OVNI: (Object Virtual Network Integrator) A New Fast Algorithm for the Simulation of Very Large Electric Networks in Real Time

by

LUIS RAFAEL LINARES-ROJAS

Elec. Eng., Universidad Central de Venezuela, Caracas, Venezuela, 1981
M.A.Sc.,The University of British Columbia, Vancouver, Canada, 1993

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Department of Electrical & Computer Engineering)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 2000

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of ELECTRICAL AND COMPUTER ENGINEERING

The University of British Columbia
Vancouver, Canada

Date AUG. 31, 2000

DE-6 (2/88)

ABSTRACT

A portable fast algorithm for solving power electric and electronic networks, and its implementation in the real-time simulator OVNI, are introduced. The implementation of OVNI, object virtual network integrator, on an off-the-shelf hardware platform, a 400 MHz Pentium-II workstation is presented. Simplified fast-models, based on those used by the EMTP[1], are included for the network elements: lumped resistors, capacitors, inductors —both linear and non-linear— and a constant parameters transmission lines model. Real-time models for HVDC rectifying and inverting bridges, and for the corresponding PI-controllers, using node hiding, were created specially for OVNI and reported in this thesis. Core saturation and zero sequence flux in three phase core transformers are modelled. Fast non-linear models are included for current and potential transformers. A fast modelling scheme to account for switching operations is presented, and its successful implementation on an industrial product, reported. Multilayer segmentation of the network, topological segmentation followed by MATE[2] segmentation, the node hiding technique, and a history sources limited encapsulation scheme are introduced. Two fast asynchronous commutation modelling techniques —DSDI[3] and BIFE[4]— to eliminate spikes and numerical oscillations are introduced. Industrial real-time test cases are included for power system protective relays, and for high-voltage DC bridges and their corresponding controllers.

[1] Acronym for Electromagnetic Transients Program.
[2] Multi-Area Thevenin Equivalent.
[3] Double Step, Double Interpolation.
[4] Backward Interpolation, Forward Extrapolation.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This research began as a quest for an algorithm to solve power system networks that was fast enough as to perform real-time equipment testing.

Testing of the algorithm focused on two cases provided by industry: a protective relay test case, and an HVDC controller test case.

The work took the EMTP's algorithm as a starting point. The EMTP turned out to be more than sixty times too slow for the second case mentioned above, and fourteen times too slow for the first case.

In the first of the test cases, that algorithm spent more than two-thirds of the time solving the nodal equation system, $[G][v] = [h]$[1]. To accelerate the solution process, precalculation of all possible $[G]$ matrices (and of their triangular decompositions) was considered. It is easier to visualize the obstacles ahead of this approach through an example (which will be detailed later in this thesis): a 1000-node network with 1000 switches would require several trillions of Earth-sized planets covered with RAM chips (continents and oceans as well) to provide for storage to such set of matrices. However, conveniently segmenting the same network, would bring down the memory requirements to less than 180 kilobytes.

Segmentation was introduced in three different forms: the one suggested by the time delay provided by transmission lines (topological segmentation), the new Multi-Area Thevenin Equivalent (expanded and presented in this thesis in its full potential for the first time), and the also new node-hiding procedure. The combination of those segmentation strategies was labelled multi-layer seg-

[1] Where $[G]$ is the network bus conductance matrix; $[v]$ is the vector of nodal voltages, to be computed; and $[h]$ is the vector of total nodal currents.

mentation. This segmentation yielded the performance looked for.

To eliminate the voltage spikes produced by switch or valve openings that occur between simulation points, a new mechanism was introduced, the new double-step and double-interpolation procedure, a technique that backtracks to the occurrence of the switching event, and then advances by a double step to fall back in synchronism with the real-time train of samples.

Buttressing the algorithm's robustness and stability, a careful integration-rule study shed new light into the effect (in the time-domain) of the phase shift that the backward Euler integration rule introduces (in the phase-domain).

The result of this work is a very fast and stable algorithm with no loss of generality. During testing, as reported in this thesis, the algorithm delivered real-time performance for the demanding test cases outlined above, and it did so on an off-the-shelf PC-Pentium 400 MHz workstation.

This thesis is divided in several *parts*, as follows:

1. *Motivation.* A brief account of the events that triggered this research;

2. *The problem.* A description of the challenge to overcome at the outset of the work;

3. *The Solution.* This is the main part of the thesis, it contains its contributions, which are scattered among several chapters: Chapter 3 presents a new look at the backward Euler integration rule; Chapter 5 introduces, in its general format, precalculation riding on top of a multi-layer form of network segmentation (topological segmentation, the new Multi-Area Thevenin Equivalent concept, and the also new node-hiding segmentation strategy); Chapter 6 describes the precalculation subtleties of periodic sources used in OVNI, and extends and generalizes the multi-area Thevenin equivalent concept to produce the very efficient tool that bring the performance needed to meet the real-time deadline targeted (less than fifty microseconds for the test case described above); Chapter 7 presents, among other things, the new double step with double interpolation back-

tracking algorithm used to eliminate the voltage spikes introduced by opening of switches between the instants where the simulation solves the network, it does that with a mimimum overhead that keeps the whole simulation within the real-time deadline;

4. *New Models.* Chapters 8, 9, 10, and 11 include measuring transformer models, some non-linear element models with fast topology-change, and a minimal functionality controller, the last two as examples of the implementation of the node hiding strategy on an element model, and on the creation of two element models that interact with one another;

5. *Implementation.* Chapters 12 and 13 describe the implementation of the simulator core and of its preprocessor with some minimal detail;

6. *Validation Tests.* Chapter 14 shows several test cases where the simulator delivered results whose accuracy is compared with those of the EMTP, those results were obtained within the real-time bandwidth targeted;

7. *Conclusions.* Finally, Chapter 15 closes the thesis with a summary of conclusions.

ACKNOWLEDGEMENTS

"No book published is ever solely the work of the author. Assistance comes from a variety of sources in as many different ways," wrote Jean M. Auel at the introduction to her best-selling novel The Clan of the Cave Bear. This cannot be more true than in the case of a thesis. So many people have contributed in one way or another to help me reach this goal, from my mother, Rita Elena, who decided to teach her three year old son to read and write, and both my wonderful elementary school teachers, Sra. Rojas and Prof. Hernandez, to my father Rafael Jose, who insisted on integrity, curiosity, and originality as the hallmark of a true human being.

My recognition and gratitude to Prof. Marti, who put in my hands this most critical part of the OVNI project, and who always believed in my skill to somehow pull the proverbial rabbit out of the hat and produce an ever faster and faster algorithm. I want to acknowledge him for his financial support, for sharing his impressive knowledge and intuition with me, but above all I want to thank him for his trust.

To Prof. Dommel, whose prompt advice and guidance during the first years of my research widened my horizons in this his land, the land of the electromagnetic transients analysis, I want to acknowledge and thank his kindness and support, .

To Mrs. Doris Metcalf, Ms. Cathleen Holtvolg, Ms. Katy Brindamour, Mrs. Gail Schmidt, Ms. Anne Coates, Mr. Alan Prince, and Mr. Ken Madore, thanks for their friendship and patience.

To Prof. Donaldson and Prof. Davies for trusting me with the young minds of the students of ECE 263, 370, 373 and thus providing me with a necessary retreat from the intense research activity (albeit for a few hours).

For all the help in preparing and setting the slides for the final presentation, thanks to my daughter Jazmin Carolina, to my son Ivan Jose, and to my colleague and friend Richard Rivas.

My appreciation and recognition goes to my friend Mr. Jesus Calvino-Fraga, who interfaced OVNI with the real-world, and to my friend Mr. Jorge Hollman who ported OVNI to his multi-PC cluster which allowed the simulator code to perform at maximum efficiency.

I would also like to thank my friend and colleague, Dr. Salvador Acevedo, for sharing his power electronics knowledge with me, and for his patience and tolerance in having his model turned inside out and upside down to accommodate OVNI's interface and the corresponding node simplification schemes. Thanks for the many productive discussions about the *nachos*-problem.

To this wonderful land, to Canada, to her people, to her future, thanks for welcoming my family and myself, and for providing the platform on which all this has been possible.

Last but not least, I thank my wife Maria Josefina, for her almost inexhaustible patience, for her love, her support, and for kick-starting me when I needed it most.

To you all, my gratitude. May God bless you all!

Luis R. Linares-Rojas.

I dedicate this thesis to these three wonderful women

Maria Josefina, my wife

Jazmin Carolina, my daughter

Rita Elena, my mother

Part I

MOTIVATION

1. INTRODUCTION

This thesis describes an effort to develop a general purpose digital simulator for electric and electronic power networks, suitable for real-time closed-loop equipment tests under flexible constraints of bandwidth and network complexity.

Simulation of an electric network can be viewed as the process to determine its state at a certain number of points along the time axis. If the network is described by its circuit theory representation, its state can be obtained as the solution to a set of non-linear coupled partial differential equations [11]. Using nodal analysis, for instance, this mathematical representation includes one of such equations for each node in the network. Even for a small network, with only a few tens of nodes, the solution task is rather demanding. When the solution needs to be obtained within the constraints of a real-time simulation[1], the problem becomes even more challenging.

The Engineering community has been able to reduce the complexity of the problem of determining the state of the network, at the price of reducing the scope of the solution as well, by classifying the network's behaviour into operational areas of interest, and applying suitable simplifying assumptions to each of those areas separately. The most important of those areas are: steady-state power flow [12, 13, 14], slow transients [15], fast transients [4], short-circuits [16, 14], and real-time equipment testing.

This thesis presents an attempt to a unified solution, and explores its validity on two counts, fast transients simulations, and real-time simulations for equipment testing; away from analog simulations and into the realm of digital

[1] i.e., a few microseconds.

simulations.

For each of the areas of interest mentioned, industry counts on specialized software based on the corresponding assumptions and restrictions. In particular, for insulation coordination analysis, the standard tool, the EMTP[2] [4], is built around the widest of the assumption sets, and uses a powerful discretization process for the problem that provides the seed for the work presented in this report. It is then convenient to establish the place of the EMTP in current power engineering practice.

During the last decades, the electromagnetic transients program —EMTP— has been gaining ground that used to be the sole domain of the expensive and bulky analog network simulator TNA[3] [17], transients computations in power systems. Today, the EMTP is the standard tool for this kind of simulations. Even if already existing TNAs remain in service, most new needs are covered by EMTP installations.

Cost and room use are two main areas where the EMTP has clear advantage when compared with the TNA. Another advantage is enhanced flexibility: very accurate models for system components can be developed and incorporated into the EMTP. Such is the case of the power transmission line, whose distributed parameters nature is not representable with the scaled-down analog models available in a TNA[4]. In spite of those advantages, in cases when testing some device requires real-time interaction between the device and the power system it is connected to, the analog simulator TNA is very often still the answer.

However, if a computer program is to attain real-time performance while simulating a power network, the program has to be capable of solving the system equations fast enough to encompass the bandwidth required for the equipment under test. In both, protective relay tests, and in HVDC controller tests, a bandwidth between 2,000 Hz and 4,000 Hz is considered adequate [18, 19, 9].

[2] Electromagnetic Transients Program

[3] Transients Network Analyser.

[4] Hybrid simulators include the best of both worlds, digital and analog, but at very high costs.

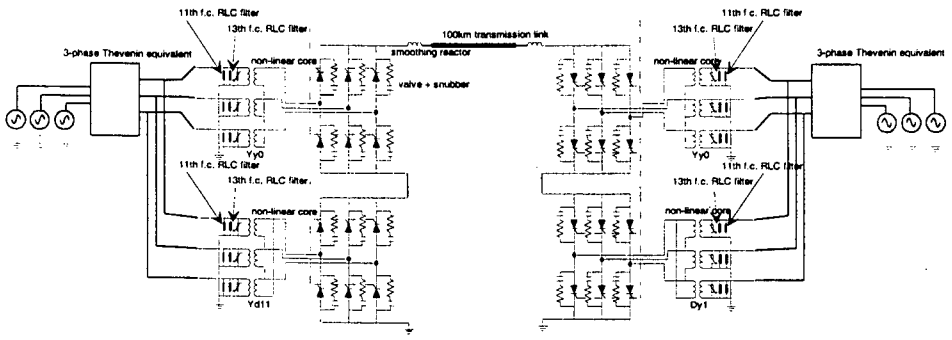


Fig. 1: An HVDC test case.

Using the trapezoidal integration rule as a reference, and keeping the maximum distortion error introduced by the rule under 10%, an integration time-step between $50\mu s$ and $100\mu s$ is then necessary [10].

To perceive the performance improvement needed, two important test cases were simulated with Microtran's EMTP[5]: the HVDC test case in Fig. 1, and the protective relays test case whose *one line diagram* is shown in Fig. 2. Both cases were run on a 200 MHz Pentium Pro workstation. For the first case, the EMTP used an average[6] of $3120\mu s$[7], that is, an improvement in speed of thirty to sixty times is necessary for real-time performance. For the second case, an improvement factor between seven and fourteen was found needed.

In spite of the strict speed requirements, the new breed of microchips and computer architectures has been attracting researchers [18, 20, 21, 22, 23, 19, 24, 25, 26, 9, 27] into trying to produce a digital real-time simulator. Most of those researchers have chosen a hardware approach. Some mimic the topology of the power network with a convenient arrangement of DSP's [18, 27] (some tried

[5] UBC's PC version of the EMTP.

[6] The critical step is —most likely— much larger than this, but internal probing into the EMTP's simulation cycle was not available.

[7] Actually, more than that, since capacitive snubbers and harmonic filters were not included in this simulation.

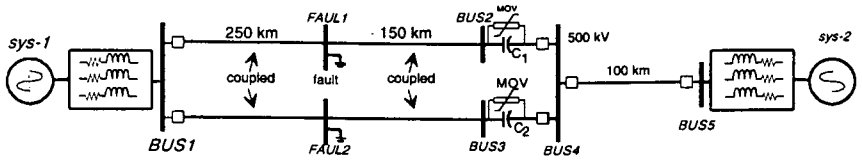


Fig. 2: A protective relay test case.

this approach with transputers in the past [20, 21]), others rely on expensive and sophisticated super-computer architectures [24] to meet real-time deadlines for reduced size test cases[8]. Some other researchers [22] have attempted a transient stability analysis of a power network by splitting the simulation loop into spawned parallel child processes, where each of these processes is assigned a node in a hypercube architecture system, according to a sophisticated mapping pattern. The results reported in [22] show a speedup of 45% when moving from one to two processors, but an additional gain in speed, for the linear part of the problem, with four processors of only 15%. If more than four processors were used, the additional overhead actually increased the total execution time. In hardware based solutions like those in [18, 20, 21], the close match between the particular network to be solved and the physical connection of boards (or transputers, in the past) may render the solution inflexible[9]. Besides, depending on customized hardware platforms, the upgrading cycle to new and faster hardware may be much slower than in the case of commercially available off-the-shelf computer systems.

In the work that occupies us, an algorithmic-software-based method is introduced. By going back to the original set of non-linear coupled partial differential equations, a global view is obtained. The increased level of complexity of

[8] Even though they originally employed supercomputers, the Mitsubishi group, with which we performed common work in 1995, has recently switched to a PC solution for the hardware [26].

[9] As of this writing, [27] implemented an elegant solution around this problem.

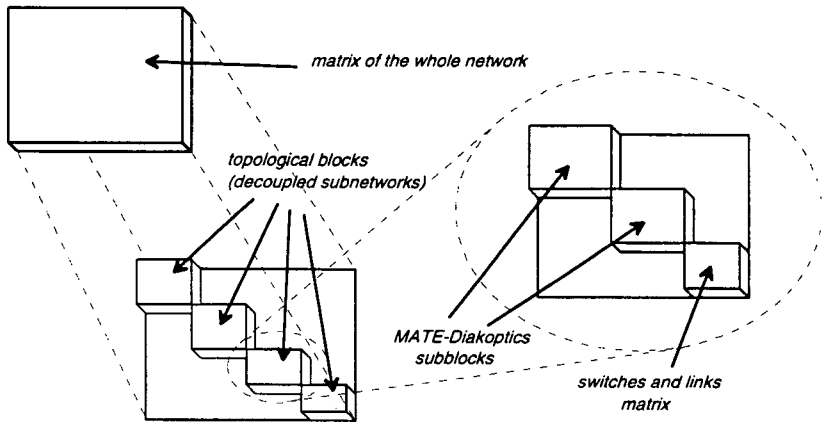


Fig. 3: Multilayer segmentation: Topological segmentation, followed by MATE-Diakoptics Segmentation.

the representation is dealt with by fragmenting the network into smaller quasi-decoupled fragments. Two fragmenting techniques are combined into a two step process, topological segmentation [9], and MATE segmentation [28, 29], Fig. 3. Further simplification and efficiency are achieved by hiding, or shading away, certain nodes, and so reducing the effective size of the network fragments even more. Also, as the smaller network fragments contain a reduced number of switches, ergo a reduced number of switching states, and contain fewer nodes, after node hiding, they become suitable for some judicious precalculation without loss of generality in the solution [29].

Apart from the speed-related issues of the solution algorithm, growth-security was also considered. The fast changing evolutionary process of real-time model development for network elements imposes the need to incorporate simplicity and flexibility in the interface between those element models and the integrator proper. That is, we need to plug-in and out new models as old ones become obsolete, as painlessly and reliably as possible. Some of those models may represent a centre of fast-changing topology to the integrator, as in the case of the HVDC model [30], or complex internal representations that must not perturb

the core of the simulator with their details, as in the case of the time domain frequency dependent transmission line model [31, 32]. Even models that include non electrical issues, such as the synchronous generator model [33], must be incorporated seamlessly and used within a common model-integrator-interface. This goal was achieved by object-oriented design techniques [34]. All models, present and future, are to be connected to the core through a common and unique interface, Fig. 4. This means that they all look and behave the same, as far as the core is concerned. In OOP[10] parlance, that common interface is provided by a "defined" generic element, (an abstract class named `elm_t`), that comprises all the behaviour groups[11] of interest to the integrator core. The result is that all models turn out to be a particular case of that abstract class, with the additions and refinements that are unique to the model in question: the models are classes that inherit the behaviour defined for the `elm_t` class.

The solution presented here relies on a fast solution algorithm, and has the advantages of enhanced flexibility and upgradability: it is not hardwired to the configuration of the network to be simulated, and its core (NI) is written in C++. The algorithm is easily portable to faster hardware platforms, as they become available, with the only concern in real-time applications of the adaptation of port-cards, amplifiers, and the corresponding synchronization signals. During the research cycle of this project, the core was developed on Intel platforms, run on Sun workstations (for portability tests), moved to IBM RISC System/6000 Model 560 machines, where it delivered real-time performance for the first time, with the first version of the integrator. More recently, the integrator was ported back to Intel machines of later vintage, workstations of the Pentium series, Pentium Pro 200 MHz, and lately to a Pentium II 400 MHz. That the simulator delivered real-time performance[12] on these inexpensive platforms is a sign of the efficiency of the underlying algorithm. The integrator is portable.

[10] Object Oriented Programming [34].

[11] In OOP parlance, *behaviour* of an object describes one of the routines that can be applied to the object.

[12] Within the target bandwidth and network size and configurations.

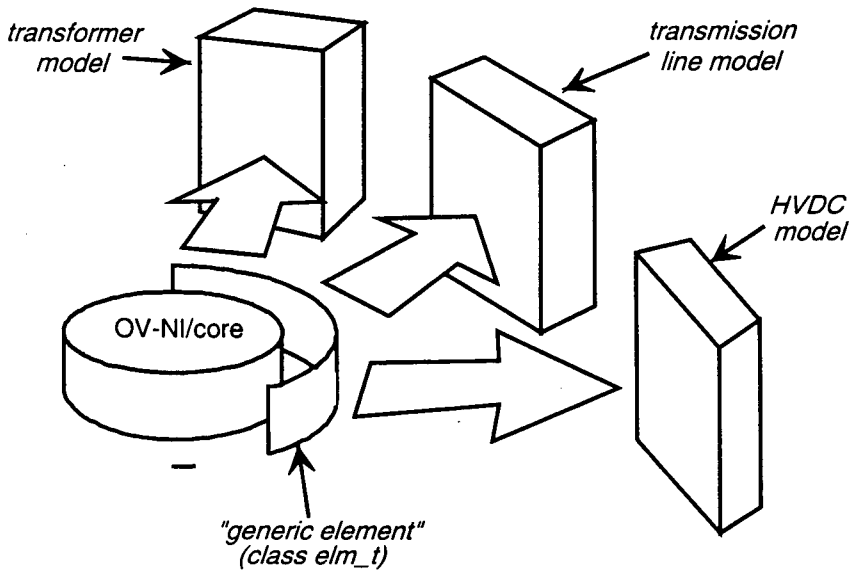


Fig. 4: All element models look and behave the same from the point of view of the simulator's core.

The design allowed the exploration of avenues for improved efficiency: latency [2], dependent on the relatively different time constants of different sections of the network; and backtracking, to cope with switching not produced at one of the time points of the simulation [35].

The integrator solution must respond as well to events generated at both interface ends, see Fig. 5. On the side of the user, OVNI interacts with OUI, OVNI's user interface [36, 37] (due either to configuration changes in the network, or to the connection or removal of probes, voltmeters, ammeters, oscilloscopes, etc.). On the hardware end, OVNI interacts with OV-XI [36, 38], the back-end hardware interface with the real world (opening or closing signals, or gate signals for controlled rectifier groups).

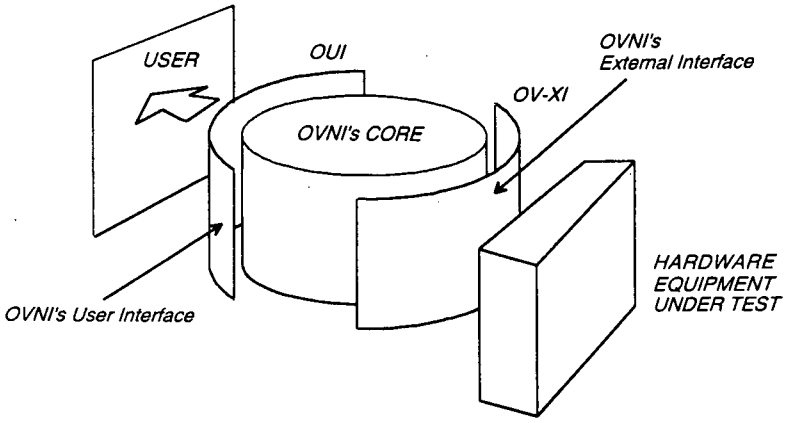


Fig. 5: Front end and back end interfaces to OVNI's core.

1.1 Research Claim and Contributions.

Our summarized claim is that "Real-time simulation of realistic power networks is possible using stock computer hardware." To demonstrate that claim, this thesis introduces as contributions: a) the use of backward Euler integration rule as a preferred method, and demonstrates its validity; b) a multi-layer segmentation scheme with: topological segmentation (introduced by lines time delay), an extended multi-area Thevenin equivalent concept segmentation, a node-hiding technique; c)a double-step double-interpolation technique to syncronize the simulation both with switching operations and with the real-time output stream with very low overhead.

Other central contribution was the implementation of the simulator around the OOP paradigm in C++, which is both efficient enough for real-time performance, and extensible to allow new models to be added without modifying the core. Also, a set of models was developed that prove the speed advantages of the proposed solution algorithm. The resulting simulator was tested on two real problems for real-time power networks simulation: a protective relay testing case, and an HVDC controller testing case.

Part II

THE PROBLEM

2. THE PROBLEM

2.1 Real-Time Simulations

Real-time simulations stem from a situation like the one depicted in Fig. 6. An observer interacts with a system. The observer perceives the behaviour of the system, sends controlling signals to it, and watches the system's response to those signals; all this in a continuous cycle.

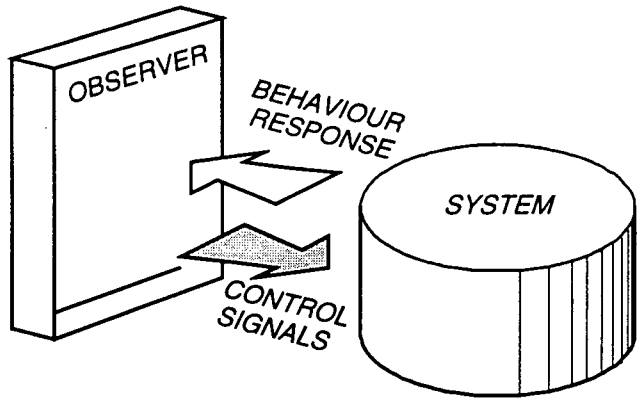


Fig. 6: An observer watching over and controlling a system.

The system could be an aeroplane, then the observer would be a pilot; or the system could be a power electric network, and the observer would be a protective relay, or an HVDC controller perhaps. In either case, if the purpose of the interaction is to evaluate the capability of the observer to perform under different circumstances, providing the observer with the real system (i.e., the aeroplane or the actual power system) is out of the question. The evaluating

agency presents the observer instead with a substitute system, a simulated one, where mistakes or malfunction will not result in an unthinkable catastrophe.

To produce a meaningful evaluation of the performance of the observer, the simulated system needs to make the observer believe that it is interacting with the real system. The simulator, the agent in charge of creating such an illusion, must receive the observer signals, process them, calculate and release the correct behaviour of the system; and do it all "fast enough" to create that illusion. Such is the task of a real-time simulator.

2.2 Digital Real-Time Simulations [1]

When the simulator is a digital one, by its own nature it cannot produce a continuous behavioural signal. Instead, the digital simulator issues a sequence of samples spaced "close enough" in time as not to miss any significant ripple in the behaviour of the system being simulated. It produces a discrete time simulation. Between the digital simulator and the observer stands a digital to analog converter and amplifying block, Fig. 7. This block fills in the gaps between the discrete samples produced by the digital simulator, and delivers a continuous time signal to the observer.

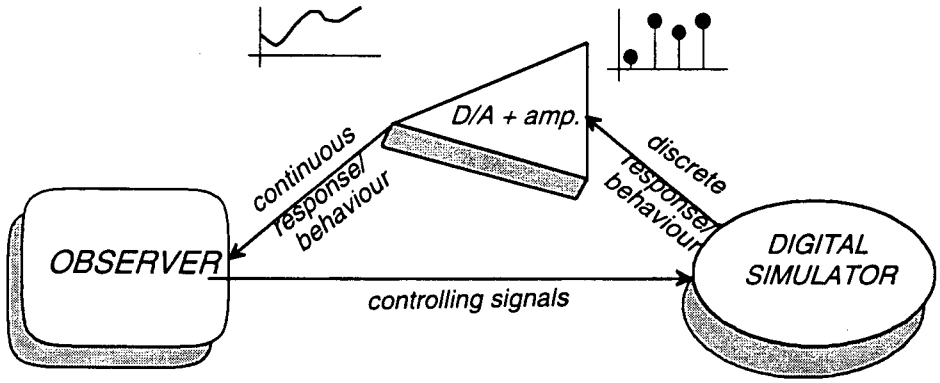


Fig. 7: From discrete to continuous, through D/A converters and amplifiers.

2.3 Frequency Bandwidth, Integration Rule, and Accuracy Limitations

The samples issued by the simulator must be close enough to one another as to include up to the highest frequency component of interest in the behaviour of the system. According to Nyquist sampling theorem [39], the relationship between the frequency of the fastest frequency component, the Nyquist frequency, f_{Ny}, and the time distance of the samples is such that at least two samples of each cycle of that component are present in the discrete signal. The time between two consecutive samples, the simulation step or integration step, Δt, relates to f_{Ny} according to Eq. (1).

$$\Delta t = \frac{1}{2 f_{Ny}} \tag{1}$$

The smaller the integration step Δt, the wider the bandwidth of the solution produced by the simulator, but the higher the performance requirements on the simulator. For a given integration step, Δt, the theoretical bandwidth of the simulation is given by the Nyquist frequency, f_{Ny}

$$f_{Ny} = \frac{1}{2\Delta t} \tag{2}$$

For an integration step $\Delta t = 50\mu s$, the theoretical bandwidth would be

$$f_{Ny} = \frac{1}{2 \times 50 \times 10^{-6}} = 10,000\text{hertz} \tag{3}$$

This bandwidth holds only if the samples are taken out through observation of the correct continuous signal. In the case of a digital simulator, the samples are produced by a painstaking numerical integration process of the equations that describe the system. The accuracy of the integration process depends on the integration rule utilized, and on the size of the integration step. The theoretical bandwidth suggested by Eq. (2) is drastically reduced by the distortion introduced by the integration rule. As will be seen in the next chapter, the

EMTP's trapezoidal rule introduces a magnitude distortion according to Fig. 8.

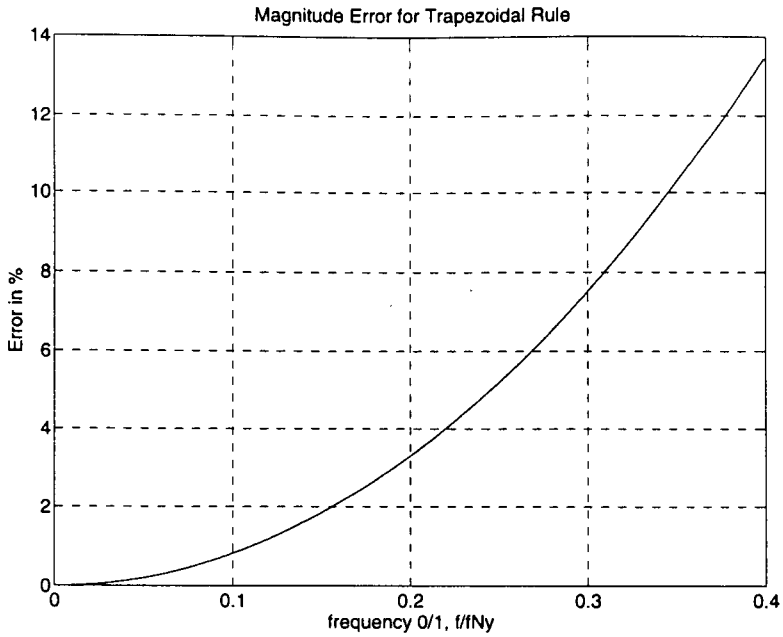


Fig. 8: Magnitude distortion introduced by Trapezoidal rule at frequencies up to 40% the Nyquist's.

In protective relay tests, also in HVDC controller tests, the range of frequencies of interest goes up to 2,000 Hz. Depending on the tolerated distortion, see Fig. 8 and Eq. (2), the integration step in those test cases should be no larger[1] than

$$\Delta t = 50\mu s \quad for \quad error \leq 3\%$$
$$\Delta t = 100\mu s \quad for \quad error \leq 10\%$$

These two results coincide with the recommendations in [17].

[1] If trapezoidal rule of integration is used.

2.4 Hard Real Time versus Soft Real Time Simulations

From what has been said so far, it is evident that the simulator is in a race to do all of its duties by the time deadline imposed by the frequency response desired and the integration rule applied. That is the real-time deadline. When the simulator fails to meet that deadline, the value of the simulation suffers. In some cases, the value of the simulation decreases with the extent by which the simulator failed to meet the real time deadline. In other cases, the value of the simulation is null if not produced within the deadline boundaries. The first kind is labelled soft real-time simulations; the second kind, hard real-time simulations [40] , see Fig. 9.

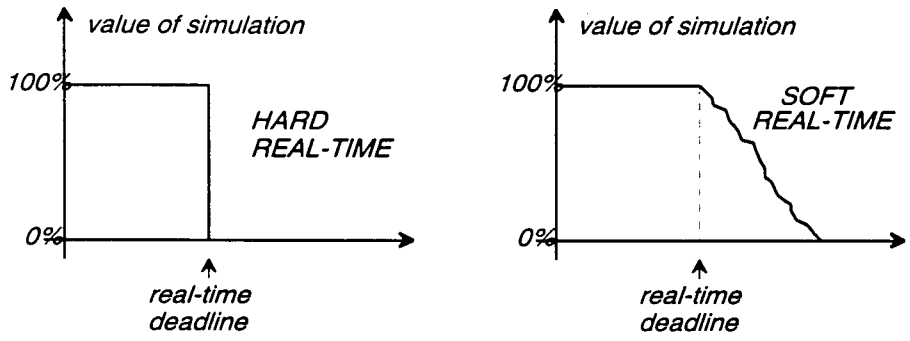


Fig. 9: Usefulness of the simulation results for: a) a hard real time simulation; b) a soft real time simulation.

For instance, when a real-time controller for a bread toaster misses its real-time deadline, it produces browner toasts, not quite the perfect one, but edible enough. The value of the simulation has been reduced, but some benefit can still be obtained from it; a sample of soft real-time simulation, Fig. 9b. On the other hand, when an auto-pilot landing real-time controller for aircrafts fails to meet its real-time deadlines, even if by a minor margin, the catastrophic results render the simulation completely invalid; this is a hard-time simulation indeed, Fig. 9a.

During the first years of the project, OVNI was considered a hard-real time

Fig. 10: Typical configuration of power networks used in protective relay testing.

simulator, however, recently, under the light shed by experience some consideration was given to whether it could be a soft-real-time simulator under certain conditions. In particular, when a critical event occurs between two output bursts (the ones at the ends of their corresponding integration steps), the distinction between outputting the correct value at the critical moment, or the extrapolated one at the proper time, was proven irrelevant in all of the test cases [35].

2.5 Network Size. Critical Complexity Network, CCN

The size of the power network to be simulated is given by the number of nodes and branches —one branch per lumped element or switch, $2n$ branches per n-phase transmission line in the discretized equivalent network [3, 4]. The computational effort necessary at each simulation step grows with the size of the network [41, 4].

In any real-time simulator, associated with a particular arrangement of hard-

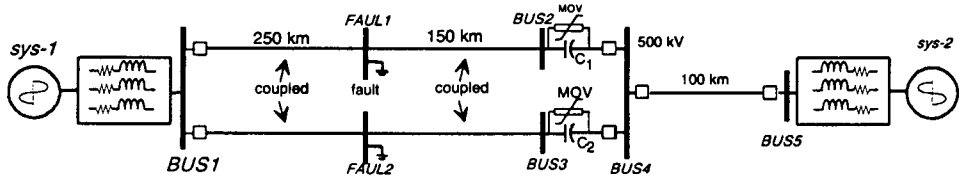


Fig. 11: Critical Complexity Network targeted for relay testing. It includes two multicircuit transmission links, and MOV protection.

ware and software, there is always a limit in the complexity of the network that can be simulated in real-time. That is defined as the *critical complexity network*, CCN. Even very crude solution algorithms are capable of real-time performance for three or four-node networks. OVNI's algorithm segmentation lends itself naturally to a multi-machine solution; i.e., a segment of the network is solved in a module, a workstation working in parallel with others in charge of different segments of the network. Thus, networks of arbitrary size can be simulated by adding additional workstations. The efficiency of the algorithm is normalized, in what follows, by the critical complexity network associated with a single module configuration, a single workstation.

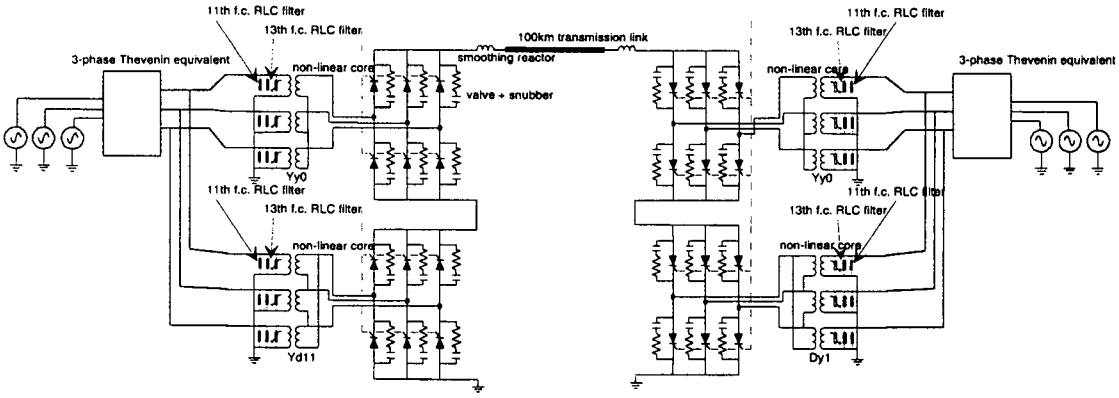


Fig. 12: Target Network for HVDC controllers testing. The controllers triggering the gate signals, used in tuning the simulator, are not shown, but were included in the simulation.

In its minimal hardware configuration, single module, two real-time test tasks have been targeted and explored for the present report; namely: protective relay testing, and HVDC controllers testing. For the first case, the critical complexity network must include sufficient detail to cover the relay's protection zone and the simulated fault or operating switches. The network outside that zone may be represented by compact multiphase coupled-impedance Thevenin's equivalent circuits [4, 33].

The compromise between performance and accuracy sketched in § 2.3 reduces the configuration of the network to be simulated to one like that in Fig. 10, which is similar to the reported test case in [18]. A more demanding test network, including multicircuit transmission links (capability included in OVNI's prototype) was used instead, the network shown in Fig. 11.

For HVDC controllers testing, the CCN includes two multiphase Thevenin equivalents for the surrounding AC-networks (one on the rectifier side, and another on the inverter side), a two-pole DC-transmission link, a 12-valve rectifying substation including the two corresponding three phase transformers, and a 12-valve inverter substation with its two three-phase transformers, Fig. 12.

Part III

THE SOLUTION

3. INTEGRATION RULES IN OVNI

3.1 Introduction

Digital simulation pivots around the integration rule chosen to solve the differential equations that describe the system being simulated. Ironically, this integration rule is also the weakest link in the entire simulation process [41].

Ever since the introduction of the EMTP[1] in the late sixties by Dommel [3], the trapezoidal integration rule became de facto the standard rule when it comes to digital solution of electric power networks [4]. That choice has been later substantiated and made more robust by the introduction of the Critical Damping Adjustment (CDA) by Martí and Lin [19] in the late eighties. Currently, the trapezoidal integration rule is tacitly accepted as the underlying platform under every attempt to achieve digital real-time simulation [18, 9, 17, 23, 42, 24, 25, 26, 27]. Even the ubiquitous fifty microseconds targeted deadline is but the consequence of:

- The needed 2 kHz bandwidth, associated with the tests described in the previous chapters.
- A tolerated magnitude distortion of 3%.
- The use of the trapezoidal integration rule.

In this chapter, several promising integration rules are examined, and OVNI's deviant choice is justified.

[1] The Electromagnetics Transients Program.

3.2 Accuracy and Stability

The validity and convenience of an integration rule in a real-time simulator is given by its accuracy, its stability, and its simplicity. In loose terms, the accuracy states how close the numerical solution, produced by the integration rule, is to the actual exact solution along all of its simulation or solution time span. The stability of a rule signals that the numerical solution will stay within a certain "distance" of the exact solution; that is, that it will not drift away eventually toward infinity. The simplicity of the rule has an impact on the overall performance of the simulation.

To evaluate the first aspect of an integration rule performance, even if both using different methods, [41] and [19] both recur to a differential equation whose exact solution is known: a first order one. In this thesis a different approach will be used to probe more deeply into the nature of the rules, but for the same reasons as those of the previous two authors, a first order system, the voltage/current relationship in an inductor is used. To normalize the results, a unit inductance was used (i.e., $L = 1\,henry$), Eq. (4).

$$v(t) = L\frac{di(t)}{dt} = \frac{di(t)}{dt} \tag{4}$$

To perceive and quantify the distortion introduced by an integration rule on a solution wave, the effect of the rule on frequency components ranging between DC and Nyquist's frequency is readily studied in the following sections. However, in order to gain a fresh insight into the behaviour of the rules under scrutiny, instead of finding the Z-domain transfer function corresponding to each rule, and mapping the z variable to the frequency domain, as in [19], a different approach is used in this chapter.

3.3 Frequency Response [2]

When the integration step, Δt, is kept fixed, as in our case, the bandwidth of the solution spans up to the Nyquist frequency, $f_{Ny} = \frac{1}{2\Delta t}$. To explore how

an integration rule responds to each frequency within this bandwidth a simple experiment was set up. A variable frequency sinusoidal voltage source is set to feed a one-henry inductor, $L = 1H$. At each frequency, the current wave, magnitude and phase shift, was obtained through the integration rule under scrutiny and compared with the actual exact phasor solution to the equation Eq. (4).

Actually, at each frequency, the effective admittance of the inductor, as rendered by the integration rule, is calculated; i.e., the quotient of the phasor representing the current wave obtained by the rule, and the phasor representing the voltage wave applied by the source. That admittance, $Y_e(\omega)$, can then be compared with the exact admittance of the inductor, $Y_x(\omega)$.

$$Y_x(\omega) = \frac{-j}{\omega L} \tag{5}$$

Then, the quotient $\frac{Y_e(\omega)}{Y_x(\omega)}$, a complex number, is plotted, in magnitude and angle along the spectrum up to the Nyquist's frequency. The closer that quotient stays to the real unit, magnitude one, phase zero, the more accurate the integration rule is.

The procedure is simple enough. However, as the frequencies get closer to Nyquist's, the reduced number of samples per cycle of the solution imposes an additional complication. A filter is used to extract and smooth out the sinusoidal wave corresponding to the particular frequency. The filter, described by Eq. (6), produces the value of the current at any point in time, t, even between the samples delivered by the integration rule, which are represented by the sequence[2] $i_d(k \cdot \Delta t)$ for $k = 0, 1, 2, \ldots$; where Δt is the sampling span or integration step.

[2] In our case, the sum spans up to the last sample produced by the simulation —a finite sum, but the infinite span was kept in Eq. (6) above as an indication of the distortion incurred when handling a finite number of samples—, and the cycle calculated by filtering was chosen as close as possible to the centre of the sequence.

$$i(t) = \sum_{k=0}^{\infty} i_d(k.\Delta t) \frac{sin[(\pi/\Delta t)(t - k.\Delta t)]}{(\pi/\Delta t)(t - k.\Delta t)} \tag{6}$$

The distorted admittance (with respect to the exact one), in magnitude and phase, as produced by each of the five studied integration rules (trapezoidal, Simpson's, Gear's second order, backward Euler's, Calviño's second order [38]) is shown in Figs. 13 and 14.

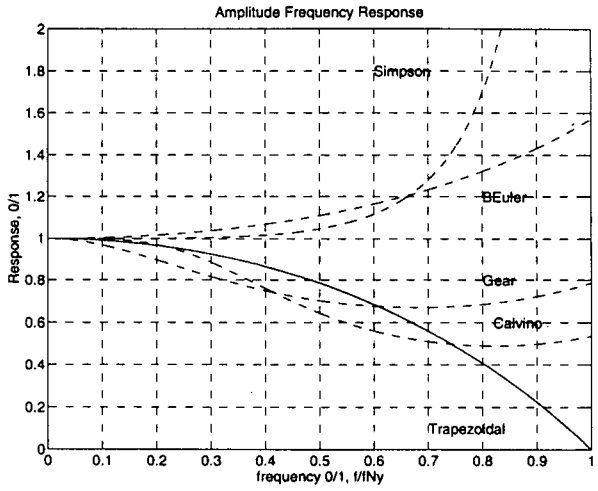


Fig. 13: Frequency response, magnitude, for the rules: trapezoidal, backward Euler's, Simpson's, Gear's second order, Calviño's second order.

3.4 Choosing OVNI's rule

From the response depicted in Figs. 13 and 14, Simpson's rule, with zero phase shift and the closest to unity magnitude response, seems to be the best choice. Its magnitude response, Fig. 13, drifts toward infinity, however, at frequencies close to Nyquist's. In other words, Simpson's rule although apparently accurate, is unstable. This last statement needs to be bounded. Simpson's needs very small[3] integration steps to remain stable (This, evidently, brings the significant high

[3] As compared with the steps used by the other rules.

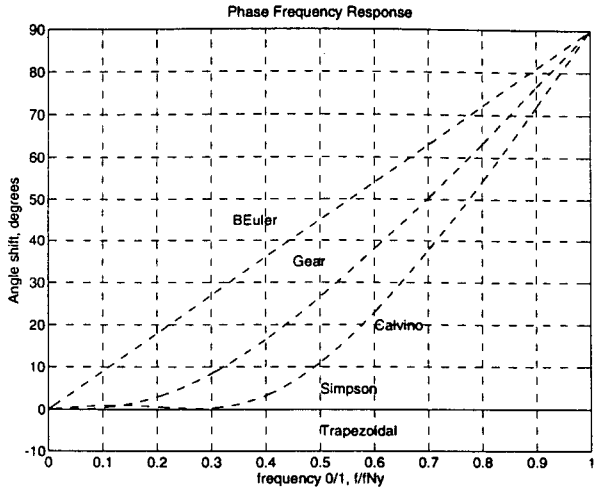


Fig. 14: Frequency response, phase shift, for the rules: trapezoidal, backward Euler's, Simpson's, Gear's second order, Calviño's second order.

frequency components of the signal into the "finite" response region of the rule). In the single phase power system used to compare performance of the rules in §3.6, Simpson's rule needed up to ten times smaller integration steps than the other rules in order not to go ballistic. Simpson's is thus disqualified.

Gear's second order rule, while maintaining a more even magnitude response along the spectrum, introduces a relatively large magnitude distortion at the most important low-frequency range, fig. 13. Even worse, Gear's rule shifts the different frequency components by a different amount along the time axis, see Fig. 14, distorting thus the shape of the wave, and smoothing out abrupt changes, or even creating false spikes on its own; see §3.6. Gear's rule is also eliminated.

Three rules remain to be reviewed: Calviño's second order, trapezoidal, and Backward Euler's.

3.4.1 Calviño's second order rule

This rule merits a separate section. It combines a closer to ideal response for the most important low frequency components, both in magnitude, and in phase shift. Overall, however, it reaches the 3% error at about the same frequency as Trapezoidal, and so it imposes the same integration step on the simulator for this accuracy limit. At the same time, the rule raises the computational burden of the simulator by up to three times, increasing effectively the computation time per step. In a case like the one in Fig. 11, where the rule's computational costs accounts for 15% of the total integration time per cycle using trapezoidal rule, the percentage used by Calviño's rule —assuming that the rest of the simulation process remains unaffected— could reach 35% of the cycle. And the whole cycle would then consume 30% more time.

The non zero phase shift of this rule endows it with power loss characteristics and improved stability, same as Gear's, or B.E. However, its non straight line phase characteristic penalizes it with the same distortion as Gear's when the integration step is not kept reasonably small: uneven frequency component shifting along the time axis.

3.4.2 Trapezoidal versus backward Euler's

Trapezoidal and backward Euler's rules show a different strength each: trapezoidal has an ideal zero phase shift, but the magnitude response of backward Euler's is significatively better, as witnessed by the relative magnitude error plot in Fig. 15.

Now we turn our attention to the weaknesses of those two rules. For the trapezoidal rule, take the voltage/current equation for an inductor, Eq. (4), and integrate it along the interval $(t - \Delta t, t)$, Eq. (7)

$$\int_{t-\Delta t}^{t} v(t)\,\mathrm{d}t = L \int_{t-\Delta t}^{t} \mathrm{d}i \tag{7}$$

The right hand side is an exact definite integral, after approximating the left

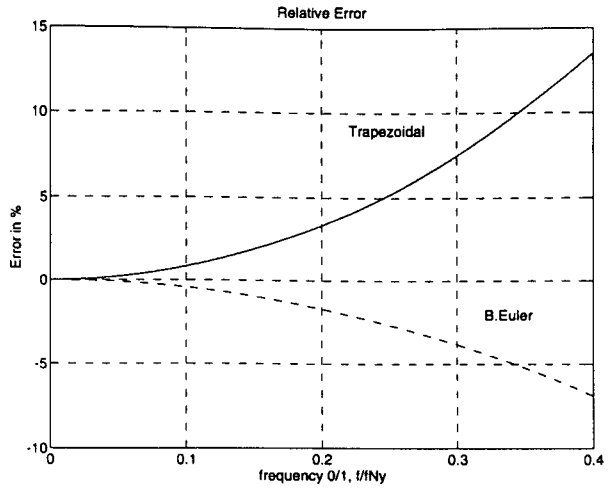


Fig. 15: Error in magnitude introduced by trapezoidal and Backward Euler's rule, up to 40% of the Nyquist's frequency.

hand side with a trapezoidal rule area [3].

$$\frac{v(t) + v(t - \Delta t)}{2} \Delta t = L\, i(t) - L\, i(t - \Delta t) \tag{8}$$

Assuming that the Z-transform of voltage $v(t)$ and current $i(t)$ in the inductor are $V(z)$ and $I(z)$ respectively, Eq. (8) can be written in the Z-transform domain [39].

$$V(z)[1 + z^{-1}] = \frac{2L}{\Delta t} I(z)\,[1 - z^{-1}] \tag{9}$$

The impedance transfer function in the Z-domain is

$$Z(z) = \frac{V(z)}{I(z)} = \frac{2L}{\Delta t} \frac{z - 1}{z + 1} \tag{10}$$

with a pole at $p = -1$. But that pole, once inserted into the natural or transient discrete time response of Eq. (11), shows an oscillatory never decaying response. This phenomenon was identified by [43, 19] and labelled critical unstability.

$$n(k) = C.p^k \quad fork = 0, 1, 2, ... \tag{11}$$

Martí and Lin [19] identified the problem and buttressed the trapezoidal rule with an intelligent switching to the more stable Backward Euler's, albeit only when those critical undampened oscillations were detected, and only for two integration steps. That procedure, Critical Damping Adjustment (CDA), is the standard arrangement in EMTP solutions today. The price for the added stability, however, is too high for real-time simulations, the integration step where CDA is found necessary incurs in twice as many computations as a regular step. So, it seems CDA is out of the question in our quest.

Finally, we consider the Backward Euler's rule apparent liability, its non zero phase shift response, Fig. 14. Traditionally, that phase response has been associated with the same sort of distortion produced by Gear's rule; i.e., wave shape distortion produced by uneven "lateral" displacement of the wave's frequency components along the time axis. In what follows, we will see that that assertion is not quite correct.

Figure 14 shows that B.E. shifts each frequency component by a different angle, but let us look more carefully into it.

The component at frequency f, according to Fig. 14, is shifted by an angle θ given by Eq. (12), but this θ phase shift is displacing the component a certain amount of time to the right on the time axis; i.e., the component is being "delayed" by δ seconds. To translate θ into δ, it is necessary to keep in mind that the time span equivalent to one degree in a fundamental frequency component corresponds to three degrees in a triple frequency component, and so on with higher frequency components, as in Eq. (13).

$$\theta = \frac{f}{f_{Ny}} 90^o \tag{12}$$

$$\delta = \frac{f}{f_{Ny}} \cdot 90^o \cdot \frac{T}{360^o} = \frac{1}{4\, f_{Ny}} = \frac{\Delta t}{2} \tag{13}$$

In other words, every frequency component is shifted the same amount of time, $\delta = \Delta t/2$. Conclusion: The wave shape should not suffer on the account of the phase shift alone. The effect is only to delay the wave by one half the integration step. To validate experimentally this conclusion, and using the same circuit arrangement as for the frequency response, each frequency component actual time delay was determined and plotted in Fig. 16 for an integration step $\Delta t = 50\,\mu s$. The deviations from the predicted $25\,\mu s$ stems from the 2,880 samples per cycle used to represent every frequency component.

The difference between backward Euler and Gear's rule is that the latter's phase characteristic is not straight line crossing zero at DC. Every rule with a "curved" phase response will suffer the same distortion penalty as Gear's. That is not the case with Backward Euler's Rule.

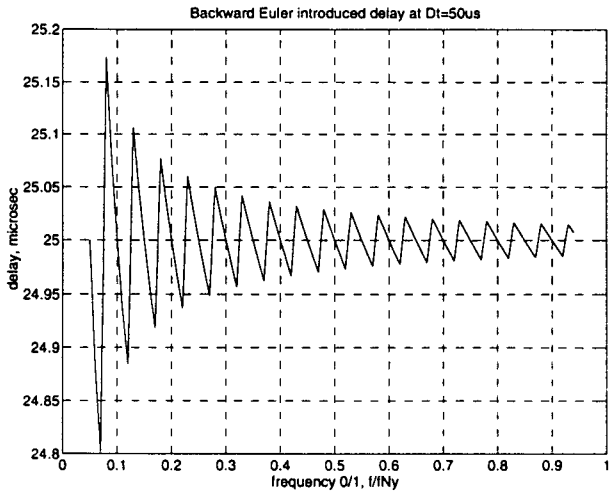


Fig. 16: Time delay introduced by backward Euler's rule at each frequency up to Nyquist's.

There are still, however, a positive and a negative side to the non zero phase response of B.E.

3.4.3 Backward Euler's, a "lossy" rule

An important implication of backward Euler's phase response, Fig. 14 is that the equivalent admittance of the inductor, as represented by the rule, has not only an imaginary part, but a real part as well. The rule represents the inductor by an inductor in parallel to a resistor. In other words, the rule's representation of the inductor (and of the capacitor too) incurrs in active power losses not present in the actual circuit. It may be said [39] that that power drainage is responsible for the absolute stability of this rule: it dampens out an otherwise never dying oscillation. But those power losses also subdue somehow the different frequency components of the wave.

It can be shown [39] that the trapezoidal rule represents an inductor L, by an equivalent approximate inductor L_e whose value depends on the frequency

$$L_e(\omega) = L \cdot \frac{\tan(\omega\,\Delta t/2)}{\omega\,\Delta t/2} \tag{14}$$

Along the same lines, backward Euler's rule represents the same inductor L by a parallel arrangement of an equivalent inductor L_e with the same value as the approximate inductor introduced by the trapezoidal rule above, in Eq. (14), but with the addition of a parallel resistor R_e with the value

$$R_e = \frac{2L}{\Delta t} \tag{15}$$

Observe that the parallel resistor value in Eq. (15) does not depend on the frequency of the signal, the way the equivalent inductance L_e in Eq. (14) does. It follows that the rule drains from each frequency component a power that is proportional to the square of the amplitude of the voltage component; i.e., "smaller" components get less damped that "larger" ones.

It is this lossy characteristic that is responsible for reducing the "spikiness" of the response, not the phase shifting. The end result is absolute stability and reduced numerical spikes. The physical spikes in the response do not get masked, or shifted, the way Gear's rule does, they are slightly damped by the

lossy characteristic of the rule. See the simulations in § 3.6.

3.5 Improved performance of OVNI and backward Euler's

Summarizing, backward Euler's rule is more stable and its magnitude response is more accurate than the contender's. If the $\frac{\Delta t}{2}$ time delay is tolerable, which was the case in the simulations run to validate OVNI, backward Euler's rule improves the overall performance of the simulator like this:

- The number of floating point operations necessary to update some sources[4] becomes null, considering that such updating accounted for 15% of the simulation time during each integration step for the case in Fig. 2, the benchmark case I, the performance improvement in this updating stage is

$$\frac{n_C}{n_C + n_L} \tag{16}$$

  where n_C is the number of capacitors, and n_L the number of inductors in the circuit.

- More importantly, as the 3% magnitude distortion barrier is reached by backward Euler's at a frequency 50% higher than the one at which trapezoidal reaches 3% error, Fig. 15, the integration step can be 50% larger for the same amplitude distortion, stretching thus the real-time deadline and the performance requirements of the final algorithm. This produces the equivalent effect of a fifty percent performance improvement on the original algorithm.

3.6 A single-phase power system test case

To observe each of the rules at work, the simple single-phase reduced power network in Fig. 17 was solved separately with each of the rules. The simulation proceeded at[5] $\Delta t = 50\,\mu s$, up to $40\,ms$. At $t = 17\,ms$ the receiving end

[4] Associated with capacitive elements. See chapter on models, 4.
[5] For all the rules, except Simpson's.

of the transmission line is shorted by the switch illustrated. As a reference, the simulations produced by the EMTP with critical damping adjustment, for $50\,\mu s$ and for $\Delta t = 70\,\mu s$ are included in Figs. 18 and 19. Those two figures show the voltage at nodes *flag* and *glen*.[6] It is interesting to observe how the EMTP+CDA solution deteriorates when changing the integration step from 50 to 70 μs, Figs. 18 and 19, which does not happen so drastically, as expected, for the plain backward Euler solution, Figs. 22 and 24.

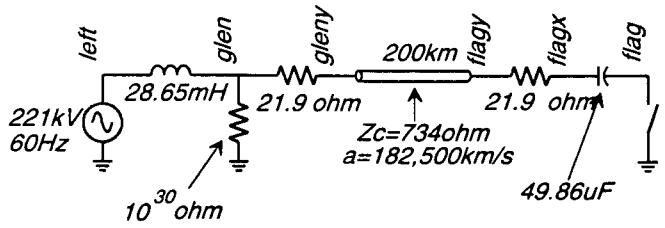


Fig. 17: Single phase power system with a short circuit on the receiving end, to test the different integration rules.

Simpson's rule proved unstable when $\Delta t = 50\,\mu s$ was used. It was necessary to reduce the integration step to five microseconds to obtain stable results, at least up to $40\,ms$.

The results obtained with each of the four rules are presented in the order from higher "spikiness" to lower: Simpson's, trapezoidal, backward Euler's, Gear's. The curves in Figs. 20 up to 24 illustrate the material exposed in the first part of this chapter. From Simpson's extreme unstability and spikiness, to Gear's phase distortion smoothing of the actual physical spikes (and introduction of false ones).

Finally, in Fig. 24, to illustrate the increased magnitude accuracy of B.E., the system is solved at $\Delta t = 70\,\mu s$. Compare those results with the ones in Fig. 22.

[6] Voltages for the same nodes (with no additional labelling) are presented for the other rules in Figs. 20 to 24.

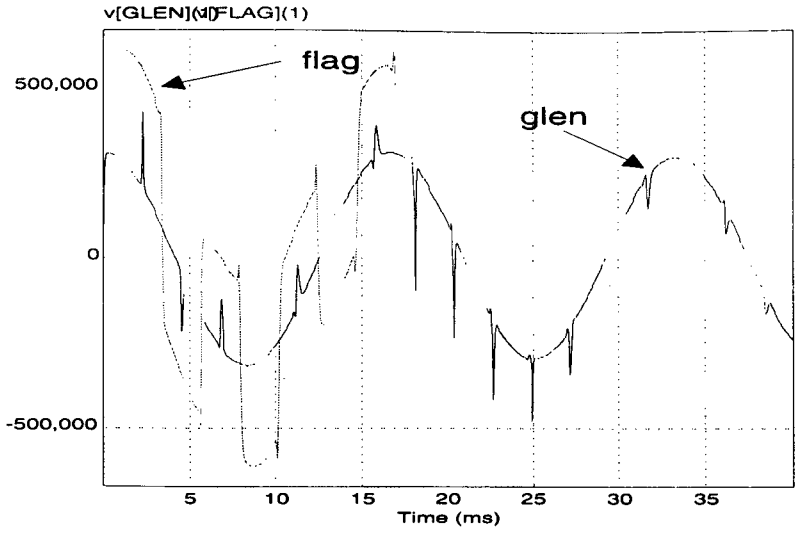


Fig. 18: Solution obtained by the EMTP with the CDA option activated with an integration step $\Delta t = 50\ \mu s$.

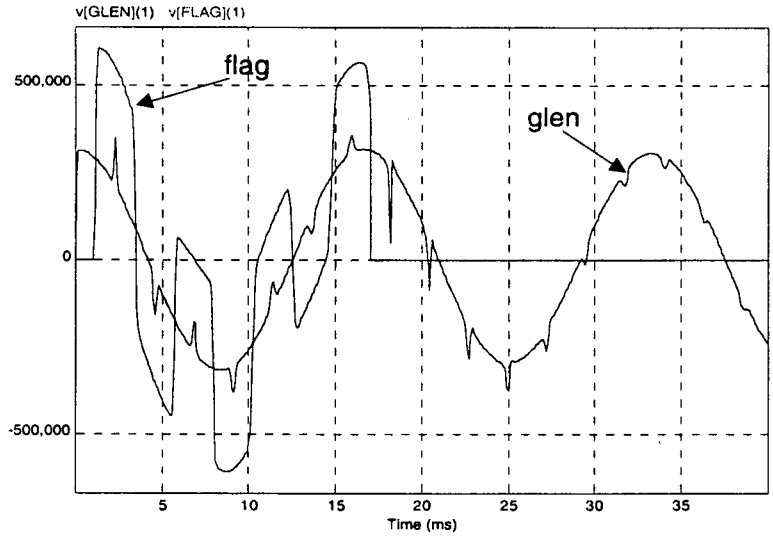


Fig. 19: Solution obtained by the EMTP with the CDA option activated with an integration step $\Delta t = 70\ \mu s$.

