{osc}[MHz]	V_1	V_2	V_3
ADS	9.400721	$2.486\angle-8.5^\circ$	$0.211\angle52.5^\circ$	$0.083\angle-111.6^\circ$
YalRF	9.400713	$2.486\angle0.0^\circ$	$0.211\angle69.5^\circ$	$0.083\angle-86.1^\circ$
QucsStudio	9.400724	$2.486\angle-14.9^\circ$	$0.211\angle39.7^\circ$	$0.083\angle-131.0^\circ$

Source: Author.

5.2.3 Common-Collector Colpitts Oscillator

The next validation example is a common-collector (CC) Colpitts (shown in Figure 38). Its intrinsic idea is very similar to the common-base, a tank circuit and an active element. The difference here is that the BJT now works as a buffer with gain below unity. Therefore, to have a loop voltage gain greater than 1 and satisfy the Barkhausen criteria for oscillation, the reactive network formed by C_1, C_2 and L (loaded by $R_1 \| R_2$), must provide voltage gain greater than the inverse of the buffer gain.

Figure 38 – CC Colpitts Oscillator Schematic



Source: Author.

Table 10 presents the parameters used in simulation. The BJT model used was simple and contained no parasitic capacitances: $I_s = 1\,\text{fA}$, $V_{af} = 60\,\text{V}$, $V_{ar} = 20\,\text{V}$, $\beta_F = 100$ and $\beta_R = 5$. The high number of harmonics (50) was chosen to present the highly distorted signal that appears at the load R_L. Strongly nonlinear signals, such as the load voltage, make HB convergence harder and are good tests for the engine stability.

Figure 39 present the voltage waveform at the inductor node. Due to the tank circuit filtering, the voltage on that node is very clean, in the sense that the fundamental is much stronger than the other harmonics, resulting in an almost purely sinusoidal signal. Again, both

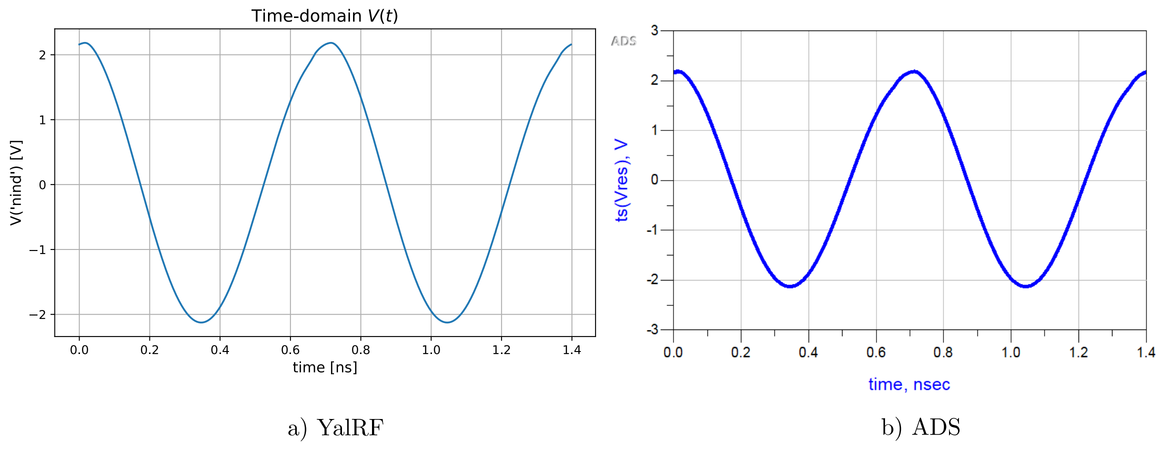
Table 10 – CC Colpitts Simulation Parameters

Parameter	Value	Unit
Harmonics	50	-
f_{guess}	1.2	GHz
V_{guess}	1	V
R_E	1.5	kΩ
R_1	3	kΩ
R_2	6.8	kΩ
R_L	50	Ω
L	6.94	nH
C_1	3.3	pF
C_2	3.9	pF

Source: Author.

ADS and YalRF were able to solve the circuit for its autonomous response and their waveforms look nearly identical. YalRF was able to converge to the oscillating condition, with frequency guesses ranging from at least 600 MHz up to 2.5 GHz. Table 11 shows a comparison of the first three harmonics.

Figure 39 – CC Colpitts Inductor waveform



Source: Author.

Table 11 – CC Colpitts Comparison of the resulting spectrum

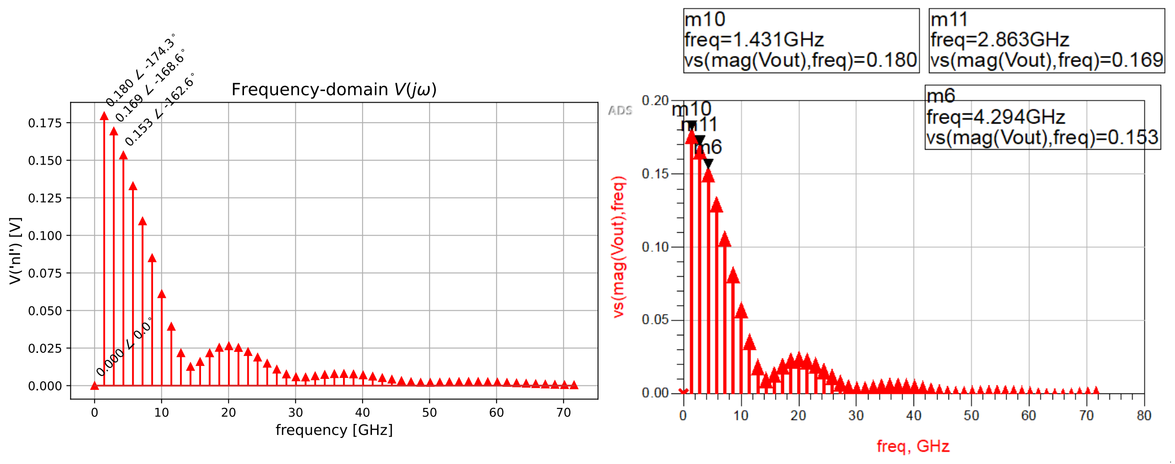
	f_{osc}[GHz]	V_1	V_2	V_3
ADS	1.4313852	$2.134\angle 1.0°$	$0.052\angle -75.5°$	$0.026\angle -69.9°$
YalRF	1.4313555	$2.132\angle 0.0°$	$0.051\angle -77.7°$	$0.026\angle -73.1°$

Source: Author.

The output taken at the 50 ohms load, shown in Figures 40 and 41, was purposely kept very distorted, to show that the algorithm converges for high nonlinearity distortions.

Fast transitions, such as the one seen, have very high harmonic content. The fact that the time-domain signals are quite similar confirms that the phase of each harmonic component is correct. Subtle differences on the time-domain waveform can be attributed to two distinct factors: first, the oversampling value used to reconstruct the time-domain waveforms is likely different. Second, ADS uses the FFT algorithm to compute the time-frequency conversions instead of the DFT matrix, used in YalRF. Although faster, to compute the Fourier Transform using the FFT, ADS has to perform HB for a number of harmonics that is a power of 2. Therefore, the actual number of harmonics being used during the HB solving steps is different and some aliasing differences are to be expected.

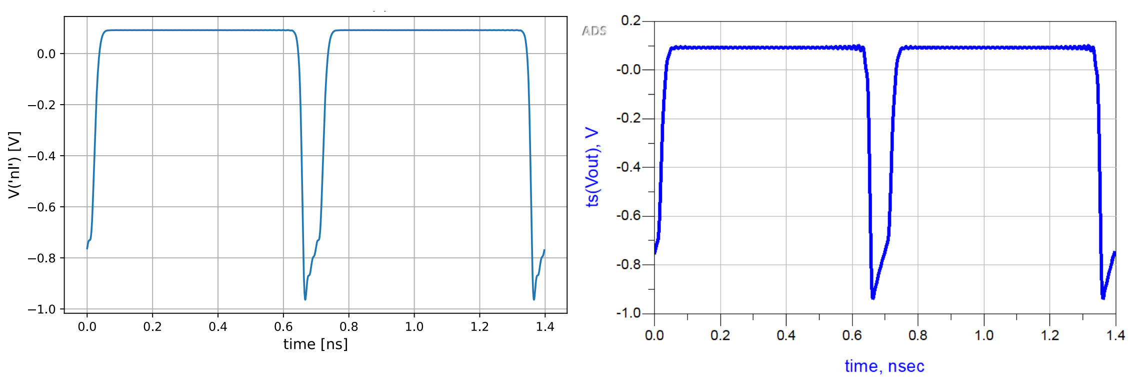
Figure 40 – CC Colpitts Load spectrum



a) YalRF b) ADS

Source: Author.

Figure 41 – CC Colpitts Load waveform



a) YalRF b) ADS

Source: Author.

