Supporting CPS Modeling Through a New Method for Solving Complex Non-holomorphic Equations

Giulio Masetti[1,2], Simone Dutto[1], Silvano Chiaradonna[1] and Felicita Di Giandomenico[1]
[1]*ISTI-CNR, Pisa, Italy*
[2]*Department of Computer Science, University of Pisa, Italy*

Keywords: Generative Programming, Newton-raphson, Complex System of Equations, Non-holomorphic Functions, Wirtinger Derivatives.

Abstract: Modeling cyber-physical systems (CPSs) for assessment or design support purposes is a complex activity. Capturing all relevant physical, structural or behavioral aspects of the system at hand is a crucial task, which often implies representation of peculiar features/constraints through non-linear equations. Values that fulfill the constraints, described with a domain specific language, are obtained solving the equations through a properly developed solution tool. Only for a limited set of CPSs it is possible to find a straightforward strategy to design the software that solves the constraints equations. In the general case, instead, the modeler has to develop an ad-hoc artifact for each different system. This is the case of non-holomorphic but real analytic complex equations, adopted to represent system components with wave behaviors. In this paper, we present a new approach to develop a software for solving such complex equations following a generative programming strategy, based on Wirtinger derivatives within the Newton-Raphson method.

1 INTRODUCTION AND RELATED WORK

Modern cyber-physical systems (CPSs) are smart systems that include engineered interacting networks of physical and computational components (NIST, 2015).

Several layers of abstraction are commonly used to characterize a CPS: components level, network of physical components, network of computational components, network of interactions among physical and cyber components, etc, each one possibly described with a different formalism or domain specific language.

Focusing on a model-based design perspective, a comprehensive model of the system is typically assembled at the beginning of the system design process, to assist the designer in making choices on the system under development. Refinements of the initially defined model can be performed in later stages, e.g triggered by observations emerging during the implementation phase.

Models represent at a suitable detailed level (that depends on the objective of the analysis carried on through modeling) both the structure of the system and its behavior during lifetime, subject to the environmental context it operates in and related phenomena, such as faults.

In CPS, the modeling of control units requires a fairly detailed description of the physical component that has to be controlled, of the adopted control policies and of related fault-tolerance mechanisms, if any.

There exists a vast literature, in particular for embedded systems (Rizano et al., 2011), about architectural choices, dependability-related concepts, programming paradigms, etc, when dealing with model creation and model-guided software implementation phases. However, when defining the physical part of the system state within a model, it is common practice not to follow a general strategy but an ad-hoc sequence of steps. This results in additional work for the modeler, as well as request for higher expertise in the specific adopted formalisms.

Inside a model, the state of the controlled components is often represented via a set of constraints upon the physical elements of the controlled system. These constraints characterize the nature and evolution of the controlled system and are modeled using some sort of declarative formalism, typically algebraic equations. This implies that one essential aspect of the modeling process is the availability of a (semi-automatic) procedure to generate the software

Masetti, G., Dutto, S., Chiaradonna, S. and Giandomenico, F.
Supporting CPS Modeling Through a New Method for Solving Complex Non-holomorphic Equations.
DOI: 10.5220/0006752306800688
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 680-688
ISBN: 978-989-758-283-7

that is responsible, within the model, for the computation of the components state by numerically solving the given equations.

Depending on the characteristics of the constraints, general methods have been considered (Broyden, 1965), (Kelley, 1995), (Khalil, 2002), but when their conditions are not satisfied, specific methods (ad hoc, i.e., depending on the CPS studied) have to be found (Bejarano and Sánchez, 1989), (Avargel and Cohen, 2010).

In this paper, the focus is on physical components subject to state changes following a periodic motion governed by equations that are not sufficiently smooth (complex non-holomorphic) to allow straightforward implementation of standard equation solution methods. Physical components governed by this class of equations can be found in several real-world systems, such as:

- electrical circuits in Alternating Current, where the steady-state is represented by the Power Flow Equations (Wang et al., 2017);
- calibration of antennas, where Radio Interferometry Measurement Equations (Tasse, 2014) are employed;
- wave profile shaping, where convolution of phasors equations (Arrillaga, 2003) define the constructive/destructive superimposition of periodic waves within a (non)-linear medium.

In all the mentioned scenarios, the physical part of the component state that appears in the model is represented using complex numbers and resorting to non-smooth equations. As an help to modelers, the contribution of this paper is the definition and implementation of a new general solution method, able to compute solution for this class of equations.

Specifically, the new implemented method, called WNR, takes the constraint equations and calculates their solution. The mathematical details are briefly discussed and each step of the method is described providing a runnable MATLAB (MathWorks, 2017) function.