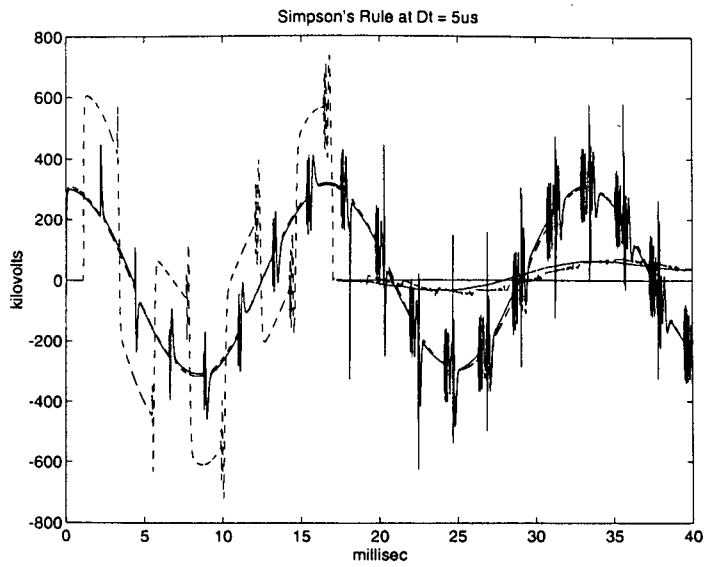


Fig. 20: Simpson's rule solution with $\Delta t = 5\mu s$. Voltages at all the nodes in the network in Fig. 17

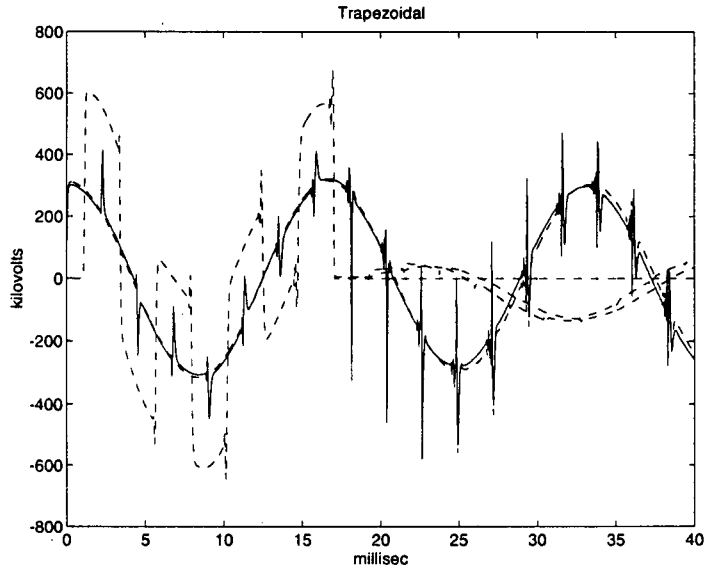


Fig. 21: Trapezoidal rule solution with $\Delta t = 50\mu s$. Voltages at all the nodes in the network in Fig. 17

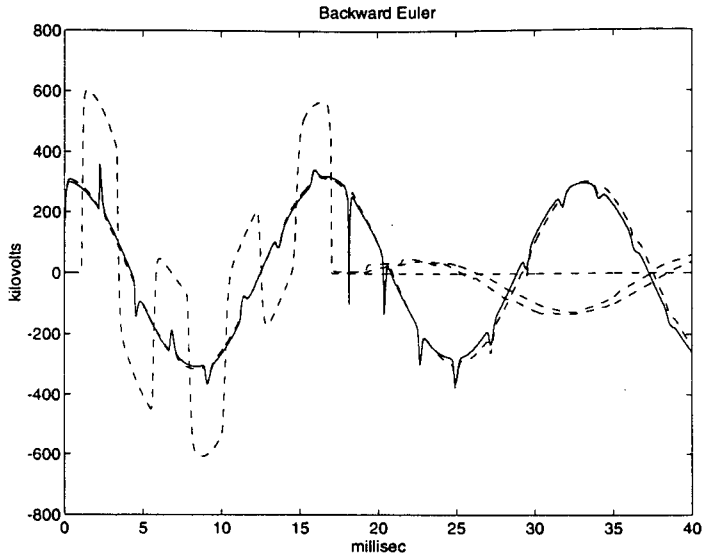


Fig. 22: Backward Euler's rule solution with $\Delta t = 50\mu s$. Voltages at all the nodes in the network in Fig. 17

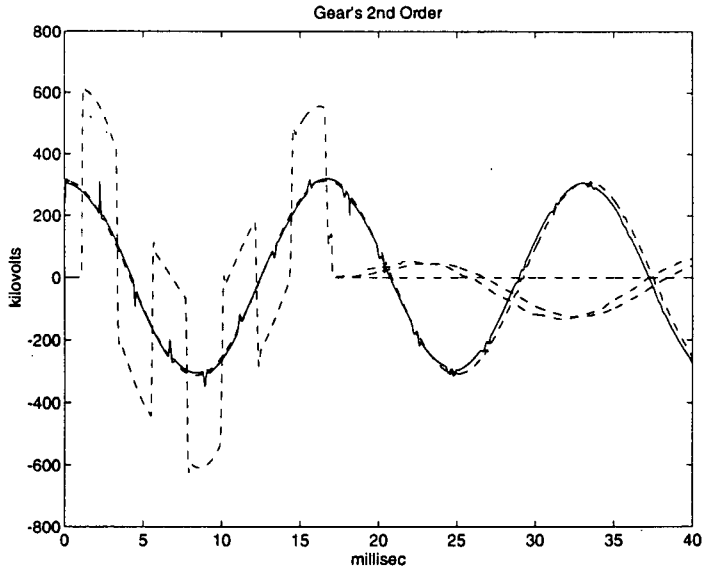


Fig. 23: Gear's second order rule solution with $\Delta t = 50\mu s$. Voltages at all the nodes in the network in Fig. 17.

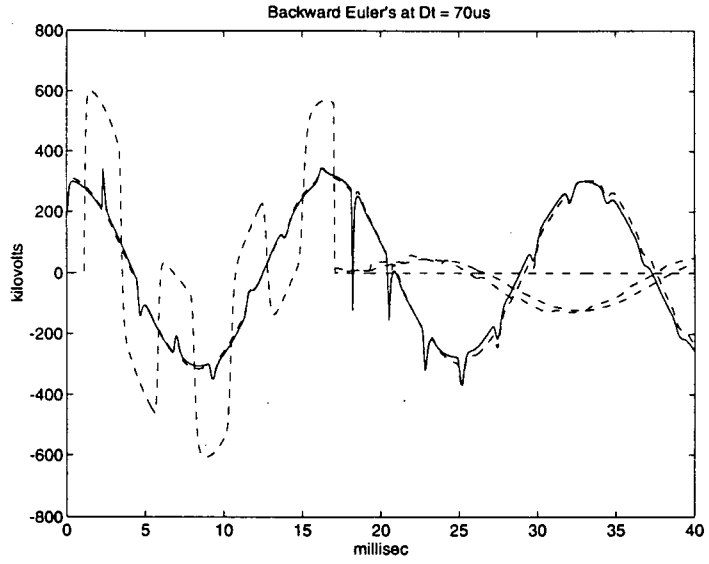


Fig. 24: Backward Euler's rule solution at an expanded integration step. $\Delta t = 70\ \mu s$.

4. DIGITAL SOLUTION, ELEMENT MODELS

4.1 Solution versus Simulation

In the context of this thesis, solution of a network is a one-time-point issue,[1] and simulation of the same network is an effort related to a sequence of time-points.

To solve an electric[2] network is to establish by whatever means, empirical or computational, the voltages in the nodes of the network, and the currents in the branches between those nodes. In a work of the nature that occupies us here, the method will be, of course, computational.

To simulate an electric network is to solve the network along a segment of the time axis. This process, by force of the tools selected, is discrete[3] in nature. To simulate the network is then reduced to solving it at a certain convenient number of points along the time interval of interest.

4.2 General purpose ODE-solvers

Once digital simulation is agreed upon (versus TNAs analog one), the process becomes one of solving the differential equation set that represents the behaviour of the network along the corresponding time interval. It is at this point where a question arises naturally, whether a regular all-purpose differential-equation-solver could do the job. After considering the tedious and error prone task of putting together the tens, hundreds, or perhaps thousands, of equations,

[1] The author is familiar with the standard use of 'solution' as applied to the closed form time expression of voltages and currents in the network, when such a closed form analytical expression is obtainable. In this work, however, solution stands for *time-point solution*, which lends itself to the discrete-time nature of the process to be implemented.

[2] In this report, no difference is made between electric and electronic circuits.

[3] As opposed to *continuous simulation*, like the ones obtained with TNAs, analog simulators.

and feed those equations to a differential-equation-solver, the answer begins to outline itself. Furthermore, a minor modification in the network could imply a major change in the set of equations. When, even after surmounting the equation building difficulty, a generic DE-solution algorithm was put to the test, simulation times obtained (even for moderate-size systems) fell well behind those delivered by the EMTP, and abysmally far behind from the deadlines imposed by real-time simulation under the bandwidth targeted. This result should come as surprise to nobody, considering the task at hand.

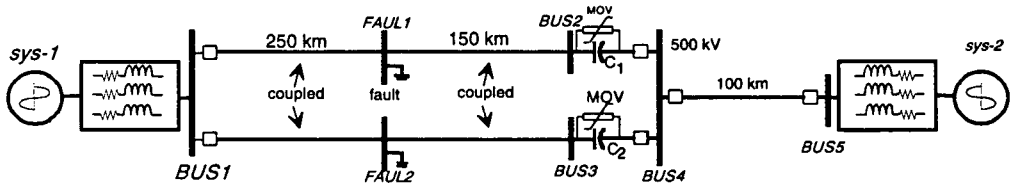


Fig. 25: A test case for relay testing.

As a sample of the performance aimed at, one of the benchmark test cases for OVNI, the target network for protective relay tests, Fig. 25, represents solving a set of forty differential equations coupled in forty unknown voltages, and doing so in less that thirty microseconds[4], a very demanding task indeed.

4.3 Discretizing the Network, not the Equations

Granted the need for a completely new tool, one faces the sometimes formidable job of putting together the set of coupled differential equations that describe the electric network (same as for a generic DE-solver), and then discretizing the equations through some convenient integration rule, according to what was said in the last chapter. The inversion of those two steps, starting point of OVNIs algorithm, was introduced by Prof. Dommel [3]. First, discretize each

[4] To allow for the necessary hardware communication overhead.

element, or rather its voltage-current characteristic equation. Then, represent the discretized v-i equation by a convenient array of resistors and sources. And finally, assemble the network using those discrete-time models for each of the original elements, see Fig. 26.

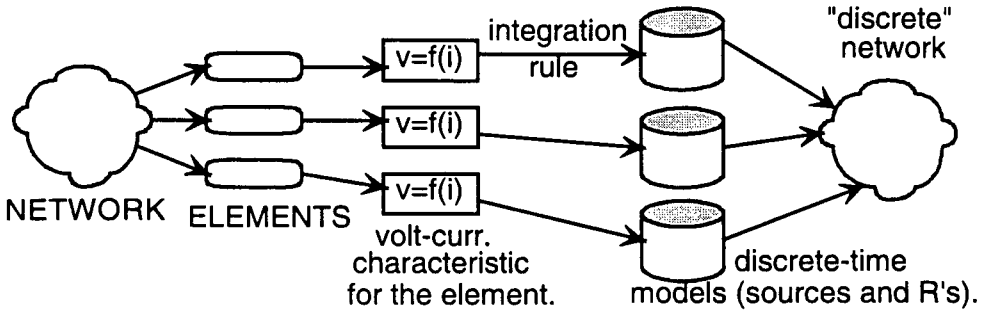


Fig. 26: Discretization process.

The resulting discretized network contains only resistances and sources, regardless of the original nature of the elements. Putting together the discretized network, and building the corresponding network equations becomes an issue of elegant simplicity.

The manner in which an element model is developed is outlined in the following section for one element, as an example.

4.4 Discrete-time model for an Element [3]

The discussion [3] in this section (§ 4.4) belongs in an appendix, but, given the flow of ideas in this discussion, it was included in-line with the rest of the text for the convenience of the reader. To illustrate what was just said in the previous section, let us consider an element, a linear inductor, Fig. 27a, and put it through the process outlined above, Fig. 26.

The v-i equation for that inductor, relates the voltage across the inductor, $v(t)$, with the current through it, $i(t)$, as in Eq. (17). Integrating both sides of this last equation along the time interval between $t - \Delta t$ and t, we obtain

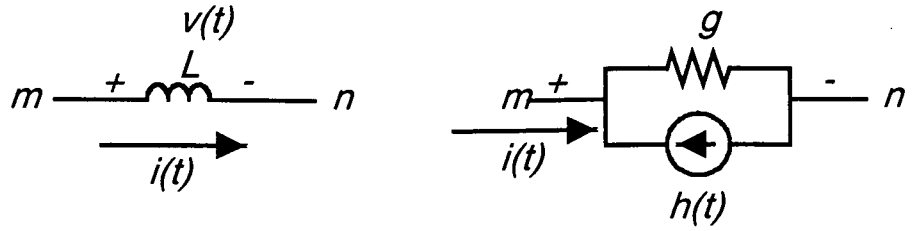


Fig. 27: a) Lumped inductor, and b) its discrete time domain model corresponding to the trapezoidal integration rule.

Eq. (18).

$$v(t) = L\,\frac{di(t)}{dt} \tag{17}$$

$$\int_{t-\Delta t}^{t} v(t)\mathrm{d}t = L.i(t) - L \cdot i(t-\Delta t) \tag{18}$$

If a numeric integration rule, let us say backward Euler's, is used to approximate the left hand side, and reorganizing the terms, the current in the inductor at the end of the interval, $i(t)$, appears as a function of the voltage across the inductor at the same point in time, $v(t)$, and the values corresponding to the initial point of the time interval $(t-\Delta t)$, that is, to the history of the inductor, Eqs. (19) and (20)..

$$i(t) = \frac{\Delta t}{L} \cdot v(t) + i(t-\Delta t) \tag{19}$$

$$i(t) = g \cdot v(t) - h(t) \tag{20}$$

But this last equation describes the current in the inductor as the sum of a current proportional to the voltage in the inductor with a historic current, or in circuit form, that the inductor behaves like a resistor with a conductance of $\Delta t/L$ in parallel to a current source, $h(t)$, that depends on historic values: a history source, Fig. 27. Granted the relative compactness of the discretization and modelling process for the linear inductor, doing the same for some network components have proved to be tasks challenging enough as to be the central

topic of doctoral theses in their own right [44]. OVNI is expected to grow that way, along the years, but to furbish its two probing tasks of equipment testing (protective relays and controllers for high-voltage-direct-current converters) a basic set of streamlined discrete-time models was included, as reported in the next sections, and in chapters 9, 10, and 11. Two groups of elements, lumped and transmission lines, have been borrowed, adapted, streamlined and optimized to take advantage of OVNI's architecture, from the ubiquitous EMTP, all other elements have been developed specifically for OVNI.

4.5 Basic models in the prototype

Once the main tasks of the simulator are introduced in the next chapter, it becomes evident, in the case illustrated in Fig. 25, that element models are responsible for 14.7% of the total time of simulation. OVNIs core stands on its own, separate from the element models. However, to test the core for its benefits and liabilities, it was necessary to furnish that core with a few basic element models, namely: resistors, inductors, capacitors (both linear and non-linear ones), single-phase and multi-phase transmission lines (both models differ significantly, the first one is not a particular case of the latter), metal-oxide-varistors (MOV), single-phase transformer units (including the modelling of their core saturation), three-phase transformer units (with modelling of saturation produced by zero sequence set of magnetomotive forces), HVDC rectifying modules and their corresponding firing angle controllers, and switching operations[5].

Given its final use, those basic models are streamlined and optimized in its execution to deliver maximum performance. Implementation of those models within the frame of high-*pluggability*[6], for OVNI, is an issue in itself described in detail in a later chapter.

[5] Switching operations are an intrinsic function of the core.

[6] The convenient removal of an obsolete model, and substitution with a better —more accurate or faster— one.

4.5.1 On Notation

In what follows the element under study is connected between nodes k —initial node— and m —final node—. Time advances in discrete steps of fixed and predetermined size Δt. At the end of the current step, $v(t)$ is the voltage across the element and $i(t)$ the current through it. The values $v(t-\Delta t)$ and $i(t-\Delta t)$ correspond to the end of the previous step (which are known, of course). The models listed in the following sections correspond to the backward Euler's integration rule. Expressions corresponding to the trapezoidal rule are obtained readily applying its approximation to the integral on the left-hand side of Eq. (18).

4.5.2 Lumped Elements [3, 4]

The resistor is simply represented by a resistance between k and m. The linear inductor and the linear capacitor are modelled by the equivalent circuit in Fig. 27b. The parameters g (equivalent discrete conductance), and $h(t)$ (history current source) in that figure are given, for backward Euler rule, by Eqs. (21) for the inductor.

$$\begin{aligned} g_L &= \frac{\Delta t}{L} \\ h_L(t) &= -i(t-\Delta t) \end{aligned} \tag{21}$$

But from Fig. 27b, the history term $h(t)$ can be written in terms of the inductor voltage at the previous time step, as $h(t) = h(t-\Delta t) - g \cdot v(t-\Delta t)$.

For the capacitor, the same model in Fig. 27b is obtained, but with the parameters given by Eqs. (22).

$$\begin{aligned} g_C &= \frac{C}{\Delta t} \\ h_C(t) &= \frac{C}{\Delta t} \cdot v(t-\Delta t) \end{aligned} \tag{22}$$

From Fig. 27b and equations (21) and (22) it is seen that the behaviour of the model depends on the element state at the previous step: its history, $h(t)$.

The equivalent conductances corresponding to the trapezoidal rule are given in Eq. (23).

$$\begin{aligned} g_L &= \frac{\Delta t}{2L} \\ g_C &= \frac{2C}{\Delta t} \end{aligned} \tag{23}$$

The history values corresponding to the trapezoidal rule of integration (rule which is used in the DSDI procedure, to be introduced in chapter 7) were simplified by Martíin [45] as

$$h_L(t) = h_L(t - \Delta t) - 2g_L \cdot v(t - \Delta t) \tag{24}$$

$$h_C(t) = 2g_C \cdot v(t - \Delta t) - h_C(t - \Delta t) \tag{25}$$

To update the history source value $h_s(t)$, with this simplified model, it is only necessary to keep track of the previous value of the source $h_s(t - \Delta t)$, since the voltage across the element is to be calculated at each time step anyway.

4.5.3 Transmission Lines [3, 5]

Transmission lines and their models are at the centre of OVNI strategies to exploit the network sparsity, as will be seen in chapter 5. Given the distributed parameter characteristic of the power transmission line, modelling it is not as straightforward as for lumped L and C elements. In what follows, it is assumed that the line parameters are independent of frequency, a necessary compromise between performance and accuracy adequate for a large number of applications [46].

It will also be assumed that both inductance and capacitance are uniformly distributed along the line [47]. The per metre values for those parameters are: L, in H/m; C, in F/m. Shunt conductance is assumed negligible and series resistance is treated as lumped into two loss-equivalent resistances at each end of the line. Each of those resistances is equal to one half the total series resistance of the line. If R, Ω/m, is the series resistance per metre of the line, and l the

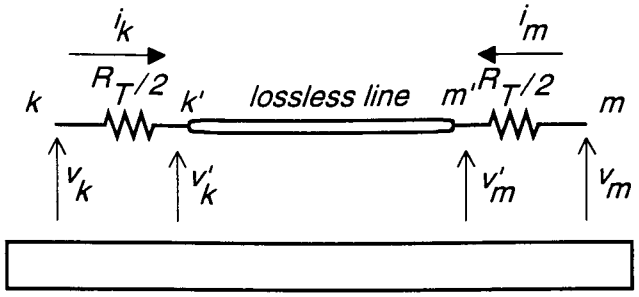


Fig. 28: Lumped losses in the transmission line model.

total length in metres, the total series resistance is $R_T = R \cdot l$. The previous simplification leaves a lossless transmission line surrounded by two $R_T/2$ resistors as in Fig. 28.

4.5.3.1 Lossless Single Phase Transmission Line Model

Dommel demonstrated [3] that the single phase lossless transmission line in Fig. 29 can be represented by the equivalent circuit in Fig. 30, where $Z_c = \sqrt{L/C}$ is the surge or characteristic impedance of the line, in Ω. History sources $h_m(t)$ and $h_k(t)$ depend on the voltage and current at the other end of the line τ seconds before [3] —$\tau = l \cdot \sqrt{LC}$ is the travelling time of the line, in seconds— according to Eqs. (26).

$$\begin{aligned} h_k(t) &= \frac{v_m(t-\tau)}{Z_c} + i_m(t-\tau) \\ h_m(t) &= \frac{v_k(t-\tau)}{Z_c} + i_k(t-\tau) \end{aligned} \tag{26}$$

That is, the behaviour of this model depends on the state of the line τ seconds before: its history. This model remains applicable to lines whose length is such that $\tau \geq \Delta t$. The model is exact, it does not depend on the selection of numeric integration rule since none is used. A decision that affects the accuracy of this model is that of the interpolation scheme for cases where τ is not a multiple of Δt.

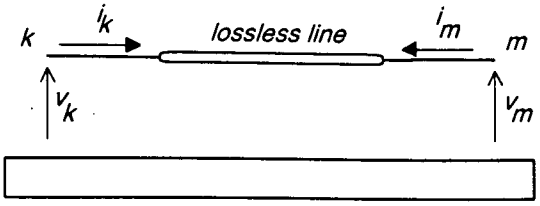


Fig. 29: Single phase lossless transmission line.

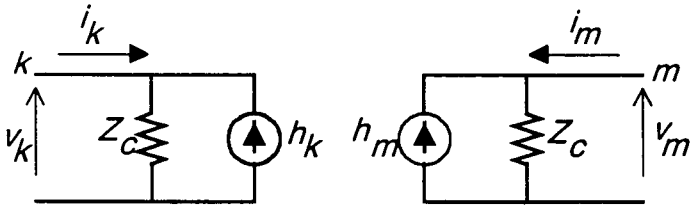


Fig. 30: Single phase lossless transmission line model.

4.5.3.2 Lossy Single Phase Transmission Line Model

Combining the model in the previous section with the lumped resistances proposed in Fig. 28, one arrives at the model sketched in Fig. 31. If the history current sources in Fig. 31 are transformed into equivalent voltage sources, the circuit becomes the one shown in Fig. 32. The history voltage source is, for node k

$$e_k(t) = v'_m(t-\tau) + Z_c \cdot i_m(t-\tau) \tag{27}$$

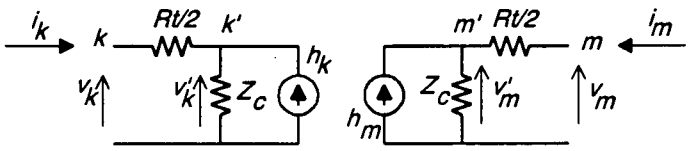


Fig. 31: Lossy single phase transmission line.

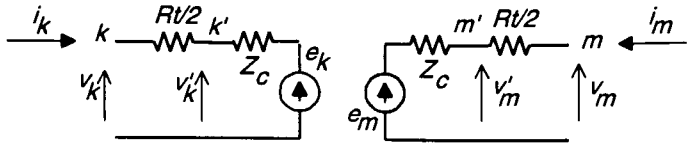


Fig. 32: History voltage source equivalent circuit.

It depends upon the voltage of the *fictitious* node m'. To reduce the workload of the simulator it is convenient to hide this node (as well as node k', on the right side) inside the model. Voltage at m' can be written

$$v'_m(t-\tau) = v_m(t-\tau) - \frac{R}{2} \cdot i_m(t-\tau) \tag{28}$$

Substituting Eq. (28) into Eq. (27) we finally obtain

$$e_k(t) = v_m(t-\tau) + \left[Z_c - \frac{R}{2}\right] \cdot i_m(t-\tau) \tag{29}$$

with an analogous expression for $e_m(t)$. Converting the voltage sources back into current sources, the complete lossy line equivalent circuit of Fig. 33 is obtained, where the history current sources are given in terms of historic values of current and voltage at the real nodes of the line according to Eq. (30) and Eq. (31).

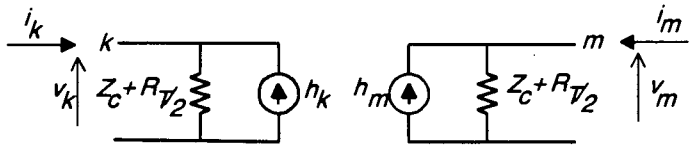


Fig. 33: Lossy line equivalent circuit.

$$h_k(t) = \frac{\left[Z_c - \frac{R_t}{2}\right]}{Z_c + \frac{R_t}{2}} \cdot i_m(t-\tau) + \frac{1}{Z_c + \frac{R_t}{2}} \cdot v_m(t-\tau) \tag{30}$$

$$h_m(t) = \frac{\left[Z_c - \frac{R_t}{2}\right]}{Z_c + \frac{R_t}{2}} \cdot i_k(t-\tau) + \frac{1}{Z_c + \frac{R_t}{2}} \cdot v_k(t-\tau) \tag{31}$$

4.5.3.3 Multi-phase transmission line

If the two ends of an n-phase transmission line are named k —the sending[7] end— and m[8] —the receiving end—, the state of the line is given by two vectors of voltages $[v_k(t)]$ and $[v_m(t)]$ and two vectors of currents $[i_k(t)]$ and $[i_m(t)]$. The parameter characterization of this line includes two full matrices of size $n \times n$, one with the inductances per metre, in H/m, $[L]$; and another with the capacitances per metre, in F/m, $[C]$, [47].

If the phase quantities given by the four vectors in the previous paragraph are transformed according to Wedepohl's [48] modal component transformation (Eqs. (32)), two vectors of modal voltages $[V_k(t)]$ and $[V_m(t)]$ —one for each end of the line— and two vectors of modal currents $[I_k(t)]$ and $[I_m(t)]$ —same as with the voltages— are obtained

$$\begin{aligned} [V_k(t)] &= [S]^{-1}[v_k(t)] \\ [V_m(t)] &= [S]^{-1}[v_m(t)] \\ [I_k(t)] &= [Q]^{-1}[i_k(t)] \\ [I_m(t)] &= [Q]^{-1}[i_m(t)] \end{aligned} \tag{32}$$

where transformation matrices $[S]$ and $[Q]$[9] depend on the physical configuration of the conductors in the line [48]. The modal transformation diagonalizes the matrices $[L]$ and $[C]$, [49], into matrices $[L_d]$ and $[C_d]$.

Voltages and currents for each mode are related in the same way that voltages and currents in the single-phase line are related [3, 48]. An equivalent circuit can be established for each mode like the one in Fig. 34, where

V_{ki} = voltage of mode i at sending end k.

V_{mi} = voltage of mode i at receiving end m.

[7] The names *sending end* and *receiving end* have historical roots in times when power networks used to be mostly radial.

[8] Actually both k and m are vectors whose entries identify the individual nodes on each end of the line.

[9] In OVNI these matrices are determined in a preprocessing step (i.e., outside the real-time loop) by MT-LINE, part of the Microtran suite.

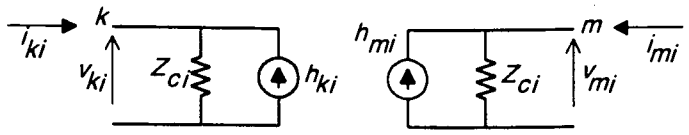


Fig. 34: Equivalent circuit for mode "i".

I_{ki} = current of mode i at sending end k.

I_{mi} = current of mode i at receiving end m.

Z_{ci} = characteristic impedance for mode i.

H_{ki} = history current source on sending end k for mode i.

H_{mi} = history current source on receiving end m for mode i.

In terms of L_{di} and C_{di}[10], the characteristic impedance for mode i is

$$Z_{ci} = \sqrt{\frac{L_{di}}{C_{di}}} \tag{33}$$

The speed of propagation for waves of mode i along the transmission line is

$$a_i = \frac{1}{\sqrt{L_{di}C_{di}}} \tag{34}$$

from which it follows that if l is the total length of the line, the travelling time for mode i is

$$\tau_i = \frac{1}{a_i} = l \cdot \sqrt{L_{di}C_{di}} \tag{35}$$

History current sources H_{ki} and H_{mi} depend on modal current and modal voltage at the other end of the line τ_i seconds before according to Eqs. (36) and (37).

$$H_{ki}(t) = \frac{V_{mi}(t-\tau_i)}{Z_{ci}} + I_{mi}(t-\tau_i) \tag{36}$$

$$H_{mi}(t) = \frac{V_{ki}(t-\tau_i)}{Z_{ci}} + I_{ki}(t-\tau_i) \tag{37}$$

[10] The i-th element in transformed inductance and capacitance diagonal matrices $[L_d]$ and $[C_d]$.

Equations (36) and (37) define the entries of two modal-history-source vectors, one for the sending end $[H_k(t)]$ and another for the receiving end $[H_m(t)]$. These two vectors are transformed back to the time domain through the corresponding inverse transformation, two history current source vectors are obtained for each phase —one for each end of the line—.

$$[h_k(t)] = [Q][H_k(t)] \tag{38}$$

$$[h_m(t)] = [Q][H_m(t)] \tag{39}$$

Applying the same inverse transformation to the decoupled modal-characteristic-conductance matrix

$$[g] = [Q] \begin{bmatrix} 1/Z_{c1} & 0 & \cdots & 0 \\ 0 & 1/Z_{c2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/Z_{cm} \end{bmatrix} [Q]^t \tag{40}$$

the full $[g]$ matrix is obtained. Matrix $[g]$ is the transmission line contribution[11] to the network bus conductance matrix $[G]$ in Eq. (45).

In the phase domain, the multiphase transmission line can be visualized by the vector/matrix-parameter equivalent circuit in Fig. 35.

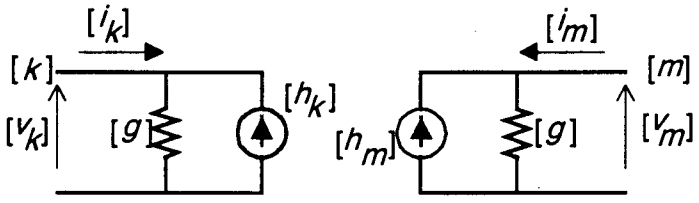


Fig. 35: Multiphase transmission line model in phase-domain. [g] is a matrix, all the other parameters are vectors.

[11] At both ends of the line.

4.5.4 Single-phase non-linear core Transformer

A new simplified model for a single phase transformer with a non-linear magnetic core whose characteristic has been piecewise linearized was prepared for this project. This model does not consider the frequency dependency of the transformer's characteristics like the one developed by Suthep and Martí [50], whose incorporation into OVNI is part of an ongoing effort.

The complete description of this model is included in chapter 9, along with that of the HVDC module created within the frame of this project as well.

4.5.5 Three-phase non-linear core Transformer

A combination of three single phase transformers modelled according to §4.5.4 provides the flexibility necessary to describe any connection group. In particular, inside the HVDC module to be described in chapter 9, two groups are detailed, namely `Yy0` and `Yd11`. The three phase model is more than a conglomerate of single phase models. It incorporates the effect of zero sequence flux linkages in the magnetic circuit of three phase units.

The complete description of the saturation modelling, for positive, negative, and zero sequence flux linkages, is postponed until chapter 11.

4.5.6 Switches

Along the simulation, some time steps bring more computational burden to the simulator than others; namely, those steps when a topological change in the network occurs. That is, when a switch or set of switches operates. Since all steps need to be of the same length, it is the computationally longest step (CLS) the one that defines the real-time bandwidth of the simulation. OVNI goes to great lengths to reduce the size of that longest step, CLS.

For the solution method chosen, as described in chapter 5, the topology of the network is described by certain matrices and either their inverses or their LU decompositions. When a switching operation occurs, it is necessary to rebuild and invert (or triangularize) those matrices, an expensive process.

Switches in OVNI are represented in one of three different forms, depending on the necessities of the simulation: as an ideal switch, with infinite impedance when open and zero impedance when closed; as a low-high resistance branch; or as a link between two MATE sub-blocks (See chapter 5).

4.5.7 HVDC Modules

To allow for the fast-switching network modelling targeted in chapter 2, high voltage direct current converters modules are included in OVNI.

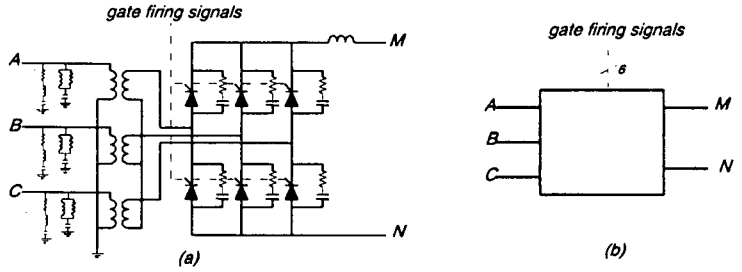


Fig. 36: OVNIs HVDC module: a) detailed view; b) block view.

To optimize the interaction between the rectifier or inverter group and the integrator, the model targetted a six-valve module, Fig. 36. The module includes the AC-side filters (11th and 13th harmonics [51]) , a three phase transformer (including saturation modelling), six valves (thyristor groups) with their corresponding RC snubbers (used to model physical snubbers, not to compensate numerical issues), and a smoothing reactor. The description in detail of this model developed for OVNI, is postponed until chapter 9.

4.5.8 HVDC-current-loop Controller

Even though the HVDC module model prepared for OVNI targets the testing of HVDC controllers, during the design and test of the HVDC module itself (particularly its commutation failure modelling features) the need for some minimal-functionality controller became evident. A constant current loop, proportional

integrative controller was prepared and optimized, and its details are described in chapter 10.

4.5.9 Metal Oxide Varistors (MOV)

In the series compensation modules included in the test case for protective relays shown in Fig. 25, metal oxide varistors (MOV) are connected in parallel with the series capacitors as a protection against overvoltages.

To represent the freezing effect of the voltage across the protected condenser when that voltage reaches the knee value is accomplished by a computationally very efficient tactic[12]: let us consider the capacitor's voltage current relationship,

$$i(t) = C \cdot \frac{dv(t)}{dt} \tag{41}$$

which can be integrated on both sides along the time axis between the points $(t - \Delta t)$ and t, approximating the integral with backward Eulers Rule:

$$i(t) \cdot \Delta t = C \cdot v(t) - C \cdot v(t - \Delta t) \tag{42}$$

$$i(t) = \frac{C}{\Delta t} \cdot v(t) - \frac{C}{\Delta t} \cdot v(t - \Delta t) = g \cdot v(t) - h(t) \tag{43}$$

As was seen in a previous section, this last expression can be represented as a circuit by the parallel of a resistor with conductance g, and a current source $h(t)$, whose value depends on the previous voltage of the capacitor, $v(t - \Delta t)$, Fig. 27b. Also, from Eq. (42), the voltage can be written as

$$v(t) = \frac{i(t) \cdot \Delta t}{C} + v(t - \Delta t) = \frac{\Delta q}{C} + v(t - \Delta t) = \frac{\Delta t}{C} \cdot i(t) + \frac{C}{\Delta t} \cdot h(t) \tag{44}$$

This last expression implies the expected result that under a constant current $i(t) = K$, the voltage grows linearly. It also implies that, still under constant current, if one refrains from updating the history source, $h(t)$, the voltage does not change.

[12] Backward Euler's rule is used in this section.

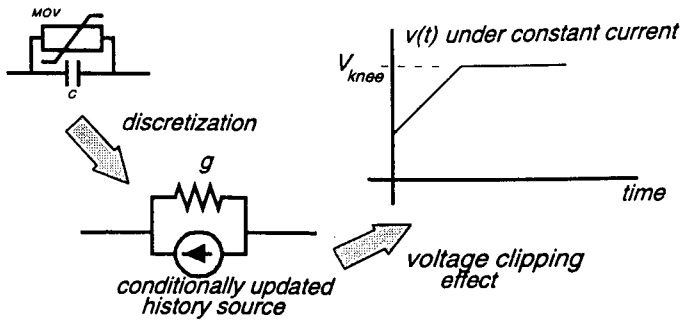


Fig. 37: Modelling the voltage clipping effect of the MOV.

The voltage clipping effect of MOVs is then simulated simply by checking the voltage across the protected condenser, and, if the voltage has reached the knee-value for the varistor, skip the updating of the corresponding history source. If anything, the activation of the varistor will reduce the execution time of the corresponding integration step, albeit by a minimal amount.

4.5.10 Measuring Transformers, ITs

It is not what happens in the network, but what the instrument transformers make of it that determines the reaction either of the protective relay, of the HVDC controller, or of any other monitoring device. Hence, it is essential to model accurately the non-linear characteristics of those transformers.

Apart from the special case of *ferroresonance*, measuring transformers have no impact on the solution of the power network [52]. It follows that the network can (and is) solved apart from the current and voltage instrument transformers. After that, the currents and voltages in the network are put through the mentioned transformers-non-linear characteristics to obtain the output which will be amplified for the monitoring or controlling devices to see.

At first, this decoupling between the network solution, and its instrument transformers' suggested the possibility of modelling the IT's on separate processing units, perhaps DSP boards. However, given the impressive improvement

in processing speed provided by the hardware industry, and for the sake of simplicity and maintainability, it became evident the convenience of running the IT's models not as special processes, but as integral parts of the main solution mechanism.

Two non-linear models were created for this project, as reported in [53]. The details of those models, one for current transformers, and another for potential transformers and CCVTs, are detailed in chapter 8, in the part on new element models developed during this project according to OVNI's guidelines for models construction.

5. SEGMENTATION AND OVNI

5.1 Introduction

The solution adopted in this project is presented in stages. First, without considering either *node hiding*, *Topological Segmentation*, or *MATE's Segmentation*. Then *Topological Segmentation*, followed by *MATE Segmentation*, and finally *Node Hiding*.

5.2 The Tasks of the Simulator

Once the original network has been conveniently discretized into a DC-resistive network, as was seen in the previous chapter, the simulation proper begins. A modified nodal analysis method [54] was selected as the framework on top of which the solution algorithm proceeds. At each integration step *five* stages or tasks can be identified, see Fig. 38.

1. *Updating History and External Sources.* Even if all sources in the discretized network are DC during each integration step, most sources change value from step to step. Some change according to an external rule, *external sources*. Others obtain a new value that depends on the previous history of voltages and currents in the elements whose models those sources are part of, *history sources*, as was seen in the last chapter. It follows that at each time step all sources need to be updated: external sources, according to their external rule, and history sources, by the elements themselves, according to their own internal rules.

2. *Accumulating Nodal Currents.* As a first approximation, to simplify the first description of the solution method, for the time being let us waive the

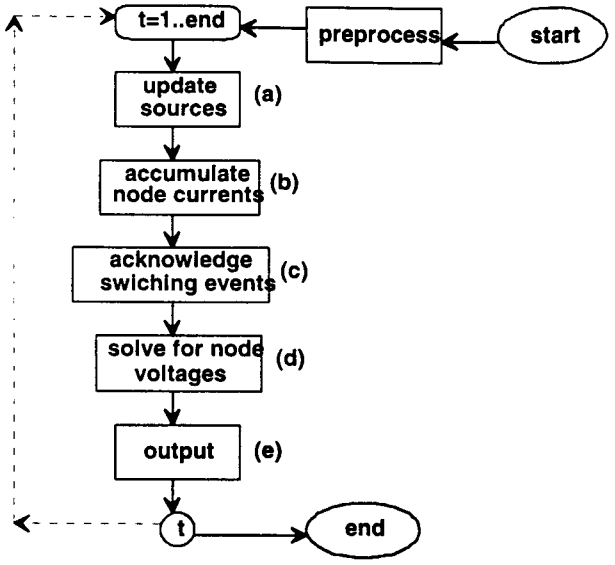


Fig. 38: Tasks in OVNI's simulation cycle.

two-layer segmentation scheme as well as the node hiding technique[1]. Also, in this first discussion, let us assume that all voltage sources have been transformed into equivalent current sources through multi-phase Thevenin's to Norton's equivalents conversions. The next step toward the solution of the network at this time step is to add up all the current sources into nodal currents, in vector $[h_\alpha]$ in Eq. (45), where α is the total of nodes in the network. In what follows, subscripts indicate the dimensions of arrays.

$$[G_{\alpha\alpha}]\,[v_\alpha] = [h_\alpha] \tag{45}$$

3. *Handling Topology-Changing Events.* The simulator acknowledges events that produce changes in the topology of the network during the current

[1] Both to be described later in this chapter.

integration step and rebuilds the discretized network conductance matrix $[G_{\alpha\alpha}]$, according to switch positions and to the status of any other topology changing device —diodes, thyristors, piecewise nonlinear elements.

4. *Solving for Nodal Voltages.* Finally, solve Eq. (45) for the nodal voltages $[v_\alpha]$. This step involves either the LU-decomposition or the inversion of the $[G_{\alpha\alpha}]$ matrix, and correspondingly, backward substitution or matrix multiplication applied to the nodal current vector $[h_\alpha]$. The first part [2] can be skipped in those integration steps where there is no topological change detected in task (3) above.

5. *Outputting Results.* Make the requested voltages and currents available for output at the corresponding data ports.

Submitting a first non-segmented implementation of the tasks listed above to a profiler, and using test case *RT-092*, see Fig. 11, it was found that the simulation step execution time was partitioned as follows:

- Updating history sources, 14.7 %.
- Accumulating Nodal Currents, 19.6 %.
- Updating Topology and Solving for nodal voltages, 65.8 %.

This partition of the execution time shows that tasks (3) and (4), updating topology and solving for nodal voltages, are the ones taking the lion's share (almost two thirds) out of the simulation loop time. In this work, a significant effort has been expended to improve the performance of these two tasks, not only from an algorithmic point of view but also at the implementation level. In this context, and to alleviate the computational burden of those tasks of the simulation[3], let us now consider the precalculation of network matrices.

[2] LU-decomposition, or matrix inversion.

[3] At the reduced price of moving much of that burden out to a *preprocessing stage.*

5.3 Precalculation of Network Matrices

To free the simulator of the enormous burden of matrix rebuilding and triangularization —or inversion— at every time step at which a topology change is detected —tasks (3) and (4) in section 5.2 on page 54—, precalculation and storing of matrices for every possible topology can be considered.

The promise of this option, however, hits the wall of feasibility in a way better described by an example. Let us consider a 1000 node network that includes 1000 switches. The admittance matrix of this network, using double precision, occupies

$$1000 \times 1000 \times 8 = 8,000,000 \text{bytes},$$

that is, almost eight megabytes[4] for a single topology matrix. Now, with 1000 switches, the network has as many as 2^{1000} possible topologies. If a matrix is to be precalculated and stored for each one of those topologies, it will need an amount of memory better described as follows: using high density *DIMM* 128 megabytes chips, at an average of 20 cm^2 per 128 megabytes, the space it occupies in an average Pentium II motherboard[5], and also assuming that all the surface of the Earth could be covered —oceans as well— with a single layer of such chips, the total memory on the Earth surface[6] would then be in the order of 3.26×10^{19} megabytes. It would still be necessary to have 8.17×10^{301} megabytes, that is, 2.51×10^{282} Earth-sized planets so covered, to prestore every possible topology matrix for the network in question. While the sparsity of those matrices, as seen in section 5.5, reduces the necessary storage to about 0.4% of the original value, we are still left in need of 1.00×10^{280} Earth-sized planets so covered.

Even under the sobering indications shed by the previous discussion, and to allow us to examine the prestorage possibility under a different light, it is

[4] 7.629 megabytes, since one megabyte is defined not as one million bytes, but as $2^20 =$ 1048576 bytes.

[5] PCPartner VIB878DS Series.

[6] Assuming that the Earth is a perfect sphere with radius equal to its equatorial and polar average: 6367 km (according to the Random House Webster's Unabridged Dictionary, 1996.)

convenient to introduce a metric, the *complexity metric.*

5.4 The Complexity Index, a metric

The computational effort of solving a network grows with the square of the number of its nodes, α. The memory requirements incurred to reduce that effort through precalculation of key network matrices grow with the power of two raised to the number of switches in the network, σ.

In this work, the complexity index of a network is introduced, and defined for coupled networks[7] as follows

$$\zeta = 2^{\sigma} \times \alpha^2 \tag{46}$$

For a network segmented into several decoupled subnetworks, the complexity index is defined as the sum of the complexity indices of each of its n subnetworks, each calculated as per Eq. (46).

$$\zeta = 2^{\sigma_1} \times \alpha_1^2 + 2^{\sigma_2} \times \alpha_2^2 + \cdots + 2^{\sigma_n} \times \alpha_n^2 = \sum_{i=1}^{n} 2^{\sigma_i} \times \alpha_i^2 \tag{47}$$

In Eq. (47), $i = 1, 2, 3, \cdots, n$ identifies the corresponding decoupled subnetwork.

5.5 Sparsity and the Solution

In general, most nodes in a power electric network are terminal to no more than three branches. This translates, for the discretized network, into a nodal analysis admittance matrix with an average of four non-zero elements per row[8]. As one considers larger networks (i.e, with a greater number of nodes) the sparsity of the matrix, defined as the percentage of null elements in the admittance matrix, grows. Using the first two sentences in this section as a guide, a network with one thousand nodes would have a matrix with an occupancy (the complement to 100 of sparsity) in the vicinity of

[7] As opposed to networks consisting of several decoupled subnetworks.

[8] Even for three phase coupled branches, the occupancy per row is still only in the vicinity of five.

$$occupancy = \frac{number\ of\ nonnull\ elm}{total\ number\ of\ elm} \times 100 = \\ = \frac{1000 \times 4}{1000 \times 1000} = 0.4\,\% \tag{48}$$

A very sparsely populated matrix indeed, with a sparsity of 99.6 %.

This enormous sparsity can be exploited to reduce the computational burden of the simulation, as well as the storage required, as described by Tinney [55]. The EMTP applies this technique in solving power electric networks [4]. However, the convenience of this very efficient storing algorithm is curtailed, at execution time, by an intense address-computation overhead. So, even with the time savings provided by skipping operations involving null elements in the matrices, timings fall short of the real time deadlines associated with the targeted bandwidth (as stated in a previous chapter). A different approach to exploiting sparsity is used in this project as described in this chapter and the next one.

5.6 *Divide et Impera.* Segmentation

Roman general Julius Caesar, c.100-44 B.C., advised *divide et impera*[9]. OVNI follows this advice to the letter. From the complexity index ς, defined in section 5.4, it follows that a smaller (one with fewer nodes) and simpler (one with fewer switches) network can be solved more rapidly, and with a reduced allocation of computational resources. Not a surprising result. To meet the desired real-time deadline, using a two-layer segmentation process, OVNI breaks the original network into a set of smaller decoupled subnetworks, each one with fewer nodes and switches than the original one. The exponential dependency of ς on the number of nodes α, and on the number of switches σ, Eqs. (46) and (47), suggests the advantage of such segmentation process.

A numerical example is in order. Consider the same 1000 node network with

[9] Latin for *divide and rule*, sometimes rendered instead as *divide and conquer*.

1000 switches introduced in section 5.3. According to Eq. (46), the complexity index of that network is

$$1000^2 \times 2^{1000} = 1.07 \times 10^{307}$$

If such network could be broken into 3-node pieces with 3 switches each (332 of them plus a 4 nodes/4 switches one), the complexity index of the segmented network would be, as per Eq. (47)

$$332 \times 3^2 \times 2^3 + 4^2 \times 2^4 = 24160$$

The complexity index has come down by more than 300 orders of magnitude. Prestoring the corresponding matrices in double precision (8 bytes per datum) would require

$$24160 \times 8 = 193280\, bytes$$

That is, less than 190 kilobytes! Admittedly, this segmentation example into 3-node subnetworks seems a bit forced, and optimistic, but it serves as an indication of the benefits to be gained through segmentation, and to bring precalculation back into the realm of feasibility. Besides, in the cases run to validate the different aspects of this project, it became evident that in three phase power networks many of the subnetworks obtained during the segmentation process, to be described later, do have 3 nodes (a fact that is exploited in the implementation).

Summarizing, segmentation reduces the size and the number of the matrices to be considered in the solution. But the question remains, how to segment the original power network?

5.7 Topological Segmentation

Figure 39 shows a typical power electric system, with power generation plants, load centres, and substations, all linked together by transmission lines. Electric signals travel along a transmission line at a speed close to that of light, but even at that speed, given the considerable length of these most visible parts of power

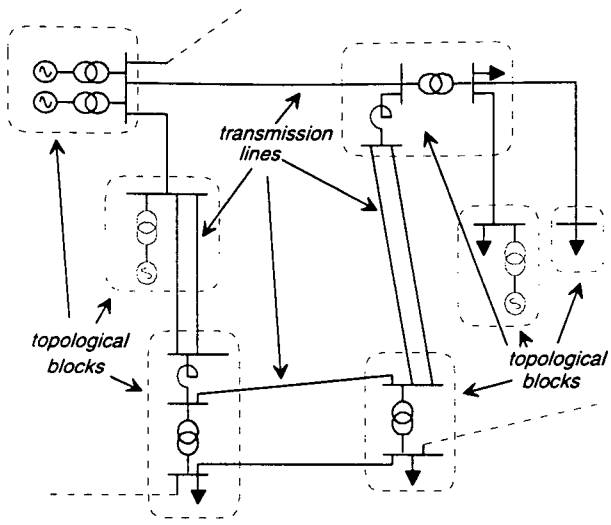


Fig. 39: A typical power electric system.

networks (very often in the hundreds of kilometres), electric phenomena at one end of a line does not reach the other end instantaneously.

As an approximate numerical example, let us consider a 300 km transmission line; approximating the propagation velocity of signals on that line to the speed of light (and using $c = 3 \times 10^8$ m/s)[10], also neglecting the differences in velocity of the different transmission modes as described by Eq. (35), one obtains the time delay with which the line passes a signal from one end to the other, $\tau = 1\, ms$, the travelling time. But in our current bandwidth of interest, with integration steps of fifty microseconds, one millisecond equates to some twenty steps. That is, whatever happens in one of the areas in Fig. 39 linked by a 300 km transmission line will not have any effect on the neighbouring areas until twenty computational cycles later. Hence, the transmission line decouples the areas it links.

An alternative way of visualizing this decoupling introduced by transmission

[10] Given the approximate nature of this example, no more accurate value was used for the speed of light, c.

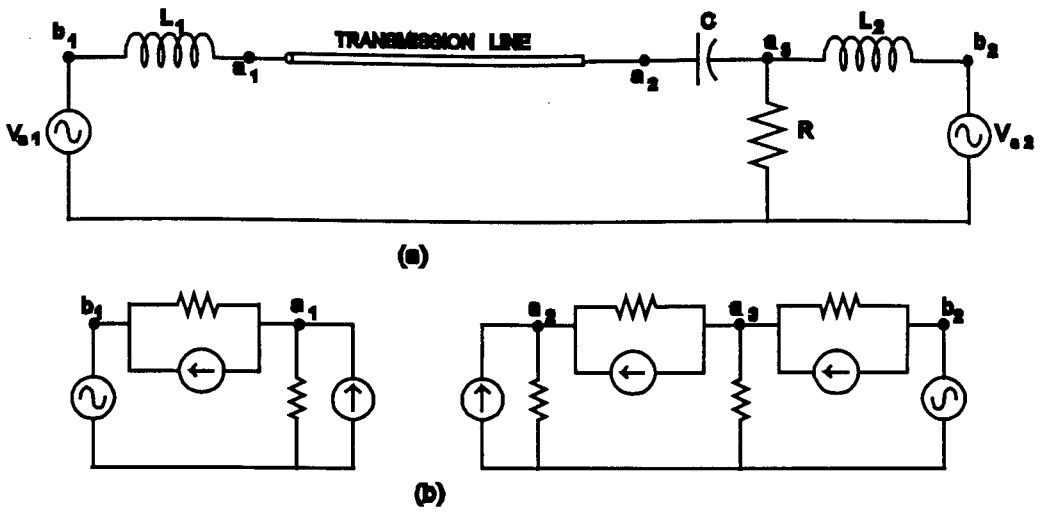


Fig. 40: a) Simple single-phase power system; b) Discrete-time equivalent circuit for system in (a).

lines in power electric networks, and the one that triggered the possibility in the mind of the author, is to take a simple power network with one single phase transmission line linking two areas as in Fig. 40a, and apply the discretization process outlined in the previous chapter to it, Fig. 40b. The decoupling introduced by the line is evident in the discretized network, where the line's model is effectively breaking the system into two blocks. In this work, block is defined as each of the parts into which the transmission link topology breaks the power network, see Fig. 41a.

This topological segmentation breaks the original problem implicit in tasks (3) and (4) in section 5.2, into several smaller problems of the same shape as that represented by Eq. (45). The sparsity of the network is exploited by this segmentation scheme by the reorganization of the nodes according to the topology boundaries defined by the transmission lines as illustrated in Fig. 41b.

It was observed that the size of those blocks ranges, for a typical power network, between 3 —the majority of the blocks— and 12, in multiples of 3

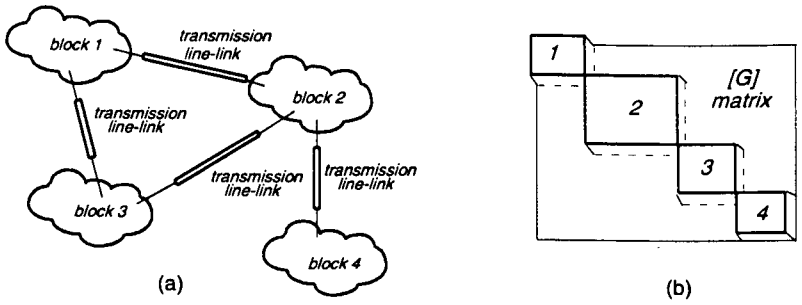


Fig. 41: a) Power network topology; b) Corresponding conductance matrix [G].

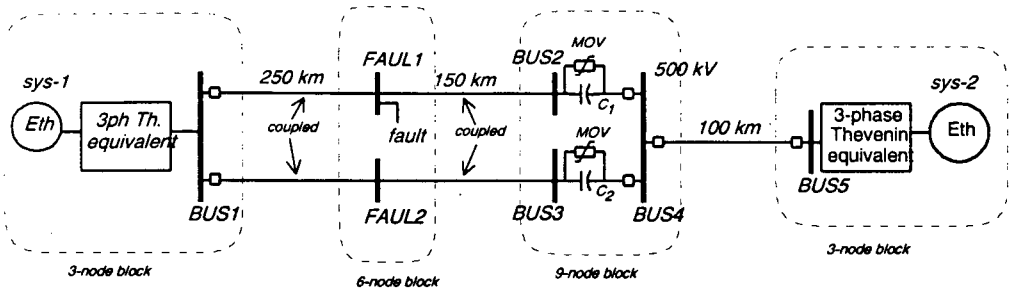


Fig. 42: Relay testing case with blocks identified.

nodes[11]. This simplifies the allocation of memory to the corresponding matrices in a way where memory address calculations are minimized, as discussed in the chapters dedicated to implementation details. For instance, the relay testing case in Fig. 42 exhibits two 3-node blocks, one 6-node block, and one 9-node block[12].

5.8 The Need for Topological Independent Segmentation, forwarding MATE [6]

When one of the blocks introduced by topological segmentation in the previous section grows past a critical size (defined by its number of nodes, branches, and

[11] Considering only those nodes which are not terminal to any voltage source.

[12] With line protection interruptors closed. This issue is dealt with through MATE segmentation, as discussed later in this thesis.

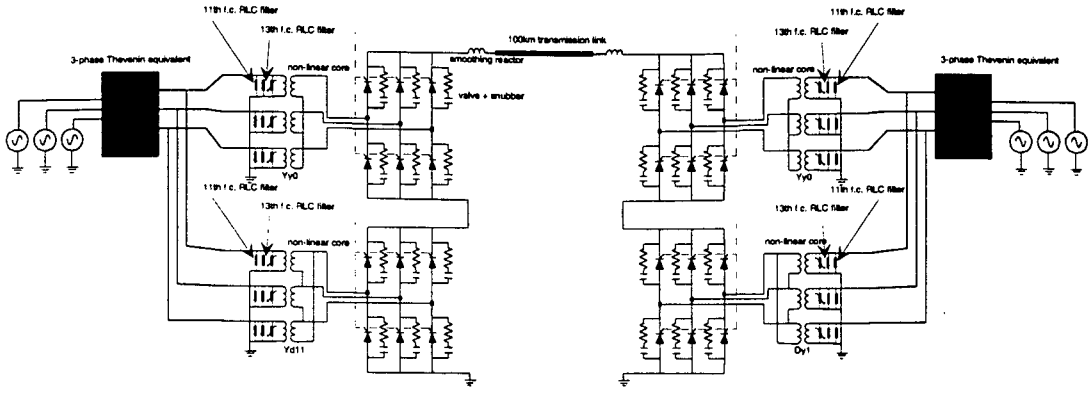


Fig. 43: A partial view of an HVDC-controller test case with two topological blocks.

sources), as in the case of the HVDC-controller testing case depicted (partially) in Fig. 43, the need for a segmentation scheme independent on the presence of transmission lines becomes evident. The MATE concept (Multiarea Thevenin Equivalents) introduced in [6] provides a framework for arbitrary system subdivision along any convenient connecting branches. The concept has been extended in this thesis to achieve maximum solution generality and maximum computational efficiency.

Instead of presenting MATE, the multi-area Thevenin equivalent segmentation concept in its extended form, an introductory simple numerical example is described. In the next section MATE, in its basic form, is described more rigorously. Finally, MATE's relationship to Classic Diakoptics is established.

In EMTP's original algorithm, ungrounded voltage sources were not included. To include an ungrounded voltage source (i.e., one not connected to the reference or ground node) we can attach to it an unknown current, I_x, and use the relationship between the voltages at the nodes of the source, as imposed by the source itself, as an additional equation[13] [54]. All of this, however, expands the dimension of the problem.

[13] Additional to the nodal equations themselves.

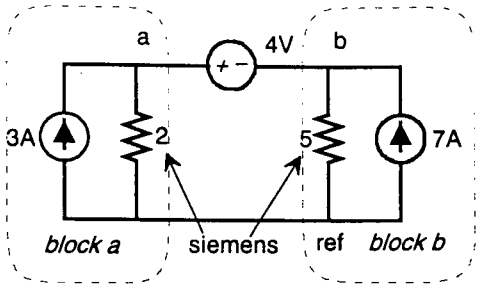


Fig. 44: Circuit with an ungrounded voltage source.

Let us begin with the simple circuit in Fig. 44, with one ungrounded voltage source. Including the current i_x through the voltage source among the unknown nodal voltages, v_a and v_b with respect to the reference node, the two nodal equations plus the v-source equation are presented in matrix form in Eq. (49).

$$\begin{bmatrix} 2 & 0 & 1 \\ 0 & 5 & -1 \\ 1 & -1 & 0 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ i_x \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ 4 \end{bmatrix} \tag{49}$$

If the voltage source were not present, the system would consist of two completely decoupled blocks described by the upper-left partition of Eq. (49), as in Eq. (50), where the first equation describes the left-hand decoupled block and the second equation the right-hand one, blocks a and b in Fig. 44 respectively.

$$\begin{bmatrix} 2 & 0 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \end{bmatrix} \tag{50}$$

After multiplying the first row by the inverse of 2, and the second row by the inverse of 5, the unitary matrix appearing on the left hand side delivers the nodal voltages if the voltage source is not present. Let us call those voltages E_a and E_b respectively, which are but the Thevenin voltages corresponding to those nodes for each of the decoupled blocks.

$$\begin{bmatrix} E_a \\ E_b \end{bmatrix} = \begin{bmatrix} 3/2 \\ 7/5 \end{bmatrix}$$

If the same process of scaling each row is applied to the top two in the original Eq. (49), and then the two first rows are used to nullify the coefficients of the third row corresponding to nodal voltages, one obtains Eq. (51).

$$\begin{bmatrix} 1 & 0 & 1/2 \\ 0 & 1 & -1/5 \\ 0 & 0 & -7/5 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ i_x \end{bmatrix} = \begin{bmatrix} 3/2 \\ 7/5 \\ 39/5 \end{bmatrix} \tag{51}$$

From the equation represented by the last row, obtaining the current i_x through the voltage source linking the two otherwise decoupled blocks of the network is simple enough. Once so obtained, i_x can be used to complement the Thevenin voltages of the nodes and produce the actual nodal voltages as in Eq. (52):

$$\begin{bmatrix} v_a \\ v_b \end{bmatrix} = \begin{bmatrix} E_a \\ E_b \end{bmatrix} - \begin{bmatrix} 1/2 \\ -1/5 \end{bmatrix} [i_x] = \begin{bmatrix} 3/2 \\ 7/5 \end{bmatrix} - \begin{bmatrix} 1/2 \\ -1/5 \end{bmatrix} [i_x] \tag{52}$$

But any arbitrarily chosen conductor can be considered as a null voltage source. That source can be used both as a link that joins and as a boundary that separates any two parts of the network. Such connecting branches are called *links* in this work, and include, in general, a resistor R_x in series with a voltage source V_x (either or both can be null), as in Fig. 45.