5.2.4 Peltz Oscillator

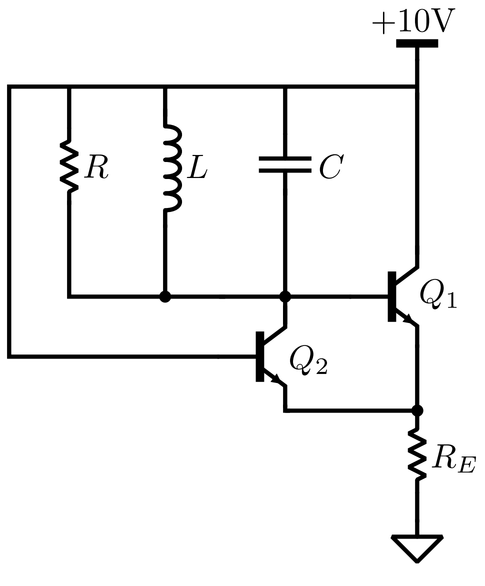
As a final example of YalRF autonomous HB capability, Figure 42 presents the schematic of a Peltz oscillator. The Peltz oscillator employs two active devices, bipolars in this example, with Q_1 connected as an emitter-follower with slightly below unity gain, and Q_2 is connected as a common-base amplifier with gain peaking at the parallel resonance of the tank circuit, when the load impedance reaches its maximum value. The oscillating frequency is determined by the LC tank as:

$$f_{osc} = \frac{1}{2\pi\sqrt{LC}} \tag{121}$$

In this configuration, the peak to peak swing that can be reached across the tank is limited by the base-collector junctions of the bipolar transistors. As the base voltage of Q_1 starts to swing above ground, its base-collector junction gets forward biased, effectively limiting the maximum voltage that can be reached on that node. Similarly, as the signal swings below ground on the collector of Q_2, its base-collector junction also gets forward biased, limiting the negative excursion of the oscillating signal.

Table 12 presents the values used for simulation. The transistor parameters used were: $I_s = 0.1\,\text{fA}$, $\beta_F = 200$ and $\beta_R = 1$.

Figure 42 – Peltz Oscillator



Source: Author.

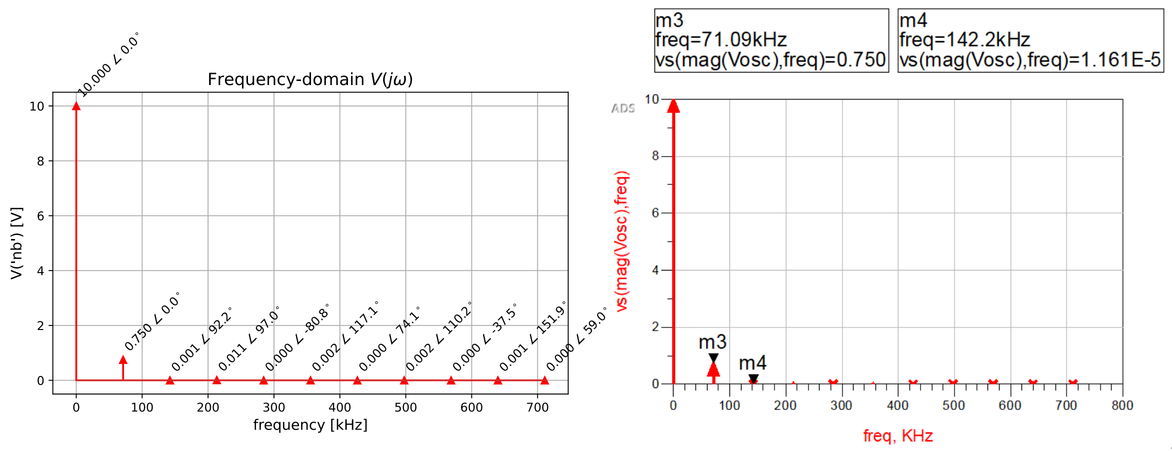
The resulting waveforms from ADS and YalRF, presented on Figures 43 and 44, confirm the affirmations about the oscillator behavior, since we can distinguish a peak to peak voltage of about 1.5 V at the tank circuit, limited on the positive and negative swing by the base-collector diodes of Q_1 and Q_2. Also, again the comparison on Table 13 shows a good agreement between the results from ADS and YalRF.

Table 12 – Peltz Oscillator Simulation Parameters

Parameter	Value	Unit
Harmonics	10	-
f_{guess}	100	kHz
V_{guess}	0.5	V
R_E	50	kΩ
R	200	kΩ
L	0.5	mH
C	10	nF

Source: Author.

Figure 43 – Peltz Oscillator output spectrum



a) YalRF b) ADS

Source: Author.

Table 13 – Peltz Oscillator Results comparison: ADS x YalRF

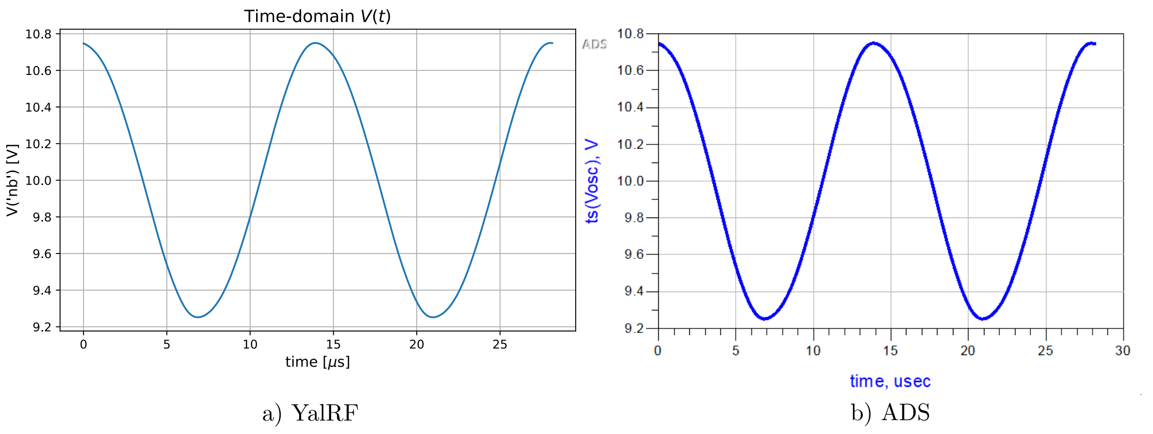
	f_{osc}[kHz]	V_1	V_3	V_5
ADS	71.085310	$0.750\angle 0.0^\circ$	$0.011\angle 96.8^\circ$	$0.002\angle 118.7^\circ$
YalRF	71.092758	$0.750\angle 0.0^\circ$	$0.011\angle 97.0^\circ$	$0.002\angle 117.1^\circ$

Source: Author.

5.3 CASE STUDY: DIODE DEMODULATOR

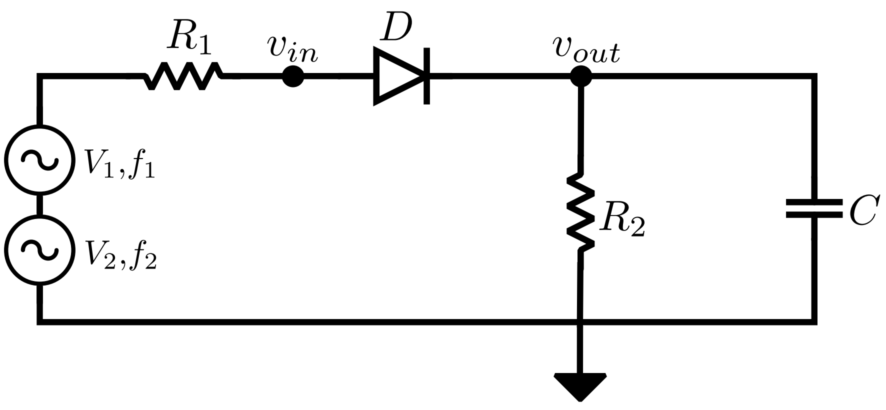
To present the two tones functionality implemented in the Harmonic Balance engine of YalRF a diode demodulator example is shown. YalRF employs the Artificial Frequency Mapping technique in order to perform the Fourier Transform of the almost-periodic signals, resultant of the two tones simulation. Figure 45 presents the schematic used. The diode model uses $I_s = 1$ fA and $N = 1$. The other parameters of the diode model, such as junction capacitance, were not used in this example. Table 14 presents the rest of the simulation parameters, where K_1 and K_2 are the number of harmonics for f_1 and f_2, respectively.

Figure 44 – Peltz Oscillator output waveform



Source: Author.

Figure 45 – Diode Demodulator Schematic



Source: Author.

The harmonic balance problem size grows very quickly with the number of tones, since the amount of frequency bins generated to contemplate the intermodulation products grows with the product of K_1 and K_2. For this example, 431 frequency values must be simulated for $K_1 = 20$ and $K_2 = 10$. This results in a Jacobian matrix of 5166 x 5166 elements and a total simulation time of 195 seconds. The maximum intermodulation product order can be truncated in order to alleviate the memory and execution time burden, but that is not currently implemented in YalRF. Nonetheless, for usages such as evaluating mixers conversion loss or amplifiers IIP3 and IIP2, a small number of harmonics can be used and the algorithm should perform well.

Since the resulting spectrum is too large and sparse to be plotted meaningfully, Table 15 shows a comparison between a few harmonics and intermodulation products obtained from simulations using YalRF, ADS and also QucsStudio. The results, again, indicate a good agreement between the different simulations. Figures 46 and 47 present the time-domain

Table 14 – Diode Demodulator Simulation Parameters

Parameter	Value	Unit
K_1	20	-
K_2	10	-
f_1	1	MHz
f_2	10	kHz
V_1	5	V
V_2	0.5	V
R_1	50	Ω
R	5	kΩ
C	2.2	nF

Source: Author.

waveform generated from the sparse resulting spectrum on YalRF and ADS.

Figure 46 shows the input signal, a strong 10 MHz carrier with a 10 kHz envelope signal of 10 % modulation depth. Figure 47 presents the demodulated output, where the low-frequency 10 kHz envelope appears at the output due to the envelope following characteristic of the demodulator circuit. Due to the simplicity of the filtering, a lot of high-frequency noise is still present at the output, which is indicated by the thickness of the output waveform, representing 1 MHz periods of charge and discharge of the output capacitor.

