Initial estimates for obtaining periodic-steady state solutions of free-running circuits

Liu, J.

Award date:
2009

Initial estimates for obtaining Periodic-Steady State solutions of free-running circuits Master's Thesis

Liu, Jie

Technische Universiteit Eindhoven, NXP Semiconductors

August 15, 2009

Authors' E-mail address : j.liu@student.tue.nl; jie.liu@nxp.com

Contents

Revision history:

Version	Date	Description
0.1	20090626	Chapter 1 and 2
0.2	20090708	add Chapter 3, part of Chapter 4
0.3	20090717	add part of Chapter 4 and 5
0.4	20090720	add section 2 in Chapter 5
0.5	20090722	an initial version
0.6	20090803	Jan's comments
0.7	20090808	Jan's 2nd comments
0.8	20090812	Theo's comments
0.9	20090814	Theo's 2nd comments

I will be available and pleased to discuss aspects from the thesis
prior to the exam on Wednesday August 19, 2009, at 14:00h.

Chapter 1

Introduction

1.1 Motivation

Integrated circuits are nowadays an important part of any electrical device. An example of an IC is shown in Figure 1.1. It is extremely expensive to change an

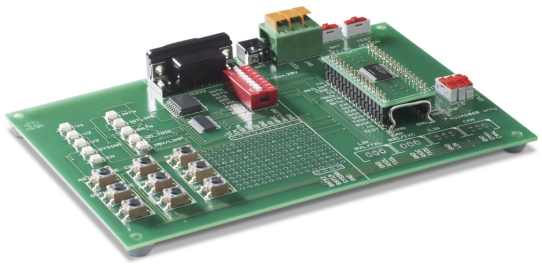


Figure 1.1: An Integrated Circuit

assembly line in a chip factory to the production of a new design; it may cost 1,000,000 euros - or even more. Hence, it is important that designing products are right-first-time. This however should not increase the time to the market.
A particular aspect that requires further study, and indeed the main topic of this thesis, is the numerical simulation of electrical circuits, in particular circuits that have oscillatory behaviour. Such electrical oscillators have important technological applications, like modulation and demodulation of radio signals, cell phone, chips on automatic and etc.
Simple examples include a phase-shift oscillator which is a simple sine wave electronic oscillator; an LC oscillator which consists of a capacitor and an inductor connected in parallel; and the Electron-Coupled Oscillator (ECO) which has very good

frequency stabilities. The behaviour of oscillators and oscillator systems nowadays are becoming much more complex.
My thesis deals with simulation aspects for so-called free-running, autonomous, oscillators. Oscillators one can divide in two different classes: *non-autonomous* and *autonomous* oscillators.
Non-autonomous (or *driven*) oscillators need a time-dependent input signal. A common situation is that the input signal is periodic, and the output signal is periodic with the same period as the input signal. In this case, the period T of the output signal is known a priori.
On the other hand, autonomous (or *free-running*) oscillators have no time-dependent input signal, which means that it is generally not possible to predict the period T a priori. Hence to simulate these oscillators numerically one needs to detect this period T and the corresponding solution. Simple long term time integration is a very stable and reliable method to solve this problem. However designers do not like to wait that long. We describe several methods that try to solve the problem "directly". However, all these methods are based around methods that use iterative improvements. To guarantee convergence one needs very good predictions for T and for the oscillation solution. With increasing frequencies this problem have become even more harder. My thesis deals with finding good predictions for T and for the oscillation solution. It also is a good example for solving a nonlinear eigenvalue and eigenvector problem.

Figure 1.2: A chip made by NXP

Figure 1.3: A few consumer products in which NXP chips are used

1.2 Overview of the thesis

In this thesis, methods for finding initial estimates to obtain a Periodic Steady State (PSS) of an electric circuit will be discussed. We will start in Chapter 2 by explaining how the equations that describe a circuit are derived. In the next chapters, I will give some basic insight in various circuit simulation techniques. In chapter 4, Several methods for PSS will be introduced. In chapter 5, I discuss a technical method for PSS analysis in detail. In chapter 6, I give some free-running (autonomous) oscillators examples. Finally, in chapter 7, I will arrive at some conclusions, and I do some recommendations for further research.

Chapter 2

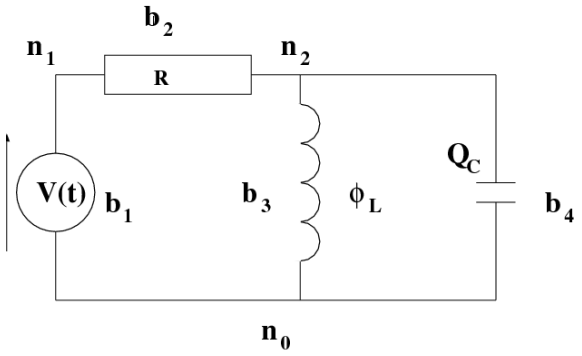
Circuit Equations

2.1 Introduction

In this chapter we introduce the equations which describe a given electronic circuit. We start with a brief introduction to electrical circuits where we present the viewpoint we take for the derivation of a mathematical model. In the next section we will describe the basic Kirchhoff's Voltage Law and Kirchhoff's Current Law. After that, we will give the description of Nodal Analysis, a method for constructing the mathematical model which describes the circuit, and a more general method for constructing this model, Modified Nodal Analysis (MNA).We end this chapter with a description of Circuit analysis, DC, AC analysis, Transient simulation and Harmonic Balance.

2.2 A Circuit

Basically, a circuit can be defined by branches and nodes. The branches, denoted by b_i, are the circuit elements, and the nodes, denoted by n_i, connect the elements to each other. We consider the following example.



This network has three nodes (n_0, n_1, n_2) and four branches, b_1 (containing a voltage source $V(t)$ between n_0 and n_1), b_2 (a resistor between n_1 and n_2), b_3 (a

inductor Φ_L between n_2 and n_0) and b_4 (a capacitor Q_c between n_2 and n_0).

We define the voltage difference across a branch b_i as v_{b_i} and the current flowing in a branch b_i as i_{b_i}. Here we also need to notice the directions. Note that we implicitly introduced directions (we will explain this more detailedly in the next section). Equations like Ohms Law for branch b_i, $v_{b_i} = R \cdot i_{b_i}$, imply that the voltage follows current, but when the direction of v_{b_i} is reversed the equation looks like $v_{b_i} = -R \cdot i_{b_i}$.

Next, we introduce the equations that reflect the topology of the circuit: These are called Kirchhoff's Laws.

2.3 Kirchhoff's Laws

The equations that reflect the topology of the circuit do not depend on the type of the branches, but only on the topology of the circuit, i.e. the way in which the branches are connected. These equations are given by two laws. These laws are, Kirchhoff's Current Law and Kirchhoff's Voltage Law. There are various formulations of these laws, which are all equivalent.

2.3.1 Kirchhoff's Current Law (KCL)

- The algebraic sum of the currents at a node is zero.

The KCL is derived from the charge continuity law. Which states that the integral of the current density $\overline{J}$, taken over any closed surface S (i.e. $\int_S \overline{J} \cdot \overline{dS}$ is the total varying current) is equal to the negative rate of change of the electric charge $-\frac{\partial q}{\partial t}$, contained in that closed surface,

$$\int_S \overline{J} \cdot \overline{dS} \quad = \quad -\frac{\partial q}{\partial t}.$$

The term $-\frac{\partial q}{\partial t}$ is either negligible or modeled by a capacitor. If we assume that the current flows only through conductors, we find

$$\begin{aligned} \int_S \overline{J} \cdot \overline{dS} &= \int_{b_1} \overline{J} \cdot \overline{dS} + \int_{b_2} \overline{J} \cdot \overline{dS} + \dots \\ &= i_{b_1} + i_{b_2} + \dots, \end{aligned}$$

since the integral of the current density over the cross section of the conductor is exactly the current flowing through that conductor.

Note that the KCL states that the sum of currents is zero across any closed surface. In particular, a closed surface may contain only one circuit node. In this case, we have the form of Kirchhoff's Current Law. This states that all currents entering a node add up to zero.

2.3.2 Kirchhoff's Voltage Law (KVL)

- The algebraic sum of the branch voltages v_{b_i} around a closed loop is zero:

$$\sum_{b_i \in \text{loop}} v_{b_i} \quad = \quad 0.$$

The KVL results from Faraday's law. Faraday's law, which states that the line integral of the electric field $\overline{E}$, taken over any closed loop is equal to the negative rate of variation of magnetic flux through the loop ($-\frac{\partial \Psi}{\partial t}$).

$$\oint \overline{E} \cdot d\bar{l} \quad = \quad -\frac{\partial \Psi}{\partial t}.$$

The last term is either assumed to be negligible or is already modeled by an inductor. Assume the branches $b_1, b_2, \ldots b_b$ form a closed loop. We have the following relation

$$\oint \overline{E} \cdot d\bar{l} \quad = \quad \int_{b_1} \overline{E} \cdot d\bar{l} + \int_{b_2} \overline{E} \cdot d\bar{l} + \ldots + \int_{b_b} \overline{E} \cdot d\bar{l}. \tag{2.1}$$

Each $\int_{b_i} \overline{E} \cdot \overline{dl}$ is a voltage drop v_{b_i} over the specific branch. Assuming that we have no coil involved, (2.1) results in

$$v_{b_1} + v_{b_2} + \ldots v_{b_b} \quad = \quad 0.$$

2.4 Nodal Analysis

Nodal analysis is a simple method for constructing circuit equations from circuits.

First we take a look at the topology of a circuit. The topology can be described by the *adjacency matrix*. The rows of this matrix correspond to circuit elements (branches) and the columns to circuit nodes. We attach a(n arbitrary) direction to each branch. Assume there are b branches and n nodes. Now we can define the adjacency matrix $\tilde{A} \in \mathbf{R}^{b \times n}$

$$\tilde{A}(i,j) := \begin{cases} 1 & \text{if node } n_j \text{ is the "from" node of branch } b_i \\ -1 & \text{if node } n_j \text{ is the "to" node of branch } b_i \\ 0 & \text{if node } n_j \text{ and branch } b_i \text{ are not connected} \end{cases}$$

Kirchhoff's Voltage Law is equivalent to the following assertion: To every node n_i a nodal voltage v_{n_i} can be assigned in such a way that for every branch b_j the branch voltage differences is given by:

$$v_{b_j} = v_{n_{j+}} - v_{n_{j-}}, \tag{2.2}$$

where $v_{n_{j+}}$ denotes the voltage in the positive node n_{j+} and v_{j-} denotes the voltage in the negative node n_{j-}. This voltage is unique except for a common constant.

We define the following vector $\tilde{\mathbf{v}}_n(t)$ as the vector with the assigned nodal voltages on time t and $\mathbf{v}_b(t)$ as the vector with the branch voltages on time t, so:

$$\begin{aligned} \tilde{\mathbf{v}}_n(t) &:= (v_{n_1}(t), v_{n_2}(t), \ldots v_{n_n}(t))^T, \\ \mathbf{v}_b(t) &:= (v_{b_1}(t), v_{b_2}(t), \ldots v_{b_b}(t))^T. \end{aligned}$$

With this notations we can write (2.2) as:

$$\tilde{A}\tilde{\mathbf{v}}_n(t) = \mathbf{v}_b(t). \tag{2.3}$$

Note that this implies that KVL holds.

We know that these equations have infinitely many solutions due to the common constant in the node voltages. If we assign one node to be the ground or reference node we can set the assigned voltage of this node to zero. Without loss of generality we take v_{n_n} as the ground node, thus $v_{n_n} = 0$. We can remove v_{n_n} from the vector of unknown node voltages $\tilde{\mathbf{v}}_n(t)$ and call the remaining vector $\mathbf{v}_n(t)$. Now we drop the column corresponding to node n from $\tilde{A}$. The result is a matrix $A \in \mathbf{R}^{b\times(n-1)}$. Using this reduced adjacency matrix A we have the relation

$$A\mathbf{v}_n(t) = \mathbf{v}_b(t). \tag{2.4}$$

It appears that we can use the same matrix A for the Kirchhoff's Current Law. We define $\mathbf{i}_b(t)$ as the vector with all branch currents on time t:

$$\mathbf{i}_b(t) := (i_{b_1}(t), i_{b_2}(t), \ldots i_{b_b}(t))^T.$$

Each row of the matrix A^T corresponds to a non-reference node, and it has a 1 entry corresponding to a branch leaving the node and a -1 entry corresponding to

a branch entering the node. When a row is multiplied with the vector of currents, we get in fact the Kirchhoff's Current Law equations.

$$A^T \mathbf{i}_b(t) = \mathbf{0}. \tag{2.5}$$

These equations, (2.4) and (2.5), are called *nodal equations*. The next step is to combine the equations that reflect the topology with the branch equations. A condition for nodal analysis is that there exists functions $\hat{\mathbf{q}}(\mathbf{v}_b, t)$ and $\hat{\mathbf{j}}(\mathbf{v}_b, t)$ such that all equations can be written as:

$$\mathbf{i}_b(t) = \frac{d}{dt}(\hat{\mathbf{q}}(\mathbf{v}_b, t)) + \hat{\mathbf{j}}(\mathbf{v}_b, t). \tag{2.6}$$

We now have the following equations

$$\begin{aligned} A\mathbf{v}_n(t) &= \mathbf{v}_b(t), && (2.7)\\ A^T \mathbf{i}_b(t) &= \mathbf{0}, && (2.8)\\ \mathbf{i}_b(t) &= \frac{d}{dt}(\hat{\mathbf{q}}(\mathbf{v}_b, t)) + \hat{\mathbf{j}}(\mathbf{v}_b, t). && (2.9) \end{aligned}$$

The equations we obtained from the KVL (2.7) can be substituted in the Branch Current Relations (2.9). This results in

$$\begin{aligned} A^T \mathbf{i}_b(t) &= \mathbf{0}, && (2.10)\\ \mathbf{i}_b(t) &= \frac{d}{dt}\hat{\mathbf{q}}(A\mathbf{v}_n, t) + \hat{\mathbf{j}}(A\mathbf{v}_n, t). && (2.11) \end{aligned}$$

This system can be made even more compact by substituting (2.11) in equation (2.10), and obtain the following compact system :

$$A^T \left(\frac{d}{dt}\hat{\mathbf{q}}(A\mathbf{v}_n, t) + \hat{\mathbf{j}}(A\mathbf{v}_n, t) \right) = \mathbf{0}. \tag{2.12}$$

An example will be used in the next section to illustrate these equations. Notice that for linear $\mathbf{q}$ and $\mathbf{j}$, we see that A^TCA and A^TGA matrices occur. Clearly A^TCA and A^TGA are $(n-1) \times (n-1)$ non-negative definite. The system (2.12) is a DAE(Differential Algebraic Equation), and for the solvability of a DAE, we refer to [7].
Nodal Analysis has some severe restrictions, the most important one being that it requires all branch equations to fit in the form (2.6). In words, Nodal Analysis requires branches to be current-defined and voltage-controlled. However not

all branch equations are of that form. A simple voltage source, for instance, is voltage-defined. The branch equation for a 5 Volt voltage source simply is:

$$v_{b_j} = 5.$$

Because of this, most circuit simulators use Modified Nodal Analysis, a variant of nodal analysis.

2.5 Modified Nodal Analysis

A short description of Modified Nodal Analysis (MNA) is:

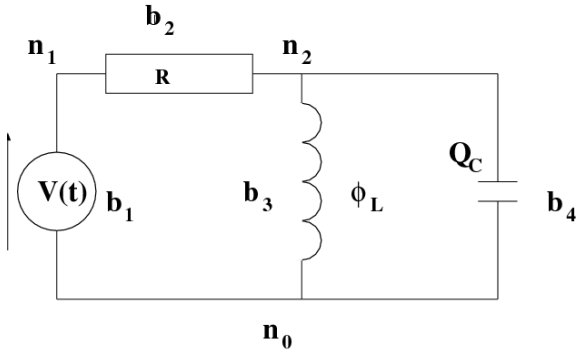
1. Write node equations by applying KCL to each node except for the ground node:
$$A^T \mathbf{i}_b(t) = \mathbf{0},$$
where the vector $\mathbf{i}_b(t)$ represents again the branch current vector, and the matrix A is the reduced adjacency matrix.

2. Replace the currents i_k of the elements which are current-defined and voltage-controlled by the voltage-current relations of these elements in the above equation (as before).

3. Treat the currents i_k of current-controled elements as unknowns additionally to the nodal voltages. Inductors typically are current controlled: the voltage difference is the time-derivative of the flux function that itself depends on the current. The most simple flux function is $\phi = L\, i_L$.
Also voltage sources are considered as "current controlled"; however in most cases the voltage source is a constant function of the current through the voltage source.

4. Add the voltage-current relations for all current-controlled elements. Those relations will implicitly define the i_k.

To explain Modified Nodal Analysis, we consider the following example.



We treat node n_0 as the reference node. Branch b_2 is a linear resistor and branch b_4 is a linear capacitor. The matrix A is given by

$$A^T = \begin{array}{c} \\ n_1 \\ n_2 \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & b_3 & b_4 \end{array} \\ \begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & +1 & -1 & -1 \end{pmatrix} \end{array}.$$

This results in

$$A^T \mathbf{i}_b(t) = \begin{bmatrix} i_{b_1}(t) - i_{b_2}(t) \\ i_{b_2}(t) - i_{b_3}(t) - i_{b_4}(t) \end{bmatrix} = \mathbf{0},$$

which are exactly the equations of the KCL for all nodes except the ground node. The next step is to replace the currents of the voltage-controlled elements by their current-voltage relation. The current i_{b_1} and i_{b_3} are not voltage-controlled, but current i_{b_2} and i_{b_4} are voltage-controlled. The latter ones give the following relations.

$$i_{b_2}(t) = R^{-1}\left(v_{n_2}(t) - v_{n_1}(t)\right), \tag{2.13}$$

$$i_{b_4}(t) = -\frac{d}{dt} C v_{n_2}(t). \tag{2.14}$$

Now we substitute the equations (2.13) and (2.14) of the voltage-controlled elements into the KCL equations. The resulting system is

$$\begin{aligned} i_{b_1}(t) - R^{-1}\left(v_{n_2}(t) - v_{n_1}(t)\right) &= 0, \\ R^{-1}\left(v_{n_2}(t) - v_{n_1}(t)\right) - i_{b_3}(t) + C\frac{d}{dt} v_{n_2}(t) &= 0. \end{aligned}$$

We treat i_{b_1} and i_{b_3} as additional unknowns.

The next step is to add the voltage-current relations for all current-controlled elements. Thus we add the relations of the voltage source and the inductor.

$$\begin{aligned} v_{n_1}(t) &= V(t), \\ v_{n_2}(t) &= \frac{d}{dt}\Phi_L(i_{b_3}(t)). \end{aligned}$$

Combining the equations results in the following system of equations:

$$\begin{aligned} i_{b_1}(t) - R^{-1}\left(v_{n_2}(t) - v_{n_1}(t)\right) &= 0, \\ R^{-1}\left(v_{n_2}(t) - v_{n_1}(t)\right) - i_{b_3}(t) + C\frac{d}{dt} v_{n_2}(t) &= 0, \\ v_{n_1}(t) &= V(t), \\ v_{n_2}(t) &= \frac{d}{dt}\Phi_L(\mathbf{i}_{b_3}(t)). \end{aligned}$$

Clearly, this can be cast in the general form of circuit equations:

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}, t) + \mathbf{j}(\mathbf{x}, t) = \mathbf{0}, \tag{2.15}$$

where $\mathbf{x}$ is the vector of unknowns. In our example:

$$\mathbf{x} \quad = \quad (v_{n_1}(t), v_{n_2}(t), i_{b_1}(t), i_{b_3}(t))^T,$$

and the functions $\mathbf{q}(\mathbf{x}, t)$ and $\mathbf{j}(\mathbf{x}, t)$ are given by:

$$\mathbf{q}(\mathbf{x}, t) = \begin{bmatrix} 0 \\ Cv_{n_2}(t) \\ 0 \\ -\Phi(i_{b_3}(t)) \end{bmatrix}, \ \mathbf{j}(\mathbf{x}, t) = \begin{bmatrix} i_{b_1}(t) - R^{-1}\left(v_{n_2}(t) - v_{n_1}(t)\right) \\ R^{-1}\left(v_{n_2}(t) - v_{n_1}\right) - i_{b_3}(t) \\ v_{n_1}(t) - V(t) \\ v_{n_2}(t) \end{bmatrix}.$$

Equation (2.15) is a so-called *differential-algebraic equation*, or DAE for short. This means that the system consists of both differential equations and algebraic equations. In our example, the first equation

$$i_{b_1}(t) - R^{-1}\left(v_{n_2}(t) - v_{n_1}(t)\right) \quad = \quad 0,$$

is an algebraic equation, while the last equation

$$-\frac{d}{dt}\Phi(i_{b_3}(t)) + v_{n_2}(t) \quad = \quad 0,$$

is a differential equation.
In practise, it is better to multiply the equation for i_{b3} with a -1. It makes the matrix $\frac{\partial q}{\partial x}$ non-negative definite. But the matrix $\frac{\partial j}{\partial x}$ will loose it's symmetry.
The above approach is in fact charge-oriented MNA used in industrial simulation by analogue circuit simulators like Pstar[1] , Titan[2] and Spectre[3] . It models currents of capacitors by $\frac{d}{dt}\mathbf{q}$ (where $\mathbf{q} = \mathbf{q}(\mathbf{v})$) in contrast to old Spice-like approaches.

2.6 DC Analysis

As described in the previous, the circuit equations typically have the following form:

$$\frac{d}{dt}\mathbf{q}(t, \mathbf{x}) + \mathbf{j}(t, \mathbf{x}) = \mathbf{0}. \tag{2.16}$$

From the system of equations 2.16, different kinds of analyses can be performed: DC analysis, AC analysis , Transient Analysis and Harmonic Balance. First, we indroduce DC Analysis.
DC is the abbreviation of Direct Current. In DC analysis one is looking for a state where all circuit variables do not vary with time. This is not always possible, but for the moment we assume that a DC equilibrium state exists. This kind of analysis is done for a *resistive* network, that means a network with no capacitors or inductors.