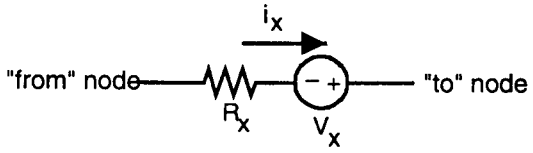


Fig. 45: An OVNI's link.

5.9 On Notation

Before continuing, let us agree on a few notational issues. In what follows, the number of nodes in a network (or subnetwork) is α; also, nodes are identified in the global network by the first few letters of the alphabet, $a\ b\ c\ldots$. The number of links[14] in the network is φ, and link branches are identified by the letters, $j\ k\ l\ldots$.

Uppercase letters stand for known quantities, lowercase letters for unknown ones. Magnitudes introduced by the segmentation process exhibit a curly hat. Physical quantities present in the original network are written without a hat. As for matrices and vectors, all vectors described are assumed to be column vectors; i.e. a row vector is indicated as a transposed column vector. Vectors display their dimension as a subscript. Matrices, also, carry their dimensions as subscripts in the order row first, column last. When a matrix is transposed, its subscripts change order.

As an example, $[Y_{\alpha\alpha}]$ is a known admittance matrix of dimensions corresponding to the total number of nodes in the network, α; as it has no hat, it corresponds to the original network. On the other hand, $\left[\widetilde{Z}_{\varphi\varphi}\right]$ is a known impedance matrix introduced by the segmentation process with dimensions corresponding to the number of links in the network, φ. As a third example, $[C_{\alpha\varphi}]$ is a connection matrix to be defined later, with a row for each node in the network, and a column for each link. The transposed of this last matrix is written $\left[C^t_{\varphi\alpha}\right]$. Observe that the subscripts are not part of the name of the matrix, but merely an indication of its dimensions.

5.10 Multi-Area Thevenin Equivalent, MATE

Once the segments into which the network is to be broken have been delineated, Fig. 46a, the branches connecting those segments are labelled links by Martí [6]. The result of the segmentation process can be seen in Fig. 46b, a cluster

[14] Branches connecting the segments produced by MATE, the Multi-Area Thevenin Equivalent.

of subblocks[15] connected by links. In that figure, the line between any two subblocks, let us say A and B, represents possibly several links connecting some nodes in subblock A to some nodes in subblock B.

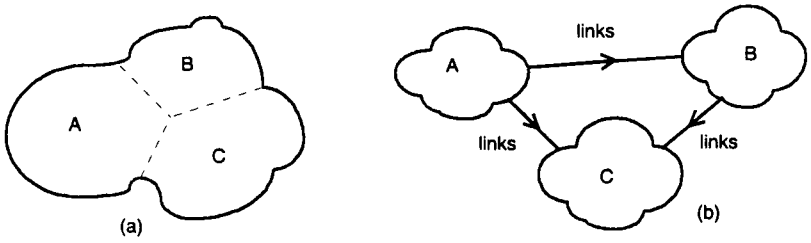


Fig. 46: a) Network with MATE's subblocks delineated; b) Subblocks connected by links, after MATE.

To solve any of the subblocks, let us say A, independently from the rest of the network, we include all link current contributions, $\left[\widetilde{i}_{\alpha}^{A}\right]$ to the subblock's own current sources $\left[I_{\alpha}^{A}\right]$. But the link current contributions, $\left[\widetilde{i}_{\alpha}^{A}\right]$, is related to the link currents vector $[i_{\varphi}]$ through a connection matrix $\left[C_{\alpha\varphi}^{A}\right]$ according to Eq. (53).

$$\left[\widetilde{i}_{\alpha}^{A}\right] = \left[C_{\alpha\varphi}^{A}\right][i_{\varphi}] \tag{53}$$

The connection matrix $\left[C_{\alpha\varphi}^{A}\right]$ has a row per each node in subblock A, and a column for every link in the whole network. That is, each element of that connection matrix relates a node in the subblock to a link in the network. That element is zero if the link does not touch the corresponding node; it is +1 if the link arrives in the node; it is -1 if the link leaves the node (To impose some regularity of formation on matrices related to this segmentation process, the author found it convenient to assign a direction to each link; a direction that coincides with the link's assumed current direction).

[15] In ONVI, MATE segmentation is applied after topological segmentation, hence it is applied to some (possibly all) of the *blocks* generated by topological segmentation. This is the reason why MATE segments are called *subblocks*.

Now we can write modified nodal equations corresponding to subblock A[16], as shown in Eq. (54) below.

$$\left[G^A_{\alpha\alpha}\right]\left[v^A_\alpha\right] = \left[I^A_\alpha\right] + \left[\tilde{i}^A_\alpha\right] \tag{54}$$

Putting together all the matrix equations of the form of Eq. (54), one for each subblock, into a single matrix equation for the segmented network (hence the hat on the corresponding conductance matrix, indicating that it is the block-diagonal matrix corresponding to the segmented cluster produced by the method, Fig. 46), one obtains Eq. (55) —written first in explicit form to illustrate its block-diagonal nature, then in a more compact form, Eq. (56)—.

$$\begin{bmatrix} G^A_{\alpha\alpha} & 0 & 0 \\ 0 & G^B_{\alpha\alpha} & 0 \\ 0 & 0 & G^C_{\alpha\alpha} \end{bmatrix} \begin{bmatrix} v^A_\alpha \\ v^B_\alpha \\ v^C_\alpha \end{bmatrix} = \begin{bmatrix} I^A_\alpha \\ I^B_\alpha \\ I^C_\alpha \end{bmatrix} + \begin{bmatrix} \tilde{i}^A_\alpha \\ \tilde{i}^B_\alpha \\ \tilde{i}^C_\alpha \end{bmatrix} \tag{55}$$

$$\left[\widetilde{G}_{\alpha\alpha}\right]\left[v_\alpha\right] = \left[I_\alpha\right] + \left[\tilde{\imath}_\alpha\right] \tag{56}$$

However, as the links contribution vector, $[i_\alpha]$, is an unknown, it belongs in the left hand side of the equation. Also, substituting Eq. (53), into Eq. (56) after the vector $[i_\alpha]$ has been moved to the left side, one obtains Eq. (57).

$$\left[\widetilde{G}_{\alpha\alpha}\right]\left[v_\alpha\right] - \left[C_{\alpha\varphi}\right]\left[i_\varphi\right] = \left[I_\alpha\right] \tag{57}$$

The system in Eq. (57), however, has more unknowns $(\alpha + \varphi)$ than equations (α). The φ additional necessary equations are provided by the φ links' voltages relationships. For one of such links, represented in Fig. 47 including its voltage source and resistance, the corresponding KVL[17] expression can be written as in Eq. (58).

$$v_{to} - v_{from} + R_x.i_x = V_x \tag{58}$$

$$v_{link} + R_x.i_x = V_x \tag{59}$$

[16] In Eq. (54), which applies only to subblock A, α is the total number of nodes in subblock A.

[17] Kirchoff's voltage law.

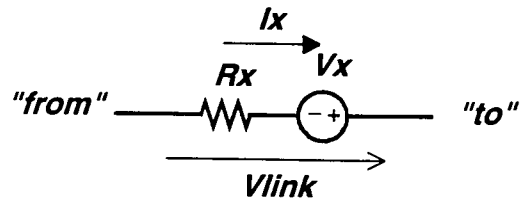


Fig. 47: A link's voltage source and resistance, and the directions assumed positive for current and voltages.

The set of all the links equations, of the form of Eq. (59), can be rewritten in matrix form as:

$$[v_\varphi] + \left[\widetilde{R}_{\varphi\varphi}\right][i_\varphi] = [V_\varphi] \tag{60}$$

The vector of link voltages $[v_\varphi]$ can be related to the nodal voltages $[v_\alpha]$ by the connection matrix $[B_{\varphi\alpha}]$ according to Eq. (61), below.

$$[v_\varphi] = [B_{\varphi\alpha}][v_\alpha] \tag{61}$$

Each element of that matrix relates a node (indicated by the column index) of the network to a link (identified by the row index) as follows:

- the element is zero if the corresponding link is not connected to the node;
- the element is +1 if the link arrives in the node;
- the element is -1 if the link leaves the node.

In short, this matrix is nothing but the transposed $[C_{\alpha\varphi}]$:

$$[B_{\varphi\alpha}] = \left[C^t_{\varphi\alpha}\right] \tag{62}$$

Applying Eq. (62) to Eq. (60) we arrive at the φ additional equations in Eq. (63), where $\left[\widetilde{R}_{\varphi\varphi}\right]$ is a diagonal matrix with the corresponding links resistances.

$$\left[C^t_{\varphi\alpha}\right][v_\alpha] + \left[\widetilde{R}_{\varphi\varphi}\right][i_\varphi] = [V_\varphi] \tag{63}$$

Equations (57) and (63) comprise the complete set of necessary equations to solve for the unknowns:

$$\left[\begin{array}{c|c} \widetilde{G}_{\alpha\alpha} & -C_{\alpha\varphi} \\ \hline C^t_{\varphi\alpha} & \widetilde{R}_{\varphi\varphi} \end{array}\right] \left[\begin{array}{c} v_\alpha \\ \hline i_\varphi \end{array}\right] = \left[\begin{array}{c} I_\alpha \\ \hline V_\varphi \end{array}\right] \tag{64}$$

$$\left[\begin{array}{ccc|c} G^A_{\alpha\alpha} & 0 & 0 & -C^A_{\alpha\varphi} \\ 0 & G^B_{\alpha\alpha} & 0 & -C^B_{\alpha\varphi} \\ 0 & 0 & G^C_{\alpha\alpha} & -C^C_{\alpha\varphi} \\ \hline C^{At}_{\varphi\alpha} & C^{Bt}_{\varphi\alpha} & C^{Ct}_{\varphi\alpha} & \widetilde{R}_{\varphi\varphi} \end{array}\right] \left[\begin{array}{c} v^A_\alpha \\ v^B_\alpha \\ v^C_\alpha \\ \hline i_\varphi \end{array}\right] = \left[\begin{array}{c} I^A_\alpha \\ I^B_\alpha \\ I^C_\alpha \\ \hline V_\varphi \end{array}\right] \tag{65}$$

As $\left[\widetilde{G}_{\alpha\alpha}\right]$ is block diagonal, its inverse is also block diagonal with Homer Brown's bus impedance matrices [15] occupying the space formerly used by the subblock; i.e.,

$$\left[Z^A_{\alpha\alpha}\right] \equiv \left[G^A_{\alpha\alpha}\right]^{-1} \tag{66}$$

Premultiplying each subblock's nodal equations by its bus impedance matrix, given by Eq. (66), we obtain

$$\left[\begin{array}{cccc} Z^A_{\alpha\alpha} & 0 & 0 & 0 \\ 0 & Z^B_{\alpha\alpha} & 0 & 0 \\ 0 & 0 & Z^C_{\alpha\alpha} & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left\{ \left[\begin{array}{cccc} G^A_{\alpha\alpha} & 0 & 0 & -C^A_{\alpha\varphi} \\ 0 & G^B_{\alpha\alpha} & 0 & -C^B_{\alpha\varphi} \\ 0 & 0 & G^C_{\alpha\alpha} & -C^C_{\alpha\varphi} \\ C^{At}_{\varphi\alpha} & C^{Bt}_{\varphi\alpha} & C^{Ct}_{\varphi\alpha} & \widetilde{R}_{\varphi\varphi} \end{array}\right] \left[\begin{array}{c} v^A_\alpha \\ v^B_\alpha \\ v^C_\alpha \\ i_\varphi \end{array}\right] = \left[\begin{array}{c} I^A_\alpha \\ I^B_\alpha \\ I^C_\alpha \\ V_\varphi \end{array}\right] \right\} \tag{67}$$

$$\left[\begin{array}{ccc|c} U^A_{\alpha\alpha} & 0 & 0 & -Z^A_{\alpha\alpha}C^A_{\alpha\varphi} \\ 0 & U^B_{\alpha\alpha} & 0 & -Z^B_{\alpha\alpha}C^B_{\alpha\varphi} \\ 0 & 0 & U^C_{\alpha\alpha} & -Z^C_{\alpha\alpha}C^C_{\alpha\varphi} \\ \hline C^{At}_{\varphi\alpha} & C^{Bt}_{\varphi\alpha} & C^{Ct}_{\varphi\alpha} & \widetilde{R}_{\varphi\varphi} \end{array}\right] \left[\begin{array}{c} v^A_\alpha \\ v^B_\alpha \\ v^C_\alpha \\ \hline i_\varphi \end{array}\right] = \left[\begin{array}{c} Z^A_{\alpha\alpha}I^A_\alpha \\ Z^B_{\alpha\alpha}I^B_\alpha \\ Z^C_{\alpha\alpha}I^C_\alpha \\ \hline V_\varphi \end{array}\right] \tag{68}$$

In Eq. (68), $\left[U^A_{\alpha\alpha}\right]$ is the unitary matrix with dimensions equal to the number of nodes in subblock A. Each of the first three rows in Eq. (68) can be written as in Eq. (69) below, which shows that if there were actually no links with other

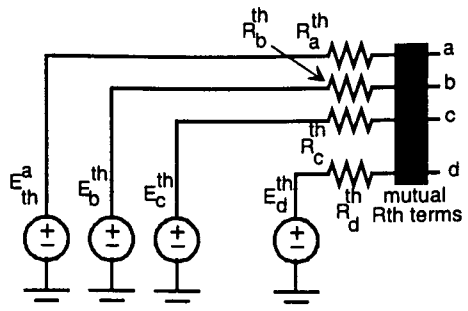


Fig. 48: MATE's Thevenin equivalent rendering for each of the subblocks. Nodes a b c d represent *docking* ones.

subblocks[18], the voltages of the nodes in subblock A would be given simply by the product $\left[Z^A_{\alpha\alpha}.I^A_\alpha\right]$. Thus we conclude that this product is nothing other than the Thevenin voltages of the subblock's nodes, Eq. (70). This last discussion also implies that the elements in the product $\left[Z^A_{\alpha\alpha}C^A_{\alpha\varphi}\right]$ are the subblock's Thevenin impedances as *seen* by the network's links, as in Eq. (71). In summary, we have

$$\left[v^A_\alpha\right] = \left[Z^A_{\alpha\alpha}I^A_\alpha\right] + \left[Z^A_{\alpha\alpha}C^A_{\alpha\varphi}\right]\left[i_\varphi\right] \tag{69}$$

$$\left[E^{thA}_\alpha\right] \equiv \left[Z^A_{\alpha\alpha}I^A_\alpha\right] \tag{70}$$

$$\left[Z^{thA}_{\alpha\varphi}\right] \equiv \left[Z^A_{\alpha\alpha}C^A_{\alpha\varphi}\right] \tag{71}$$

The network nodal voltages can then be written as

$$\left[v_\alpha\right] = \left[E^{th}_\alpha\right] + \left[Z^{th}_{\alpha\varphi}\right]\left[i^\varphi\right] \tag{72}$$

On the other hand, the last row in Eq. (68), after a manipulation that will be used explicitly in the section on node hiding, can be simplified to:

$$\left[\widetilde{R}_{\varphi\varphi} + C^t_{\varphi\alpha}Z^{th}_{\alpha\varphi}\right]\left[i_\varphi\right] = \left[V_\varphi - C^t_{\varphi\alpha}E^{th}_\alpha\right] \tag{73}$$

$$\left[\widetilde{Z}_{\varphi\varphi}\right]\left[i_\varphi\right] = \left[V_\varphi - C^t_{\varphi\alpha}E^{th}_\alpha\right] \tag{74}$$

[18] That is, if $i_\varphi = 0$.

The solution of this last smaller system of equations yields the φ link currents, $[i_\varphi]$, that, once substituted into Eq. (72), produce the corrected nodal voltages, $[v_\alpha]$. The matrix $\left[\widetilde{Z}_{\varphi\varphi}\right]$ will be the object of further study later in this thesis.

In the next two sections, the relationship between MATE and Kron's Diakoptics is discussed, along with a comment on the compensation method. Let us now meet the imagery behind the equations, as generated by MATE: Equation (72) depicts each subblock as a multisource Thevenin equivalent with as many self resistances as there are docking-nodes[19] in the subblock, and a Thevenin mutual coupling stage, as in Fig. 48.

5.11 MATE and Diakoptics [7, 8]

It was at a point well into the process of developing OVNI that the connection between MATE and Kron's Diakoptics became clear, after a comment by Dr. Dommel triggered several weeks of bibliographical research through the work of Krön; using both Krön's [7] own original tensor analysis on the subject, and Brumeller's [8] exploitations on Kron's work. The result of those weeks of work is summarized in this section.

Kron takes the *original network* and separates it into an *equivalent network* and a *removed network*. Then he reasons that if current sources are applied to the equivalent network, sources that inject into it the very same currents that were fed before by what is now the removed network, and —at the same time— the removed network is excited by voltage sources that apply to it the same voltages that appeared in it when it was part of the whole original network; then all voltages and currents in the two new networks (equivalent and removed) will be the same they were in the original one.

Using tensor analysis, Kron arrived indeed at equations equivalent to Eqs. (57) and (63) [20] obtained in the previous section, which are known as the Diakoptics

[19] Nodes to which links are connected

[20] With $V_\varphi = 0$, since Krön does not consider voltage sources in the removed network but for the ones applied to it to compensate for its removal from the rest of the network.

fundamental equations[21], minus the Thevenin equivalent interpretation, and minus the extensions to be described in chapter 6, both of which smooth out the implementation of the segmentation process.

5.12 MATE and the Compensation Method

When MATE is applied to isolate a nonlinear element from the linear part of the network, and using connecting links with no resistance or voltage source, we obtain the EMTP compensation method [4].

5.13 Node Hiding and Element Models

During tasks (3) and (4) described in Sec. 5.2, OVNI solves a form of Eq. (45) for the nodal voltages of the network. If some of those nodes could be hidden away from the integrator, OVNI, the latter's task would be a simpler and faster one. At some point a solution for those hidden nodes will be necessary. However, if the solution for the hidden nodes could be assigned to code written specifically for the topology and characteristics of the region comprising these nodes, two gains would be obtained: the hiding is in itself a form of segmentation with the advantages seen in Sec. 5.6; and the customized code would bring increased efficiency.

But customized code sounds like *anathema* in a work set to achieve a general purpose simulator. This does not need to be so. The models for system elements[22] include more often than not many nodes, a few of which are connection nodes to other elements in the network, physical or *external nodes*[23], the rest having been introduced by the modelling process, model or *internal nodes*[24], see Fig. 49. The element model is a region of the network with known topology and characteristics for which customized code can be written, and the internal nodes are good candidates to be hidden away from the main network solver,

[21] Actually, a complimentary form of the Diakoptics fundamental equations, since the original ones relate to loop currents method, and not to nodal analysis, as noted by Brumeller.

[22] See section 4.5.3.3 on page 46 for the multi-phase transmission line model, as an example.

[23] Also called nodes type a, or simply a-nodes, in this work.

[24] Also, b nodes.

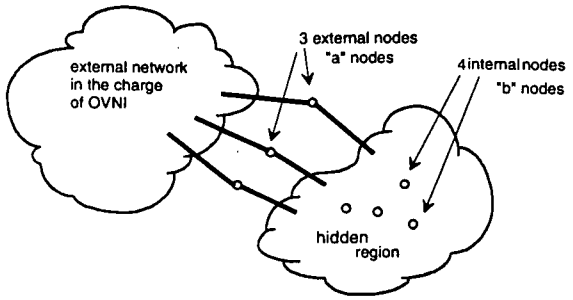


Fig. 49: Node Hiding: Internal nodes and external nodes.

OVNI. But the hidden nodes have to have some impact on the network, that impact is studied in what follows.

For a particular hidden-node region, let us identify the external node quantities by the subscript a, and the internal node quantities by the subscript b. The voltages[25] of the external nodes of the region are in the vector $[v_a]$ and the corresponding nodal currents[26] in $[h_a]$. For the internal nodes, voltages are $[v_b]$, and currents $[h_b]$. Nodal equations can be written for all those nodes:

$$\begin{bmatrix} G_{aa} & G_{ab} \\ G_{ba} & G_{bb} \end{bmatrix} \begin{bmatrix} v_a \\ v_b \end{bmatrix} = \begin{bmatrix} h_a \\ h_b \end{bmatrix} \tag{75}$$

Now, for a moment, let us assume that the voltages of the external nodes are known (they are calculated by the integrator core, OVNI, and passed as data down to the hiding region code). From the second matrix equation in Eq. (75),

$$[v_b] = [G_{bb}]^{-1} \left([h_b] - [G_{ba}] [v_a] \right) \tag{76}$$

This means that if the total current contributions to internal nodes, $[h_b]$, are known, the hidden-node region can use Eq. (76) to determine, from the given and known value of $[v_a]$ the voltages of the internal nodes, $[v_b]$. Equation (76)

[25] With respect to the reference node.
[26] Before *hiding* some of the nodes.

can be custom coded for the region (the element model).

Let us now write the equation for the external nodes implicit in Eq. (75),

$$[G_{aa}]\,[v_a] + [G_{ab}]\,[v_b] = [h_a] \tag{77}$$

Substituting the expression for internal nodes voltages in Eq. (76) into Eq. (77),

$$[G_{aa}]\,[v_a] + [G_{ab}]\,[G_{bb}]^{-1}\,[h_b] - [G_{ab}]\,[G_{bb}]^{-1}\,[G_{ba}]\,[v_a] = [h_a] \tag{78}$$

$$\left([G_{aa}] - [G_{ab}]\,[G_{bb}]^{-1}\,[G_{ba}]\right)[v_a] = [h_a] - [G_{ab}]\,[G_{bb}]^{-1}\,[h_b] \tag{79}$$

This means that the hidden-node region contribution to the external network conductance matrix is the modified matrix in Eq. (80) with dimension equal to the number of external nodes, a.

$$\left[G_{aa}^{hidden}\right] = \left([G_{aa}] - [G_{ab}]\,[G_{bb}]^{-1}\,[G_{ba}]\right) \tag{80}$$

From Eq. (79), the hiding of the nodes modifies the current contribution from the hiding zone into the external nodes according to

$$\left[h_a^{hidden}\right] = [h_a] - [G_{ab}]\,[G_{bb}]^{-1}\,[h_b] \tag{81}$$

Summarizing, the contribution of the region to the general network is:

$$\left[G_{aa}^{hidden}\right][v_a] = \left[h_a^{hidden}\right] \tag{82}$$

As the general network solver, the integrator knows nothing about hidden nodes. Managing the matrices defined by Eqs. (80) and (81) is the sole task of the subregion's code, customized and optimized. At each time step, OVNI solves for all external nodes in the network, through topological and MATE segmentation schemes, according to the combined implementation described in the next chapter. Then, those external nodes voltages are passed down to the hidden-node regions (element models), which use Eq. (76) to obtain the internal nodes voltages, necessary to update the region's history sources. Next, the region updates all its sources, independent and history ones, and accumulates the corresponding contributions to internal and to external nodes into $[h_b]$ and $[h_a]$

respectively. Before releasing the contribution of the region to OVNI, the region corrects $[h_a]$ as in Eq. (81). Finally, the subregion checks for any internal topological changes (switching) and produces and passes to OVNI the corresponding $[G_{aa}^{hidden}]$ matrix. Given the reduced size of a hidden-node region, it is likely to include a small number of switches, which implies a few possible topologies, with a few possible reduced matrices $[G_{aa}^{hidden}]$. This means that all those matrices can be precalculated and prestored before the actual simulation begins with enormous gains in speed, and only a minor penalty in memory usage.

As an example of how element models can take advantage of node hiding to improve the overall performance of the simulator, part of the work described in this thesis included the implementation of an HVDC module model according to the guidelines described above. The resulting model and its implementation in OVNI are described in Chapter 6. To test the mentioned model it was necessary to create a basic firing-angle controller, which is described in Chapter 7.

The manipulation described in Eqs. (75) to (79), but only for equation systems where the equations to be eliminated have a zero right hand side, was introduced by G. Kron [7]. In this sense the reduction described in this section is a generalization of Kron's Reduction and such is the name used for it henceforth, Generalized Kron's Reduction. The concept of node hiding is also used in other modelling approaches where the internal structure of the element is reduced down to its external nodes. For example, Marti's frequency dependent transmission line model [44], or the Ward Equivalent technique used in stability analysis [56]. The Node Hiding concept as presented here, however, does not have the limitations of the Ward Equivalent described in [56].

5.14 Node Hiding. A numerical example

At this point, in order to clarify and settle ideas, a numerical example of the node hiding technique seems convenient. Consider the 4-node circuit in Fig. 50 (where all values are either amperes or siemens, as appropriate). To validate the solution obtained through node hiding, let us first solve the network with

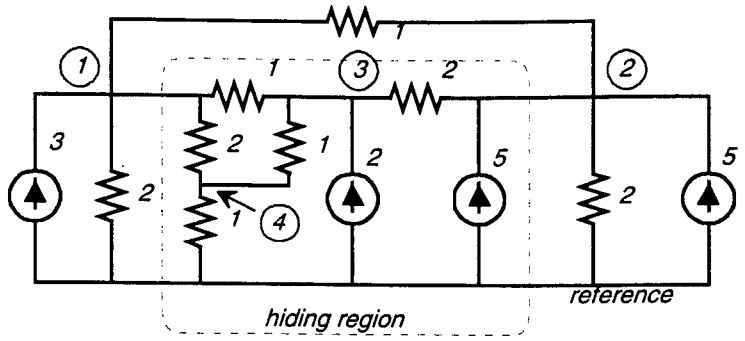


Fig. 50: Complete network with the node hiding region delineated. External nodes: (1) and (2). Internal nodes: (3) and (4).

standard nodal analysis. The equations for the complete system are:

$$\begin{bmatrix} 6 & -1 & -1 & -2 \\ -1 & 5 & -2 & 0 \\ -1 & -2 & 4 & -1 \\ -2 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 2 \\ 0 \end{bmatrix} \tag{83}$$

And the solution of this system defines the nodal voltages

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 2.479 \\ 3.948 \\ 3.630 \\ 2.147 \end{bmatrix} \tag{84}$$

Let us now use node hiding instead. In Fig. 51, the subnetwork chosen to be the hidden-node region (presumably an element's model) has its external and internal nodes clearly identified.

In Fig. 52, the *hidden-node region* is represented as a black-box to emphasize the opacity of the zone as seen by the simulator, who is in charge of the external (and reduced) network, as seen in this figure. The contribution to the external nodes relayed from currents fed into internal nodes of the hiding region is, as

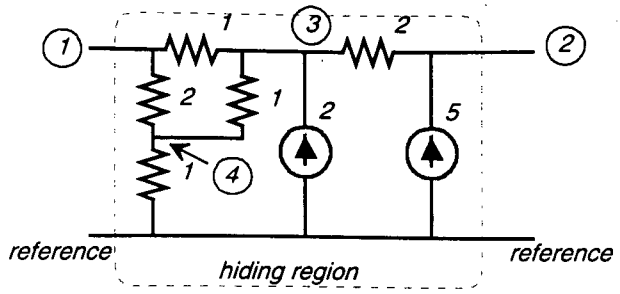


Fig. 51: Hiding zone: an element's model. See external nodes (1) and (2), and internal nodes (3) and (4).

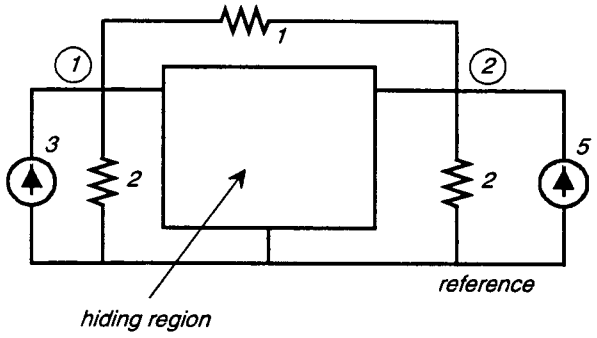


Fig. 52: "External" network, as seen by OVNI, with hidding region represented as a "black-box".

per Eq. (81),

$$\begin{bmatrix} h_1^{relayed} \\ h_2^{relayed} \end{bmatrix} = - \begin{bmatrix} -1 & -2 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.066... \end{bmatrix} \; amps \quad (85)$$

The external network, minus the *hidden-node region* (HR), has two nodes: (1) and (2), and the conductance matrix (also minus *HR*):

$$[G_{ex}] = \begin{bmatrix} 3 & -1 \\ -1 & 3 \end{bmatrix} \quad (86)$$

The hidden-node region contributes the matrix $[G_{HR}]$ to the external system. This matrix is computed at a preprocessing stage from the hiding region's "whole" matrix (the one that describes HR with all its nodes, and not connected to the outside world):

$$[G_{HR}] = \left[\begin{array}{cc|cc} 3 & 0 & -1 & -1 \\ 0 & 2 & -2 & 0 \\ \hline -1 & -2 & 4 & -1 \\ -2 & 0 & -1 & 4 \end{array}\right] \tag{87}$$

From Eq. (80),

$$[G_{HR}] = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} - \begin{bmatrix} -1 & -2 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}^{-1} \begin{bmatrix} -1 & -2 \\ -2 & 0 \end{bmatrix} = \begin{bmatrix} 1.4 & -0.8 \\ -0.8 & 0.9\hat{3} \end{bmatrix} \tag{88}$$

The total external network is represented by the sum of $[G_{ex}]$ and $[G_{HR}]$:

$$[G] = \begin{bmatrix} 4.4 & -1.8 \\ -1.8 & 3.9333 \end{bmatrix} \tag{89}$$

Current contribution from HR is given by Eq. (81):

$$[h_{HR}] = \begin{bmatrix} 0 \\ 5 \end{bmatrix} + \begin{bmatrix} 0.8 \\ 1.06666 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 6.06666 \end{bmatrix} \tag{90}$$

The external solver receives Eq. (90) results and solves:

$$\begin{bmatrix} 4.4 & -1.8 \\ -1.8 & 3.933 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix} + \begin{bmatrix} 0.8 \\ 6.0666 \end{bmatrix} \tag{91}$$

The external solver "sees" only Eq. (91), and computes:

$$\begin{aligned} v_1 &= 2.479\ V \\ v_2 &= 3.948\ V \end{aligned} \tag{92}$$

Which are the same results obtained in Eq. (84) from the standard solution. At this point HR takes those values in Eq. (92) provided by the simulator and uses Eq. (76) to find its internal node voltages:

$$\begin{bmatrix} v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 4 & -1 \\ -1 & 4 \end{bmatrix}^{-1} \left(\begin{bmatrix} 2 \\ 0 \end{bmatrix} - \begin{bmatrix} -1 & -2 \\ -2 & 0 \end{bmatrix} \begin{bmatrix} 2.479 \\ 3.948 \end{bmatrix} \right) = \begin{bmatrix} 2.630 \\ 2.147 \end{bmatrix} \tag{93}$$

All tasks in Eqs. (88, 90, and 93) are under the charge of HR, leaving the simulator's core the much lighter burden of solving Eq. (91). When HR is an element model, its topology is of predictable and limited change nature, ergo its matrices and operations in Eqs. (88, 90, and 93) can be greatly optimized. This will be examined further in chapter 9.

6. SOURCES, LINKS AND EXPANDED MATE

6.1 Introduction

Two main issues of the solution are described in this chapter: the representation of current and voltage sources in OVNI; and an extension of MATE to handle more efficiently ungrounded voltage sources that are not part of a link. In this sense, this chapter deals with the first task of the simulator, as seen in Sec. 5.2

6.2 Precalculation of Source Values

Sources in OVNI fall into one of these categories:

- DC sources,
- Time-periodic sources[1].

For the second category, periodic sources, determination of their values at each time step requires some computational effort (from a minimum of time-boundary testing, in the case of a square wave, up to the expensive and sophisticated numerical involvement of a sine wave[2])

To reduce the impact of source updating, source values are calculated and stored in tables before the simulation begins. Those tables are made available to the integrator during the simulation.

The first attempt to do this was to use one cycle of the source's signal. To represent one cycle of a periodic source with a period of T seconds, see Fig. 53,

[1] Sinusoidal, sawtooth, square wave, triangular sources, etc.

[2] On a Pentium II, a sine computation takes the numerical coprocessing subblock of the CPU up to 30 times that of a sum's [57].

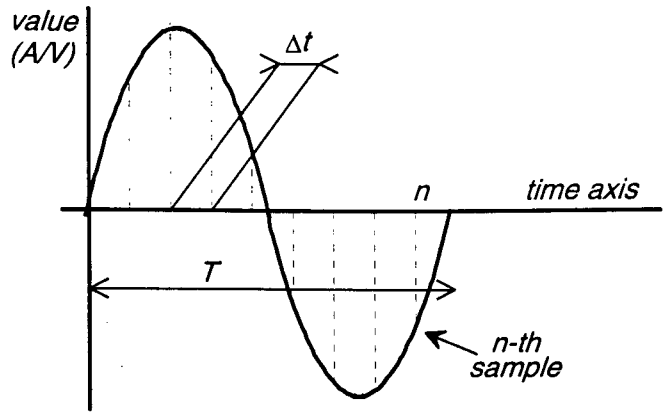


Fig. 53: The n prestored samples of a sinusoidal source.

in a simulation with an integration step Δt, n samples are necessary, as given by Eq. (94), where "int" is the *integer part* function.

$$n = \text{int}\left(\frac{T}{\Delta t}\right) + 1 \tag{94}$$

This simplification, however, brings the problem of sample mismatch at the end of the cycle in cases where the integration step is not a divisor of the source's period. Observe sample n in Fig. 54, the last one of the source's prestored samples (if only one cycle of the source is so treated). At the next integration step, identified in that figure as $n + 1$, the integrator expects the correct value for the source, V_{right}. Instead, the table index wraps around and produces the value labelled V_{wrong} in the figure.

The sampling mismatch is effectively reducing the frequency of the source's wave, and introducing higher frequency components. In short, this method distorts the signal of the represented source.

The reduction in the source's effective frequency, in percentage, is given by Eq. (95), where n is the number of prestored samples as calculated by Eq. (94).

$$\Delta f(\%) = \left(1 - \frac{T}{n\Delta t}\right) 100 \tag{95}$$

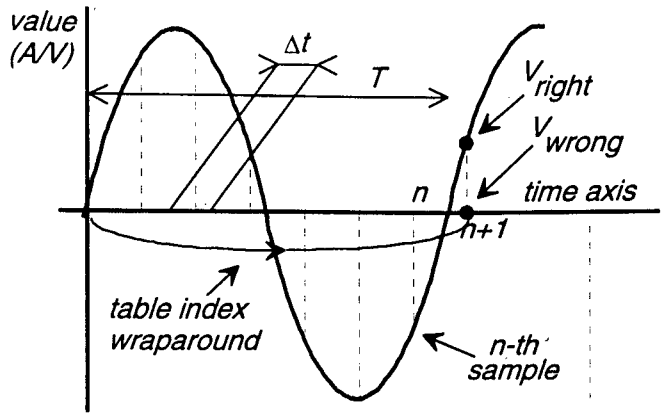


Fig. 54: Wraparound of prestored source's samples.

In a simulation with $\Delta t = 50\mu s$, a 60 Hz source is represented by its $n = 334$ samples. Its frequency decreases 0.2% in the process, down to 59.88 Hz. At this rate, in only ten cycles of simulation, the source's phase lags 7.2^o. The additional distortion implied by the introduction of high frequency components is of relatively little consequence, being very small to begin with, and further damped by the frequency response characteristics of the integration rule used in the solution process, see § 3.4.

To avoid the mismatch discussed above, the preprocessor in OVNI prestores in the source's table, not the number of samples that fit into one source cycle, but the number of samples n that fit in the least common multiple (LCM) of the source's period T, and the dominant integration step Δt, Eq. (96)

$$n = \frac{\text{lcm}(T, \Delta t)}{\Delta t} + 1 \tag{96}$$

Where, to make a valid use of the integer function "lcm", both T and Δt are truncated to microseconds with no fractional part.

For instance, in the case of $\Delta t = 50\mu s$, and a source frequency of 60 hertz, the preprocessor should store 1000 samples, and not just 334. That is, in this case it takes three source's cycles to resynchronize the precalculated table with the simulation discrete samples stream. But that table, in double precision

IEEE format, occupies more than 64 kilobytes. If the source is an odd one, with a frequency different from the rated frequency, Eq. (96) can be used. If, instead, the source is just one of many with that frequency in the network under simulation, it may be considered to change instead the integration step Δt up or down to the nearest divisor of the period associated with that frequency, Eq. (97).

$$\Delta t_{adj} = \frac{T}{\text{int}\left(\frac{T}{\Delta t}\right)} \tag{97}$$

For the same case introduced in the last paragraph, and using Eq. (97), an adjusted integration step could be calculated as $\Delta t_{adj} = 50.048$ or $49.898\mu s$.

Using this adjusted integration step reduces the number of necessary samples per source (to only 334 in the example that occupies us; i.e., in double precision, slightly more than two and a half kilobytes worth of memory). This is all accomplished without a significant change in the bandwidth of the simulation.

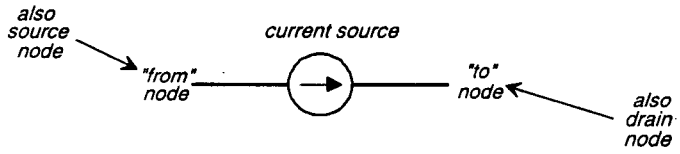


Fig. 55: A current source in OVNI: its nodes.

6.3 Current Sources

The simulator uses a variant of nodal analysis to solve each one of the fragments into which the network has been broken by the multi-layer segmentation process described in sections 5.7 and 5.10. Nodal analysis accounts for current sources in a natural way, their values are computed at each time step, and those values are duly accumulated into the corresponding nodal current vector —$[h_\alpha]$ in Eq. (45)— at the proper time.

The two nodes of a current source are identified, in this work, as the *drain*

node and the *source node*, according to Fig. 55.

6.4 Voltage Sources

In this work, voltage sources were the original motivation to explore Ho's modified nodal analysis [54]. Later, in the light of MATE, new possibilities entered the picture. However, given the strict speed requirements on the simulator, additional options were explored and implemented. In OVNI different[3] internal representations of voltage sources are used depending on the answer to these few questions:

- Is the current in that source needed?
- Is one of the nodes of the source connected to the ground or reference node? That is, the source is grounded.
- Is the source part of a user-defined MATE boundary?

In the next few sections, the different options used are introduced.

6.4.1 Grounded Voltage Sources —GVS

Inclusion of a voltage source in MATE's solving scheme, § 5.2, creates a new link[4] equation and its corresponding unknown current. Inclusion of a voltage source in Ho's modified nodal analysis [54] introduces the current in the source as an additional unknown, along with the corresponding equation. In short, a voltage source inclusion in OVNI's solution scheme seen so far expands the system of equations by one more row and one extra column. In exchange for the additional work the method delivers the current (and implicitly, the power too) for that voltage source. That is true even for grounded voltage sources (GVS)[5]

[3] All of this remains transparent to the user.

[4] Granting that the source is part of a segmenting user-defined boundary.

[5] A source connected between ground —or the reference node— and a certain node that is called here *the GVS node* or, more often, the *k*-node —*k* as in *k*nown.

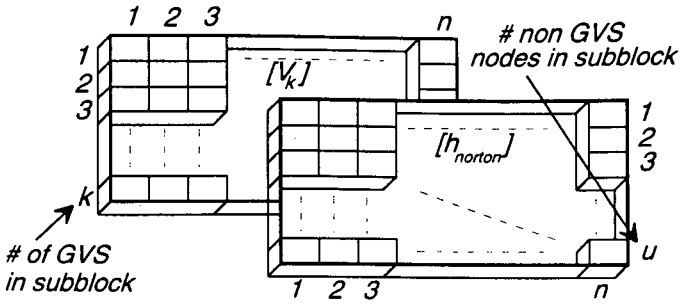


Fig. 56: Precalculated time matrices corresponding to grounded voltage sources in a subblock.

However, MATE's equations are redundant in the case of a *GVS*, since they imply calculation of the voltage of every node in the subblock, including the ungrounded nodes of GVS's; and the voltages of *GVS-nodes* are already known.

So, in cases where neither current nor power in a *GVS* is needed, a different and more efficient path of computation is taken.

First, we order the nodes in the subblock in such a way that all the nodes which are terminal to *GVS*'s[6] occupy the last k positions among the subblock's nodes. The other nodes in the subblock occupy the first u positions[7].

Nodal equations can be written for the subblock as

$$\begin{bmatrix} G_{uu} & G_{uk} \\ G_{ku} & G_{kk} \end{bmatrix} \begin{bmatrix} v_u \\ v_k \end{bmatrix} = \begin{bmatrix} h_u \\ h_k \end{bmatrix} \tag{98}$$

As v_k, the voltage vector of the *GVS* nodes with respect to the reference node, is known, the first row of Eq. (98) can be written

$$\begin{aligned} [G_{uu}][v_u] + [G_{uk}][v_k] &= [h_u] \\ [G_{uu}][v_u] &= [h_u] - [G_{uk}][v_k] \end{aligned} \tag{99}$$

[6] Labelled by OVNI *known nodes* or k-nodes

[7] Named by OVNI *unknown nodes* or u-nodes)

The product on the far right in Eq. (99) is the vector of Norton equivalent current sources corresponding to the k GVS in the subblock, feeding the u non-GVS nodes, $[h_{norton}]$.

Vectors $[v_k]$ and $[h_{norton}] = [G_{uk}][v_k]$ are both precalculated and stored as matrices. The $[v_k]$'s precalculated matrix has as many rows as there are GVS nodes in the subblock, k. $[h_{norton}]$'s precalculated matrix has one row per non-GVS node in the subblock, u. Both matrices have as many columns as the least common multiple[8], n, of the numbers $n_1, n_2, \ldots, n_k$ of prestored samples for each GVS in the subblock. See Fig. 56.

$$n = \text{lcm}(n_1, n_2, n_3, \ldots, n_k) \tag{100}$$

6.4.2 An example on Grounded Sources, MATE versus Norton

To contrast the efficiency of the approach in § 6.4.1, when compared with MATE solution for grounded sources, a simple numerical example is included in this section.

In the simple network with one grounded voltage source, GVS, shown in Fig. 57, both solutions are compared.

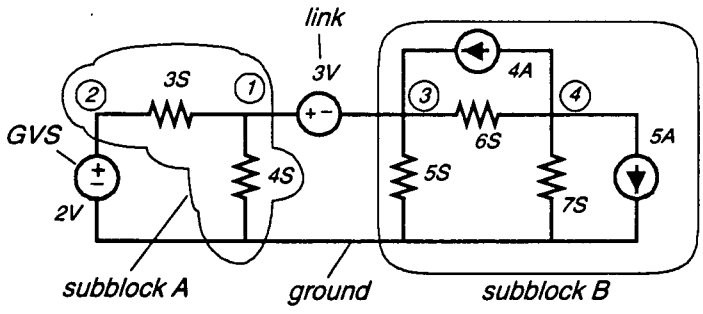


Fig. 57: Network with one grounded voltage source accounted for as a *link*.

First, let us solve the problem considering that every voltage source in the network is a MATE link. This assumption produces the two subblocks outlined

[8] To avoid voltage distortion due to sample/step mismatch, as was seen in section 6.2

in Fig. 57, where subblock A is *linked* to ground by the 2V source/link and to subblock B by the 3V source/link.

The modified nodal analysis equations, reorganized according to MATE input requirements are in Eq. (101), where the first two rows correspond to subblock A, the next two rows to subblock B, and the last two rows to the links x and y.

$$\left[\begin{array}{cc|cc|cc} 7 & -3 & 0 & 0 & 1 & 0 \\ -3 & 3 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 11 & -6 & -1 & 0 \\ 0 & 0 & -6 & 13 & 0 & 0 \\ \hline 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} v_1 \\ v_2 \\ \hline v_3 \\ v_4 \\ \hline i_x \\ i_y \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ \hline 4 \\ -9 \\ \hline 3 \\ 2 \end{array}\right] \tag{101}$$

Manipulating Eq. (101) according to MATE, § 5.10, produces

$$\left[\begin{array}{cc|cc|cc} 1 & 0 & 0 & 0 & 0.2500 & 0.2500 \\ 0 & 1 & 0 & 0 & 0.2500 & 0.5833 \\ \hline 0 & 0 & 1 & 0 & -0.1215 & 0 \\ 0 & 0 & 0 & 1 & -0.0561 & 0 \\ \hline 0 & 0 & 0 & 0 & -0.3715 & -0.2500 \\ 0 & 0 & 0 & 0 & -0.2500 & -0.5833 \end{array}\right] \left[\begin{array}{c} v_1 \\ v_2 \\ \hline v_3 \\ v_4 \\ \hline i_x \\ i_y \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \\ \hline -0.0187 \\ -0.7009 \\ \hline 2.9813 \\ 2 \end{array}\right] \tag{102}$$

The last two rows in Eq. (102) are MATE's link equations that, once solved, yield the currents in the two links

$$i_x = -8.0354A \qquad i_y = 0.0152A$$

These link currents, inserted into the Thevenin equations represented by the four first rows of Eq. (102) result in the node voltages

$$v_1 = 2.0051V \qquad v_2 = 2.0000V$$
$$v_3 = -0.9949V \qquad v_4 = -1.1515V$$

Now, if instead of making a link out of the grounded voltage source, subblock A absorbs and transforms it according to § 6.4.1 —the subblock's nodal current vector is null, since it is computed with the subblock disconnected from the rest of the network and there are no current sources in this subblock—.

$$\begin{bmatrix} 7 & -3 \\ -3 & 3 \end{bmatrix} \begin{bmatrix} v_1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{103}$$

This is the same as Eq. (98), from which the simplification in Eq. (99) is

$$[7][v_1] = [0] - [-3][2] \tag{104}$$

With this simplification of subblock A, Eq. (101) is reduced to

$$\left[\begin{array}{c|cc|c} 7 & 0 & 0 & 1 \\ \hline 0 & 11 & -6 & -1 \\ 0 & -6 & 13 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \left[\begin{array}{c} v_1 \\ \hline v_3 \\ v_4 \\ \hline i_x \end{array}\right] = \left[\begin{array}{c} 6 \\ \hline 4 \\ -9 \\ \hline 3 \end{array}\right] \tag{105}$$

Thus the MATE equation in Eq. (102) becomes Eq. (106), where the link system of equations has been reduced in dimension.

$$\left[\begin{array}{c|cc|c} 1 & 0 & 0 & 0.1429 \\ \hline 0 & 1 & 0 & -0.1215 \\ 0 & 0 & 1 & -0.0561 \\ \hline 0 & 0 & 0 & -0.2644 \end{array}\right] \left[\begin{array}{c} v_1 \\ \hline v_3 \\ v_4 \\ \hline i_x \end{array}\right] = \left[\begin{array}{c} 0.8571 \\ \hline -0.0187 \\ -0.7009 \\ \hline 2.1242 \end{array}\right] \tag{106}$$

This MATE system produces the same results reported above, minus the current through the grounded voltage source, but with fewer operations than were necessary to solve the original system in Eq. (102).

6.4.3 Ungrounded Voltage Sources, UVS

When an ungrounded voltage source occurs in a branch designated by the user as a MATE segmentation boundary (i.e., as a *link* branch), Fig. 58, the solution for the source falls in line with the basic MATE algorithm, as seen in § 5.10 on page 67.

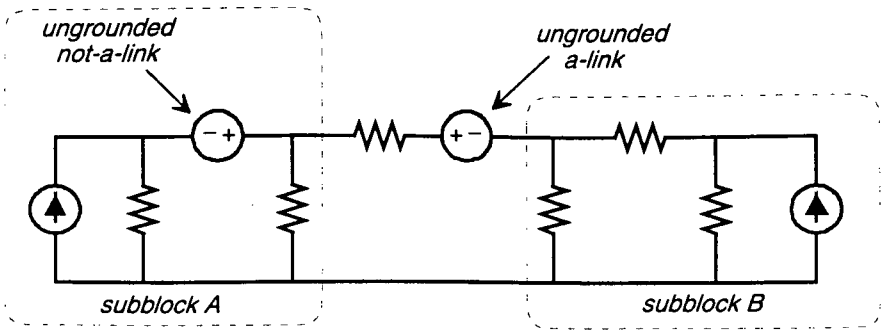


Fig. 58: Ungrounded voltage sources in OVNI: a) a link; b) not a link.

If the ungrounded source, however, is not within a *link* branch, Fig. 58, its solution falls with the Extended MATE algorithm, seen in § 6.5 on page 92.

6.4.4 Voltage Sources "Ownership"

In a network that has been broken, first into blocks (topological segmentation, § 5.7), then into subblocks (MATE segmentation, § 5.10), the issue of where voltage sources belong is not trivial. From what was said in § 6.4.1 and § 6.4.3, the dealing with voltage sources belongs with the solution of a block or of a subblock, as follows:

- *Grounded Voltage Sources (GVS)* belong inside the corresponding subblock, which is the responsible for including them in the solution.
- *Ungrounded Voltage Sources (UVS)* Two cases:
  - *Link Sources (ULS).* In the case when the source is part of a MATE's user defined boundary, the source is dealt with as one of MATE's links and handled directly by the enclosing block.
  - *Non-link Sources (UNLS).* In the case when the source is not part of a MATE's user defined boundary, the source, obviously, belongs inside a subblock, and it is solved for inside that subblock.

Voltage sources { grounded ⟶ in subblock.
ungrounded { link → in block.
not link → in subblock.

Fig. 59: Voltage sources "ownership", in OVNI.

Figure 59 summarizes this section.

6.5 Extended MATE

Before the need for an extension to MATE is established, let us begin by revisiting its imagery. In this section, the subscript convention introduced in § 5.9 to indicate a matrix or vector dimensions is not used; subscripts to matrices and vectors indicate the subblock they belong to.

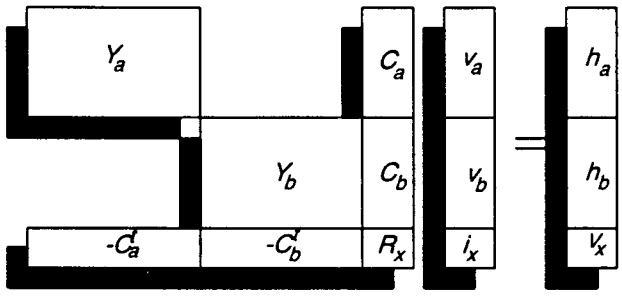


Fig. 60: KCL nodal equations and KVL voltage sources equations, getting ready for standard MATE.

Consider a topological block that has been segmented into two subblocks by a set of MATE's links. The block's nodal KCL[9] equations plus its links KVL[10] can be represented pictorically as in Fig. 60. A subblock A is in search of its nodal voltages $[v_a]$, and is described by

- its bus admittance matrix $[Y_a]$,
- its nodal currents vector $[h_a]$,

[9] Kirchoff's Currents Law.
[10] Kirchoff's Voltages Law.

- and its connection matrix $[C_a]$.

The block works to determine its links currents $[i_x]$. The links themselves are described by

- the links resistance matrix[11] $[R_x]$, and
- the links voltage sources vector $[V_x]$.

After a preprocessing stage outlined by chapter the subblock A is described by its bus impedance matrix $[Z_a]$ (the inverse of $[Y_a]$), its Thevenin impedance matrix $[Z_{Ta}]$ (i.e., the product of $[Z_a]$ and $[C_a]$), and its Thevenin voltages vector $[E_a]$ (product of $[Z_a]$ and $[h_a]$). This convenient way of MATE's for identification of matrices and vectors in the problem stems from a basic assumption: that a subblock can only contain current sources or GVS's.

In this section, MATE is extended to override those restrictions; albeit at the price of losing the physical meaning of matrices and vectors in the solution. Let us first see what the extension is, then explore its use in a short numerical example.

The extensions necessary to deal with a subblock that includes UVS's —like the subblock A in Fig. 62— are (for that subblock, see Fig. 61)

- Extend its nodal voltage vector $[v_a]$ with a vector of UVS's currents at the bottom $[i_{sa}]$, to produce the *extended* vector $[v_a^*]$.
- Extend its nodal current vector $[h_a]$ with a vector of UVS's voltages at the bottom $[v_{sa}]$, to produce the *extended* vector $[h_a^*]$.
- Extend its connection matrix $[C_a]$ at the bottom with as many null rows as there are UVS's in the subblock, to generate the *extended* connection matrix $[C_a^*]$.

[11] A diagonal matrix with one entry per link.

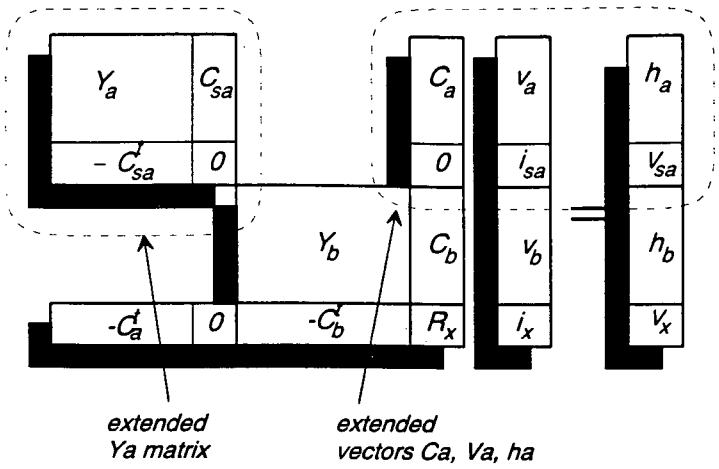


Fig. 61: Extended matrices and vectors for the subblock with UVS's. Extended MATE.

- Extend its admittance matrix $[Y_a]$ with the internal UVS's connection matrix $[C_{sa}]$, as in Fig. 61, to generate the *extended* matrix $[Y_a^*]$.
- Finally, ignore the names of vectors and matrices in this subblock and build the *extended* matrices indicated in what follows —whose names are kept for the sake of mnemotecnic association, since they are not impedances or voltages anymore—, and then proceed as in standard MATE. The matrices are:
  - Extended or *pseudo* bus impedances, $[Z_a^*] = [Y_a^*]^{-1}$,
  - Extended or *pseudo* Thevenin impedances $[Z_{Ta}^*] = [Z_a^*][C_a^*]$, and
  - Extended or *pseudo* Thevenin voltages $[E_a^*] = [Z_a^*][h_a^*]$.

6.5.1 Extended MATE: A numerical example

In Fig. 62, a single block network has been broken into two subblocks by the 4-ohm 3-volt link. The subblock on the right, A, includes an ungrounded voltage source (UVS). In this example, there are two voltage sources: one is part of a link, the 3-volt source; the other, is ungrounded and part of a subblock.

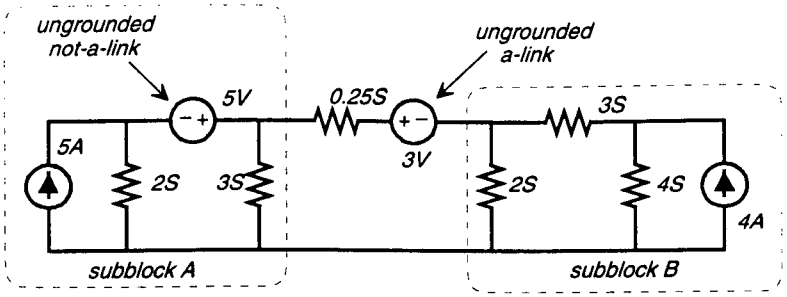


Fig. 62: Subblock with and ungrounded voltage source.

Introducing the currents in the voltage sources, i_x and i_k, as in the figure, the nodal equations are

$$\begin{aligned} \text{Node (1):} \quad 2v_1 + i_x &= 5 \\ \text{Node (2):} \quad 3v_2 - i_x - i_k &= 0 \\ \text{Node (3):} \quad 5v_3 - 3v_4 + i_k &= 0 \\ \text{Node (4):} \quad 3v_4 - 3v_3 + 4v_4 &= 4 \end{aligned} \tag{107}$$

The voltage sources introduced two unknowns, their currents i_x and i_k. They introduce two equations as well

$$\begin{aligned} \text{UVS source:} \quad v_2 - v_1 &= 5 \\ \text{Link source:} \quad v_2 - v_3 + 4i_k &= 3 \end{aligned} \tag{108}$$

Solving the system of seven equations comprised by Eq. (107) and Eq. (108) we obtain

$$\begin{aligned} v_1 &= -1.9793 \text{ V}; \quad & v_4 &= 0.7573 \text{ V}; \\ v_2 &= 3.0207 \text{ V}; \quad & i_x &= 8.9587 \text{ A}; \\ v_3 &= 0.4337 \text{ V}; \quad & i_k &= 0.1033 \text{ A}. \end{aligned}$$

The equations, written in matrix form and including the UVS equation in subblock A's equations according to *extended* MATE, are

$$\left[\begin{array}{ccc|cc|c} 2 & 0 & 1 & 0 & 0 & 0 \\ 0 & 3 & -1 & 0 & 0 & -1 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 5 & -3 & 1 \\ 0 & 0 & 0 & -3 & 7 & 0 \\ \hline 0 & 1 & 0 & -1 & 0 & 4 \end{array}\right] \left[\begin{array}{c} v_1 \\ v_2 \\ i_x \\ \hline v_3 \\ v_4 \\ \hline i_k \end{array}\right] = \left[\begin{array}{c} 5 \\ 0 \\ 5 \\ \hline 0 \\ 4 \\ \hline 3 \end{array}\right] \tag{109}$$

Premultiplying the rows corresponding to subblock A by the pseudo bus impedance matrix of the subblock; and also premultiplying the rows corresponding to subblock B by the subblock's pseudo bus impedance matrix, Eq. (109) becomes

$$\left[\begin{array}{ccc|cc|c} 1 & 0 & 0 & 0 & 0 & -0.2 \\ 0 & 1 & 0 & 0 & 0 & -0.2 \\ 0 & 0 & 1 & 0 & 0 & 0.4 \\ \hline 0 & 0 & 0 & 1 & 0 & 0.2692 \\ 0 & 0 & 0 & 0 & 1 & 0.1154 \\ \hline 0 & 1 & 0 & -1 & 0 & 4 \end{array}\right] \left[\begin{array}{c} v_1 \\ v_2 \\ i_x \\ \hline v_3 \\ v_4 \\ \hline i_k \end{array}\right] = \left[\begin{array}{c} -2 \\ 3 \\ 9 \\ \hline 0.4615 \\ 0.7692 \\ \hline 3 \end{array}\right] \tag{110}$$

The link matrix in this case has a single element. It is calculated from the pseudo-Thevenin impedances to produce the system of equations,

$$\left[\begin{array}{ccc|cc|c} 1 & 0 & 0 & 0 & 0 & -0.2 \\ 0 & 1 & 0 & 0 & 0 & -0.2 \\ 0 & 0 & 1 & 0 & 0 & 0.4 \\ \hline 0 & 0 & 0 & 1 & 0 & 0.2692 \\ 0 & 0 & 0 & 0 & 1 & 0.1154 \\ \hline 0 & 0 & 0 & 0 & 0 & 4.4692 \end{array}\right] \left[\begin{array}{c} v_1 \\ v_2 \\ i_x \\ \hline v_3 \\ v_4 \\ \hline i_k \end{array}\right] = \left[\begin{array}{c} -2 \\ 3 \\ 9 \\ \hline 0.4615 \\ 0.7692 \\ \hline 0.4615 \end{array}\right] \tag{111}$$

From the last equation, the link's current, is readily obtained as

$$i_k = \frac{0.4615}{4.4692} = 0.1033\,\text{A}$$

That value is then substituted in the other equations to determine the remaining voltages and current. This process produces the same results for nodal voltages and currents in voltage sources (links' and UVS) as the ones obtained at the beginning of this section from the network equations, as expected.

$$\left[\begin{array}{c} v_1 \\ v_2 \\ i_x \\ \hline v_3 \\ v_4 \end{array}\right] = \left[\begin{array}{c} -2 \\ 3 \\ 9 \\ \hline 0.4615 \\ 0.7692 \end{array}\right] - \left[\begin{array}{c} -0.2 \\ -0.2 \\ 0.4 \\ \hline 0.2692 \\ 0.1154 \end{array}\right] [0.1033] = \left[\begin{array}{c} -1.9793 \\ 3.0207 \\ 8.9587 \\ \hline 0.4337 \\ 0.7573 \end{array}\right] \tag{112}$$

7. SWITCHES AND ASYNCHRONOUS COMMUTATION

7.1 Introduction

When a switch operates, it alters the topology and size of the network. When a switch opens[1], it creates two nodes where there was only one. When a switch closes, it collapses one of its two nodes. In this chapter, representation of switches and their associated switching operations in OVNI are presented. The pros and cons of node collapsing are revised.

