A symbolic-numeric method to analyse nonlinear differential equations in Fourier domain

Nicolas Ratier, Michaël Bruniaux

A Symbolic–Numeric Method To Analyse Nonlinear Differential Equations in Fourier Domain

NICOLAS RATIER, MICKAEL BRUNIAUX
Institut FEMTO–ST, département LPMO, CNRS UMR 6174
Ecole Nationale Supérieure de Mécanique et des Microtechniques
32 avenue de l'Observatoire, 25044 Besançon Cedex
FRANCE
nicolas.ratier@lpmo.edu http://www.lpmo.edu/~ratier

Abstract: - This paper presents a method to express, in symbolic form, any system of algebrico–differential equations into a nonlinear system of Fourier coefficient of the unknowns. The solution of the nonlinear equations generated in the last step of the method gives an approximation of the steady–state solution of the differential equations. Main applications should be found in the domain of ultra–stable oscillator circuits. The paper explains how to face with the two main difficulties of this symbolic computation: the processing of the nonlinear components and the control of the large number of coefficients. The solution proposed is inspired from compiler construction techniques.

Key-Words: - Nonlinear differential equation, Fourier Series, Symbolic–Numeric method.

1 Introduction

The motivation of this work described in this paper is the interest that circuit designers have in simulation techniques that could tackle the problem of finding steady–state solutions for nonlinear circuit, particularly for ultra–stable crystal oscillators.

Crystal oscillators use very high–Q resonators to achieve very high stability and low noise. The Q of the best resonators can be as high as 1000000 leading to a Q of up to 100000 in well designed oscillators. A Q of 100000 implies that the time constant of the turn–on transient for such an oscillator is roughly 100000 cycles of the oscillation frequency in length. Clearly, transient simulation of such circuit to steady–state is very painful; however steady state is required in order to determine important characteristics, such as the output power, amplitude and frequency of the oscillator.

The method usually used to simulate this kind of circuit is the so–called "harmonic–balance" method [1]. Roughly, in this numerical method the linear subcircuits are computed in the frequency domain and the nonlinear subcircuits are computed in the time domain. The name stems from an approach based on current balancing between the linear and nonlinear subcircuits.

Our method replaces, by symbolic computation, the nonlinear differential equation system that describes the whole circuit by a system of nonlinear equations of Fourier coefficient whose solution is an approximation of the steady–state response of the circuit. The harmonic analysis methods impose the steady–state conditions by virtue of Fourier expansion of the unknown functions. Simulation times are, therefore, independent on the length of transients. The ultimate aim is to perform a real time simulation of ultra–stable oscillator circuits.

This method is faced with two major difficulties: The processing of the nonlinear components and the control of a very large number of coefficients. This paper proposed a solution inspired from compiler construction technique to solve these two problems.

2 Principle of the symbolic harmonic method

$P(T_0)$ denotes the set of all periodic functions of bounded variation with period T_0. Given a differential equation system of the form (Eq. 1) where $u \in P(T_0)$ is the stimulus waveform, x is the unknowns waveform to be found and f is continuous and real.

$$f(x, x', u) = 0 \tag{1}$$

If we assume that the solution x exists, is real, and belongs to $P(T_0)$, x can be written as

a Fourier series (Eq. 2) where $\omega_0 = 2\pi/T_0$.

$$x(t) = X_0 + \sum_{k=1}^{\infty} X_k \cos(k\omega_0 t) + \sum_{k=1}^{\infty} X_{-k} \sin(k\omega_0 t) \tag{2}$$

Substitute the assumed solution and its derivative into f. Note that $x \in P(T_0)$ implies $x' \in P(T_0)$, and since $u \in P(T_0)$ as well, $f(x, x', u) \in P(T_0)$. Write the resulting equation as Fourier series (Eq. 3).

$$f(x, x', u) = F_0 + \sum_{k=1}^{\infty} F_k \cos(k\omega_0 t) + \sum_{k=1}^{\infty} F_{-k} \sin(k\omega_0 t) \tag{3}$$

Using the orthogonality of sinusoids at different frequencies, rewrite (Eq. 3) as a system of nonlinear equation (Eq. 4), one for each harmonic defined by the assumed solution.

$$F(X, k) = 0 \text{ for all } k \in Z \tag{4}$$

Unfortunately, even by truncating the Fourier series to N terms, the method described above is difficult to apply by symbolic computation. A direct and naive application of the method to build the nonlinear system leads to an exponential growth of the number of terms. Moreover, the second problem is the management of the nonlinear terms of the differential equation.