[1] Pstar is a in-house circuit simulator of NXP Semiconductors
[2] Simulator of Infineon AG
[3] Commercial simulator from Cadence Design Systems, Inc.

(In practice these elements will be neglected). Thus we replace the $\mathbf{q}(\mathbf{x}, t)$ by $\mathbf{0}$. We define DC equation as

$$\mathbf{j}(\mathbf{x}) \quad = \quad \mathbf{0}.$$

In general, this system of equations is nonlinear and has to be solved iteratively, for instance by the Newton-Raphson method. This method will be explained in Chapter 4.
Note that the DC-solution always satisfies the algebraic equations of the system of DAEs: these equations did not change. All other analyses can be viewed as analyses to study the effects due to perturbations to a system that initially was in DC-state (f.i. by adding time-dependent sources). For transient analyses the DC solution will be the initial solution at time $t = 0$.

2.7 AC Analysis

When performing small-signal or AC analysis, one is interested in what happens after adding a small time varying signal $\mathbf{e}(t)$ to a circuit that is in DC situation. This means that we include capacitors and inductors to our resistive network. After linearization around the DC solution, or another "operating point" , a *linear dynamic* network will show up. We start with considering

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}_0 + \mathbf{x}(t)) + \mathbf{j}(\mathbf{x}_0 + \mathbf{x}(t)) \quad = \quad \mathbf{e}(t)$$

and assume $\|\mathbf{x}\| \ll 1$ and $\|\frac{d}{dt}\mathbf{x}\| = O(\|\mathbf{x}\|)$. We expand the functions $\mathbf{q}$ and $\mathbf{j}$ around the operating point $\mathbf{x}_0$. If the time varying components of the signals are "small" enough, the first two terms of the Taylor expansion will accurately approximate the functions, and we get

$$\frac{d}{dt}\left[\mathbf{q}(\mathbf{x}_0) + \left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) + \ldots\right] + \mathbf{j}(\mathbf{x}_0) + \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) + \ldots \quad = \quad \mathbf{e}(t).$$

Next, we perform differentiation with respect to time and obtain

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}_0) + \frac{d}{dt}\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) + \ldots + \mathbf{j}(\mathbf{x}_0) + \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) + \ldots \quad = \quad \mathbf{e}(t).$$

But $\mathbf{q}(\mathbf{x}_0)$ is constant in time, and therefore $\frac{d}{dt}\mathbf{q}(\mathbf{x}_0) = 0$. In addition the Jacobian matrix $\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0}$ does not depend on time. Each "$+\ldots$" consists of higher order terms of $\mathbf{x}$ and possibly of time derivation of $\mathbf{x}$. This whole contribution can be treated as $O(\|\mathbf{x}\|^2)$ effect. Thus, in first order, we have

$$\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \frac{d}{dt}\mathbf{x}(t) + \mathbf{j}(\mathbf{x}_0) + \left.\frac{\partial \tilde{\mathbf{j}}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) \quad = \quad \mathbf{e}(t).$$

Move $\mathbf{j}(\mathbf{x}_0)$ to the right. Hence the above equation reduces to

$$\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \frac{d}{dt}\mathbf{x}(t) + \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) \quad = \quad \tilde{\mathbf{e}}(t) = \mathbf{e}(t) - \mathbf{j}(\mathbf{x}_0).$$

Using the notation

$$\begin{aligned} C &= \left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \\ \text{and} \quad G &= \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0}, \end{aligned}$$

we arrive at a linear DAE

$$C\frac{d}{dt}\mathbf{x}(t) + G\mathbf{x}(t) \quad = \quad \tilde{\mathbf{e}}(t). \tag{2.17}$$

Because 2.17 is a linear system, the behaviour of a solution $\mathbf{x}(t)$ can be studied in the frequency domain, by considering the relation between the corresponding Fourier components of $\mathbf{x}(t)$ and of $\tilde{\mathbf{e}}(t)$. Let us write $\mathbf{x}(t) = Xe^{i\omega t}$, and $\tilde{\mathbf{e}}(t) = Ee^{i\omega t}$, where X and E are time independent vectors and ω is the common angular frequency. In this notation the equation becomes

$$(i\omega C + G)\,\mathbf{X} \quad = \quad \mathbf{E}. \tag{2.18}$$

If we assume that this equation can be solved, then $\|\mathbf{x}(t)\| = \|\mathbf{X}\|$
$\leq \| (i\omega C + G)^{-1} \| \|\mathbf{E}\| \ll 1$, if $\|\mathbf{E}\| \ll 1$. Note that also $\|\mathbf{x}'(t)\| = O(\|\mathbf{x}(t)\|)$ holds. But if we want to solve (2.18), it is not needed that $\|\mathbf{E}\|$ is small, because $\mathbf{X}$ is linearly scaled with $\mathbf{E}$.
Small-signal analysis is often called Alternating Current (AC) analysis. This is because the small-signal added to the vector $\mathbf{x}_0(t)$ is often a sinewave. Physically sinewaves can be interpreted as alternating currents.

2.8 Transient Analysis

The solution of (2.16) is time-dependent. For deriving its time-profile or waveform one has to integrate this equation. This is called Transient Analysis. A full-domain (also called transient) simulation of the circuit is necessary. In other words, this means solving the system

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}, t) + \mathbf{j}(\mathbf{x}, t) \quad = \quad \mathbf{0}$$

numerically on the interval $[0, T]$. Normally, the initial state $\mathbf{x}_0$ is known. Then it comes to the initial value problem (IVP):

$$\begin{cases} \frac{d}{dt}\mathbf{q}(t, \mathbf{x}) + \mathbf{j}(t, \mathbf{x}) = 0, \\ \mathbf{x}(0) = \mathbf{x}_0. \end{cases}$$

More details on numerically approximating the solution of this IVP will be discussed in Chapter 5.

2.9 Harmonic Balance

Harmonic Balance (HB) is a nonlinear frequency-domain analysis; in fact, it is a generalisation of the linear AC analysis. If the problem under consideration is linear, Harmonic Balance reduces to an analysis in which the linear equation for the higher harmonics are similar to the AC equations. These equations are only coupled to the DC equation. If this equation is linear as well, all equations for a Fourier component are decoupled from all other ones. Harmonic Balance is based on the so-called *Discrete Fourier Transform*, so we will explain this first.

2.9.1 The Discrete Fourier Transform

Let $g : \mathbb{R} \to \mathbb{R}$ be a Riemann-integrable, periodic function with period T. Then we have numbers $\gamma_k \in \mathbb{C}$, $k \in \mathbb{Z}$, satisfying:

$$g(t) = \sum_{k=-\infty}^{\infty} \gamma_k e^{i\omega k t} \text{ for } t \in D, \tag{2.19}$$

where $\omega := 2\pi/T$, $D \subseteq \mathbb{R}$ such that $\mathbb{R} \backslash D$ has measure 0. The right-hand side of equation 2.19 is called the *Fourier series* of g.

If g is continuous, we have $D = \mathbb{R}$. If g is sufficiently smooth, the γ_k's will vanish exponentially fast for $|k| \to \infty$. So in that case, it makes sense to approximate g with a truncated Fourier series:

$$g(t) \approx \hat{g}(t) := \sum_{k=-K}^{K} \gamma_k e^{i\omega k t}. \tag{2.20}$$

If we define $\hat{\boldsymbol{\gamma}} := [\gamma_{-K}, \gamma_{-K+1}, \ldots, \gamma_K]^T$, equation 2.20 can be rewritten as:

$$\hat{g}(t) = (\boldsymbol{\phi}(t) \cdot \hat{\boldsymbol{\gamma}}) = (\boldsymbol{\phi}(t))^T \hat{\boldsymbol{\gamma}}, \text{for all t} \tag{2.21}$$

where

$$\boldsymbol{\phi}(t) = [e^{i\omega(-K)t}, e^{i\omega(-K+1)t}, \ldots, e^{i\omega K t}]^T. \tag{2.22}$$

If we select $2K + 1$ points $t_{-K}, \ldots, t_K \in [0, T)$, we can define the vector $\hat{\mathbf{g}} := [g(t_{-K}), \ldots, g(t_K)]^T$. This vector is called a *time-profile* of g, since it consists of a finite number of samplings of g in the time-domain. We can also define the matrix Φ as:

$$\Phi := \begin{pmatrix} (\boldsymbol{\phi}(t_{-K}))^T \\ \vdots \\ (\boldsymbol{\phi}(t_K))^T \end{pmatrix}.$$

We can now derive from equation 2.21 the following compact relation between $\hat{\mathbf{g}}$,

Φ and $\hat{\boldsymbol{\gamma}}$.

$$\begin{aligned} & \hat{g}(t) = (\boldsymbol{\phi}(t))^T \hat{\boldsymbol{\gamma}} \\ \Longrightarrow \quad & \begin{pmatrix} \hat{g}(t_{-K}) \\ \vdots \\ \hat{g}(t_K) \end{pmatrix} = \begin{pmatrix} (\boldsymbol{\phi}(t_{-K}))^T \\ \vdots \\ (\boldsymbol{\phi}(t_K))^T \end{pmatrix} \hat{\boldsymbol{\gamma}} \\ \underset{\substack{\text{def. of} \\ \hat{\mathbf{g}} \text{ and } \Phi}}{\Longrightarrow} \quad & \hat{\mathbf{g}} = \Phi\hat{\boldsymbol{\gamma}} \text{ and } \hat{\boldsymbol{\gamma}} = \Phi^{-1}\hat{\mathbf{g}}. \end{aligned} \tag{2.23}$$

For most choices for the points t_k, the matrix Φ will be invertible. This is especially the case if the t_k's are chosen to be equally-spaced over the interval $[0, T)$. If Φ is invertible, we have of course $\hat{\boldsymbol{\gamma}} = \Phi^{-1}\hat{\mathbf{g}}$. Note that the matrix Φ^{-1} converts a finite (discrete) sampling of g in the time domain, into a truncated set of Fourier coefficients. Hence the matrix Φ^{-1} is called the *Discrete Fourier Transform* (DFT), and the matrix Φ is called the *Inverse Discrete Fourier Transform* (IDFT). For a well written on DFT and IDFT part, we refer to [31].

2.9.2 Applying the DFT in Harmonic Balance

If the functions $\mathbf{q}$ and $\mathbf{j}$ in equation 2.16 are replaced by their Fourier expansion,

$$\mathbf{q}(t, \mathbf{x}) =: \sum_{k=-\infty}^{\infty} Q_k e^{i\omega k t} \quad \text{and} \quad \mathbf{j}(t, \mathbf{x}) =: \sum_{k=-\infty}^{\infty} J_k e^{i\omega k t},$$

we get:

$$\sum_{k=-\infty}^{\infty} \frac{d}{dt} Q_k e^{i\omega k t} + \sum_{k=-\infty}^{\infty} J_k e^{i\omega k t} = \mathbf{0}. \tag{2.24}$$

Again, we can truncate the Fourier series,

$$\hat{\mathbf{q}}(t, \mathbf{x}) =: \sum_{k=-K}^{K} Q_k e^{i\omega k t} \quad \text{and} \quad \hat{\mathbf{j}}(t, \mathbf{x}) =: \sum_{k=-K}^{K} J_k e^{i\omega k t}.$$

resulting in:

$$\sum_{k=-K}^{K} \frac{d}{dt} Q_k e^{i\omega k t} + \sum_{k=-K}^{K} J_k e^{i\omega k t} = \mathbf{0}, \tag{2.25}$$

Equation 2.25 can be transformed into the following set of equations:

$$\begin{aligned} i\omega(-K)Q_{-K} + J_{-K} &= \mathbf{0}, \\ &\vdots \\ i\omega K Q_K + J_K &= \mathbf{0}. \end{aligned} \tag{2.26}$$

which is equivalent to :

$$\begin{pmatrix} i\omega(-K) & & \\ & \ddots & \\ & & i\omega(K) \end{pmatrix} \begin{pmatrix} Q_{-K} \\ \vdots \\ Q_K \end{pmatrix} + \begin{pmatrix} J_{-K} \\ \vdots \\ J_K \end{pmatrix} = 0 \tag{2.27}$$

If we define $\hat{Q} := [Q_{-K}, Q_{-K+1}, \ldots, Q_K]^T$, $\hat{J} := [J_{-K}, J_{-K+1}, \ldots, J_K]^T$ and $\Omega := \mathrm{diag}(i\omega(-K), \ldots, i\omega K)$. The system (2.27) can be write:

$$\Omega\hat{Q} + \hat{J} = 0 \tag{2.28}$$

Similar in equation 2.21.

$$\hat{\mathbf{q}}(t, \mathbf{x}) = (\boldsymbol{\phi}(t))^T \hat{Q} \tag{2.29}$$

$$\Longrightarrow \vec{\mathbf{q}}(\begin{pmatrix} \mathbf{x}_{-K} \\ \vdots \\ \mathbf{x}_K \end{pmatrix}) = \begin{pmatrix} \hat{q}(t_{-K}, \mathbf{x}_{-K}) \\ \vdots \\ \hat{q}(t_K, \mathbf{x}_K) \end{pmatrix} = \begin{pmatrix} (\boldsymbol{\phi}(t_{-K}))^T \\ \vdots \\ (\boldsymbol{\phi}(t_K))^T \end{pmatrix} \hat{Q}$$

$$\Longrightarrow \vec{\mathbf{q}} = \Phi\hat{Q}. \tag{2.30}$$

Also we have $\vec{\mathbf{j}} = \Phi\hat{J}$. Equation (2.28) can be rewritten as:

$$\Omega\Phi^{-1}\vec{\mathbf{q}}(\begin{pmatrix} \mathbf{x}(t_{-K}) \\ \vdots \\ \mathbf{x}(t_K) \end{pmatrix}) + \Phi^{-1}\vec{\mathbf{j}}(\begin{pmatrix} \mathbf{x}(t_{-K}) \\ \vdots \\ \mathbf{x}(t_K) \end{pmatrix}) = \mathbf{0}. \tag{2.31}$$

Now define $\hat{\boldsymbol{\xi}} := \Phi^{-1}[\mathbf{x}(t_{-K}), \ldots, \mathbf{x}(t_K)]^T$. Equation 2.31 can now be written as:

$$\mathbf{F}_{HB}(\hat{\boldsymbol{\xi}}) := \Omega\Phi^{-1}\vec{\mathbf{q}}(\Phi\hat{\boldsymbol{\xi}}) + \Phi^{-1}\vec{\mathbf{j}}(\Phi\hat{\boldsymbol{\xi}}) = \mathbf{0}. \tag{2.32}$$

Equation 2.32 is called the *Harmonic Balance form*. This is the equation that is solved in the Harmonic Balance algorithm.
The equations are in the frequency domain and the unknown $\hat{\boldsymbol{\xi}}$ consists of Fourier coefficients. It is clear that the system given by (2.32) is a non-linear algebraic set of equations in the frequency-domain.

2.9.3 Solving the Harmonic Balance form

Equation 2.32 is often solved using some type of Newton-Raphson iteration. Hence the Jacobi-matrix $d\mathbf{F}_{HB}/d\hat{\boldsymbol{\xi}}$ needs to be computed. Differentiation of $\mathbf{F}_{HB}$ yields:

$$\begin{aligned} \frac{d\mathbf{F}_{HB}}{d\hat{\boldsymbol{\xi}}} &= \frac{d}{d\hat{\boldsymbol{\xi}}}\left(\Omega\Phi^{-1}\vec{\mathbf{q}}(\Phi\hat{\boldsymbol{\xi}}) + \Phi^{-1}\vec{\mathbf{j}}(\Phi\hat{\boldsymbol{\xi}})\right) \\ &= \Omega\Phi^{-1}\frac{d\vec{\mathbf{q}}}{d\hat{\Phi\boldsymbol{\xi}}}(\Phi\hat{\boldsymbol{\xi}})\Phi + \Phi^{-1}\frac{d\vec{\mathbf{j}}}{d\Phi\hat{\boldsymbol{\xi}}}(\Phi\hat{\boldsymbol{\xi}})\Phi \\ &= \Omega\Phi^{-1}\vec{C}(\Phi\hat{\boldsymbol{\xi}})\Phi + \Phi^{-1}\vec{G}(\Phi\hat{\boldsymbol{\xi}})\Phi, \end{aligned} \tag{2.33}$$

where the block diagonal matrices $\vec{C}$ and $\vec{G}$ are defined by:

$$\vec{C}(\begin{pmatrix}\mathbf{x}_{-K}\\ \vdots \\ \mathbf{x}_{K}\end{pmatrix}) := \begin{pmatrix} C(t_{-K},\mathbf{x}_{-K}) & & \emptyset \\ & \ddots & \\ \emptyset & & C(t_K,\mathbf{x}_K)\end{pmatrix}$$

and

$$\vec{G}(\begin{pmatrix}\mathbf{x}_{-K}\\ \vdots \\ \mathbf{x}_{K}\end{pmatrix}) := \begin{pmatrix} G(t_{-K},\mathbf{x}_{-K}) & & \emptyset \\ & \ddots & \\ \emptyset & & G(t_K,\mathbf{x}_K)\end{pmatrix}.$$

The matrix formulations for $\Phi^{-1}C\Phi$ and $\Phi^{-1}G\Phi$ [5] surprisingly become :

$$\Phi^{-1}C\Phi = \begin{pmatrix} C_0 & C_{-1} & \dots & C_{-K} & C_K & \dots & C_1 \\ C_1 & C_0 & C_{-1} & \dots & C_{-K} & \dots & C_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ C_{K-1} & C_{K-2} & \cdots & C_{-1} & C_{-2} & \vdots & C_K \\ C_K & C_{K-1} & \dots & C_0 & C_{-1} & \dots & C_{-K} \\ C_{-K} & C_K & \dots & C_1 & C_0 & \dots & C_{-(K-1)} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ C_{-2} & \dots & C_K & \dots & \dots & C_0 & C_{-1} \\ C_{-1} & \dots & C_{-K} & C_K & \dots & C_1 & C_0 \end{pmatrix}$$

$$\Phi^{-1}G\Phi = \begin{pmatrix} G_0 & G_{-1} & \dots & G_{-K} & G_K & \dots & G_1 \\ G_1 & G_0 & G_{-1} & \dots & G_{-K} & \dots & G_2 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ G_{K-1} & G_{K-2} & \cdots & G_{-1} & G_{-2} & \vdots & G_K \\ G_K & G_{K-1} & \dots & G_0 & G_{-1} & \dots & G_{-K} \\ G_{-K} & G_K & \dots & G_1 & G_0 & \dots & G_{-(K-1)} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ G_{-2} & \dots & G_K & \dots & \dots & G_0 & G_{-1} \\ G_{-1} & \dots & G_{-K} & G_K & \dots & G_1 & G_0 \end{pmatrix}$$

so from (2.33), we have $\Longrightarrow$:

$$\frac{d\mathbf{F}_{HB}}{d\hat{\boldsymbol{\xi}}} = \begin{pmatrix} \Omega_1 & & \\ & \Omega_2 & \\ & \ddots & \\ & & \Omega_{2K+1} \end{pmatrix} \begin{pmatrix} C_0 & \dots & C_1 \\ \vdots & C_0 & \vdots \\ \dots & \dots & \dots \\ C_{-1} & \dots & C_0 \end{pmatrix} + \begin{pmatrix} G_0 & \dots & G_1 \\ \vdots & G_0 & \vdots \\ \dots & \dots & \dots \\ G_{-1} & \dots & G_0 \end{pmatrix} \tag{2.34}$$

Notice that the matrix $\Phi^{-1}C\Phi$, and $\Phi^{-1}G\Phi$ have the matrix property that each descending diagonal from left to right is constant. We call this kind of matrix a Toeplitz matrix, or a diagonal-constant matrix. And it has the following benifits:

1. A Toeplitz matrix is simple to implement and to invert.
2. It is simple to take products of the matrix with itself, or with another Toeplitz matrix, and for more detail see [6].
3. We do not need to store a whole matrix, we only need one line of the matrix, like C_{-K} ... C_K in our example. In this way efficient Krylov methods can be implemented.

The matrix at the left-hand side of (2.34) is not a Toeplitz matrix, but Krylov methods to solve the system can benefit from the Toeplitz properties of its main components.

Chapter 3

Periodic Steady State Solution

A Periodic Steady State (PSS) solution of a circuit is a solution $\mathbf{x} \in C([0,T], \mathbf{R}^N)$ to the following problem:

$$\frac{d}{dt}\mathbf{q}(t,\mathbf{x}) + \mathbf{j}(t,\mathbf{x}) = \mathbf{0} \qquad \text{for } 0 \le t \le T, \tag{3.1a}$$

$$\mathbf{x}(0) = \mathbf{x}(T). \tag{3.1b}$$

Here T may or may not be known a priori. If T is known a priori, the circuit is called non-autonomous, if $\mathbf{q}(t,\mathbf{x}) = \mathbf{q}(\mathbf{x})$ and $\mathbf{j}(t,\mathbf{x}) = \mathbf{j}(\mathbf{x})$ and T is not known a priori , it is called autonomous. If T is not known a priori, it forms an additional unknown in the above system. This is the case in circuits like a free-running oscillator, i.e. an oscillator that is not "driven" by external periodic input signals. For non-autonomous circuits, the resulting problem is a two-point boundary value problem.

3.1 Free-running Oscillators

A free-running oscillator is a circuit defined by the following properties:

1. The circuit equations do not depend explicitely on time t. i.e. the circuit equations are *autonomous*, hence they look like this:

$$\frac{d\mathbf{q}(\mathbf{x})}{dt} + \mathbf{j}(\mathbf{x}) = \mathbf{0}. \tag{3.2}$$

   This excludes any time-dependent sources, or other time-dependent circuit elements.

2. A periodic solution $\mathbf{x}(t)$ exists which is *not* a DC solution. In this chapter, we will refer to a solution $\mathbf{x}(t)$ which satisfies these conditions as a *non-trivial* periodic solution. The DC solution will be called the trivial periodic solution.