In real time simulations of the kind targeted in this work, the calculated samples of some signals are issued to the external devices[2] in an evenly time-spaced stream of samples. More often than not, open switching operations do not occur at the moment of issuing the samples, i.e. asynchronous commutation, Fig. 63. A technique to cope with the voltage or current *spikes* generated by those asynchronous opening of switches is introduced in this chapter.

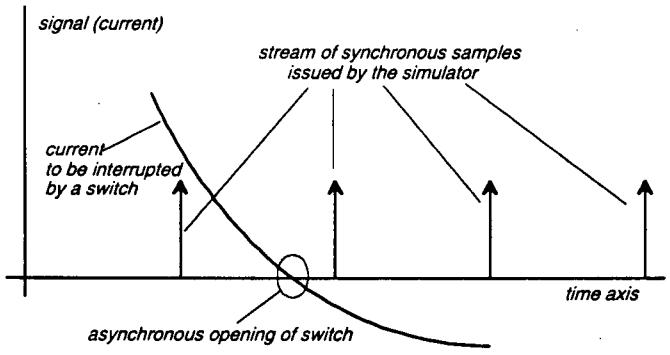


Fig. 63: Samples output stream, and asynchronous commutation.

[1] An ideal switch.
[2] D/A, amplifiers, etc.

7.2 Switch Closing, Collapsing Nodes

If switches are modelled as either ideal conductors —when closed—, or as perfect insulators —when opened—, the general topology of the network (as reflected in the system's matrix) is modified with each switching operation. That is, the number of nodes in the problem is reduced each time an ideal switch closes, and viceversa. In non real-time simulations, such situations can be exploited to speed-up those simulation intervals when switches are closed. In this case, the system becomes somewhat smaller and so does its matrix, which is now easier to triangularize or to invert, as necessary, Fig. 64.

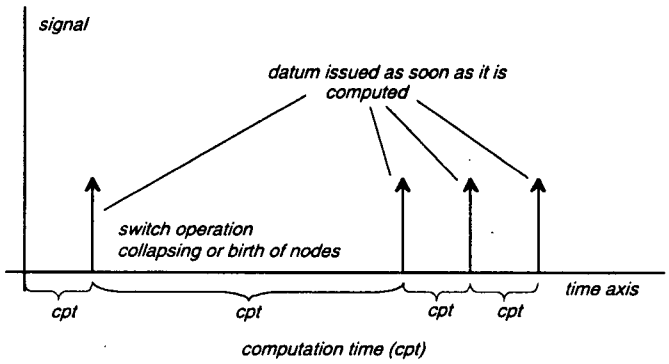


Fig. 64: Short and long integration steps. Non real-time simulation. Data are issued as soon as they are available.

The integration step that takes the longest time to compute is the one that takes precedence over all others in a real-time simulation. That is, to preserve the frequency spectrum of the output signal channeled through the digital-to-analog converters, amplifiers, and out to the real world, samples are issued at equally distanced intervals along the time axis, as in Fig. 65. In that figure some integration steps take longer to compute than others (*long* steps), but there is always a *filler* time slice added to wait for the real time deadline. That *filler* is used by the hardware to transmit the data.

It follows that a main target in this project has been the reduction of the *long* integration step depicted in Fig. 64. Precalculation of matrices, as was seen

in sections 5.3 to 5.10, was advanced with such a goal in sight. The collapsing —or re-insertion— of nodes introduces an overhead on the *long* integration steps that, in theory, could be compensated by the reduction in computation burden during the *short* integration steps. However, as the length of that *long* step is the determining factor of the bandwidth of the simulation, that overhead becomes overwhelming.

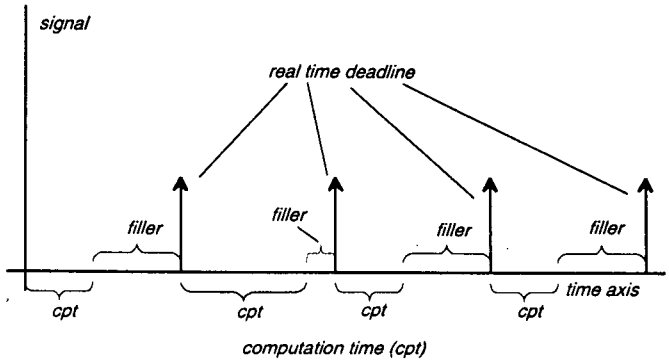


Fig. 65: Short and long integration steps. Filler time slices. Data output stream in a real time simulation.

Thus, in real time simulations it may not be to our advantage to reduce the number of nodes and the order of the system's matrix. In fact, such a re-arrangement is a costly one because of the management overhead (i.e., nodes reallocation, matrices re-dimensioning, and so). Also, in our efforts, addresses are sometimes precalculated for components in structures and arrays, and off-setting such positions in memory, when the number of nodes is reduced —or increased— during the simulation, carries with it a penalty in execution time terms.

The approach used has been instead to distribute the computational burden more evenly over the integration steps. The *short* steps become longer, but the dominant *long* steps become much shorter, with an improved simulation bandwidth as a result. In short, the size of the matrices, and the number and position of allocated nodes, remains unchanged along the simulation as seen in § 7.4.

7.3 Expanding a System of Linear Equations

As a basic framework, let us consider the possibility of introducing additional pseudo equations —and their corresponding pseudo unknowns— into systems of linear equations. The added equation will introduce a *repeated unknown*, that is, an unknown that is already in the system and associated with an existing equation. In this way, the new pseudo unknown solution value equates the value of the unknown it is mirroring. Let us clarify this with an example.

Consider a system of algebraic linear equations represented by the matrix equation in Eq. (113). The system's solution is included to the right of the equation.

$$\begin{bmatrix} 10 & -5 & -3 \\ -5 & 7 & -1 \\ -3 & -1 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ -6 \end{bmatrix} \Rightarrow \begin{array}{rcl} x_1 &=& 1.5364 \\ x_2 &=& 1.0927 \\ x_3 &=& -0.0331 \end{array} \tag{113}$$

Let us introduce a pseudo unknown, x_4, that mirrors x_2. This is done by means of a fourth equation whose mutual terms with all the equations but that of the mirrored unknown are the same as in the original equation. The new equation has no coupling with the original equation and viceversa. The coefficient of the pseudo unknown in the new equation is equal to the coefficient of the mirrored unknown in the original equation. The expanded system is shown in Eq. (114).

$$\begin{bmatrix} 10 & -5 & -3 & 0 \\ -5 & 7 & -1 & 0 \\ -3 & -1 & 9 & 0 \\ -5 & 0 & -1 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 0 \\ -6 \\ 0 \end{bmatrix} \Rightarrow \begin{array}{rcl} x_1 &=& 1.5364 \\ x_2 &=& 1.0927 \\ x_3 &=& -0.0331 \\ x_4 &=& 1.0927 \end{array} \tag{114}$$

As expected, the value of the pseudo unknown x_4, is the same as that for the legitimate unknown x_2. The introduced pseudo unknown could well be the voltage of a would-be collapsed node *four*, and as such it would share the same voltage as node *two*.

7.4 Closing a Switch without collapsing a Node

In the previous section, the possibility for introducing fictitious equation-unknown pairs that mirrored equations-unknowns already in the system was presented. In this section, that possibility is used to keep constant the dimensions of the network matrices when there is a switching operation. This constancy allows for a simplified and more efficient addressing scheme for use of precalculated matrices in the subblocks of the network.

In a network, when an open switch between nodes i and j closes, the only two equations to modify are the equations for those two nodes. The process can be summarized more clearly in pseudo code as follows. Let $[A]$, be the nodal analysis bus conductance matrix associated to an n node network. If nodes i and j are welded together by the closing of a switch, each element a_{kp} of matrix $[A]$ changes according to the process described in Fig. 66.

```
for k = 1 ... n; that is, for every row k
  if k ≠ i and k ≠ j then
    a_ik ← a_ik + a_jk
    a_jk ← a_ik
  endif
endfor
a_ii ← a_ii + a_jj − 2a_ij
a_jj ← a_ii
for every i, j|i ≠ j do
  a_ij ← a_ji ← 0.0
```

Fig. 66: Closing a switch between nodes i and j.

Coefficients for self terms for both nodes i and j, a_{ii} and a_{jj}, become the sum of their former values minus the former coupling between the two nodes, a_{ij} and a_{ji}. Then, the coupling between the two nodes becomes zero and all other elements in both equations are now the sum of the equations' corresponding coefficients.

7.4.1 A Numerical Example

In this section, a numerical example illustrates the procedure described in § 7.4 to avoid collapsing nodes when a switch bridging them closes. Consider the circuit in Fig. 67, with a switch between nodes 2 and 4 originally open.

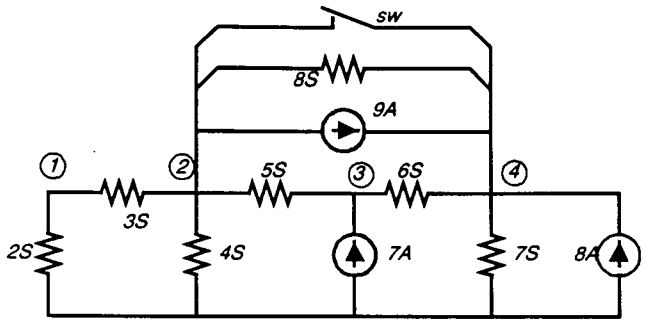


Fig. 67: Case to illustrate how to avoid collapsing nodes.

Let us begin writing the nodal equations before the switch closes. All four nodes display linearly independent equations, Eq. (115).

$$\begin{bmatrix} 5 & -3 & 0 & 0 \\ -3 & 20 & -5 & -8 \\ 0 & -5 & 11 & -6 \\ 0 & -8 & -6 & 21 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 0 \\ -9 \\ 7 \\ 17 \end{bmatrix} \tag{115}$$

Once the switch closes, if we choose to *collapse* the two nodes connected by the switch, 2 and 4, into a single one, 2, the network has now only three nodes, and its nodal equations are in Eq. (116). The solution to this system is on the right of the equation.

$$\begin{bmatrix} 5 & -3 & 0 \\ -3 & 25 & -11 \\ 0 & -11 & 11 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \\ 7 \end{bmatrix} \Rightarrow \begin{cases} v_1 = 0.7377 \text{ V} \\ v_2 = 1.2295 \text{ V} \\ v_3 = 1.8659 \text{ V} \end{cases} \tag{116}$$

Applying now the procedure that was described in § 7.4 to keep constant

both the number of nodes and the dimension of the matrix in Eq. (115) produces Eq. (117) that correctly predicts that the voltages of nodes 2 and 4 will be equal once the switch is closed. Thus we have

$$\begin{bmatrix} 5 & -3 & 0 & 0 \\ -3 & 25 & -11 & 0 \\ 0 & -5 & 11 & -6 \\ -3 & 0 & -11 & 25 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \\ 7 \\ 8 \end{bmatrix} \Rightarrow \begin{cases} v_1 = 0.7377 \text{ V} \\ v_2 = 1.2295 \text{ V} \\ v_3 = 1.8659 \text{ V} \\ v_4 = 1.2295 \text{ V} \end{cases} \tag{117}$$

Let us now recapitulate. It goes without saying that solving the smaller system in Eq. (116) is simpler that solving the larger system in Eq. (117). However, as it is the size of the larger of the two the one that imposes its weight on the bandwidth of the real time simulation and, more important, it is the additional burden of building, triangularizing, and changing addresses for the smaller matrix that is being avoided here. OVNI uses the constant size subblock procedure in § 7.4 in its preprocessing stage to generate and prestore constant size subblock matrices of the type of the one in Eq. (117).

7.5 Switch openings

In power networks, when an AC-switch is signaled to open, it waits until the next time that the current through it goes through zero[3], see Fig. 68.

In EMTP simulations the detection of the *zero crossing* occurs when the current through the switch waiting for opening changes sign, at b in Fig. 69. To avoid computational overhead, the actual zeroing of the current through the switch is not made until the next integration step after the *zero crossing* is detected, at c in Fig 69. This is an acceptable and efficient solution, given small enough integration steps, in most cases. In power electronics circuits, however, and in situations where the slope of the current just before the change of sign is large, this approach triggers spurious voltage spikes in highly inductive

[3] We will refer to this moment as the *zero crossing* of the current in the switch.

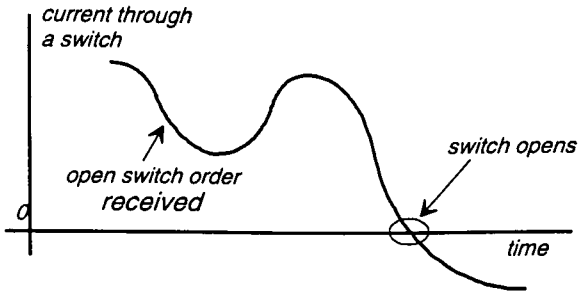


Fig. 68: Switch opening event: signal, and actual opening.

neighbouring networks.

To perceive the way such spurious spikes come to be, consider a situation where the current slope in Fig. 69 between points a and b is very steep. When the *zero crossing* is detected at point b, the value of the current has already drifted far away from zero[4]. If that value at b is issued to the rest of the network as the current in the switch, when the current is zeroed at c, the effective derivative of the current will be too big. That high current derivative is bound to produce voltage spikes in nearby inductive elements. Such was the case of the HVDC

[4] Actually, under a rapid changing current situation like this, the current at a is also far from zero, and nulling the current at b would still produce voltage spikes, but those would be legitimate voltage spikes that should appear in the actual circuit.

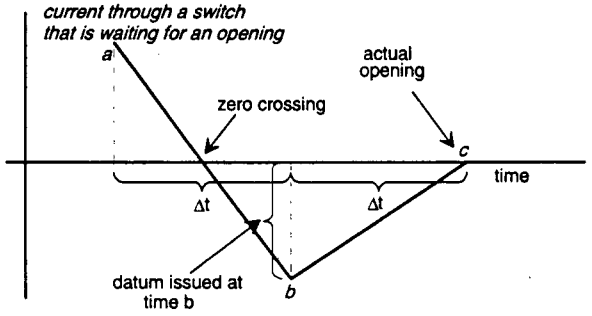


Fig. 69: Zero crossing and actual opening of a switch.

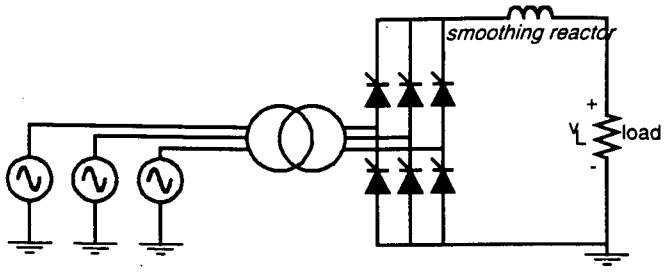


Fig. 70: Six valve rectifier circuit.

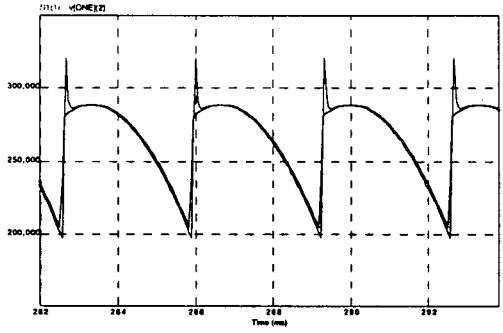


Fig. 71: Voltage before smoothing reactor.

rectifier bridge illustrated in Fig. 70, where the voltage at the load, and the spikes produced by the solution method, are illustrated in Fig. 71.

In short, the simulator has to honor the request to open the switch as soon and exactly at the point where the zero crossing occurs. As the zero crossing is not evident to the simulator until the change of sign is detected, the zero crossing will already be in the past. A possible solution, in non real-time simulations, is to *backtrack* to the actual moment when the zero crossing occurred [57, 58], and issue the data at that particular moment in time, with the time stamp of the actual zero crossing itself. The result is a shortened integration step right at the opening of the switch. After that, the simulation proceeds at the regular

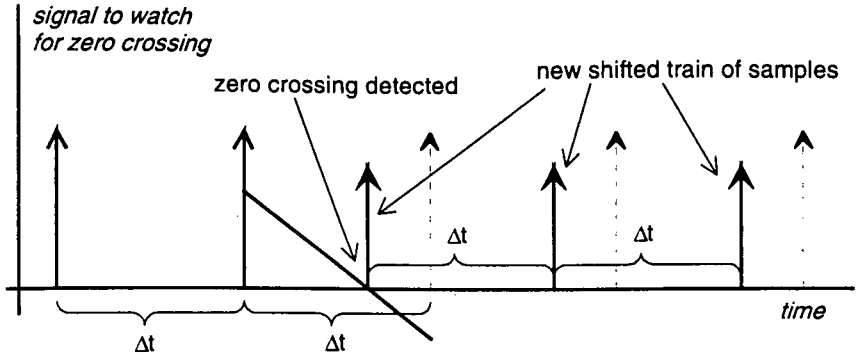


Fig. 72: Non real time backtracking.

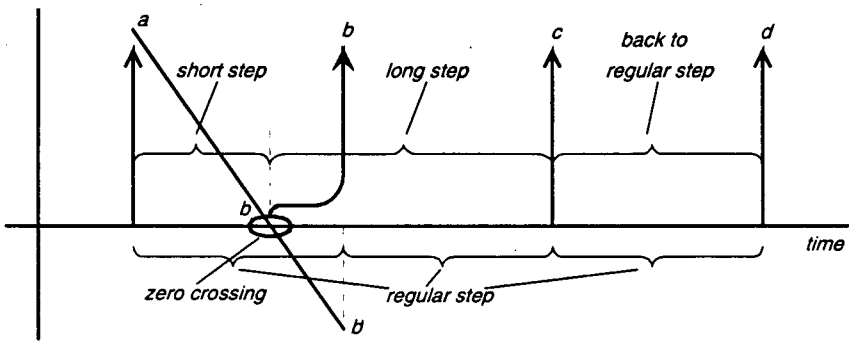


Fig. 73: Simple non regressive backtracking.

integration step; i.e., all future samples are slightly *shifted* to the left, Fig. 72.

7.6 Asynchronous Commutation in OVNI

In real-time simulations, however, it is not possible to go back in time, to backtrack. In OVNI a compromise was made, see Fig. 73. Instead of releasing the completely wrong value at b', a lesser evil approach is taken, the correct value at the zero crossing, $Y(b)$ is issued slightly later, at b'.

But even this can be too expensive. To obtain the data at the zero crossing, b in Fig. 73, not only the trivially zeroed current in the switch in necessary, but

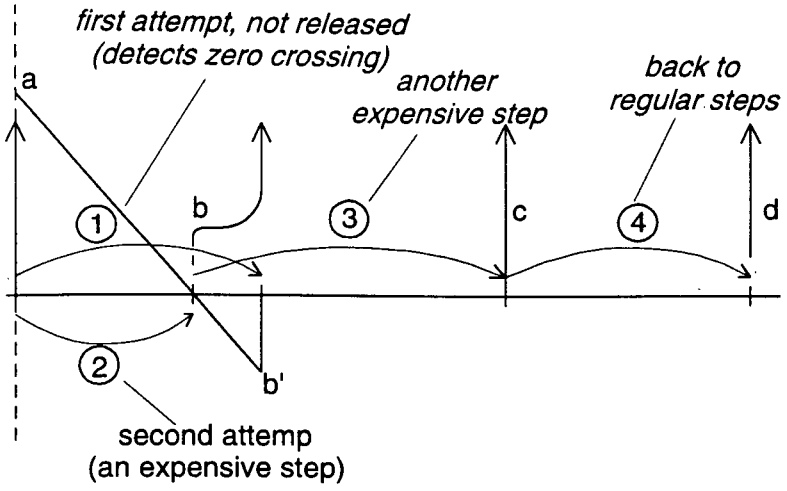


Fig. 74: Accurate but too expensive backtracking.

the voltage at every node, and the history sources —and any other sources as well— at the same point b. In short, we could go back to a, and advance by the now known smaller Δt_{short} to point b.

But to do this, the network matrices would need recalculation since they depend on the integration step size. As the reduced integration step Δt_{short} size is not known before the simulation, such matrices cannot be precalculated. That is, two complete step computations are necessary to produce the data to be issued at b —one of those computations is even more expensive than a regular one—. As a result, the bandwidth of the simulation is likely to fall to half its targeted value —far less, actually, given the additional overhead of matrix calculation and triangularization or inversion—.

After the data just calculated, at b, is issued at b', the simulator would have to advance the enlarged integration step Δt_{long} to fall back in step (at c) with the real time samples stream, Fig. 73. This results in another expensive recalculation of matrices. See Fig. 74

A simpler and shorter approach is taken by OVNI. A linear interpolation of voltages and history source values between points a and b' produces the state

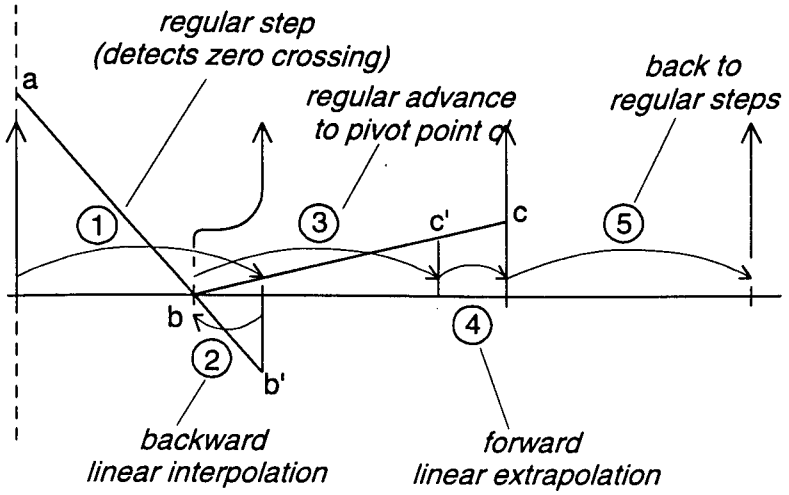


Fig. 75: BIFE: Backward interpolation, forward extrapolation.

of the network at b, the zero crossing, much faster. The problem is now how to advance from b up to c in Fig. 73. One possibility, see Fig. 75, is to use the available matrices for Δt and advance computationally from b to c', and then use linear extrapolation to reach the values needed at c, where they are issued. Then the simulation resumes. Such a solution produces satisfactory results and was reported in [35]. A disadvantage with this technique, however, is the prediction involved in the procedure (even though small). An improvement to that technique, which does not involve prediction is presented next. This technique will be called the inverse *Critically Damped Adjustment*, inverse *CDA* or "*ADC*, or simply "DSDI" (Double Step Double Interpolation). The process is described below.

7.6.1 Double Step-Double Interpolation, *DSDI*

OVNI uses, as was seen in § 3.5, the backward Euler integration rule[5] to discretize the equations of the network.

From chapter 4, let us compare the discrete time equivalent conductance of

[5] Abbreviated in what follows as *BE*.

an inductor, L, in a simulation with an integration step, Δt, where *BE* was used to discretize the differential equations, Eq. (118),

$$g_{BE} = \frac{\Delta t}{L} \tag{118}$$

with the equivalent conductance produced by the trapezoidal rule of integration[6], for the same inductor using the same integration step, Eq. (119).

$$g_{TR} = \frac{\Delta t}{2L} \tag{119}$$

It follows that, for the inductor, *TR* produces a conductance half the value of that produced by *BE*. But this can also be interpreted as *if one uses an integration step twice as big with TR than with BE, both rules produce the same equivalent conductance [10].* This situation applies for all discretizations in the network and we can say that *if one uses an integration step twice as big with TR, than with BE, both rules produce the same network matrices.*

Up to the zero crossing, OVNI has been integrating with BE, and its associated precalculated network matrices. OVNI is at b—after the backtracking obtained with linear interpolation between a and b'—. Now, using now a double sized integration step, and TR as integration rule, it advances past c, up to c' in Fig. 76 with the same precalculated matrices already available. The next *output* point, c, is reached by a safe interpolation between b and c'.

7.7 DSDI's OVNI Modified Tasks Schedule

To accommodate for the *double step double interpolation* scheme (DSDI), the solution tasks described in § 5.2 need to be revised and extended. In particular, the updating of history sources —as a request issued by the simulator to the element models— has to include additional functionality, as follows.

Refer to Fig. 77, where a flowchart of the tasks of the simulator, which includes DSDI, is illustrated. The best way to describe the operation of the DSDI as implemented by OVNI is to go through the flowchart. First —assuming that

[6] The trapezoidal rule of integration is referred to in what follows as *TR*.

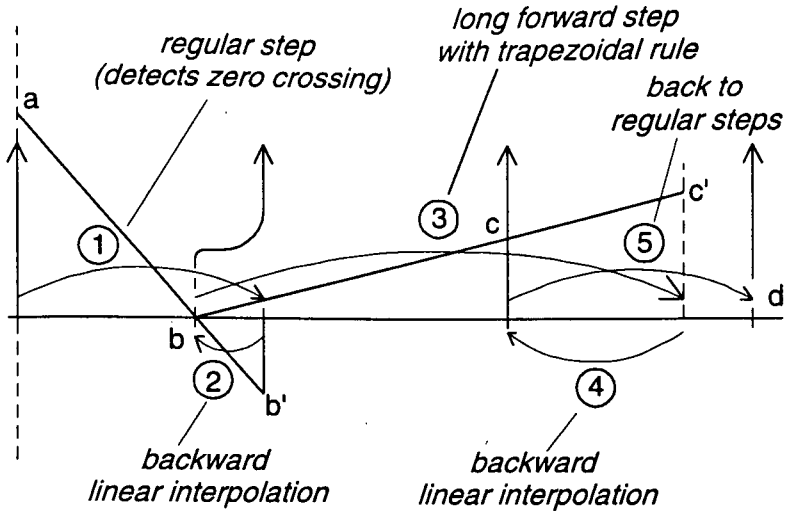


Fig. 76: DSDI used in OVNI. The most expensive step takes one regular integration step with precalculated matrices, plus one inexpensive linear interpolation.

all history and independent current sources have been evaluated already, or given initial values if this is the first time step—, accumulate nodal currents into the vector $[\Sigma h]$. Second, solve the nodal system of equations, $[G][v] = [\Sigma h]$, for the external nodes voltages, v. Third, check to see if this is the time for a double step —that is, if flag *interpolateDoubleStep*[7] is set—. Let us assume, in this first run, that this is not the case, that this is a regular step. Fourth, update element history sources for the next step. Then determine internal node voltages, and let the elements check for internal switch opening operations. If such an event occurs, the corresponding element sets a flag *interpolate*[8], and determines the percentage of backtracking necessary to hit the exact point where the zero crossing occurred in the current through the just opened switch. That percentage[9] is *bt*. As part of the same block in the flowchart, a separate method is activated, that of checking for switch opening events in the prescheduled events

[7] Represented in the flowchart by an asterisk enclosed into a circle.
[8] Represented in the flowchart by an asterisk.
[9] Actually it is a per unit value. See next section for a detailed discussion of this item.

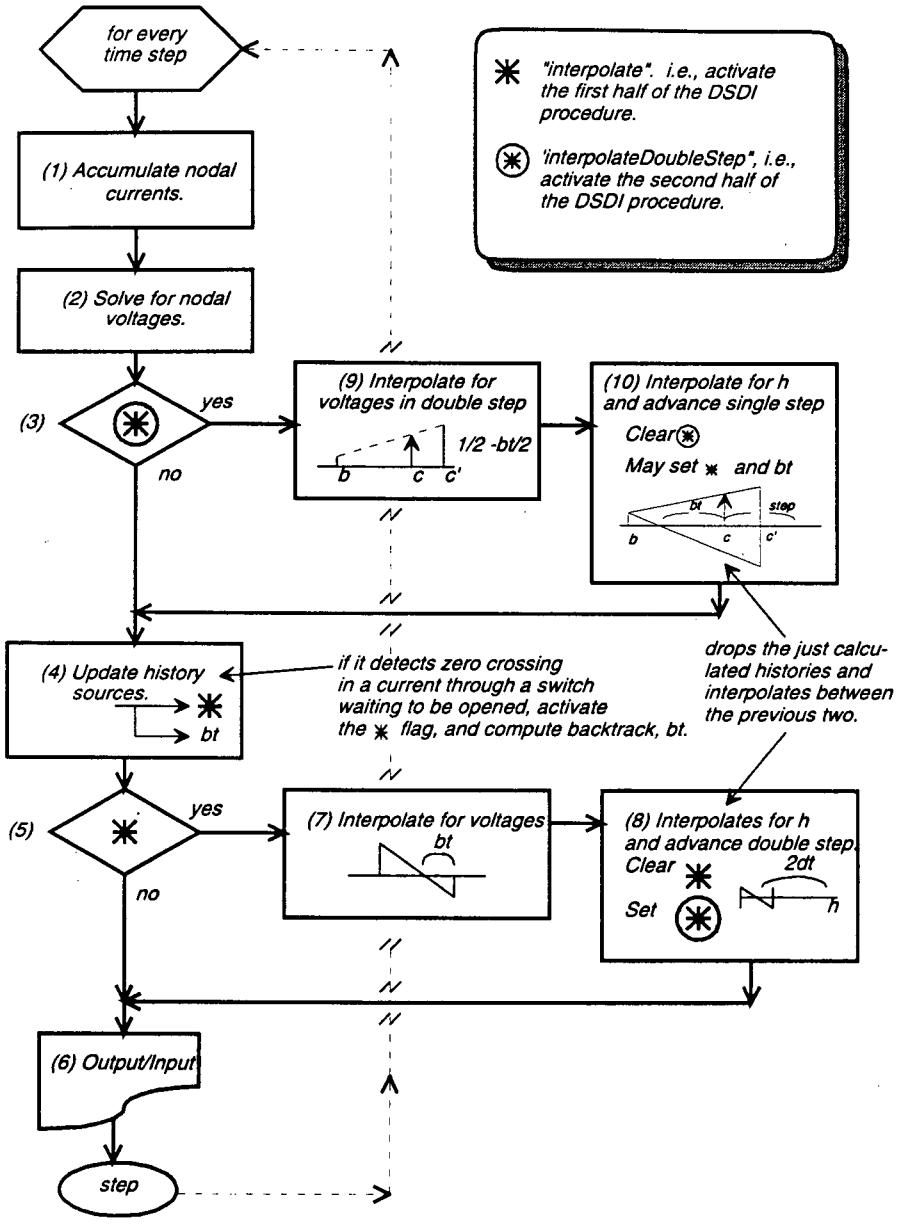


Fig. 77: OVNI's modified flowchart to include DSDI. Elements handle three instances of their histories: h_{next}, h_{now}, h_{before}. When they "decide" to backtrack, they discard h_{next}, and interpolate between the other two.

list —either with the input data case, or by a controlling device under test—. If an external switch opening is detected, a corresponding *bt* backtracking is calculated, along with the setting of the same *interpolate* flag. Fifth, check if the *interpolate* flag is set, that is, if a switch opening operation was encountered in the previous task. In this first run let us follow the main path of the flowchart; i.e, assume no switch opening was met, then: Sixth, the node voltages and any other output variables are made available for the D/A converters and amplifiers for output. Input logical signals from the real world are received and the corresponding switching events are scheduled by the event handler.

Let us assume instead that a switch opening occurred in the fourth task above, then the test in the fifth task will branch the execution into: Seventh, if several switches opened, use the backtracking percentage *bt*, corresponding to the zero crossing that occurred the last. Using the last two calculated values for nodal voltages, interpolate for the ones corresponding to the moment of the *chosen* zero crossing. Eighth, request each element to discard the most recently calculated history sources values, and interpolate between the previous two values —which the element has to keep at each time step—. The element interpolates too for internal nodes voltages and from that interpolated point in time updates the history source advancing with the formula corresponding to the trapezoidal rule of integration; i.e., a double step (step (3) in Fig. 76. The integrator clears the *interpolate* flag, and sets the flag *interpolateDoubleStep*. Now, task number six outputs the voltages and signals just interpolated (value at point b' in the figure of reference).

Now the simulator is at the top of the flowchart again, but with a set *interpolateDoubleStep* flag. It goes through tasks number one and two, and obtains external node voltages at point c' in Fig. 76. When the simulation reaches task three this time, it branches into tasks number nine and ten, following a clearing of flag *interpolate* by the simulator, it performs a new double set of interpolations —but using not *bt*, but $\left[\frac{1}{2} - \frac{bt}{2}\right]$—: one interpolation for external node voltages, and another for the elements history sources and internal node voltages. From

the interpolated values of their history sources, and using the backward Euler's rule of integration, the elements advance a single step and reenter synchronism with the output stream of data. The elements, and the switching events handler, check for any switch opening occurring between the point b in Fig. 76, and the recently interpolated values. If a switch opening condition is met, the corresponding element at c, or the switching events handler, reactivates the flag *interpolate*. The simulator now clears flag *interpolateDoubleStep*, which brings the simulation back either to the *normal* backward Euler's single stepping —if *interpolate* was not just set—, or to the first half of the DSDI procedure —if an AC switch opening was just detected and the *interpolate* flag was activated—.

7.8 Single Step and Double Step Interpolation Details

Only AC switches waiting for an opening operation have their currents monitored for a zero crossing, either inside the model where they reside, or among the corresponding block's links, or even perhaps with one of a subblock's switches. DC switches operate synchronously with the simulation stream and are not subject to the problems tackled by the DSDI procedure.

When an AC switch has been "marked" for opening, its model (if it is part of one), or the switching events handler of the simulator (if it is not), keeps a computational eye on its current waiting for a zero crossing. Such a current will be referred to, in what follows as the *reference current*. In Fig. 78a, a zero crossing in a reference current has just occurred between points a and b; i.e., between values r_a and r_b[10]. How far back into the last integration step the zero crossing, and the interpolation, will have to go is given by the backtracking, bt.

$$bt = \frac{r_b}{r_b - r_a} = \frac{-3}{-12} = 0.25 \tag{120}$$

Once a backtrack has been found necessary, either in tasks four or ten in

[10] In the first implementations of OVNI—before DSDI—, this zero crossing was detected by a painful extraction of the sign bit within the IEEE double precision floating point representation of both values, and a subsequent digital *and* operation. Given the relative timings of Intel's Pentium *fmul*, floating point multiplications, a simple test for $r_a \times r_b \leq 0$ is fast enough.

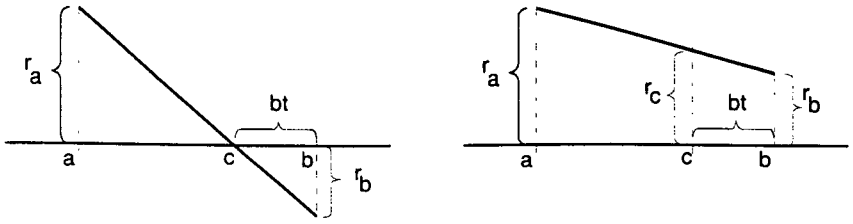


Fig. 78: Linear interpolation between points (a, r_a) and (b, r_b). bt is the per unit backtracking necessary.

Fig. 77, all other variables (voltages and history source values) are interpolated for in tasks seven and eight, according to Eq. (121) and Fig. 78b.

$$y_c = y_b + (y_a - y_b)bt \tag{121}$$

Then, at the next pass through the loop in Fig. 77, after voltages and history values have already been determined for the double step point d,

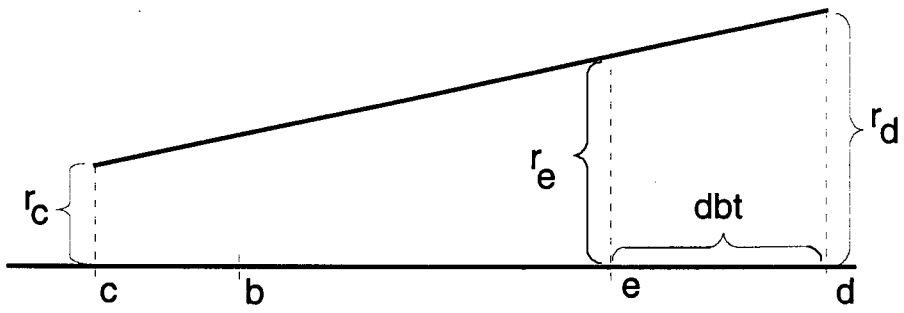


Fig. 79: Interpolation across the double step span.

in Fig. 79, an interpolation is performed in tasks nine and ten, with a modified backtrack factor dbt that relates to the available bt in Eq. (120) above according to Eq. (122), using the same form of Eq. (121). Thus

$$dbt = \frac{(1 - bt)\Delta t}{2\Delta t} = \frac{1}{2} - \frac{bt}{2} \tag{122}$$

$$y_e = y_d + (y_c - y_d)dbt \tag{123}$$

Finally, if during task number ten a new switch opening is detected between b' and c in Fig. 79[11], a modified backtracking factor, mbt is established in the same way as bt was obtained in Eq. (120).

[11] Not between b' and the point obtained at the end of the double step advance, point c' in Fig. 76.

Part IV

OVNI ELEMENT MODELS

8. OVNI ELEMENT MODELS

8.1 Introduction

Several new models developed during this project allowed for the testing of OVNI's performance under the two test cases targeted in chapter 2: protective relay testing, and HVDC controller testing. This part of the report describes those models.

The models are:

- *metal oxide varistors* (MOVs) already described in § 4.5.9 on page 51;
- *measuring transformers*, introduced in § 4.5.10 and detailed in the following two sections, § 8.2 and § 8.3;
- *HVDC modules*, detailed in chapter 9, included to illustrate the general format that element models developed for OVNI should follow, in particular that model shows how to implement the "node hiding" concept introduced in this thesis, § 5.13, inside a model to streamline the simulation;
- *a simplistic HVDC controller* model was developed only to explore the HVDC module functionality, and is described in chapter 10 as an example of an OVNI model that interacts directly with another element model, all within the frame of OVNI's solution.

8.2 Current Transformers

A non-iterative model for the current transformer (CT) that incorporates the saturation characteristics of the CT's core was presented in [53]. In this model, the secondary current of the CT, i_s, is calculated from the primary current, i_p

—determined by the integrator core—, and from the present saturation state of the CT magnetic core.

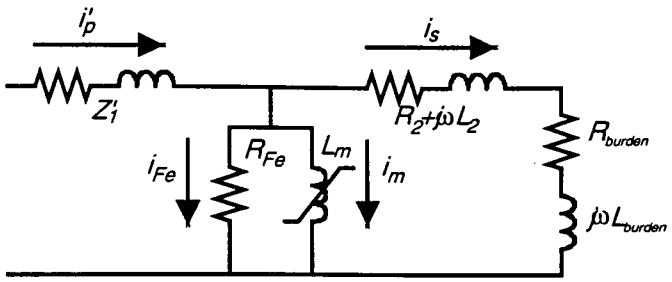


Fig. 80: Equivalent circuit of current transformer (minus the ideal transformer) referred to the burden side.

A. Equivalent Circuit

Figure 80 shows an equivalent circuit of the current transformer with all quantities referred to the secondary side. In that figure:

- i'_p= primary current referred to the secondary side.
- i_{Fe}= current in phase with the fundamental component of the voltage induced in the core; i.e., current through resistor R_{Fe}, for the approximation of iron core losses.
- i_m= magnetizing current through non-linear inductor.
- i_s= secondary current.

Since the CT perceives its primary current as applied by a current source, the primary leakage impedance, Z_1, does not affect the results; therefore it is not needed.

The current in the primary can be written as the sum of three component currents, Eq. (124). Each of those components can be expressed as a function of the flux linkages in the transformer core, λ, which leads to a single equation for i_s (the output of the model) as a function of i'_p (input from the integrator).

$$i'_p = i_{Fe} + i_m + i_s \tag{124}$$

B. Core Loss Branch

The voltage across the core loss resistance is also the voltage induced by the magnetic flux linkages, λ, in the core.

$$v = R_{Fe} \cdot i_{Fe} \tag{125}$$

$$v = \frac{d\lambda}{dt} \tag{126}$$

Integrating Eq. (126), then applying the trapezoidal rule to approximate the voltage integral, and finally substituting Eq. (124) into the resulting expression, the right-hand side of Eq. (126) becomes $(\lambda_{new} - \lambda_{old})/(\Delta t)$; and the left-hand side becomes $R_{Fe}(i_{Fe-new} + i_{Fe-old})/2$, where subscripts "new" and "old" refer to the values at the present time step t and the preceding time step $(t - \Delta t)$, respectively.

$$i_{Fe-new} = c_{Fe} \cdot \lambda_{new} + h_{Fe-old} \tag{127}$$

where h_{Fe-old} is a history term evaluated as follows

$$h_{Fe-old} = -c_{Fe} \cdot \lambda_{Fe} - i_{Fe-old} \tag{128}$$

where the constant coefficient is defined as

$$c_{Fe} = \frac{2}{R_{Re}\Delta t} \tag{129}$$

C. Magnetizing Branch

The non-linear relationship between magnetization current, i_m, and flux linkages, λ, for the magnetizing branch can be approximated by the piecewise linear curve, Fig. 81.

With the operating point in the linear segment starting at $(\lambda_{start}, i_{start})$, and defining L as the slope of that segment,

$$i_m - i_{start} = \frac{1}{L} \cdot (\lambda - \lambda_{start}) \tag{130}$$

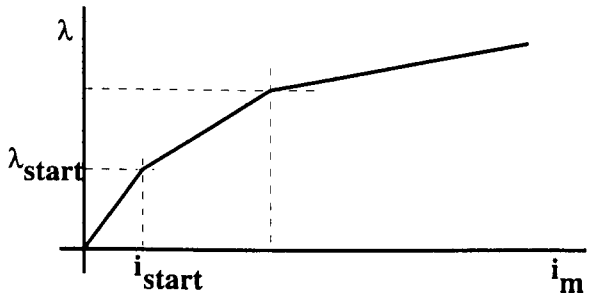


Fig. 81: Piecewise linear representation of magnetization in flux path.

Defining the known constant k_m for each segment, as in Eq. (131), Eq. (130) can be rewritten as in Eq. (132).

$$k_m = i_{start} - \frac{\lambda_{start}}{L} \tag{131}$$

$$i_m = \frac{1}{L} \cdot \lambda + k_m \tag{132}$$

D. Secondary side branch

If the secondary leakage impedance is combined with the burden into a total secondary impedance

$$R_s + j\omega L_s = (R_2 + R_{burden}) + j\omega(L_2 + L_{burden}) \tag{133}$$

the voltage v, in Eqs. (125) and (126) is also

$$v = R_s i_s + j\omega L_s \cdot \frac{di_s}{dt} \tag{134}$$

Eliminating v from Eqs. (134) and (126), integrating the resulting expression and applying the trapezoidal integration rule

$$\frac{\lambda_{new} - \lambda_{old}}{\Delta t} = R_s \cdot \frac{i_{s-new} + i_{s-old}}{2} + L_s \cdot \frac{i_{s-new} - i_{s-old}}{2} \tag{135}$$

Defining the history term h_s as in next equation, we can solve for i_{s-new} as in Eq. (137) and obtain

$$h_{s-old} = -c_s\lambda_{old} - d_s i_{s-old} \quad (136)$$

$$i_{s-new} = c_s\lambda_{new} + h_{s-old} \quad (137)$$

with the two constants

$$c_s \equiv \frac{1}{L_s + \frac{R_s\Delta t}{2}} \qquad d_s \equiv c_s\left(\frac{R_s\Delta t}{2} - L_s\right) \quad (138)$$

E. Secondary current as function of primary current

From Eqs. (124, 127, 132 and 137) we obtains

$$i'_p = i_{Fe} + i_m + i_s = \left(c_{Fe} + \frac{1}{L} + c_s\right)\lambda + (h_{Fe} + k_m + h_s) \quad (139)$$

Express the flux linkages, λ, as a function of the secondary current, i_s, from Eq. (137), and the desired expression of $i_s = f(i'_p)$ is obtained as follows

$$i_s = k_1\left(i'_p - h_{Fe} - h_m - h_s\right) + h_s \quad (140)$$

where k_1 is the constant defined as

$$k_1 = \frac{c_s}{c_{Fe} + \frac{1}{L} + c_s} \quad (141)$$

If the history terms (h'_s) are known from values at the preceding time step, the secondary current can be obtained from the primary current from Eq. (140) with only one multiplication and four additions.

F. Updating history terms

Once the new secondary current i_{s-new} has been calculated at time t, all history terms need to be updated to advance the solution by Δt. In the updating calculations, the term $c_s \cdot \lambda_{new}$ is used instead of λ_{new}, and is obtained from Eq. (137),

$$c_s\lambda_{new} = i_{s-new} - h_{s-old} \quad (142)$$

The history term for the secondary side branch follows from Eq. (136),

$$h_{s-new} = -c_s \lambda_{new} - d_s i_{s-new} \tag{143}$$

The history term for the core loss branch is obtained from the formula

$$h_{Fe-new} = k_{Fe}\,(c_s \lambda_{new}) - h_{Fe-old} \tag{144}$$

which follows from Eq. (128) when i_{Fe-new} is replaced with its expression from Eq. (127). The constant k_{Fe} is defined as

$$k_{Fe} \equiv -\frac{2c_{Fe}}{c_s} \tag{145}$$

These updating formulas add another two multiplications and three additions to the effort required in each time step, for a total of three multiplications and seven additions. There is also a check needed to see if $c_s \cdot \lambda_{new}$ in Eq. (143) has moved the operating point into another segment in the piecewise linear representation of the magnetization curve.

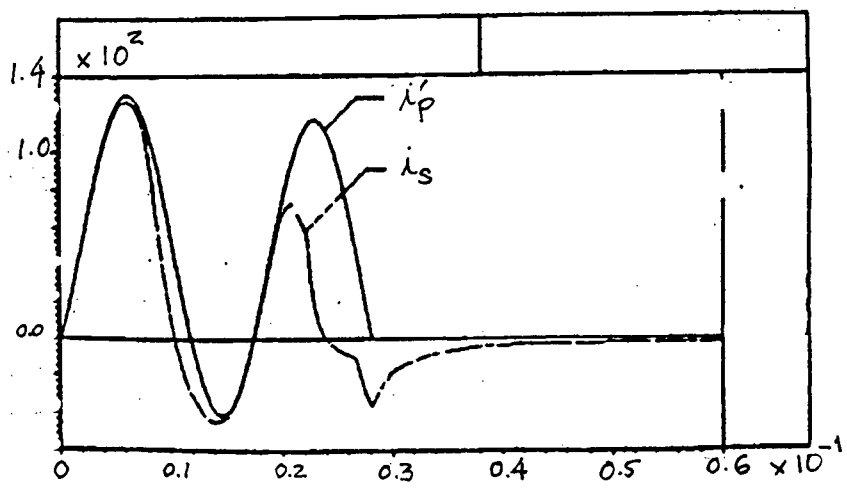


Fig. 82: Secondary current match between OVNI's model and EMTP's discrete elements one.

G. Validation of the model

Figure 82 compares the result of this algorithm with the one obtained with the standard EMTP solution method. Both answers are practically identical.

The case used for this test was taken from a field test comparison described in [59], where six segments were used to represent the magnetization curve. For the duplication of test results described in [60], simulation results with a two-segment representation were almost as accurate as those from more detailed representations. In the two-segment case, the *knee point* seems to carry more weight than all other parameters of the saturation curve.

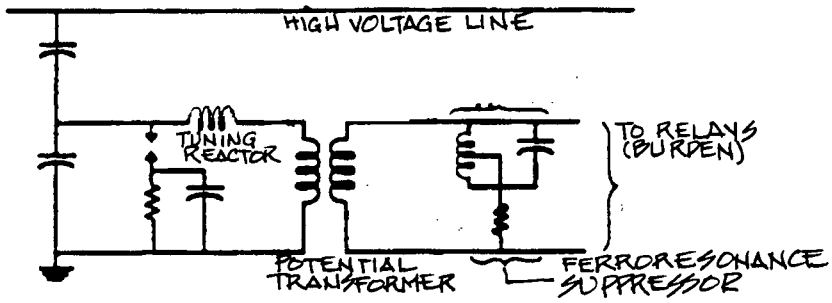


Fig. 83: Coupling Capacitor Voltage Transformer, CCVT.

8.3 Coupling-Capacitor Voltage Transformers

Figure 83 shows a simplified schematic of a coupling capacitor voltage transformer. A detailed wide frequency band model of a CCVT is complicated due to the magnetic and capacitive interactions in the various parts of the component magnetic devices (tuning reactor, potential transformer, and ferroresonance suppresor) [61, 62].

8.3.1 Potential transformer and reactors

Figure 84 shows a lumped parameter equivalent circuit for a single phase two-winding transformer [50]. Terminals 1–3 are input, and 2–4, output. This model is valid for frequencies up to hundreds of kilohertz. Several stray capacitances in-

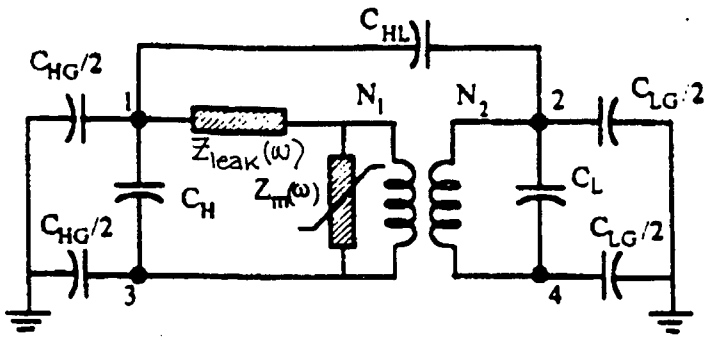


Fig. 84: Lumped parameter high frequency equivalent circuit of a two winding transformer.

side the device have been included: winding to winding (C_{HL}), turn to turn (C_H, C_L) and winding to ground (C_{HG}, C_{LG}), together with the frequency dependent leakage impedance ($Z_{leak}(\omega)$) and the core magnetization branch ($Z_m(\omega)$) which is possibly nonlinear and frequency dependant.

It is convenient to relocate the core magnetization branch across the outside terminals of the winding closest to the core (usually the low-voltage winding). Then this branch can be modelled in as much detail as desired and allowed by the simulation time constraints. Saturation and hysteresis characteristics of the branch can influence the low frequency response of the solution [63].

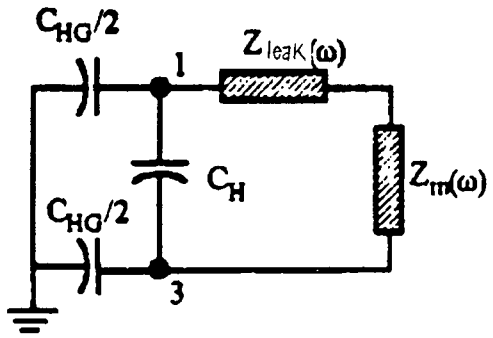


Fig. 85: High frequency model of a reactor.

In devices, like the tuning reactor and the ferroresonance suppressor in

Fig. 83, that have only one coil, the equivalent circuit, Fig. 85, is *half* the equivalent circuit for a two winding transformer shown in Fig. 84. In a reactor the *magnetization* branch Z_m is the main impedance in the circuit. However, reactors are designed and built so that they do not saturate and both Z_{leak} and Z_m can be modelled as linear frequency dependent R-L branches. Those branches can be synthesized in a similar way as Z_{leak} in the two winding transformer.

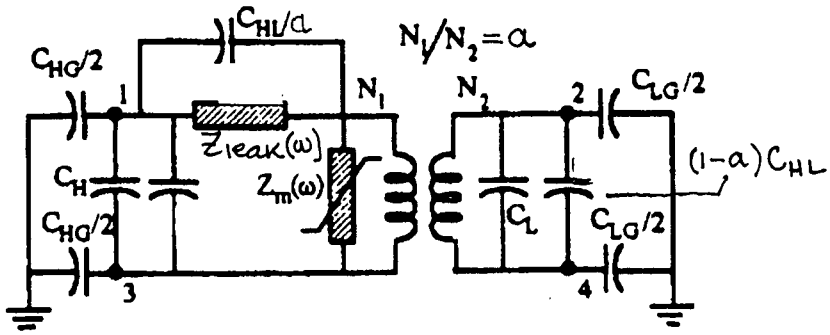


Fig. 86: High frequency equivalent circuit for a two winding transformer.

8.3.2 Simplified equivalent circuit

A circuit transformation can be used to move C_{HL}, the capacitance that bridges both sides of a two winding transformer, to one side [64], as in Fig. 86. The resulting circuit has a capacitance in parallel with Z_{leak} plus additional capacitances in parallel with C_H and C_L.

Once the *outermost* capacitances at ports 1–3 and 2–4 have been removed (computationally), the impedance Z_{leak} in parallel with C_{HL}/a is the short circuit impedance measured at a short circuit test. Chimklai and Martí [50] present a method to obtain, from simple measurements, the various capacitances in the equivalent circuit, as well as the short circuit impedance Z_{short} (Z_{leak} in parallel with C_{HL}/a). In Fig. 87, a typical measured short circuit impedance response can be seen.

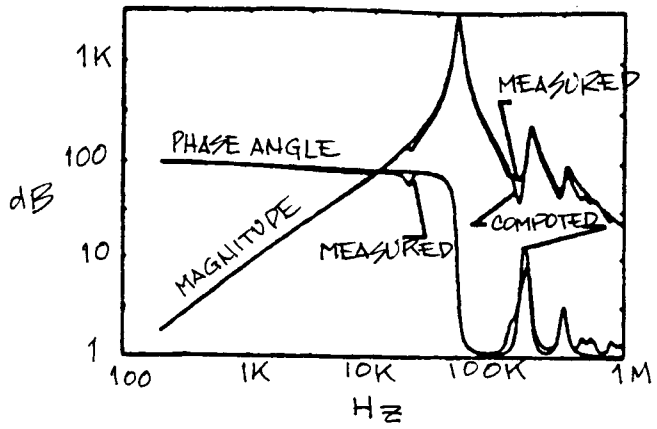


Fig. 87: Frequency response (Z_{short}) of a two winding transformer. Measured and synthesized responses.

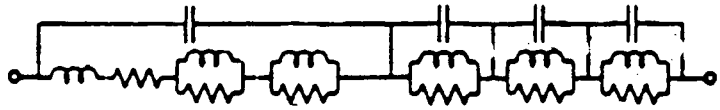


Fig. 88: Synthesized RLC network used to approximate $Z_{short}(\omega)$, multiple peak high-accuracy synthesis.

It is shown in [50] that $Z_{short}(\omega)$ (Fig. 87) can be matched very accurately with a number of RLC blocks as in Fig. 88, one block per resonant peak.

8.3.3 CCVT model for real-time simulation

A very accurate CCVT model can be obtained by combination of the PT model described above with corresponding models for the tuning reactor, ferroresonance suppression circuit and the C's of the capacitive divider, in a similar manner to that suggested in [63]. However, in the suit of tests targeted by this simulator, the accepted bandwidth (once the distortion of the integration rule has been accounted for) is only of 2 to 4 kHz.

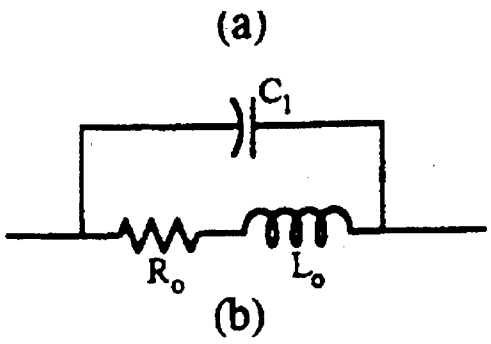


Fig. 89: Simplified model to represent only the main peak delivers acceptable accuracy.

It was considered that under these conditions it is sufficient to approximate the first resonant region in Fig. 87. A very reasonable approximation of this region can be achieved with a simple RLC combination, as in Fig. 89. In this minimal approximation, R_0 and L_0 can be taken as the 60Hz values, while C_1 is calculated to match the first resonance peak.

This procedure delivers a two port model that is independent of the burden. For cases where the burden is known, see § 8.3.4 for a convenient and efficient alternative.

8.3.4 Potential Transformer Model, PT

For cases where value of the burden to the PT is known, a simpler approach is used. This model for the potential transformer (PT), used in OVNI, was presented in [65]. The model approximates the PT's non-flat frequency response in Fig. 90 by a two-pole transfer function of the form

$$\frac{V_{out}(s)}{V_{in}(s)} = k \cdot \frac{s}{(s+p_1)(s+p_2)} \tag{146}$$

The output voltage is computed, in the time domain, as a function of the input voltage and the magnetization history of the PT's core. From the frequency response in Fig. 90, the two finite poles are: $p_1 = 251\,\text{rad/sec}$ and $p_2 = 628\,\text{rad/sec}$. The constant $k = 950$. Figure 91 shows the magnitude re-

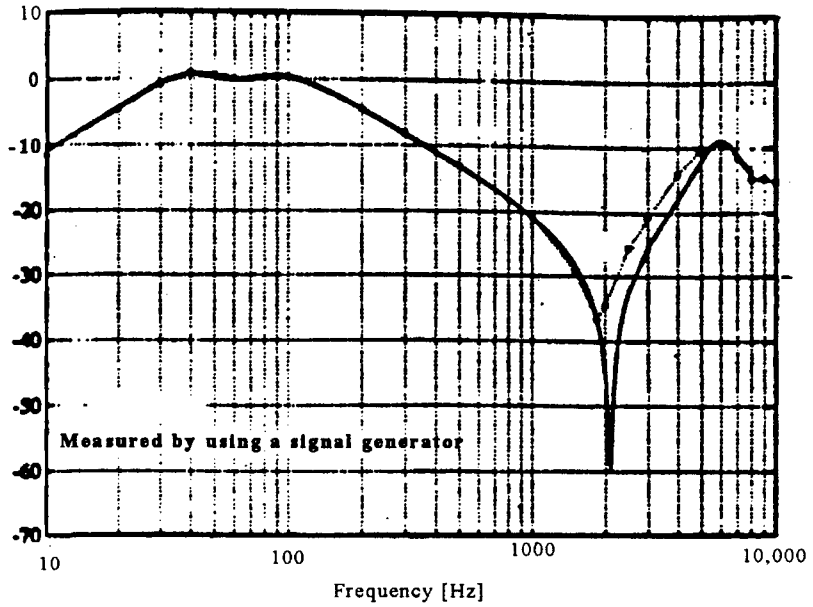


Fig. 90: PT's frequency response, $20\log(V_{out}/V_{in})$ dB versus frequency in hertz.

sponse of the approximating function in the continuous time domain rendered in the frequency domain by a Laplace transform.

With backward Euler's rule, the z-transform of the transfer function is produced by the substitution in Eq. (146) of $s = \frac{1}{\Delta t}\frac{z-1}{z}$. After some manipulation, the z-domain transfer function for B.E. is

$$H(z) = \frac{k_3(z-1)z}{k_1 z^2 + k_2 z + 1} \tag{147}$$

where $k_1 = 1 - p_2\Delta t - p_1\Delta t + p_1 p_2 \Delta^2 t$, $k_2 = p_1\Delta t + p_2\Delta t - 2$, and $k_3 = k \cdot \Delta t$. The magnitude of the resulting discrete time function response can be seen in Fig. 93. In the bandwidth targeted by this simulator the response approximates satisfactorily the one of the real PT in Fig. 90.