The rest of the paper has the following structure. After the description of the mathematical context in Section 2, the new method is discussed in Section 3. A simple but representative case study, adopted to illustrate how the method works, is depicted in Section 4. Finally, conclusions and directions for future work are briefly drawn in Section 5.

2 CONTEXT

The evolution of a CPS is modeled by states and transitions among states. Each state is commonly described as a list of state variables. Depending on their characteristics, the state variables belong to real numbers ($\mathbb{R}$) or, more generally, to complex numbers ($\mathbb{C}$). The state transitions of a system are ruled by theoretical relations and practical aspects, that are collected in a set of constraints, implemented with a list of equations in which some state variables are *known* while others are *unknown*.

Considering the array $\mathbf{z} = (z_1, z_2, \ldots, z_n) \in \mathbb{C}^n$ containing all unknowns, the system constraints are given by $\mathbf{f}(\mathbf{z}) = \mathbf{0} \in \mathbb{C}^m$, i.e., a shorter notation for the list of equations:

$$\begin{cases} f_1(z_1, z_2, \ldots, z_n) = 0 \\ f_2(z_1, z_2, \ldots, z_n) = 0 \\ \qquad\qquad \vdots \\ f_m(z_1, z_2, \ldots, z_n) = 0 \end{cases} \tag{1}$$

It is important to observe that, even when the system variables and constraints are described in $\mathbb{C}$, real formulations and implementations are preferred. This is because nowadays computer architectures work with real numbers. In fact, since they consider each complex number as a couple of real numbers, complex calculus involves more calculations and, consequently, it is less precise and slower with respect to the real one.

Depending on the characteristics of the functions f_i, a problem like (1) belongs to one of the two following classes: *linear* problems and *non-linear* problems. As can be seen in the following subsections, while the first type of problems can be easily studied, solution approaches exist only for a little part of the second ones.

2.1 Linear Problems

The problem (1) is called *linear* if it can be written as:

$$\begin{cases} a_{11}z_1 + a_{12}z_2 + \cdots + a_{1n}z_n = b_1 \\ a_{21}z_1 + a_{22}z_2 + \cdots + a_{2n}z_n = b_2 \\ \qquad\qquad \vdots \\ a_{m1}z_1 + a_{m2}z_2 + \cdots + a_{mn}z_n = b_n \end{cases} \tag{2}$$

It is possible to use a matricial form to collect, in a single contracted equation, the system (2). The result is given by:

$$\mathbf{A} \cdot \mathbf{z} = \mathbf{b} \tag{3}$$

where $\mathbf{A}$ is a matrix containing all the numbers a_{ij} and $\cdot$ is the matrix-vector product.

Provided that the equations in (2) are linearly independent, there are three possible cases:

1. if $m < n$, i.e., there are less equations than unknowns, the problem has infinite solutions;
2. if $m = n$, i.e., the number of equations is equal to the number of unknowns, and $\mathbf{A}$ is non-singular, the problem has exactly one solution;
3. if $m > n$, i.e., there are more equations than unknowns, the problem has no solutions.

If the problem admits at least one solution, it can be solved by considering the inverse of the matrix $\mathbf{A}$, because $\mathbf{z} = \mathbf{A}^{-1} \cdot \mathbf{b}$. The implementation of the inverse matrix computation has been widely studied and optimized, and can be found in most of mathematical software, e.g. MATLAB (MathWorks, 2017).

2.2 Non-linear Problems

All problems like (1) that can not be written as in (2) are classified as non-linear problems. Rarely direct approaches can solve this kind of problems and sometimes it is impossible to know if a solution exists. Fortunately, there exist iterative approaches that allow to find approximations of one of the problem solutions.

When in (1) $m = n$, one of the most used and implemented iterative solution approaches is the *Newton-Raphson* method (NR) (Kelley, 1995). In its steps, starting from an initial guess of $\mathbf{z}$, the new approximation is determined using the *Jacobian* $\mathbf{J}$ of the initial non-linear problem. However, since the Jacobian of a set of real (or, respectively, complex) equations is the matrix containing all their real (complex) derivatives, in order to use NR, the problem has to contain all derivable (holomorphic, i.e., complex derivable) equations (Kreutz-Delgado, 2009).