For real life applications we are interested in non-trivial periodic solutions $\mathbf{x}(t)$. The case is that T will, in general, not be known. To make the system easy to solve, we scale the interval $[0, T]$ to the interval $[0, 1]$ (or some other canonical interval you might like), by defining $\tau := t/T$. (3.2) can then be rewritten as:

$$\frac{1}{T}\frac{d\mathbf{q}(\mathbf{x}(\tau T))}{d\tau} + \mathbf{j}(\mathbf{x}(\tau T)) = \mathbf{0}. \tag{3.3}$$

By defining $\hat{\mathbf{x}}(\tau) := \mathbf{x}(\tau T)$, we solve the scaled Boundary Value Problem:

$$\frac{1}{T}\frac{d\mathbf{q}(\hat{\mathbf{x}})}{d\tau} + \mathbf{j}(\hat{\mathbf{x}}) = \mathbf{0}, \quad 0 \leq \tau \leq 1, \tag{3.4a}$$

$$\hat{\mathbf{x}}(0) = \hat{\mathbf{x}}(1), \tag{3.4b}$$

$$T > 0. \tag{3.4c}$$

Notice that the system 3.4 is not well-posed, since the solution is not unique. If $\hat{\mathbf{x}}(\tau)$ is a solution to 3.4, then $\hat{\mathbf{x}}(\tau + \tau_0)$ is also a solution for each $\tau_0 \in \mathbb{R}$. and there might be even more solutions.
And also if we try to solve equation 3.4 by using a Newton method, this will fail, since the Newton matrix will become singular when the solution is approached. Hence an extra condition is needed to make the problem well-posed; this is reasonable, since we added an extra unknown T to the problem, so we need an additional equation and this is called phase-shift condition.

3.2 Floquet-theory

3.2.1 Introduction

In this Section we study approaches for determining disturbances of the solution of the free oscillator when the autonomous differential equation (3.3) is perturbed by some noise term. Thus, we study

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}) + \mathbf{j}(\mathbf{x}) + \mathbf{n}(t) \quad = \quad \mathbf{0} \in \mathbb{R}^N \tag{3.5}$$

where $\mathbf{n}(t)$ represents the perturbation applied to (3.3).
A natural approach starts by an attempt with linearizing (3.5) by writing $\mathbf{x}(t) = \mathbf{x}_{\mathrm{PSS}}(t) + \mathbf{x}_{\mathrm{n}}(t)$. Assuming that the period T does not change, this gives a Linear Time Varying differential equation for $\mathbf{x}_{\mathrm{n}}$

$$\frac{d}{dt}(C(t)\mathbf{x}_{\mathrm{n}}(t)) + G(t)\mathbf{x}_{\mathrm{n}}(t) + \mathbf{n}(t) \quad = \quad \mathbf{0}. \tag{3.6}$$

where

$$C(t) \quad = \quad \frac{\partial \mathbf{q}(\mathbf{x})}{\partial \mathbf{x}(t)}\Big|_{\mathbf{x}_{\mathrm{PSS}}}, \tag{3.7}$$

$$G(t) \quad = \quad \frac{\partial \mathbf{j}(\mathbf{x})}{\partial \mathbf{x}(t)}\Big|_{\mathbf{x}_{\mathrm{PSS}}} \tag{3.8}$$

For $\mathbf{n}(t) = \mathbf{U}e^{i\nu t}$ a nice differential equation can be derived for the 'rotated' solution $\mathbf{y}_{\mathrm{n}} = e^{-i\nu t}\mathbf{x}_{\mathrm{n}}$

$$\frac{d}{dt}(C(t)\mathbf{y}_{\mathrm{n}}(t)) + [G(t) + i\nu C(t)]\mathbf{y}_{\mathrm{n}}(t) + \mathbf{U} \quad = \quad \mathbf{0}. \tag{3.9}$$

Here all coefficients are periodic in t with period T. This implies that the time shifted solution $\mathbf{y}_{\mathrm{n}}(t+T)$ also is a solution of the differential equation. However, it does not imply that $\mathbf{y}_{\mathrm{n}}(t)$ is periodic with period T. For example, the differential equation

$$y'(t) + \cos(t)y(t) - 1 \quad = \quad 0, \tag{3.10}$$

has coefficients that are periodic with period 2π. For its solution we find

$$y(t) \quad = \quad e^{-\sin(t)} \int_0^t e^{\sin(s)} \mathrm{d}s + e^{-\sin(t)} y(0), \tag{3.11}$$

$$y(2\pi) \quad = \quad \int_0^{2\pi} e^{\sin(s)} \mathrm{d}s + y(0) \quad > \quad y(0). \tag{3.12}$$

Indeed $y(t + 2\pi)$ is also a solution of (3.10) (so the solution satisfies the time-shifting property), but $y(t)$ clearly is not periodic with period 2π. The solution even becomes unbounded for $t \to 0$, so it can not be periodic or quasi-periodic.
The following graph shows the solution of (3.10) for the time interval[0,120]. For

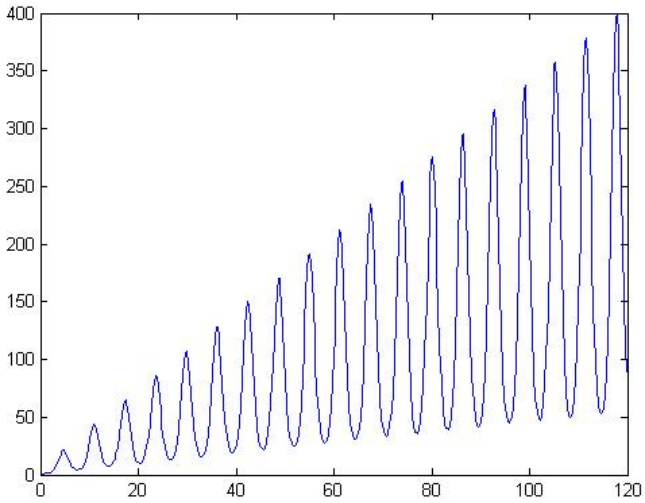


Figure 3.1: Solution of (3.10) for $0 \leqslant t \leqslant 120$

forced oscillators such a result can not occur and then $\mathbf{y}_{\mathrm{n}}(t)$ indeed is periodic with period T and can be expanded in a Fourier series (which is the basis for the noise

analysis in this case). But for perturbation of free oscillators, we clearly can not assume that the period T remains unaffected. We even have to face non-periodic functions as result. For this reason one considers solutions $\mathbf{x}(t)$ of (3.5) of the form $\mathbf{x}(t) = \mathbf{x}_{\mathrm{PSS}}(t + \alpha(t)) + \mathbf{x}_{\mathrm{n}}(t)$ in which $\alpha(t)$ is some additional non-trivial function one has to determine too.

3.2.2 Set up for Floquet theory

We will provide the necessary background of Floquet theory when applied to oscillator problems. In [20, 21] this theory is considered for linear homogeneous DAEs

$$A(t)\mathbf{x}'(t) + B(t)\mathbf{x}(t) \quad = \quad 0 \tag{3.13}$$

and extended to the case in which $A(t), B(t)$ are periodic with period T. In [3, 28] the autonomous case

$$\mathbf{x}' \quad = \quad A(t)\mathbf{x}(t) + B(t)\mathbf{b}(t) \tag{3.14}$$

is considered. The most adequate description that applies to circuit simulation is found in [4]. Here the DAE

$$\frac{\mathrm{d}}{\mathrm{d}t}(C(t)\mathbf{x}) + G(t)\mathbf{x} + B(t)\mathbf{b}(t) \quad = \quad 0 \tag{3.15}$$

is starting point for discussing how to treat phase noise. For phase noise purposes

$$B(t) \quad = \quad B(\mathbf{x}_{\mathrm{PSS}}(t)) \tag{3.16}$$

is evaluated at the Periodic Steady-State solution $\mathbf{x}_{\mathrm{PSS}}$ of the autonomous problem (3.3). Hence C, G, B are periodic with period T. The inhomogenous source term in (3.15), $B(t)\mathbf{b}(t)$, is considered to consist of a normalized perturbation function $\mathbf{b}(t)$, that is modulated by the periodical function $B(t)$. In practice, $\mathbf{b}(t)$ may be defined most conveniently in the frequency domain, in which case an Inverse Fourier Transform is needed to determine the time domain equivalent. Note that the product of $B(t)$ and of $\mathbf{b}(t)$ is defined in the time domain.
We note that $\mathbf{x}_{\mathrm{PSS}}(t)$ satisfies the homogeneous part of (3.15)

$$\frac{\mathrm{d}}{\mathrm{d}t}(C(t)\mathbf{x}) + G(t)\mathbf{x} \quad = \quad 0 \tag{3.17}$$

We assume the case of index 1 DAE's.
The Floquet Theorem for the homogeneous equation (3.14) is well written in [Ch 6.3 [32]]

Theorem 3.1. *We consider equation*

$$x^{'} = A(t)x \tag{3.18}$$

with $A(t)$ a continuous T-periodic $n \times n$ matrix. Each fundamental matrix $\Phi(t)$ (composed of n independent solutions) of equation 3.18 can be written as the product of two $n \times n$ matrices

$$\Phi(t) = P(t)e^{Bt} \tag{3.19}$$

with P(t) T-periodic and B a constant $n \times n$- matrix.

3.2.3 Independent Solutions

Let,

$$\begin{aligned} \mathbf{S}(t) &= \{\mathbf{z} \in \mathbb{R}^N \mid (G(t) + \frac{\mathrm{d}}{\mathrm{d}t}C(t))\mathbf{z} \in \mathrm{Im}(C(t))\}, & (3.20)\\ \mathbf{N}(t) &= \mathrm{Ker}(C(t)). & (3.21) \end{aligned}$$

Then one has

$$\begin{aligned} \mathbf{S}(t) \cap \mathbf{N}(t) &= 0, & (3.22)\\ \mathbf{S}(t) \oplus \mathbf{N}(t) &= \mathbb{R}^N. & (3.23) \end{aligned}$$

We assume that $\mathbf{S}(t)$ is m-dimensional. There are N independent solutions of the homogeneous problem: $\mathbf{u}_1(t)e^{\mu_1 t}, \ldots, \mathbf{u}_m(t)e^{\mu_m t}, \mathbf{u}_{m+1}(t), \ldots, \mathbf{u}_N(t)$. The first $\mathbf{u}_1(t), \ldots, \mathbf{u}_m(t)$ are a basis of $\mathbf{S}(t)$; the last, $\mathbf{u}_{m+1}(t), \ldots, \mathbf{u}_N(t)$, are a basis of $\mathbf{N}(t)$. The $\mu_1, \ldots, \mu_m$ are so-called Floquet exponents; the $e^{\mu_1 T}, \ldots, e^{\mu_m T}$ are Floquet multipliers. For a stable autonomous index 1 problem we can assume that $\mu_1 = 0$ and that $\mathrm{Re}(\mu_i) < 0$ for $i = 2, \ldots, m$. In this case we can choose $\mathbf{u}_1(t) = \mathbf{x}'_{\mathrm{PSS}}(t)$.

3.2.4 Adjoint Problem

The homogeneous adjoint (or dual) system corresponding to (3.15) is

$$C^T(t)\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{y} - G^T(t)\mathbf{y} = 0 \tag{3.24}$$

Similar to the non-adjoint case we introduce

$$\begin{aligned} \mathbf{S}^T(t) &= \{\mathbf{z} \in \mathbf{R}^n \,;\mid G^T(t)\mathbf{z} \in \mathrm{Im}(C^T(t))\}, & (3.25)\\ \mathbf{N}^T(t) &= \mathrm{Ker}(C^T(t)), & (3.26) \end{aligned}$$

adjoint system have the properties Also $\mathbf{S}^T$ is m-dimensional. The adjoint problem has N independent solutions: $\mathbf{v}_1(t)e^{-\mu_1 t}, \ldots, \mathbf{v}_m(t)e^{-\mu_m t}, \mathbf{v}_{m+1}(t), \ldots, \mathbf{v}_N(t)$, where $\mathbf{v}_1(t), \ldots, \mathbf{v}_m(t)$ are a basis of $\mathbf{S}^T(t)$ and the last, $\mathbf{v}_{m+1}(t), \ldots, \mathbf{v}_N(t)$, are a basis of $\mathbf{N}^T(t)$.

3.2.5 Bi-Orthgonality

It is easy to verify that if $\mathbf{x}$ and $\mathbf{y}$ are solutions of (3.17) and (3.24), respectively, then $\mathbf{y}^T(t)C(t)\mathbf{x}(t) = \mathbf{y}^T(0)C(0)\mathbf{x}(0)$, for all $t \geq 0$ (and thus is constant).

Proof. We prove that: $\mathbf{z}(t) = \mathbf{y}^T(t)C(t)\mathbf{x}(t)$ is constant, by proving that $\frac{d}{dt}\mathbf{z}(t) = 0$ for all t.

$$\begin{aligned}
\frac{d}{dt}\mathbf{z}(t) &= \frac{d}{dt}\left[\mathbf{y}^T(t)C(t)\mathbf{x}(t)\right] \\
&= \left(\frac{d}{dt}\mathbf{y}^T(t)\right)C(t)\mathbf{x}(t) + \mathbf{y}^T(t)\cdot\underbrace{\frac{d}{dt}\left[C(t)\mathbf{x}(t)\right]}_{=-G(t)\mathbf{x}(t)} \\
&= \left(\frac{d}{dt}\mathbf{y}^T(t)\right)C(t)\mathbf{x}(t) - \mathbf{y}^T(t)\cdot G(t)\mathbf{x}(t) \\
&= \left[\left(\frac{d}{dt}\mathbf{y}^T(t)\right)C(t) - \mathbf{y}^T(t)G(t)\right]\mathbf{x}(t) \\
&= \left[\underbrace{C(t)^T\frac{d}{dt}\mathbf{y}(t) - G(t)^T\mathbf{y}(t)}_{=0}\right]^T\mathbf{x}(t) \\
&= 0
\end{aligned}$$

□

More specifically, the bases $\mathbf{u}_1(t), \ldots, \mathbf{u}_N(t)$ and $\mathbf{v}_1(t), \ldots, \mathbf{v}_N(t)$ can be chosen such that, the $N \times N$ matrix $U(t)$ with as columns the $\mathbf{u}_i(t)$ and the $N \times N$ matrix $V(t)$ with as rows the $\mathbf{v}_i(t)$ satisfy a bi-orthogonality relation w.r.t. $C(t)$ and a nearly one w.r.t. $G(t)$

$$V(t)C(t)U(t) = \begin{pmatrix} I_m & 0 \\ 0 & 0 \end{pmatrix}, \tag{3.27}$$

$$V(t)G(t)U(t) = \begin{pmatrix} J_m^1 & 0 \\ 0 & J_m^2 \end{pmatrix}. \tag{3.28}$$

Here I_m is a $m \times m$ identity matrix. J_m^2 are suitable block matrices. J_m^1 is a $m \times m$ block matrix.

3.2.6 State-Transition Matrix, Monodromy Matrix

Assuming a consistent initial condition $\mathbf{x}(0) = \mathbf{x}_0 \in \mathbf{S}(0)$, the solution $\mathbf{x}_H(t)$ of (3.17) can be written as [12] :

$$\begin{aligned}
\mathbf{x}_H(t) &= \sum_{i=1}^{m} \mathbf{u}_i(t) \exp(\mu_i t) \mathbf{v}_i^T(0) C(0) \mathbf{x}_0, && (3.29)\\
&= \Phi(t,0)\mathbf{x}_0, && (3.30)\\
\Phi(t,s) &= \Theta(t,s)C(s), && (3.31)\\
\Theta(t,s) &= U(t)D(t-s)V(s), && (3.32)\\
D(t-s) &= \mathrm{Diag}(\exp(\mu_1(t-s)), \ldots, \exp(\mu_m(t-s)), 0, \ldots, 0) && (3.33)
\end{aligned}$$

If $\mathbf{x}_0$ is not a consistent initial value, one can write $\mathbf{x}_0 = \mathbf{x}_0^{(S)} + \mathbf{x}_0^{(N)}$, where $\mathbf{x}_0^{(S)} \in \mathbf{S}(0)$ and $\mathbf{x}_0^{(N)} \in \mathbf{N}(0)$. Clearly $C(0)\mathbf{x}_0 = C(0)\mathbf{x}_0^{(S)}$, and $\mathbf{x}_H(t)$ depends on $\mathbf{x}_0^{(S)}$, rather then on $\mathbf{x}_0$.
An inhomogeneous solution of (3.15) can be written as

$$\begin{aligned}
\mathbf{x}_P(t) &= \mathbf{x}_H(t) + \sum_{i=1}^{m} \mathbf{u}_i(t) \int_0^t \exp(\mu_i(t-s)) \mathbf{v}_i^T(s) B(s) \mathbf{b}(s) \mathrm{d}s + \Gamma(t) B(t) \mathbf{b}(t),\\
&= \mathbf{x}_H(t) + \int_0^t \Theta(t,s) B(s) \mathbf{b}(s) \mathrm{d}s + \Gamma(t) B(t) \mathbf{b}(t)
\end{aligned}$$

Here $\Gamma(t)$ is a matrix with $\mathrm{Ker}(\Gamma(t)) = \mathrm{Span}(C(t)\mathbf{u}_1(t), \ldots, C(t)\mathbf{u}_m(t))$.

Monodromy Matrix

The monodromy matrix is the matrix $\Phi(t,0)$ after one period, i.e. $\Phi(T,0)$ (this matrix one naturally studies when one considers shooting methods). Because of the periodicity of the $\mathbf{u}_i$, we see that the $\mathbf{u}_i(0)$, for $i = 1, \ldots, m$ are eigenvectors of the monodromy matrix with corresponding eigenvalues $\exp(\mu_i T)$, and that the remaining $\mathbf{u}_i(0)$, for $i = m+1, \ldots, N$, are eigenvectors for the $(N-(m-1))$-fold eigenvalue 0.

State-Transition Matrix Adjoint Problem

The adjoint problem (3.24) has state-transition matrix

$$\begin{aligned}
\Psi(t,s) &= V^T(t) D(s-t) U^T(s) C^T(s), && (3.34)\\
&= \sum_{i=1}^{m} \exp(-\mu_i(t-s)) \mathbf{v}_i(t) \mathbf{u}_i^T(s) C^T(s) && (3.35)
\end{aligned}$$

Similar to the non-adjoint case, the $\mathbf{v}_i(0)$ are eigenvectors of the associated monodromy matrix $\Psi(T,0)$.

3.3 Stability Analysis

From the course of differential equations, we know that the solutions of the equation $x^{'} = Ax$ are stable when the spectrum of A is contained in the left half plane of $\mathbb{C}$. Therefore, it is a surprise that for linear, nonautomous equations,

$$x^{'} = A(t)x, \tag{3.36}$$

the eigenvalues of the matrix A(t) are in general of no use in determining the stability of solutions.
The most striking example are provided by matrices A(t) with constant negative eigenvalues when the system 3.36 has exponentially growing solutions. The following example introducced by Vinograd is typical [33] :

$$A(t) = \begin{pmatrix} -1 - 9cos^2(6t) + 12sin(6t)cos(6t) & 12cos^2(6t) + 9sin(6t)cos(6t) \\ -12sin^2(6t) + 9sin(t)cos(6t) & -1 - 9sin^2(6t) - 12sin(6t)cos(6t) \end{pmatrix} \tag{3.37}$$

The eigenvalues of $A(t)$ for any t are negetive but the unstable solution is illustrated as follows (starting from $(-2.44, 1.23)$):

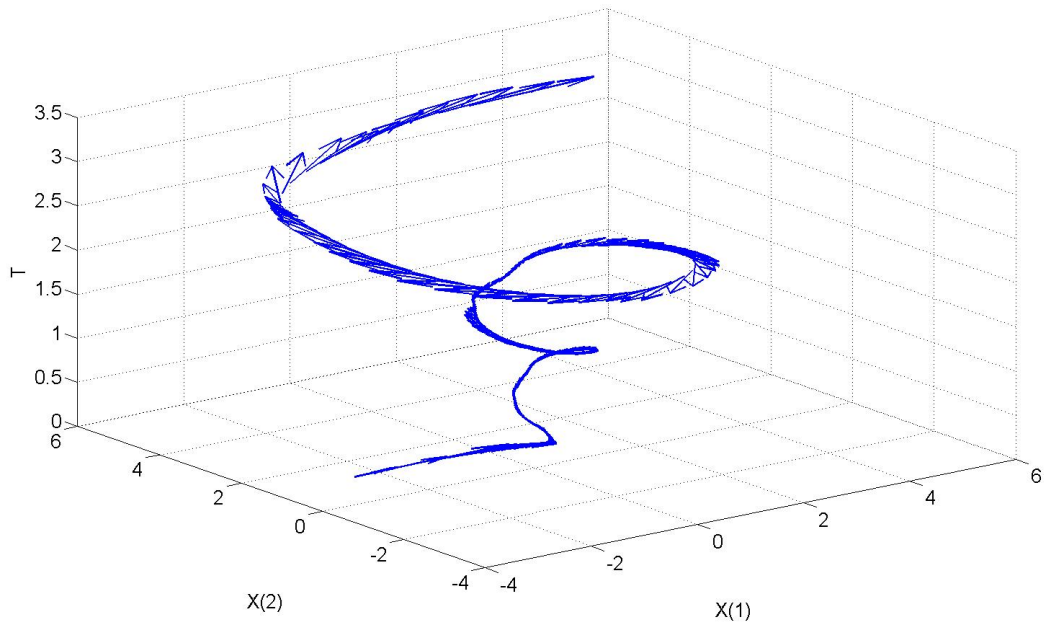


Figure 3.2: *An unstable solution to Vinograd's example with initial conditions* $(y_1, y_2) = (-2.44, 1.23)$

Similar systems can be found in the literature: The example by Markus and Yamabe of an unstable system of the form 3.36 in which $A(t)$ has complex eigenvalues with negative real parts is frequently cited [7, 22]. Other examples are found in Hinrichsen [9], Wu, [33]. When these examples were first encountered, it was considered as something miraculous. In most cases it is not explained how the form of the matrix $A(t)$ was defined, or how the unstable solution was constructed. Now we give a explanation.