The transfer function used in simulating the PT's nonlinear frequency characteristics is equivalent to feeding the input voltage as V_{source} in Fig. 92, and computing the source's current according to Eq. (148). That current is stripped of its units and its magnitude equates the output voltage of the PT.

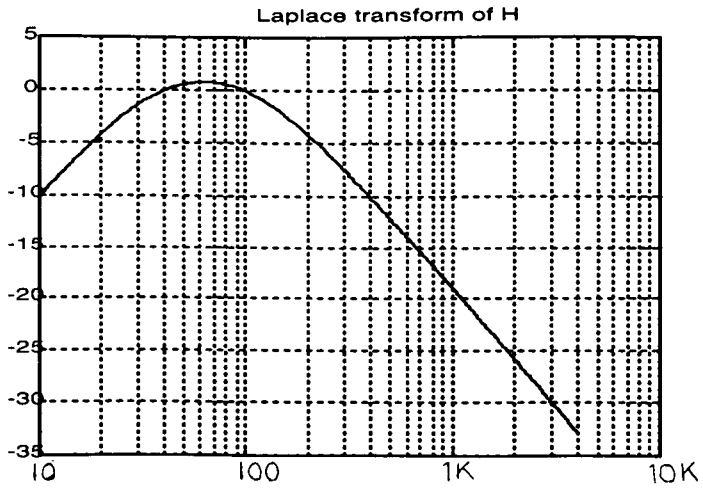


Fig. 91: Approximated PT's frequency response, as rendered by the two polo continuous time Laplace transfer function.

$$i(t) = \frac{k_3}{k_1} \cdot v(t) - [\frac{k_3}{k_1} \cdot v(t - \Delta t) + \frac{k_2}{k_1} \cdot i(t - \Delta t) + \frac{1}{k_1} \cdot i(t - 2\Delta t)] \qquad (148)$$

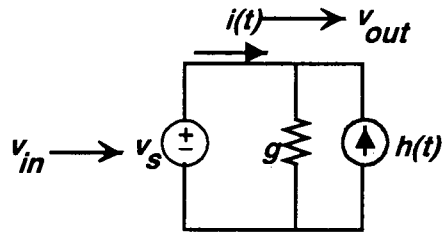


Fig. 92: Equivalent circuit used to approximate the response of the PT.

In the equivalent circuit in Fig. 92, the conductance $g = k_3/k_1$, and the history current source $h(t) = \frac{k_3}{k_1} \cdot v(t - \Delta t) + \frac{k_2}{k_1} \cdot i(t - \Delta t) + \frac{1}{k_1} \cdot i(t - 2\Delta t)$. The model includes, as in equation above, four multiplications and three additions per integration step.

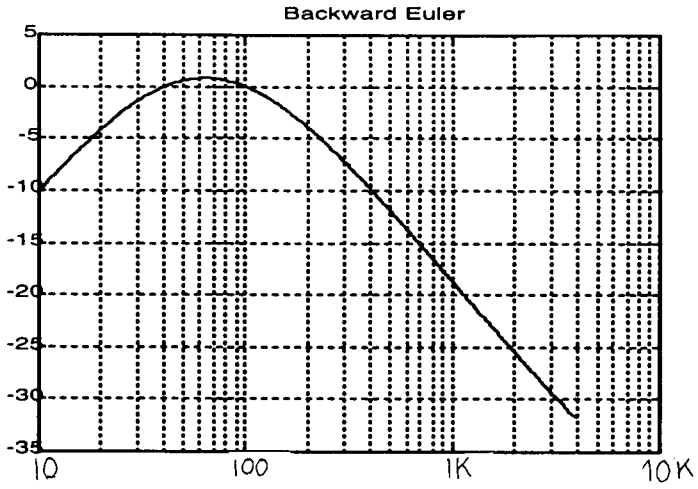


Fig. 93: Approximated PT's frequency response, as rendered by the two polo discrete time Backward Euler transfer function.

9. THE HVDC MODEL

9.1 Introduction

The HVDC model described in this chapter is the result of a team effort [30] in which this author was responsible for developing the solution algorithm to achieve real-time performance.

The idea behind the model is to represent a six-valve module like the one on Fig. 94, with the same technique introduced in [9]; i.e., to consider a valve operation in the same way that switching operations were included in [9]. That is, for every possible switch/valve open/close combination, the corresponding block/module conductance matrix is precalculated and prestored for fast retrieval during the simulation proper. All those matrices are prestored in a vector of matrices. That vector is indexed by an integer variable, *iVlvStatus*, whose internal bit representation corresponds to the open/close state of each one of the switches/valves in the block/module, Fig. 95.

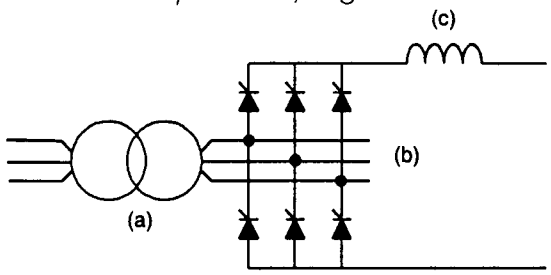


Fig. 94: Six valve module modelled for OVNI and its three parts: a) the three-phase transformer; b) the six-valve bridge; c) the smoothing reactor.

At a first attempt, MATE alone was used to separate several of those six valve modules in a 24 valve case, but still the timings —even if significantly faster than the EMTP's 3120 μs/step on a 200 MHz Pentium Pro workstation—

fell in the vecinity of 770μsec/step[1]. Then, OVNI's *Node Hiding* scheme was applied to each module, as described in § 5.13 on page 74. It was this last technique, implemented as described in this chapter, that brought the timings down to 81 sec/step. On OVNI's current 400 MHz machine, performance falls comfortably within the real-time deadline targeted.

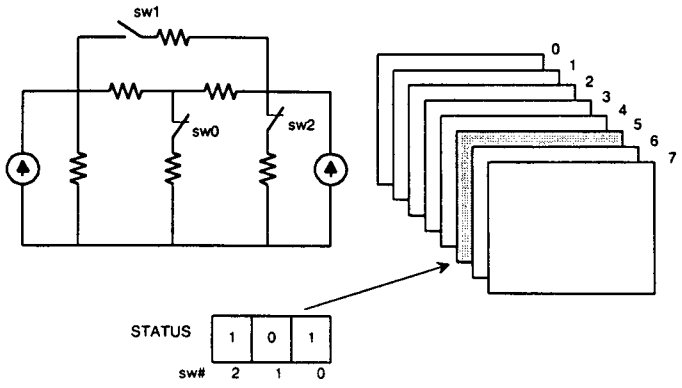


Fig. 95: Matrix precalculation scheme for blocks used in OVNI [9].

9.2 The three-phase linear transformer model

Starting with a linear single-phase unit, the 3-phase transformer model is built. Hence, it is convenient to begin with that single-phase transformer model.

9.2.1 Single-phase transformer model

The single-phase transformer model takes into account: a) the short circuit impedance, or rather, its inverse, Y, and b) the transformers ratio, a. See Figs. 96.

From Fig. 96a, the current in the primary can be written in terms of the voltages as:

$$I_1 = Y\,(V_1 - aV_2) = YV_1 - aYV_2 \tag{149}$$

[1] Results obtained in a previous work programmed in Ada95.

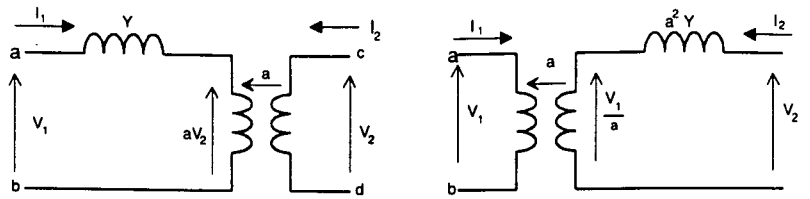


Fig. 96: a) Single-phase transformer, Z_{sc} referred to the primary; b) Z_{sc} referred to the secondary.

From Fig. 96b, the corresponding expression for the secondary current is:

$$I_2 = a^2Y\,(V_2 - V_1/a) = -aYV_1 + a^2YV_2 \tag{150}$$

In matrix form Eqs. (149) and (150) can be expressed:

$$\begin{bmatrix} Y & -aY \\ -aY & a^2Y \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} \tag{151}$$

If no node in the single phase transformer is grounded, and they are connected to nodes a, b, c, and d, as indicated in Figs. 96, voltages V_1 and V_2, as well as the primary and secondary currents can be written in terms of the voltages of each of those four nodes with respect to the reference node (ground), wherever it may be in the adjacent network. Thus we have

$$\begin{array}{lllllll} V_1 & = & V_a - V_b, & I_a & = & I_1, & I_c & = & I_2 \\ V_2 & = & V_c - V_d, & I_b & = & -I_1, & I_d & = & -I_2 \end{array} \tag{152}$$

In this case the 2×2 matrix in Eq. (151) becomes the 4×4 matrix in Eq. (153), which makes no assumptions on the way the single-phase transformer is connected within the network.

$$\begin{bmatrix} Y & -Y & -aY & aY \\ -Y & Y & aY & -aY \\ -aY & aY & a^2Y & -a^2Y \\ aY & -aY & -a^2Y & a^2Y \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \\ V_d \end{bmatrix} = \begin{bmatrix} I_a \\ I_b \\ I_c \\ I_d \end{bmatrix} \tag{153}$$

9.2.1.1 Transformer Data

For each of single-phase units in a three-phase bank, this data is to be collected:

- kV_1, rated kilovolts on primary.
- kV_2, rated kilovolts on secondary.
- MVA, rating of single-phase unit.
- Z_{sc}, short circuit impedance in percentage.

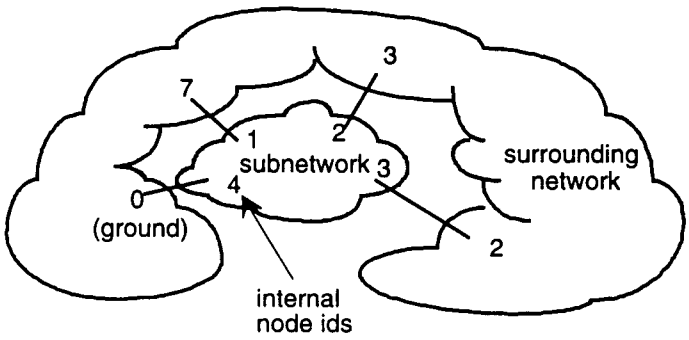


Fig. 97: Internal versus external node identification.

The short circuit impedance is assumed to be purely inductive. The short circuit or series inductance is (where f, is the frequency in hertz):

$$L_{SC} = \frac{kV_1^2}{MVA} \times \frac{Z_{SC}}{100} \times \frac{1}{2\pi f} \tag{154}$$

The transformer's ratio, regardless of which side is high-voltage, is, for the purpose of this model given by

$$a = \frac{kV_1}{kV_2} \tag{155}$$

Three conductances are then calculated from the L_{SC} and a values thus obtained; namely: g_{11}, g_{12}, and g_{22}, defined as follows (using Backward-Euler integration rule, where Δt is the discretization integration step chosen):

$$g_{11} = \frac{\Delta t}{L_{SC}}, \quad g_{12} = a.g_{11}, \quad g_{22} = a^2 g_{11} \tag{156}$$

Then the single-phase unit $[Y]$ matrix can be written simply as:

$$[Y] = \begin{bmatrix} g_{11} & -g_{11} & -g_{12} & g_{12} \\ -g_{11} & g_{11} & g_{12} & -g_{12} \\ -g_{12} & g_{12} & g_{22} & -g_{22} \\ g_{12} & -g_{12} & -g_{22} & g_{22} \end{bmatrix} \tag{157}$$

9.2.2 The three-phase transformer matrix/model

In general, a 4-node subnetwork represented by its 4×4 $[Y_s]$ matrix, and connected to a surrounding network at nodes m, n, p, and q, (as indicated in Fig. 97) contributes to the networks $[Y_n]$ matrix as sketched in Fig. 98, and outlined in the C-code in the listing in Fig. 101.

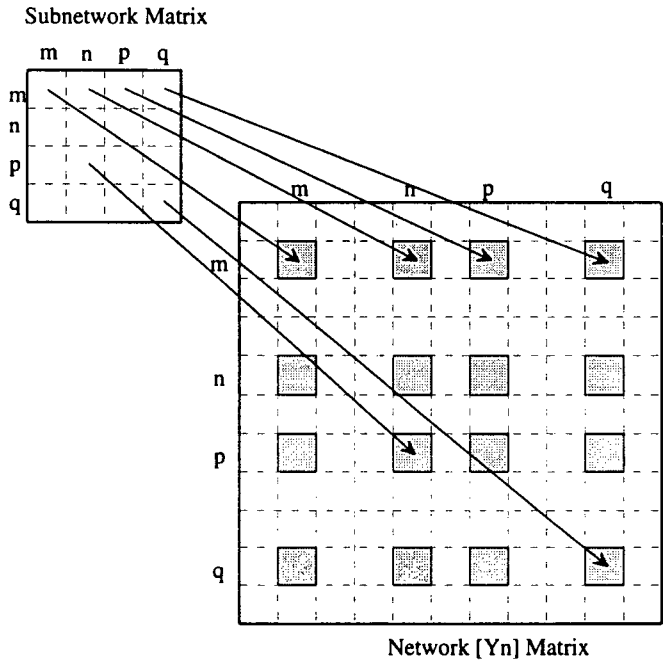


Fig. 98: Including the subnetwork's matrix into the network's matrix.

As an example of the way to include the subnetworks admittance matrix,

$[Y_s]$ into the network's $[Y_n]$, let us detail the inclusion of one of the elements. In the subnetwork illustrated in Fig. 97, the nodes identified by the subnetwork as 1, 2, 3, and 4, are actually (from the point of view of the network) nodes 7, 3, 2, and ground. The element $Y_s(1,3)$ has to be added to the network's $Y_n(7,2)$.

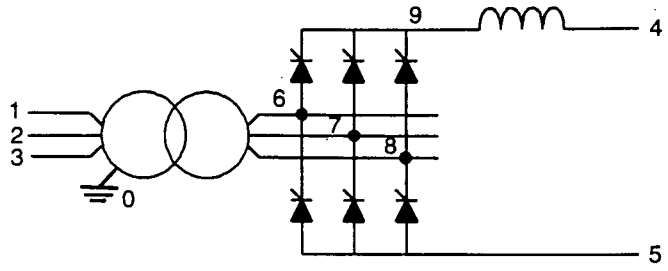


Fig. 99: Node numbering in the hvdc module.

9.2.2.1 A complete three-phase example

As a complete numerical example, let us build the Y-matrix of an hvdc-module, Fig. 99, minus the 6-valve bridge, and minus the smoothing reactor.

Each single-phase units data is: 50 MVA, 100/230 kV, Zsc = 10%, 50 Hz, and the discretization will be done using backward Euler's integration rule and an integration step of $\Delta t = 50\,\mu$s. Using the formulas in Eqs. (154,155 and 156)

$$L_{SC} = \frac{100^2}{50} \times \frac{10}{100} \times \frac{1}{2\pi 50} = 0.06367\,H \tag{158}$$

$$a = \frac{100}{230} = 0.4348 \tag{159}$$

$$\begin{aligned} g_{11} &= 50 \times 10^{-6}/63.67 \times 10^{-3} = 0.7854\,mS \\ g_{12} &= 0.4348 \times 0.7854 = 0.3415\,mS \\ g_{22} &= 0.4348^2 \times 0.7854 = 0.1485\,mS \end{aligned} \tag{160}$$

The single-phase transformer matrix is, according to Eq. (157)

	a or 1	b or 2	c or 3	d or 4
1 or x	1	0	6	8
2 or y	2	0	7	6
3 or z	3	0	8	7

Tab. 9.1: (Matrix "node") Connection nodes for transformers x, y, and z. Rows are the transformers, and columns are the nodes.

$$[Y_t] = \begin{bmatrix} 0.7854 & -0.7854 & -0.3415 & 0.3415 \\ -0.7854 & 0.7854 & 0.3415 & -0.3415 \\ -0.3415 & 0.3415 & 0.1485 & -0.1485 \\ 0.3415 & -0.3415 & -0.1485 & 0.1485 \end{bmatrix} \tag{161}$$

Now, with three of those single phase units, let's call them transformers x, y, and z, in a `Yd11` connection, we can add each of their contributions to the module's matrix $[Y_N]$. In Fig. 100, the details of the connection to the module nodes are shown. Those nodes are tabulated in Table 9.1, the matrix "node". The process is better described by the C-code in the listing in Fig. 101.

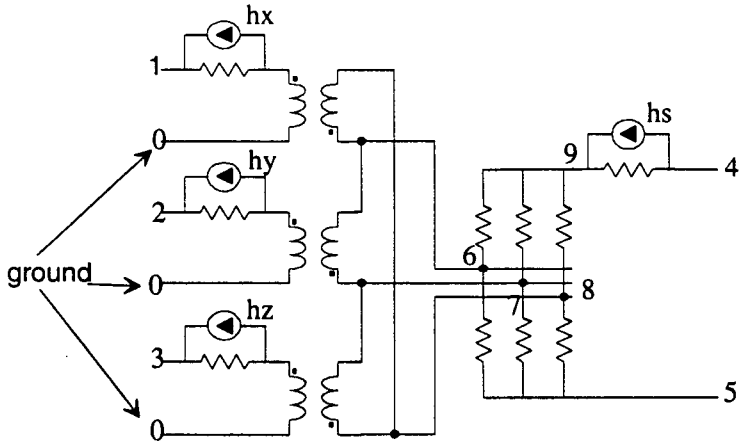


Fig. 100: Yd11 three-phase connection of single phase units.

The resulting matrix for the nine node module is

```
for( tr = 1; tr <= 3; tr++ ){
   for( row = 1; row <= 4; row++ ){
      for( col = 1; col <= 4; col++ ){
         extNode1 = node[tr][row];  // Network node number.
         extNode2 = node[tr][col];  // Network node number.
         Ym[ extNode1 ][ extNode2 ] += Yt[row][col];
      }
   }
}
```

Fig. 101: Procedure to incorporate the single phase units Yt matrices into the module's Yn matrix.

$$[Y_N] = \begin{bmatrix} 0.7854 & 0 & 0 & 0 & 0 & -0.3415 & 0 & 0.3415 & 0 \\ 0 & 0.7854 & 0 & 0 & 0 & 0.3415 & -0.3415 & 0 & 0 \\ 0 & 0 & 0.7854 & 0 & 0 & 0 & 0.3415 & -0.3415 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.3415 & 0.3415 & 0 & 0 & 0 & 0.2969 & -0.1485 & -0.1485 & 0 \\ 0 & -0.3415 & 0.3415 & 0 & 0 & -0.1485 & 0.2969 & -0.1485 & 0 \\ 0.3415 & 0 & -0.3415 & 0 & 0 & -0.1485 & -0.1485 & 0.2969 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (162)$$

9.2.3 Adding the 6-valve bridge and the smoothing reactor

Each one of the six valves in the bridge is modelled as a resistor with one of two possible values depending on whether the valve is open (OFF) or closed (ON). The values chosen for the resistance are 1 mΩ when the valve is conducting (ON) or 1 GΩ when it is not conducting (OFF).

The combination of ON/OFF values states for the six valves is what we call the status of the bridge. The current status of the bridge is kept in an integer variable (status) where the six least significant bits store the state of each of the

six valves. Those bits are set to one for ON valves, and reset to zero for OFF valves. See Fig. 102.

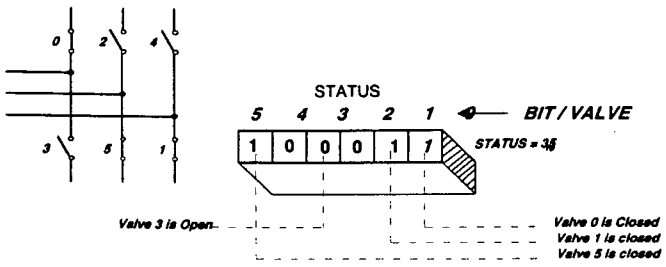


Fig. 102: Status of the bridge as a bitwise variable.

For 6-valves there are $2^6 = 64$ possible combinations of ON/OFF states (even if some are not possible under normal conditions). For each of those 64 combinations one can precalculate (during the preprocessing stage) the corresponding $[Y]$ matrix of the whole module. The 64 matrices thus obtained are stored in a vector of matrices with 64 elements, subscripted from 0 to 63, see Fig. 103. It is worth noting that the variable "status" contains in the first six bits of its binary representation the OPEN/CLOSE status of each of the valves in the bridge. That variable "status" when interpreted as a digital integer indexes the proper $[Y]$ matrix to be used to represent the module at any time-step.

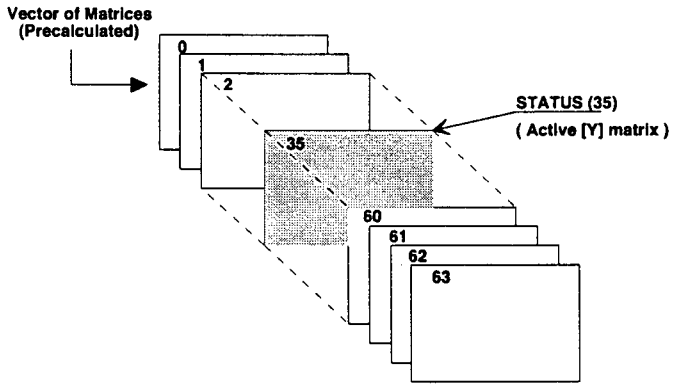


Fig. 103: The vector of precalculated $[Y]$ matrices.

The smoothing reactor contributes to each of the sixty four matrices with its discrete equivalent conductance, according to the selected integration rule. For the backward Euler's rule, the equivalent conductance of that reactor is

$$g_{smooth} = \frac{\Delta t}{L_{smooth}} \tag{163}$$

This conductance value will be added to positions —according to Fig. (99)— (9, 9) and (6, 6), and subtracted from positions (6, 9) and (9, 6) in each of the $[Y]$ matrices calculated above.

9.3 History sources in the hvdc-module

Now that the resistive contribution of the hvdc-module to the network's $[Y]$ matrix, in any of its sixty four possible `on/off` valve combinations has been taken care of, let us focus our attention on the current history sources. In the hvdc module, there is one history source from the discrete model of the smoothing reactor, and one history source for each single-phase transformer unit, corresponding to the discretization of its short circuit inductance, see Fig. 104.

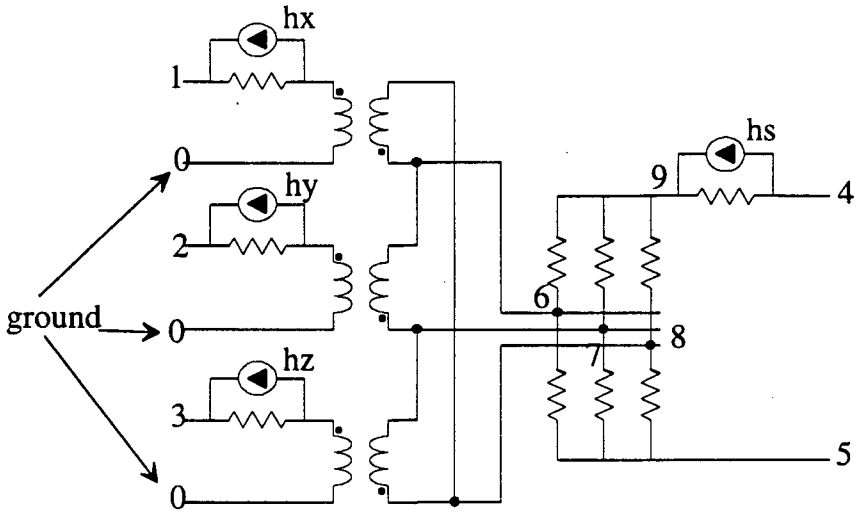


Fig. 104: Discrete time model of the hvdc 6-valve module.

Before considering the accounting of each of those history sources into the

total nodal currents, let us examine the history source in the single-phase transformer unit as depicted in Fig. 105.

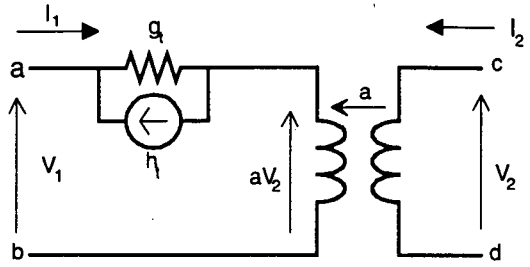


Fig. 105: A single phase discretized short circuit inductance.

The short circuit inductance equivalent discretized history source contributes to the total nodal currents of node "a" with a value of plus-h_t, and to node "c" with minus-h_t; it also contributes to nodal currents of node "c" with minus-$a \cdot h_t$, and to node "d" with plus-$a \cdot h_t$. The voltage across the short circuit inductance that is used to update the history source h_t is v_L , as expressed in Eq. (164)

$$v_L = v_a - v_b - a\,(v_c - v_d) \tag{164}$$

9.3.1 Examples

To illustrate the whole process of history current accumulation into the nodes of the hvdc-module, let us consider the two transformer connections included in the code: `Yy0` and `Yd11`. In Fig. 107 the `Yd11` connection shows clearly where the different single-phase units are connected within the module. From there and according to what was said in the previous section, the nodal current vector for the `Yy0` connection is shown in Fig. 106.

9.4 Hvdc matrices

The 9 × 9 (or 10 × 10) G-matrix (the Y-matrix is real, thus it is a G — conductance— matrix) of the hvdc module relates the total nodal currents, $[h]$, in the module with the voltages of its nodes, $[v]$, according to Eq. (165).

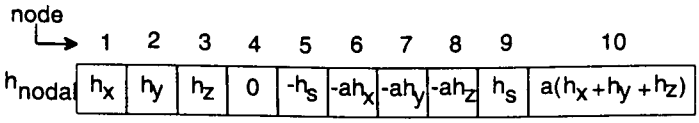


Fig. 106: Total nodal currents for 'Yd11' connection.

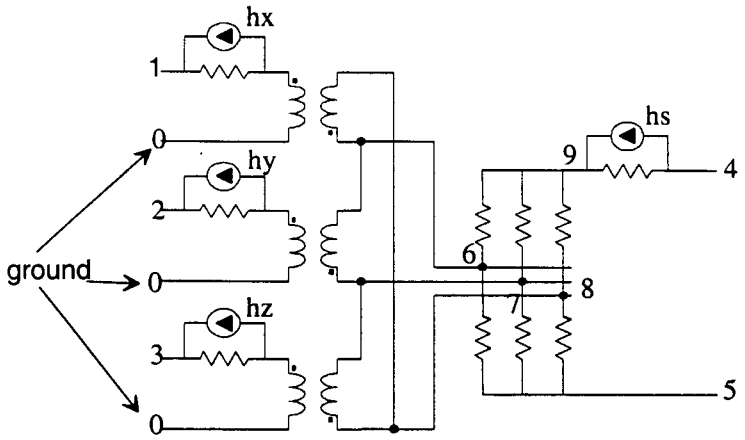


Fig. 107: Hvdc module with a 'Yd11' transformer connection.

$$[G]\,[v] = [h] \tag{165}$$

From all the nodes in the model, only the first five will remain visible to the solver, let us call them "a" nodes, and the rest "b" nodes. Making use of this definitions, and subscripts, Eq. (165) can be written using a matrix partition as in Eq. (166).

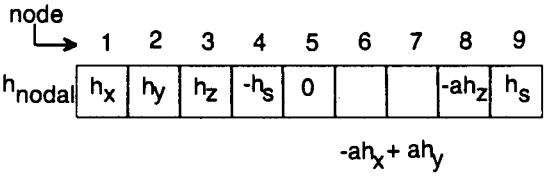


Fig. 108: Total nodal currents for 'Yd11' connection.

$$\begin{bmatrix} G_{aa} & G_{ab} \\ G_{ba} & G_{bb} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \end{bmatrix} = \begin{bmatrix} h_a \\ h_b \end{bmatrix} \tag{166}$$

Next, and using the Generalized-Kron's reduction, the system of equations in Eq. (166), becomes the reduced one in Eq. (167).

$$[G_{red}]\,[v_a] = [h_{red}] \tag{167}$$

where

$$\begin{aligned} G_{red} &= G_{aa} - G_{ab}G_{bb}^{-1}G_{ba} \\ h_{red} &= h_a - G_{ab}G_{bb}^{-1}h_b \end{aligned} \tag{168}$$

9.5 Interface of the hvdc model and OVNI

At each time step, the driver determines the voltages for the "a" nodes, as defined above, based on the history values h_A calculated by the module in the previous step; i.e., OVNI calculates v_A.

Next, it is the module's model turn again. It receives v_A, the voltages of the "connection-nodes", and counting on the availability of the h_B history values calculated by the model itself during the previous time step, the model proceeds to establish the voltages for the "b" nodes:

$$v_b = G_{bb}^{-1}\,(h_b - G_{ba}v_a) \tag{169}$$

Now, with all the modules nodes voltages, v_A and v_B (just calculated), the model computes h_A and h_B for the next time step. Before returning the h_A vector to OVNI, it includes the effect of the reduced nodes like this:

$$h_a^{new} \leftarrow h_a^{prev} - G_{ab}G_{bb}^{-1}h_b \tag{170}$$

where h_a^{prev} is the value of the currents vector h_a, before accounting for the internal nodes contributions in vector h_b; h_a^{new} is the vector once the contributions have been included, and ready to be exported to OVNI as external source's history terms; the matrix product $G_{ab}G_{bb}^{-1}$ is precalculated and identified as G_{mix} in the code.

Finally, the module's model returns to OVNI, the core.

10. HVDC-BRIDGE CONTROLLER

To explore some of the limitations and capabilities of the HVDC-bridge model developed previously, a basic current control loop model was introduced. This model incorporates a simple proportional-integrative amplifier, receives as input the DC-output current of the rectifier group and, as synchronization signals, the input voltages to the bridge groups, and issues the gate signals corresponding to each of the twelve valves in a pair `Yy/Yd` transformer-bridge group, see Fig. 109.

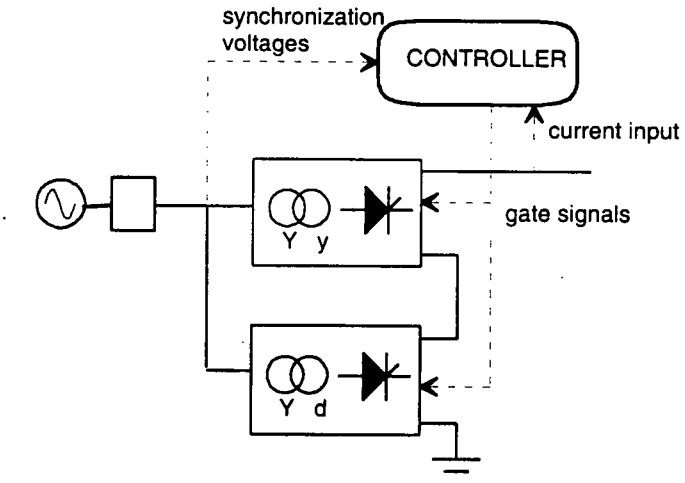


Fig. 109: Inputs and output of the simplified current controller.

10.1 Block View of the Controller

The controller strives to maintain the DC current at a desired value, the reference value. The controller adjusts the firing angle of the valves as it considers

necessary to achieve that goal.

In Fig. 110, a block schematic is shown that illustrates the general structure of the controller modelled. The D.C. raw current, read from one of the HVDC bridges after smoothing, is put through a filter to keep only the D.C. component. The filtered D.C. current is compared against a reference value, and the difference is labelled the error for the purposes of the PI amplifier, the next stage. The PI block produces the raw or apparent needed change in the firing angle, $\Delta\alpha^*$. This proposed change in alpha is then clipped, if necessary, to keep the firing angle within the limits imposed by the user. Next, in the cycle-position stage, the controller determines whether, including the proposed change, it is time to trigger the next valve. If triggering time conditions are met, the valve-scheduler takes over, produces the necessary gate signal, and activates the ramp-cycle synchronizer that, using as input voltages on the primary of the transformer-group, reset the ramp-cycle counters.

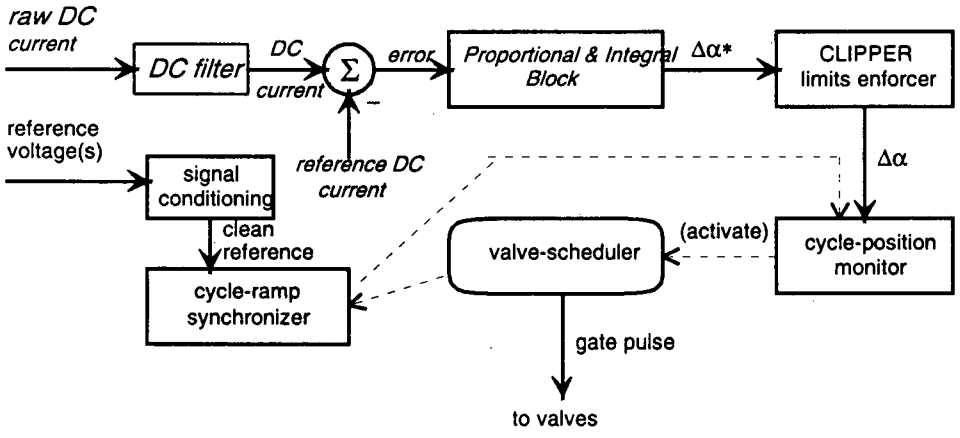


Fig. 110: Controller model block diagram.

10.2 Stage One: The DC filter

The signal that needs to be kept at the chosen reference value, is the output current of the rectifying HVDC-bridge. Regardless of the smoothing effect of the

inductive stage in the bridge, the output current still contains some harmonics that need to be filtered out before the current is put to the controller. To extract the DC component from the current, a simple RC filter was used.

10.3 Proportional-Integrative Block

In this stage[1], and using as input the error (ϵ), that is, the difference between the DC component of the HVDC bridge output current and the reference value, the necessary change in the firing angle is computed as sketched in Eq. (171).

$$\begin{aligned} \epsilon &= I_{DC} - I_{reference} \\ \Delta\alpha^* &= K_p\,\epsilon + K_I \int \epsilon \cdot dt \end{aligned} \tag{171}$$

To discretize the second part of Eq. (171), one can observe that it describes the current voltage relationship of the series RC circuit fed by a current source, as illustrated in Fig. 111a. In that circuit, the current has a value of epsilon, ϵ; the voltage is delta alpha asterix, $\Delta\alpha^*$; the resistance has a value K_p; and the capacitance a value $1/K_I$.

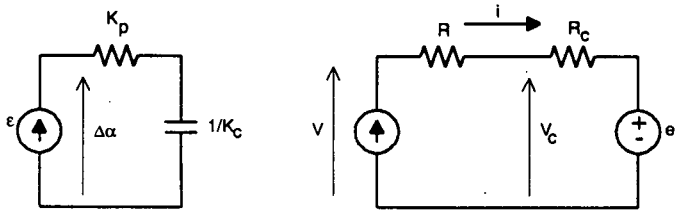


Fig. 111: RC equivalent circuit for the PI block.

Once discretized, the equivalent circuit of the PI block appears in Fig. 111b. If the backward Euler's integration rule is used to discretize the circuit, R_c and $e(t)$ are given by Eq. (172).

[1] Even if in analog control systems the PI-block is implemented by an amplifer with the appropriate feedback, and as such has been referred to as the "PI-amplifier" in that context, in the case of a digital controller, the use of the name block seems more appropriate.

$$\begin{aligned} R &= K_p \\ Rc &= \tfrac{\Delta t}{C} = K_I.\Delta t \\ e(t) &= V_c(t-\Delta t) \end{aligned} \tag{172}$$

From the circuit in Fig. 111b, the voltage across the capacitor can be determined as

$$V_c = V - R \cdot i \tag{173}$$

Combining Eq. (173) with Eq. (172c), the history voltage source $e(t)$ is expressed

$$e(t) = V(t-\Delta t) - R \cdot i(t-\Delta t) \tag{174}$$

We know that the total voltage, V, is but delta alpha asterix, the correction in the firing angle. We also know that the current, i, is the error ϵ. Simplifying our notation for values in the previous integration step by applying an apostrophe to them[2], Eq. (174) becomes

$$e = \Delta'\alpha - K_p\,\epsilon' \tag{175}$$

where $\Delta'\alpha$ is the correction of the firing angle at the previous time step; and ϵ' is the error at the previous time step.

The total voltage across the RC group, V, is then calculated

$$v = e + (R + R_c)i = e + R_T \cdot i \tag{176}$$

Substituting previous equations into Eq. (176), the last one becomes Eq. (177).

$$\Delta\alpha = \Delta'\alpha - K_p \cdot \epsilon' + (K_p + K_I \cdot \Delta t)\,\epsilon \tag{177}$$

where $\Delta'\alpha - K_p \cdot \epsilon'$ is called *hist* in the code since it depends on previous step's values of error and angle change; also $(K_p + K_I \cdot \Delta t)$ is the value of the resistor R_T in the code.

[2] That is, for any function of time, $x = f(t)$, $x = x(t)$, and $x' = x(t - \Delta t)$.

At every time step, the necessary change in the firing angle is then computed,

$$\Delta\alpha = hist + R_T \cdot \epsilon \tag{178}$$

Now all that remains is an efficient formula to update that history value, *hist*. At the previous time step the correction is also given by Eq. (179), i.e.

$$\Delta'\alpha = hist' + R_T \cdot \epsilon' = hist' + K_p.\epsilon' + K_I.\epsilon' \tag{179}$$

Substituting this into the definition of *hist* implicit in Eq. (177), the updating formula for the history value *hist* is

$$hist = hist' + K_I\ \Delta t\, \epsilon' = hist' + R_c\, \epsilon' \tag{180}$$

At every time step Eqs. (178, 180) are used. The first one to determine the necessary change in firing angle, the other one to calculate the *hist* value that will be used at the next time step.

10.4 Cycle position monitor and the Valve Scheduler

The controller issues gate signals both for the six valves of a `Yy0`-transformer-bridge module as for the six valves of a `Yd11`-transformer-bridge module, as intimated by Fig. 109. However, to simplify the explanation of the *cycle position monitor* (and of the *valve scheduler*), it is better to review the process when applied only to one of the modules, let us say the `Yy0` one. After the method is explained, the combined effect of both types of bridges is accounted for.

In what follows, and to simplify the description of the processes, valves in the bridge are numbered from zero to five, and connected to phases A, B, and C of the transformer secondary according to Fig. 112. Further down this section, when need arises to refer to valves in both types of modules, `Yy0` and `Yd11`, the valves will be labelled: Y_0, Y_1, Y_2, Y_3, Y_4, and Y_5 for the `Yy0` module; and D_0, D_1, D_2, D_3, D_4, and D_5 for the `Yd11` module.

Figure 113 shows the voltages in phases A, B, and C, connected to the bridge as depicted in Fig. 112. In Fig. 113a, also, if firing angle is set to zero degrees (this controller is symmetric in the sense that the same firing angle is

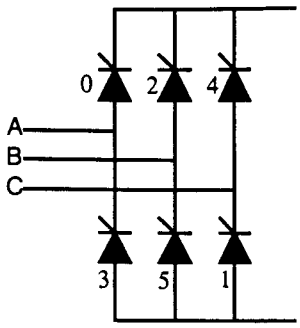


Fig. 112: A full-wave valve bridge, with valves and phases identified.

applied to all valves), valves need to be fired at the time points and sequence there indicated. Time points which are separated by the constant (under no controller modulation) interval of sixty degrees (translated into time units, to be sure).

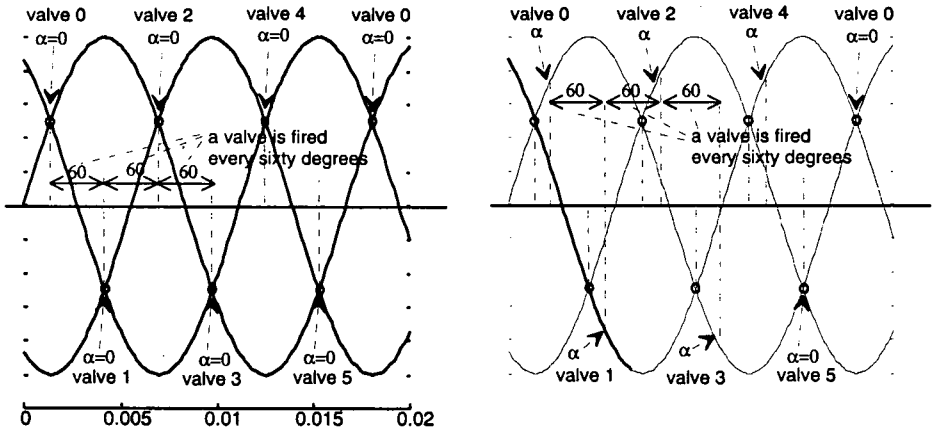


Fig. 113: a) Firing time points when alpha is zero; b) Firing points when alpha is not zero.

This effect can be produced by two separate but combined data processes: a counter, *tick*, that goes from zero to sixty degrees, the reference, held in variable *tickRef* (with some modification, to be seen), and is compared at each time step

against its limit, sixty. When the counter, *tick*, hits the limit, a valve needs to be fired; which valve to fire, is the question answered by the second data process, an infinite periodic sequence, 0, 1, ..., 5, 0, 1, ..., 5, etc., simulated by an array of six elements, *aValveSequence*, and an index that wraps around, *iNextValveToFire*. These two data processes can be visualized, the first by a saw-tooth ramp, as in Fig. 114; and the second one, by a circular array, see Fig. 115.

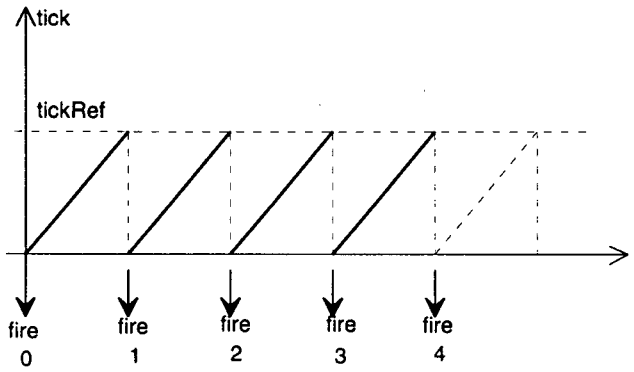


Fig. 114: The ramp signal and the model's variables for a = 0.

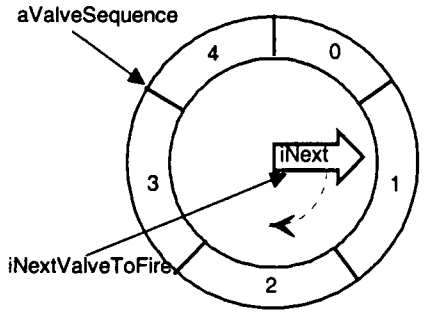


Fig. 115: Data structure to select next valve to be fired, when the ramp so requests.

If a Yd-module is controlled by the same control unit than the previous Yy-module, gate signals have to be issued each 30°, one for the Yd-module, and next one for the Yy-module, according to the sequence: D_0, Y_0, D_1, Y_1, D_2, Y_2, D_3,

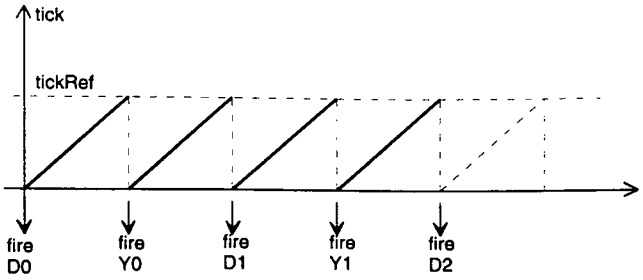


Fig. 116: The ramp signal and the model's variables for $\alpha = 0$, when gate signals are issued for Yy and Yd modules.

Y_3, D_4, Y_4, D_5, Y_5, and repeat. That is twelve possibilities. In this case *tickRef* is 30°, and *aValveSequence* is complemented by a parallel array: *aValveGroup*, that indicates if the next valve to be fired is in a Yd-module, zero-code, or in a Yy-module, one-code. In this case, the index variable, *iNextValveToFire*, wraps around at 11 down to zero. See Figs. 116 and 117.

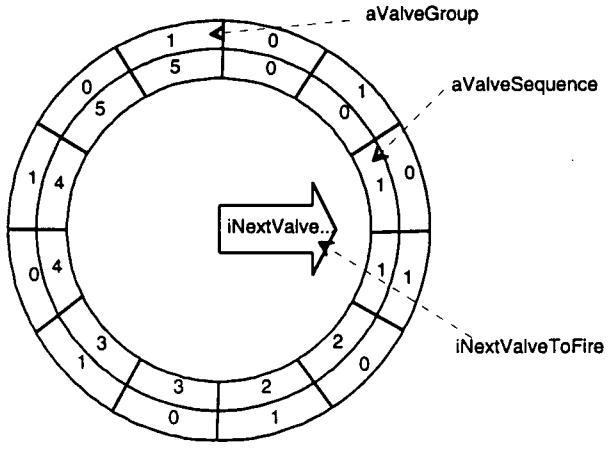


Fig. 117: Scheduling the next valve to be fired: *index*, *iNextValveToFire*; and arrays: *aValveGroup* and *aValveSequence*.

10.5 Cycle Ramp Synchronizer

At the beginning of each cycle of the input voltage, the position of the tick counter within the cycle of the reference signal needs to be determined. To do so, the reference point $\alpha = 0$ in Fig. 113a, is obtained as the moment when the two voltage reference signals, voltages of phases a and c, are equal and positive. In the first implementation of the controller, a semi-infinite bus with constant frequency is assumed at the primary of the transformers feeding the bridges, under that assumption synchronization becomes a simple task of keeping track of the number of integration steps that have passed by, and comparing the count with the number of steps per cycle, i.e. no need for the additional input sketched in Fig. 109. However, under more general conditions, that signal may come from a less ideal source and present some higher harmonic content that forces upon us the introduction of some kind of filtering to extract the fundamental of the voltages in phases a and b before comparing them. For details of this reference fundamental extraction see § 10.7. at the end of this chapter. There, a simplified and sufficiently accurate filtering scheme with high computational efficiency is described. This was the filter adopted for this controller.

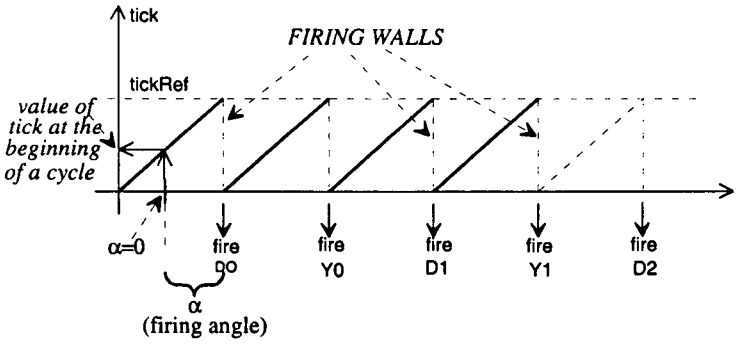


Fig. 118: Firing walls and initial value of the tick ramp counter at the beginning of each reference cycle.

In the previous section, to determine when to fire a valve, we used a step counter, *tick*, incremented at each time step, and checked if it had hit one of

the vertical edges of the sawtooth wave we used to explain the operation of the firing process in Figs. 112 and 116, repeated here as Fig. 118 for convenience. Let us call those vertical edges firing-walls. So the process of firing is reduced to counting steps, and waiting for the counter, *tick*, to reach the next firing-wall. At the beginning of a cycle, that is when v_a is equal to v_b and positive, we determine how far from the next firing-wall the tick counter is. This process also sets the index *iNextValveToFire* at the right position within the arrays in Fig. 117, above.

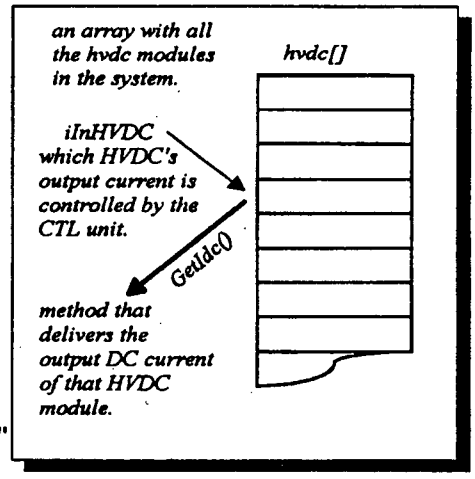
All this process was implemented in the method *CalcTickIniCycle* of the class `ctl_t`, the class that describes any controller entity. The name of the method stands for "Calculate the value of Tick at the Initial moment of the Cycle". As input, it takes the firing angle α, and returns two values: the correct value for *tick*, and the position for index *iNextValveToFire*. Depending on whether the firing angle is in the intervals between one and 30°, or between 30° and 60°, or any of the other 30° wide intervals shown in Fig. 118, the two output values are calculated as in the following code listing:

```
int clt_t::CalcTickIniCycle( double alpha ){
// It returns the position of "tick" at beginning of cycle,
// and sets up index iNextValveToFire.
   if( 1 <= alpha && alpha < 30 ){
      iNextValveToFire = 0;
      return int( 30.5 - alpha );
   }else if( alpha < 60 ){
      iNextValveToFire = 0;
      return int( 60.5 - alpha );
   }else if( alpha < 90 ){
      iNextValveToFire = 0;
      return int( 90.5 - alpha );
   }else if( alpha < 120 ){
      iNextValveToFire = 0;
      return int( 120.5 - alpha );
   }else if( alpha < 150 ){
      iNextValveToFire = 0;
      return int( 150.5 - alpha );
   }else if( alpha < 180 ){
      iNextValveToFire = 0;
      return int( 180.5 - alpha );
   }else{
      // Error condition!
   }
}
```

10.6 Modulating the firing angle

At each time step, and using the formulas in Eqs. (178) and (180), the controller determines the necessary change in the firing angle, $\Delta\alpha$. This change in alpha is the "time distance" that the firing walls need to be displaced to the right, or, what is equivalent, by how much we need to move the tick counter to the left (easier, since it is a single operation). Then, the tick counter is compared versus the next firing wall, if there is a hit a request to fire is issued as seen in previous sections. This functionality is implemented in the method *SenseAndSetUpGateSignals*(), the core of the controller, in the listing that follows.

```
// This code does not use DC prefiltering.
// First, the proportional/integrative section:

error = hvdc[ iInHVDC ].GetIdc() - iRef;  // In amperes.
delAlpha += Rt * error + hist;            // Accumulate change.
hist += Rc * error;                       // Update history term.

// Clipping, to respect min and
// max values of alpha.

newAlpha = alpha + delAlpha;
if( newAlpha > alphaMax ){
   delAlpha = alphaMax - alpha;
}else if( newAlpha < alphaMin ){
   delAlpha = alphaMin - alpha;
}

// Yes, alpha goes in degrees,
// but time here is discrete,
// so convert delAlpha into
// "steps".

delAlphaSteps = ...

// Now, let's check if the "ramp"
// hit the firing wall!

tick++;   // Up goes the ramp!
if( tick - delAlphaStep >= tickRef ){  // Bang! Time to FIRE!
   alpha += delAlpha;  // Change in firing angle accepted!
   delAlpha = 0;       // We can start accumulating change again.
   tick = 0;           // Reset the "ramp".

   // Now, let's schedule...what valve to fire!?
   activeGroup = aValveGroup[ iNextValveToFire ];
   activeValve = aValveSequence[ iNextValveToFire ];

   // Point to next valve, for the next time step.
   if( ++iNextValveToFire > 11 ) // Wrap around, 0..11 valves!
      iNextValveToFire = 0;                    (CONTINUES...)
```



```
(...COMES FROM PREVIOUS PAGE)

   // Now issue the gate signals.

   // Sets the bit in the "gateSgnl" of the "activeGroup"
   // (0=delta, 1=wye) corresponding to the "activeValve".

   gateSgnl[ activeGroup ] |= 1 << activeValve;

   // To control the pulse width, we reset the counter,
   // which, incremented every time step, is used to decide
   // when to turn the gate signal off for that "activeValve"
   // of that "activeGroup".

   pulseWidthCounter[ activeGroup ][ activeValve ] = -1;

} // Ends IF firing wall was hit!
```

gateSgnl[][] — bits(valves)

	5	4	3	2	1	0
delta (0)	0	0	1	1	0	1
wye (1)	1	0	0	1	1	0

gate signals are efficently passed to the HVDC bridges, as bits encased into a convenient integer variable, gateSgnl.

Next, the controller gets synchronized with the voltage signals at the primary of the transformer, as was seen in a previous section, and, finally, the controller checks for gate pulses due for termination in each of the twelve valves, as in the listing on the next page.

Now some final implementation notes. To simplify the counting of degrees at each step, during initialization the controller calculates the coefficient

$$stepsPerDegree = 1.0/(360 * FREQ * deltaT)$$

Also during initialization, the controller converts the counter limit, 30^o, into

```
for( group = 0; group < 2; group++ ){ // For D or Y connextions,
   for( valve = 0; valve < 6; valve ){ // For each valve,
      // If the pulse reached its limit in width...
      if( ++pulseWidthCounter[ group ][ valve ] >= pulseWidth ){
         pulseWidthCounter[ group ][ valve ] = -30000;  // Large negative!
         gateSngl[ group ] &= ~( 1 << valve ); // Turns off gate signal!
      }
   }
}
```

integration step count and puts it in variable *tickRef.*

$$tickRef = integer\_part\_of(30 * stepsPerDegree + 0.5)$$

10.7 Filtering the angle reference voltage

In this work a simplified and computationally highly efficient filter was used to extract the fundamental component out of the angle reference voltage ($V_{AC} = V_A - V_C$), a parallel RLC filter, as illustrated in Fig. 119, tuned to the AC network rated frequency. The bandwidth should be narrow enough as to filter out the high frequency components introduced in this voltage by the switching of the valves; but chosen appropriately, it can include both 60 and 50 Hz with the same parameter values. It was chosen to tune it to $f_o = 50$ Hz, with a bandwidth $B = 34.3$ Hz, with half power frequencies at $f_1 = 35.7$ Hz, and $f_2 = 70$ Hz respectively.

In this type of filter, the resonance frequency in rad/sec is given by

$$\omega_o = \frac{1}{\sqrt{LC}}$$

Also, the bandwidth B can be calculated:

$$B = \omega_2 - \omega_1 = \frac{1}{R}$$

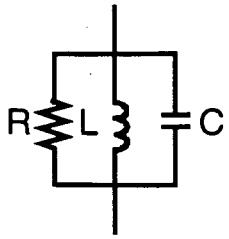


Fig. 119: Filtering the angle reference voltage signal.

The resonance frequency is the geometric mean of the half power frequencies:

$$\omega_o = \sqrt{\omega_1 \omega_2}$$

From all said above, and the last three equations, the filter parameters chosen were: $R = 45.81\,\Omega$; $L = 0.1\,\mathrm{H}$; $C = 101.32\,\mu F$.

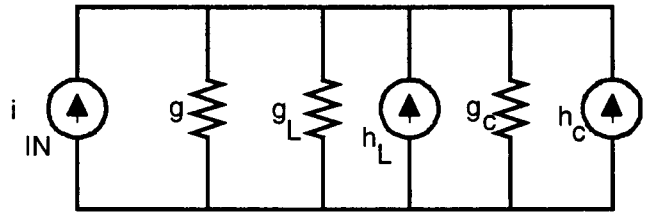


Fig. 120: Discretized version of the reference angle voltage filter.

In Fig. 120, the discretized version of the filter can be seen. The input signal (voltage between phases a and c of the primary of the transformer) is fed into this filter as a current i_{IN} and the filtering proper is achieved in only two sums and one multiplication:

$$v_{filtered} = R_{eq}(h_c + h_L + i_{IN}),$$

where i_{IN} is numerically identical to v_{AC}.

Using backward Euler's as the integration rule, during initialization of the controller the constant discrete equivalent conductances for the capacitor, g_C, and for the inductor, g_L, are calculated as

$$g = \frac{1}{R}, \quad g_C = \frac{C}{\Delta t}, \quad g_L = \frac{\Delta t}{L}$$

Then, also during preprocessing (i.e., at initialization), the equivalent resistance is determined according to

$$R_{eq} = \frac{1}{g + g_L + g_C}$$

At every time step, the two history sources seen in Fig. 12 and used in the filtering equation are updated according to

$$h_L = h'_L - g_L \cdot v'$$

$$h_C = g_c \cdot v',$$

where v' is the previous time step value of the filtered voltage.

The effect of this filter can be appreciated in Fig. 121, where voltage V_{ac} is compared to the output of the filter, its fundamental.

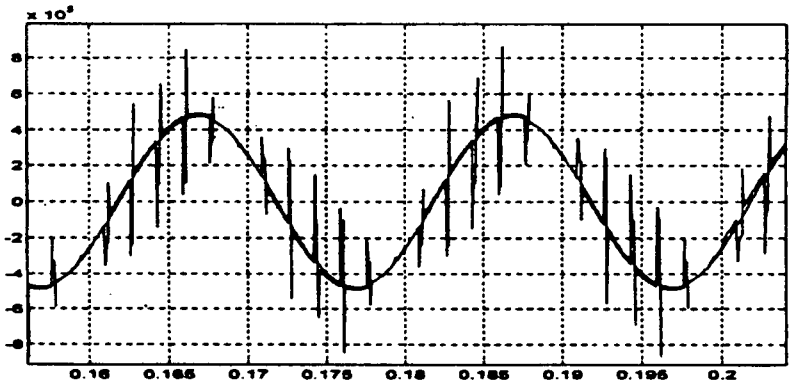


Fig. 121: Reference angle voltage V_{ac} and its fundamentals obtained by the filter described in this section.

11. MODELLING SATURATION IN POWER TRANSFORMERS

The transformer model described in the HVDC multi-state model section of this report is a linear one; i.e., saturation in the core is not considered. To incorporate the effect of magnetic saturation, two different situations were considered, namely: three-phase banks of single-phase units [4]; and three-phase units with coils mounted on a three-leg core [66].

11.1 Saturation in single phase units

When the three-phase transformer is a bank of single phase units, independence of magnetic paths in each of the three phases simplifies modelling of saturation in the core. Magnetization of the core is accounted for, in this case, by a non-linear inductor connected across the low-voltage side of the single-phase unit, as in Fig. 122.

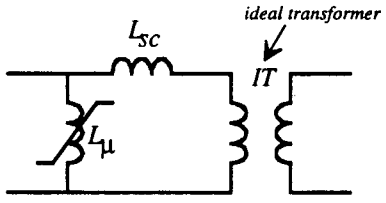


Fig. 122: Magnetization branch in a single-phase transformer (non-linear)

Given the magnetization characteristic of power transformers for these applications, see Fig. 123a, magnetization currents when the core is not saturated can be safely neglected. Hand in hand with the previous modelling compromise goes the accuracy and convenience of representing the magnetization characteristic of the core by a two slope curve, as in Fig. 123b.

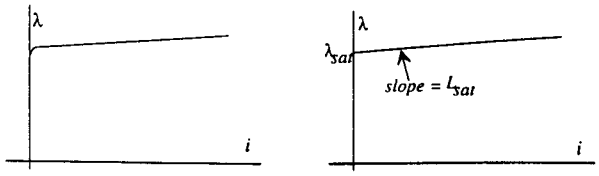


Fig. 123: a) Magnetization of a transformer core (typical); b) Two-segments piecewise magnetization curve used.

Summarizing: a magnetization branch is not included in the equivalent circuit unless saturation is detected during simulation; once saturation is sensed, an inductor is introduced as shown in Fig. 124, with the value of the slope in the saturated part of the characteristic in Fig. 123. Data required from the user includes, the flux-linkages value for the saturation knee in Fig. 123, λ_{sat}, and the slope of the saturated part of that curve, L_{sat}, that is:

$$L_{sat} = \frac{d\lambda}{di}, \quad for \lambda > \lambda_{sat} \tag{181}$$

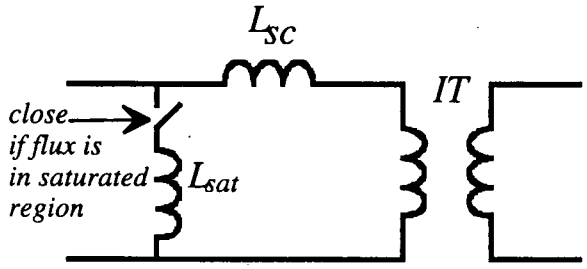


Fig. 124: Saturation modelling for a single phase transformer.