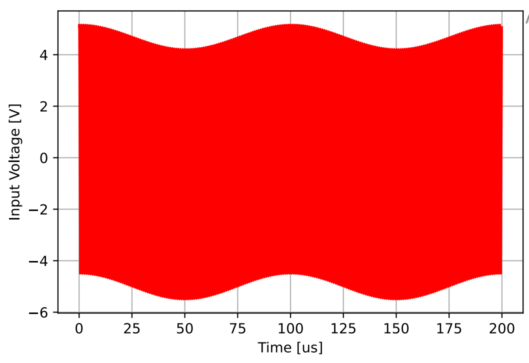
Table 15 – Diode Demodulator Output Spectrum: YalRF x ADS x QucsStudio

Frequency [kHz]	**YalRF**	**ADS**	**QucsStudio**
DC	3.800	3.798	3.800
10	$0.460\angle-3.1°$	$0.460\angle-3.1°$	$0.460\angle-3.1°$
990	$0.008\angle-112.2°$	$0.008\angle-112.6°$	$0.008\angle-112.0°$
1000	$0.108\angle-84.3°$	$0.108\angle-84.3°$	$0.108\angle-84.3°$
1010	$0.008\angle-49.1°$	$0.008\angle-49.6°$	$0.008\angle-49.3°$
1990	$0.004\angle-103.7°$	$0.004\angle-104.6°$	$0.004\angle-104.0°$
2000	$0.051\angle-79.8°$	$0.051\angle-79.8°$	$0.051\angle-79.7°$
2010	$0.004\angle-40.6°$	$0.004\angle-41.6°$	$0.004\angle-41.0°$
2990	$0.002\angle-94.2°$	$0.002\angle-95.6°$	$0.002\angle-94.7°$
3000	$0.030\angle-75.0°$	$0.031\angle-74.9°$	$0.031\angle-74.9°$
3010	$0.002\angle-31.1°$	$0.002\angle-32.6°$	$0.002\angle-31.7°$

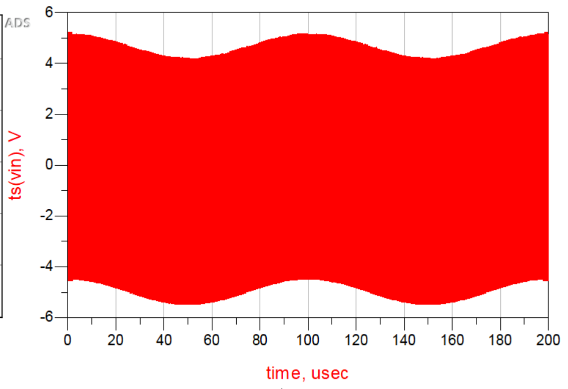
Source: Author.

The two-tones feature of the HB simulation requires more in-depth testing, but the results obtained so far indicate a good agreement with ADS and QucsStudio implementations. Using the two-tone analysis, mixers and amplifiers can be studied using YalRF. Performance improvements are still welcome, such as limiting the number of generated intermodulation products.

Figure 46 – Diode Demodulator Input Waveform



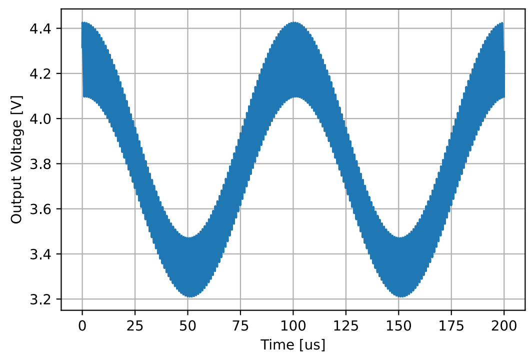
a) YalRF



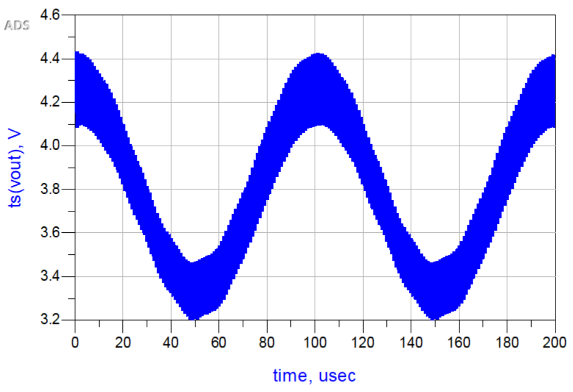
b) ADS

Source: Author.

Figure 47 – Diode Demodulator Output Waveform



a) YalRF



b) ADS

Source: Author.

6 CONCLUSION AND FUTURE WORKS

This dissertation presented the development of YalRF, a Python-written circuit simulation engine with support to Harmonic Balance analysis and its extension to use with autonomous circuits. After a broad review of simulation techniques, the DC, AC and transient algorithms were implemented into YalRF. A more in-depth review of the HB technique was conducted and a two-tones version of the algorithm was implemented. The Auxiliary Generator Technique was chosen to be implemented for the oscillator analysis into HB.

Several devices were implemented and verified through simulations. Linear resistors, capacitors and inductors are available for all analyses. Also, different types of voltage and current sources are accessible. The diode and Gummel-Poon BJT were the main focus of the nonlinear devices. The BJT was extensively used in DC, AC, transient and HB analysis. All the oscillator circuits shown employed BJTs, some with nonlinear parasitic capacitances and other nonidealities of the model. The diode and BJT exponential characteristic was limited by using a hyperbolic tangent to avoid overflow. The simulations showed very good agreement of YalRF with other commercial tools. A simple MOSFET implementation is also included in YalRF, but requires further development.

The HB algorithm was tested under multiple scenarios. The differential amplifier presented a standard single-tone analysis with multiple harmonics. A sweep of the input voltage was performed, while measuring the gain compression. The fast convergence of the HB for different levels of excitation showed a good stability of the implementation. A few oscillator structures were tested using the HB Oscillator Analysis. The frequency and magnitude of oscillators found by YalRF agreed well with the results from ADS and QucsStudio. For the circuits presented, usually a reasonable range of frequency guesses can be used as initial condition with the HB Oscillator Analysis still converging. Since several iterations of HB are run during an oscillator simulation, those tests also work as a very good indicator of the algorithm stability. There are cases where the oscillator analysis suffers to reach convergence for some particular frequency and voltage guess. The simulator then can become trapped into long source-stepping routines, attempting to reach the solution. This can greatly slow down the oscillator analysis. Finally, a two-tone simulation was performed on a diode demodulator circuit. The results obtained validated the implementation of the frequency mapping used for the quasiperiodic signals. Currently, only two-tones are available for HB, although the method can be generalized to multi-tones.

The Van Der Pol and CB Colpitts oscillators displayed some of the advantages of using Python for circuit simulation. The post-processing of the data generated by YalRF is greatly simplified due to the existence of several Python libraries for plotting and data analysis with a lot of online support. The possibility to integrate simulation and mathematical analysis of circuits in the same Python environment can be very attractive to the study and modeling of circuit topologies.

The field of circuit simulation is very multidisciplinary. For this work, reviews on the areas of differential equations, linear algebra, numerical methods, network analysis, computer programming and RF theory had to be conducted. Some device physics is required to understand the models of nonlinear elements. In this context, this dissertation worked as a very broad introduction to the field of circuit simulation, and contributed greatly to the author's formation aiming for future state-of-the-art work. The developed software contains a lot of the groundwork required for novel techniques to be explored in the future. Compared to other open-source alternatives, to the author's knowledge, it is the only simulator with HB analysis of oscillators.

6.1 FUTURE WORK

A number of features are planned to be included to the code base to increase the software's usefulness:

- Improvements in memory and speed usages, which were not a primary concern during development, such as the use of FFT in HB;
- Expand the two-tones implementation of HB to multi-tones;
- The implementation of envelope simulation for RF modulated signals;
- Use of Krylov iterative methods, preconditioners, and GPU techniques, to enhance the speed of HB solving;
- An important area on the analysis of RF circuits is the noise characteristic. The implementation of noise analysis and the measurement of phase noise in oscillators is considered an important next step to the project;

Using YalRF as a starting point, research now can be conducted into topics such as:

- Advanced methods for oscillator analysis using HB, such as using device-line measurements in order to characterize oscillators and improve initial condition to find the oscillatory regime;
- Usage of the HB algorithm for systems with multiple autonomous responses or both, autonomous and nonautonomous response, such as self-oscillating mixers;
- Phase noise modelling under multiple frameworks, taking into consideration the time-varying nature of oscillators;
- Integrate neural network based device models to YalRF and research into automated device modelling techniques for system level simulations, leveraging the extensive amount of machine learning tools available for Python;
- Evaluate the benefits of model order reduction techniques for HB;

REFERENCES

ANACONDA INC. **Miniconda**. [S.l.: s.n.]. Last Accessed: 2021-06-20. Available from: `https://docs.conda.io/en/latest/miniconda.html`.

APRILLE, T.J.; TRICK, T.N. Steady-state analysis of nonlinear circuits with periodic inputs. **Proceedings of the IEEE**, v. 60, n. 1, p. 108–114, 1972. DOI: `10.1109/PROC.1972.8563`.

ARSENOVIC, Alex. **scikit-rf: Open Source RF Engineering**. [S.l.: s.n.], 2009. `www.scikit-rf.org`. Last Accessed: 2021-03-24.

BAILY, Everett Minnich. **Steady-State Harmonic Analysis of Nonlinear Networks**. 1968. PhD thesis – Stanford University.

BANDALI, Bardia; GAD, Emad; BOLIC, Miodrag. Accelerated Harmonic-Balance Analysis Using a Graphical Processing Unit Platform. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 33, n. 7, p. 1017–1030, 2014. DOI: `10.1109/TCAD.2014.2304696`.