3 Build the ODE tree

The technique and data structure for translating the ODE[1] into nonlinear equations of Fourier coefficients is inspired by compiler techniques [2]. The ODE is seen as a language written in an human–readable grammar (the "concrete syntax"). Our translation program reads the ODE and computes a structured representation of it (into an "abstract syntax"). This representation is then processed to produce compact equations of Fourier coefficients. The abstract syntax makes a clean interface between the parser and the later phases of the translation. The abstract syntax tree conveys the phrase structure of the source program, with all parsing issues resolved but without any semantic interpretation.

The concrete grammar of ODE is written as syntax diagrams for clearness in Fig. 6. The conversion of the diagrams into left–recursive Backus–Naur–Form (BNF) is straightforward. The rules for reading syntax diagrams are the following: Every diagram represents one language element, non–terminals are represented by rectangular boxes, and terminals by oval boxes, the run through the diagram starts at the leftmost point.

The `UNKNOWN` terminals are elements of the unknown vector $x(t)$. The `FUNCTION` terminals are either constants or functions already in harmonic form, like the stimulus waveform $u(t)$. The grammar accepts only the transcendental function `EXP`, `COS` and `SIN`. It rejects any other transcendental function (log, tan, ...), inverse of a function, and composite function. To fit all ODE to the given grammar (Fig. 6), one proceeds by introducing an additional unknown to the equation system. This transforms the initial differential equation system into an algebro–differential system. This handling is trivial as shown in the following example.

$$\cdots + \tan V_1 + \cdots + \arccos V_2 + \cdots = 0 \tag{5}$$

becomes

$$\cdots + H_1 + \cdots + H_2 + \cdots = 0 \tag{6}$$

$$H_1 \cos V_1 = \sin V_1 \tag{7}$$

$$\cos H_2 = V_2 \tag{8}$$

The latter equations are now recognized by the grammar. The same method can be directly applied to the inverse of a function and to the composite functions.

The abstract syntax of the ODE is written below in OCaml–like way [3]. This representation has many advantages. It allows one to separate the syntactic description of a language with its semantic analysis and transformation. Moreover, abstract syntax is usually simpler than concrete syntax. Since abstract syntax has a unique tree structure, it is not necessary to reflect binding strength using different types, as we did with different nonterminals to avoid ambiguity.

```
type odeTree =
       SUM  of odeTree * odeTree
     | PROD of odeTree * odeTree
     | DIFF of odeTree
     | EXP  of odeTree
     | COS  of odeTree
     | SIN  of odeTree
     | UNK  of unknown
     | FCT  of function
```

[1] Ordinary Differential Equation

The mapping of concrete grammar to abstract syntax is illustrated for Eq. (9).

$$-a\sin(\omega t) + G_1 V_1(t) + C_1 \frac{dV_1}{dt}(t) \quad = \quad 0 \quad (9)$$

The tree of Fig. 1 which decorates the "abstract syntax" represents Eq. (9). The abstract syntax shows that the tree is binary so that each node involves only one algebraic operation (`SUM`, `PROD`, `DIFF`, ...). Moreover the leafs are either unknown functions ($V_1(t)$) or function already in series form ($-a\sin\omega t$, C_1 and G_1 constant).