11.2 Saturation in three-phase units

When coils for the three phases are mounted on a three-phase magnetic core, interaction of magnetic flux among the three legs and the surrounding media creates a more complicated scenario. In this case, the non saturated three phase unit is modelled by its open circuit test values: Y_{oc}^{0} and Y_{oc}^{+}, the open circuit admittance for zero sequence, and the open circuit admittance for positive (and

negative) sequence. An approximation is made since $Y_{oc}^{0} >> Y_{oc}^{+1}$, when the phase self and mutual admittances are calculated according to Eqs. (182), we can safely neglect Y_{oc}^{+} and write the magnetization non-saturated matrix as in Eq. (183). Saturation in three-phase units is modelled by

$$\begin{aligned} Y_s &= \frac{Y^o+2Y^+}{3} \approx \frac{Y^o}{3} \\ Y_m &= \frac{Y^o-Y^+}{3} \approx \frac{Y^o}{3} \end{aligned} \tag{182}$$

$$[Y_{ns}] = \begin{bmatrix} \frac{Y^o}{3} & \frac{Y^o}{3} & \frac{Y^o}{3} \\ \frac{Y^o}{3} & \frac{Y^o}{3} & \frac{Y^o}{3} \\ \frac{Y^o}{3} & \frac{Y^o}{3} & \frac{Y^o}{3} \end{bmatrix} \tag{183}$$

A non-saturated three phase transformer unit is represented by the models of each of the phases with a coupled group of magnetization admittances that shunts the three phases to ground, as in Fig. 125.

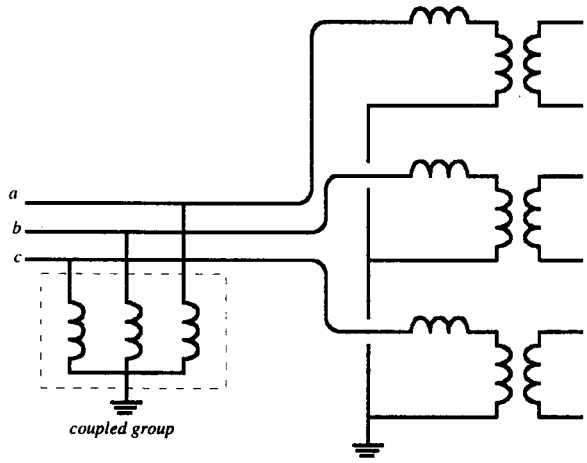


Fig. 125: Non-saturated three phase core transformer.

Flux in each phase in monitored and, if in a particular phase, flux enters the saturated region of the magnetization characteristic, a saturated magnetization equivalent inductance, with the same value as described in Eq. (181), is

[1] For three-leg cores.

introduced in shunt with that phase, as illustrated in Fig. 126, for the case of phase-a's leg saturation.

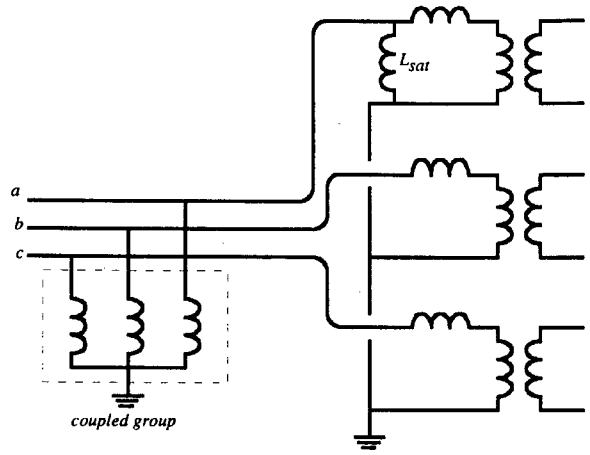


Fig. 126: Three phase core transformer with phase-a's leg saturated.

So, all counted, there are eight possible $[Y]$ matrix contributions for three phase unit, be it a three phase core device, or a bank of single phase units. The eight possibilities account for all possible saturation states in the three legs, magnetically independent or not.

11.3 Keeping track of a phase-leg's flux

To determine whether a core leg is saturated, it is necessary to keep track of each leg's flux linkages at every time step. To do this, beginning with the phase voltage, $v(t)$, at the end of the last processed integration step, starting with Faraday's Law (see Eq. 184); then integrating both sides along the time interval between $(t - \Delta t)$ and t, and using the backward Euler's integration rule to approximate the definite integral on the left side, we obtain the discrete equation on the right hand part of Eq. (184).

$$v = \frac{d\lambda}{dt} \quad \Rightarrow \quad v(t) \cdot \Delta t \approx \lambda(t) - \lambda(t - \Delta t) \tag{184}$$

The last expression in Eq. (184), once reorganized as in Eq. (185) can be

used to keep track of the phase's flux linkages at each time step, provided one knows the flux linkages at the previous time step, and the current voltage across the phase coil (on the side on which L_{sat}, and λ_{sat} were specified).

$$\lambda(t) = \lambda(t - \Delta t) + v(t) \cdot \Delta t \tag{185}$$

If the leg-flux linkages just calculated happen to *jump over* the saturation knee defined by λ_{sat}, the saturated inductor L_{sat} needs to be introduced in the model shunting the corresponding phase (as in Fig. 126, for the case where saturation of the leg corresponding to phase a was detected).

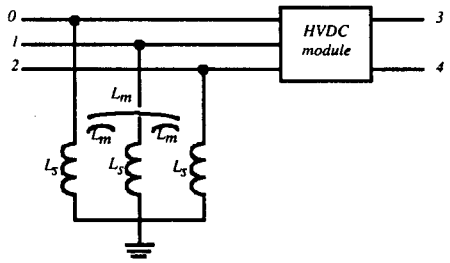


Fig. 127: Non-saturated magnetization in three phase core transformers.

11.4 Modification of the HVDC-module model to include saturation

First, let us consider the non-saturated magnetization branch, *NSB* (only for three-leg cores), and its relationship with the linear HVDC-module model, Fig. 127. From the open circuit test data provided by the user, Y^o and Y^+, one determines the coupled group of L's, the *NSB*, as represented by the admittance matrix $[Y_{ns}]$, Eqs. (186, 187, and 188).

$$Y_s = \frac{Y^o + 2Y^+}{3} \tag{186}$$

$$Y_m = \frac{Y^o - Y^+}{3} \tag{187}$$

$$[Y_{ns}] = \begin{bmatrix} Y_s & Y_m & Y_m \\ Y_m & Y_s & Y_m \\ Y_m & Y_m & Y_s \end{bmatrix} \tag{188}$$

The inductance matrix, or rather its inverse, $[L]^{-1}$, is readily obtained from Eq. (188) result as explained in Eq. (189) below.

$$[L]^{-1} = 2\pi f \cdot [Y_{ns}] \tag{189}$$

Where f, in hertz, is the frequency of the open circuit test.

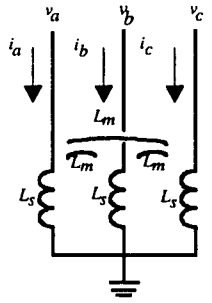


Fig. 128: Phase voltages and non-saturated magnetization currents.

As we are representing the non-saturated magnetization phenomenon by a group of coupled inductances as seen in Fig. 127, the relationship between phase voltages (grouped in vector $[v]$) and non-saturated magnetization currents (grouped in vector $[i]$) is:

$$[v] = \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix}; \quad [i] = \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix}, \quad [v] = [L]\frac{d}{dt}[i] \tag{190}$$

Where $[L]$ is the inductance matrix whose inverse has been obtained in Eq. (189). Integrating Eq. (190) between instants $(t - \Delta t)$ and t, and simplifying notation: $[v] = [v(t)]$ and $[v'] = [v(t - \Delta t)]$, same for currents.

$$\int_{t-\Delta t}^{t} [v] \cdot \mathrm{dt} = [L]\,([i] - [i'])\, \tag{191}$$

Next, we approximate the left-hand side integral in Eq. (191) using backward Euler's rule, and obtain Eq. (192).

$$[v] \cdot \Delta t = [L]\,([i] - [i'])\tag{192}$$

Premultiplying Eq. (192) by $[L]^{-1}$, from Eq. (189), and solving for $[i]$, the non-saturated currents at the end of the active integration step:

$$[i] = \Delta t \cdot [L]^{-1}\,[v] + [i']\tag{193}$$

Defining the matrix $[G_{ns}]$ and the vector $[h_{ns}]$ as in Eq. (194), Eq. (193) can be rewritten in its canonical form, shown in Eq. (195).

$$[G_{ns}] \equiv \Delta t \cdot [L]^{-1}; \quad [h_{ns}] \equiv -[i']\tag{194}$$

$$[i] = [G_{ns}][v] - [h_{ns}]\tag{195}$$

Inclusion of this $[G_{ns}]$ matrix into the $[G]$ matrix of the HVDC-module is illustrated in Fig. 129.

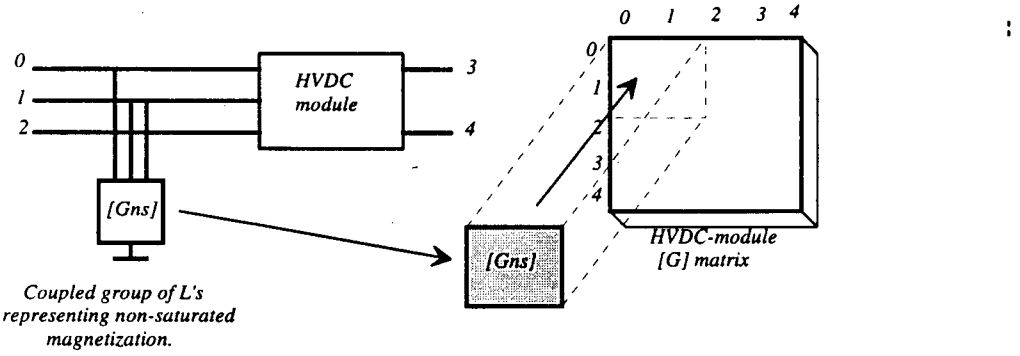


Fig. 129: Including the non-saturated magnetization matrix, $[G_{ns}]$, into the HVDC-module $[G]$ matrix.

When any leg becomes saturated, an inductor with a value L_{sat}, supplied by the user (see § 11.1 in this chapter, pag. 161), is connected between either node 0, 1, or 2 (depending on which leg became saturated) and ground. See Fig. 130. In this case the equivalent conductance, using the backward Euler's integration rule:

$$g_{sat} = \frac{\Delta t}{L_{sat}} \tag{196}$$

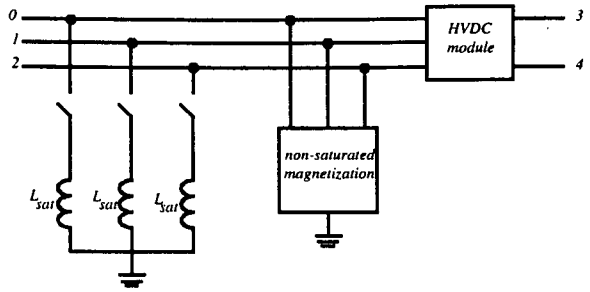


Fig. 130: Modelling saturation in the core.

is added to the diagonal element in the module's $[G]$ matrix, corresponding to that node. The inclusion of the three switches outlined in Fig. 130, raises the number of possible status of the whole extended module (that is, including the magnetization effect) from 2^6 up to 2^9. To represent the status of the extended module, three more bits will be necessary in the status word, which brings the number of precalculated G-matrices per HVDC module up to:

$$2^{(6+3)} = 512 \text{ matrices}$$

Those matrices continue to be 5×5 ones, which in double precision representation amount to:

$$5 \times 2 \times 8 \times 512 = 40{,}960 \text{ bytes} = 40 \text{ kbytes}$$

a small memory investment for the enormous performance benefit obtained.

11.5 History sources introduced by magnetization modelling

Six new current history sources need to be included and updated by the model. One for each saturated branch, and three from the coupled group of non-saturated magnetization model, as shown in Fig. 131.

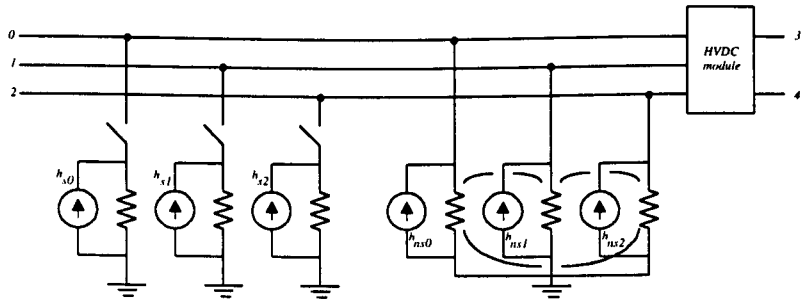


Fig. 131: The six history sources introduced to model magnetization in the transformer.

Each of the saturated branch history sources, h_s, is updated, after using backward Euler's rule for discretizing the corresponding inductance, by the formula in Eq. (197), where g_{sat} is defined in Eq. (196) for inductor k, where $k = 0, 1, 2$.

$$h_{sk}^{next\_step} = h_{sk} - g_{sat} \cdot v_k \tag{197}$$

For the non-saturated modelling group, the three history sources, connected as in Fig. 131, are updated by Eq. (198) —again using backward Euler's.

$$[h_{ns}^{next\_step}] = [h_{ns}] - [G_{ns}][v] \tag{198}$$

11.6 Effect of the saturation modelling in the primary current

In Fig. 132 below, the current in the phase a of the primary of an HVDC module transformer is shown ignoring saturation of the core, that is, a linear core transformer is assumed.

In Fig. 133, for the same situation depicted in Fig. 132, the saturation of the transformer core has been modelled as described in this report, and occurrence of *peaks* and *valleys* in that current corresponds to the expected results.

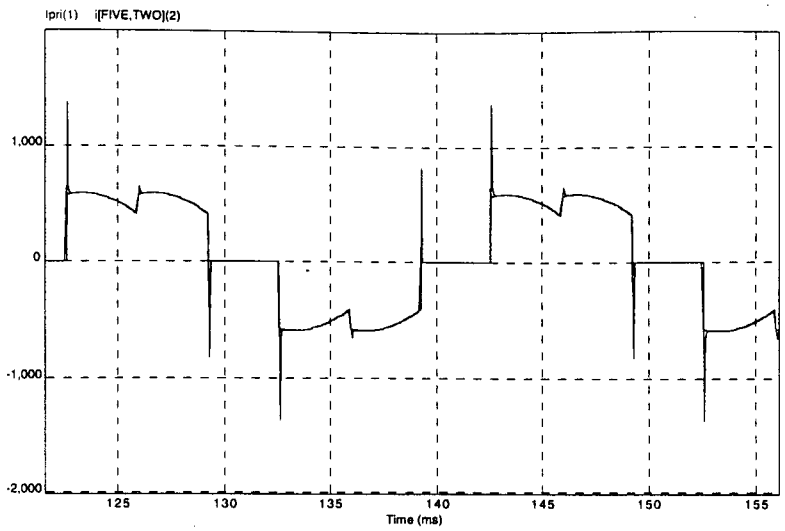


Fig. 132: Primary current with a linear core under steady state conditions, OVNI's model and EMTP simulation. The large spikes belong to OVNI's before DSDI, §7.7. Microtran/EMTP avoids them using CDA [10].

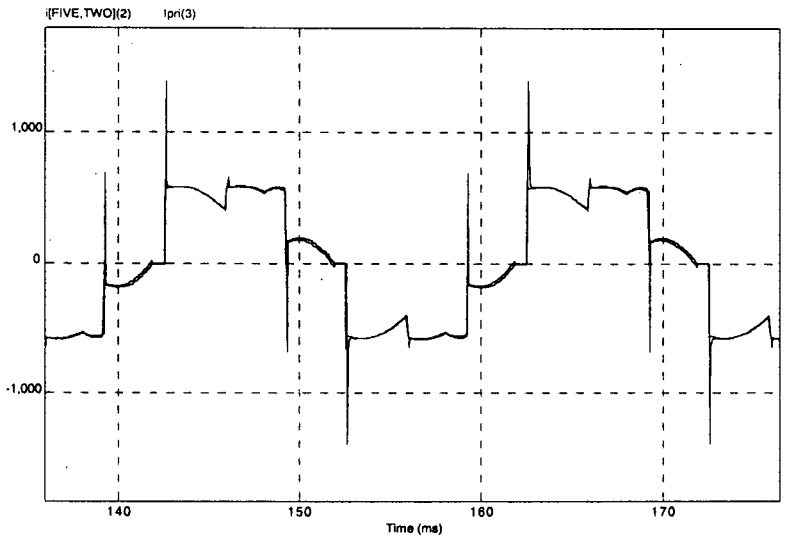


Fig. 133: Primary current with a saturated core under steady state conditions, OVNI's model and EMTP simulation. See caption to Fig. 132.

Part V

IMPLEMENTATION

12. OVNI, THE SIMULATOR'S ENGINE

12.1 Introduction

The best, up to date, and most complete description of the core is the code itself. As the next best thing, this chapter describes succinctly the implementation of the integrator proper. OVNI is an OOP application written in C++[1].

Each one of the major parts of the solution: clock, network, events, blocks, subblocks, elements, history sources, etc., is represented in OVNI as an object, an instantiation of some class, as described in what follows.

12.2 Input Data File

Data to the engine comes in human readable format. That data file is created by OVPP, the preprocessor, described in the previous chapter.

The file presents several labelled "environments[2]." Each environment starts with the keyword .BEGIN followed by a label that identifies the particular environment: GENERAL, ELEMENTS, BLOCKS, EVENTS, etc. Each environment finishes with the keyword .END and the same label used in the opening .BEGIN of that environment.

Inside most environments, there are subenvironments, for instance, in the ELEMENTS environment there is a subenvironment for transmission lines, .BEGIN LINES ..... .END LINES, and inside again a separate subsubenvironment for each particular line: .BEGIN LINE-3 ..... .END LINE-3.

Data items inside environments begin with a label separated by a colon from its value, for example: *charac_impedance_per_mode:* 724 615 615.

All of this makes the input file easy to read for the user, if need arises, but not to write, which is the complex job of OVPP.

---

[1] The first version of its core was written originally in Ada95 [67].

[2] Which are akin to LaTeX's environments [71].

12.3 Names in OVNI

A uniform hungarian [70] notation was used in labelling variables and types in the core's code. Variable names begin with a lowercase letter, functions with an uppercase one, constants enjoy a full uppercase name, type descriptors end with an underscore-tee. The different entities isolated by the solution are abbreviated by a three letter code, as follows in the extract from the code included in Fig. 134.

```
//----------STANDARD ABREVIATIONS----------
// BLK = block.
// CHM = chameleon.
// CSR = current source.
// ELM = element.
// EVN = event (switching event, etc.)
// HSR = history source.
// LNK = link.
// NET = network.
// NOD = node.
// NUM = number (as in number of items. ex.: numNod, numBlk, etc.)
// PAR = (suffix) parameter (to a function, when name is ambiguous.)
// SCH = switch.
// SIM = simulation.
// SRC = source.
```

Fig. 134: Standard abbreviations in OVNI.

The hungarian notation prefixes adapted for use in the code can be seen in Fig. 135, below.

```
//----------VARIABLE NAMES CONVENTIONS----------
// This program uses the "Hungarian" convention to name its variables.
// In particular, variables whose names start with a lower case:
//   p = pointers. Ex.: pNod, is a pointer to a node structure.
//   a = array.    Ex.: aNod, is an array of node structures.
//                      apNod, array of pointers to node structures.
//   i = index into an array. Ex.: iNod, index into an array of nodes.
//   g = global variable.
//   m = module variable. (Gobal within the module).
//   c = count.    Ex.: cNodPend, count of nodes that need processing.
//   e = element of an array.
//   d = difference between two variables of the same type.
//   x = parameter to a function. Used when there is ambiguity.
```

Fig. 135: Hungarian notation prefixes as used in OVNI.

Also, modular variables (those visible only within a C++ file) begin with a lowercase *m* letter. Global variables, when they exist at all, exhibit a lowercase *g* as the first letter of their names. As an example, the network node registry array, to be

introduced in the next section, is a module visible variable within the *net_t* file: *mapNodZer*. It is a module visible (m), array (a), of pointers (p) to nodes (nod), indexed from zero up (zer).

12.4 From nodes to the network

The simplest and most basic entity in OVNI's description of the network is the node. A node was represented originally by the structure in Fig. 136. Two pieces of data define the state of a node, its voltage, *v* (defined with respect to the reference node), and its total current, *h*.

```
struct nod_t{
   double h;      // Total current entering the node.
   double v;      // Voltage to reference node.
};
```

Fig. 136: Structure that represented originally a node in OVNI.

The network simulated in OVNI is a conglomerate of nodes associated according to a connectivity matrix defined by the elements. Nodes connected galvanically,[3] but not including both ends of any link, are clustered together into an array which is put inside a subblock entity, an instantiation of the class *sub_t*, as in

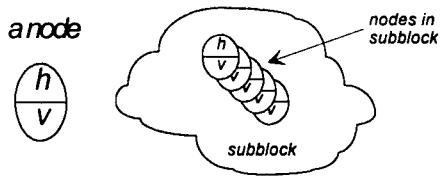


Fig. 137: Node array inside a subblock object.

Fig. 137.

Regardless of where the actual nodes are (each as an element of an array in one of the subblock objects described later in this chapter), to keep track of each one of them, the network (unique instantiation of the class *net_t*) maintains an array of pointers to the nodes which is initialized right before the simulation begins, the

[3] That is, not connected to both ends of the same transmission line.

```
//————————————— NOD_T ————————————————-
// Every node in the network is accounted for in an array, mapNodZer,
// which is a data member of the "net" class. Each node corresponds
// to one element into this array. In this implementation the array's
// name is 'mapNod' (an array of pointers to structures 'nod_t).
// The total number of external nodes in the network (nodes which are
// visible to the core of the integrator, as oppossed to nodes inside
// the models themselves). Ground, or reference node, corresponds to
// the zeroth element, mapNodZer[0]. Example: to refer to the actual
// name of the node whose index is 6... mapNodZer[6]->name.
// Also, ground/reference node is assigned block #-1, and sub-block #-1,
// which of course is a non-existing blk/sub.  (See definitions above).

struct nod_t
{
   char sName[MAX_LENGTH_NODE_NAMES+1]; // Actual name of the node.
   int  iBlk;              // Which topological block it belongs in.
   int  iSub;              // Which sub-block inside that 'block'.
   int  iPosInSub;         // Relative position of node within sub-block.
   REAL *pH;               // Pointer to actual node current field.
```

Fig. 138: A node registration item, an element of the node registry array.

network nodes registry, *mapNodZer*. The array element is seen in Fig. 138, and the

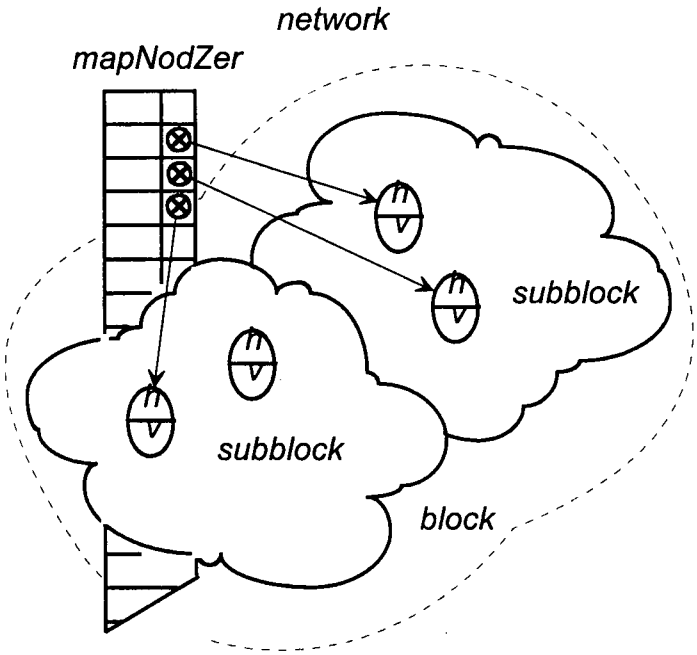


Fig. 139: Network registry of nodes, and their spatial relationship with the nodes, subblocks, blocks, and the network.

```
//———————————— HSR_T ————————————————
// Every external history source is one instantiation of this class.

class hsr_t
{
private:
   REAL value;          // Actual value of the current source.
   REAL *pNodFromH;     // Pointer to actual 'from' node current field.
   REAL *pNodToH;       // Pointer to actual 'to' node current field.
public:
   REAL *GiveAdrHsr() {return &value;};
   void SetFromNodH(REAL *p) {pNodFromH=p;};
   void SetToNodH(REAL *p)   {pNodToH  =p;};
   void DrainAndPour() {*pNodFromH-=value; *pNodToH+=value;};
```

Fig. 140: The external history source class, *hsr_t*.

registry array itself and its spatial relationship with the nodes, in Fig. 139.

The next most basic entity in OVNI is the history source, an object instantiated from the class *hsr_t*, seen in Fig. 140. History sources are allocated at the request of the corresponding element, and clustered together into an array under the supervision of the network object itself. Each history source is granted access to the current field of each of its nodes (the *h* field in Figs. 136 and 137). And, when fulfilling the request for allocation, the network provides the client element with access to the "value" field of the source. This allows for the element updating of its history sources without the overhead of message passing imposed by a more orthodox OOP[4] approach. See Fig. 141.

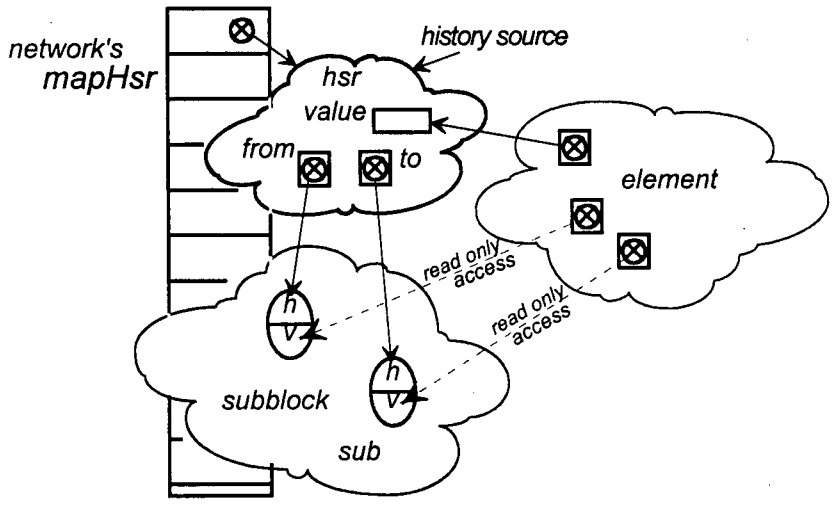


Fig. 141: Relationship among the elements, their history sources and the subblock's

In increasing order of complexity, our attention now turns to the element class, to the models of the physical elements that embody the network. Element models, in their immense and ever growing variety, are not part of the simulator core. But being the intense centers of activity they are, the data exchange with the core, and a generic outline to a highly efficient implementation of models was provided in a previous chapter (under the heading of node-hiding). That outline is implemented as

```
//------ELM_T------------------------------
// Every element in OVNI is a descendant of this common abstract class.
//
class elm_t{
public:
   virtual void ReadData(FILE *f);
   virtual void UpdateHsr();
protected:
   INT cNumXNod;              // Number of external nodes.
   INT cNumXHsr;              // Number of external history sources.

   INT *aXNodId;              // External nodes, as id'd by the network.

   pConsReal_t *apXNodV;      // Pointers to the X-nodes voltages.
   REAL **apXHsr;             // Pointers to the external history sources.

   void GetBasicData(FILE *f);
```

Fig. 142: The element abstract class *elm_t*.

an abstract[5] class *elm_t*, Fig. 142, from which more concrete ones will be derived as descendants, inheriting in the process the service definitions necessary for the operation of the core. The core will exchange messages and services with the elements through the common interface provided by that abstract class. Every element model is kept track of by a pointer in an element registry array maintained by the network object, *mapElm*. The relationship of those elements to the external nodes they are connected to, and to the external history sources they are contributing to the network, is seen in Fig. 141.

Subblocks are associated to one another either by links (in the MATE sense), or by transmission links. Subblocks connected by links are said to belong in the same block, according to the convention established in the discussion on MATE segmentation. In this sense, a block can be perceived as a cluster of subblocks. In OVNI's

[4] Object Oriented Programming.

[5] In OOP parlance, an abstract class is one from which no actual objects are created.

current implementation, a block is an instance of class *blk_t*, and contains an array of subblocks, *aSub*. The links that determine the cluster of subblocks identified as a block are described by a simple structure with fields that identify the subblocks and nodes it is connected to, its resistance (if any), and its associated voltage source (if any). Such links are grouped in an array inside each corresponding block object. Independent voltage sources associated to links are part of the block, they are accounted for as an array of structures with vectors of precalculated values, with an associated number of samples, and the ubiquitous time index. Grounded voltage sources belong inside subblock objects, and include not only the array of precalculated values, the number of samples, and the time index, but also all the precalculated components outlined in section 9.4.1.

12.5. Classes in OVNI

Several of the classes in the engine have a unique instantiation: network, clock, simulation, event handler. They were developed originally as Ada83 modules that became ADT's modules in C++. All other classes spawn multiple objects representing every block, history source, element, etc. This last group is implemented as C++ classes.[6]

OVNI's classes description begins with the entity of the problem space that suggested OOP technology as a convenient paradigm to implement the solution, the element.

12.5.1 The Element Class, *elm_t*

Decades of EMTP experience have made it clear that an application of the nature of OVNI's is bound to start with a core, presented in this thesis, and develop with contributions of element models coming from several sources, and at different times in the future.

As model developers plug their creations into OVNI, the core must be able to continue to operate and remain unchanged. That unburdens model developers, and protects the core from careless unwilling introduction of errors[7] into its main code.

OOP promise of *encapsulation* provides for such protection. However, given

[6] They were Ada95 tagged records, in the original conception of the solution.
[7] The euphemistically called *bugs*.

allow for (in theory) all future possibilites of the needs of models, as supplied by the core. This need is provided by OOP's *polymorphism*, that makes it possible for OVNI to deal with all element models, present and future, as if they were "forms" of an abstract sort of element, an abstract class in OOP parlance, an *elm_t*. All elements, included and to be developed, are instances of classes that derive their functionality from that abstract class. Class *elm_t* is a common ancestor to all element classes in OVNI.

Interaction between the core and the element models include: requests by the core (the client) for the element (the server) to update the external history sources that the element contributes to the network around it; request by the *chamaeleonic* (non-linear) element (the client) to the core (the server) to update the corresponding subblock conductance matrix due to the element's recent topological change; request by the element (client again) to the core (server) to provide the addresses (without permission to write) of the voltage fields of the nodes that the element is connected to; request by the element (client) to the core (server) to provide the addresses (with writing permission) of the current fields of the history sources that the element will have to update at every time step, when so required by the core (see the first service listed here.)

The interaction between an element and the rest of the network occurs at the so called *external* nodes. The element keeps track of the number of nodes that connect it to the rest of the network (external nodes), the number of history sources that the element contributes to those external nodes (external history sources), and identification of each of those nodes, and of each of those history sources. In the current implementation, to improve the performance of the solution, elements are kept abreast of the nodes voltages and allowed to update the corresponding external history sources without incurring into method calling overhead. In short, elements are provided by the core with pointers to constant values that take them to the nodes (which belong inside a certain subblock object) voltage fields directly, but prevent them from modifying those values unwittingly. Otherwise, the element would have to request the corresponding information service from the network, that would pass the message to the block in question, who in turn would advance the message down to the subblock that contains the node, a lengthy and expensive message passing. The pro of the taken approach is improved efficiency, the cons include the waiving

of some independence between the classes. But that last price has been kept at a minimum by making the data accessed by the element, atomic, that is, of an unstructured type (floating point values).

12.5.1.1 Methods provided by *elm_t*

In Fig. 142, a view of the abstract ancestor class of all the elements in OVNI, *elm_t* is included. In that figure, the arguments common to all elements in OVNI can be identified: the number of connection nodes (external nodes), the number of history sources the element contributes to the network (external history sources), an array of pointers to the external nodes voltages (visualized in Fig. 141), necessary for the element during the updating of its history.

Two services are provided by the class *elm_t*:

a) *ReadData*, a request issued to the element during the initialization of the case that the element acknowledges by reading its data from the input file, and initializing all of its data structures. At that stage also, the element requests of the network, *net_t*, the addresses of the voltage fields corresponding to the connection nodes of the element, and the addresses of the current fields of the history sources that the element is feeding into the network;

b) *UdateHsr*, a request issued by the network during the simulation, at each time step, to take the nodal voltages and recompute the history sources of the element (at this stage, also, the element decides its topological changes, if any, and notifies the network, through the corresponding messages —see the section on the network, below— of the necessary changes affecting the corresponding subblock matrices.

12.5.2 The history source class, *hsr_t*

History sources, even if conceptually belonging within the element models, are represented as objects of the class *hsr_t*, and are grouped together inside the network object, ready to service the network in its request for accumulating nodal currents. The link with the corresponding element model is made once, at the initialization stage, when the source provides the element with the address of its current field, see Fig. 140.

Fig. 140.

12.5.3 The subblock class, sub_t

Each of MATE's subblocks, as described in a previous chapter, is implemented as an instantiation of class *sub_t*, Fig. 143.

```
class sub_t{
public:
   // Gets sub's data from file 'f'.
   void  Init(
      FILE* f,                          // Already opened and posit. file.
      int iBlkCode,                     // Pos.enclosing block in net.
      int iSubCode );                   // Pos.of subblock in enclosing block.
   void  UpdateEth();                   // Update Thevenin's voltages.
   const REAL** GetPtrZMatrix();        // Delivers address of Z mat.
   const REAL *GetPtrEthVector();       // Delivers address of Eth vec.

   // Given the enveloping block's links currents in vector 'aLnkVolts',
   // do calculate the subblock's nodes voltages.
   void CalcNodeVolts( const REAL* aLnkCurr );

   // To allow the block to register its links, the block passes
   // the number of links, and the two allocated and initialized
   // vectors 'aLnkNod' and 'aLnkEnters'.
   void RegisterLnk( INT cNumLinkPar, INT *aLnkNodPar, BOOL *aLnkEntersPar );

   // To hook up a new 'aZ' matrix by the corresponding chameleon elm.
   void HookUpMatrix( REAL **aNewZMat );

private:
   INT    iBlk;                        // In which blk is this sub.
   INT    iSub;                        // What sub is this in that blk.
   INT    cNumNod;                     // Number of nodes in sub.
   INT   *aNodId;                      // Net's indentifiers for the nodes.
   REAL  *aNodV;                       // Nodes voltages (1..n).
   REAL  *aNodH;                       // Nodes total currents (1..n).
   REAL **aZ;                          // Z matrix, i.e. inv( G[] ).
   REAL  *aEth;                        // Thevenin voltages (1..n)

   INT    cNumLnk;                     // Number of links in block.
   INT   *aLnkNod;                     // Links connection nodes in sub.
   BOOL  *aLnkEnters;                  // TRUE = Link enter this sub.

   BOOL   chameleon;                   // TRUE = this subblock is a cham.
```

Fig. 143: The subblock class *sub_t*.

12.5.4 The block class, blk_t

Subblocks are clustered into an array inside the corresponding block, together with the corresponding links, etc. As can be seen in Fig. 144.

```
class blk_t
{
public:
   blk_t(FILE *f,INT iBlkCode);
   void HookChamMatrix(INT iSub, REAL **pNewMatrix);
   void HookVoltToLnk(INT iLnk, REAL *p);
   void CalcNodeVolts();

protected:
   INT blkId;           // Identification code of this block.

   INT cNumLnk;         // Number of 'links' in this block.
   INT cLnkClosed;      // Number of closed links in the block.
   lnk_t **apLnk;       // Array of pointers to 'links' in this blk.
                        // Open 'links' are at the bottom of the array.
   lnk_t **apLnkId;     // Same as 'apLnk', but invariable. To id. them.

   // The following arrays are indexed according to the order of links
   // as in 'apLnk', that is closed links at the top.
   REAL **aLnkMat;      // 'link' matrix for the block.
   REAL *aLnkCurr;      // Array of links currents.
   REAL *aLnkVolt;      // RHS of link current equation system.

   REAL *aCholDiag;     // Auxiliary vector used in solving links equations.

   // This array follows the order in 'apLnkId'.
   REAL *aLnkCurrUnsorted; // Same as 'aLnkCurr' but with the order of
                        // the links as created by the preprocessor.

   INT cNumSub;         // Number of 'sub-blocks' inside this block.
   sub_t **apSub;       // Array of pointers to sub-blocks.

   boolean chmChanged;  // TRUE=one of block's chameleons changed!
   boolean swtEventOccurred; // TRUE=a link just closed or opened.
   boolean lnkMatJustRebuilt; // TRUE=aLnkMat has not been triangularized.

   void SortLnk();      // Puts closed links first in array 'apLnk'.
   void BuildLnkMat();  // Build the links matrix.
   void HandleTopologyChanges(); // When links operate or chameleons change
```

Fig. 144: The *blk_t* class, template for every block in the network.

5.5 The clock object, tck

The clock is a single instantiated object, implemented thus as an ADT walled inside its implementation file "tck.cpp", and interfaced to the rest of the core by its header "tck.h", as in Fig. 145.

The clock unit defines its own integer arithmetic, which allows it to count, with one microsecond steps, up to 3.2×10^{974} trillion years. This accounts for the non-measurability of the continuous simulation premise, that is, on-line monitoring of power networks, envisaged as one of the applications of this thesis.

```
typedef long count_t;              // 0..999 999 999 (1 bil minus one).
                                   // Note: this type does not enforce
                                   // these limits, so it's up to prog.

void TckResetCounter();            // Sets to zero the counter.
void TckAdvanceCounter();          // Advance count by one.
void TckGetCounter                 // Returns...
       (count_t &xHigh,            //   high nine digits of ticker,
        count_t &xMid,             //   and middle nine digits,
        count_t &xLow);            //   and least signif. nine digits.
void TckSetMax                     // Sets the maximum tick limit at:
       (count_t xHigh,             //   high nine digits of ticker,
        count_t xMid,              //   and middle nine digits,
        count_t xLow);             //   and least signif. nine digits.
boolean TckCountBelowMax();        // TRUE = Simulation has not ended.
```

Fig. 145: Header of the clock object, the ticker, *tck_t*.

12.5.6 The simulation object, *sim*

The simulation is the top level object, right under the director[8]. As a single instantiation entity, it was implemented as an ADT encapsulated inside a C++ file, *sim.cpp*, and providing services to the rest of the implementation as defined in its header *sim.h*, which is included in Fig. 146.

```
void SimInitialize                 // Reads file/netwrk, resets timer.
        ( char *nameInFile,        // Name of input file (the netwkr).
          char *nameOutFile,       // Name of the mail output file.
          char *nameLogFile );     // Name of the event/log file.

void SimDoTheLoops();              // Loop along time axis, doing the
                                   // simulation proper.
```

Fig. 146: Services provided by the simulation object.

12.5.7 The network class, *net*

The network, *net*, under the command of the simulation object, *sim*, and in continuous consulting with the clock object, *tck*, and with the event handler object, is the great activator in OVNI's core. It (the net) contains the array of node registry, that points into every external node in the system, *mapNodZer*; the array of elements in the system, *mapElm*; the array of blocks, *mapBlk*; and several other bits.

[8] The OOP section of the code that issues the cues for the object-actors to perform.

12.6. How classes within OVNI relate to each other

Up to this point, most of the classes and entities in OVNI have been introduced, and some inkling of the way they relate to each other has been intimated. In Fig. 147, the inclusion relationship is sketched. There the simulation, *sim*, commands the network, *net*, to consult the event manager, *evn*, and the clock keeper, *tck*, at the highest level of the process.

The network, *net*, contains the elements (as forms of *elm_t*), the blocks (as instantiations of *blk_t*), the history sources (instantiations of *hsr_t*). The blocks contain the subblocks (instantiations of *sub_t*), the links (of type *lnk_t*). The sub-blocks contain the nodes, the grounded voltage sources, the independent current sources, and the non-link switches.

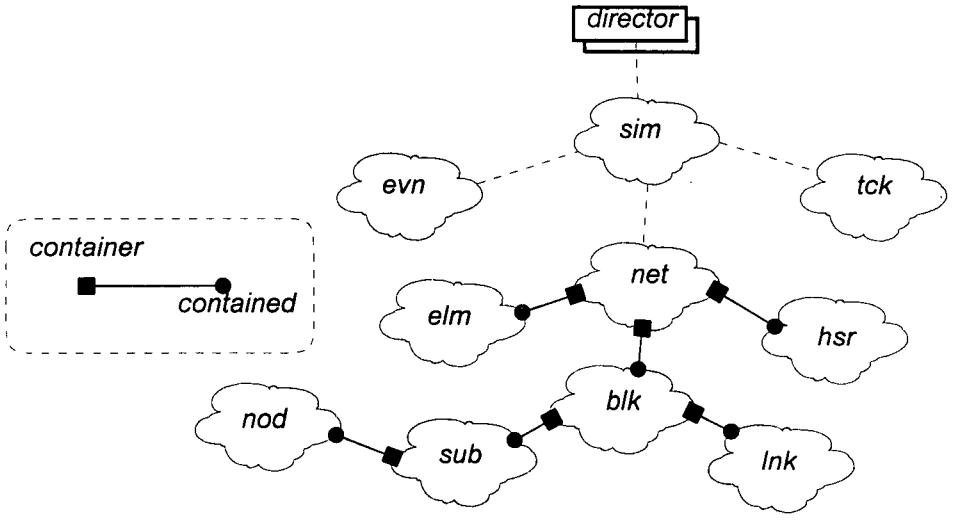


Fig. 147: Container/contained relationship of classes in OVNI.

12.7. Main tasks of the simulator's engine

In a previous chapter of this thesis, the main tasks were revised. In this section the implementation of those tasks, as services requested by some object and provided by some other object among those described in the fist part of this chapter, follows.

12.7.1 Initialization

During initialization, the director activates the simulation, *sim*, through the service request *SimInitialize,*[9] and passes to it the names of three files: the input data file, the log file, and the output file (used to validate the solution, or to keep record of some variable, not in real time simulations). The simulation checks the existence of the input file, and opens the three files.

The simulation reads the general data associated with the present run and initializes the clock, *tck*, through the service *TckSetMax* and *TckSetDeltaT*.

Next, the simulation requests the network, *net*, to read its data and initialize its data structures, through the service request *NetInitialize*. The network proceeds to read its data, and sets up its internal structures for registration of nodes, for history sources, for elements, for blocks, etc.

Finally, the network signals each element to read its data and proceed with model initialization, through the request *pElm->ReadData* (where *pElm* is a pointer

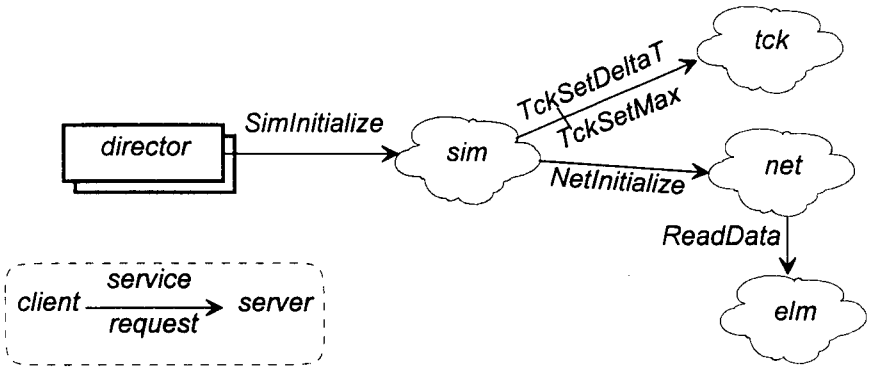


Fig. 148: Initialization Task of the Engine.

to a particular element).

12.7.2 Simulating the case

The centre of activity of the core is the *loop* itself, the set of subtasks that are

[9] The "dot" notation is conspicuously absent of this first stage, since the objects involved (sim, net, tck, evn) were implemented as ADTs and not as instantiations of a class. As a matter of notation, in such cases, OVNI's code maintains the three letter acronym of the object as the prefix of all its methods (service requests).

executed at each simulation time step. At this stage, right after the initialization task, the director requests of the simulation object to *SimDoTheLoops*. Each of the subtasks comprised by this service request was a major issue of the project and deserve its own subsection, as follows.

12.7.2.1 Updating History Sources

At the begining of each simulation step, the network *net*, traverses its register of elements in the case, and requests of each to update the history sources that belong to it. This is done through the service request *UpdateHsr*.

During the service to this request, each element goes through some common steps:

a) Grab its external nodes voltages —available to it directly since the registration part of the initialization—;

b) Through the node hiding equations described in chapter 5, determine the voltage of its internal nodes for the current time step;

c) Determine the necessary changes to its topology, if any, and submit (if necessary) a request to the network to receive a new "contribution" matrix and to "hook" it to the corresponding subblock matrix;

d) Calculate the external and internal history sources, and again through the node hiding equations mentioned above, refer all of them to equivalent external history sources that are then delived directly to the corresponding history source object —the element has direct acess to its history sources value fields since the registration stage of the initialization, as was seen in a previous section of this chapter—.

12.7.2.2 Accumulating nodal currents

Once the value for each current source in the case is known (independent and history souces), the version of the nodal analysis method used in the core solution requires that the currents being fed to each node are summed into a node total, the nodal currents. Each node accumulates this total current in its *h* field, see Fig. 137.

At this stage, in service of the request *NetAccumNodalCurrents* (issued by the simulation object *sim*) first the network *net*, clears the current fields of all the

external nodes in the case (the network gains access to them, through the data in its node registration record array *mapNodZer*), then the *net* requests of each of its registered history source objects, the service *DrainAndPour*. Each history source was given direct access to its two nodes *h*-fields during the registration stage of the initialization, so the history source can add its value to the *h*-field of its destination node and subtract it from the corresponding field of its origin node.

12.7.2.3 Solving for nodal voltages

The task of solving for voltages at the nodes has been the major concern at the outset of this project, it befits this task to end the description of the implementation of the core. The simulation *sim*, issues the service request *NetCalcNodalVoltages* to the network *net*. The network traverses its registry of topological blocks and requests of each one of them to determine the voltages of the nodes it encloses through the service request *CalcNodeVolts*.

```
//-------------------------------------(4)
// It is the blocks who know how to calculate the node voltages, gi-
// ven the node currents. As blocks are contained in the network,
// the last one commands each of its blocks, one after another, or
// all at the same time, if this procedure is spawned among several
// processors. There must not be any data contention, since the
// only written data is that of nodes voltages, and those voltages
// belong inside each block's sub-blocks.

void NetCalcNodalVoltages()      // Determine the voltage of each node.
{
   int i;

   for(i=1; i<=mcNumBlk; i++)    // Requests to each block to ...
      mapBlk[i]->CalcNodeVolts(); // ...compute its nodes voltages.
}
```

The block[10] so activated enters in its MATE computation cycle (as described in a previous chapter):

a) If during the previous integration step the event handler *evn* issued a topology change signal, the block reshufles its links, to keep the closed ones at the top of its list, which allows for the same Cholesky solution method procedure to be applied to each of the open-close-links combinations. To do this without losing track of which link is which, the block maintains two arrays of pointers to the enclosed links,

[10] If the block contains a single subblock (no internal subsegmentation), it issues the subblock the service request *apSub[1]->CalcNodVolt()*, and that is all that is necessary at this stage.

links, a fixed one, that is set and left during the initialization process, that is used to access the currents and state of the link at any moment, and a movable one, that is reshufled to keep on top pointers to the closed links (the so called *active* ones). In this case also, the block needs to recompute its links matrix, by quick polling its contributing subblocks according to the latest topology as determined by the links status. Another possibility contemplated at this stage is that there was no external event serviced, but one of the internal *chamaeleon* (non linear) elements changed topology, in this case the block simply recomputes its links matrix;

b) The block signals each of its subblocks to establish the latter's Thevenin voltages of its nodes, service *apSub[i]->UpdateEth()*;

c) The block polls each of its subblocks to contribute with the corresponding recently calculated Thevenin voltages to its *right hand side* vector of the links current system of equations;

d) The block solves the links system of equations for the links currents;

e) Finally the block passes the links currents to each of its subblocks and requests of it to produce the corrected nodal voltages, subblock method *apSub[i]->CalcNodeVolts(apLnkCurrUnsorted)*;

13. OVPP, THE PREPROCESSOR

13.1 Introduction

The simulator's core was implemented in two main modules: the preprocessor, OVPP, described in this chapter; and the engine, OVNI, described in the next chapter.

The preprocessing stage of the simulator, OVPP, is an OOP[1] application built around one custom designed circular double-linked list class, *list_t*, and five descendent classes: the node-list object-class, the subblock class, the subblock list class, the block class, and the block list object-class. The preprocessor was developed[2] in C++ and compiled on the target machine with the GNU g++ compiler.

The operative word in the design of the simulator has been *precalculation.* Most of the precalculation involved has been moved to the preprocessing stage to keep the engine lean and fast. This chapter begins with a description of the raw data input file to the preprocessor, and continues with a description of the preprocessor proper.

13.2 The Preprocessor Input File

The input data to the preprocessor is in human readable form for the convenience of the case creator. It is in a free format text file with the extension *OVP*. The basic template of such file can be generated with the included utility OV-TMP. The indentation shown in the following example is there to make reading easier

[1] Object Oriented Programming [34]

[2] Originally OVNI began as an OOP project developed on Ada95 [67], but delays in the availability of industrial strength compilers forced the move toward C++. [68]

for the human user, it is not necessary for the program, but recommended.

The significant data in the file begins with the keyword *.BEGIN_FILE.* Before that keyword, any comment can be included for identification purposes. The data in the file is divided into nine sections[3]: *GENERAL DATA, LUMPED, LINES, HVDC, CONTROLLERS, COUPLED, SOURCES, SWITCHES, OUTPUT.* Each section ends with the keyword *.END* followed by the corresponding section label. Some of the data items are preceded by a label, included by the utility OV-TMP[4]. Each of those labels explains the meaning of the following data item (see Fig. 149 on p. 191).

13.2.1 General Data

The *GENERAL_DATA* section includes two data items: the one labelled "deltaT:" for the integration step in seconds; the other identified "totalTime:" for the total simulation time, also in seconds.

13.2.2 Lumped Elements

Following the respectable tradition of the EMTP (see chapter 1), even though all element models, with the single exception of the distributed transmission line model, are represented as lumped equivalents, OVNI refers to lumped linear resistors, inductors, and capacitors, as lumped elements.

This section begins counting the total number of lumped elements (linear resistors, inductors, and capacitors) in the network[5], "number_of_lumped:". For each lumped element, a line[6] that includes (see Fig. 151), without labels, an uppercase letter R, L, or C, depending on the nature of the element; a parameter, in ohms, milihenrys, or microfarads, depending on the nature of the element;

[3] This number of sections will increase as more models are attached to the simulator.

[4] The preprocessor input file, *.OVP, is to be generated by OVNI's graphic user interface, OUI [37]

[5] In a future version of the preprocessor this data item, along with the similar ones for other element types, will be dropped.

[6] A separate line is not absolutely necessary, given the free format of the file, but highly convenient for the human reader.

```
.BEGIN FILE
   .BEGIN GENERAL_DATA
      ...
   .END GENERAL_DATA
   .BEGIN LUMPED
      ...
   .END LUMPED
   .BEGIN COUPLED
      ...
   .END COUPLED
   .BEGIN LINES
      ...
   .END LINES
   .BEGIN CONTROLLERS
      ...
   .END CONTROLLERS
   .BEGIN HVDC
      ...
   .END HVDC
   .BEGIN SOURCES
      ...
   .END SOURCES
   .BEGIN OUTPUT
      ...
   .END OUTPUT
.END FILE
```

Fig. 149: General structure of the preprocessor input file.

and the two nodes that the element is connected to, each node as a string of up to six letters (The ground or reference node can be entered as either GND, GROUND, or EARTH.)

13.2.3 Intrablock "links" and Switches

Switches in OVNI can be represented by either a MATE intrablock link, or an ideal or resistive intrasubblock switch. Two data sections accommodate each of those categories. The *SWITCHES* section includes labels for: "number_of_switches:", and for each of the switches included in this section: "initial_node:" (a seven character string name), and "final_node:", for the two nodes

```
.BEGIN GENERAL-DATA
   delta_t:     50.0e-6
   total_time: 50.0e-3
.END GENERAL_DATA
```

Fig. 150: Section on general data for a case with an integration step of fifty microseconds and a total simulation time of fifty miliseconds.

```
.BEGIN LUMPED
   number_of_lumped:  3
   R  20.0  TOPO  BURRO
   L  20.0  PERRO GATO
   C  20.0  MAKO  TIZA
.END LUMPED
```

Fig. 151: Section on lumped elements: including one resistor of 20Ω connected between nodes $TOPO$ and $BURRO$; an inductor of $20\,mH$, and a capacitor of $20\mu F$.

of the switch; a flag "initially_closed_yes/no:" that indicates the initial status of the switch; the number of open operations prescheduled for this switch, "number_of_openings:"; and the number of close operations, "number_of_closings:"; then, after the label "open:" and separated by spaces, a list of the times for each opening operation; same as for open operations, for close operations there is the label "close:", see Fig. 152.

Intrablock links include a few additional fields: "ohms:", the resistance of the link, in ohms; "volts_are_DC_yes/no:", a flag that indicates if the voltage source associated with this link is DC or sinusoidal AC; "volts:", amplitude of

```
.BEGIN SWITCHES
   number_of_switches: 1
   .BEGIN SWITCH-0
      initial_node: TOTUMA
      final_node:   COBIJA
      initially_closed_yes/no: no
      number_of_openings: 2
      number_of_closings: 3
      open: 700e-6 1200e-6
      close: 300e-6 1000e-6 5000e-6
   .END SWITCH-0
.END SWITCHES
```

Fig. 152: This switch data section includes a single switch: the one between nodes $TOTUMA$ and $COBIJA$, a switch open at the beginning of the simulation, with two open operations, one at seven hundres microseconds, the other at twelve hundres microseconds.

voltage wave (or value of DC one); "hertz:", frequency of source, if it is an AC one, zero otherwise; "radians:", phase shift in radians of the AC voltage sine wave; and "frozen_closed_yes/no:", a flag that identifies this node as a mock up switch, one that will never open[7].

13.2.4 Transmission Lines

For transmission lines, the constant parameter model is included. The *LINES* section starts with the number of transmission lines in the network, "number_of_lines:", followed by a line section for each line included.

The line section presents five fields, labelled: "number_of_phases:"; "Zc:", followed by a blank space separated list of values for the characteristic impedance, in ohms, of each of the transmission modes[8] of the line; "delay:", followed by a list of the delays, in seconds, for each of the transmission modes; "nodes:", a list of strings for the nodes listed phase by phase, and send end to receiving end (following each node is a yes/no flag that requests for the current in that particular

[7] As is the case when a link is introduced to separate two sections of a block where there is no switch.

[8] In the order mode zero, mode one, etc..

phase/node to be computed and output), see Fig. 153; "q-matrix:", the modal transformation matrix[9] corresponding to the line geometrical configuration.

```
.BEGIN LINES
   number_of_lines: 1
   .BEGIN LINE-0
      number_of_phases: 3
      Zc: 637.9 278.7 328.1
      delay: 0.5e-3 0.35e-3 0.35e-3
      nodes: N1 no N4 yes N2 no N5 no N3 no N6 no
      q-matrix:
         0.592428855 -0.41233620 -0.70710678
         0.545945520  0.81237774  0.00000000
         0.592428855 -0.41233620 -0.70710678
   .END LINE-0
.END LINES
```

Fig. 153: In this case, only one three phase transmission line has been included in the network.

13.2.5 Grounded Voltage Sources

Voltage sources in the simulator are included by one of two devices: as a permanently closed link, § 13.2.3, with the corresponding voltage source; or, in cases where the source is grounded and we do not care about its current, as a "grounded voltage source" that is included inside a subblock, for an improved performance of the simulator.

A grounded voltage source includes: the single non ground node the source is connected to, "node:"; a flag to indicate if the source is DC or AC sinusoidal, "DC_source_yes/no:"; and the same fields already described in § 13.2.3, "volts:", "herts:", and "radians:".

13.2.6 High Voltage DC rectifier/inverter, HVDC

HVDC modules data sections, *HVDCs*, include:

- "controlled_by:", identifies the controller that triggers this module's valves.

[9] Generated by MTLine, or a similar utility program.

- "transformer_Y/D:", indicates whether the three-phase transformer in the module has a $Yy0$ or a $Dy11$ connection;
- "dc_line_reactor?_y/n:", a flag that signals if this module includes a smoothing reactor;
- "dc_reactor_value_in_mH:", value of the smoothing reactor;
- "nodes:", to which five nodes this module is connected to, five strings (of no more than six characters each) separated by blank spaces;
- "starting_Mode:", a list of six strings (each either *ON*, or *OFF*), corresponding to the initial status of the six valves in the module;
- "MVA:", capacity of the three-phase transformer in the module;
- "KV1:", line voltage rating of the transformer primary, in kilovolts;
- "KV2:", line voltage rating of the secondary;
- "Zsc%:", short circuit impedance[10] in percentage;
- "holding_current:", minimum value of the current through a valve at which a closed valve so remains;
- "threshold_voltage:", minimum value of a valve voltage at which it starts to conduct;
- "Tq:", recovery time for the valves, in seconds, the time the valve needs to vacate its depletion zone around the junction;
- "number_of_failures:", number of preprogrammed simulated valve failures, for testing of controlling schemes. For each *failure*, include a line that specifies the instant of the failure, which valve will fail[11], the number of

[10] Or reactance rather.
[11] Valves are identified by an integer from one to six.

integration steps that the failure will be sustained, and the type of failure (misfire, follow through).

```
.BEGIN CONTROL
   number_of_controllers: 1
   .BEGIN CONTROL-1
      Kp: 0.0001
      Ki: 0.000005
      reference_current: 1800
      firingAngle: 15
      alfa_min: 5
      alfa_max: 170
      pulse_width: 90
      sensedHVDC: 1
      numberOfControlledHVDC: 2
      controlled_HVDC: 1 2

      print_output_controller_yes/no: no
      print_ramp_yes/no: no
      print_alfa_yes/no: no
      print_wye_gates_yes/no: no
      print_delta_gates_yes/no: no
      print_DC_current_yes/no: no
      print_pre_reactor_voltage_yes/no: no
   .END CONTROL-1
.END CONTROL
```

Fig. 154: HVDC controller data.

13.2.7 HVDC Controllers

The original purpose of the HVDC model was to provide for a scenario to test the corresponding controllers. However, to test and verify the model itself, it was necessary to develop and model a simplified controller, a proportional/integrative controller targeted on a given reference value for the DC current, and modifying the firing angle of the module valves.

The data describing the controller includes the fields: "Kp:", proportional constant; "Ki:", integrative constant; "reference_current:", desired value for the DC current; "firingAngle:", initial firing angle for the gate signals, in de-

grees; "alfa_min:", minimum possible setting for the firing angle α, in degrees; "alfa_max:", maximum possible setting for the firing angle, α, in degrees; "pulseWidth:", width of gate firing pulses, in degrees; "sensedHVDC:", what HVDC module's DC current is being monitored and controlled; "numberOfControlledHVDC:", how many HVDC modules are triggered by this controller; "controlled_HVDC:", a list of the numbers that identify all the HVDC modules fired by this controller.

For in depth studies of the process of control, a few additional output signals can be requested as illustrated in Fig. 154.

```
enum nodType_t{LUMPED, SOURCE, LINE};

struct nod_t{
   nodType_t nodType;
   char      name[7];
   int iBlk;     // What block contains it (1..cNumBlk).
   int iSub;     // What subblock inside that block.
   bool pendingScndPass; // TRUE=pending for a second pass.
};
```

Fig. 155: Every node is represented by a 'nod_t' structure and registered in a cell of the list 'nodList_t'.