BIZZARRI, Federico; BRAMBILLA, Angelo; CODECASA, Lorenzo. Harmonic Balance Based on Two-Step Galerkin Method. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 63, n. 9, p. 1476–1486, 2016. DOI: `10.1109/TCSI.2016.2575978`.

CHANG, C. R. et al. Computer-aided analysis of free-running microwave oscillators. **IEEE Transactions on Microwave Theory and Techniques**, v. 39, n. 10, p. 1735–1745, 1991. DOI: `10.1109/22.88545`.

CLARKE, Kenneth K.; HESS, Donald T. **Communication Circuits: Analysis and Design**. First Edition. [S.l.]: Addison-Wesley, 1971.

EGAMI, S. Nonlinear, Linear Analysis and Computer-Aided Design of Resistive Mixers. **IEEE Transactions on Microwave Theory and Techniques**, v. 22, n. 3, p. 270–275, 1974. DOI: `10.1109/TMTT.1974.1128210`.

ELAD, D.; MADJAR, A.; BAR-LEV, A. A New Approach to the Analysis and Design of Microwave Feedback Oscillators. In: 1989 19th European Microwave Conference. [S.l.: s.n.], 1989. P. 369–374. DOI: `10.1109/EUMA.1989.333991`.

GAD, E. et al. A circuit reduction technique for finding the steady-state solution of nonlinear circuits. **IEEE Transactions on Microwave Theory and Techniques**, v. 48, n. 12, p. 2389–2396, 2000. DOI: `10.1109/22.898988`.

GUMMEL, H. K.; POON, H. C. An integral charge control model of bipolar transistors. **The Bell System Technical Journal**, v. 49, n. 5, p. 827–852, 1970. DOI: `10.1002/j.1538-7305.1970.tb01803.x`.

JAHN, Stefan et al. **Qucs - Technical Documentation**. [S.l.: s.n.], 2007. Available from: `http://qucs.sourceforge.net/docs/technical/technical.pdf`.

KUNDERT, K. S.; SANGIOVANNI-VINCENTELLI, A. Simulation of Nonlinear Circuits in the Frequency Domain. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 5, n. 4, p. 521–535, 1986. DOI: `10.1109/TCAD.1986.1270223`.

KUNDERT, K. S.; SORKIN, G. B.; SANGIOVANNI-VINCENTELLI, A. Applying harmonic balance to almost-periodic circuits. **IEEE Transactions on Microwave Theory and Techniques**, v. 36, n. 2, p. 366–378, 1988. DOI: `10.1109/22.3525`.

KUNDERT, Kenneth S.; WHITE, Jacob K.; SANGIOVANNI-VINCENTELLI, Alberto. **Steady-State Methods for Simulating Analog and Microwave Circuits**. First Edition. [S.l.]: Springer Science + Business Media Dordrecht, 1990.

LEE, Thomas H. **The Design of CMOS Radio-Frequency Integrated Circuits**. Second Edition. [S.l.]: Cambridge University Press, 2004.

LEE, Wai-Kong; ACHAR, Ramachandra; NAKHLA, Michel S. Dynamic GPU Parallel Sparse LU Factorization for Fast Circuit Simulation. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 26, n. 11, p. 2518–2529, 2018. DOI: `10.1109/TVLSI.2018.2858014`.

LIEBIG, Thorsten et al. openEMS – a free and open source equivalent-circuit (EC) FDTD simulation platform supporting cylindrical coordinates suitable for the analysis of traveling wave MRI applications. **International Journal of Numerical Modelling: Electronic Networks, Devices and Fields**, v. 26, n. 6, p. 680–696, 2013. DOI: `https://doi.org/10.1002/jnm.1875`.

MAAS, Stephen A. **Nonlinear Microwave and RF Circuits**. Second Edition. [S.l.]: Artech House, 2003.

MARGRAF, Michael. **QucsStudio - A Free and Powerful Circuit Simulator**. Available from: `http://dd6um.darc.de/QucsStudio/`.

MCCALLA, William J. **Fundamentals of Computer-Aided Circuit Simulation**. First Edition. [S.l.]: Springer, 1988.

NAJM, Farid N. **Circuit Simulation**. First Edition. [S.l.]: John Wiley & Sons, 2010.

NAKHLA, M.; VLACH, J. A piecewise harmonic balance technique for determination of periodic response of nonlinear systems. **IEEE Transactions on Circuits and Systems**, v. 23, n. 2, p. 85–91, 1976. DOI: `10.1109/TCS.1976.1084181`.

NGOYA, E.; LARCHEVEQUE, R. Envelop transient analysis: a new method for the transient and steady state analysis of microwave communication circuits and systems. In: 1996 IEEE

MTT-S International Microwave Symposium Digest. [S.l.: s.n.], 1996. 1365–1368 vol.3. DOI: `10.1109/MWSYM.1996.512189`.

NGOYA, Edouard; SUÁREZ, Almudena, et al. Steady state analysis of free or forced oscillators by harmonic balance and stability investigation of periodic and quasi-periodic regimes. **International Journal of Microwave and Millimeter-Wave Computer-Aided Engineering**, v. 5, n. 3, p. 210–223, 1995. DOI: `https://doi.org/10.1002/mmce.4570050308`.

PUPALAIKIS, Pete. **Signal Integrity**. [S.l.: s.n.], 2018. `https://github.com/TeledyneLeCroy/SignalIntegrity`. Last Accessed: 2021-03-24.

QUERE, R. et al. Large signal design of broadband monolithic microwave frequency dividers and phase-locked oscillators. **IEEE Transactions on Microwave Theory and Techniques**, v. 41, n. 11, p. 1928–1938, 1993. DOI: `10.1109/22.273418`.

RODRIGUES, P. J. C. A general mapping technique for Fourier transform computation in nonlinear circuit analysis. **IEEE Microwave and Guided Wave Letters**, v. 7, n. 11, p. 374–376, 1997. DOI: `10.1109/75.641425`.

ROGERS, John; PLETT, Calvin. **Radio Frequency Integrated Circuit Design**. First Edition. [S.l.]: Artech House, 2003.

SANDIA, National Laboratories. **Xyce - Parallel Electronic Simulator**. Available from: `https://xyce.sandia.gov/`.

SHICHMAN, H.; HODGES, D.A. Modeling and simulation of insulated-gate field-effect transistor switching circuits. **IEEE Journal of Solid-State Circuits**, v. 3, n. 3, p. 285–289, 1968. DOI: `10.1109/JSSC.1968.1049902`.

STEER, M. B. et al. fREEDA: An Open Source Circuit Simulator. In: 2006 International Workshop on Integrated Nonlinear Microwave and Millimeter-Wave Circuits. [S.l.: s.n.], 2006. P. 112–115. DOI: `10.1109/INMMIC.2006.283523`.

STROGATZ, Steven H. **Nonlinear Dynamic and Chaos with Applications to Physics, Biology, Chemistry and Engineering**. Second Edition. [S.l.]: Westview Press, 2015.

SUÁREZ, Almudena. **Analysis and Design of Autonomous Microwave Circuits**. First Edition. [S.l.]: John Wiley & Sons, 2009.

VIRTANEN, Pauli et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. **Nature Methods**, v. 17, p. 261–272, 2020. DOI: `10.1038/s41592-019-0686-2`.

XUAN, Yongnan; SNOWDEN, C.M. A New Generalised Approach to the Design of Microwave Oscillators. In: 1987 IEEE MTT-S International Microwave Symposium Digest. [S.l.: s.n.], 1987. P. 661–664. DOI: `10.1109/MWSYM.1987.1132497`.

ZHANG, Qi-Jun et al. Simulation and Automated Modeling of Microwave Circuits: State-of-the-Art and Emerging Trends. **IEEE Journal of Microwaves**, v. 1, n. 1, p. 494–507, 2021. DOI: `10.1109/JMW.2020.3033780`.

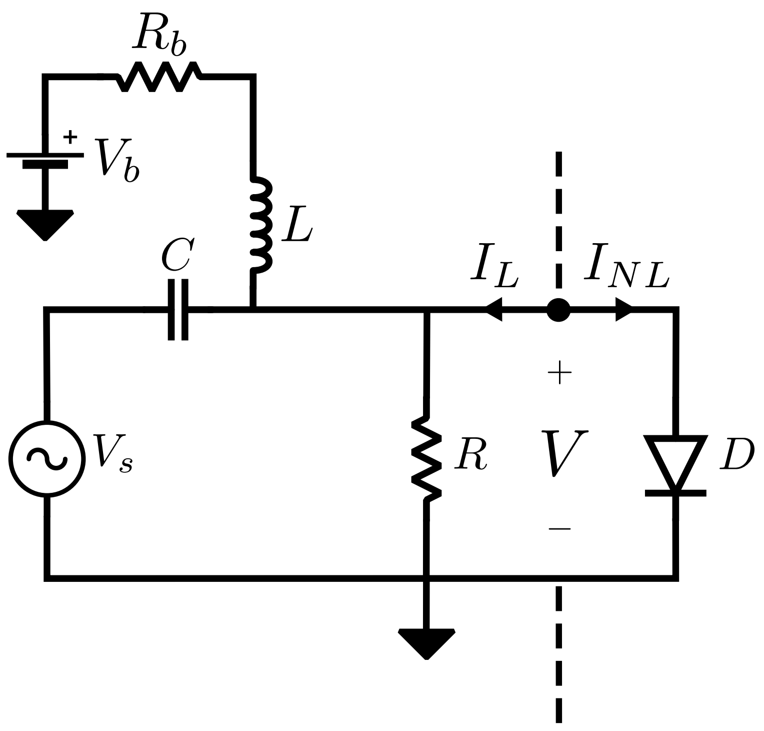
APPENDIX A – IMPLEMENTATION EXAMPLE OF THE HARMONIC BALANCE ALGORITHM

The Octave code below presents the numerical approach to the Harmonic Balance solution of a diode loaded circuit with an AC coupled source and a DC bias voltage present (see Figure 48). This circuit is an approximate model of the input of a common-emitter amplifier stage using a BJT, where the diode represents the base-emitter junction. It can be used in this configuration to evaluate the distortion that happens at a BJT, when its base is DC biased and excited by a strong signal. The presented approach does not use the Fourier Transform to perform the evaluations in frequency-domain, instead it employs a Bessel function expansion to obtain the Fourier coefficients of the exponential nonlinearity. The analysis was hard-coded for 3 harmonics. The frequency-domain derivative of the current phasors w.r.t the voltage phasors was calculated using a Secant approximation, which can be written as,

$$f'(x_2) \approx \frac{f(x_2) - f(x_1)}{x_2 - x_1} \tag{122}$$

where x_2 and x_1 must be very close together to increase the numerical precision.