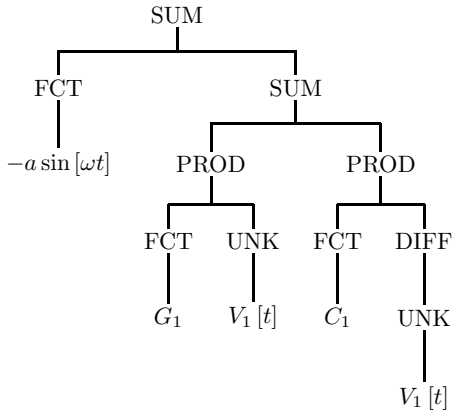

Fig. 1: Binary tree of Eq. 9

4 Parse the ODE tree

The problem is to express each equation in harmonic form, with the constraint that the coefficient number generated during this phase does not increase exponentially. The abstract syntax processing consists of progressively reducing the tree in harmonic form from the bottom to the top: at each node new coefficients expressed in function of the previous ones will be generated. At the final step, it will remain only a Fourier series with manageable coefficients.

- `UNK(V1(t))` is replaced by its Fourier series limited to N terms, i.e. it returns:

```
  A00V1
+ A01V1 cos  wt + B01V1 sin  wt
+ A02V1 cos 2wt + B02V1 sin 2wt
+ A03V1 cos 3wt + B03V1 sin 3wt
```

- `FCT(G1)` is already under the righ form, so it is replaced by itself.
- `SUM(S1,S2)` with

```
S1 = A00S1 + A01S1 cos wt
           + B01S1 sin wt + ...
S2 = A00S2 + A01S2 cos wt
           + B01S2 sin wt + ...
```

is reduced by generating new coefficients

```
TT1 = A00S1 + A00S2
TT2 = A01S1 + A01S2
TT3 = B01S1 + B01S2
... = ...
```

and return

```
TT1 + TT2 cos wt + TT3 sin wt + ...
```

- `PROD(S1,S2)`, `DIFF(S1)` are reduced like `SUM(S1,S2)`, however, the equations associated with the generated coefficients are a little bit more complex than the straightforward case of `SUM(S1,S2)`.

5 Build the nonlinear function trees

The problem is to represent the exponential, cosinus and sinus of a Fourier series into the form of a Fourier expansion. Moreover, the coefficients of the obtained Fourier series must be easy to handle.

The solution of the problem is quite similar to the previous one. It is based on constructing a binary tree and then generating new coefficients in function of the previous ones during the tree parsing. In this problem, the concrete grammar to analyse is written in Fig. 2.

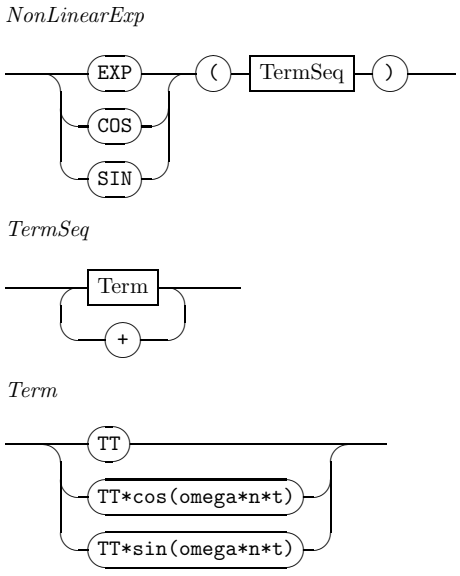

Fig. 2: Concrete grammar of nonlinear functions

The construction of the tree is based on the addition formulae. Of course, each transcendental function `EXP`, `COS`, `SIN` will have a different associated tree. The reduction of the tree will be possible thanks to their associated Bessel series.

A recursive application of the addition formula for the exponent function (Eq. 10) allows one to define an abstract grammar that represents a binary tree of `EXP`.

$$\exp(a+b+c+\cdots) = \exp(a)\exp(b+c+\cdots) \quad (10)$$

For example, the expression

```
EXP(A00S1 + A01S1 COS  wt + B01S1 SIN  wt
          + A02S1 COS 2wt + B02S1 SIN 2wt)
```

is represented by the following binary tree (Fig. 3)