13.3 Classes in the Preprocessor

As was mentioned in § 13.1, the preprocessing stage of the simulator, OVPP, was built around an ancestor class, *list_t*, a double link circular list class. Four other application specific classes inherit their basic methods and data elements from the *list_t* class. Those are: *nodList_t*, a list-class derived as public from *list_t* with all the nodes[12] in the network; *sub_t*, a class —a public descendant of *list_t*— that describes the activities associated with a subblock in the network[13]; *subList_t*, a list of all subblocks in the network; *blk_t*, to describe blocks in the

[12] External nodes, as defined in § 5.13.

[13] A subblock in the sense seen in § 5.10 on page 67.

network; and *blkList_t*, a list of all the blocks in the network, a description of the segmented network that, once set up, is ready for output.

The best way to describe the functionality of each of the object classes introduced above is to present its data and method elements.

13.3.1 The *list_t* Class

The original ancestor of the hierarchical family of classes in OVPP is *list_t*, a double-link circular list [69]. It is build around a basic unit of data, the list-cell, a structure with the type definition shown in Fig. 156.

```
typedef void *ptr_t;    // A generic pointer.

struct cell_t{
   cell_t *pPrevCell;  // Ptr. to previous cell.
   cell_t *pNextCell;  // Ptr. to next cell.
   ptr_t   pContents;  // Ptr. to data hooked to cell.
};

typedef cell_t *pCell_t; // Ptr. to a cell.
```

Fig. 156: A cell in the *list_t* class.

The *list_t* class contains a hook to hang the actual list of cells, the head of the list, a pointer to the "head" cell, a neutral-no-data cell that serves as a binding between the first cell and the last cell, as seen in Fig. 157.

The public methods available to the client of the *list_t* class are listed, and described very briefly, in Fig. 158.

13.3.2 The *nodList_t* class

The *nodeList* is a linked list that inherits as public from the ancestor *list_t*, § 13.3.1. Its cells contain each one of the external nodes in the network. Its interface is presented in Fig. 159. Each element within this list is represented by a structure of the type in Fig. 160.

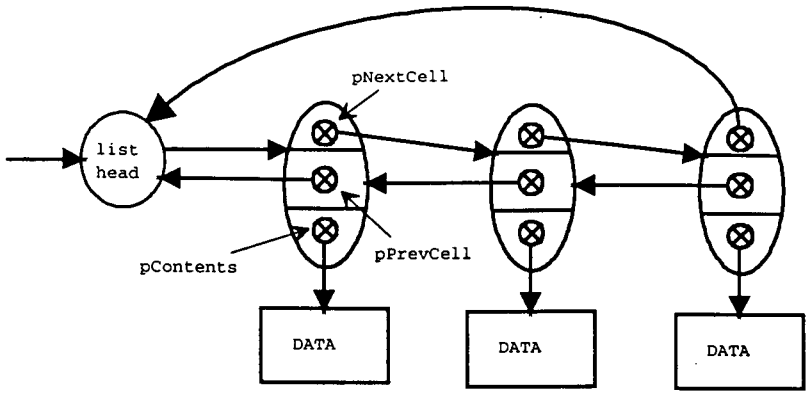


Fig. 157: The "head" cell and the circular linked list defined by *list_t*.

13.3.3 The *sub_t* class

Each subblock in the problem is represented in the preprocessor by *sub_t*, a linked list of nodes (each node is identified by the integer code generated during the registration that specifies the position of the node within the network list *nodList_t*, see § 13.3.2.) In Fig. 161, the methods and data items of the subblock class, *sub_t*.

13.3.4 The subblock list, *subList_t*, class

All the subblocks in the network are included in the list *subList*, an instantiation of the class *subList_t*, public descendant of the *list_t* class, Fig. 162. During the process of subdivision of the network nodes (in *nodList*) into subblocks, according to the MATE criteria, the nodes are registered (by the *nodList* itself) with the subblock list, *subList*), that passes the node to its corresponding subblock object, see § 13.3.3.

13.3.5 The *blk_t* class

Each block in the problem is represented in the preprocessor by *blk_t*, a linked list of subblocks (*sub_t* instantiations) (identified by the integer code generated during the registration that specifies the position of the subblock within the

```
class list_t(
protected:
   pCell_t  pHead;    // Head of the linked list (See Fig. 11.9).
   int cNumCell;      // Current number of cells in the list.
public:
   list_t();          // Constructor.
  ~list_t();          // Destructor.
   pCell_t NewCell(ptr_t pData, pCell_t pPrevCell; pCell_t pNextCell);
      // 'NewCell' creates and initializes a new cell. it does not link it.
   inline bool IsListEmpty(); // TRUE = the list contains no cells.
   inline int GetNumCells(); // Returns the number of cells.
   inline pCell_t GetHead(); // Returns ptr. to the head cell.
   inline pCell_t GetFirstCell(); // Rtn. ptr. to first cell.
   inline pCell_t GetLastCell(); // Rtn. ptr. to last cell.
   pCell_t GetNextCell(pCell_t pCell); // Rtn. ptr. to next cell to given one.
   pCell_t GetPrevCell(pCell_t pCell); // Rtn. ptr. to previous cell.
   void InsCell(pCell_t pCutOffCell; ptr_t pData); // Insert in front of a cell.
   void DelCell(pCell_t pCell); // Delete a cell from the list.
   pCell_t GetCellContains(ptr_t pData); // Rtn. ptr. to cell with this data.
   int GetCellPos(pCell_t pCell); // Rtn. position of this cell in the list.
   pCell_t GetCellAtPos(int pos); // Rtn. ptr. to cell at position 'pos'.
   void InsInFront(ptr_t pData); // Insert in front of the list.
   void InsAtEnd(ptr_t pData); // Insert this data at the end of the list.
};
```

Fig. 158: Methods and data items in the *list_t* class.

network subblock list *subList_t*, see § 13.3.4. The methods and data items in the block, *blk_t*, class are parallel to those in the subblock class, *sub_t* in Fig. 161, with the differences corresponding to the contents of the block class (subblocks instead of nodes).

13.3.6 The block list, *blkList_t*, class

All the blocks in the network are included in the list *blkList*, an instantiation of the class *blkList_t*, public descendant of the *list_t* class. This class mimics the functionality of the subblock list (*subList*), with the difference that the first one contains blocks, into which subblocks are registered, while the latter contains subblocks into which nodes are registered.

During the process of subdivision of the network subblocks (in *subList*) into blocks, according to the connectivity provided by the links and the decoupling introduced by the transmission lines, the subblocks are registered (by the *subList* itself) with the block list, *blkList*), that passes the subblock to its corresponding block object.

```
class nodList_t::public list_t{
public:
   nodList_t; // Constructor.
   int PutNod(char *sName, nodType_t eNodType);
   int GetNumNodes();
   pNod_t GetNodPtr(int iPos);
   pNod_t GetNodPtr(char *sName);
   int GetNodPos(char *sName);
   char *GetNodName(int iPos);
   nodType_t GetNodType(int iPos);
   nodType_t GetNodType(char *sName);
   void SetNodPending(int iPos);
   bool PendNodLeft();
   int GetFirstNodInSub(int *iSub);
   int GetNextNodInSub();
};
```

Fig. 159: Methods and data items in the *nodList_t* class.

The methods and data items of this *blkList_t* class are parallel to those in *subList_t*, with the differences due to the nature of the contents of each list.

At the beginning of the preprocessing, once all the elements have been loaded from the input data file, each one of them is given the opportunity to register their external nodes into the network node list *nodeList*. The elements register their nodes by requesting the service "PutNode" from the *nodeList* object. That service returns to the element the numeric code that identifies the registered node in the network. The registration service provided by "PutNode" also requires from the element a categorization of the node as belonging to a *LUMPED* element, or to a *LINE*, or perhaps to a grounded voltage *SOURCE*.

13.4 Main Tasks of the Preprocessor

The goal of the preprocessor is the creation of the input file to the engine, to OVNI. In that file, the network has already been broken into blocks, and these into subblocks. That file identifies each element (or part of it), and every node

```
enum nodType_t{LUMPED, SOURCE, LINE};

struct nod_t{
   nodType_t eType;
   char sName;
   int iBlk;
   int iSub;
   bool bPend;
};
```

Fig. 160: Each node in the network list is an instance of this structure.

as belonging to this or that subblock, which is part of the corresponding block.

Besides all that, the conductance matrices for each subblock corresponding to every possible open/close switch combination within the subblock (or an indication of the "chameleonic" non linear element associated to it) need to be calculated (precalculated, if you will, since this is the preprocessing stage.) Such goal is broken in the following sections into "tasks". The tasks are described as message exchange between the objects in the application, under the prompt of the director (the hub of the preprocessor, a task scheduler.)

13.4.1 Creation of a list of all the nodes

The first task the preprocessor tackles is the creation of a list of all the nodes in the network, *nodList*. The director (hub) prompts every element in the system, *apElm*[], to register its nodes with the list *nodList* through the method "nodList.PutNod". In this task, the elements are clients, and the list of nodes is the server. The service provided, PutNod, takes a node as described below, and returns to the client a numeric global-network identification code for the node[14].

Nodes passed for registration through the PutNod method, are identified by

[14] Nodes carry two identification codes, the global network code, and another that identifies it within the subblock that contains it.

```
struct chm_t{                          // A chameleon is seen inside a subblock.
   int   cNumNodes;                    // Number of nodes in the chameleon.
   int  *aiPosNodInSub;                // Array of position of the nodes with
 };

typedef chm_t *pChm_t;

class sub_t:public list_t{             // Represents a subblock.
   int iBlk;                           // What block contains this subblock.
   bool bPend;                         // Subblock pending for inclusion in block.

   pCell_t pFirstSrcNode;              // Ptr. to cell with first src. node.
   double **aMatrix;                   // [Gaa Gab] Rectangular matrix.
   double **aGab;                      // [Gab]. (just ptrs into 'matrix').
   list_t lChm;                        // List with all chameleons in block.

   void CreateMatrices();
   pChm_t GetPtrChm(int iChm);
public:
   sub_t();
   void PutNod(int iNodPos, nodType_t eType);
   int NodPosInSub(int iNodPos);
   int ListPosOfNodInSub(int iNodPosInSub);
   void AddMatrixElm(int row, int col, double value);

   int NumNodes();
   int NumSrcNodes();
   int NumNonSrcNodes();

   int AddChm(int cNumNod, int *aiNodPosInSub); // Add a chameleon with n nodes in p[].
   int NumChm();                       // Returns the number of chameleons.
   int NumNodInChm(int iChm);          // Number of nodes in chameleon.
   int GetPosInSubOfChmNod(int iChm, int iNodPosInChm);
                                       // Chameleons are numbered 1..,
                                       // Subblock nodes also 1.., but nodes
                                       // inside a chameleon are 0...

   double GetGaaElm(int row, int col); // Returns an elm of [Gaa].
   double GetGabElm(int row, int col); // Returns an elm of [Gab].
};
```

Fig. 161: Methods and data items in the *sub_t* class.

their string-name, and characterized as either: *lumped, source, or line* nodes, depending on whether they are terminal to a grounded voltage source, or to a line, or to something else. A *line* identifier takes precedence over any of the other two. A *source* identifier takes precedence over the *lumped* identifier only. Ideal voltage sources are not tolerated connected to a line node[15]. See Fig. 163.

Each cell of the *nodList* double-linked list, contains a node represented by the structure in Fig. 155.

[15] Use a link-source in such a case.

```
struct chm_t{                          // A chameleon is seen inside a subblock.
   int   cNumNodes;                    // Number of nodes in the chameleon.
   int  *aiPosNodInSub;                // Array of position of the nodes with
 };

typedef chm_t *pChm_t;

class sub_t:public list_t{             // Represents a subblock.
   int iBlk;                           // What block contains this subblock.
   bool bPend;                         // Subblock pending for inclusion in block.

   pCell_t pFirstSrcNode;              // Ptr. to cell with first src. node.
   double **aMatrix;                   // [Gaa Gab] Rectangular matrix.
   double **aGab;                      // [Gab]. (just ptrs into 'matrix').
   list_t lChm;                        // List with all chameleons in block.

   void CreateMatrices();
   pChm_t GetPtrChm(int iChm);
public:
   sub_t();
   void PutNod(int iNodPos, nodType_t eType);
   int NodPosInSub(int iNodPos);
   int ListPosOfNodInSub(int iNodPosInSub);
   void AddMatrixElm(int row, int col, double value);

   int NumNodes();
   int NumSrcNodes();
   int NumNonSrcNodes();

   int AddChm(int cNumNod, int *aiNodPosInSub); // Add a chameleon with n nodes in p[].
   int NumChm();                       // Returns the number of chameleons.
   int NumNodInChm(int iChm);          // Number of nodes in chameleon.
   int GetPosInSubOfChmNod(int iChm, int iNodPosInChm);
                                       // Chameleons are numbered 1...
                                       // Subblock nodes also 1.., but nodes
                                       // inside a chameleon are 0...

   double GetGaaElm(int row, int col); // Returns an elm of [Gaa].
   double GetGabElm(int row, int col); // Returns an elm of [Gab].
};
```

Fig. 162: Methods and data items in the *subList_t* class.

13.4.2 Grouping Subblocks

The list of nodes knows whether any of its nodes has been associated to a subblock, or block. The director (hub) requests of the list *nodList* to provide a non grouped node, along with the identifier of a subblock that has not been created yet (information the *nodList* has, since its nodes keep the code of the subblocks and blocks they belong, if they do, or an invalid code if they don't. See Fig. 164.)

In this case the client is the hub, and the server method is "nodList.GetFirstNodInSub". The two data items received by the hub are the active node, iNodActive, and the subblock under creation, iSubInCreation.

The active node is presented by the hub to each element. The element checks

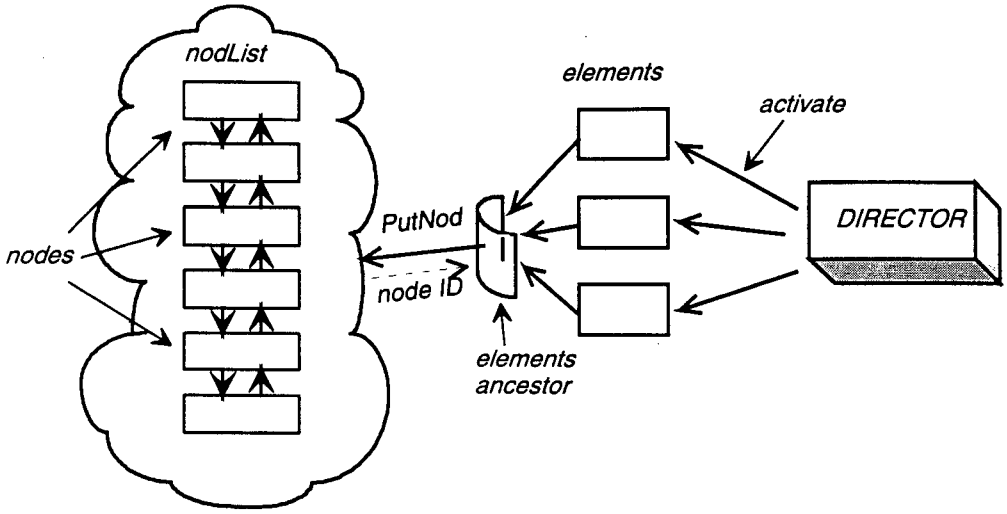


Fig. 163: Interaction of classes in OVPP during node registration.

if it is connected to the node. If it is, the element adopts the subblock under creation as its own, and so it records it. In this case, also, the element notifies that all the other element's nodes[16] are to be marked as pending for more possible elements connected to them and part of the same subblock. In short, as pending to be proposed as active nodes during the next step. The notification to the list of nodes is made by the element through the method "nodList.SetNodPending."

Now the director keeps asking of the *nodList* for another node in the current subblock (which it produces as one of the nodes already marked as *pending*), though the method "nodList.GetNextNodInSub." This node becomes the active node and is subject to the same process as the first one was. This goes on until there is no pending node left in the *nodList*, which means that the subblock is complete.

Every time the nodes list, *nodList*, is asked by another node in the block being

[16] With the very important exception of transmission line nodes, which are grouped as *left* group, and *right* group. In this case each group is treated as a separate lumped element.

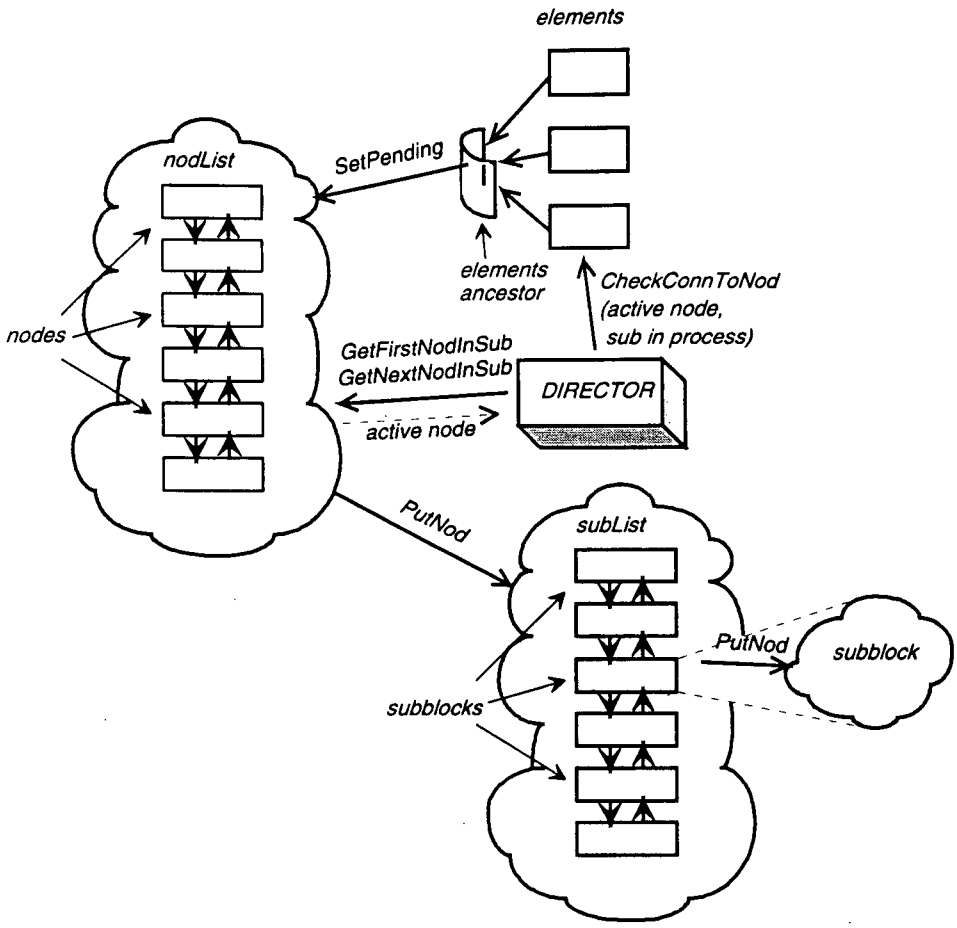


Fig. 164: Interaction of classes in OVPP during assembling of subblocks.

assembled, *nodList* registers that node with the list of subblocks, *subList*, that in turn sends the node to one of its component object subblocks, an instantiation of class *sub_t*.

At this point, the hub asks the *nodList* for another first node in a non created subblock, and the process continues, until no unaligned node is left.

13.4.3 Calculate Subblock Matrices

Now that the subblocks have been partitioned, and properly organized within the *subList* object, it is time to determine the conductance matrix of each of

those subblocks. That goal is reached through three subtasks, as follows. See Fig. 165.

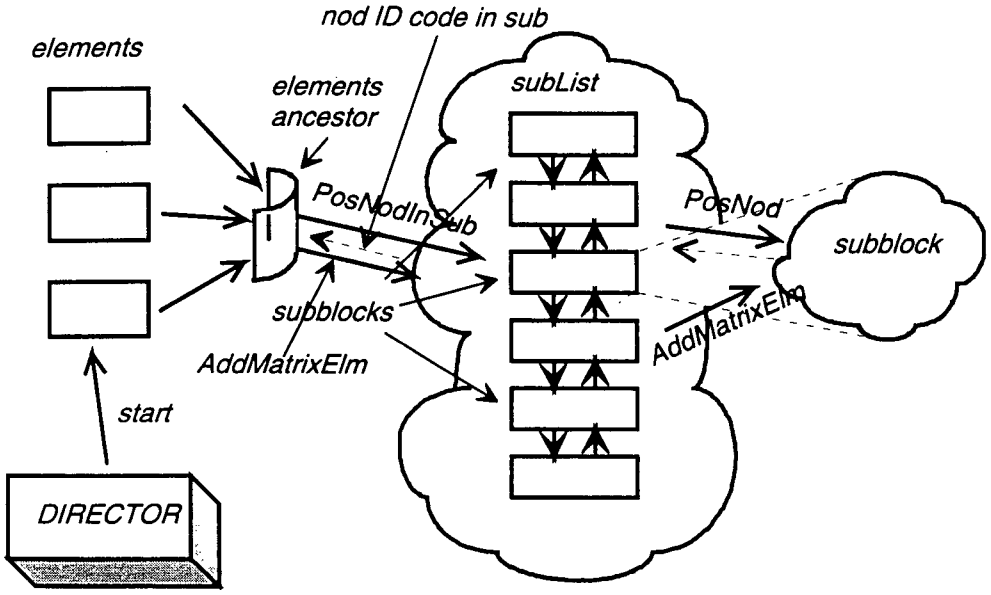


Fig. 165: Interaction of classes in OVPP during subblock matrix calculation.

13.4.3.1 Refer element nodes to subblocks

In this task, the director prompts each element in the network to obtain the subblock relative identification code for each one of its external nodes. This will allow, further down the road, for the elements to contribute their conductance matrix into the subblock's at the correct positions.

Each element, a client, request from the subblock list, *subList*, the server, for each node identified by its global code, and its subblock (which the element obtained in the previous task, see § 13.4.2), what is the subblock id code for that node. The service invoked is "subList.NodPosInSub."[17]

[17] The reason for the name of the method is that the id code for the node is but the relative position of that node inside the subblock.

13.4.3.2 Add Element Contributions to Subblocks

Now each element is directed by the hub to pour its conductance matrix contribution into the corresponding subblock's, by the service provided by the list of subblocks, "subList.AddMatrixElm."

13.4.3.3 Associate Chameleons to Subblocks

The completion of the subblocks matrices is delayed for subblocks that include non linear elements, called *chameleons* in OVNI and in OVPP. In this task, the hub prompts each one of the elements that if it is itself a chameleon, to come forward and request inclusion into its corresponding subblock from the *subList*, via the service "subList.AddChm."

The *subList* object passes the chameleon's inclusion request down to the corresponding block object, again through the service provided by the method "blk.AddChm."

13.4.4 Grouping Blocks

Now that the subblocks have been dealt with, the director starts the grouping of subblocks into blocks, as established by the links provided in the data file. To do this, the hub goes through the same steps followed to form the subblocks, but using *subList* instead of *nodList*, since before we were grouping nodes, and now we are grouping subblocks. The connectivity of subblocks is governed by links, in the very same way connectivity of nodes is defined by elements. Then it follows that the director will use links as elements were used before, see Fig. 166.

Putting it all together, first the director lets each link in the network to request from the *nodList* to which subblocks it is connected. Then the *subList* is asked by the director to provide the first non aligned subblock (i.e., a subblock not included into any block yet), and the corresponding new block id code. The service is "subList.GetFirstSubInBlk." That subblock becomes the *active* subblock, which is presented to each link in the network. The link, in turn, checks if it is connected to the active subblock, if it is, record the block being

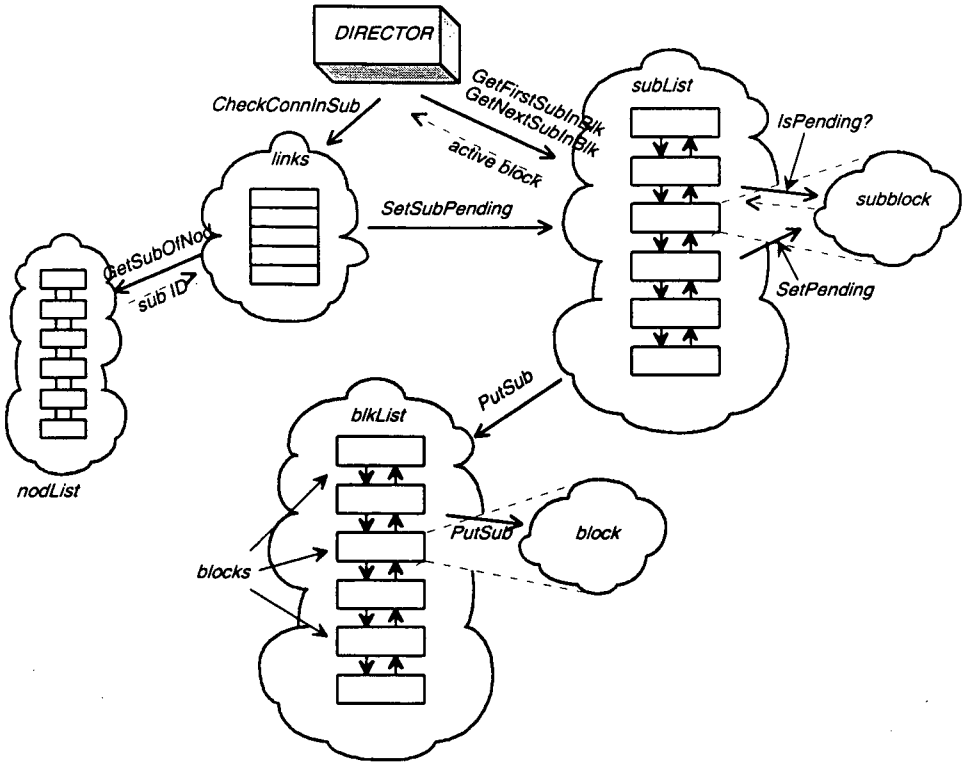


Fig. 166: Interaction of classes in OVPP during block grouping.

assembled as its own and requests from *subList* to mark the other subblock the link is connected to as "pending." At the end of this stage, all subblocks linked to the active one have been marked as pending and will be part of the same block being grouped.

Now the director asks of the *subList* for another subblock within the same block being assembled, the service is "subList.GetNextSubInBlk." This subblock becomes the active one and goes through the same process as the first one. The hub keeps asking for more subblocks in the current block, until none is left, the blocks have been assembled.

At each time the *subList* provided a subblock in the current block, *subList* registers the subblock with the list of blocks *blkList*, that passes the registration

request down to the corresponding block object, and instantiation of class *blk_t*.

At the end of this task the list of blocks has been filled, and the blocks know which subblocks belong in them. The program is ready for output.

Part VI

VALIDATION

14. VALIDATION TESTS

14.1 Introduction

Validation of an algorithm for real time simulation of any kind of system involves two aspects: validation of its accuracy and validation of its speed. Those are the two main aspects considered in this chapter.

However, OVNI's accuracy and speed are built on: a) a judicious choice of backward Euler's integration rule, supported and improved by b) the double step double interpolation backtrack-advance procedure; c) precalculation and triggering of new states made possible by d) topological segmentation, e) MATE segmentation, and f) the *node hiding* element model streamlining strategy; where MATE segmentation allowed for the efficient choice of Cholesky's algorithm to obtain links currents.

This chapter's organization follows, as close as possible, the order of the topics listed in the previous paragraph. In the next section, Sec. 14.2, a quick review of the tests on integration rule issues reported in chapter 3. Following it, Sec. 14.3, test cases that illustrate the drastic effect of DSDI on two switching circuits. Then, in section 14.4, the aspect of 'speed' mentioned on the first paragraph of this section. That speed exploration includes subsections for timings corresponding to the two target cases outlined in chapter 1, the relay test case (Sec. 14.4.1), and the HVDC controller case (Sec. 14.4.2). That speed section also includes a subsection, 14.4.3, that reviews the performance advantages of MATE segmentation and the associated precalculation. Subsection 14.4.4 explores Cholesky's perormance by itself. To end this chapter, in section 14.5, a suit of tests that explores the accuracy of ONVI's simulation on different situations associated with the main two test cases, is included

Part[1] of the tests reported in this chapter were run on a Pentium II 200 MHz workstation with 32 Mbytes of RAM, 2 Gbytes hard drive at the Real Time Simulation

[1] The reason for the duality of the hardware platform is historic. As the project evolved, the tests to assert real-time performance were applied to the then current version of the simulator on the available workstation, and the results published. To keep match between the published results (albeit slower than the ones obtainable with the newer and faster hardware platform), the same 200 MHz machine results were kept in those sections of this thesis report.

Laboratory at UBC, CICSR 043, as specified in the section describing the particular test; some of the tests were run on an AMD-K6(2) 400 MHz workstation with 256 Mbytes of RAM, and 15 Gbytes hard drive, as indicated in the corresponding section. The most critical performance tests were confirmed by measurements done separately at Mitsubishi Corporation, Tokyo, Japan; and at Electricite de France, Direction des Etudes et Recherche at Clamart, Paris, France.

14.2 Integration Issues

OVNI's integration process profits from the stability and accuracy of the backward Euler's integration rule, as established in chapter 3. The associated tests were included in that chapter. In particular the experiments that advanced backward Euler's as a rule with no phase shift associated distortion, Figs. 14 and 16, and the tests on a simplified single-phase power network that explored the possibility suggested by Fig. 15, that with backward Euler's rule an integration step almost 50% larger can be used for the same 3% magnitude distortion (which provides a speed advantage to OVNI), Figs. 22 and 24.

In cases where switching operations occurs off synchronism with the sampling process, the anomalous introduction of inverse currents through opening switches, and the consequent opening of non-zero currents, was solved by the introduction of the double step double interpolation procedure, in section 7.6.1. The tests that explore that procedure are included in the next three pages, under section 14.3.

14.3 Asynchronous Commutation

To explore the validity of the DSDI resynchronization shift to accommodate asynchronous commutation, two cases already presented in [35] are included. First, a relatively simple two-diode full wave rectifier circuit, Fig. 167. And next, a six-valve three-phase rectifier group, Fig. 170.

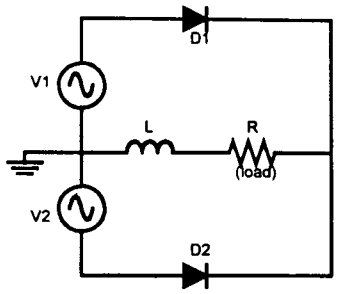


Fig. 167: A two-diode full wave rectifier case.

Both cases were run first on the EMTP algorithm to illustrate the occurrence of large spurious current spikes in the diodes as a result of asynchronous commutation of the diodes.

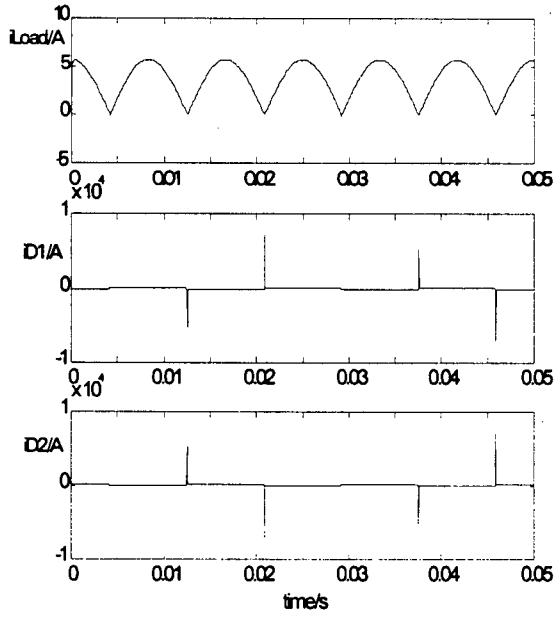


Fig. 168: For the two-diode rectifier, current in the load, current in diode one, current in diode two. Observe the current spikes of almost ten thousand amperes, when the load current peaks at less than five amperes.

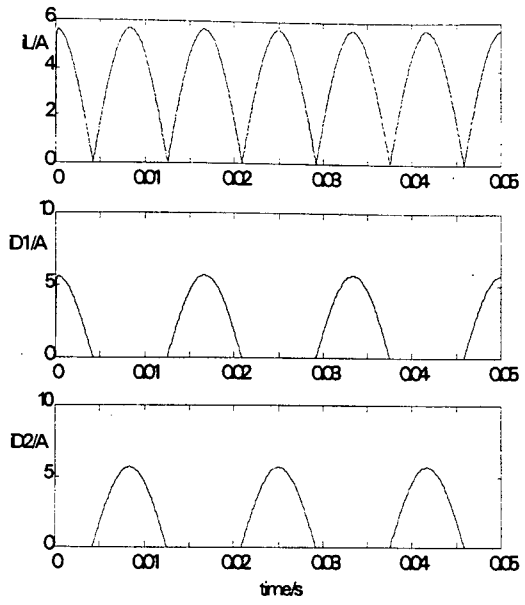


Fig. 169: DSDI output for two-diode rectifier case. a) Current in the load; b) Current in diode one; c) Current in diode two.

For the two-diode rectifier, with voltage sources of 200 Vrms, 60 Hz, a load of 50 ohms, and a reactor of 1 mH [35], the EMTP algorithm's results for: load current, current in first diode, and current in second diode, are illustrated in Fig. 168. The same circuit, once the DSDI resynchronization shift has been included in the solution, produced, for the same currents just mentioned, the spike-free results presented in Fig. 169.

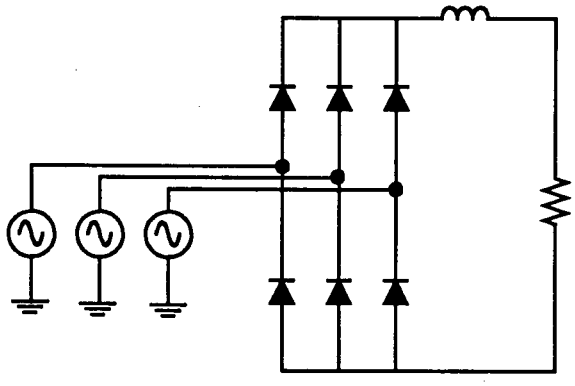


Fig. 170: A six-valve three-phase rectifier group.

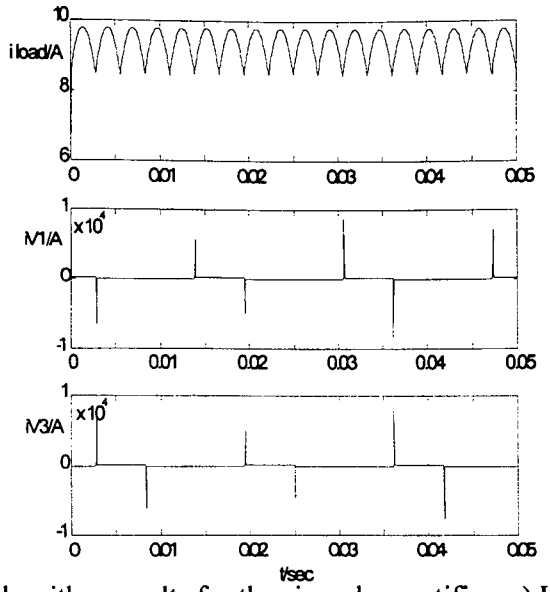


Fig. 171: EMTP algorithm results for the six-valve rectifier: a) Load current; b) Valve one current; c) Valve three current.

For the six-diode case in Fig. 170, the asynchronous commutation of diodes one and two produced, in the EMTP algorithm, the spikes illustrated in Fig. 171. The solution with DSDI of the same six-valve case generated the spike free results shown in Fig. 172.

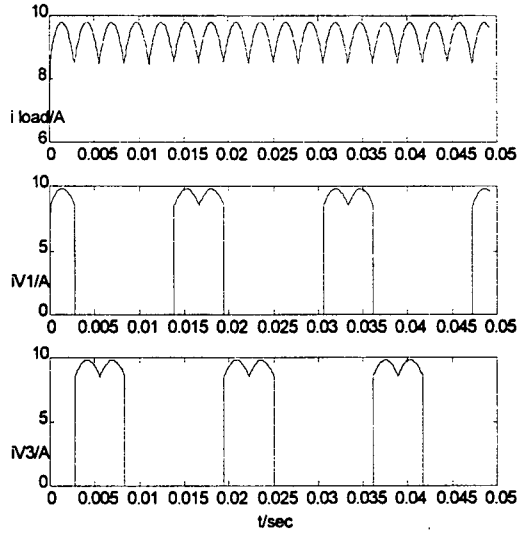


Fig. 172: DSDI results for the six-valve three-phase case. a) Load current; b) Valve one current; c) Valve three current.

14.4 Speed

The performance of the final product, the software that embodies the algorithm developed during this project, is to be measured, not as overall simulation speed (i.e., how many seconds of *computing time* it takes to take so many seconds of *real time*.) but as the speed associated with the integration step that takes the longest to compute, the slowest integration step, the critical step. The time consumed by that critical step can be related by the complexity of the computations involved, or perhaps to the refreshing of the associated cache.

To predict what will be the conditions that produce such a critical step in a simple case is possible. However, in a more realistic case, observation and measurement of the execution time of each time step of the simulation is desirable. Such measurements were possible thanks to the fine grain time routines created by Mr. Jesus Calvino-Fraga. The timings obtained were also subjected to validation (in the most critical cases reported in this thesis) by the research centre of L'Electricite de France, Direction des Etudes et Recherches, at Clamart, Paris, France, under the direct supervision of the author. Independently, the same set of timings were corroborated by Mitsubishi Corporation of Japan. Both agencies accepted the accuracy validation against the EMTP as an acceptable method.

To validate performance, the two target systems introduced at the begining of this thesis are used, the case for protective relay testing, and the case for HVDC controllers testing.

14.4.1 Relay Testing

The relay testing case in Fig. 200 was the workhorse on top of which every single test of OVNI's solution was tried at one point or another. This configuration was proposed by the industry, and includes two segments of two three-phase transmission lines running along the same right-of-way, coupled to each other (a six-phase coupled group), one segment before a fault site, and another after it. It also includes series capacitive compensation and metal oxide varistor protection. On the right healthy side of the system, a three-phase transmission link brings in an equivalent three-phase Thevenin for the network "on the right." The same applies for the network "on the left."

This configuration has been running for weeks on the single workstation version of the hardware solution on top of which OVNI runs, this single workstation hardware implementation, thanks to Mr. Jesus Calvino-Fraga [38]. In this implementation, the switches that simulate single, dual, and three-phase faults at the bus of contingency, are wired to the simulator as three physical switches. It runs, on that single 400MHz, Pentium Pro workstation at 35 microseconds per step, well within the targeted bandwidth to accommodate for the necessary data exchange overhead.

The same case, in an illustration of the flexibility of OVNI's solution algorithm and code, has been running also on the parallel processing five-workstation version of OVNI, a hardware implementation possible thanks to Mr. Jorge Hollman [71]. There, Mr. Hollman prepared the 234-node test case in Fig. 179, and run it on the five 400MHz Pentium PC cluster. The solution times generated by OVNI on this parallel processing platform were 46 microseconds per integration step for the case just mentioned (this case would run on a single machine at 164 microseconds per step.).

14.4.2 HVDC Systems

The extended HVDC substation model, including saturation and zero sequence modelling and nine AC filters rendered the timings shown in the table below. These timings were obtained on a Pentium Pro 200 MHz workstation (the inclusion of the filters and the saturation models amounted to about 1% of the model's history sources updating time.) [30]

Case	Description	# valves	Microtran (us)	DU-99(us)
1	Monopolar 6-pulse converter	6	459	26
2	Monopolar 12-pulse converter	12	983	46
3	Bipolar 6-pulse coverter	12	897	51
4	Bipolar 12-pulse converter	24	3,120	81

Table 14.1 Solution time, per integration step, in microseconds, of four HVDC cases. Comparison between the EMTP and OVNI's prototype solutions times on a Pentium Pro 200 MHz.

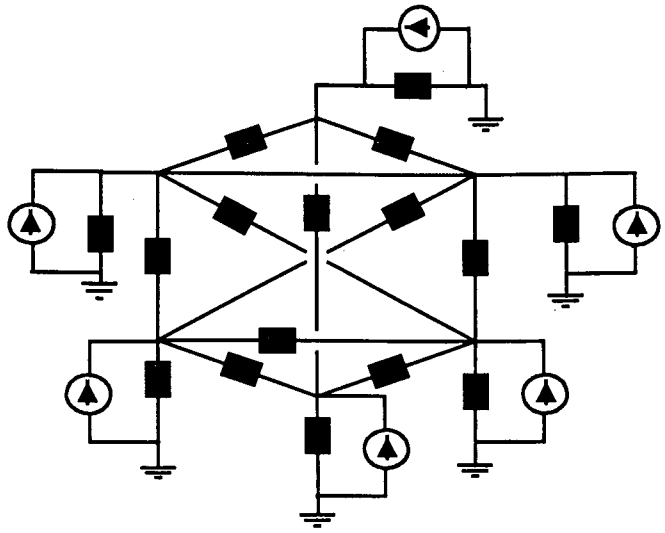


Fig. 173: One of the six sections in the test network used to benchmark MATE

14.4.3 MATE vs. Conventional Solution

To constrast the speed of MATE segmentation with the standard unsegmented one, the same network was run on the same hardware, with both methods. The network was built linking together network sections like the one in Fig. 173. Every node in the section is grounded through a resistor, and connected to every other node in the section through some other resistor. Also, there is a current source from ground to every node in the section. In Fig 173, one of such sections, with six nodes, is illustrated. Each section (subblock) is connected to two other sections through two links, see Fig. 174. Each link includes a voltage source in series with a resistor. The sections are so connected in a ring (i.e., the last section is linked to the first one.) Tests were run with networks consisting from two to six sections, where each section had from two to six nodes. The resulting timings are given in Tables 14.2, for MATE, and 14.3, for the standard unsegmented solution. The performance differences can be better appreciated in the 3D graphics included in the following pages: Fig. 175 (solution times for MATE segmented algorithm for different networks with sections from two up to six nodes); Fig. 176 (solution times for a standard unsegmented solution algorithm); and Fig. 177 (percentage of improvement from the standard to the MATE's algorithm for the 36 networks tested in this section).

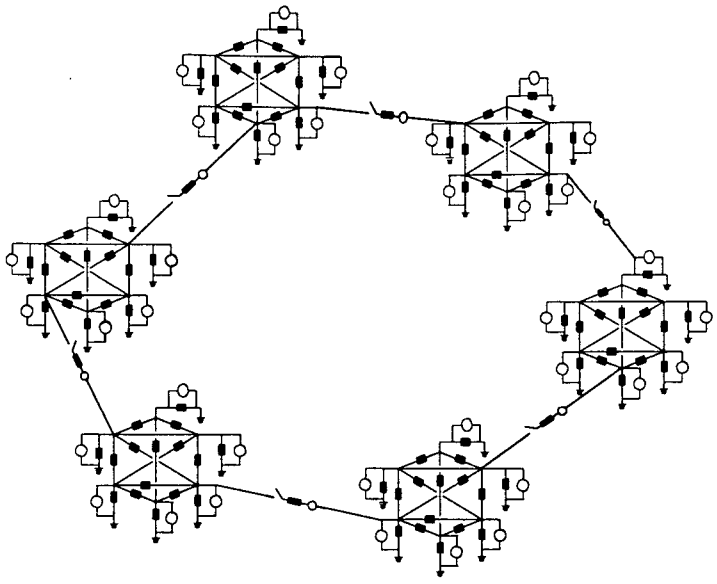


Fig. 174: Six node sections connected in a ring.

nodes in each subblock

Num. subblocks	2	3	4	5	6
2	2.9	3.5	4.2	4.3	5.4
3	5.1	5.3	6	6.9	7.8
4	6.7	6.8	6.9	8.4	9.5
5	8.2	9.1	10.2	11.3	12.2
6	11.5	11.7	12.7	13.1	13.3

Table 14.2. Solution time (in microseconds) for a single-block network with several subblocks, and of varying nodes per subblock, with the MATE segmentation algorithm.

Num. subblocks \ nodes in each subblock	2	3	4	5	6
2	15.7	24.6	38.6	55.5	75.7
3	31.7	56.3	88	136	191
4	57	106	172	260	375
5	91.7	173	287	445	649
6	142	262	446	697	1,033

Table 14.3. Solution times in microseconds with standard unsegmented algorithm, for networks formed by <row> number of sections (subblocks), where each section

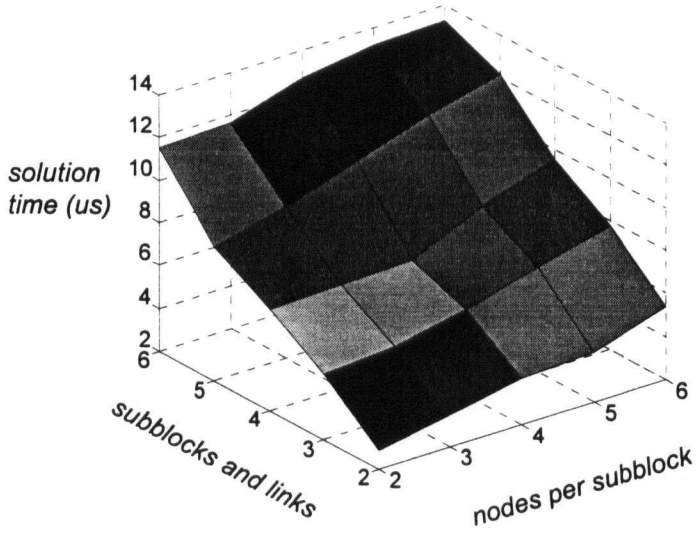


Fig. 175: Solution times, in microseconds, for MATE algorithm, corresponding to a network of so many (2..6) subblocks with a given amount (2..6) of nodes per block.

14.4.4 Cholesky vs. LU Decomposition

At every time step, and in each of the topological blocks, the simulator needs to solve a system of algebraic equations. The system whose solution produces the currents in the block's MATE links. In cases where the subblocks included in the block do not

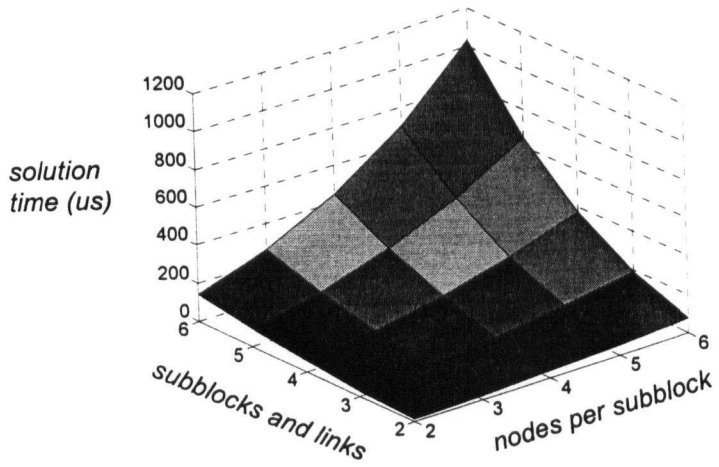


Fig. 176: Solution times for the unsegmented standard solution algorithm corresponding to a network with so many subblocks (1..6) with a number of nodes in each subblock (1..6).

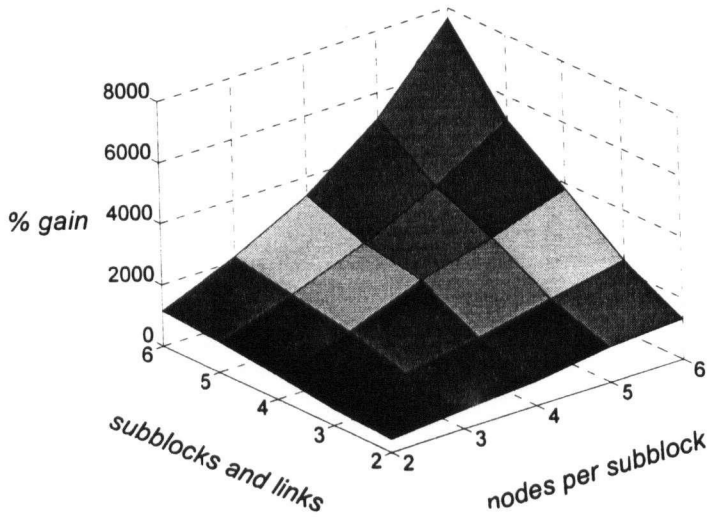


Fig. 177: In percentage, how much faster MATE is compared to the standard unsegmented algorithm for the kind of networks studied.

contain non-linear elements or switches, the links matrices can be precalculated, but in general that is not so[2].

The solution of the links system was first approached with a robust and efficient LU decomposition algorithm with partial pivoting. The results were satisfactory but

[2] In the current implementation, the links matrices are never precalculated. But it may be convenient to do so in a future revision.

size (n x n)	LU (usec)	Cholesky(us)
2 x 2	3.9	1.3
3 x 3	5.1	2.1
4 x 4	6.3	3.8
5 x 5	8.8	4.6
6 x 6	13	6.4
7 x 7	17.2	8
8 x 8	22	11
9 x 9	27.3	14.7
10 x 10	33.1	17.6
11 x 11	41.5	22.2
12 x 12	49	26.4

Table 14.4. Solution times of Cholesky method versus LU decomposition, in microseconds.

left room for improvement in cases where HVDC module controllers were included. To stretch the performance of the engine, an alternative solution method was revised: Cholesky's. Cholesky works only on system whose equation matrix is positive definite. That a matrix A is positive definite means, geometrically, that when the rotational transformation implied by the matrix A is applied to a vector V in the same hyperdimensional space where the matrix rotation is defined)a geometrical interpretation of space), the resulting vector turns out to be closer than ninety degrees from the original one. In the more succinct notation of vector analysis [73]:

$$v \cdot A \cdot v > 0$$

Where the dot represents matrix vector multiplication, and also vector dot product. Another interpretation to a matrix A being positive definite is that given in chapter 11 of [73], a matrix whose eigenvalues are all positive. But a positive eigenvalue implies a decaying natural response mode, which is always the case for the networks simulated here. Summarizing, Cholesky is safe to apply in the case of interest. The advantages in speed of the modified implementation with no pivoting used in this

work of Cholesky's versus LU decomposition is patent in the tests run below. Those tests were performed on systems with 2, 3, ... 12 links. In each case, Cholesky's algorithm beat LU decomposition by a factor close to two. This suit of tests was run on a AMD-K6 II 400MHz workstation, and compiled with Visual C++ version 5.0 with all optimizations switches on (release version).

14.4.5 Precalculation vs. Live Computation

In cases where all the subblocks in a block have fixed topology, i.e. they contain neither switches nor non-linear elements (no *chamaeleons*), the link matrices corresponding to every possible open-close link combination can be precalculated. In this case, the matrices inverses are prestored. The same 36 cases used to benchmark the previous two sections are used for this section too. The solution times obtained are included in Table 14.5. Graphically, the solution times can be seen in Fig. 178, and the improvement in percentage, with respect to non-precalculated MATE, in Fig. 179.

Num. subblocks \ nodes in each subblock	2	3	4	5	6
2	0.9*	1.7	2	2.9	3.8
3	0.95*	2.6	3.3	4.1	4.8
4	2.6	3.3	4.3	5.3	6.3
5	3.4	4.2	5.3	6.6	7.7
6	4.3	5.5	5.8	8	9.2

Table 14.5. Solution time (in microseconds) for a single-block network with several subblocks, and of varying nodes per subblock, using precalculation for the links matrices. (*) Under the granularity of the timer routines.

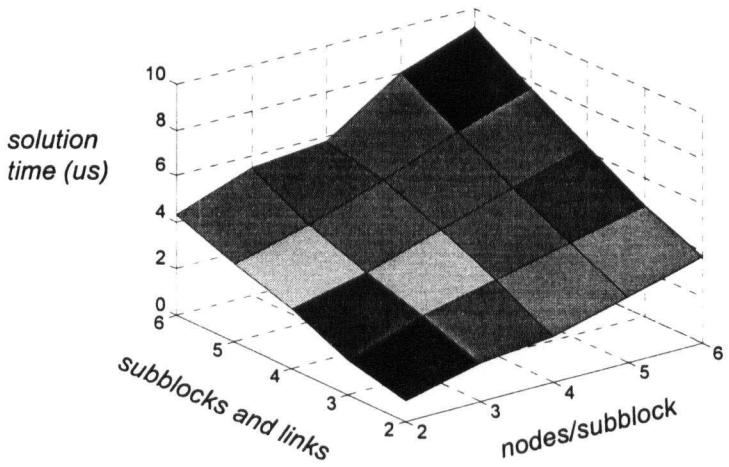


Fig. 178: Solution time for precalculated MATE link matrices, corresponding to blocks with subblocks from two to six, and with two to six nodes per subblock.

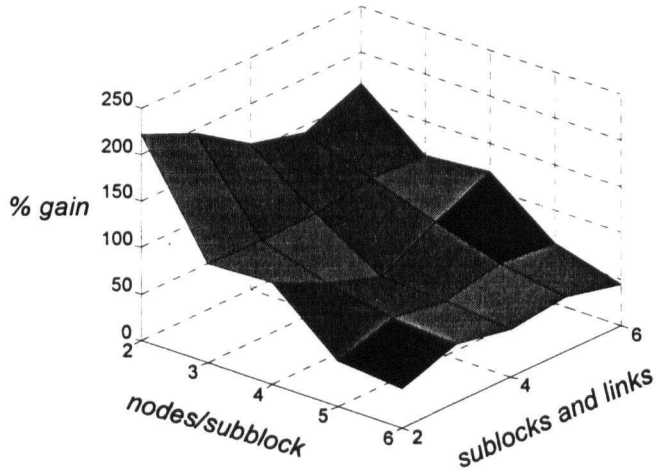


Fig. 179: Percentage gains of precalculating the links matrices vs. calculating them on the run, both within MATE's framework, for different number and sizes of

14.5 Accuracy

To validate the accuracy of a simulation algorithm and its software, the ideal validation tool would be the actual system being simulated. A contrast between the predicted behaviour of the system as simulated, versus the observed behaviour of the real system. In the case of a simulator for an electric power network for the kind of tests targeted in this project, such procedure is out of the question: It is impractical to subject the actual power network of a province (or part of) to this or that contingency that can be contrasted against the one predicted by the simulator.

The next best path[3], and the one used in this case, is to validate the accuracy of the simulator against an already thoroughly validated simulator (albeit a non real time one), the EMTP in this case. The EMTP is the industry standard for transients simulation in power electric networks. The EMTP brings with it more than three decades of validation at hundreds of sites all over the world.

For this section, the tests associated with separate issues of the algorithm, models, solution process, etc., are included in separate subsections.

14.5.1 HVDC Module and its Controller Model

The HVDC module model, the corresponding controller model, in OVNI's implementation were tested and validated for steady state and under fault operating conditions by performing comparisons with the Electromagnetic Transients Program EMTP (Microtran® Version). This section presents a detailed description of these test cases. These simulations do not include asynchronous switching compensation techniques to reinitialize the solution during current commutations and, therefore, present the characteristic spikes of fixed time step solutions. Simulations illustrating the effectiveness of such compensation techniques are presented in a separate section.

The validation tests reported in this section were performed in comparison with Microtran® version 2.08h, with the version of the simulator's code dubbed DU-99[4]. Both programs were run on a Pentium Pro 200 MHz workstation under Windows 95. The five test cases included in this suite are: a) Steady-state; b) Saturation Model for

[3] Another possibility would have been to contrast the simulator against a TNA simulation, but the EMTP, as intimated in the introduction to this thesis, has substituted the TNA for most applications.

[4] Acronym coined at the UBC-RT Lab after the author commented on the non-AI nature of the version used as driver for these tests. That version was nicknamed "Dumbo." So are labelled the test curves.

the three-phase transformers including zero sequence flux; c) AC fault; d) DC fault; e) Commutation failure.

14.5.1.1 Steady-State Validation Test

To assess the HVDC model validity under steady state, a single HVDC six-valve case was set up and run on DU-99, and on the EMTP/Microtran®. In this test, saturation modelling of the transformer was turned off. Saturation model validation is presented in Section 14.5.1.2.

Signals on both sides of the model, as calculated by DU-99, were compared with the corresponding ones obtained with EMTP/Microtran®, namely: primary current of the transformer, Fig. 181, and 182; and DC voltage at the load, Fig. 183 and 184.

Apart from the commutation spikes, the match between the two programs is very good. The reason why the commutation spikes do not appear in Microtran's output is that Microtran does not plot the first half step of the combined trapezoidal/backward Euler's CDA implementation. The HVDC model uses only the backward Euler's rule at full-size integration steps and all simulation steps are plotted.

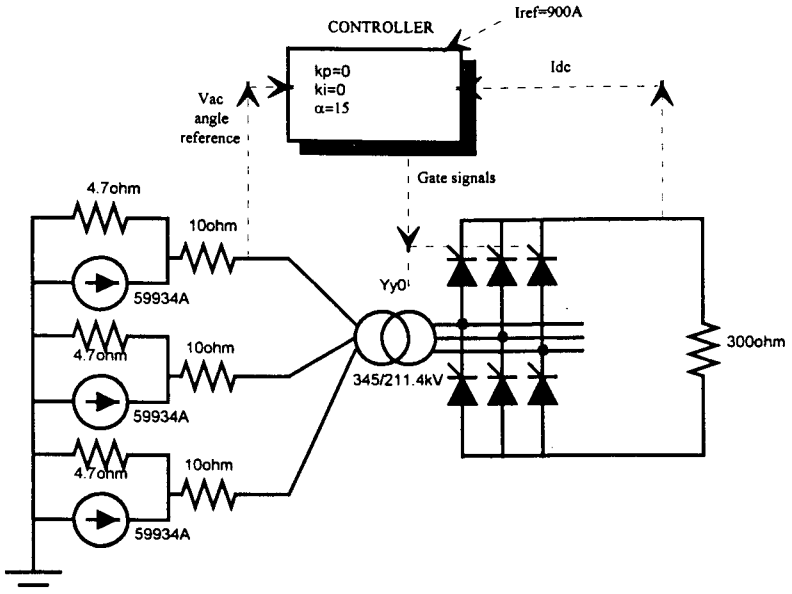


Fig. 180: Single module, six-valve test case used to validate the HVDC module under steady state.

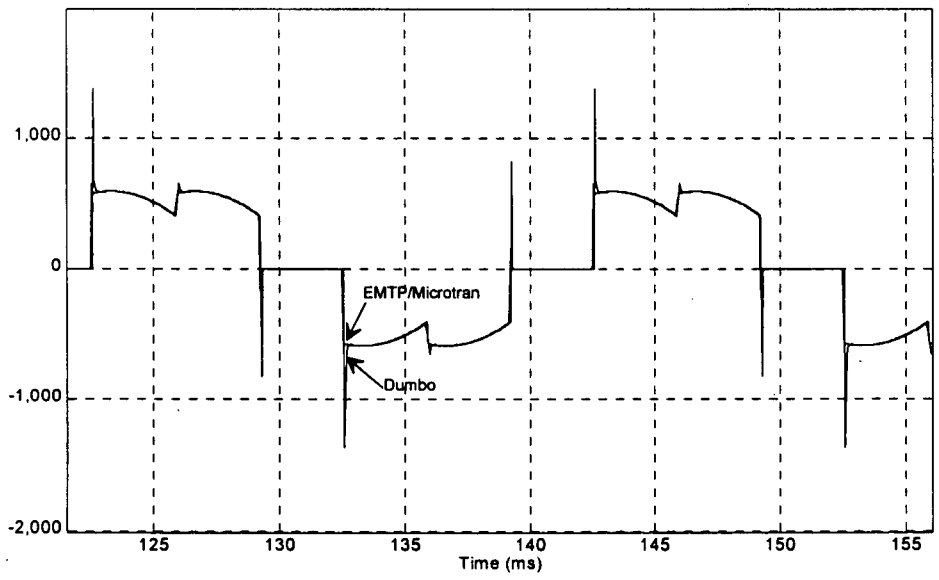


Fig. 181: Primary current, steady state, linear transformer core. EMTP/MICROTRAN and Dumbo (DU-99).