Figure 48 – Input model of a common-emitter amplifier stage using a BJT.



Source: Author.

```
## Harmonic Balance solving of a Diode
## Clipping circuit with DC bias

```

```
# circuit definitions
Vb = 0.7;           # bias voltage
Vs = 0.075;         # input tone magnitude
f = 1e6;            # input tone frequency
w = 2 * pi * f;     # angular frequency
R = 50              # resistor value
Rb = 10;            # resistor in series with DC supply
Rs = 0;             # resistor in series with AC source
C = 1               # dc block
L = 1               # dc feed


# diode definitions
k   = 1.38064852e-23; # boltzmann constant
q   = 1.60217662e-19; # electron charge
T   = 300;            # temperature
Is = 1e-14;           # reverse bias saturation constant
phiT = k * T / q;     # thermal voltage

# harmonic balance definitions
K = 3;              # number of harmonics
S = 2 * K + 1;      # number of time points for the DFT

# linear current flowing through resistor and capacitor
Il_f = zeros(K+1,1);
Il_t = zeros(S,1);

# nonlinear current flowing through the diode
Inl_f = zeros(K+1,1);
Inl_t = zeros(S,1);

# initial voltage guess
VDC = 0.5;
VAC = 0.1;
V_f = complex([VDC; VAC; 0; 0]);

for i = 0:30

    printf('\n-------- Iteration: %d --------\n\n', i);

    # calculate the linear current at each harmonic
    Il_f(1) = V_f(1) / R + (V_f(1) - Vb) / Rb;
    Il_f(2) = V_f(2) / R + V_f(2) / (j*1*w*L + Rb) + ...
              (V_f(2) - Vs/2) * 1 / (1/(j*1*w*C) + Rs);
    Il_f(3) = V_f(3) / R + V_f(3) / (j*2*w*L + Rb) + ...
               V_f(3) * 1 / (1/(j*2*w*C) + Rs);
    Il_f(4) = V_f(4) / R + V_f(4) / (j*3*w*L + Rb) + ...
               V_f(4) * 1 / (1/(j*3*w*C) + Rs);

    # calculate the nonlinear current at each harmonic based
    # on the Fourier expansion of the function exp(x * cos(wt))
    # in Bessel functions
    x0 = real(V_f(1)) / phiT;
    x1 = 2 * real(V_f(2)) / phiT;
    Inl_f(1) = Is * exp(x0) * besseli(0,x1);
    Inl_f(2) = Is * exp(x0) * besseli(1,x1);
    Inl_f(3) = Is * exp(x0) * besseli(2,x1);
    Inl_f(4) = Is * exp(x0) * besseli(3,x1);

    # calculate time-domain current for verification purposes
    t = [0:S-1] / (f*S);
    I_t = Is*exp(x0) * (besseli(0,x1) + ...
           2*besseli(1,x1)*cos(1*w*t) + ...
           2*besseli(2,x1)*cos(2*w*t) + ...
           2*besseli(3,x1)*cos(3*w*t));

    ## to calculate dF/dV which is the Jacobian ##
    ## we need dIl/dV and dInl/dV ##

    # dIldV is equivalent to the admittance matrix and since
    # it is linear there are no terms converting harmonics
    # like dV1/dI2 (only block diagonal elems)
    Y0 = 1 / R + 1 / Rb;
    Y1 = 1 / R + 1 / (j*1*w*L + Rb) + 1 / (1/(j*1*w*C) + Rs);
    Y2 = 1 / R + 1 / (j*2*w*L + Rb) + 1 / (1/(j*2*w*C) + Rs);
    Y3 = 1 / R + 1 / (j*3*w*L + Rb) + 1 / (1/(j*3*w*C) + Rs);

    dIl0_dV0 = Y0;
    dIl1_dV1 = Y1;
    dIl2_dV2 = Y2;
    dIl3_dV3 = Y3;

    # incremental voltages to approximated dInl/dV
    inc = 0.01;
    x0inc = 1.01*x0;
    x1inc = 1.01*x1;

    # assembling the dInl_dV terms
    dInl0_dV0 = (Is*exp(x0inc)-Is*exp(x0))/inc * besseli(0,x1)
    dInl0_dV1 = Is*exp(x0)*(besseli(0,x1inc)-besseli(0,x1))/inc
    dInl0_dV2 = 0;
    dInl0_dV3 = 0;

    dInl1_dV0 = (Is*exp(x0inc)-Is*exp(x0))/inc * besseli(1,x1)
    dInl1_dV1 = Is*exp(x0)*(besseli(1,x1inc)-besseli(1,x1))/inc
    dInl1_dV2 = 0;
    dInl1_dV3 = 0;

    dInl2_dV0 = (Is*exp(x0inc)-Is*exp(x0))/inc * besseli(2,x1)
    dInl2_dV1 = Is*exp(x0)*(besseli(2,x1inc)-besseli(2,x1))/inc
    dInl2_dV2 = 0;
    dInl2_dV3 = 0;

    dInl3_dV0 = (Is*exp(x0inc)-Is*exp(x0))/inc * besseli(3,x1)
    dInl3_dV1 = Is*exp(x0)*(besseli(3,x1inc)-besseli(3,x1))/inc
    dInl3_dV2 = 0;
    dInl3_dV3 = 0;

    # calculate the elements that go into the Jacobian
    # matrix: dIl/dV + dInl/dV
    df0_dV0 = dIl0_dV0 + dInl0_dV0;
    df0_dV1 = dInl0_dV1;
    df0_dV2 = dInl0_dV2;
    df0_dV3 = dInl0_dV3;

    df1_dV0 = dInl1_dV0;
    df1_dV1 = dIl1_dV1 + dInl1_dV1;
    df1_dV2 = dInl1_dV2;
    df1_dV3 = dInl1_dV3;

    df2_dV0 = dInl2_dV0;
    df2_dV1 = dInl2_dV1;
    df2_dV2 = dIl2_dV2 + dInl2_dV2;
    df2_dV3 = dInl2_dV3;

    df3_dV0 = dInl3_dV0;
    df3_dV1 = dInl3_dV1;
    df3_dV2 = dInl3_dV2;
    df3_dV3 = dIl3_dV3 + dInl3_dV3;

    # assemblying the Jacobian matrix
    J = [df0_dV0 df0_dV1 df0_dV2 df0_dV3;
         df1_dV0 df1_dV1 df1_dV2 df1_dV3;
         df2_dV0 df2_dV1 df2_dV2 df2_dV3;
         df3_dV0 df3_dV1 df3_dV2 df3_dV3];

    # error between linear and nonlinear currents
    F = Inl_f + Il_f;
    V_f = V_f - pinv(J) * F;

end

# print outputs of interest (error, voltage and diode current)
err    = sum(abs(F))
Vmag   = abs(V_f)
Vph    = rad2deg(angle(V_f))
Inlmag = abs(Inl_f)
```

APPENDIX B – PYTHON CODE FOR YALRF TESBENCHES

This section presents the Python code used for some of the testbenches presented in Chapter 5. The first example is a simple resistive divider to introduce YalRF's API. The other scripts were only slightly simplified to remove excessive plotting shenanigans[1].