3.3.1 Instability of the Frozen Coefficient Equations

Our goal is to explain how it is possible that the matrix $A(t)$ in 3.36 has negative eigenvalues and the solutions are unstable. If it happens, the norm of the solution, $\|x(t)\| = \sqrt{x(t) \cdot x(t)}$, must increase over time [18] so that

$$\frac{d}{dt}\|x(t)\|^2 = 2x^{'}(t) \cdot x(t) = 2[A(t)x(t)] \cdot x(t) > 0 \tag{3.38}$$

for at least some values of t. Fix $t_0 > 0$ and consider the autonomous frozen coefficient system:

$$x^{'} = A(t_0)x, \tag{3.39}$$

obtained from (3.36) by "freezing" the matrix $A(t)$ at time t_0. Condition (3.38) implies that there must be a t_0 such that the frozen coefficient system (3.39) has solutions whose distance from the origin increases during some interval of time.
The first goal is to characterize the class of matrices:

$\mathcal{B}$= {B is a 2×2 matrix whose eigenvalues have negative real part
and $x \cdot Bx > 0$ for some $x \in \mathbb{R}^2$}

By (3.38) , solutions to (3.36) can only be unstable if $A(t) \in \mathcal{B}$ for some t. For instance, in the case of Vinograd's example given in 3.37, we see that for $x^* = [1\ 1]^T$ we have $x^* \cdot A(0)x^* > 0$.
So the conclusion is that there must exist an x and a t such that $x \cdot A(t)x > 0$.
For 3.36 to have unstable solutions can be reached using Lyapunov functions: if $x \cdot x^{'} = x \cdot A(t)x \leq 0$ for any t and x, then let $V(x) = x \cdot x$, we can follow the computation,

$$V^{'}(x) = 2(x^{'} \cdot x) = 2(A(t)x \cdot x) \leq 0, \tag{3.40}$$

and no solutions can cross the level curves of $V(x)$.

3.3.2 Stability in Linear Nonautonomous Equations

We recall that the exponential of a matrix A is defined as:

$$e^A = \sum_{n=1}^{\infty} \frac{A^n}{n!} \tag{3.41}$$

For more discussion of the basic properties and applications of matrix exponentials, see [10].
We now return to find the specific nonautonomous equations with unstable solutions. We start with a matrix B taken from the class $\mathcal{B}$ described in the previous, and rotate the corresponding vector field x at a constant angular velocity, which means: let $G(\omega) = \begin{pmatrix} 0 & -\omega \\ \omega & 0 \end{pmatrix}$. Define that:

$$\begin{aligned} R(t,\omega) = e^{tG(\omega)} &= e^{\begin{pmatrix} 0 & -\omega t \\ \omega t & 0 \end{pmatrix}} \\ &= \sum_{n=1}^{\infty} \frac{\begin{pmatrix} 0 & -\omega t \\ \omega t & 0 \end{pmatrix}^n}{n!} \\ &= \begin{pmatrix} 0 & -\omega t \\ \omega t & 0 \end{pmatrix} + \frac{1}{2!}\begin{pmatrix} -(\omega t)^2 & 0 \\ 0 & -(\omega t)^2 \end{pmatrix} + \frac{1}{3!}\begin{pmatrix} 0 & (\omega t)^3 \\ -(\omega t)^3 & 0 \end{pmatrix} + ... \\ &= \begin{pmatrix} cos(\omega t) & -sin(\omega t) \\ sin(\omega t) & cos(\omega t) \end{pmatrix} \end{aligned}$$

which rotates the plane by an angle ωt, or equivalently, at angular velocity ω. Let $A(t) = R(t,\omega)B[R(t,\omega)]^{-1}$ so that the vector field $A(t)x$ is obtained from Bx by a rotation over an angle ωt. The equations we consider henceforth will be of the form

$$x^{'} = A(t)x = (R(t,\omega)B[R(t,\omega)]^{-1})x. \tag{3.42}$$

and the solution to 3.42 is ,

$$x(t) = R(t,\omega)e^{[B-G(\omega)]t}x(0). \tag{3.43}$$

The approach to solving this equation can be found in [18]. We can also easily verify the solution :

$$\begin{aligned} x' &= R' \cdot e^{[B-G]t} \cdot x(0) + R \cdot e^{[B-G]t} \cdot [B-G] \cdot x(0) \\ &= \underbrace{e^{tG}}_{R} \cdot G \cdot e^{[B-G]t} \cdot x(0) + R \cdot e^{[B-G]t} \cdot [B-G] \cdot x(0) \\ &\qquad \text{notice that } e^{Pt} \cdot P = P \cdot e^{Pt} \text{ for all P} \\ &= R \cdot G \cdot e^{[B-G]t} \cdot x(0) + R \cdot [B-G] \cdot e^{[B-G]t} \cdot x(0) \\ &= R \cdot B \cdot e^{[B-G]t} \cdot x(0) \\ &= R \cdot B \cdot R^{-1} \cdot x(t) \text{rm if x(t) is the solution of that form} \\ &= A(t)x \end{aligned}$$

Note that we did not require B and G are commute. In general

$$X \cdot Y \neq Y \cdot X \Rightarrow e^{X+Y} \neq e^X \cdot e^Y.$$

Simple example is when $X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $Y = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, we get $e^{X+Y} = \begin{pmatrix} 7.3891 & 7.3891 \\ 20.0855 & 148.4132 \end{pmatrix}$, but $e^X \cdot e^Y = \begin{pmatrix} 27.4746 & 74.6837 \\ 57.3164 & 155.8022 \end{pmatrix}$.
Now existence of an unstable solution for differential equation 3.36 with the family of matrices $A(t)$ having constant eigenvalues with negative real part has come down to another problem: Given a matrix B, can we find a matrix $G(\omega)$ such that $B - G(\omega)$ has positive eigenvalues. If so, the $x(t)$ is an unstable solution.and we must take care in this case.

3.3.3 Stability for Periodic Coefficient Equations

From the above subsection we know the eigenvalues of $A(t)$ do not determine the stability of the solution to (3.42), so people may ask what additionally is needed to guarantee stable solutions. When $A(t)$ is periodic, the answer is Floquet exponents. The theory has been stated in the section before, and we repeat the solution when $A(t)$ is periodic.

$$x(t) = P(t)e^{Mt}, \tag{3.44}$$

where $P(t)$ has period T and M is a constant matrix $\neq 0$.
The eigenvalues of e^{MT} are called characteristic multipliers of $A(t)$, and a Floquet exponent of $A(t)$ is a complex number μ such that $e^{\mu T}$ is a characteristic multiplier of $A(t)$. In particular, the eigenvalues of M are Floquet exponents of $A(t)$. The system $x'(t) = A(t)x(t)$ is asymptotically stable if the real parts of the Floquet exponents are negative.
From the systems of the form 3.42, $A(t)$ has period $T = \frac{2\pi}{\omega}$ and the discussion in section 2 implies that $P(t) = e^{G(\omega)t}$ and $M = B - G(\omega)$. Thus the eigenvalues of $B - G(\omega)$ are the Floquet exponents of the system 3.42 and the signs of their real parts determine the stability of the system. This is the same conclusion we reached in subsection 3.3.2.

Chapter 4

Methods for Periodic Steady State

4.1 Finite Difference Method

The Finite Difference Method tries to solve the DAE and the boundary equations all "at once", i.e. by accumulating the linearised equations in one big matrix and solving the resulting matrix equation. The disadvantage of this method is that the size of this matrix can be huge; however, the matrix has a sparse structure, which can be exploited to keep memory usage reasonable.

First, the system of (3.1) has to be discretised. For this, choose $M+1$ points $0 = t_0 < t_1 < \cdots < t_M = T$. We will choose the θ-method for the discretisation:

$$\mathbf{F^N} = \frac{\mathbf{q}(t_j, \mathbf{x}_j) - \mathbf{q}(t_{j-1}, \mathbf{x}_{j-1})}{t_j - t_{j-1}} + \theta \mathbf{j}(t_j, \mathbf{x}_j) + (1-\theta)\mathbf{j}(t_{j-1}, \mathbf{x}_{j-1}) = \mathbf{0} \in \mathbb{R}^N. \tag{4.1}$$

combined with the periodicity conditions:

$$\mathbf{x}_0 - \mathbf{x}_M = \mathbf{0}. \tag{4.2}$$

and applying Newton-Raphson:

$$\frac{\partial \mathbf{F^N}}{\partial \mathbf{x}}(\mathbf{x}^{k+1} - \mathbf{x}^k) = -\mathbf{F^N}(\mathbf{x^k}). \tag{4.3}$$

Where $\frac{\partial \mathbf{F^N}}{\partial \mathbf{x}}$ is defined by:

$$\begin{pmatrix} I & & & & -I \\ B_1 & A_1 & & & \\ & B_2 & A_2 & & \\ & & \ddots & \ddots & \\ & & & B_M & A_M \end{pmatrix} \tag{4.4}$$

Define $A_j := \frac{1}{\Delta t_j} C_j + \theta G_j$ and $B_j := \frac{-1}{\Delta t_{j-1}} C_{j-1} + (1-\theta) G_{j-1}$. Here T is known so we fomulate the system for non-autonomous problems. For non-autonomous problems (4.3) is enough, because T is known.
But note that for autonomous problems T (or the frequency f) has become an additional unknown, and we also need to make the solution unique. In that case, we have to add one more condition, which is called the phase-shift condition.

$$\begin{aligned} \mathbf{F}^D(\mathbf{x}, f) &= \mathbf{0}, & (4.5)\\ \mathbf{p}^T\mathbf{x} - c &= 0, & (4.6) \end{aligned}$$

where $\mathbf{F^D} : \mathbf{R^{MN}} \rightarrow \mathbf{R^{MN}}$ is given by

$$\begin{aligned} \mathbf{F^D}_0(\mathbf{x}, f) &= f\,\frac{\mathbf{q}(\mathbf{x}(t_0)) - \mathbf{q}(\mathbf{x}(t_{M-1}))}{\Delta t_0} + [\theta\mathbf{j}(\mathbf{x}(t_0)) + (1-\theta)\mathbf{j}(\mathbf{x}(t_{M-1}))],\\ \mathbf{F^D}_i(\mathbf{x}, f) &= f\,\frac{\mathbf{q}(\mathbf{x}(t_i)) - \mathbf{q}(\mathbf{x}(t_{i-1}))}{\Delta t_i} + [\theta\mathbf{j}(\mathbf{x}(t_i)) + (1-\theta)\mathbf{j}(\mathbf{x}(t_{i-1}))],\\ & \quad 1 \le i \le M-1. \end{aligned}$$

In (4.6), $\mathbf{p} \in \mathbf{R^{MN}}$ is some given vector. and c is a constant, which should be determined in the range of $\mathbf{x}$.
and applying Newton-Raphson yields:

$$\begin{aligned} \mathcal{Y}\begin{pmatrix} \mathbf{x}^{k+1} - \mathbf{x}^k \\ f^{k+1} - f^k \end{pmatrix} &= -\begin{pmatrix} \mathbf{F^D}(\mathbf{x}^k, f^k) \\ \mathbf{p}^T\mathbf{x}^k - c \end{pmatrix} & (4.7)\\ \mathcal{Y} &= \begin{pmatrix} Y & \mathbf{Z} \\ \mathbf{p}^T & 0 \end{pmatrix}, & (4.8) \end{aligned}$$

in which

$$\begin{aligned} \mathbf{Z} &= \frac{\partial}{\partial f}\mathbf{F^D}(\mathbf{x}^k, f^k), & (4.9)\\ Y &= \frac{\partial}{\partial \mathbf{x}}\mathbf{F^D}(\mathbf{x}^k, f^k), & (4.10) \end{aligned}$$

There are some options [11] for the phase-shift condition $\mathbf{p}$, and here we choose the form

$$\mathbf{p}^T = \mathbf{e}_j^T,$$

in which $\mathbf{p}$ is a unit vector. Also let (4.6) only affects the first time level, which means

$$\mathbf{p}^T\mathbf{x}(0) = c.$$

4.2 Newton-Raphson method for DC

The Newton-Raphson method ([30]) is used to find zeros of a given function $\mathbf{f}$. In circuit analysis we usually have a vector-valued function $\mathbf{f} : \mathbf{R}^n \to \mathbf{R}^n$, with $n > 1$. Thus

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(x^1, \ldots, x^n) \\ \vdots \\ f_n(x^1, \ldots, x^n) \end{bmatrix}.$$

Assume $\mathbf{x} = \xi$ is a zero of $\mathbf{f}$, and assume that $\mathbf{x}_0$ is an approximation to ξ and that $\mathbf{f}$ is differentiable for $\mathbf{x} = \mathbf{x}_0$ then $\mathbf{f}(\xi)$ can be approximated by

$$\mathbf{0} \;=\; \mathbf{f}(\xi) \;\;\approx\;\; \mathbf{f}(\mathbf{x}_0) + D\mathbf{f}(\mathbf{x}_0)(\xi - \mathbf{x}_0),$$

where

$$D\mathbf{f}(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x^1} & \cdots & \frac{\partial f_1}{\partial x^n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x^1} & \cdots & \frac{\partial f_n}{\partial x^n} \end{bmatrix}_{\mathbf{x}=\mathbf{x}_0}, \; \xi - \mathbf{x}_0 = \begin{bmatrix} \xi^1 - x_0^1 \\ \vdots \\ \xi^n - x_0^n \end{bmatrix}.$$

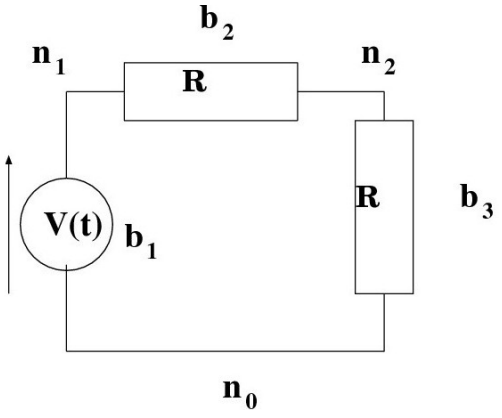
If the Jacobian $D\mathbf{f}(\mathbf{x}_0)$ is not singular, we can define

$$\mathbf{x}_1 \;\;=\;\; \mathbf{x}_0 - (D\mathbf{f}(\mathbf{x}_0))^{-1}\,\mathbf{f}(\mathbf{x}_0),$$

and $\mathbf{x}_1$ may be taken as a closer approximation to the zero ξ. The generalized Newton-Raphson method for solving systems of equations is given by

$$\mathbf{x}_{i+1} \;\;=\;\; \mathbf{x}_i - (D\mathbf{f}(\mathbf{x}_i))^{-1}\,\mathbf{f}(\mathbf{x}_i). \tag{4.11}$$

We apply this Newton-Raphson method to find the DC equilibrium of the following nonlinear network.



This network consists of a voltage source and two resistors. Branch b_2 is a linear resistor, with resistance $R = 1$, and branch b_3 a nonlinear resistor. We have the following relation:

$$\begin{aligned} i_{b_3} &= (v_{n_0} - v_{n_2})^3 + 2(v_{n_0} - v_{n_2})^2 + (v_{n_0} - v_{n_2}) \\ &= -v_{n_2}^3 + 2v_{n_2}^2 - v_{n_2}. \end{aligned}$$

The second equality holds because we choose n_0 as ground node, and thus $v_{n_0} = 0$. The equations can again be written in the general form

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}) + \tilde{\mathbf{j}}(\mathbf{x}) - \mathbf{e}(t) = 0.$$

Here

$$\mathbf{x} = \begin{bmatrix} v_{n_1} \\ v_{n_2} \\ i_{b_1} \end{bmatrix}, \tilde{\mathbf{j}}(\mathbf{x}) = \begin{bmatrix} -i_{b_1} + \frac{v_{n_2} - v_{n_1}}{1} \\ v_{n_1} - 2v_{n_2} + 2v_{n_2}^2 - v_{n_2}^3 \\ v_{n_1} \end{bmatrix}, \mathbf{e}(t) = \begin{bmatrix} 0 \\ 0 \\ V(t) \end{bmatrix},$$

and $\mathbf{q}(\mathbf{x}) = \mathbf{0}$.

To apply DC analysis we take $V(t) = 20$. In short, we are looking for a zero of the function $\mathbf{f}(\mathbf{x})$, with $\mathbf{x} = (x^1, x^2, x^3)^T$ and

$$\mathbf{f}(\mathbf{x}) = \tilde{\mathbf{j}}(\mathbf{x}) - \mathbf{e}(t) = \begin{bmatrix} -x^3 - x^1 + x^2 \\ x^1 - 2x^2 + 2\left(x^2\right)^2 - \left(x^2\right)^3 \\ x^1 - 20 \end{bmatrix}.$$

There are 2 tolerances given, $ATOL = 10^{-7}$ and $FTOL = 10^{-6}$. We stop iterating if

$$\begin{aligned} \|\mathbf{x}_{i+1} - \mathbf{x}_i\| &\leq ATOL, \text{stopping on correction} \\ \text{or } \|\mathbf{f}(\mathbf{x})\| &\leq FTOL, \text{stopping on residue} \end{aligned}$$

where $\|\cdot\|$ is the Euclidean norm. We start with the value $\mathbf{x}_0 = (20, 16, 4)^T$. In Table 4.1 we list the results. In Figure 4.1 we see the log of the approximation error ($\|\mathbf{x}_i - \xi\|$) and the log of the residue ($\|\mathbf{f}(\mathbf{x_i})\|$) plotted as function of the iteration number i. We can see from this plot that this example is quadratically convergent, which is a well-known theoretical result, means

$$\|\mathbf{x}_{i+1} - \xi\| \leq C\|\mathbf{x}_i - \xi\|^2.$$

And the last few iterations converge even stronger.

number of iteration i	approximation $\mathbf{x}_i$	approximation error $\|\mathbf{x}_i - \xi\|$	residue $\|\mathbf{f}(\mathbf{x_i})\|$	correction error $\| \mathbf{x}_{i+1} - \mathbf{x}_i\|$
0	$(20.0000, 16.0000, 4.0000)^T$			
1	$(20.0000, 10.9065, -9.0935)^T$	10.8078	3596	14.0493
2	$(20.0000, 7.5399, -12.4601)^T$	6.0466	1061.3	4.7611
3	$(20.0000, 5.3626, -14.6374)^T$	2.9675	310	3.0791
4	$(20.0000, 4.0543, -15.9457)^T$	1.1172	87.4	1.8503
5	$(20.0000, 3.4310, -16.5690)^T$	0.2357	21.9	0.8815
6	$(20.0000, 3.2738, -16.7262)^T$	0.0135	3.7	0.2222
7	$(20.0000, 3.2643, -16.7357)^T$	4.8053e-005	0.2	0.0135
8	$(20.0000, 3.2643, -16.7357)^T$	6.0853e-010	0	0
9	$(20.0000, 3.2643, -16.7357)^T$	0	0	0

Table 4.1: Results of Newton-Raphson iteration.

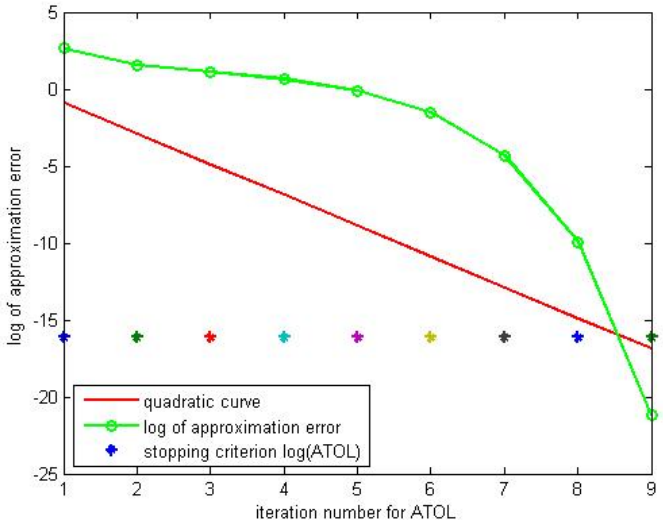


Figure 4.1: Approximation error of Newton-Raphson method.

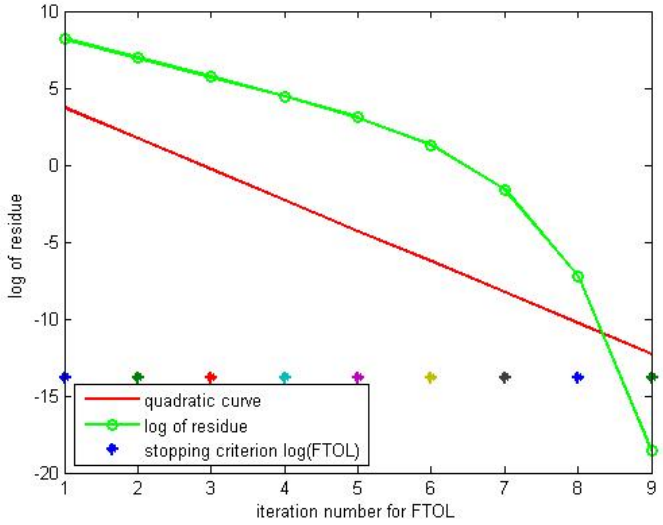


Figure 4.2: Residue of Newton-Raphson method.

4.3 Shooting method

In this section we are trying to solve an equation of the form:

$$G(\mathbf{x}_0) = F(\mathbf{x}_0) - \mathbf{x}_0 = 0. \tag{4.12}$$

by some type of Newton method. This is called *Single Shooting*. TThe matrix $\Phi(\mathbf{x}_0) := dF(\mathbf{x}_0)/d\mathbf{x}_0$ is needed. This matrix can be computed as a by-product of the transient computation of $\mathbf{x}(T)$ from $\mathbf{x}_0$.