An implementation of NR with MATLAB (MathWorks, 2017) can be given by:

```
function [z, conv] = NR(z, par, max_it, tol)
conv = 0; i = 0;           % initialize
F = feval(z, par);         % evaluate f(z)
normF = norm(F, inf);
if normF < tol             % check tolerance
  conv = 1; exit;
end
while(conv==0 && i<max_it) % do NR iterations
  i = i + 1;               % update it.counter
  J = jeval(z, F, par);    % evaluate J
  dz = -(J \ F);           % calc. update step
  z = z + dz;              % update z
  F = feval(z, par);       % evaluate f(z)
  normF = norm(F, inf);
  if normF < tol           % check convergence
      conv = 1;
  end
end
```

where the inputs are: (i) the initial guess `z`, (ii) a list of parameters `par` containing all known values, (iii) the maximum number of iterations `max_it` and (iv) the tolerance `tol`. The evaluation of $\mathbf{f}$ and $\mathbf{J}$ is performed by the functions `feval` and `jeval`, respectively. The functions `feval` and `jeval` are defined and implemented by the modeler. `feval` has inputs `z` and `par`; `jeval` has inputs `z`, `par` and `F`, i.e., the result of the current evaluation of $\mathbf{f}$. The definition of `jeval` can include built-in automatic derivative operators provided by MATLAB. The solution `dz = -(J \ F)` of the linear system $\mathbf{J} \cdot \mathbf{dz} = -\mathbf{f}(\mathbf{z})$ determined by the Jacobian is used to approximate $\mathbf{z}$ (`z = z + dz`).

Despite NR is a powerful and fast method, for some problems it does not converge (`conv = 0`) and hence the solution cannot be found.

When the functions f_i of $\mathbf{f}$ are not complex derivable, and then NR is not applicable, solutions can be found only using an ad-hoc method.

3 THE NEW APPROACH: WNR

Our aim is to improve NR in order to obtain a method that can find solutions when classic NR fails or can not be used. The proposed method is called Wirtinger Newton Raphson (WNR) because of the particular derivatives used. WNR applies to systems described by real derivable complex equations that can be holomorphic or non-holomorphic.

In fact, each function $f_i : \mathbb{C}^n \to \mathbb{C}$ in $\mathbf{f}$ can be seen as $\tilde{f}_i : \mathbb{R}^{2n} \to \mathbb{C}$ when the cartesian coordinates $z_j = x_j + \imath y_j$ are considered. If all $\tilde{f}_i$ are derivable with respect to x_j and y_j (as real variables), then the following method, here implemented in MATLAB, can be used:

```
function [z, conv] = WNR(z, par, max_it, tol)
conv = 0; i = 0;           % initialize
F = feval(z, par);         % evaluate f(z)
normF = norm(F, inf);
if normF < tol             % check tolerance
  conv = 1; exit;
end
while(conv==0 && i<max_it) % do WNR iterations
  i = i + 1;               % update it.counter
  G = wfunc(F);            % NEW function
  J = wjeval(z, F, par);   % NEW Jacobian
  dw = -(J \ G);           % NEW update step
  dz = wstep(dw);          % calc. update step
  z = z + dz;              % update z
  F = feval(z, par);       % evaluate f(z)
  normF = norm(F, inf);
  if normF < tol           % check convergence
      conv = 1;
  end
end
```

The theoretical steps behind the new functions `wfunc`, `wjeval` and `wstep` are described in the Subsection 3.1, while their implementation in MATLAB is presented in Subsection 3.2.

3.1 Mathematical Process

After writing the functions $\tilde{f}_i : \mathbb{R}^{2n} \to \mathbb{C}^n$ by considering the complex variables in f_i as $z_j = x_j + \imath y_j$, if all the $\tilde{f}_i$ are real derivable, a new kind of complex derivatives can be defined. They are called *Wirtinger derivatives* (Kreutz-Delgado, 2009) and are the following partial differential operators:

$$\begin{aligned} \frac{\partial^W f_i}{\partial z_j} &= \frac{1}{2}\left(\frac{\partial \tilde{f}_i}{\partial x_j} - \imath \frac{\partial \tilde{f}_i}{\partial y_j}\right) \\ \frac{\partial^W f_i}{\partial \bar{z}_j} &= \frac{1}{2}\left(\frac{\partial \tilde{f}_i}{\partial x_j} + \imath \frac{\partial \tilde{f}_i}{\partial y_j}\right) \end{aligned} \tag{4}$$

where $\bar{z}_j = x_j - \imath y_j$ is the complex conjugate of z_j.

Despite their formal definition, Wirtinger derivatives of f_i can be calculated as complex derivatives with respect to z_j (or, respectively, $\bar{z}_j$) while considering $\bar{z}_j$ (z_j) constant, so that the passage to $\tilde{f}_i$ is merely formal.