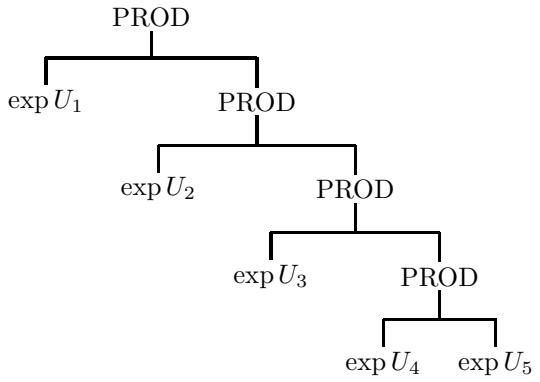

Fig. 3: Binary tree of exponential

with

U_1 = `A00S1`
U_2 = `A01S1 cos( wt)`
U_3 = `B01S1 sin( wt)`
U_4 = `A02S1 cos(2wt)`
U_5 = `B02S1 sin(2wt)`

The method used for `EXP` can be applied to `COS` and `SIN`, it is just a little bit more complicated because the trees for `COS` and `SIN` are mutually recursive.

A recursive application of the addition formula for cos (resp. sin), see Eq. (11) and Eq. (12), allows to define an abstract grammar that represents a binary tree of `COS` (resp. `SIN`). Note that the construction of the tree for `COS` involves the tree of `SIN`, and conversely.

$$\cos(a+b+c+\cdots) = \quad (11)$$
$$\cos(a)\cos(b+c+\cdots) - \sin(a)\sin(b+c+\cdots)$$
$$\sin(a+b+c+\cdots) = \quad (12)$$
$$\sin(a)\cos(b+c+\cdots) + \cos(a)\sin(b+c+\cdots)$$

For example

```
COS(A00S1 + A01S1 cos  wt + B01S1 sin  wt)
```

is represented by the binary tree shown in Fig. 4, with

U_1 = `A00S1`
U_2 = `A01S1 cos(wt)`
U_3 = `B01S1 sin(wt)`

Similary,

```
SIN(A00S1 + A01S1 cos  wt + B01S1 sin  wt)
```

is represented by the binary tree shown in Fig. 5, with

U_1 = `A00S1`
U_2 = `A01S1 cos(wt)`
U_3 = `B01S1 sin(wt)`

6 Parse the nonlinear function trees

The reduction of the nodes (`PROD`, `PLUS`, `MINUS`) have been previously explained. The reduction of the leafs is based on the associated Bessel series of each transcendental function (exponential, cosinus, sinus). Indeed, it is well known that $\exp(z\cos\theta)$ and $\exp(z\sin\theta)$, respectively $\cos(\cdot)$ and $\sin(\cdot)$, can be expressed in harmonic form [4].

These harmonic expansions are recalled in Eq. 13 to Eq. 18 for completeness, where $I_k(z)$ are the Bessel functions of the second kind and of integer order, and $J_k(z)$ are the Bessel functions of first kind and of integer order. The leaf `EXP(A00S1)`, respectively `COS(.)` and `SIN(.)`, is already in the right form.

$$\exp(z\cos\theta) = I_0(z) + 2\sum_{k=1}^{\infty} I_k(z)\cos(k\theta) \quad (13)$$

$$\exp(z\sin\theta) = I_0(z) + 2\sum_{k=1}^{\infty}(-)^k I_{2k}(z)\cos(2k\theta) + 2\sum_{k=1}^{\infty}(-)^k I_{2k+1}(z)\sin((2k+1)\theta) \quad (14)$$