B.1 RESISTIVE DIVIDER

```python
from yalrf import YalRF

y = YalRF('Voltage Divider')

y.add_resistor('R1', 'n1', 'n2', 100)
y.add_resistor('R2', 'n2', 'gnd', 25)

v1 = y.add_vdc('V1', 'n1', 'gnd', 1)

dc = y.add_dc_analysis('DC1')

y.run('DC1')
y.print_dc_voltages('DC1')
y.print_dc_currents('DC1')
```

B.2 DIFFERENTIAL AMPLIFIER

```python
import time
import numpy as np
import matplotlib.pyplot as plt
from yalrf import YalRF, Netlist
from yalrf.Analyses import AC, HarmonicBalance, MultiToneHarmonicBalance

y = YalRF('Differential Amplifier')

# circuit parameters
ibias = 50e-3
vbias = 1.2
vcc = 5
```

[1] The complete version of the scripts can be found at: `https://github.com/victorpreuss/YalRF`

```python
vin = 60e-3
rload = 50

# VCC
i1 = y.add_idc('I1', 'nx', 'gnd', dc=vcc)
g1 = y.add_gyrator('G1', 'nx', 'nvcc', 'gnd', 'gnd', 1)

# vin1
i2 = y.add_iac('I2', 'ny', 'gnd', ac=vin, phase=0, freq=10e6)
i3 = y.add_idc('I3', 'ny', 'gnd', dc=vbias)
g2 = y.add_gyrator('G2', 'ny', 'nb1', 'gnd', 'gnd', 1)

# vin2
i4 = y.add_iac('I4', 'nz', 'gnd', ac=vin, phase=+180, freq=10e6)
i5 = y.add_idc('I5', 'nz', 'gnd', dc=vbias)
g3 = y.add_gyrator('G3', 'nz', 'nb2', 'gnd', 'gnd', 1)

# collector loads
r1 = y.add_resistor('R1', 'nvcc', 'nc1', rload)
r2 = y.add_resistor('R2', 'nvcc', 'nc2', rload)

# differential pair
q1 = y.add_bjt('Q1', 'nb1', 'nc1', 'ne')
q2 = y.add_bjt('Q2', 'nb2', 'nc2', 'ne')

q3 = y.add_bjt('Q3', 'nbx', 'ne', 'gnd')
q4 = y.add_bjt('Q4', 'nbx', 'nbx', 'gnd')

i6 = y.add_idc('I6', 'nbx', 'gnd', dc=ibias)

q1.options['Is'] = 8.11e-14
q1.options['Nf'] = 1
q1.options['Nr'] = 1
q1.options['Ikf'] = 0.5
q1.options['Ikr'] = 0.226
q1.options['Vaf'] = 113
q1.options['Var'] = 24
q1.options['Ise'] = 1.06e-11
q1.options['Ne'] = 2
```

```python
q1.options['Isc'] = 0
q1.options['Nc'] = 2
q1.options['Bf'] = 205
q1.options['Br'] = 4
q1.options['Cje'] = 2.95e-11
q1.options['Cjc'] = 1.52e-11
q1.options['Cjs'] = 0.

q2.options = q1.options
q3.options = q1.options
q4.options = q1.options

begin = time.time()

# run harmonic balance
hb = MultiToneHarmonicBalance('HB1', 10e6, 10)

vi = []
vout = []
V0 = None
for vin in np.arange(100e-6, 50.1e-3, 2e-3):

    # update input voltage
    i2.ac = vin/2
    i4.ac = vin/2

    # run harmonic balance
    converged, freqs, Vf, _, Vt = hb.run(y, V0)
    V0 = hb.V

    # get input and output information
    vi.append(hb.get_v('nb1')[1] - hb.get_v('nb2')[1])
    vout.append(hb.get_v('nc2')[1])

end = time.time()

vi = np.array(vi, dtype=complex)
vout = np.array(vout, dtype=complex)
gain = 20 * np.log10(np.abs(vout / vi))
```

```

dc1 = y.add_dc_analysis('DC1')
xdc = y.run('DC1')

y.print_dc_voltages('DC1')

print('Shape of V: {}'.format(hb.V.shape))
print('Running time: {}'.format(end-begin))
print('Ic of Q3: {}'.format(q3.oppoint['Ic']))

ac1 = y.add_ac_analysis('AC1', start=1e6,
                               stop=10e9,
                               numpts=2000,
                               sweeptype='linear')
xac = y.run('AC1', xdc)

freqs = y.get_freqs('AC1')
vi_ac = y.get_voltage('AC1', 'nb1') - y.get_voltage('AC1', 'nb2')
vout_ac = y.get_voltage('AC1', 'nc2')
gain_ac = 20 * np.log10(np.abs(vout_ac / vi_ac))

plt.figure()

vv = np.abs(vi) * 1e3
plt.plot(vv, gain)
plt.xlabel('Input Voltage [mV]')
plt.ylabel('Voltage Gain [dB]')
plt.title('HB Gain Compression')
plt.grid()
idx = np.argmin(np.abs(gain-gain[0]+1))
plt.plot(vv[0], gain[0], color='red', marker='o')
label = '{:.3f} dB\n@ {:.0f} uV'.format(gain[0], vv[0] * 1e3)
plt.annotate(label, (vv[0] , gain[0]), textcoords="offset points",
             xytext=(0,-45), ha='left', va='bottom', rotation=0, fontsize=15)
plt.plot(vv[idx], gain[idx], color='red', marker='o')
label = '{:.3f} dB\n@ {:.0f} mV'.format(gain[idx], vv[idx])
plt.annotate(label, (vv[idx] , gain[idx]), textcoords="offset points",
             xytext=(0,5), ha='left', va='bottom', rotation=0, fontsize=15)

```

```
plt.figure()
plt.semilogx(freqs, gain_ac, color='red')
plt.xlabel('Frequency [Hz]')
plt.ylabel('Voltage Gain [dB]')
plt.title('AC Small-Signal Gain')
plt.grid()

idx = np.argmin(np.abs(gain_ac-gain_ac[0]+3))
plt.plot(freqs[0], gain_ac[0], color='blue', marker='o')
label = '{:.3f} dB\n@ {:.0f} MHz'.format(gain_ac[0], freqs[0] / 1e6)
plt.annotate(label, (freqs[0] , gain_ac[0]), textcoords="offset points",
             xytext=(5,-35), ha='left', va='bottom', rotation=0, fontsize=15)
plt.plot(freqs[idx], gain_ac[idx], color='blue', marker='o')
label = '{:.3f} dB\n@ {:.0f} MHz'.format(gain_ac[idx], freqs[idx] / 1e6)
plt.annotate(label, (freqs[idx] , gain_ac[idx]), textcoords="offset points",
             xytext=(5,0), ha='left', va='bottom', rotation=0, fontsize=15)

plt.show()
```

B.3 VAN DER POL OSCILLATOR

```
import numpy as np
import matplotlib.pyplot as plt
from yalrf import YalRF, Netlist
from yalrf.Analyses import MultiToneHarmonicBalance

y = Netlist('Van Der Pol Oscillator')

# circuit parameters
C = 1
L = 1
alpha = 2.5

y.add_inductor('L', 'np', 'gnd', L)
y.add_capacitor('C', 'np', 'gnd', C)
r = y.add_resistor('R', 'np', 'gnd', -1/alpha)

# this element is i = alpha * v^3
```

```python
x = y.add_cubicnl('X1', 'np', 'gnd', alpha)

numharmonics = 20
freq = 0.13
V0 = 1.2

hb = MultiToneHarmonicBalance('HB1')
hb.options['maxiter'] = 100

plt.figure(figsize=(8,6))
plt.xlabel('Voltage [V]')
plt.ylabel('Current [A]')
plt.xlim((-1.5,1.5))
plt.ylim((-2,2))
plt.grid()

# sweep of alpha values for Van Der Pol Oscillator
for alpha in [0.1, 1.0, 1.8, 2.5]:

    r.R = -1/alpha
    x.alpha = alpha

    converged, freqs, Vf, _, _ = hb.run_oscillator(netlist=y,
                                                   f0=freq,
                                                   numharmonics=numharmonics,
                                                   V0=V0,
                                                   node='np')

    # calculate the inductor current as: I = V / (jwL)
    v = hb.get_v('np')
    i = np.zeros(v.shape, dtype=complex)
    for idx in range(1,len(freqs)):
        i[idx] = v[idx] / (1j * (2 * np.pi * freqs[idx]) * L)

    t, vt = hb.convert_to_time(v)
    t, it = hb.convert_to_time(i)

    plt.plot(vt, it, label=r'$\alpha$ = {:.1f}'.format(alpha))

plt.legend(loc='upper right')
plt.show()
```

B.4 COMMON-BASE COLPITTS OSCILLATOR

The sequence of scripts represent the Jupyter Notebook cells used for this example.

```python
import time
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
from scipy.constants import k, e

from yalrf import YalRF, Netlist
from yalrf.Analyses import AC, MultiToneHarmonicBalance
```

B.4.1 Negative Resistance

```python
y = YalRF('Common-Base Colpitts Oscillator')

# circuit parameters
vcc = 5
vbias = 1
ibias = 1e-3
rl = 850
fosc = 50e6
l = 50e-9
ct = 1 / (l * (2 * np.pi * fosc) ** 2)

n = 0.3 # capactive divider ratio (feedback)
c1 = ct / (1 - n)
c2 = ct / n

# VCC
y.add_idc('I1', 'nx1', 'gnd', dc=vcc)
y.add_gyrator('G1', 'nx1', 'nvcc', 'gnd', 'gnd', 1)

# Vbias
```

```python
y.add_idc('I2', 'ny1', 'gnd', dc=vbias)
y.add_gyrator('G2', 'ny1', 'nb', 'gnd', 'gnd', 1)

# Ibias
y.add_idc('I3', 'gnd', 'ne', dc=ibias)

# AC supply
iin = y.add_iac('I4', 'nin', 'gnd', 1)

# DC feed and DC block
y.add_inductor('Lfeed', 'nvcc', 'nc', 1e-3)
y.add_capacitor('Cblk', 'nc', 'nin', 1e-6)

# passives
C1 = y.add_capacitor('C1', 'nc', 'ne', c1)
C2 = y.add_capacitor('C2', 'ne', 'gnd', c2)

# bjts
q1 = y.add_bjt('Q1', 'nb', 'nc', 'ne')

q1.options['Is'] = 1e-15
q1.options['Bf'] = 100

# run DC sim
dc1 = y.add_dc_analysis('DC1')
xdc = y.run('DC1')

y.print_dc_voltages('DC1')

# run AC sim
ac1 = y.add_ac_analysis('AC1', start=10e6,
                               stop=100e6,
                               numpts=100,
                               sweeptype='logarithm')
xac = y.run('AC1', xdc)

freqs = y.get_freqs('AC1')
vin = y.get_voltage('AC1', 'nin')
iin = iin.ac
```

```

Rin = np.real(vin / iin)
Xin = np.imag(vin / iin)

Rin_calc = - (ibias / (k * 300 / e)) / ((2 * np.pi * freqs)**2 * c1 * c2)
Xin_calc = np.imag(1 / (1j * (2 * np.pi * freqs) * c1) +
                   1 / (1j * (2 * np.pi * freqs) * c2))

freqs /= 1e6
plt.figure(figsize=(12,4))
plt.subplot(121)
plt.plot(freqs, Rin, label='Simulation')
plt.plot(freqs, Rin_calc, label='Calculated')
plt.grid()
plt.legend()
plt.xlabel('Frequency [MHz]')
plt.ylabel('$R_{in}$')
plt.subplot(122)
plt.plot(freqs, Xin, label='Simulation')
plt.plot(freqs, Xin_calc, label='Calculated')
plt.grid()
plt.legend()
plt.xlabel('Frequency [MHz]')
plt.ylabel('$X_{in}$')
```