The *Wirtinger Jacobian* (Kreutz-Delgado, 2009) of $\mathbf{f}$ is defined analogously to the classic complex one. However, since there are $2n$ Wirtinger derivatives for each of the n functions f_i, considering only the Wirtinger derivatives of $\mathbf{f}$ returns a rectangular matrix. Hence, in order to make square the Wirtinger Jacobian, the Wirtinger derivatives of $\bar{\mathbf{f}}$, i.e., the complex conjugate of $\mathbf{f}$, are also considered. The resulting structure of the Wirtinger Jacobian is given by:

$$\mathbf{J}^W = \left(\begin{array}{c|c} \frac{\partial^W \mathbf{f}}{\partial \mathbf{z}} & \frac{\partial^W \mathbf{f}}{\partial \bar{\mathbf{z}}} \\ \hline \frac{\partial^W \bar{\mathbf{f}}}{\partial \mathbf{z}} & \frac{\partial^W \bar{\mathbf{f}}}{\partial \bar{\mathbf{z}}} \end{array}\right) \in \mathbb{C}^{2n,2n} \tag{5}$$

where the entries of its submatrices are:

$$\begin{aligned} \left(\frac{\partial^W \mathbf{f}}{\partial \mathbf{z}}\right)_{ij} &= \frac{\partial^W f_i}{\partial z_j} \\ \left(\frac{\partial^W \mathbf{f}}{\partial \bar{\mathbf{z}}}\right)_{ij} &= \frac{\partial^W f_i}{\partial \bar{z}_j} \\ \left(\frac{\partial^W \bar{\mathbf{f}}}{\partial \mathbf{z}}\right)_{ij} &= \frac{\partial^W \bar{f}_i}{\partial z_j} = \overline{\left(\frac{\partial^W f_i}{\partial \bar{z}_j}\right)} \\ \left(\frac{\partial^W \bar{\mathbf{f}}}{\partial \bar{\mathbf{z}}}\right)_{ij} &= \frac{\partial^W \bar{f}_i}{\partial \bar{z}_j} = \overline{\left(\frac{\partial^W f_i}{\partial z_j}\right)} \end{aligned} \tag{6}$$

It is important to observe that the Wirtinger derivatives of $\bar{\mathbf{f}}$ can be calculated as complex conjugates of the Wirtinger derivatives of $\mathbf{f}$, so that the number of computations needed can be halved.

The innovative idea in WNR is to use in each step of NR the Wirtinger derivatives and Wirtinger Jacobian instead of their classic derivatives and Jacobian. Consequently, instead of the linear system $\mathbf{J} \cdot \mathbf{dz} = -\mathbf{f}(\mathbf{z})$, the new system solved in each step of WNR is:

$$\mathbf{J}^W \cdot \begin{pmatrix} \mathbf{dz} \\ \overline{\mathbf{dz}} \end{pmatrix} = - \begin{pmatrix} \mathbf{f}(\mathbf{z}) \\ \overline{\mathbf{f}(\mathbf{z})} \end{pmatrix} \tag{7}$$

Moreover, exploiting the particular structure of $\mathbf{J}^W$, we manage to obtain a new formulation of the linear system based on real numbers (7). Thanks to this real formulation the new approach can be easily implemented and the results can be achieved with less calculations than using the complex version. Here are the steps to get the final formulation used in WNR:

1. following by (6), $\mathbf{J}^W$ can be written as:

$$\mathbf{J}^W = \left(\begin{array}{c|c} \frac{\partial^W \mathbf{f}}{\partial \mathbf{z}} & \frac{\partial^W \mathbf{f}}{\partial \bar{\mathbf{z}}} \\ \hline \overline{\left(\frac{\partial^W \mathbf{f}}{\partial \bar{\mathbf{z}}}\right)} & \overline{\left(\frac{\partial^W \mathbf{f}}{\partial \mathbf{z}}\right)} \end{array}\right) = \left(\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \overline{\mathbf{B}} & \overline{\mathbf{A}} \end{array}\right) \tag{8}$$

2. if a matrix presents a structure like the one described in (8), it can be transformed into a *centrohermitian* matrix (Lee, 1980), defined as:

$$\mathbf{M} \in \mathbb{C}^{m,m} | M_{ij} = \overline{M}_{m+1-i,m+1-j}, \text{ for } 1 \le i,j \le m \tag{9}$$

The transformation passes through the indexes permutation:

$$\pi : i \mapsto 3n+1-i \quad \text{for} \quad n < i \le 2n \tag{10}$$

that inverts the order of the last n rows or columns. In order to transform $\mathbf{J}^W$ into a centrohermitian matrix, π has to be applied to both its rows and columns. The best way to do that is considering the matrix associated to the permutation π, given by:

$$\mathbf{P}_\pi = \left(\begin{array}{c|c} \mathbb{I}_n & \mathbf{0}_n \\ \hline \mathbf{0}_n & \mathbb{J}_n \end{array}\right) \in \mathbb{R}^{2n,2n} \tag{11}$$

where $\mathbb{I}_n$ and $\mathbb{J}_n$ are the *identity* and *exchange* matrix, respectively, defined as follows:

$$\mathbb{I}_n = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}, \quad \mathbb{J}_n = \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix} \in \mathbb{R}^{n,n} \tag{12}$$