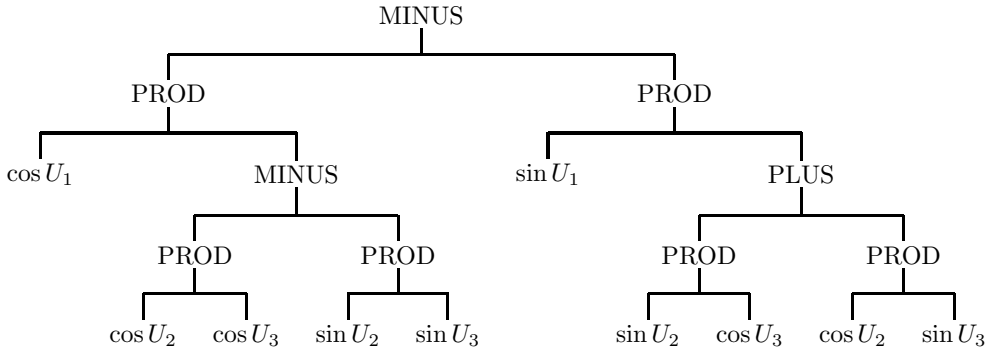

Fig. 4: Binary tree of cosinus

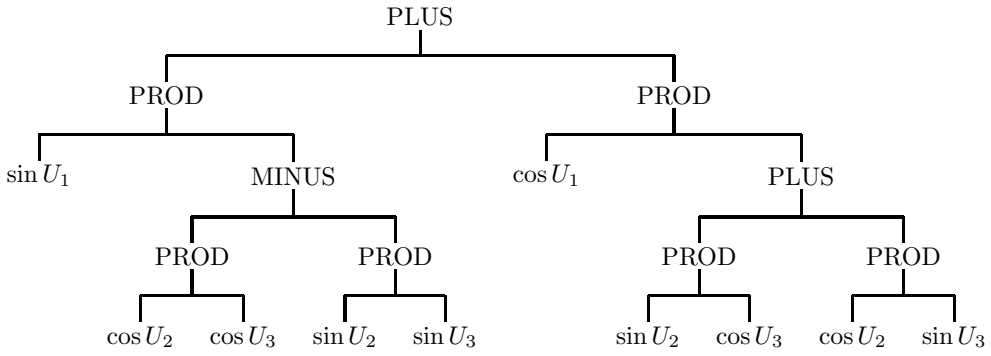

Fig. 5: Binary tree of sinus

$$\cos(z\cos\theta) = J_0(z) + 2\sum_{k=1}^{\infty}(-)^k J_{2k}(z)\cos(2k\theta) \quad (15)$$

$$\cos(z\sin\theta) = J_0(z) + 2\sum_{k=1}^{\infty} J_{2k}(z)\cos(2k\theta) \quad (16)$$

$$\sin(z\cos\theta) = J_0(z) + 2\sum_{k=1}^{\infty}(-)^k J_{2k+1}(z)\cos((2k+1)\theta) \quad (17)$$

$$\sin(z\sin\theta) = J_0(z) + 2\sum_{k=1}^{\infty} J_{2k+1}(z)\sin((2k+1)\theta) \quad (18)$$

7 Computation cost

In this section, we compare the efficiency of our method in terms of the number of arithmetic operations. Table 1 indicates the number of necessary additions and multiplications to transform a given nonlinear function in a series form. The so–called direct method is a straigthforward application of the well–known method recalled in section 2. The third column (proposed method) is relative to the symbolic method described in this paper.

	Direct		Proposed	
	Nb +	Nb ∗	Nb +	Nb ∗
$V(t)^2$	25	46	25	46
$V(t)^3$	98	314	105	209
$V(t)^4$	277	1129	243	504
$V(t)^5$	647	3269	453	949
$\exp(V(t))$	935	6594	427	609

Table 1: Computation cost

It should be pointed out that our method is not a simple calculation refinement that one can do without. As an example, the present method is necessary in order to treat any ODE involving an exponential function of the unknown. This is very common in electronics where semiconductor devices are modelled by an equation of the form $\exp(qV(t)/kT)$ [5].

8 Conclusion

It has been demonstrated how to replace, by symbolic computation, a nonlinear differential equation system by a system of nonlinear equations of Fourier coefficients whose solution is an approximation of the steady–state solution of the differential equation.

A solution based on abstract grammar and tree parsing to manage the large number of coefficients inherent to symbolic computation has been proposed. At the opposite of all other harmonic methods, the linear and the nonlinear parts of the differential equation are processed in an uniform way in the Fourier domain.

The numerical simulation times of the equations generated in the last step of this method is independent on the length of transients, because the symbolic harmonic method imposes the steady–state conditions by virtue of Fourier expansion of the unknowns.

No mention has been made about how to solve the system of nonlinear equations generated in the last step of the symbolic harmonic method. The resulting system is highly nonlinear and sparse. Efforts are currently made to develop specific and efficient numerical algorithms in this direction.