B.4.2 Large-Signal Transconductance

```
vin = 10e-3
fin = 50e6
vcc = 5
vbias = 1
ibias = 1e-3

net = Netlist('Large Gm')

# VCC
net.add_idc('I1', 'nx', 'gnd', dc=vcc)
net.add_gyrator('G1', 'nx', 'nc', 'gnd', 'gnd', 1)

```

```python
# input voltage (DC+AC signals)
net.add_idc('I2', 'ny', 'gnd', dc=vbias)
iin = net.add_iac('I3', 'ny', 'gnd', ac=vin, freq=fin)
net.add_gyrator('G2', 'ny', 'nb', 'gnd', 'gnd', 1)

# bias current
net.add_idc('I4', 'gnd', 'ne', dc=ibias)

# decoupling capacitor
net.add_capacitor('C1', 'ne', 'gnd', 1e-6)

# BJT
q1 = net.add_bjt('Q1', 'nb', 'nc', 'ne')
q1.options['Is'] = 1e-15
q1.options['Bf'] = 100

hb = MultiToneHarmonicBalance('HB1', fin, 20)
hb.options['maxiter'] = 100

vi = np.arange(100e-6, 301e-3, 5e-3)
ic = np.zeros(vi.shape)
V0 = None
i = 0
for vin in vi:

    # update input voltage
    iin.ac = vin

    # run harmonic balance
    converged, freqs, Vf, _, _ = hb.run(net, V0)
    V0 = hb.V

    # get time-domain current waveform from BJT and convert to frequency
    ic_td = np.array(q1.Ic, dtype=complex)
    ic_fd = hb.DFT @ ic_td
    ic[i] = np.abs(ic_fd[1] + 1j * ic_fd[2])
    i += 1

# small-signal BJT transconductance
```

```python
gm = ibias / (k * 300 / e)

plt.figure()
plt.plot(vi * 1e3, ic / vi / gm, label='Simulation')
plt.plot(vi[vi>=50e-3] * 1e3, 2 * ibias / vi[vi>=50e-3] / gm,
         label='Approximation')
plt.grid()
plt.legend()
plt.xlabel('$Vin$ [mV]')
plt.ylabel('$G_m / g_m$')
plt.show()
```

B.4.3 Oscillator Analsysis

```python
y = Netlist('HB Oscillator Analysis')

# circuit parameters
vcc = 5
vbias = 1
ibias = 1e-3
rl = 850
fosc = 50e6
l = 50e-9
ct = 1 / (l * (2 * np.pi * fosc) ** 2)

n = 0.3 # capactive divider ratio (feedback)
c1 = ct / (1 - n)
c2 = ct / n

# rs = l ** 2 * (2 * np.pi * fosc) ** 2 / rl

numharmonics = 20

f0 = 1 / (2 * np.pi * np.sqrt(l * ct))
V0 = 2 * ibias * rl * (1 - n)

# declare harmonic balance solver
hb = MultiToneHarmonicBalance('HB1')
hb.options['maxiter'] = 100
```

```python

# VCC
y.add_idc('I1', 'nx1', 'gnd', dc=vcc)
y.add_gyrator('G1', 'nx1', 'nvcc', 'gnd', 'gnd', 1)

# Vbias
y.add_idc('I2', 'ny1', 'gnd', dc=vbias)
y.add_gyrator('G2', 'ny1', 'nb', 'gnd', 'gnd', 1)

# Ibias
y.add_idc('I3', 'gnd', 'ne', dc=ibias)

# passives
# y.add_resistor('Rs', 'nvcc', 'nvccx', rs)
y.add_resistor('Rl', 'nvcc', 'nc', rl)
y.add_inductor('L1', 'nvcc', 'nc', l)
C1 = y.add_capacitor('C1', 'nc', 'ne', c1)
C2 = y.add_capacitor('C2', 'ne', 'gnd', c2)

# bjts
q1 = y.add_bjt('Q1', 'nb', 'nc', 'ne')

q1.options['Is'] = 1e-15
q1.options['Bf'] = 100

# single iteration
converged, freqs, Vf, _, _ = hb.run_oscillator(netlist=y,
                                               f0=f0,
                                               numharmonics=numharmonics,
                                               V0=V0,
                                               node='nc')

hb.plot_v('nc')
hb.plot_v('ne')
plt.figure()
plt.plot(q1.Ic)
plt.show()
```

B.4.4 Oscillator Parameter Sweep

```python
y = Netlist('Oscillator Parameter Sweep')

# circuit parameters
vcc = 5
vbias = 1
ibias = 1e-3
rl = 850
fosc = 50e6
l = 50e-9
ct = 1 / (l * (2 * np.pi * fosc) ** 2)

n = 0.3 # capactive divider ratio (feedback)
c1 = ct / (1 - n)
c2 = ct / n

# rs = l ** 2 * (2 * np.pi * fosc) ** 2 / rl

numharmonics = 20

f0 = 1 / (2 * np.pi * np.sqrt(l * ct))
V0 = 2 * ibias * rl * (1 - n)

# declare harmonic balance solver
hb = MultiToneHarmonicBalance('HB1')
hb.options['maxiter'] = 100

# VCC
y.add_idc('I1', 'nx1', 'gnd', dc=vcc)
y.add_gyrator('G1', 'nx1', 'nvcc', 'gnd', 'gnd', 1)

# Vbias
y.add_idc('I2', 'ny1', 'gnd', dc=vbias)
y.add_gyrator('G2', 'ny1', 'nb', 'gnd', 'gnd', 1)

# Ibias
y.add_idc('I3', 'gnd', 'ne', dc=ibias)

```

```python
# passives
# y.add_resistor('Rs', 'nvcc', 'nvccx', rs)
y.add_resistor('Rl', 'nvcc', 'nc', rl)
y.add_inductor('L1', 'nvcc', 'nc', l)
C1 = y.add_capacitor('C1', 'nc', 'ne', c1)
C2 = y.add_capacitor('C2', 'ne', 'gnd', c2)

# bjts
q1 = y.add_bjt('Q1', 'nb', 'nc', 'ne')

q1.options['Is'] = 1e-15
q1.options['Bf'] = 100

# single iteration
converged, freqs, Vf, _, _ = hb.run_oscillator(netlist=y,
                                               f0=f0,
                                               numharmonics=numharmonics,
                                               V0=V0,
                                               node='nc')

hb.plot_v('nc')
hb.plot_v('ne')
plt.figure()
plt.plot(q1.Ic)
plt.show()

# start timing the sweep
begin = time.time()

# loop varying 'n'
freqs = []
vtank = []
vtank2 = []
vtank3 = []
ipk = []
pwr = []
vtank_calc = []
eff = []
nvec = np.geomspace(0.05, 0.6, 20)
```

```python
iter_cnt = 0
for n in nvec:
    # update netlist
    C1.C = ct / (1 - n)
    C2.C = ct / n

    # run oscillator analysis
    if iter_cnt == 0:
        hb.run_oscillator(y, f0, numharmonics, V0, 'nc')
    else:
        hb.run_oscillator(y, f0, numharmonics, V0, 'nc', useprev=True)

    # sample the results
    freqs.append(hb.freq / 1e6)
    ipk.append(np.max(q1.Ic))
    vtank2.append(np.abs(hb.get_v('nc')[2]) / np.abs(hb.get_v('nc')[1]))
    vtank3.append(np.abs(hb.get_v('nc')[3]) / np.abs(hb.get_v('nc')[1]))
    vtank.append(np.abs(hb.get_v('nc')[1]))
    vtank_calc.append(2 * ibias * rl * (1 - n))
    eff.append(np.abs(hb.get_v('nc')[1])**2 / (2 * rl) / (vcc * ibias))

    iter_cnt += 1

# stop timing the sweep
end = time.time()
print('Shape of V: {}'.format(hb.V.shape))
print('Running time: {}'.format(end-begin))

plt.figure()
plt.plot(nvec, np.array(eff) * 100)
plt.xlabel('$n$')
plt.ylabel('Efficiency [%]')
plt.grid()

plt.figure()
plt.plot(nvec, vtank, label='Simulated')
plt.plot(nvec, vtank_calc, label='Calculated')
plt.xlabel('$n$')
plt.ylabel('$V_{out}$ [V]')
plt.grid()
plt.legend()

plt.figure()
plt.plot(nvec, vtank2)
plt.xlabel('$\eta$')
plt.ylabel('$V_{tank2}$ [V]')
plt.grid()

plt.figure()
plt.plot(nvec, vtank3)
plt.xlabel('$\eta$')
plt.ylabel('$V_{tank3}$ [V]')
plt.grid()

plt.figure()
plt.plot(nvec, ipk)
plt.xlabel('$\eta$')
plt.ylabel('$I_{c,pk}$ [A]')
plt.grid()

plt.figure()
plt.plot(nvec, freqs)
plt.xlabel('$\eta$')
plt.ylabel('$f_{osc}$ [MHz]')
plt.grid()
plt.show()
```