Hence, the centrohermitian transformation of $\mathbf{J}^W$ is obtained by left and right multiplying $\mathbf{J}^W$ by the permutation matrix $\mathbf{P}_\pi$, so that:

$$\mathbf{P}_\pi \left(\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \overline{\mathbf{B}} & \overline{\mathbf{A}} \end{array}\right) \mathbf{P}_\pi = \left(\begin{array}{c|c} \mathbf{A} & \mathbf{B}\mathbb{J}_n \\ \hline \mathbb{J}_n\overline{\mathbf{B}} & \mathbb{J}_n\overline{\mathbf{A}}\mathbb{J}_n \end{array}\right) \tag{13}$$

that is a matrix satisfying (2) and, consequently, a centrohermitian matrix;

3. every centrohermitian matrix $\mathbf{M} \in \mathbb{C}^{m,m}$ is similar to a real matrix (Lee, 1980), i.e., there exists a matrix $\mathbf{Q} \in \mathbb{C}^{m,m}$ that is *unitary* ($\mathbf{Q}^{-1} = \overline{\mathbf{Q}^t}$) and for which $\overline{\mathbf{Q}^t}\mathbf{M}\mathbf{Q} \in \mathbb{R}^{m,m}$.
   Therefore, it is possible to find a real version of the centrohermitian Wirtinger Jacobian obtained in (13). In fact, when considering the unitary matrix given by:
$$\mathbf{Q} = \frac{1}{\sqrt{2}} \left(\begin{array}{c|c} \mathbb{I}_n & \imath\mathbb{I}_n \\ \hline \mathbb{J}_n & -\imath\mathbb{J}_n \end{array} \right) \in \mathbb{C}^{2n,2n} \tag{14}$$
   the complex matrix described in (13) is similar to the following real matrix:
$$\overline{\mathbf{Q}^t} \left(\begin{array}{c|c} \mathbf{A} & \mathbf{B}\mathbb{J}_n \\ \hline \mathbb{J}_n\overline{\mathbf{B}} & \mathbb{J}_n\overline{\mathbf{A}}\mathbb{J}_n \end{array} \right) \mathbf{Q} = \left(\begin{array}{c|c} \Re(\mathbf{A}+\mathbf{B}) & \Im(\mathbf{B}-\mathbf{A}) \\ \hline \Im(\mathbf{A}+\mathbf{B}) & \Re(\mathbf{A}-\mathbf{B}) \end{array} \right) \tag{15}$$
   where $\Re(\dots)$ and $\Im(\dots)$ indicate respectively the real and imaginary part of a complex number;
4. the formulation based on real numbers of the system (7) solved in each step of WNR can be obtained by the transformations of $\mathbf{J}^W$ described in (13) and (3).
   Starting from (7) written as:
$$\left(\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \overline{\mathbf{B}} & \overline{\mathbf{A}} \end{array} \right) \cdot \left(\begin{array}{c} \mathbf{dz} \\ \hline \overline{\mathbf{dz}} \end{array} \right) = - \left(\begin{array}{c} \mathbf{F} \\ \hline \overline{\mathbf{F}} \end{array} \right) \tag{16}$$
   where $\mathbf{F} = \mathbf{f}(\mathbf{z})$, applying (13) to (16) gives:
$$\left(\begin{array}{c|c} \mathbf{A} & \mathbf{B}\mathbb{J}_n \\ \hline \mathbb{J}_n\overline{\mathbf{B}} & \mathbb{J}_n\overline{\mathbf{A}}\mathbb{J}_n \end{array} \right) \cdot \left(\begin{array}{c} \mathbf{dz} \\ \hline \mathbb{J}_n\overline{\mathbf{dz}} \end{array} \right) = - \left(\begin{array}{c} \mathbf{F} \\ \hline \mathbb{J}_n\overline{\mathbf{F}} \end{array} \right) \tag{17}$$
   Finally, the similitude introduced in (3) applied to (4) returns:
$$\left(\begin{array}{c|c} \Re(\mathbf{A}+\mathbf{B}) & \Im(\mathbf{B}-\mathbf{A}) \\ \hline \Im(\mathbf{A}+\mathbf{B}) & \Re(\mathbf{A}-\mathbf{B}) \end{array} \right) \cdot \mathbf{dw} = -\mathbf{G} \tag{18}$$
   where:
$$\mathbf{dw} = \left(\begin{array}{c} \Re(\mathbf{dz}) \\ \Im(\mathbf{dz}) \end{array} \right), \quad \mathbf{G} = \left(\begin{array}{c} \Re(\mathbf{F}) \\ \Im(\mathbf{F}) \end{array} \right) \tag{19}$$

Hence, (18) can be used as real formulation of the linear system in each step of WNR that, consequently, can be described using only real quantities.