The shooting method can now be described by the following pseudo-code:

$\mathbf{x}_0 :=$ some initial guess
{integrate (3.1a) over $[0, T]$, T is known:}
$\mathbf{x}_T, \mathbf{\Phi} := F(\mathbf{x}_0), \Phi(\mathbf{x}_0)$

while $\|\mathbf{x}_0 - \mathbf{x}_T\| \geq \epsilon$ **do**
    {compute Newton update:}
    $\mathbf{x}_0 := (I - \Phi)^{-1}(\mathbf{x}_T - \Phi\mathbf{x}_0)$
    {integrate (3.1a) over $[0, T]$:}
    $\mathbf{x}_T, \mathbf{\Phi} := F(\mathbf{x}_0), \Phi(\mathbf{x}_0)$

Here, ϵ is the convergence criterium of the Newton algorithm. If equation 3.1a is linear, the while-loop will be executed only once. So the shooting method can be

simplified for linear problems to yield:

$$\begin{aligned}
&\mathbf{x}_0 := \text{ some initial guess} \\
&\{\text{integrate (3.1a) over } [0,T]:\} \\
&\mathbf{x}_T, \mathbf{\Phi} := F(\mathbf{x}_0), \Phi(\mathbf{x}_0) \\
&\{\text{compute Newton update:}\} \\
&\mathbf{x}_0 := (I-\Phi)^{-1}(\mathbf{x}_T - \Phi\mathbf{x}_0)
\end{aligned}$$

4.4 Newton Method for PSS

Newton Method is a variant on the shooting method. Suppose that some initial profile $\mathbf{x}^{(0)}(t)$ is provided. We are now going to find a better guess $\mathbf{x}^{(1)}$, which is defined as:

$$\mathbf{x}^{(1)}(t) := \mathbf{x}^{(0)}(t) + \delta\mathbf{x}^{(1)}(t). \tag{4.13}$$

By substituting this guess into (3.1a), we obtain:

$$\begin{aligned}
&\frac{d}{dt}\mathbf{q}(t,\mathbf{x}^{(1)}) + \mathbf{j}(t,\mathbf{x}^{(1)}) = \mathbf{0} \\
\underset{(4.13)}{\Longrightarrow} &\frac{d}{dt}\mathbf{q}(t,\mathbf{x}^{(0)} + \delta\mathbf{x}^{(1)}) + \mathbf{j}(t,\mathbf{x}^{(0)} + \delta\mathbf{x}^{(1)}) = \mathbf{0} \\
\underset{\text{Taylor}}{\Longrightarrow} &\frac{d}{dt}C_0\delta\mathbf{x}^{(1)} + G_0\delta\mathbf{x}^{(1)} + \mathbf{s}^{(0)} + O(\|\delta\mathbf{x}^{(1)}\|^2) = \mathbf{0},
\end{aligned} \tag{4.14}$$

where

$$C_0(t) := C(t,\mathbf{x}^{(0)}(t)) = \frac{\partial\mathbf{q}(t,\mathbf{x}^{(0)}(t))}{\partial\mathbf{x}}, \tag{4.15}$$

$$G_0(t) := G(t,\mathbf{x}^{(0)}(t)) = \frac{\partial\mathbf{j}(t,\mathbf{x}^{(0)}(t))}{\partial\mathbf{x}}, \tag{4.16}$$

$$\mathbf{s}_0(t) := \frac{d}{dt}\mathbf{q}(t,\mathbf{x}^{(0)}) + \mathbf{j}(t,\mathbf{x}^{(0)}). \tag{4.17}$$

If the higher-order terms are neglected in equation 4.14, we get the following linear DAE for $\delta\mathbf{x}^1$:

$$\begin{aligned}
\frac{d}{dt}C_0\delta\mathbf{x}^{(1)} + G_0\delta\mathbf{x}^{(1)} &= -\mathbf{s}_0 \\
\big[\frac{d}{dt}C_0(t)\big]\delta\mathbf{x}^{(1)} + C_0(t)\frac{d}{dt}\delta\mathbf{x}^{(1)} + G_0(t)\delta\mathbf{x}^{(1)} &= -\mathbf{s}_0 \\
\big[\frac{d}{dt}C_0(t) + G_0(t)\big]\delta\mathbf{x}^{(1)} + C_0(t)\frac{d}{dt}\delta\mathbf{x}^{(1)} &= -\mathbf{s}_0,
\end{aligned} \tag{4.18a}$$

which can be combined with the periodicity condition:

$$\mathbf{x}^{(0)}(0) + \delta\mathbf{x}^{(1)}(0) = \mathbf{x}^{(0)}(T) + \delta\mathbf{x}^{(1)}(T). \tag{4.18b}$$

This linear system can be solved using the linear shooting method, as described in the previous section. Using the solution $\delta\mathbf{x}^{(1)}$ a new (and hopefully better)

approximation $\mathbf{x}^{(1)} := \mathbf{x}^{(0)} + \delta\mathbf{x}^{(1)}$ can be computed. This process can be repeated until a sufficiently good solution has been found. The resulting algorithm can be described in pseudo-code as

$\mathbf{x}^{(0)} :=$ some initial guess , not necessarily periodic

{compute G_0, C_0 and $\mathbf{s}_0$:}

$$\begin{aligned} G_0(t) &:= G(t, \mathbf{x}^{(0)}(t)) && \text{for } 0 \le t \le T \\ C_0(t) &:= C(t, \mathbf{x}^{(0)}(t)) && \text{for } 0 \le t \le T \\ \mathbf{s}_0(t) &:= \frac{d}{dt}\mathbf{q}(t, \mathbf{x}^{(0)}) + \mathbf{j}(t, \mathbf{x}^{(0)}) && \text{for } 0 \le t \le T \end{aligned}$$

$i := 0$

while $\|s_i\| \ge \epsilon$ **do**

solve $\delta\mathbf{x}^{(i+1)}$ in $\frac{d}{dt}C_i\delta\mathbf{x}^{(i+1)} + G_i\delta\mathbf{x}^{(i+1)} = -\mathbf{s}_i$,

$$\mathbf{x}^{(i)}(0) + \delta\mathbf{x}^{(i+1)}(0) = \mathbf{x}^{(i)}(T) + \delta\mathbf{x}^{(i+1)}(T).$$

$\mathbf{x}^{(i+1)} := \mathbf{x}^{(i)} + \delta\mathbf{x}^{(i+1)}$

{compute G_{i+1}, C_{i+1} and $\mathbf{s}_{i+1}$:}

$$\begin{aligned} G_{i+1}(t) &:= G(t, \mathbf{x}^{(i+1)}(t)) && \text{for } 0 \le t \le T \\ C_{i+1}(t) &:= C(t, \mathbf{x}^{(i+1)}(t)) && \text{for } 0 \le t \le T \\ \mathbf{s}_{i+1}(t) &:= \frac{d}{dt}\mathbf{q}(t, \mathbf{x}^{(i+1)}) + \mathbf{j}(t, \mathbf{x}^{(i+1)}) && \text{for } 0 \le t \le T \end{aligned}$$

$i := i + 1$

There are two essential differences between (single) shooting and Newton:

1. Wave-form-Newton uses the whole function $\mathbf{x}$ on the interval $[0, T]$ in its iteration. Single shooting stores only the value $\mathbf{x}(0)$. Of course, practical implementations of wave-form-Newton will only store $\mathbf{x}$ on a finite number of points $t_0, \ldots, t_N$.

2. The approximations computed by Wave-form-Newton satisfy the periodicity equation 3.1b during the whole iteration, except for the initial guess $\mathbf{x}^{(0)}$. However, the DAE 3.1a is not satisfied by the intermediate results. On the other hand, the shooting method satisfies 3.1a during the whole equation, but 3.1b is not satisfied by the intermediate results. This can be summarised by saying that the shooting method takes equation 3.1a as an invariant and tries to establish 3.1b, while Wave-form-Newton keeps 3.1b as an invariant and tries to establish 3.1a.

Finally, we want to remark that the pseudo-code above should not be taken too literal. Especially, the matrices G_i and C_i and the vector $\mathbf{s}_i$ can be computed while solving the linear system. Since they are only needed at the current time point in the solution process of the ODE 4.18b, they do not have to be stored for all time-points. So less memory is needed. However, Wave-form-Newton will still need much more memory than single shooting, since single shooting stores the solution at only one time-point, while Wave-form-Newton needs the solution at all the time-points $t_0, \ldots, t_N$.

4.5 Other methods

Another method for determining the PSS of a circuit is *epsilon extrapolation*, which can be found in [11]. A second method is the accelerated-Poincaré method, which has super-linear convergence for determing the PSS solution. It can be found in [13].

Chapter 5

Numerical method and initial estimate

5.1 Time integration method for IVP

Let us look back the problem described in Chapter 2. In this section we focus on the free-running oscillator which generates the equations:

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}) + \mathbf{j}(\mathbf{x}) = \mathbf{0}, \tag{5.1a}$$

$$\mathbf{x}(0) = \mathbf{x}(T). \tag{5.1b}$$

and we assume x_{pss} is a Periodic Steady State solution for the system (5.1), so it fulfills:

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}_{pss}(t)) + \mathbf{j}(\mathbf{x}_{pss}(t)) = \mathbf{0}. \tag{5.2}$$

Notice that here we assume the period T is an additional unknown. We can transform (5.2) like:

$$\begin{aligned} &\frac{d}{dt}\mathbf{q}(\mathbf{x}_{pss}(t)) + \mathbf{j}(\mathbf{x}_{pss}(t)) = \mathbf{0} \\ \Longrightarrow\ &\frac{\partial \mathbf{q}}{\partial \mathbf{x}} \cdot \frac{d}{dt}\mathbf{x}_{pss}(t) + \mathbf{j}(\mathbf{x}_{pss}(t)) = \mathbf{0} \end{aligned}$$

Define:

$$C(t) = \left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_{pss}(t)} \tag{5.3}$$

and we first assume that $C(t)$ is non-singular.

$$\text{Let } \mathbf{y}(t) = \frac{d}{dt}\mathbf{x}_{pss}(t) \tag{5.4}$$

$$\Longrightarrow C(t) \cdot \mathbf{y}(t) + \mathbf{j}(\mathbf{x}_{pss}(t)) = \mathbf{0}, \tag{5.5}$$

We differentiate (5.5) with respect to t:

$$\Longrightarrow \frac{d}{dt}\{C(t)\cdot \mathbf{y}(t)+\mathbf{j}(\mathbf{x}_{pss}(t))\}=\mathbf{0} \tag{5.6}$$

$$\Longrightarrow \frac{d}{dt}[C(t)\cdot \mathbf{y}(t)]+\left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_{pss}(t)}\cdot \mathbf{x}_{pss}^{'}(t)=\mathbf{0} \tag{5.7}$$

Define:

$$G(t)=\left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_{pss}(t)}, \tag{5.8}$$

$$\begin{aligned}
&\Longrightarrow \frac{d}{dt}[C(t)\cdot \mathbf{y}(t)]+G(t)\cdot \mathbf{y}(t)=\mathbf{0}.\\
&\Longrightarrow \frac{d}{dt}[C(t)\cdot \mathbf{y}(t)]=-G(t)\cdot \mathbf{y}(t).\\
\Longrightarrow &\frac{d}{dt}[C(t)\cdot \mathbf{y}(t)]=[-G(t)\cdot C(t)^{-1}]\cdot [C(t)\cdot \mathbf{y}(t)].
\end{aligned} \tag{5.9}$$

Then we define:

$$A(t)=-G(t)\cdot C(t)^{-1},\ \ \mathbf{w}(t)=C(t)\cdot \mathbf{y}(t). \tag{5.10}$$

To solve (5.2) is equivalent to solve the equation:

$$\frac{d}{dt}\mathbf{w}(t)=A(t)\cdot \mathbf{w}(t). \tag{5.11}$$

Now we set up a transient simulation with the following Initial Value Problem:

$$\frac{d}{dt}\mathbf{w}(t)=A(t)\cdot \mathbf{w}(t), \tag{5.12a}$$

$$\mathbf{w}(t)=C(t)\cdot \mathbf{y}(t), \tag{5.12b}$$

$$\mathbf{y}(t)=\frac{d}{dt}\mathbf{x}_{pss}(t), \tag{5.12c}$$

$$\mathbf{x}_{pss}(0)=\mathbf{x}_0 \tag{5.12d}$$

We define a number of grid points N, so that $\Delta t=\frac{T_{large}}{N}$ and $t_i=i\times\Delta t$ for $i=1,2,...N$. At each grid point of the time domain, we freeze the coefficient matrix $A(t)$ and $C(t)$. On $t_0\le t<t_1$, let $A(t)=A(t_0)$, and on $t_i\le t<t_{i+1}$ let $A(t)=A(t_i)$, so we have the equation:

$$\frac{d}{dt}\mathbf{w}(t)=A(t_i)\cdot \mathbf{w}(t)\quad \text{i=1,2,...,N} \tag{5.13}$$

Now apply the well-known results from linear algebra on finding two linearly independent solutions for each homogeneous linear system. Note that the general solution for (5.13) on each time domain is :

$$\mathbf{w} = \sum_{k=1}^{m} \alpha_k p_k e^{\lambda_k t} \qquad \text{m is the number of non-trivial eigenvalues of } A(t_i) \text{ involved.} \tag{5.14}$$

λ_i is the eigenvalue of the matrix $A(t_i)$, p_i is the corresponding eigenvector and α_i is the coefficient for each independent solution.
Then we solve (5.12b) and (5.12c) by a time integration method. Recall from (5.3) that $C(t)$ is nonsingular, so (5.12b) can be easily scaled by:

$$\mathbf{y}(t) \quad = \quad C^{-1}(t) \cdot \mathbf{w}(t).$$

For solving (5.12c) we choose Simpson's rule which has a 5th order of accuracy, and we find

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{\Delta t}{6}[\mathbf{y}(t_n) + \mathbf{y}(\frac{t_{n+1} + t_n}{2}) + \mathbf{y}(t_{n+1})] \quad \text{t} \in [0, t_{large}]. \tag{5.15}$$

We still have two questions to be answered. The first one is how to compute the eigenpair(λ, P) and the second one is how to know the initial value of $\mathbf{y}$.
The answer to the first question will be introduced in the next section. For the second one, we apply the following way:
choose another time step $\Delta t_1 \ll \Delta t$ and do one step time integration method to $\mathbf{x}_0$ and get an $\mathbf{x}_1$, then define:

$$\mathbf{y}_0 = \frac{\mathbf{x}_1 - \mathbf{x}_0}{\Delta t_1} \tag{5.16}$$

In pseudo-code, this algorithm looks as follows.
$\mathbf{x}_0 :=$ some initial guess for $\mathbf{x}$

$\mathbf{y}_0 = \frac{1}{\Delta t_1}(\mathbf{x}_1 - \mathbf{x}_0)$

$i := 0$

repeat

compute $C(t_i)$ and $G(t_i)$ from $\mathbf{x}_i$

$\mathbf{w}_i = C(t_i) \cdot \mathbf{y}_i$

determine α_k, λ_k, $\mathbf{p}_k$ by eigenvalue algorithm (DPA and RQI)

$\mathbf{w}_{i+1} = \sum_{k=1}^{n} \alpha_k P_k e^{\lambda_k \cdot i \cdot \Delta t}$

$$\mathbf{y}_{i+1} = C^{-1}(t_i) \cdot \mathbf{w}_{i+1}$$

compute $\mathbf{x}_{i+1}$ from (5.15)

$$i := i + 1$$

until $i = N_t = \frac{1}{\Delta t} T_{large}$ which is the number of time steps.
This gives a sequence $\mathbf{x_0}$,$\mathbf{x_1}$,...,$\mathbf{x_N}$ in which $\mathbf{x_1}$ is calculated with accurate time integration method and the other values are calculated by a coarse grid, but by the special method described above.

5.1.1 BDF Method for singular C

In the definition of 5.3, we assume C is invertible for all x, but in practice, C will be almost always singular, because of the algebraic equations. Then, the solution has to satisfy a number of algebraic equations. Because these algebraic equations also apply in $t = 0$, a proper initial solution has also to satisfy the algebraic equations. One clear criterion pops up when we try to integrate the DAE by the backward (or implicit) Euler method.This method approximates the derivative of $q(x(t))$ in (5.1a) with a finite difference:

$$\frac{q(x_k) - q(x_{k-1})}{t_k - t_{k-1}} + j(x_k) = 0, \quad k \geq 1, \tag{5.17}$$

Here x_k is an approximation to $x(t_k)$ and $t_0 = 0 < ... < t_{k-1} < t_k < ... \leq$T. The algebraic equation (5.17) is not necessarily linear. Newtons method can be used to solve x_k from (5.17). Then for each Newton iteration a linear system of the form:

$$Ax_k^{(i)} = b^{(i)}, \tag{5.18}$$

has to be solved, where A is the Jacobian associated with (5.17):

$$A = \frac{\alpha_0}{h} C + G, \quad \text{with } C = \frac{\partial q}{\partial x}|_{x=x_k^{(i-1)}}, \ G = \frac{\partial j}{\partial x}|_{x=x_k^{(i-1)}} \tag{5.19}$$

$\alpha_0 = 1$, and $h = t_k - t_{k-1}$.The backward Euler scheme of (5.17) uses a first order approximation for the derivative $\frac{d}{dt}q(x)$.Higher order schemes can be used as well; for instance

$$\frac{d}{dt}q(x_k) \approx \frac{\sum_{i=0}^{m} \beta_i q(x_{k-i})}{t_k - t_{k-1}} \tag{5.20}$$

with parameters β_i and $m < 7$. This leads to the so called BDF (backward differentiation formula) methods. Variable order BDF methods with variable step sizes are often used in practice in transient analysis of circuits.The linear systems arising in BDF methods have a structure similar as (5.18), with (5.19), but with a different value of α_0. Numerical problems may arise in the BDF method if the DAE (5.1a)

is of index > 1, see [2], Ch.3 .

With the stepsize h becoming smaller and smaller we require that $\frac{\alpha_0}{h}C + G$ becomes invertible. This is equivalent to require that $C + \lambda G$ is invertible for small λ, a condition on the pencil (C, G). For well-posedness reasons we even will require that the inverses of $C + \lambda G$ will be uniformly bounded for $\lambda \in B(0, \delta)$ (for some $\delta > 0$).
A forward Euler scheme for (5.1a) is obtained by replacing $j(x_k)$ by $j(x_{k-1})$ in (5.17). However, in this case the Jacobian $A = C/h$, is likely to be singular, and it also does not guarantee that $j_i(x_k) = 0$ for the algebraic equations. Therefore, forward Euler and also other explicit schemes are not suitable for (5.1).

5.2 Dominant pole algorithm and rayleigh quotient iteration for eigenvalue problem

In this Section we give the answer to how to compute the eigenvalues and eigenvectors for the system (5.12a). Some of the methods are well known, full space method like QR and QZ method, and sub-space methods like Lanczos, Arnoldi and Jacobi-Davidson method [15]. Here we will use the dominant pole algorithm and the rayleigh quotient iteration [15, 16].
The orientation of this section comes from the following Single Input Single Output (SISO) electrical system

$$\begin{cases} E\frac{d}{dt}\mathbf{x} &= A\mathbf{x}(t) + \mathbf{b}u(t) \\ y(t) &= \mathbf{c}^T\mathbf{x}(t), \end{cases}$$

where $A, E \in \mathbb{R}^{n\times n}$, $\mathbf{x}(t), \mathbf{b}, \mathbf{c} \in \mathbb{R}^n$ and $u(t), y(t) \in \mathbb{R}$. We define the transfer matrix $H(s)$ by :

$$H(s) = \mathbf{c}^T(sE - A)^{-1}\mathbf{b}, \tag{5.21}$$

where $s \in \mathbb{C}$.
Let the eigenvalues λ_j of (E, A), ie. the values λ_j for which $\det(E-\lambda_j A) = 0$, and the corresponding right and left eigenvectors be given by the triplets $(\lambda_j, \mathbf{x}_j, \mathbf{y}_j)$, and let the right and left eigenvectors corresponding to finite eigenvalues be scaled so that $\mathbf{y}_j^* E\mathbf{x}_j = 1$. Note that $\mathbf{y}_j^* E\mathbf{x}_k = 0$ for $j \neq k$ and $\lambda_j \neq \lambda_k$. Note that in practice E can be singular, and hence there are eigenvalues at infinity. However, these eigenmodes have no contribution to the transfer function and can be neglected. The transfer function $H(s)$ can be expressed as a sum of residues R_j over first order, non-defective poles [17]:

$$H(s) = \sum_{j=1}^{n} \frac{R_j}{s - \lambda_j}, \tag{5.22}$$

where the residues R_j are

$$R_j = (\mathbf{x}_j^T \mathbf{c})(\mathbf{y}_j^* \mathbf{b}).$$

An upperbound for $|H(s = i\omega)|$ is given by

$$|H(i\omega)| \leq \sum_{j=1}^{n} \frac{|R_j|}{\text{Re}(\lambda_j)},$$

where for every individual term the upperbound

$$\frac{|R_j|}{\text{Re}(\lambda_j)}$$

is reached for $i\omega = i\text{Im}(\lambda_j)$. A pole λ_j that corresponds to a residue R_j with large $|R_j|/|\text{Re}(\lambda_j)|$ is called a *dominant* pole, i.e. a pole that is well observable and controllable in the transfer function. This can also be observed from the corresponding Bode magnitude plot of $H(s)$, where peaks occur at frequencies close to the imaginary parts of the dominant poles of $H(s)$. An approximation of $H(s)$ that consists of $k < n$ terms with $|R_j|/|\text{Re}(\lambda_j)|$ above some value, determines an effective transfer function behavior [29]:

$$H_k(s) = \sum_{j=1}^{k} \frac{R_j}{s - \lambda_j}. \tag{5.23}$$

The problem of concern can now be formulated as: Given a SISO linear, time invariant system $(E, A, \mathbf{b}, \mathbf{c})$, compute $k \ll n$ dominant poles λ_j and the corresponding right and left eigenvectors $\mathbf{x}_j$ and $\mathbf{y}_j$.