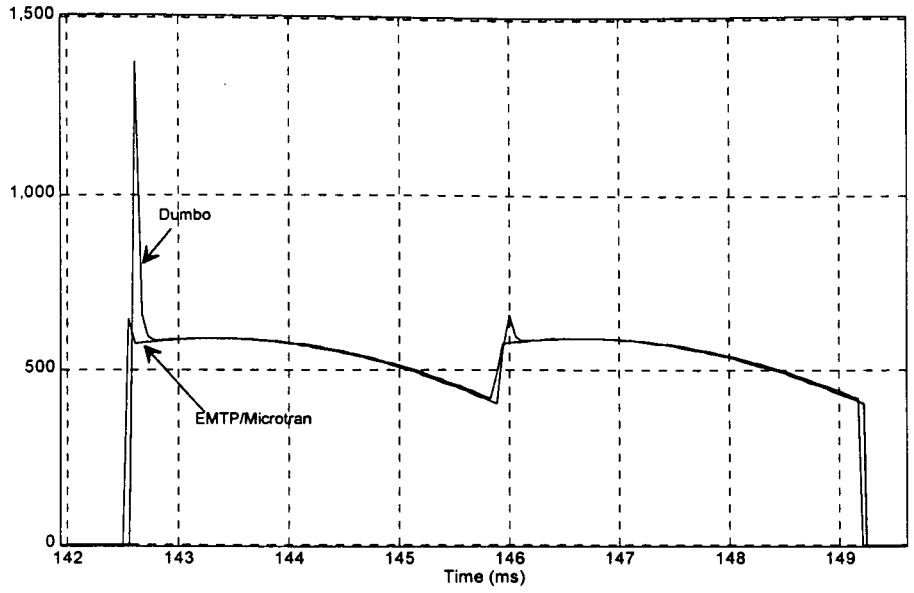


Fig. 182: Primary current, steady state, linear transformer core, EMTP/Microtran and Dumbo. A detail view.

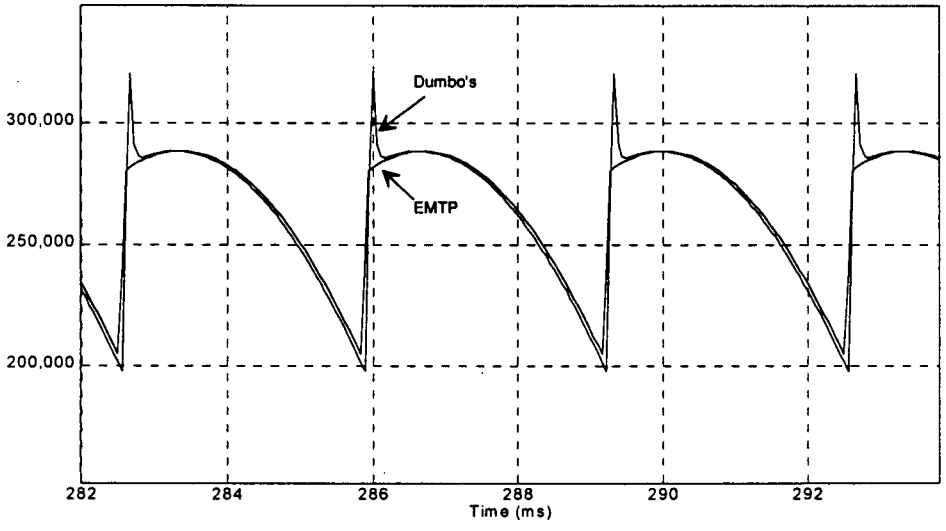


Fig.183: DC voltage in steady state: EMTP/Microtran and Dumbo.

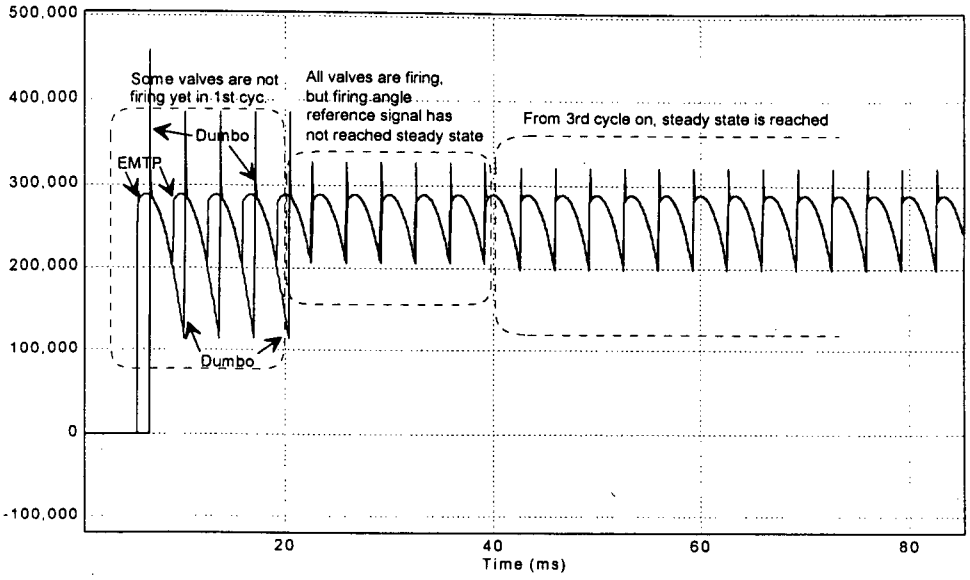


Fig. 184: Voltage before and at steady state: EMTP/Microtran and Dumbo. Initialization, two cycles for Dumbo.

14.5.1.2 Saturation of Transformer Core

To assess the validity of the proposed model for three-leg three-phase transformers, the same single HVDC six-valve case used for steady state assessment (Fig. 180) was simulated with nonlinear inductors in the EMTP/Microtran case file, and compared with the results produced by DU-99 with the saturation module enabled.

Signals on both sides of the model, as calculated by DU-99, were compared with the corresponding ones obtained with the EMTP/Microtran, namely: primary current of the transformer, Fig 186 and 187; and DC voltage at the load, Fig. 188. Output voltage change was not observable as compared with the case with no saturation. This was expected, given the relatively low impedance between the bridge and the ideal sources of the Thevenin equivalent of the AC power group. Saturation distortion of primary current is noticeable, as can be seen by comparing Fig. 185 from the validation test for steady state with no saturation, against Fig. 186 and 187.

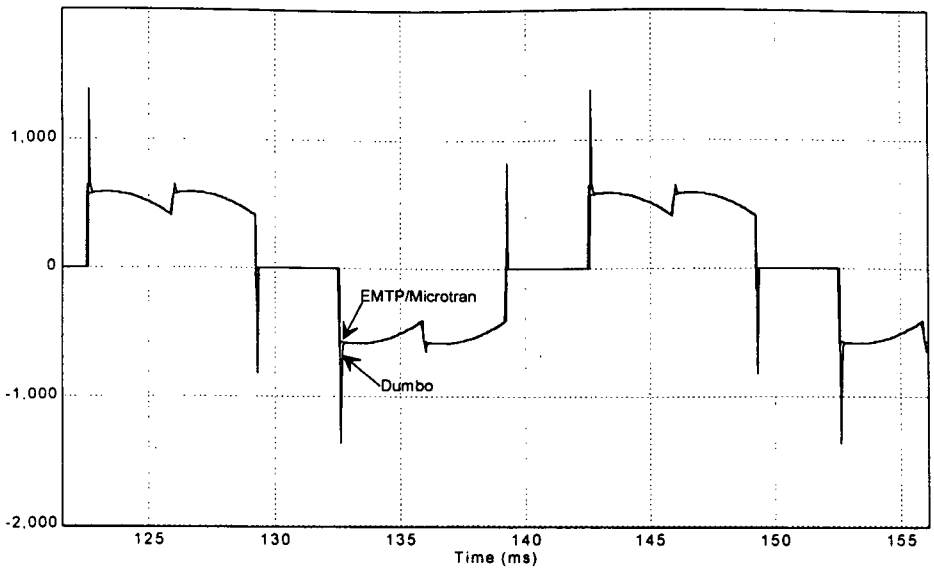


Fig. 185: Zoom on the primary current, steady state, linear transformer core. EMTP/Microtran and Dumbo. Note: Both coincide but for the spikes introduced by Dumbo without DSDI activated, Sec. 7.7, Microtran avoids them using CDA.

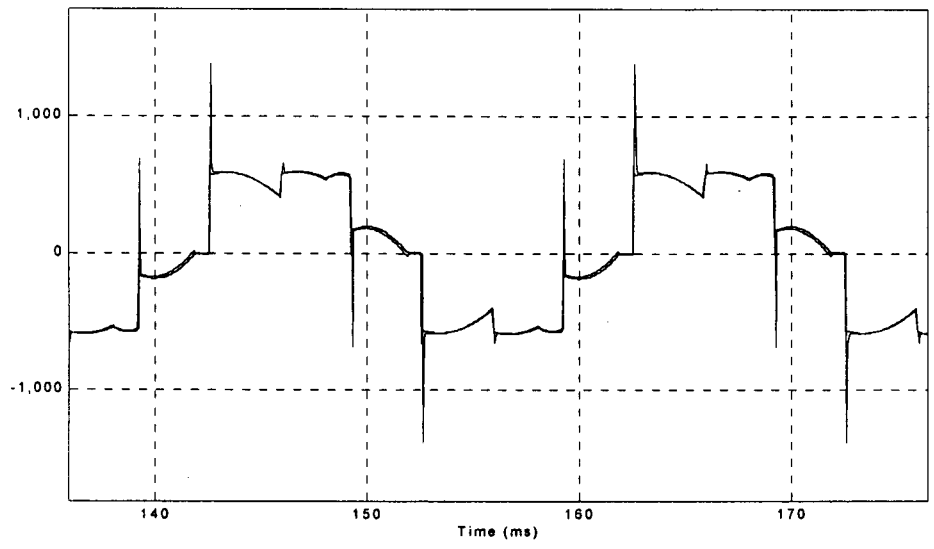


Fig. 186: Primary current, steady state, non-linear transformer core. EMTP/Microtran and Dumbo. See note in caption for Fig. 185.

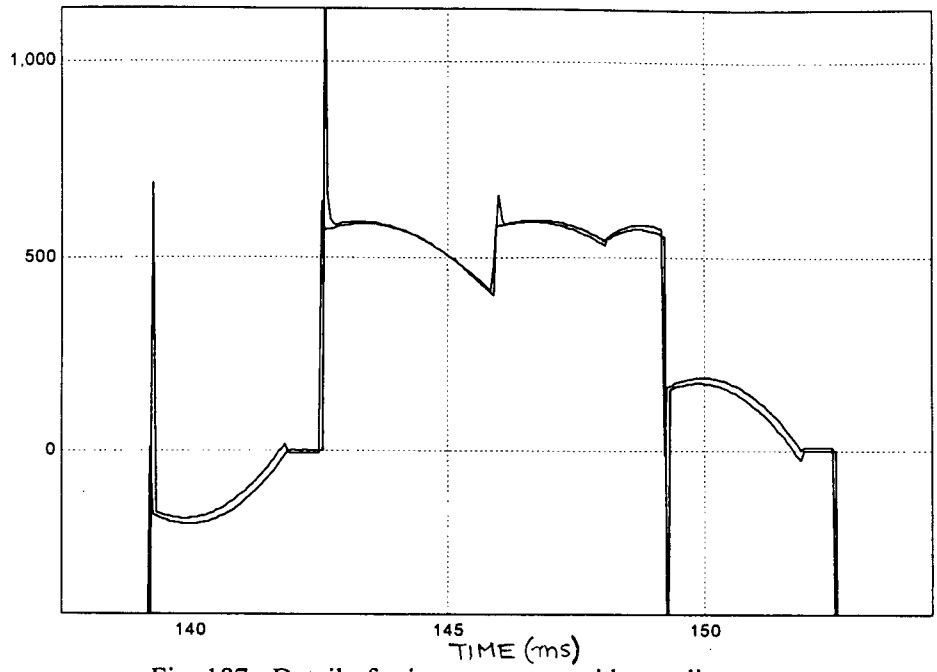


Fig. 187: Detail of primary current with non-linear core.
EMTP/Microtran and Dumbo. See note to Fig. 185.

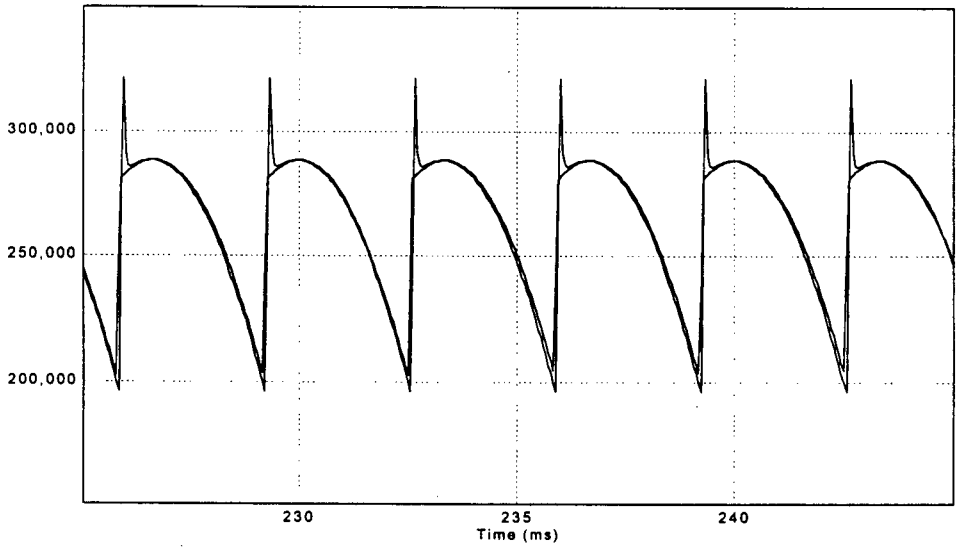


Fig. 188: DC voltage, with non-linear transformer core.
EMTP/Microtran and Dumbo. See note to Fig. 185.

14.5.1.3 Single Phase AC Fault

To validate the performance of the HVDC module during faults on the AC side of the bridge, a 12-valve rectifier case, Fig. 189, was prepared. The test case was run both on DU-99 and on Microtran.

The DC current leaving the HVDC module, IDC, can be seen in Fig. 190, which shows both Microtran's and DU-99 results.

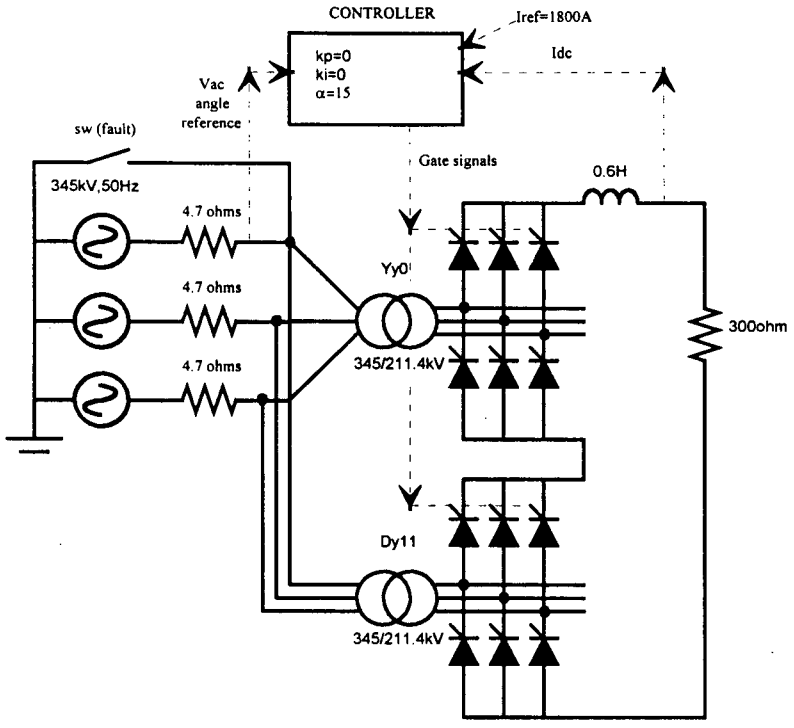


Fig. 189: 12-valve case to validate behaviour of HVDC model under AC faults.

The small initial one degree firing angle difference is due to Microtran taking the reference angle from the ideal sources versus DU-99 taking the reference angle from the primary of the tranformers. That difference becomes greatly amplified under the large currents imposed by the short circuit on the AC side. In Fig. 191, the "ideal" reference voltage for firing angle used by Microtran is compared to the "actual" sub-station reference voltage used by DU-99 to synchronize its gate signals. The small initial error, rounded to one degree, incurred by Microtran becomes a huge 18 degree

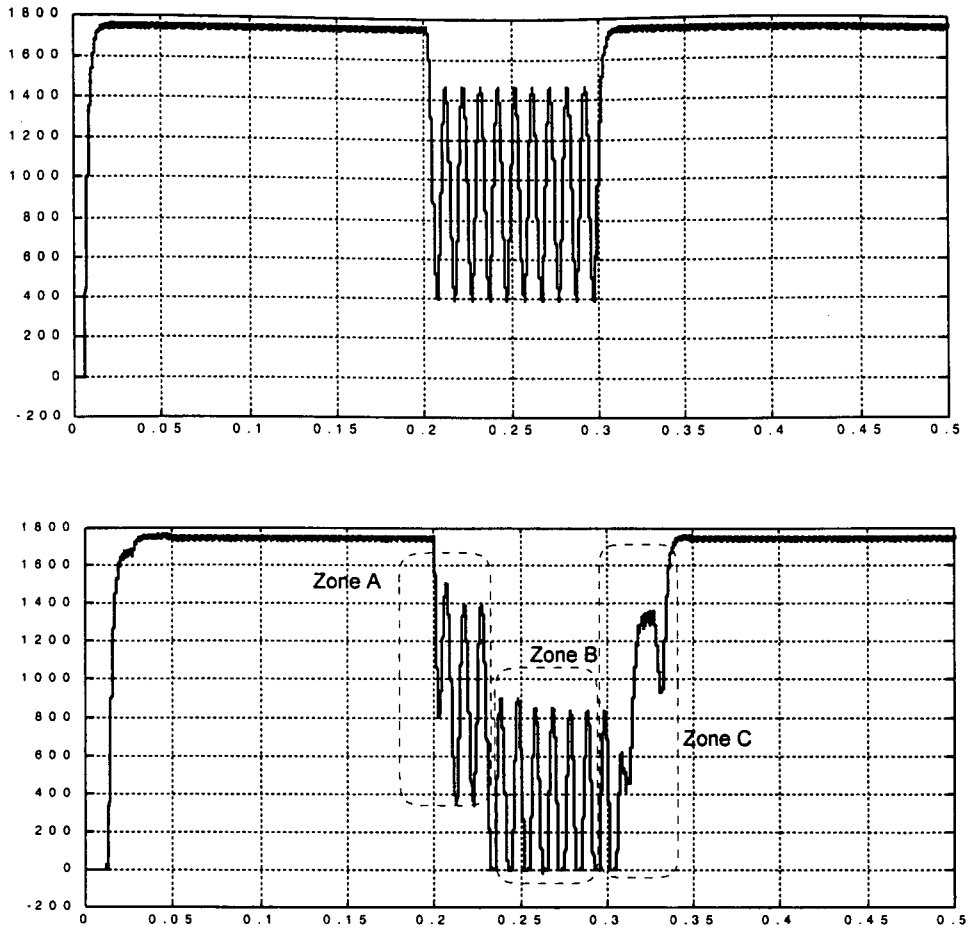


Fig. 190: DC current, as calculated by: a) Microtran; b) Dumbo. Note: Microtran version used did not have variable control signal implemented.

firing angle error during the fault, as seen in Fig. 191.

In Fig. 190b, which shows the DC current as calculated by DU-99, one can identify three zones of interest against the uniformity of Microtran's results in Fig. 190a. In zone A of Fig. 190b, DU-99's controller is still firing the bridge's valves using the angle reference obtained at the last going-up zero crossing of the reference voltage, Fig. 191, which is the same as Microtran is doing. This is the reason for the match between both results in this zone. Zone B starts when, into the faulted period, the reference voltage crosses zero going up again. At that point, DU-99 notices the shift of almost 18 degrees, and corrects its firing signals to maintain the prescribed fifteen degrees, while Microtran continues to use the same reference, introducing an effective firing angle off by the above mentioned 18 degrees. Right after the fault ends, see

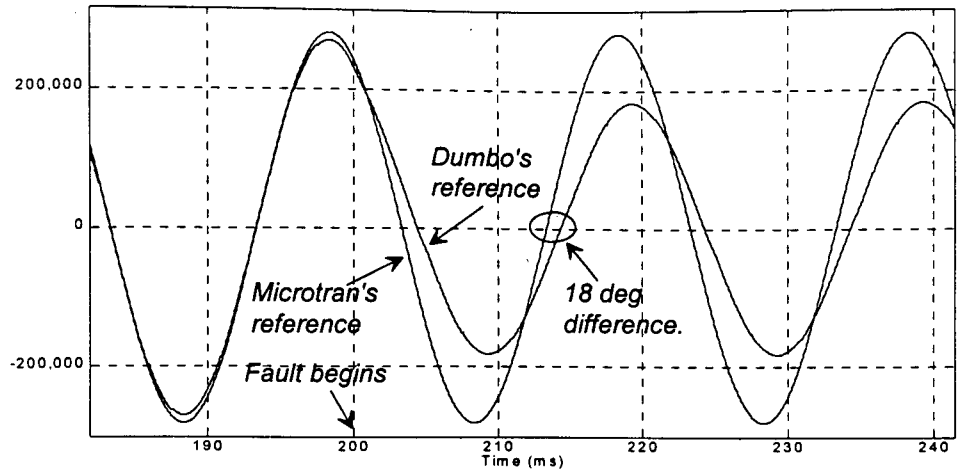


Fig. 191: Angle reference voltage for a) Microtran; b) Dumbo. Observe the small phase error before the fault, and the large error during the shortcircuit.

Fig. 192, DU-99 continues to use its previous angle reference, and then, past the fault end, at the next going-up zero crossing of its reference, DU-99 readjusts its firing angles correspondingly to keep the desired fifteen degrees.

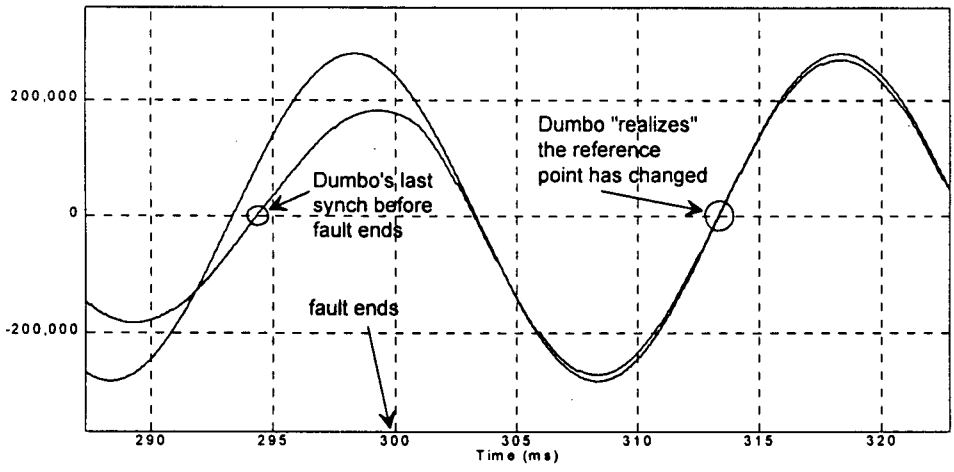


Fig. 192: Angle reference voltage for Microtran and Dumbo near the end of the fault.

14.5.1.4 DC Fault

The case used to validate the HVDC module against the EMTP/Microtran was the twelve valve double bridge Yy/Yd case illustrated in Fig. 193. A switch across the DC load simulated a low impedance short-circuit. To observe the recovery of the model after a DC fault removal, the switch simulating the short circuit opens after a short time.

In Fig. 194, the DC current across the smoothing reactor is shown before, during, and after the DC short circuit was applied at t = 0.2 sec. and removed at t = 0.3 sec.

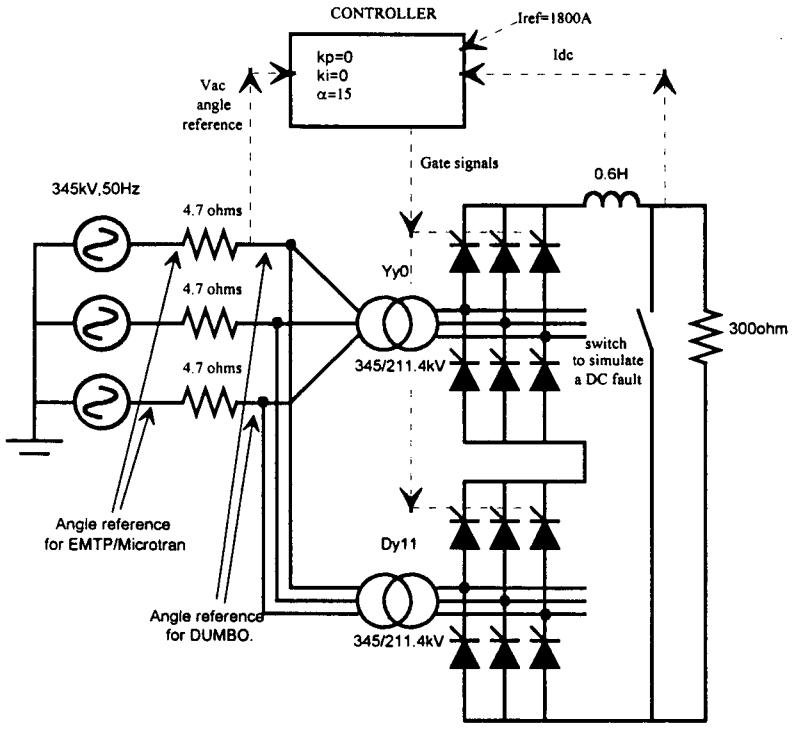


Fig. 193: A double bridge, twelve valve case used to explore the HVDC module during and after a low impedance fault on the DC side.

EMTP/Microtran predicts a slightly smaller DC current during the fault. The reason for that lies in the way Microtran measures the firing angle of the valves. Microtran's reference angle is taken from the voltage between phases A and C of the ideal sources in the Thevenin equivalent representing the external system.

DU-99 uses instead, as angle of reference, the phase of the voltage between

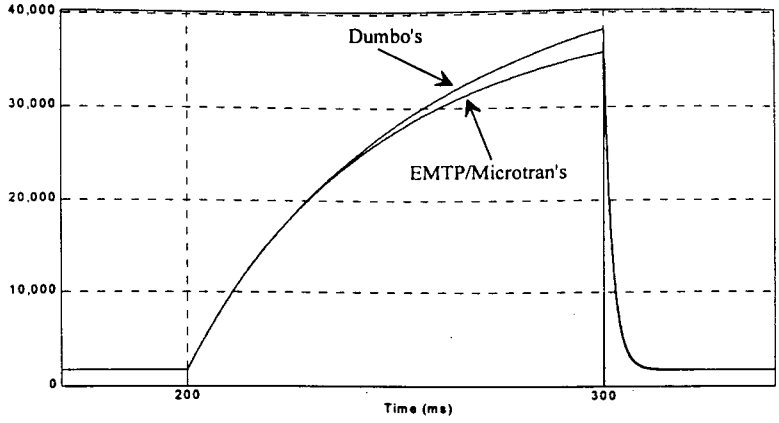


Fig. 194: Fault current (DC-side): a) EMTP/Microtran; b) Dumbo.

phases A and C of the primary of the three-phase transformer, see detail in Fig. 193. The phase difference between the two references used by the two programs is less than one degree under steady state conditions, that is, before the fault (after it as well), Fig. 195. However, during the high currents period of the fault, both references drift away from each other as shown in Fig. 195 and 196. And, Microtran's firing at fifteen degrees from its ideal reference is translated into an effective firing angle of more than forty degrees, Fig. 196, this reducing the feeding DC voltage, and the predicted current.

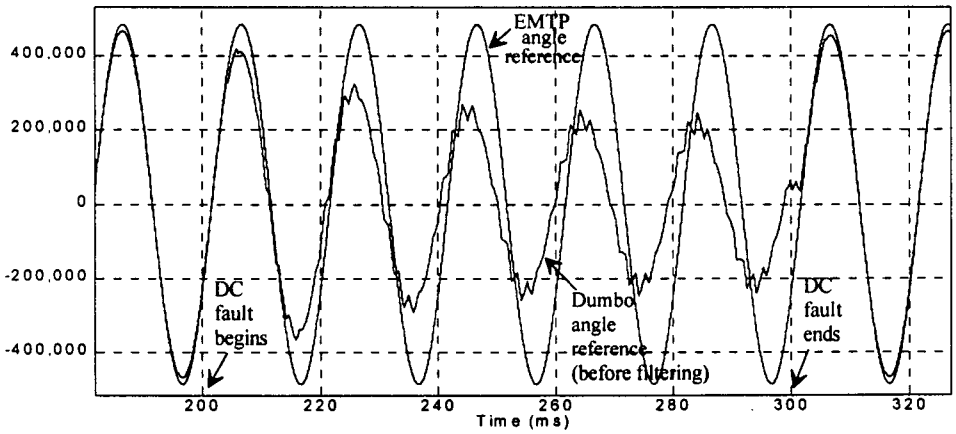


Fig. 195: Angle reference signals for EMTP/Microtran and Dumbo. Before, during, and after the DC fault.

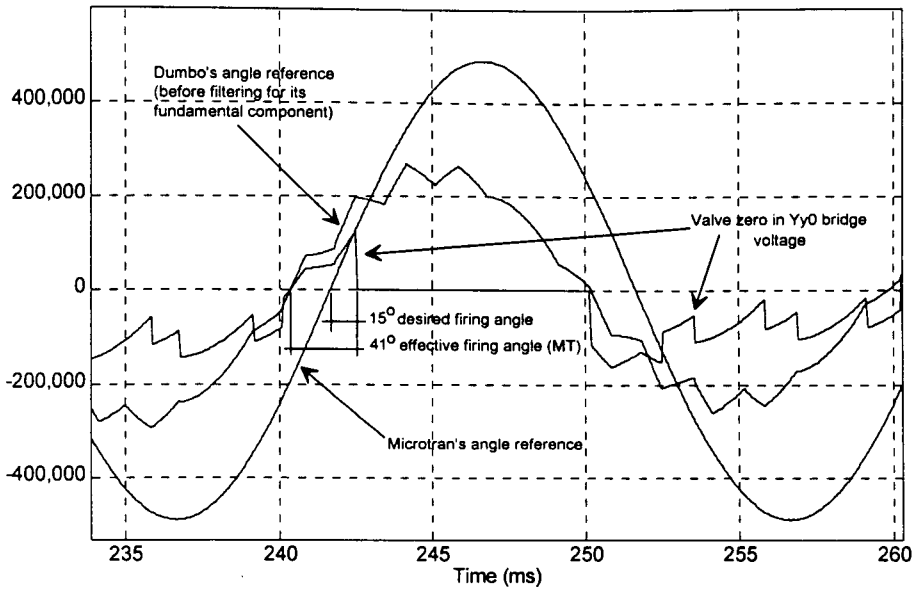


Fig. 196: During the DC fault period: a) Firing angle reference voltage, Vac for Microtran;
b) Reference voltage Vac, for Dumbo;
c) Voltage across valve zero in the Yyo bridge as obtained by Microtran.

14.5.1.5 Commutation Failure

In a case with two bridges (twelve valves) operating as an inverter fed by an 800 kV dc voltage source and a resistor (Thevenin equivalent of the rectifier group), during an AC single phase fault, the first valve of the Yy0 bridge group fails to open, and prevents its next-in-sequence to operate: a commutation failure scenario. In Fig. 197, current through both valves is shown; the failed attempt of valve zero to go off, and of valve two (Valves are numbered 0, 1, 2, 3, 4, and 5, in the normal firing sequence) to take over, is illustrated.

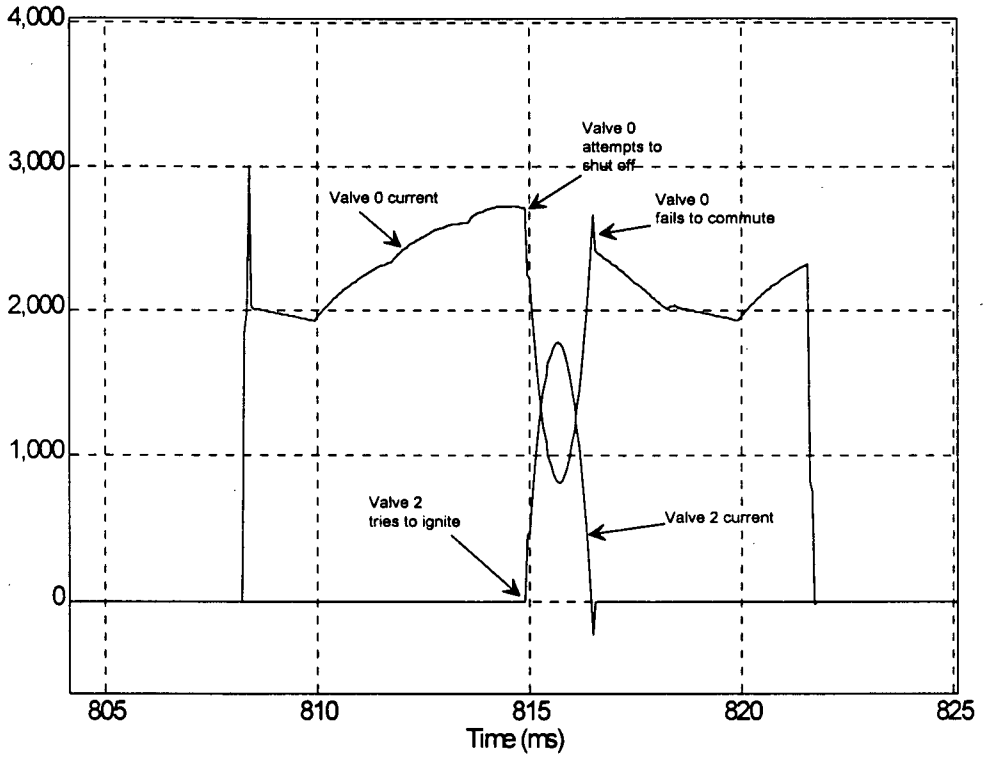


Fig. 197: Valves zero and two of HVDC module Yy0

The configuration of the test case is shown in Fig. 198. The proportional-integrative controller is set at 1880 A, with Kp = 0.0001 and Ki = 0.00001. Firing angle begins at 115 degrees, and is left to the care of the controller to maintain the reference DC current. The AC single phase to ground fault is simulated by the switch included in Fig. 198.

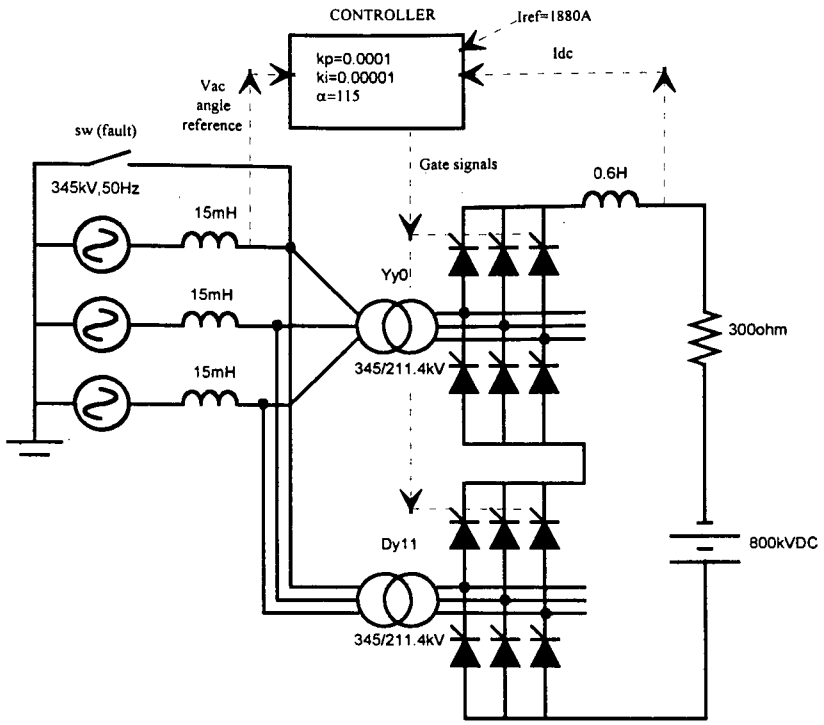


Fig. 198: Twelve valve, double bridge inverter case used to investigate commutation failure modelling.

Figure 199 shows the steady state obtained by the controller on the inverter, with the settings mentioned before. This figure also illustrates the recovery of the controller-module group after the fault is cleared.

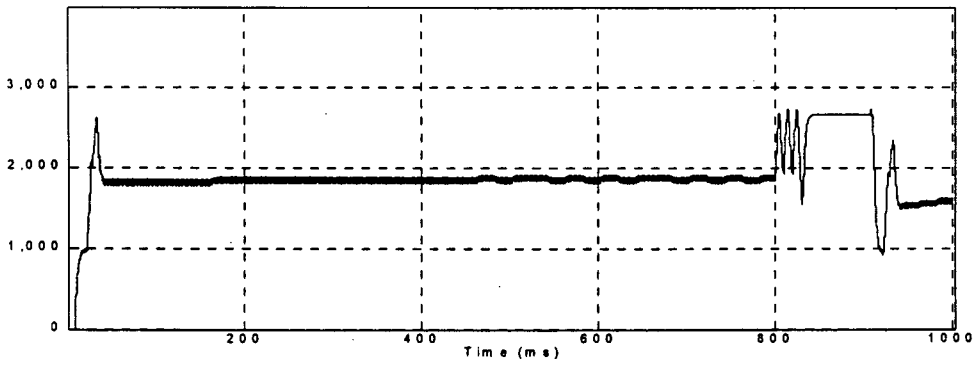


Fig. 199: DC current before, during, and after the AC single phase fault, in the inverter.

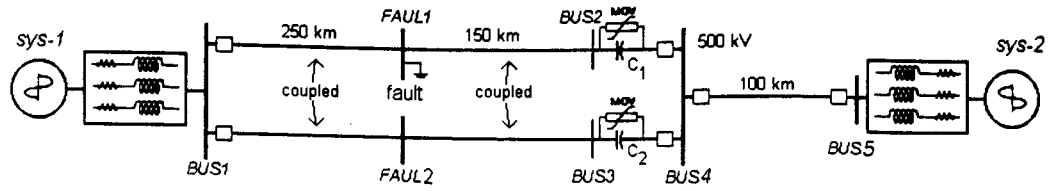


Fig. 200: Fault event simulation for relay testing, with two multi-circuit segments and MOV protection of series compensation.

14.5.3 Relay Testing

The accuracy of the simulator was put to the test case in [9], a case proposed by industry, Fig. 200. The two segments of the power network neighbouring the transmission system where the relay is to operate are represented by three-phase Thevenin equivalents. The case includes two segments of six phase links (two coupled three-phase lines running along the same right of way). One segment, 250 km, before the fault; the other, 150 km, to the right of the fault site ending in a series compensating capacitors protected by MOVs. The system is linked, on the right, by a 100 km three-phase line to the equivalent of the power network labelled sys-2, and directly on the left, to the power system labelled *sys-1*.

The voltage on one of the "healthy" phases (b) at the bus of the fault, when plotted both by the EMTP and by OVNI, are shown on Fig. 201. To the naked eye, there seems to be no difference. When subject to some numerical scrutiny, it turns out to be a difference 0.0005 % between the two solutions[5] [67].

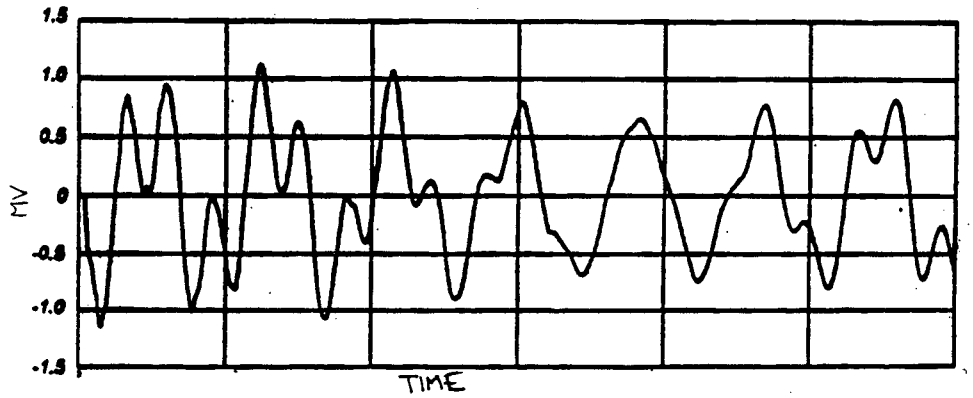


Fig.201: Voltage on phase b at FAULT1.

[5] Up to 0.0025 %, if the percentage is taken with respect to the EMTP value at each time, instead of using the maximum value of the EMTP's solution as percentage reference.

Part VII

CONCLUSIONS

15. CONCLUSIONS AND FUTURE WORK

This project began as a quest for a low cost real time simulator for power networks. Even though the solution has the potential of tackling simulations traditionally solved with load flow, stability programs as well, once the necessary element models have been attached to the core developed in this work, the focus has been kept on achieving real time on two counts: testing protective relays, and testing HVDC controllers.

A bandwidth of 2 kHz at a maximum 3 % magnitude distortion was taken as sufficient. Backward Euler's integration rule was found clean of the traditional blame attached to it, namely: it was found that it delays all frequency components of the signal it processes by the same time shift, half the integration step used. In short, backward Euler's rule, with a magnitude response distortion better that trapezoidal's, and also more stable, was chosen as the rule for the integrator. The integration step necessary under this conditions ranges in the vicinity of 70 μsec/step. The real time deadline.

To meet the deadline mentioned in the previous paragraph, precalculation was presented in a way that does not preclude the generality of the solution, nor taxes the system memory requirements beyond reasonable limints.

To make precalculation a viable option, a three level segmentation scheme was introduced: (a) topological segmentation, followed by (b) MATE, (the multi-area Thevenin equivalent concept) with critical fast topology changing elements (or areas under certain conditions) being used as node shrouds under (c) the node hiding technique introduced in this thesis.

Once MATE segmentation was set in place, Cholesky's linear system solution method was included to find MATE's links currents, which brought a reduction

by half of that particular step.

The MATE concept was extended to take advantage of the presence of grounded voltage sources, and methods to optimize the building of the MATE's links matrix and to process links was introduced. Also included was a procedure to cope with ideal switch operations without collapsing or creating nodes in the network.

Several new models were created for this project: an HVDC module model, as a sample of the benefits of node hiding (node hiding brought the HVDC simulations, already using MATE, from the vicinity of one thousand microseconds per step, down to within the real time deadline); a controller for HVDC valves. A novel technique for modelling the effect of zero sequence magnetic flux in the three-phase core of the HVDC model was also introduced.

To cope with asynchronous operation of switches, the ADC (or DSDI) method of backtracking was introduced. This non iterative procedure prevents the occurrence of numerically induced spikes in the solution.

The problem described in the introductory chapter, and detailed in the "problem" part of this report, has been successfully resolved: real time simulation of an electric network for equipment testing on inexpensive off-the-shelf hardware platforms. Performances of 35 μsec/step for protective relay testing cases, and of 27 μsec/step for HVDC controllers testing cases, were achived on a single Pentium Pro 400 MHz workstation.

The non hardware specific algorithm and code produced make it easy to move on to newer and faster machines as they become available. The solution algorithm, and its code, segment the network in a way that allows for "coarse grain" paralellization, as shown in the results in [73], where the algorithm solved a 234-node power network at a rate of 45 μsec/step on a parallel cluster of five Pentium type processors.

A method to investigate the frequency response, and stability of "hybrid" integration rules that have no closed form transfer function to which a Z-transform process can be applied, has been introduced.

Last but not least, demonstration of the advantages of backward Euler's rule as the main one in the simulator gives the solution presented in this report a touch of elegant simplicity and stability.

15.1 Future work

The author is currently investigating the possibilities of taking advantage of the presence of voltage sources as a simplifying factor to reduce further the complexity of the network.

The models created by Dr. Kwok-Wai in [33] need to be attached to OVNI to pursue load-flow type of simulations.

Further study of latency exploitation [2], to account for the coupling of neighbouring zones running at different integration steps is necessary, and its implementation in OVNI is necessary.

Inclusion of models for electrical machinery, DC motors, induction and synchronous machines; also of frequency dependent transmission lines models.

BIBLIOGRAPHY

[1] P.R. Lawrence and K. Mauch, *Real-Time Microcomputer System Design: An Introduction*, McGraw-Hill Book Company, Inc., New York, 1987.

[2] L.R. Linares and J.R. Martí, "Sub-area latency in a real-time power network simulator," in *Proceedings IPST'95*, IPST'95, Ed., Lisbon, Portugal, September 1995, International Conference on Power System Transients, pp. 541–545.

[3] Hermann W. Dommel, "Digital computer solution of electromagnetic transients in single- and multiphase networks," in *Transactions of Power Apparatus and Systems, Vol PAS-88, No. 4*, IEEE, Ed. IEEE Power Society, April 1969, pp. 388–399.

[4] Hermann W. Dommel, *EMTP Theory Book*, Microtran Power Systems Analysis Corporation, Vancouver, B.C., second edition, 1992.

[5] J.R. Martí, B.W. Garret, H.W. Dommel, and L.M. Wedepohl, "Transients simulation in power systems: Frequency domain and time domain analysis," in *Power System Planning & Operation Section*, CEA, Ed., Montreal, Quebec, Canada, March 1985, Canadian Electrical Association.

[6] J.R. Martí, "Internal report on multi-area thevenin equivalent," A handwritten write-up where Dr. Martí presents MATE through an example circuit with three two/three node subnetworks connected with resistive links. He compares there MATE's number of operations with Woodbury's method and with the standard solution., January 1994.

[7] Gabriel Kron, *Tensor Analysis of Networks*, Wiley and Sons, New York, 1939.

[8] A. Brameller, M.N. John, and M.R. Scott, *Practical Diakoptics for Electrical Networks*, Lowel & Brydone, London, 1969.

[9] J.R. Martí and L.R. Linares, "Real-Time EMTP-based transients simulation," in *IEEE Trans. on Power Systems, Vol. 9, No. 3*, IEEE, Ed., Vancouver, B.C., Canada, August 1994, IEEE Power Society, pp. 1309–1317.

[10] J.R. Martí and Jiming Lin, "Suppression of Numerical Oscillations in the EMTP," in *IEEE Proceedingson PES'88*, Power Engineering Society, Ed., Portland, Oregon, July 29-29 1988, IEEE.

[11] Paul M. Chirlian, *Basic Network Theory*, McGraw-Hill Book Company Inc., New York, 1969.

[12] L.A. Dunstan, "Digital load flow studies," in *Trans. AIEE, Vol. 73, pt. IIIA*, AIEE, Ed. AIEE, 1954, pp. 825–832.

[13] W.D. Stevenson, *Elements of Power System Analysis*, McGraw-Hill Book Company, New York, second edition, 1962.

[14] G.W. Stagg and A.H. El-Abiad, *Computer Methods in Power Systems Analysis*, McGraw-Hill Book Company, New York, 1968.

[15] E.W. Kimbark, *Power System Stability*, John Wiley and Sons, Inc., New York, 1948.

[16] H.E. Brown and C.E. Person, "Short circuit studies of large systems by the impedance matrix method," in *Proc. Power Ind. Computer Appl.*, Power Ind. Computer Appl., Ed., Pittsburgh, Pa., 1967, Power Ind. Computer Appl., pp. 335–342.

[17] Institute of Electrical Engineering Universidad Nacional de San Juan, "Transients network analyser," in *Web page at www.iee.unsj.edu.ar*, San Juan, Mendoza, Argentina, 2000.

[18] P.G. McLaren, R. Kuffel, R Wierckx, J. Giesbrecht, and L. Arendt, "A real time digital simulator for testing relays," in *IEEE Transactions on Power Delivery, Vol. 7, No. 1*, IEEE, Ed. IEEE Power Society, January 1992, pp. 207–213.

[19] M. Kezunoviç, M. Aganagic, V. Skendzic, J. Domaszewicz, J.K. Bladow, D.M. Hamai, and S.M. McKenna, "Transients computation for relay testing in real-time," in *Transactions on Power Delivery, Vol. 9, No. 3*, IEEE, Ed. IEEE Power Society, July 1994, pp. 1298–1307.

[20] R.C. Durie and C. Pottle, "An extensible real-time digital transient network analyser," in *Trans. PWRS, Paper WM 175-0, Winter Meeting*, IEEE, Ed. IEEE Power Society, January 1992.

[21] D.M. Falcao, E. Kaszkurewicz, and H.L. Almeida, "Application of parallel processing techniques to the simulation of power system electromagnetic transients," in *Trans. on PWRS, Paper WM 287-3, Winter Meeting*, IEEE, Ed. IEEE Power Society, January 1992.

[22] S.Y. Lee, H.D. Chiang, K.G. Lee, and B.Y. Ku, "Parallel power system transient stability analysis on hypercube multiprocessors," in *Trans. on Power Systems, Vol. 6, No. 3*, IEEE, Ed. IEEE Power Society, August 1991, pp. 1337–1342.

[23] Soumagne J.-C., Mercier P., Rizzi J.-C., Do V.Q., Sybille G., and Giroux P., "Development of the IREQ simulator," *Proceedings of 1997 International Conference on Digital Power Systems Simulators*, pp. 31–6, 1997.

[24] Electricité de France, "Arene: The digital transient network analyser,"

in *Web page: http:// im.edfgdf.fr/ er/ en/ genser/ presta/ logici/ arene/ arene.htm*, Paris, France, 1997.

[25] Kulicke B., Lerch E., Ruhle O., and Winter W., "Netomac-calculating, analyzing and optimizing the dynamic of electrical systems in time and frequency domain," *Proceedings of International Conference on Power Systems Transients (IPST'99)*, pp. 1–6, 1999.

[26] Fujimoto Y., Yuan Bin, Taoka H., Tezuka H., Sumimoto S., and Ishikawa Y., "Real-time power system simulator on a PC cluster," *Proceedings of International Conference on Power Systems Transients (IPST'99)*, pp. 671–6, 1999.

[27] Kuffel R., Giesbrecht J., Maguire T., Wierckx R.P., and McLaren P., "Rtds-a fully digital power system simulator operating in real time," *Proceedings of Stockholm Power Tech International Symposium on Electric Power Engineering*, pp. 49–54 vol.4, 1995.

[28] J.R. Martí and L.R. Linares, "OVNI: An object approach to real-time power system solutions. part i: Mate diakoptics.," in *Submitted*, IEEE, Ed., 2000.

[29] L.R. Linares and J.R. Martí, "OVNI: An object approach to real-time power system solutions. part i: Object oriented solution.," in *Submitted*, IEEE, Ed., 2000.

[30] S. Acevedo, L.R. Linares, J.R. Martí, and Y. Fujimoto, "Efficient HVDC converter model for real time transients simulation," in *Proceedings on Power Delivery, Vol. 1, No. 1*, IEEE, Ed., San Diego, CA, June 1998, IEEE Power Society, pp. 1–7.

[31] Fernando Castellanos, *Full frequency dependent phase-domain modelling of transmission lines and corona phenomena*, Ph.D. thesis, The University of British Columbia, Vancouver, B.C., Canada, 1997.

[32] Fernando José Marcano, "Modelling of transmission lines using idempotent decomposition," M.S. thesis, The University of British Columbia, Vancouver, B.C., Canada, April 1996.

[33] Louie Kwok-Wai, *Aggregation of Voltage and Frequency Dependent Electrical Loads*, Ph.D. thesis, The University of British Columbia, Vancouver, B.C., Canada, July 1999.

[34] Grady Booch, *Object-oriented analysis and design with applications*, vol. 1 of *Benjamin/Cummings series in object-oriented software engineering*, Benjamin/Cummings Pub. Co., Redwood City, Calif., 2nd. ed. edition, 1994.

[35] K. Strunz, L. Linares, J.R. Martí, O. Huet, and X. Lombard, "Efficient and Accurate Representation of Asynchronous Network Structure Changing Phenomena in Digital Real Time Simulators," in *IEEE Trans. on Power Systems*, IEEE, Ed. IEEE Power Society, December 1998.

[36] J.R. Martí, L.R. Linares, J. Calviño, and H.W. Dommel, "OVNI: An Object Approach to Real-Time Power System Simulators," in *IEEE Trans. on POWERCON'98*, IEEE, Ed., Beijing, China, August 18-21 1998, IEEE Power Society, pp. 977–981.

[37] Roberto Rosales S., "Simulation environment for a real time simulator," M.S. thesis, The University of British Columbia, Vancouver, B.C., Canada, April 1997.

[38] Jesús Calviño-Fraga, "Implementation of a real time simulator for relay testing," M.S. thesis, The University of British Columbia, Vancouver, B.C., Canada, April 1999.

[39] J.R. Martí, "Notes on elec 560," Graduate course given by Dr. Martí during the winter of 1992 at the University of British Columbia., 1992.

[40] Gregory Bond, "Notes on software testing," Graduate course given by Dr. Bond during the winter of 1994 at the University of British Columbia., 1994.

[41] Leon O. Chua, *Computer-aided analysis of electronic circuits : algorithms and computational techniques*, Prentice-Hall, Englewood Cliffs, N.J., 1975.

[42] Kezunovic M. and Galijasevic Z., "An advanced PC-based digital simulator for protective relay testing," *Proceedings of 1997 International Conference on Digital Power Systems Simulators*, pp. 9–14, 1997.

[43] Fernando L. Alvarado, Robert H. Lasseter, and Juán J. Sánchez, "Testing of Trapezoidal Integration with damping for the Solution of Power Transient Problems," *IEEE Trans. PAS 102*, , no. 12, pp. 3787–3790, December 1983.

[44] José Martí, *The problem of frequency dependence in transmission line modelling*, Ph.D. thesis, The University of British Columbia, Vancouver, B.C., Canada, 1981, vii, 200 leaves : diagrs. ; 28 cm. Bibliography: leaves 198-200.

[45] J.R. Martí, "Internal report on updating lumped elements sources," January 1991.

[46] Hermann W. Dommel, "Notes on elec 551," Graduate course given by Dr. Dommel during the winter of 1992 at the University of British Columbia., 1992.

[47] L.F. Woodruff, *Principles of Electric Power Transmission*, John Wiley and Sons, Inc., New York, second (13th printing) edition, 1956.

[48] L.M. Wedepohl, "Application of matrix methods to the solution of travelling wave phenomena in polyphase systems," *Proc. IEE*, vol. 110, no. 12, pp. 2200–2210, 1963.

[49] Frank Salazar, "Modelación modal de lineas de transmisión," M.S. thesis, Universidad Central de Venezuela, Caracas, Venezuela, 1982.

[50] S. Chimklai and J.R. Martí, "Simplified three-phase transformer model for electromagnetic transient studies," *Trans. on Power Delivery*, vol. 94SM410-1, pp. 8 pages, July 1994.

[51] Ned Mohan, Tore M. Undeland, and William P. Robbins, *Power Electronics*, vol. 1, John Wiley & Sons, Inc., second edition, 1995.

[52] Araujo A.E.A., Soudack A.C., and Marti J.R., "Ferroresonance in power systems: chaotic behaviour," *IEE Proceedings C (Generation, Transmission and Distribution)*, vol. 140, no. 3, pp. 237–40, 1993.

[53] J.R. Martí, L.R. Linares, and H.W. Dommel, "Current transformers and coupling-capacitor voltage transformers in real-time simulations," *IEEE Transactions on Power Delivery*, vol. 12, no. 1, pp. 164–8, January 1997.

[54] Chung-Wen Ho, E. Ruehly, and P.A. Brennan, "The modified nodal approach to network analysis," in *IEEE Transactions on Circuits and Systems vol.CAS-22, no.6*, IEEE, Ed., June 1975, pp. 504–509.

[55] W.F. Tinney and J.W. Walker, "Solution of sparse network equations by optimally ordered triangular factorizations," in *IEEE Proceedings, Vol. 55*, IEEE, Ed. IEEE, November 1967, pp. 1801–1809.

[56] A. Monticelli, S. Deckmann, García A., and B. Scott, "Real-Time External Equivalent for Static Security Analysis," *Trans. on Power Apparatus and Systems*, vol. PAS98, no. 2, pp. 498–504, March/April 1979.

[57] Quantasm Corporation, "Assembly language tools, tips and tricks (web page link)," in *Web page: http:// www.quantasm.com// opcode_f.html*, 2000.

[58] A.E.A Araujo, H.W. Dommel, and J.R. Martí, "Converter Simulations with the EMTP: Simultaneous Solution and Backtracking Technique," *Proc. IEEE Intl. Conference, Athens, Greece*, vol. Athens Power Tech APT'93, Semptember 5–8 1993.

[59] Araujo A.E.A., Dommel H.W., and Marti J.R., "Simultaneous solution of power and control systems equations," *IEEE Transactions on Power Systems*, vol. 8, no. 4, pp. 1483–9, 1993.

[60] Hermann W. Dommel, *Case Studies for Electromagnetic Transients*, Microtran Power Systems Analysis Corporation, Vancouver, B.C., second edition, 1993.

[61] M. et al. Kezunoviç, C.W. et al. Fromen, and F. Philips, "Experimental evaluation of emtp-based current transformer models for protective relay transient study," in *Transactions on Power Delivery, Vol. 9, No. 1*, IEEE, Ed. IEEE Power Society, January 1994, pp. 405–413.

[62] M. et al. Kezunoviç, C.W. et al. Fromen, and S.L. Nilson, "Digital models of coupling capacitor voltage transformers for protective relay transient studies," in *Transactions on Power Delivery, Vol. 7, No. 4*, IEEE, Ed. IEEE Power Society, October 1992, pp. 1927–1935.

[63] L. Kojovic, M. et al. Kezunoviç, and C.W. et al. Fromen, "A new method for the CCVT performance analysis using field measurements. signal processing and EMTP modeling," in *Transactions on Power Delivery, Vol. 9, No. 4*, IEEE, Ed. IEEE Power Society, October 1994, pp. 1907–1915.

[64] J.R. Lucas, P.G. McLaren, W.W.L. Keerthipala, and R.P. Jayasinghe, "Improved simulation models for current and voltage transformers in relay studies," in *IEEE Transactions on Power Delivery, Vol. 7, No. 1*, IEEE, Ed. IEEE Power Society, January 1992, pp. 152–159.

[65] G.R. Slemon and A. Stranghen, *Electric Machines*, vol. 1, Addison-Wesley. pp. 139–141., 1980.

[66] M. Kezunovic, et al., J.R. Martí, H.W. Dommel, and L.R. Linares, "Real Time Simulator for Relay Testing, Progress report #FC65-90WA 07990 to

the Department of Energy, Western Area Power Administration," August 1991.

[67] H.W. Dommel, "Conversation with Dr. Dommel on possibilites to model the zero sequence component of three-phase core transformers magnetization," At Dr. Martí's office, June 1998.

[68] John G. P. Barnes, *Programming in Ada 95*, Addison-Wesley, 1996.

[69] Bjarne Stroustrup, *The C++ programming language*, Addison-Wesley, third edition.

[70] Nicklaus Wirth, *Algorithms + Data Structure = Programs*, vol. 1 of *Series on Automatic Computation*, Prentice-Hall International, Inc., London, first edition, 1976.

[71] Steve McConnell, *Code Complete*, vol. 1, Microsoft Press, Redmond, Wash., first edition, January 1993.

[72] Michel Goossens, Frank Mittelbach, and Alexander Samarin, *The LaTeX Companion*, vol. 1, Addison-Wesley, Reading, Massachusetts, first edition, 1993.

[73] Jorge Ariel Hollman, "Real time distributed network simulation with PC clusters," M.S. thesis, The University of British Columbia, EE, Vancouver, Canada, December 1999.

[74] Luis Rafael Linares-Rojas, "A real time simulator for power electric networks," M.S. thesis, The University of British Columbia, EE, Vancouver, Canada, April 1993.

[75] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes in C*, Cambridge Univ. Press, Cambridge, Mass., January 1993.