3.2 MATLAB Functions for WNR

The modeler can ignore the mathematical details presented in Subsection 3.1 since they are all collected in the functions `wfunc`, `wjeval` and `wstep` of `WNR`. In fact, these functions can be implemented directly using the results of Section 3.1 and in particular the matricial equation (18).

Their descriptions and MATLAB implementations are given by:

- `wfunc`: this function takes the current evaluation `F` of **f** and returns the vector `G` that appears in the right-hand side of (18) and that is described in (19), so that:

```
function G = wfunc(F)
G = [real(F);
     imag(F) ];
```

- `wjeval`: calling `J` the real Wirtinger Jacobian defined in (3) and used in (18) , `wjeval` is defined as:

```
function J = wjeval(z, F, par)
n = length(z);
W = wJ(z,F,par);
A = W(1:n, 1:n);
B = W(1:n, n+1:2*n);
J = [real(A+B), imag(B-A);
     imag(A+B), real(A-B) ];
```

  The function `wJ` evaluates the Wirtinger Jacobian by computing the Wirtinger derivatives, that can be implemented in two different ways:
  (i) by using the definition (4) with MATLAB built-in real derivatives, after writing the variables with their real and imaginary parts;
  (ii) by exploiting built-in complex derivatives with respect to the variables while considering their conjugates constant and vice versa;

- `wstep`: this function converts back to `dz` the result `dw = -(J \ G)` of the linear system (18), and is given by:

```
function dz = wstep(dw)
n = length(dw)/2;
dz = dw(1:n) + 1i*dw(n+1:2*n);
```

  This is because **dw**, as described in (19), contains $\Re(\mathbf{dz})$ and $\Im(\mathbf{dz})$.

In Figure 1 the possible choices of the modeler are schematically presented. It is important to observe that, when NR or WNR can be used, the modeler has only to define the functions that evaluate **f** and its (classical or Wirtinger) derivatives. Conversely, if the modeler chooses or is obliged to use an ad hoc method, a specific formulation for the constraints has to be found and the corresponding ad-hoc solving procedure has to be developed and implemented.

4 CASE STUDY

The CPS adopted as case study to illustrate the proposed method is presented in the following. In order to deal with a simple, and at the same time general, scenario, the focus here is only on the cyber part of

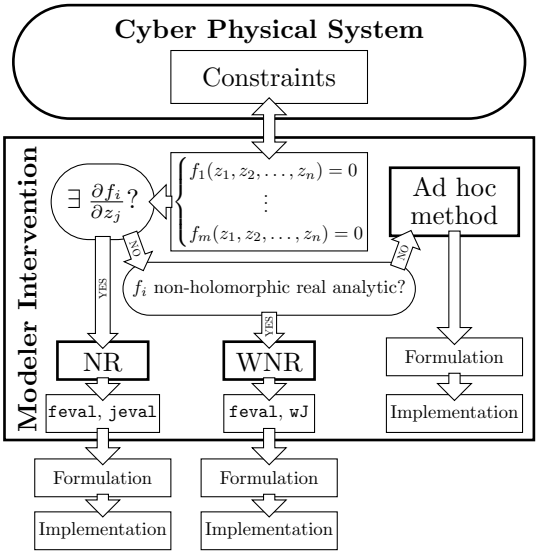

Figure 1: Diagram describing how the the modeler is supported by the proposed WNR approach, compared with NR and ad hoc methods, in the overall process of modeling and solving constraints of CPSs.

the system, abstracting away the details of the physical part. The high level logical architecture, depicted in Figure 2, includes a sensor, the control unit and an actuator.