5.2.1 The Dominant Pole Algorithm

The poles of transfer function (5.21) are the $\lambda \in \mathbb{C}$ for which $\lim_{s\to\lambda} H(s) = \infty$. Consider now the function $T : \mathbb{C} \longrightarrow \mathbb{C}$

$$T(s) = \frac{1}{H(s)}. \tag{5.24}$$

For a pole λ of $H(s)$, $\lim_{s\to\lambda} T(s) = 0$. In other words, the poles are the roots of $T(s)$ and a good candidate to find these roots is Newton's method. This idea is the basis of the Dominant Pole Algorithm (DPA) [23]. The derivative of $T(s)$ with respect to s is given by

$$T'(s) = -\frac{H'(s)}{H^2(s)}. \tag{5.25}$$

The derivative of $H(s)$ in (5.21) to s is

$$H'(s) = -\mathbf{c}^*(sE - A)^{-1}E(sE - A)^{-1}\mathbf{b}, \tag{5.26}$$

where it is used that the derivative of the inverse of a square matrix $A(s)$ is given by $d[A^{-1}(s)]/ds = -A^{-1}(s)A'(s)A^{-1}(s)$. Equations (5.25) and (5.26) lead to the following Newton scheme:

$$\begin{aligned} s_{k+1} &= s_k - \frac{T(s_k)}{T'(s_k)} \\ &= s_k + \frac{1}{H(s_k)} \frac{H^2(s_k)}{H'(s_k)} \\ &= s_k + \frac{H(s_k)}{H'(s_k)} \\ &= s_k - \frac{\mathbf{c}^*(s_k E - A)^{-1}\mathbf{b}}{\mathbf{c}^*(s_k E - A)^{-1}E(s_k E - A)^{-1}\mathbf{b}}. \end{aligned} \tag{5.27}$$

Formula (5.27) was originally derived in [1]. Using $\mathbf{v} = (s_k E - A)^{-1}\mathbf{b}$ and $\mathbf{w} = (s_k E - A)^{-*}\mathbf{c}$, an implementation of this Newton scheme is Algorithm 1, also known as the Dominant Pole Algorithm ([23]). Algorithm 1 converges asymp-

Algorithm 1 The Dominant Pole Algorithm (DPA)

Require: System $(E, A, \mathbf{b}, \mathbf{c})$, initial pole estimate s_1; tolerance $\epsilon \ll 1$

Ensure: Dominant pole λ and corresponding right and left eigenvectors $\mathbf{v}$ and $\mathbf{w}$

1: Set $k = 1$

2: **while** not converged **do**

3: Solve $\mathbf{v} \in \mathbb{C}^n$ from

$$(s_k E - A)\mathbf{v} = \mathbf{b}$$

4: Solve $\mathbf{w} \in \mathbb{C}^n$ from

$$(s_k E - A)^*\mathbf{w} = \mathbf{c}$$

5: Compute the new pole estimate

$$s_{k+1} = s_k - \frac{\mathbf{c}^*\mathbf{v}}{\mathbf{w}^* E \mathbf{v}}$$

6: The pole $\lambda = s_{k+1}$ with $\mathbf{x} = \mathbf{v}_k/\|\mathbf{v}_k\|_2$ and $\mathbf{y} = \mathbf{w}_k/\|\mathbf{w}_k\|_2$ has converged if

$$||A\mathbf{x} - s_{k+1}E\mathbf{x}||_2 < \epsilon$$

7: Set $k = k + 1$

8: **end while**

totically quadratically, but the solution depends on the initial shift and hence is not guarenteed to be the most dominant pole.

The algorithm 1 computes a single dominant pole. In order to compute more poles, as is required in practice, the algorithm can be extended to a subspace accelerated algorithm, using deflation to avoid computing the same pole again. The resulting algorithm, Subspace Accelerated DPA (SADPA), is described in detail

in [15]. This variant also uses a selection criterion to select the most dominant approximation in every iteration.

5.2.2 Two-sided Rayleigh Quotient Iteration

Algorithm 2 Two-sided Rayleigh quotient iteration

Require: System $(E, A, \mathbf{b}, \mathbf{c})$, initial pole estimate s_0 ; tolerance $\epsilon \ll 1$

Ensure: Approximate pole λ and corresponding right and left eigenvectors $\mathbf{v}$ and $\mathbf{w}$

1: $\mathbf{v}_0 = (s_0E - A)^{-1}\mathbf{b}$, $\mathbf{w}_0 = (s_0E - A)^{-1}\mathbf{c}$, and $s_1 = \rho(\mathbf{v}_0, \mathbf{w}_0)$

2: Set $k = 1$, $\mathbf{v}_0 = \mathbf{v}_0/\|\mathbf{v}_0\|_2$, and $\mathbf{w}_0 = \mathbf{w}_0/\|\mathbf{w}_0\|_2$

3: **while** not converged **do**

4: Solve $\mathbf{v} \in \mathbb{C}^n$ from

$$(s_kE - A)\mathbf{v} = E\mathbf{v}_{k-1}$$

5: Solve $\mathbf{w} \in \mathbb{C}^n$ from

$$(s_kE - A)^*\mathbf{w} = E\mathbf{w}_{k-1}$$

6: Compute the new pole estimate

$$s_{k+1} = s_k - \frac{\mathbf{c}^*\mathbf{v}}{\mathbf{w}^*E\mathbf{v}}$$

7: Set $\mathbf{v}_k = \mathbf{v}_k/\|\mathbf{v}_k\|_2$ and $\mathbf{w}_k = \mathbf{w}_k/\|\mathbf{w}_k\|_2$

8: The pole $\lambda = s_{k+1}$ with $\mathbf{x} = \mathbf{v}_k$ and $\mathbf{y} = \mathbf{w}_k$has converged if

$$||A\mathbf{x} - s_{k+1}E\mathbf{x}||_2 < \epsilon$$

9: Set $k = k + 1$

10: **end while**

We first define the two-sided Rayleigh quotient:

Definition 5.2.1. *The generalized two-sided Rayleigh quotient* $\rho(\mathbf{x}, \mathbf{y})$ *is given by* $\rho(\mathbf{x}, \mathbf{y}) \equiv \mathbf{y}^*A\mathbf{x}/\mathbf{y}^*E\mathbf{x}$*, provided* $\mathbf{y}^*E\mathbf{x} \neq 0$.

The two-sided Rayleigh Quotient Iteration(RQI) [24, 25] is shown in Algorithm 2. The only difference with DPA is that the right-hand sides in step 3 and 4 of Algorithm 1 (DPA) are kept fixed, while the right-hand sides in step 4 and 5 of Alg. 2 (RQI) are updated every iteration.

If we compare the convergence behavior of DPA and RQI, we find if we provide an initial pole guess s_0, the DPA will find the dominant pole closest to imaginary axis, and RQI will convergence to the pole closest to s_0. We show a simple example with $E = I$ and $A = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$ and this matrix has 3 eigenvalues -1, i

and $-i$. We apply DPA and RQI to this matrix with the same initial guess $-0.5 + 0.1i$. From Fig. 5.1, the DPA converges to the pole closest to the imaginary axis $(+i)$, and in Fig. 5.2, the RQI converges to the pole closest to the initial shift, (-1). Note the RQI has cubic rate of convergence [15] and DPA has quadratic. The price one has to pay for the cubic convergence, is the smaller convergence region. In general, DPA has a much larger convergence region [16]. This is shown in Fig 5.3 (with courtesy of J. Rommes). While DPA takes advantage of the information in the right-hand side $\mathbf{b}$ and $\mathbf{c}$, RQI only takes advantage of this in the first iteration. So the input source $\mathbf{b}$ and $\mathbf{c}$ are more improtant for DPA especially for the large system.

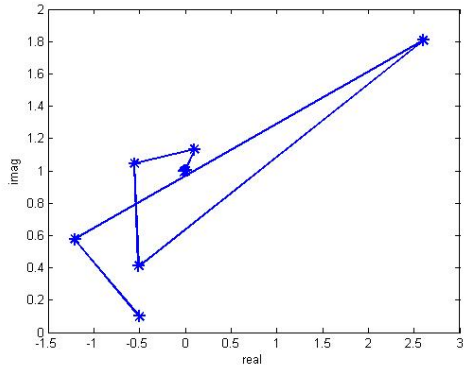


Figure 5.1: *Estimate of pole after each iteration by DPA with the initial guess*$-0.5 + 0.1i$

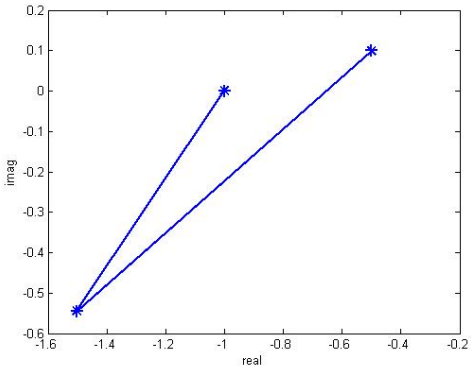


Figure 5.2: *Estimate of pole after each iteration by RQI with the initial guess*$-0.5 + 0.1i$

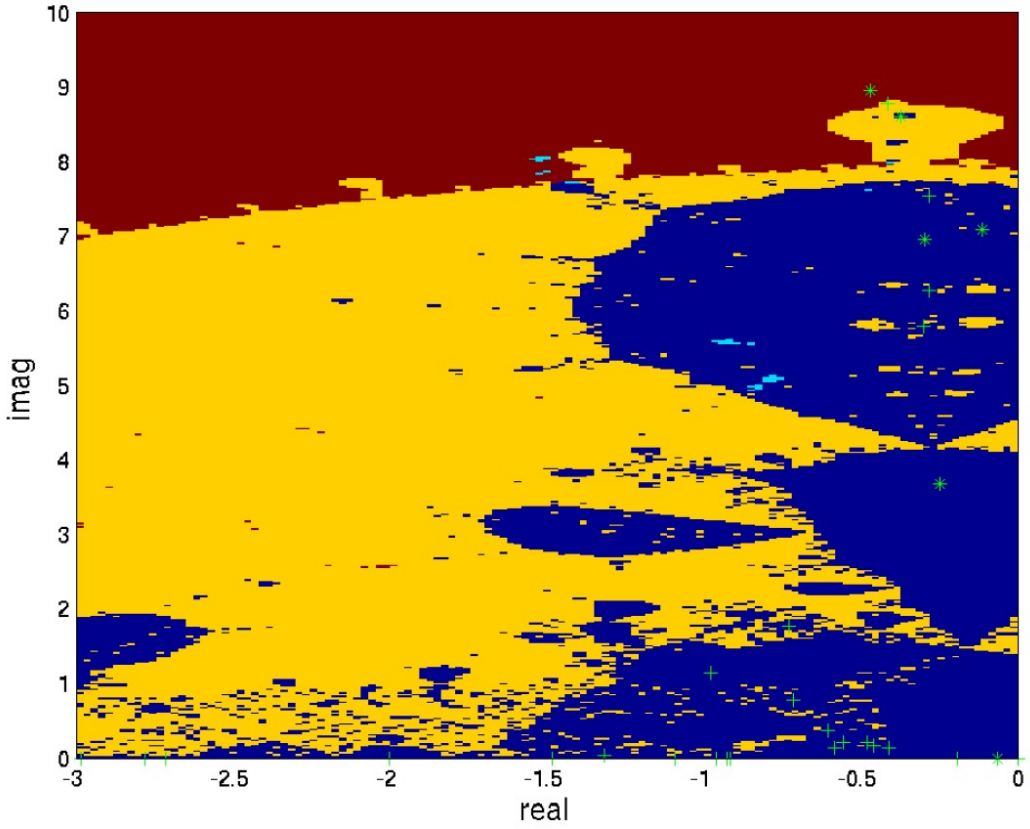


Figure 5.3: *(courtesy J. Rommes)* $\lambda = 0.47 + 8.9i$ *: DPA: red + yellow, RQI: red + light blue*

5.3 Initial estimate for Newton procedure

The Newton procedure for finding the periodic steady state solution always need an initial estimate. No doubt that a good initial guess will give a much better and faster iteration. To find these is the motivation for this section.
The initial conditions must satisfy the following requirements:

1. The initial guess $\mathbf{x}_0$ must not be a DC solution.
   (a DC solution is a trival soluton and we want the non-trivial periodic solution.)
2. The initial guess should preferable be periodic
3. It should be ”close” to the exact PSS solution

In the past no specific method for defining the initial estimate was given. People give a sine like curve or some other curve to ”expect” a good result. Now we give

an idea to define an initial estimate.
We notice that from Section 1 we produce a transient like solution which involve an eigenvalue problem. We may use the information from this procedure.

5.3.1 Initial estimate for the solution $\mathbf{X}_{PSS}$

We illustrate our approach by considering the next benchmark problem.

$$\frac{dx}{dt} = y + \varepsilon(1 - \sqrt{x^2 + y^2})x, \tag{5.28a}$$

$$\frac{dy}{dt} = -x + \varepsilon(1 - \sqrt{x^2 + y^2})y, \tag{5.28b}$$

with $\varepsilon = 0.1$, t in $[0, 10\pi]$ and initial value $[-1.5, 0.5]'$. The exact PSS solution of this problem is:

$$\begin{aligned} x(t) &= a\sin(t - c) \\ y(t) &= a\cos(t - c) \end{aligned}$$

With a and c are constant coefficients and period $T = 2\pi$. In Fig 5.5, by perform-

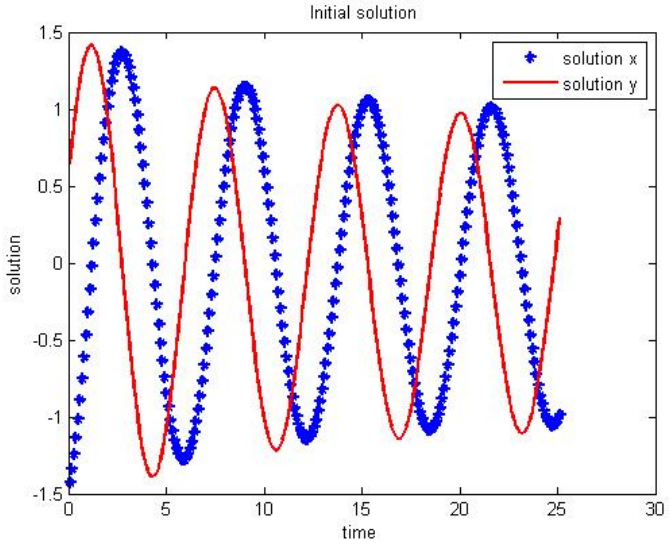


Figure 5.4: $x(t)$ and $y(t)$ for $0 \leq t \leq 8\pi$ after Transient simulation

ing transient integration, we find that the simulation will converge towards to the limit cycle, So a part of the solution in Fig. 5.4 should be a good estimate. Note that we should select one period of the solution for the Newton iteration, but T is an unknown in the autonomous case. The T can be detected by several methods to be introduced in the next subsection. From Fig 5.6, we give the simulation for a long time at least several periods (only $y(t)$ shows), For each period T_i, we also

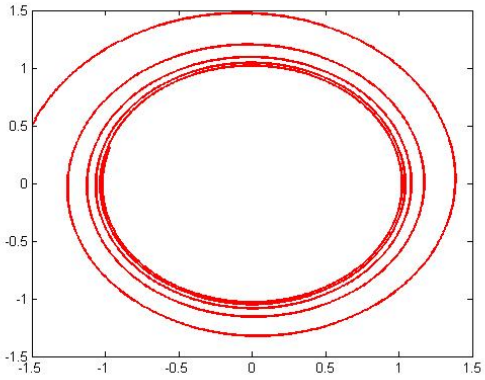


Figure 5.5: *Plot of* $(x(t), y(t))$ *for various values of* t *and* $(x(0), y(0)) = (-1.5, 0.5)$

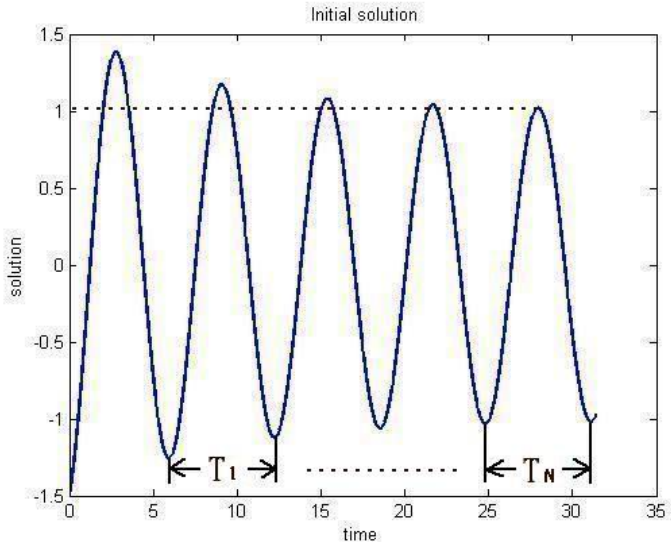


Figure 5.6: choice of solution

need to determine the solution in one interval of periodicity. We select the solution of T_1 and T_N for comparison in Fig. 5.6. The solution in the period T_N is the last period of the solution in our simulation, and it has already convergence to the limit cycle in Fig 5.5. The solution in the period T_1 is the second period and not converge yet, so the solution in period T_N is better than the solution during the period T_1. But in our experiments when starting with the solution of T_1 even approachs a good convergence result.

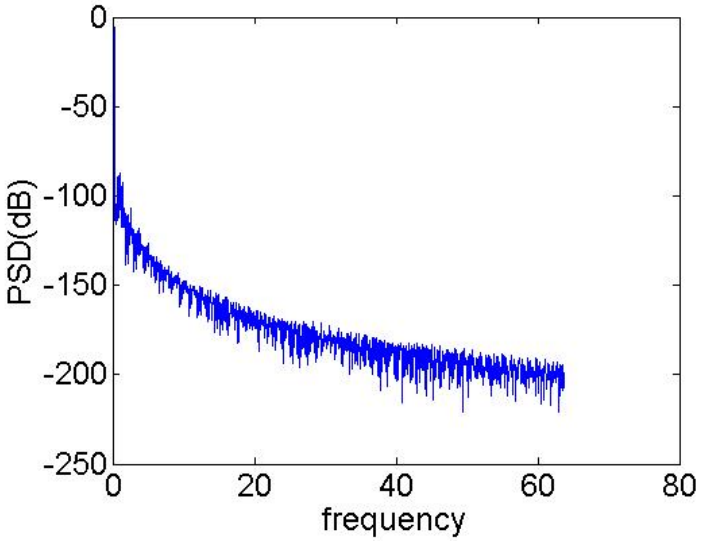


Figure 5.7: FFT to the initial solution in 5.4

5.3.2 Initial estimate the period T

Here we apply three methods for the estimation of the period T to the previous example:

1. The first method is based on Fast Fourier transform (FFT). Once we have generated a solution by transient simulation, we apply the FFT to this solution and consider the power spectrum. For the single frequency oscillator, the peak of the magnitude corresponds to the frequency of the oscillator. From this we can calculate T.

   In Figure 5.7, the x-axis is the frequency and the y-axis is the power spectral density(PSD), being the squared modulus of the Fourier transform of the time series. By a Matlab program, we compute the frequency $f = 0.1592$, so the period $T = 1/f = 6.2833$, which is a very good estimate for the exact period =2π.

2. The second method is based on the Dominant Pole Algorithm. In the transient simulation, to solve (5.12a) we apply the Dominant Pole Algorithm to compute the dominant eigenvalue and corresponding eigenvector for each time step, and we have the following relation

   $$Im(\lambda_i) = 2\pi f_i = 2\pi/T_i \tag{5.29}$$

   From Figure 5.8 we can see that T_i also converge to the exact period $T = 2\pi$.

3. The third method is based on the Rayleigh Quotient Iteration. This can seem as the improvement to the second method. When computing a eigenvalue for

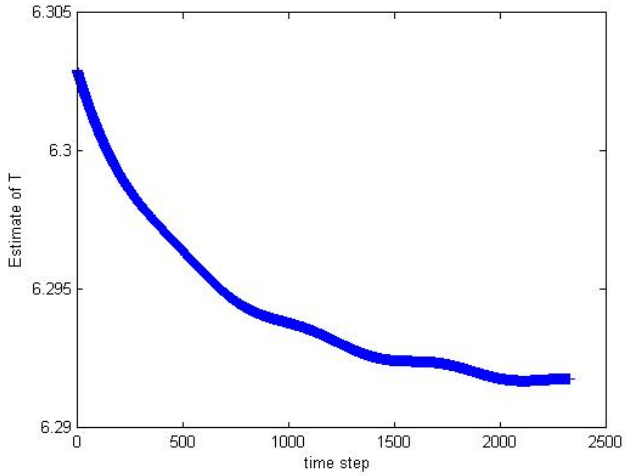


Figure 5.8: Estimate of T on each time step

each time step, the Rayleigh Quotient Iteration can convergence to the pole closest to the initial guess of the pole. This means if we know the range of the pole we want apriori, RQI can converge faster to the pole than DPA.

Chapter 6

Example

6.1 Benchmark oscillator

$$\frac{dx}{dt} = y + h(\sqrt{x^2 + y^2})x, \tag{6.1a}$$

$$\frac{dy}{dt} = -x + h(\sqrt{x^2 + y^2})y, \tag{6.1b}$$

The ODE (6.1) has the following properties

- It has at least one PSS solution, namely the stationary state with $\mathbf{x} = \mathbf{0}$. However, this solution is unstable.
- For every $r_k > 0$ satisfying $h(r_k) = 0$, we see that the circle described by $x^2 + y^2 = r_k^2$ is a limit cycle.