References:

[1] Kenneth S. Kundert, Jacob K. White and Alberto Sangiovanni–Vincentelli, Steady–State Methods for Simulating Analog and Microwave Circuits, Kluwer Academic Publishers, 2003

[2] Andrew W. Appel, Modern Compiler Implementation in ML, Cambridge University Press, 1998

[3] Emmanuel Chailloux and Pascal Manoury and Bruno Pagano., Developing Applications With Objective Caml, Online version (04/2005): http://caml.inria.fr/pub/docs/oreilly-book/

[4] Milton Abramowitz and Irene A. Stegun, Handbook of Mathematical Functions, Dover Publications, Inc., 1972

[5] G. Massobrio and P. Antognetti, *Semiconductor Device Modeling with SPICE*, MacGraw-Hill, Inc, second edition, 1993.

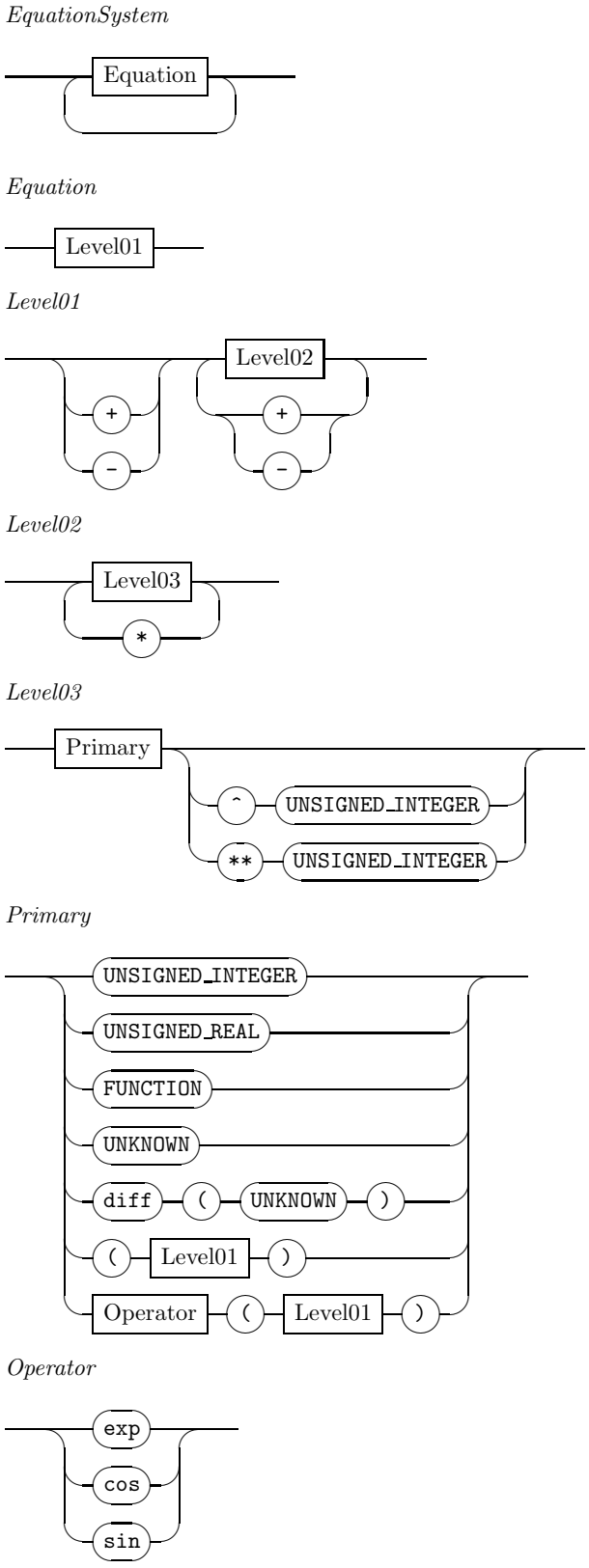

Fig. 6: Concrete grammar of ODE