The sensor collects those data coming from the environment that show periodic behavior and it is assumed that the control unit receives a signal described as a function of time t of the form:

$$u(t) = \sum_{h=0}^{n-1} u_h sin(h\omega t + \phi_h) = \Im\left(\sum_{h=0}^{n-1} P_{u,h} e^{\imath h\omega t}\right) \quad (20)$$

where $P_{u,h} = u_h e^{\imath\phi_h} \in \mathbb{C}$ is the phasor of module u_h and phase ϕ_h, as presented in (Arrillaga, 2003). Suppose that $u(t)$ is related but not equal to the actual state of the controlled component, that is described by a signal expressed as:

$$v(t) = \sum_{h=0}^{n-1} v_h sin(h\omega t + \psi_h) = \Im\left(\sum_{h=0}^{n-1} P_{v,h} e^{\imath h\omega t}\right) \quad (21)$$

However, u_h and ϕ_h can change their values and the aim of the control unit is to compute an estimation $\tilde{v}(t)$ of $v(t)$, i.e., new values of $P_{v,h} = v_h e^{\imath\psi_h}$. If $\tilde{v}(t)$ is out of bounds, the control unit has to bring the system back in a safe state sending instructions to the actuator.

This case study is commonly encountered when dealing with non-linear physical systems whose non-linearities have to follow a prescribed behavior. The control unit consists of an iterative process in which, whenever the input signal is found to be unsafe, a new

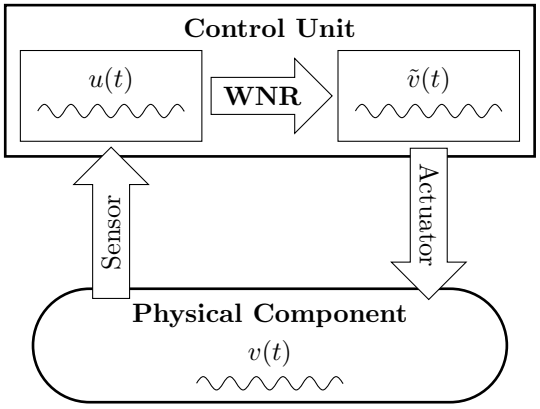

Figure 2: Simple diagram describing the case study.

setting of the parameters that describe $v(t)$ has to be found.

For the case study, the input signal $u(t)$ and the unknown state $v(t)$ are related by:

$$u(t) = v(t)v(t) \quad (22)$$

Since the imaginary part of a complex number z can be calculated as:

$$\Im(z) = -\frac{\imath}{2}(z - \bar{z}) \quad (23)$$

the expressions in (20) and (21) can be written as:

$$u(t) = -\frac{\imath}{2}\left(\sum_{h=0}^{n-1} P_{u,h} e^{\imath h\omega t} - \sum_{h=0}^{n-1} \overline{P}_{u,h} e^{-\imath h\omega t}\right) \quad (24)$$

$$v(t) = -\frac{\imath}{2}\left(\sum_{h=0}^{n-1} P_{v,h} e^{\imath h\omega t} - \sum_{h=0}^{n-1} \overline{P}_{v,h} e^{-\imath h\omega t}\right) \quad (25)$$

Using (24) and (25) in (22), the phasors of $u(t)$ in relation to those of $v(t)$ are given by:

$$P_{u,h} = \frac{\imath}{2}\sum_{k=0}^{n-1}\Big[\mathbb{1}_{k\geq h}\left(P_{v,k}\overline{P}_{v,k-h} + \mathbb{1}_{h>0}P_{v,k}\overline{P}_{v,k-h}\right) - \mathbb{1}_{k\leq h}P_{v,k}P_{v,h-k}\Big] \quad (26)$$

where:

$$\mathbb{1}_{a\lesseqgtr b} = \begin{cases} 1 & \text{if } a \lesseqgtr b \\ 0 & \text{elsewhere} \end{cases} \quad (27)$$

For $h = 0, \ldots, n-1$, the n variables z_h of $\mathbf{z}$ are the phasors $P_{v,h}$, while the n functions f_h of $\mathbf{f}$ are:

$$f_h(\mathbf{z}) = f_h(P_{v,0}, \ldots, P_{v,n-1}) = P_{u,h} - u_h e^{\imath\phi_h} \quad (28)$$

where u_h, ϕ_h are constants and $P_{u,h}$ are given by (26).

The constraints $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ originated assembly (26) and (28), are non-linear non-holomorphic complex equations.

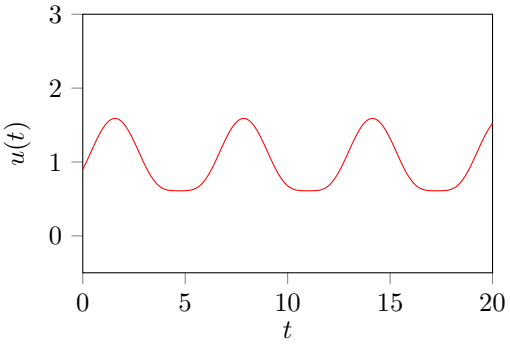

Figure 3: Input signal $u(t)$ received from the sensor.