Some interesting choices for h are

- For $h(r) = \cos r$ there is an infinite number of stable and unstable limit cycles. Every zero of h corresponds to a limit cycle of (6.1). This choice of h shows that it is possible for several limit cycles to exist.
- For $h(r) = \varepsilon(1 - r)$, there is exactly one stable limit cycle, namely the unit circle. However, the solution $\mathbf{x}(t) \equiv \mathbf{0}$ is an unstable stationary solution. The single limit cycle is stable; convergence speed towards this limit cycle is determined by the parameter $\varepsilon > 0$. We did experiments on different $\varepsilon = 0.0001, 0.001, 0.01, ...$ and we found the closer ε approaches 0, the slower convergence becomes. Also we found the $\varepsilon = 0.1$ has the best convergence property. The fact that we can tune convergence speed with ε makes this particular problem a suitable benchmark problem.

We take $h(r) = \varepsilon(1 - r)$ and $\varepsilon = 0.1$. Figure 6.1 shows both the initial solution (dot) and the final PSS solution (continuous line) on a scaled interval $[0, 1]$. The initial value for $T = 6.2833 \approx 2\pi$. The initial solution was determined by one

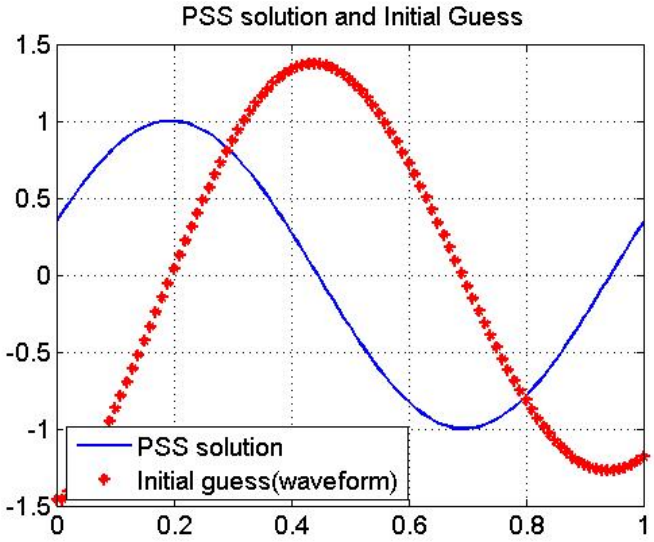


Figure 6.1: Initial Guess and Exact Solution $x(t)$

period of the numerical solution by the special method. We see that the Newton procedure gives a correction to the PSS solution starting from the Initial Guess.
Figure 6.2 shows details of convergence process of the Newton process. The upper

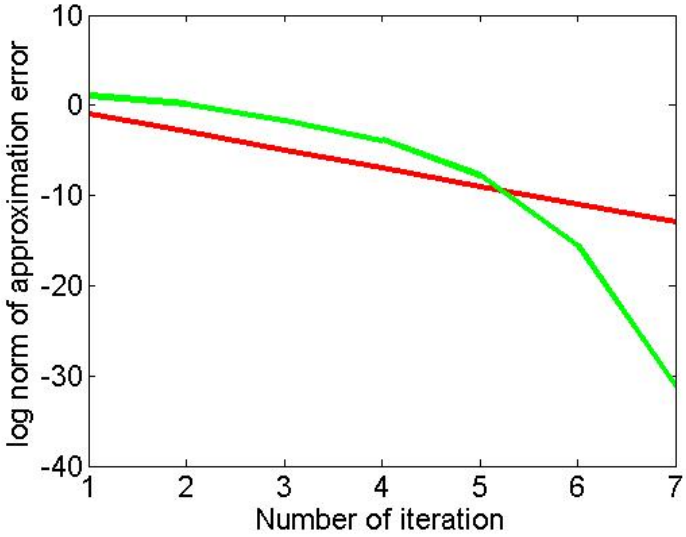


Figure 6.2: log norm of approximation error for each Newton iteration

line is the log norm of approximation error (of the corrections) for each iteration step, the lower line is the quadratic convergence line. We see that during the first

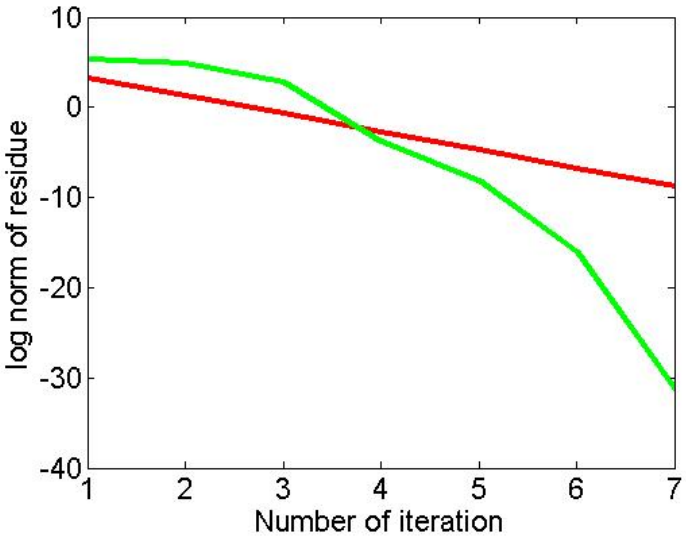


Figure 6.3: log norm of residue for each Newton iteration

few iterations we did not reach the quadratic convergence rate, but at last it gives super-quadratic convergence. And this results is similar to the plot of log norm of the residue in Fig. 6.3.

6.2 LC oscillator

For many applications [8, 19, 27] oscillators can be modeled as an LC tank with a nonlinear resistor as shown in Fig 6.4. This circuit is governed by the following differential equations for the unknowns (v, i) :

$$C\frac{dv(t)}{dt} + v(t)\frac{1}{R} + i(t) + S\tanh(\frac{Gn}{S}v(t)) = 0, \tag{6.2a}$$

$$L\frac{di(t)}{dt} - v(t) = 0, \tag{6.2b}$$

where C, L and R are the capacitance, inductance and resistance, respectively. The nodal voltage is denoted by v and the branch current of the inductor is denoted by i. The voltage controlled nonlinear resistor is defined by S and G_n, where S determines the oscillation amplitude and G_n is the gain.
A lot of work has been done for the simulation of this type of oscillators. Here we will give the PSS solution and the performance of the Newton procedure starting from our initial guess. For the given RLC circuit we used the following parameter values $L = 10^{-9}/(2 \times \pi)[H]$, $C = 1/23.041 \times 10^{-9}/(2 \times \pi)[F]$, $R = 1000[\Omega]$, $S = 1/R$ $Gn = -1.1/R$. The specific parameters L and C

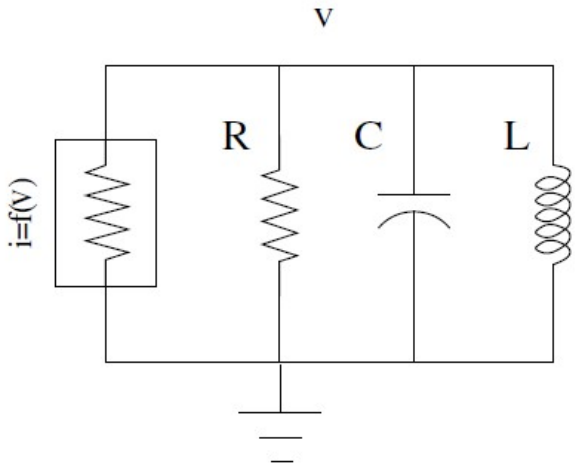


Figure 6.4: Voltage controlled oscillator, $f(v) = S\tanh(\frac{Gn}{S}v(t))$

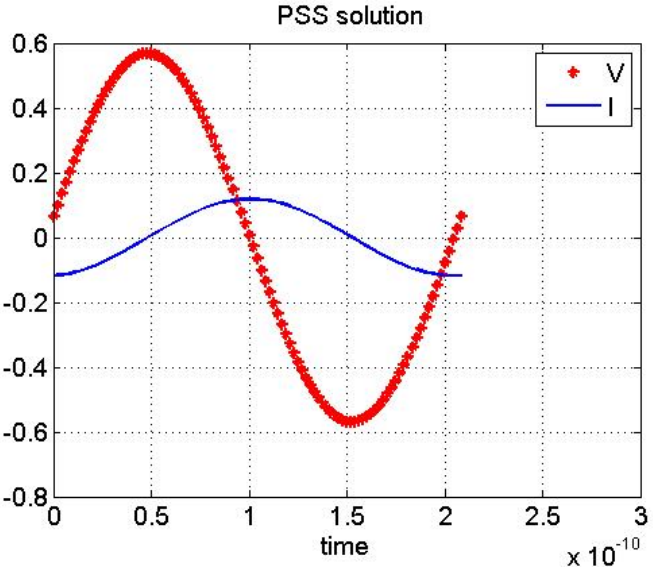


Figure 6.5: PSS Solution

are to give $f = 4.8Ghz$.

In Figure 6.5, we see the PSS solution for this LC-oscillator. The solution of the voltage $v(t)$ has a maximum value of 0.6 no matter what initial value is given due to it's special structure.

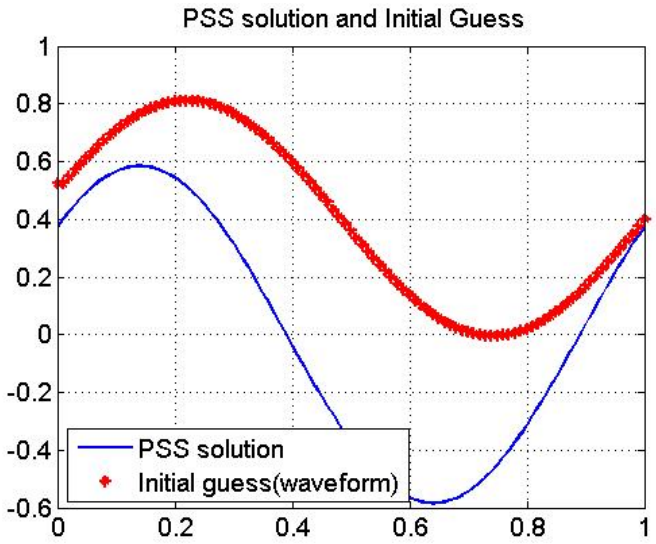


Figure 6.6: Compare Initial guess and PSS solution

Figure 6.6 gives the initial guess (dotted curve) and the PSS solution (continuous line) on the same scaled time domain $[0, 1]$.

In Table 6.1, the total number of iterations is listed. We compare our initial guess

Tol	10^{-5}	10^{-10}	10^{-12}	10^{-13}	10^{-15}
DC	8	9	9	10	34
TR	6	7	7	7	15

Table 6.1: *The numbers of iterations by the modified DC initial guess and TR initial guess*

with the initial value produced by a modified DC solution which is the DC solution plus a small perturbated cosine wave. When decreasing the tolerance for the corrections of the Newton procedure from 10^{-5} to 10^{-15}, we can see that when starting with the modified DC solution the number of iterations increases faster than when starting with the solution provided by the transient method.

In Fig 6.7, these two kinds of initial guess are compared with PSS solution, we can see that the waveform guess generated by our numerical method is much more sinusoidal.

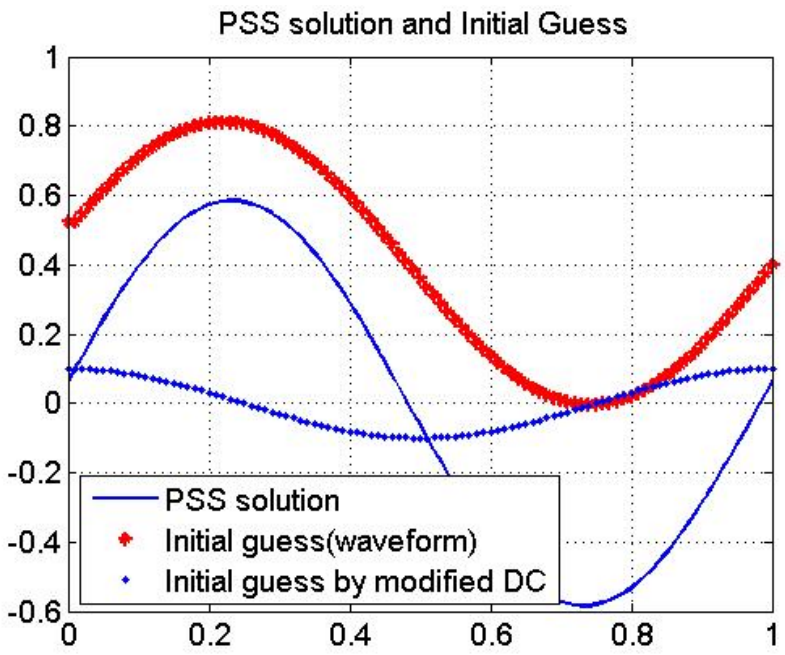


Figure 6.7: Comparison of two initial guesses with PSS

Chapter 7

Conclusion

7.1 Conclusions

The following conclusions can be reached:

1. In this report, several methods for finding a PSS have been introduced.
2. A numerical method for giving a good initial estimate has been used, it saves numerical cost compared to other initial guesses.
3. Stability analysis for a ODE system with different cases has been listed, also some counterexamples are provided.
4. Several methods for computing eigenvalues and eigenvectors are compared, specific eigenvalues can be selected by DPA or RQI.
5. Three ways to estimate the frequency of the oscillator are used, and both of them are efficient.

7.2 Future research

1. It seems that not all the information from eigenvalues and eigenvectors is used. The eigenvector for the specific dominant pole may become a good initial guess for the time derivative of the PSS solution.
2. Comparisons of the initial estimate generated by our numerical method to other ways of initial guess can be implemented.
3. Research directed towards applying this initial estimate to other different and more complicated oscillators.

References

[1] BEZERRA, L. H. : *Written discussion to [23]. IEEE Trans. Power Syst. 11*, 1 (Feb 1996), 168.

[2] K. E. Brenan, S.L. Campbell, L.R. Petzold,: *Numerical solution of initial-value problems in differential-algebraic equations*. SIAM, Philadelphia, PA, 1996.

[3] A. Demir, A. Mehrotra, J. Roychowdhury: *Phase noise in oscillators: A unifying theory and numerical methods for characterisation*, DAC98, San Francisco, June 1998. Extended version in IEEE Trans. on Circuits and Systems - I: Fundamental Theory and Applications, Vol. 47, No. 5, pp. 655-674, 2000.

[4] A. Demir: *Phase noise in oscillators: DAEs and coloured noise sources*, Proc. ICCAD'98, Int. Conf. on Computer Aided Design, Nov. 8-12, 1998, San Jose, CA, USA, pp. 170-177, 1998.

[5] Peng Li and Wei Dong *Parallel Preconditioned Harmonic Balance for Analog Circuit Analysis* ,Texas A and M University, SIAM 09 CSE, Presentation

[6] R. M. Gray :*Toeplitz and Circulant Matrices: A Review* Deptartment of Electrical Engineering, Stanford University, 2006

[7] J. K. Hale :*Ordinary Differential Equations*, 2nd ed., Robert E. Krieger, Huntington, NY, 1980

[8] D. Harutyunyan , J. Rommes, E.J.W. ter Maten, W.H.A. Schilders *Simulation of mutually coupled oscillators using nonlinear phase macromodels* CASA Report, Eindhoven, Tu/e 2008

[9] D. Hinrichsen, A. Pritchard,: *Mathematical Systems Theory*, Springer-Verlag, Berlin, 2005

[10] M. W. Hirsch, S. Smale, AND R. L. Devaney, :*Differential Equations, Dynamical Systems, and an Introduction to Chaos*, 2nd ed. Pure Appl. Math. (Amst.) 60, Elsevier/ Academic Press, Amsterdam, 2004

[11] S.H.M.J. Houben : *Algorithms for periodic steady state analysis on electric circuits* Eindhoven : Eindhoven University of Technology, 1999

[12] E.J.W. ter Maten, J.G. Fijnvandraat, S.H.M.J. Houben, J.M.F. Peters *Time-domain fd-method for the periodic steady-state of free running oscillators* ED&T Analogue Simulation, Technical Note, 2002

[13] S.H.M.J. Houben,: *Circuits in motion : the numerical simulation of electrical oscillators* , PhD Thesis, Eindhoven : Technische Universiteit Eindhoven, 2003

[14] ROMMES, JOOST, AND MARTINS, N. :*Efficient computation of multivariable transfer function dominant poles using subspace acceleration.* PREPRINT 1344, UTRECHT UNIVERSITY, JAN. 2006.

[15] ROMMES, JOOST, *Methods for eigenvalue problems with applications in model order reduction* PHD THESIS, UTRECHT UNIVERSITY, 2007.

[16] ROMMES, JOOST, SLEIJPEN,G.L.G., *Convergence of the dominant pole algorithm and Rayleigh quotient iteration*, SIAM JOURNAL ON MATRIX ANALYSIS AND APPLICATIONS, VOL. 30, ISSUE 1, 2008, PP. 346-363

[17] KAILATH, T. :*Linear Systems*. PRENTICE-HALL, 1980.

[18] KREŠIMIR JOSIĆ , ROBERT ROSENBAUM :*Unstable Solutions of Nonautonomous Linear Differential Equations* , SIAM 2008 ,VOL. 50 , NO. 3 . PP. 570-584.

[19] X. LAI AND J. ROYCHOWDHURY : *Capturing oscillator injection locking via nonlinear phase-domain macromodels*, IEEE TRANS. MICRO. THEORY TECH., 52(9):2251-2261, SEPTEMBER 2004.

[20] R. LAMOUR, R. MÄRZ, R. WINKLER: *How Floquet Theory applies to Index 1 differential algebraic equations*, J. OF MATH. ANALYSIS AND APPLICS, VOL. 217, PP. 372-394, 1998.

[21] R. LAMOUR: *Floquet-Theory for differential-algebraic equations (DAE)*, ZAMM, VOL. 78-3, PP. S989-S990, 1998.

[22] L. MARKUS , H. YAMABE, :*Global stability criteria for differential systems*, OSAKA MATH. J. 12(1960) , PP. 305-317.

[23] MARTINS, N., LIMA, L. T. G., AND PINTO, H. J. C. P. : *Computing Dominant Poles of Power System Transfer Functions.* IEEE TRANS. POWER SYST. 11, 1 (FEB 1996), 162–170.

[24] AM OSTROWSKI: *On the convergence of the Rayleigh quotient iteration for the computation of the characteristic roots and vectors* ARCH.RATIONAL MECH. ANAL. 3 (1959), 325-340

[25] B.N. PANTAZIS AND D.B. SZYLD, *The Rayleigh quotient iteration and some generalizations for nonnormal matrices*, MATH.COMP. 28 (1974) NO. 127, 679-693

[26] G. PETERS AND J.H.WILKINSON, *Inverse iteration, ill-conditioned equations and Newton's method*, SIAM REVIEW 21(1979) , NO.3, 339-360

[27] B. RAZAVI :*A study of injection locking and pulling in oscillators*, IEEE J. SOLID-STATE CIRC.,39(9):1415-1424, SEPTEMBER 2004.

[28] D. RAOULX: *Oscillator's phase noise: theory and simulation*, PHILIPS RESEARCH, LEP - TECHNICAL REPORT NO. C 2000-751, 2000.

[29] SMITH, J. R., HAUER, J. F., TRUDNOWSKI, D. J., FATEHI, F., AND WOODS, C. S. : *Transfer Function Identification in Power System Application.* IEEE TRANS. POWER SYST. 8, 3 (AUG 1993), 1282–1290.

[30] J. STOER AND R. BULIRSCH *Introduction to Numerical Analysis* SPRINGER 1992.

[31] LIOYD N. TREFETHEN :*Spectral methods in MATLAB* PHILADELPHIA , SIAM, 2000

[32] FERDINAND VERHULST :*Nonlinear Differential Equations and Dynamical Systems* SECOND EDITION SPRINGER

[33] M.- Y. WU, *Some new results in linear time-varying systems*, IEEE TRANS, AUTOMAT, CONTROL, 20(1975), PP. 159-161.