B.4.5 Efficiency Optimization

```python
y = Netlist('Oscillator Optimization')

# circuit parameters
vcc = 5
vbias = 1
ibias = 1e-3
rl = 850
fosc = 50e6
l = 50e-9
```

```python
ct = 1 / (l * (2 * np.pi * fosc) ** 2)

n = 0.2 # capactive divider ratio (feedback)
c1 = ct / (1 - n)
c2 = ct / n

# rs = l ** 2 * (2 * np.pi * fosc) ** 2 / rl

numharmonics = 20

f0 = 1 / (2 * np.pi * np.sqrt(l * ct))
V0 = 2 * ibias * rl * (1 - n)

# declare harmonic balance solver
hb = MultiToneHarmonicBalance('HB1')
hb.options['maxiter'] = 100

# VCC
y.add_idc('I1', 'nx1', 'gnd', dc=vcc)
y.add_gyrator('G1', 'nx1', 'nvcc', 'gnd', 'gnd', 1)

# Vbias
y.add_idc('I2', 'ny1', 'gnd', dc=vbias)
y.add_gyrator('G2', 'ny1', 'nb', 'gnd', 'gnd', 1)

# Ibias
y.add_idc('I3', 'gnd', 'ne', dc=ibias)

# passives
# y.add_resistor('Rs', 'nvcc', 'nvccx', rs)
y.add_resistor('Rl', 'nvcc', 'nc', rl)
y.add_inductor('L1', 'nvcc', 'nc', l)
C1 = y.add_capacitor('C1', 'nc', 'ne', c1)
C2 = y.add_capacitor('C2', 'ne', 'gnd', c2)

# bjts
q1 = y.add_bjt('Q1', 'nb', 'nc', 'ne')

q1.options['Is'] = 1e-15
```

```python
q1.options['Bf'] = 100

def objFunc(x, info):
    global hb
    # update netlist
    C1.C = ct / (1 - x)
    C2.C = ct / x
    # initial condition for oscillator
    f0 = 1 / (2 * np.pi * np.sqrt(l * ct))
    V0 = 2 * ibias * rl * (1 - x)
    # run oscillator analysis
    hb.run_oscillator(y, f0, numharmonics, V0, 'nc')
    info['itercnt'] += 1
    return - (np.abs(hb.get_v('nc')[1])**2 / (2 * rl) / (vcc * ibias))


begin = time.time()
bounds = (0.05, 0.5)
args = ({'itercnt' : 0},)
xopt = optimize.minimize_scalar(fun      = objFunc,
                                bounds   = bounds,
                                method   = 'bounded',
                                args     = args,
                                tol      = 1e-3)
end = time.time()

print('Shape of V: {}'.format(hb.V.shape))
print('Running time: {}'.format(end-begin))

print(xopt)
print(xopt.x)
hb.print_v('nc')
hb.plot_v('nc')
plt.show()
```

B.5 COMMON-COLLECTOR COLPITTS OSCILLATOR

```python
import time
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize

from yalrf import YalRF, Netlist
from yalrf.Analyses import HarmonicBalance, MultiToneHarmonicBalance


y = Netlist('Oscillator')

# circuit parameters
vcc = 5
c1 = 3.3e-12
c2 = 3.9e-12
r1 = 3e3
r2 = 6.8e3
re = 1.5e3
l1 = 6.944431e-9

# VCC
y.add_idc('I1', 'nx', 'gnd', dc=vcc)
y.add_gyrator('G1', 'nx', 'nvcc', 'gnd', 'gnd', 1)

# bias resistors
y.add_resistor('R1', 'nvcc', 'nb', r1)
y.add_resistor('R2', 'nb', 'gnd', r2)

# emitter resistance
y.add_resistor('RE', 'ne', 'gnd', re)

# capacitor feedback network
y.add_capacitor('C1', 'nb', 'ne', c1)
y.add_capacitor('C2', 'ne', 'gnd', c2)

# resonating inductor
y.add_inductor('L1', 'nind', 'gnd', l1)
```

```python

# dc feed and dc block (TODO: improve)
y.add_inductor('Lfeed', 'nvcc', 'nc', 1e-3)
y.add_capacitor('Cblk1', 'nb', 'nind', 1e-6)

# load connection
y.add_capacitor('Cblk2', 'nc', 'nl', 1e-6)
y.add_resistor('RL', 'nl', 'gnd', 50)

# bjt
q1 = y.add_bjt('Q1', 'nb', 'nc', 'ne')

q1.options['Is'] = 1e-15
q1.options['Bf'] = 100
q1.options['Br'] = 5
q1.options['Vaf'] = 60
q1.options['Var'] = 20

numharmonics = 50
freq = 1.2e9
V0 = 1

hb = MultiToneHarmonicBalance('HB1')
hb.options['maxiter'] = 100

converged, freqs, Vf, _, _ = hb.run_oscillator(netlist=y,
                                               f0=freq,
                                               numharmonics=numharmonics,
                                               V0=V0,
                                               node='nind')

hb.print_v('nind')
hb.plot_v('nind')
hb.plot_v('nl')
plt.show()
```

B.6 PELTZ OSCILLATOR

```python
import matplotlib.pyplot as plt
from yalrf import Netlist
from yalrf.Analyses import MultiToneHarmonicBalance

net = Netlist('Peltz Oscillator')

# circuit parameters
vcc = 10
r = 200e3
l = 0.5e-3
c = 10e-9
re = 50e3

# VCC
net.add_idc('I1', 'nx', 'gnd', dc=vcc)
net.add_gyrator('G1', 'nx', 'nvcc', 'gnd', 'gnd', 1)

# tank circuit
net.add_resistor('R1', 'nvcc', 'nb', r)
net.add_inductor('L1', 'nvcc', 'nb', l)
net.add_capacitor('C1', 'nvcc', 'nb', c)

# emitter resistance
net.add_resistor('RE', 'ne', 'gnd', re)

# bjts
q1 = net.add_bjt('Q1', 'nb', 'nvcc', 'ne')
q2 = net.add_bjt('Q2', 'nvcc', 'nb', 'ne')

q1.options['Is'] = 1e-16
q1.options['Bf'] = 200
q1.options['Br'] = 1
q2.options = q1.options.copy()

numharmonics = 10
freq = 80e3
V0 = 0.1
```

```python

hb = MultiToneHarmonicBalance('HB1')
hb.options['maxiter'] = 100

converged, freqs, Vf, _, _ = hb.run_oscillator(netlist=net,
                                               f0=freq,
                                               numharmonics=numharmonics,
                                               V0=V0,
                                               node='nb')

hb.print_v('nb')
hb.plot_v('nb')
plt.show()
```

B.7 DIODE DEMODULATOR

```python
import numpy as np
import matplotlib.pyplot as plt
from yalrf import YalRF, Netlist
from yalrf.Analyses import MultiToneHarmonicBalance
import time

y = YalRF('Diode AM Demodulator')

i1 = y.add_iac('I1', 'nx', 'gnd', ac=5, freq=1e6)
g1 = y.add_gyrator('G1', 'nx', 'nz', 'gnd', 'gnd', 1)

i2 = y.add_iac('I2', 'ny', 'gnd', ac=0.5, freq=10e3)
g2 = y.add_gyrator('G2', 'ny', 'n1', 'nz', 'gnd', 1)

r1 = y.add_resistor('R1', 'n1', 'nd', 50)
r2 = y.add_resistor('R2', 'n2', 'gnd', 5e3)

c1 = y.add_capacitor('C1', 'n2', 'gnd', 2.2e-9)

d1 = y.add_diode('D1', 'nd', 'n2')

d1.options['Is'] = 1e-15
```

```python
d1.options['N'] = 1
d1.options['Area'] = 1

begin = time.time()

hb = MultiToneHarmonicBalance('HB1', [1e6, 10e3], [20, 10])
hb.options['reltol'] = 1e-3
hb.options['abstol'] = 1e-6

converged, freqs, Vf, _, _ = hb.run(y)

end = time.time()

hb.print_v('n2')
# hb.plot_v('n2')
# plt.show()

print('Running time: {}'.format(end-begin))
print('HB problem size: {}'.format(hb.V.shape))
print('Number of frequency bins: {}'.format(freqs.shape))

vd = hb.get_v('nd')
vout = hb.get_v('n2')

# define time window to be plotted and time-step
T = 200e-6
tstep = 1. / (8. * np.max(freqs))
numpts = int(T / tstep)

# inefficient algorithm for time-domain waveform of quasiperiodic signal
t = np.zeros(numpts)
vt = np.zeros(numpts)
vdt = np.zeros(numpts)
for s in range(numpts):
    t[s] = s * tstep
    vt[s] = vout[0].real
    vdt[s] = vd[0].real
    for k in range(1, len(freqs)):
        f = freqs[k]
        vt[s] = vt[s] + vout[k].real * np.cos(2. * np.pi * f * t[s]) - \
                        vout[k].imag * np.sin(2. * np.pi * f * t[s])
        vdt[s] = vdt[s] + vd[k].real * np.cos(2. * np.pi * f * t[s]) - \
                          vd[k].imag * np.sin(2. * np.pi * f * t[s])

plt.figure()
plt.plot(t * 1e6, vt)
plt.grid()
plt.xlabel('Time [us]')
plt.ylabel('Output Voltage [V]')

plt.figure()
plt.plot(t * 1e6, vdt, 'r')
plt.grid()
plt.xlabel('Time [us]')
plt.ylabel('Input Voltage [V]')

plt.show()
```