Appendix A

appendix

A.1 Matlab Code

A.1.1 Code for eigenvalues algorithm and Initial guess

Different Circuit can be chosen.
Provide different initial guess and 3 ways for computing eigenvalues and eigenvector: DPA , RQI and matlab commend eig.

```
function [XPSS,XPSS0,XREAL,KREAL]=DPAdtest3(epsilon,circuit,x)
circuit =char('Jie_LC');
 plotconv=0;
 usexpss0=0;
 useeig=0;
 usedpa=1;
iniv=[];
X=[];
XPSS0=[];%estimate for XPSS
XREAL=[];% Estimate for Xreal
KREAL=[];
ALPHAA=[];
XPSS=[];
RIGHTEV=[];
POLES=[];

T=2*1e-9;
delta0=2*1e-12;
Nt=T/delta0;     % number of time steps
delta=1/(Nt);

 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%initial value
 x =[0.5 , -0.5]';
 %x=  1.0e-009 * [   1 ,   1]';
```

```
    [jukhhjkh,Y] = ode45(@(t,y)  rigidLC(t,y),[0 delta0/100],[x']);
%  [jukhhjkh,Y] = ode45(@(t,y) rigid(t,y,epsilon),[0 delta/100],[x']);
%  [jukhhjkh,Y] = ode45(@(t,y)  rigid(t,y,epsilon),[0 T],[x']);
%[jukhhjkh,Y] = ode45(@(t,y)  rigidLC(t,y),[0 T],[x']);
  x1=(Y(length(Y),:))';

%  hold on
 % plot(jukhhjkh,Y(:,1),'r-')
 %return
      y=(x1-x)/delta0*100;
      xpss=x;
     eval(circuit);
     y=C*y;

for n = 1 : Nt
  eval(circuit);
    A=-G*inv(C);
     if useeig
        [V,D]=eig(A);
         leftev=V;
         poles=D;
     end

     if usedpa
        E=eye(2);
        b=rand(2,1);
        c=b;
        d=0;
        p=2;

        s0=zeros(p,1);
        s0(1)=1.0e+012 *(  -0.0041 + 3.0160i );
        s0(2)=1.0e+012 *(  -0.0041 - 3.0160i );
        s0(3)=80i;
        tol=1e-2;
        imagtol=1e-7;
        normres=1;
        k=0;
        poles = [] ;
        leftev = [] ;
        rightev = [] ;
        residues = [] ;
        nr_it = 0 ;
        iter = 0 ;
        %shifts = s0 ;
        nr_it=[];

        it=0;
    while it<p
```

```
        norms=1;
         st=s0(it+1);
            while norms > tol & iter<10000
            k=k+1;
            iter=iter+1;
            nr_it(k)=iter;
            sEmA = st * E - A ;

             v = sEmA \ b ;
             w = sEmA' \ c ;
             st=(w' * A * v) / (w' * E * v);

             xnorm=v/norm(v);
             ynorm=w/norm(w);
             res=A*xnorm-st*(E*xnorm);
             normres(k)=norm(res);
             norms=norm(res);
      end
      v=v/(w' * E * v);
         %Deflation b and c
        b=b- E * v * w' *b;
        c=c- E' * w * v' *c;

        it=it+1;
        poles(it,it)=st;
        leftev=[leftev,xnorm];
        rightev=[rightev,ynorm];
end

  end


  %  lambda=imag(poles(1,1))/2/pi
 %     disp(' ')
%      pause
% plotpo=plot(n,imag(poles(1,1))/2/pi,'*');
%          xlabel('time step ');
%      ylabel('Estimate of frequency');
% set(plotpo,'LineWidth', 2)
% hold on

%%%%%%%%%%%%%%compute alpha %%%%%%%%%%%%%%%%

iniv=[leftev(:,1)*exp(poles(1,1)*n*delta0) , leftev(:,2)*exp(poles(2,2)*n*de

%pause
 andy=inv(iniv)*y;

 alpha=andy(1); % changed everytime
```

```
 beta=andy(2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%plot the error and number of iteration
 %    plot(nr_it, normres,'k-+');
%hold on

%%%%%%%%%%%%%%%%%%%%%%%%%%%check for solution
  y=zeros(2,1);
%   alpha=1;
%   beta=1;

 y=alpha*leftev(:,1)*exp(poles(1,1)*(n+1)*delta0);
 y=beta*leftev(:,2)*exp(poles(2,2)*(n+1)*delta0)+y;

 y=inv(C)*y;
%y=C*y;

%xpss=xpss+y;

% first order euler

%xpss=xpss+delta0*y;

 kreal=(real(alpha)*real(leftev(:,1))-imag(alpha)*imag(leftev(:,1))+
 real(beta)*real(leftev(:,2))-imag(beta)*imag(leftev(:,2)))/abs(poles(1,1));
 xreal=kreal*(real(poles(1,1))*cos(real(poles(1,1))*(n+1)*delta)+
 imag(poles(1,1))*sin(imag(poles(1,1))*(n+1)*delta));
 XREAL=[XREAL,xreal];
 KREAL=[KREAL,kreal];
%mid point rule

%xpss=xpss + delta *[0.5*?(y+alpha*rightev(:,1)*exp(poles(1,1)*(n+1)*delta)+beta*rig

%simpson rule
%xpss
xpss=xpss+delta0/6*[y+alpha*leftev(:,1)*exp(poles(1,1)*(n+1)*delta0)+beta*leftev(:,2
%xpss
%pause


      n=n+1;
      X=[ X , y ];
      XPSS=[XPSS, xpss];
      x=xpss;
      y=C*y;  % scale back to w

end
```

```
%%%%%%%%%%%%%%%%%Plot PSSsolution      X with Y%%%%%%%%%%%%%%%
%length(n)
% plot3(XPSS(1,:),XPSS(2,:),n,'*')
% %grid; axis( [-1.5 1.5 -1.5 1.5] );
% %  hold on
 %   plot(XPSS(1,:),XPSS(2,:));
% length(XPSS);
 if plotconv
    t=delta0:delta0:T;
%    length(t);
% %   length(XPSS);
% %   %hold on
% % %  plot(t,XPSS,'k-',t,XPSS0,'r',t,XREAL,'g')
% % %plot(t,XREAL,'g',t,KREAL,'r')
  plotxpss=plot(t,XPSS)
  set(plotxpss, 'LineWidth', 2);
 % T0=2*pi/imag(poles(1,1))
 end
```

A.1.2 Code for PSS

```
     % Using Newton .method for model problem  dx/dt = -y + epsilon(1-r)x
%                                          dy/dt=  x + epsilon(1-r)y

clc
clear all

%initial Value
circuit =char('Jie_LC');
N=2; % dimension of the solution
steps=100;     % number of time steps
dt = 1/steps;
bigx = zeros(N*steps + 1,1);
epsilon=1e-1; %coefficient of the model problem
tol=1e-10;
amp=0.4;
%amp=0.0000001;
x =[amp , -amp]';
%x=[-1.5, 0.1]';
%T0=2*pi;
T0=2.08*1e-10;    %must be a good estimate

oscnode=1;
plotfactor=0;   %plot the damping factor
```

```
plotconv=0;     %plot the norm of error
plotconv2=0;    % plot the pss solution
plotconv3=1;    % compare the initial guess and pss solution
plotxy=0;       % plot x with y
plotpoles=0;    %plot the pole on each iteration
eiAB=0;         % see the poles of matrix A and B
useprecondition=0;
useDCsolution=0;
method=1;
X=[];
POLESA=[];
POLESB=[];
LEFTEV=[];
RIGHTEV=[];
POLESAm=[];
errPOLES=[];

x0=zeros(N,1);
debug=0;
damping=1;
maxdT=T0/10;
%[XPSS]=DPAd_nonlinear(epsilon);
% [XPSS,XPSS0,XREAL,KREAL]=DPAd_nonlinear2(epsilon,circuit);
[XPSS,XPSS0,XREAL,KREAL]=DPAdtest3(epsilon,circuit,x);
%size(XPSS)
%lines = plot([0 XPSS(1,1:100)],'.r');
%return
[p,q]=size(XPSS0);
%plot(real(XPSS(2,:)))
%return
 %Give initial value for bigx and x0
%     x0=XPSS(:,q);
%      for i = 1:steps
%         bigx((i*N-N+1):(i*N),1) = XPSS(:,q-i+1);
%       end


 if useDCsolution

    xDC=1.0e-006 *[    0.1000 ,   0.1623]';

      for i = 1:steps
        bigx((i*N-N+1):(i*N),1) = xDC;
      end
     for i = 1:steps
       bigx((i-1)*N+oscnode,1) = bigx((i-1)*N+oscnode,1) + amp*cos(2*pi*i/steps);
     end
 else
```

```
    x0=XPSS(:,1);
         for i = 1:steps
        bigx((i*N-N+1):(i*N),1) = XPSS(:,i);
         end

 end


%      for i = 1:steps
%        bigx((i-1)*N+oscnode,1) = bigx((i-1)*N+oscnode,1) + amp*cos(2*pi*i/

    bigx(N*steps+1,1) = T0;


   step=0;
size(bigx);


if plotconv
 nrms = [];
end

fprintf(1,'Method: convoluting with ');
if method == 0
  fprintf(1,'cosine\n');
else method == 1
  fprintf(1,'delta function\n');
end



while 1

    [bigf,bigPhi,bigA,bigB,bigY] = getBigStuff(bigx, N , circuit , steps , o
    %rank(bigY)
%     %return
%     A=bigA;
%     B=bigB;
%     E=eigs(A,B);
    if eiAB
%     size(bigA);
%     size(bigB);
%
        p=1;
        A=bigA;
        B=bigB;
```

```
        E=B;
        %E=eye(length(bigA));
        %b=rand(length(bigA),1);
        b=rand(length(bigA),1);
        c=b;
        tol=1e-8;
        s0=-0.9;
%        E=eig(A,B);

%        E=eig(bigY);
%        POLESA=[POLESA,E];
%        [polesA,leftev] = DPAd(E,bigY,b,c,tol,s0);
        [polesA,leftev,rightev] = DPA(E,A,b,c,tol,s0);
        LEFTEV=[LEFTEV,leftev];
        RIGHTEV=[RIGHTEV,rightev];
        POLESA=[POLESA,polesA];
%        polesAm=eigs(bigY,1,'sr');
%        POLESAm=[POLESAm,polesAm];
        %POLESB=[POLESB,polesB];
%        det(bigY)
    end


    %     [bigdx] = solve(bigPhi,bigf,tol);
    if useprecondition
        preleftev=[leftev',0];
        prerightev=[rightev ; 0];
%         kkk=prerightev(1)/bigf(1)
%         prerightev(2)
%         kkkk=prerightev(2)-kkk*bigf(2)
%        alpha=-(preleftev * bigPhi * prerightev) \ (preleftev * bigf);
%        alpha=-  ((leftev') * bigf(1:length(bigf)-1)) /
((leftev')*bigPhi(1:length(bigf)-1 , 1:length(bigf)-1 )*rightev ) ;
        %return
%        bigdx=norm(alpha)*prerightev;
        %size(preleftev)
        %return
%        bigdx=(-preleftev*bigPhi)\ (preleftev *bigf);
%         bigdx=-[(((leftev')*bigPhi(1:length(bigf)-1 , 1:length(bigf)-1 ))
%\ (leftev'*bigf(1:length(bigf)-1) - leftev'*bigPhi(1:200 , 1)*1) ); 1];
    else
          bigdx = -bigPhi\bigf;




%        preleftev=[www',0];
%bigdx=(-preleftev*bigPhi)\ (preleftev *bigf);
```

```
     end
        error_linear_system=norm(bigPhi*bigdx+bigf);

        nrm = norm(bigdx,inf);

        step = step + 1;
    fprintf(1,'Step %d: %e\n', step, nrm);

    waveform = bigdx(1:steps*N,1);
    T = bigdx(steps*N+1, 1);

%         plot(step, (rightev),'+-r',step,nrm,'+g')
%         plot(real(rightev), imag(rightev),'b')
%          hold on
    %calculate damping factor
     factor = 1;

     normwaveform = norm(waveform,inf);
    %  fprintf(1,'Norm of waveform: %e\n', normwaveform);
     if factor*normwaveform >= damping
        factor = damping/normwaveform;
     end

     fprintf(1,'Suggested change in T: %e\n', T);
     fprintf(1,'Suggested change in waveform: %e\n', normwaveform);

    if factor*abs(T) >= maxdT
         factor = maxdT/abs(T);
    end
    if plotfactor
        plot(step,factor,'-.ob')
         hold on
    end
    %update x0 and T
    if factor ~= 1
        fprintf(1, 'Damped with factor %d\n', factor);
    end

    bigx = bigx + factor*bigdx;
    fprintf(1, 'T = %e\n', bigx(steps*N+1));
     fprintf(1, 'f = %e\n', 1/bigx(steps*N+1));
     if plotconv
        nrms = [nrms; log(nrm)]; % we can see the order of error
     end

  fprintf(1, '\n error norm = %e\n', nrm);
  if nrm <= tol
    break
  end
```

```
%     if debug
%    plotx = x0;
%    plott = 0;
%
%     for i = 1:steps
%       plotx(:,i) = bigx((i*N-N+1):(i*N),1);
%       plott(:,i) = bigx(steps*N+1)*i*dt;
%     end
%
%     plotx=[x0 plotx]; % davit added
%     plott=[0 plott]; % davit added
%
%     lines = plot(plott, plotx);
%     title([circuit ': debug']);
%     xlabel('time');
%
%     zoom on;
%
%     eval(strcat(circuit,'_labels'));
%     legend(labels);

   % pause;
  end




x0 = bigx((steps*N-N+1):(steps*N),1);
T = bigx(steps*N+1, 1);
%step
%size(nrms)
%choose two point to generate the order of Newton method

if plotconv
    y1=nrms(step-2);
    y2=nrms(step-6);
    st=1:1:step;
    ylin=-2*st + y2;
       plot(st,ylin,'r',st,nrms,'g')
%  plot(st,nrms,'r');
%  h = legend('kreal')
    hold on
  %  pause;
end

plotx = x0;
plott = 0;
```

```
for i = 1:steps
  plotx(:,i) = bigx((i*N-N+1):(i*N),1);
  plott(:,i) = T*i*dt; % scale back to the real time domain
end

plotx=[x0 plotx]; % davit added
plott=[0 plott]; % davit added

xpss=plotx;
tpss=plott;

%size(plott)
%size(XPSS(2,:))

if plotconv2
    set(gca,'FontSize',14)
 %   lines = plot(plott, plotx,plott,[0 XPSS(2,1:100)],'.r',plott,[0 XPSS(1,
%lines = plot(plott,[0 XPSS(2,1:100)],'.r',plott,[0 XPSS(1,1:100)],'.b');
lines = plot(plott, plotx);
set(lines,'LineWidth',2)
    grid on
    title([' PSS solution']);
    xlabel('time');
    [N,trikajin]=size(plotx);
%     l=[];
%     for i=1:N
%         l=[l
%             ['x(' num2str(i) ')']];
%     end
l= [['V']
    ['I']];
    legend(l)
    zoom on;

    %   eval(strcat(circuit,'_labels'));
    %   legend(labels);
end
%size(plotx)
plot0=0;
for i = 1:steps
  plot0(:,i) = i*dt; % scale back to the real time domain
end
plot0=[0,plot0];
%size(plot0)
if plotconv3
        set(gca,'FontSize',14)
    lines = plot(plot0, plotx(1,:),plot0,[XPSS(1,1) XPSS(1,1:100)],'*r');
%   lines = plot(plot0, plotx(1,:),plot0,[0 XREAL(1,1:100)],'.r');
   set(lines,'LineWidth',2)
```

```
    grid on
    title([' PSS solution and Initial Guess']);
    hhh = legend('PSS solution','Initial guess(waveform)', 3);
set(hhh,'Interpreter','none')
end

if plotxy
plot(plotx(1,:),plotx(2,:));
end

nrmP=[];
if plotpoles
    str=1:1:step;
%    poles1=POLESA(3);
%    poles2=POLESA(4);
%    poleslin=poles1+(str-3)*(poles2-poles1);
%    poleslin=poles1-(str-3)*2;
%    plot(str,real(POLESA),'r',str,poleslin,'g');
    for i=1:step
        errPOLES=POLESA(i)+1;
        normsPOLES(i)=norm(errPOLES,inf);
%        nrmP=[nrmP ; log(normsPOLES(i))];
%        nrmP(i)=log(normsPOLES(i));
    end
    nrmP=log(normsPOLES);
    size(nrmP);
        k1=nrmP(3);
        k2=nrmP(4);
        poleslin=k1-(str-3)*2;
%        nrmP
%        (normsPOLES)
%         size(str)
%         size(poleslin)
%    plot(str,nrmP,'r',str,poleslin,'g');
    xlabel('Number of iteration');
    ylabel('norm of the errors lambda+1');
end
```

A.1.3 Code for Newton method solving DC problem

```
x0=[20,16,4]';
Xini=x0;
XINI=[];
ATOL=1e-7;
FTOL=1e-6;
nrms=[];
X=[];
F=[];
```

```
cor=1;
res=1;
step=0;
while res >= FTOL

%bigf=zeros(3,1);
bigf=[-x0(3) - x0(1) + x0(2) , x0(1) - 2*x0(2) + 2* (x0(2))^2 - (x0(2)

bigDf=[-1, 1 -1 ; 1 , -2+4*x0(2) - 3*(x0(2)^2) 0 ; 1 , 0 , 0];

x1= x0 - inv(bigDf)*bigf;
X=[X,x1];
cor=norm(x1- x0);
step=step + 1;
x0=x1;
nrms = [nrms; log(cor)];
res=norm(bigf,inf);
F=[F, log(res)];
%  plot(step,cor,'*r')
%          hold on
end
%  st=1:1:step;
%  plot(st, nrms,'r')
 y1=F(step-2);
    y2=F(step-6);
    st=1:1:step;
%    XINI(st)=Xini;
    ylin=-2*st + y2;
       lines=plot(st,ylin,'r',st,F,'-go' , st , log(1e-6),'*');
       set(lines,'LineWidth',2);
xlabel('iteration number for FTOL')
ylabel('log of residue ')
%h= [['quadratic curve']  ['numerical results'] ,-1];

 %   legend(h)
%    zoom on;
h = legend('quadratic curve','log of residue', 'stopping criterion log
set(h,'Interpreter','none')
```

A.1.4 Computing the matrix needed for Newtion problem

```
function [bigf,bigPhi,bigA,bigB,bigY] = getBigStuff(bigx, N , circuit
theta=0.5;
dt = 1/steps;
```

```
  Cs = cell(steps,1);
  Gs = cell(steps,1);
  qs = cell(steps,1);
  js = cell(steps,1);

  for i = 1:steps
    x = bigx((i*N-N+1):(i*N),1);

    j = 0;
  t = i*dt;

    eval(circuit);

    Cs{i,1} = C;
    Gs{i,1} = G;
    qs{i,1} = q;
    js{i,1} = j;

  end

  T = bigx(steps*N+1,1);

  bigf = zeros(N*steps+1,1);
  bigPhi = spalloc(N*steps+1,N*steps+1,2*steps*N*N+N*steps+steps);
 bigA=zeros(N*steps,N*steps);
 bigB=zeros(N*steps,N*steps);
 bigY=zeros(N*steps,N*steps);
% bigA = spalloc(N*steps,N*steps,2*steps*N);
% bigB = spalloc(N*steps,N*steps,2*steps*N);
% bigY = spalloc(N*steps,N*steps,2*steps*N);
  j = steps;

  for i = 1:steps
    bigf((i*N-N+1):(i*N),1) = (qs{i,1}-qs{j,1})/dt + T*theta*js{i,1} + T*(1-t
    bigPhi((i*N-N+1):(i*N),(i*N-N+1):(i*N)) = Cs{i,1}/dt + T*theta*Gs{i,1};
    bigPhi((i*N-N+1):(i*N),(j*N-N+1):(j*N)) = - Cs{j,1}/dt + T*(1-theta)*Gs{j
    bigPhi((i*N-N+1):(i*N),N*steps+1) = theta*js{i,1} + (1-theta)*js{j,1};
    bigA((i*N-N+1):(i*N),(i*N-N+1):(i*N)) = Cs{i,1}/dt;
    bigA((i*N-N+1):(i*N),(j*N-N+1):(j*N)) = - Cs{j,1}/dt;
    bigB((i*N-N+1):(i*N),(i*N-N+1):(i*N)) = T*theta*Gs{i,1};  %omit the T par
    bigB((i*N-N+1):(i*N),(j*N-N+1):(j*N)) = T*(1-theta)*Gs{j,1};
    bigY((i*N-N+1):(i*N),(i*N-N+1):(i*N)) = Cs{i,1}/dt + T*theta*Gs{i,1};
    bigY((i*N-N+1):(i*N),(j*N-N+1):(j*N)) = - Cs{j,1}/dt + T*(1-theta)*Gs{j,1
```

```
    %bigPhi((i*N-N+1):(i*N),N*steps+1) = theta*js{i,1} + (1-theta)*js
    j = i;
  end

  if method == 0
    % convolution with cos


     bigf(N*steps+1,1) = 0.5*steps*amp;

    for i = 1:steps
      bigf(N*steps+1,1) = bigf(N*steps+1,1) - cos(2*pi*i/steps)*bigx(
      bigPhi(N*steps+1, (i-1)*N+oscnode) = cos(2*pi*i/steps);
    end


  else
    % convolution with delta spike
    bigf(N*steps+1,1) = bigx(oscnode,1) - amp;
    bigPhi(N*steps+1,oscnode) = 1;
  end
```

A.1.5 FFT to find the T

```
function [h,f_all,value_all]=data_plot(frequency,finj,V,dt,window_leng
if nargin==4
    window_length=20;
end
% F0=frequency;
Tstep=dt;

l_sim=length(V);

wtx=V';

mtx=abs(fft(wtx')).^2;
mtx=mtx/max(mtx);

%ploting
Fs=1/Tstep;
df=Fs/l_sim;
f=[df:df:Fs];

% % % % % % % % % % % % plot(f,10*log10(mtx))
```

```
% % % % % % % % % % % % return

%  f1=frequency-0.5*frequency;
%  f2=frequency+0.5*frequency;

% f1=df;
% f2=10*frequency;

f1=frequency-0.001*frequency;
f2=frequency+400*frequency;
l_sim;


n1=floor(f1/df);
n2=floor(f2/df);


if label==1

set(gca,'FontSize',14);

h=plot((f(n1:n2)-df),10*log10(mtx(n1:n2)))
hold on
limit=axis;
 plot([frequency frequency],[ limit(3) limit(4) ],'--b')
xlabel('frequency')
ylabel('PSD(dB)')

%  grid minor

elseif label==2
% plot using kaiser window

set(gca,'FontSize',18);
 win=kaiser(l_sim,window_length);


wtx=V'.*win;
mtx=abs(fft(wtx')).^2;
mtx=mtx/max(mtx);
h=plot((f(n1:n2)-df),10*log10(mtx(n1:n2)))

f_all=(f(n1:n2)-df);
value_all=10*log10(mtx(n1:n2));
```

```
% hold on
%   plot([qw qw],[ limit(3) limit(4) ],'-.r')



% l=get(gca,'XTick');
% set(gca,'XTick',sort([l 4.6e+9 4e+9 3e+9 2e+9 1e+9 5e+9 6e+9 7e+9]))
xlabel('frequency')
ylabel('PSD(dB)')

end
```

```
% figure(4)
% plot(t/T,x(1,:))
% legend('x1')
%
% % omega=2*pi*frequency;
% % omega=omega+omega/100*injection_param.tol;
% % b_out=max(x(1,:))*sin(omega*t);
% % hold on
```

```
% % plot(t/T,b_out,'-r')
% % legend('x1','injec')
% % axis([0 cycle_value -1.5 1.5])
%
% figure(5)
% set(gca,'FontSize',14);
% plot(t/T,x(2,:))
% legend('x2')

% omega=2*pi*frequency;
% omega=omega+omega/100*injection_param.tol;
% b_out=max(x(2,:))*sin(omega*t);
% hold on
% plot(t/T,b_out,'-r')
% legend('x2','injec')
% axis([0 cycle_value -max(x(2,:))-max(x(2,:))/2 max(x(2,:))+max(x(2,:))/2])
```