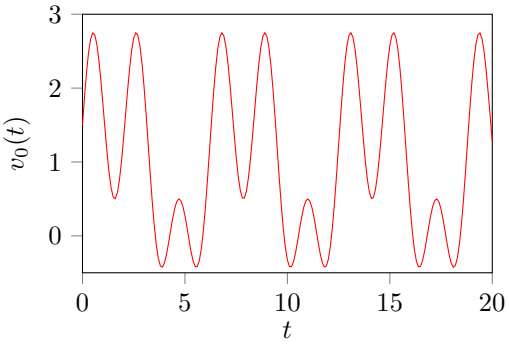

Figure 4: Initial guess of the controlled component signal.

Notice that equation (26) can be directly translated in MATLAB so that the implementation of WNR described in Section 3 can be used.

In particular, as said in Subsection 3.2 and depicted in Figure 1, the modeler can ignore all the mathematical passages and use directly the MATLAB implementation presented. The only functions to be determined to obtain the working solver `WNR` are `feval` and `wJ`. `feval` can be directly obtained from (26), so that the resulting implementation is given by:

```
function F = feval(z, par)
n = length(par);
Pu = par;    % par = [P_{u,0};...;P_{u,n-1}]
Pv = z;      % z = [P_{v,0};...;P_{v,n-1}]
F = zeros(n,1);
for h = 0:n-1
  for k = 0:n-1
    if (h <= k)
      F(h+1) += Pv(k+1)*conj(Pv(k-h+1));
      if (0 < h)
        F(h+1) += Pv(k+1)*conj(Pv(k-h+1));
      end
    end
    if (k <= h)
      F(h+1) -= Pv(k+1)*Pv(h-k+1);
    end
  end
end
F = .5*i*F;
F = F - Pu;
end
```

The implementation of `wJ` needs to calculate Wirtinger derivatives of $\mathbf{f}$. As said in Subsection 3.1 Wirtinger derivatives can be computed by simply deriving the function $\mathbf{f}$ with respect to the variables $z_h = P_{v,h}$ while considering their conjugates constant and with respect to the variables $\overline{z}_h = \overline{P}_{v,h}$ while considering $P_{v,h}$ constant. Starting from (26) and following these instructions, the results are given by:

$$\frac{\partial^w P_{u,h}}{\partial P_{v,k}} = \frac{\imath}{2}\left[\mathbb{1}_{k\geq h}\left(\overline{P}_{v,k-h} + \mathbb{1}_{h>0}\overline{P}_{v,k-h}\right) - \mathbb{1}_{k\leq h} 2P_{v,h-k}\right]$$

$$\frac{\partial^w P_{u,h}}{\partial \overline{P}_{v,k}} = \frac{\imath}{2}\mathbb{1}_{k\leq n-1-h}\left(P_{v,k+h} + \mathbb{1}_{h>0}P_{v,k+h}\right) \tag{29}$$

so that the implementation of `wJ` in MATLAB is:

```
function W = wJ(z,F,par)
n = length(par);
Pu = par;    % par = [P_{u,0};...;P_{u,n-1}]
Pv = z;      % z = [P_{v,0};...;P_{v,n-1}]
W = zeros(n,2*n);
for h=0:n-1
  for k=0:n-1
    if (h <= k)
      W(h+1,k+1) += conj(Pv(k-h+1));
      if (0 < h)
        W(h+1,k+1) += conj(Pv(k-h+1));
      end
    end
    if (k <= h)
      W(h+1,k+1) -= 2*Pv(h-k+1);
    end
    if (k <= n-1-h)
      W(h+1,n+k+1) += Pv(k+h+1);
      if (0 < h)
        W(h+1,n+k+1) += Pv(k+h+1);
      end
    end
  end
end
W = .5*i*W;
end
```

In order to exercise the case study in a concrete scenario, we consider the event in which $n = 4$ and the input signal $u(t)$ received from the sensor, as shown in Figure 3, is given by:

$$\begin{aligned} P_{u,0} &= 1\imath\,, \quad P_{u,1} = 0.5\,, \\ P_{u,2} &= -0.1\imath\,, \quad P_{u,3} = 0.01 \end{aligned} \tag{30}$$

The initial guess $v_0(t)$ of the controlled component signal is depicted in Figure 4 and characterized by:

$$\begin{aligned} P_{v_0,0} &= 1\imath\,, \quad P_{v_0,1} = 1\,, \\ P_{v_0,2} &= 0.5\imath\,, \quad P_{v_0,3} = 1 \end{aligned} \tag{31}$$

Starting from $v_0(t)$ an estimation $\tilde{v}(t)$ of the component signal has to be computed. Running `WNR` as described in Section 3, with an imposed tolerance of

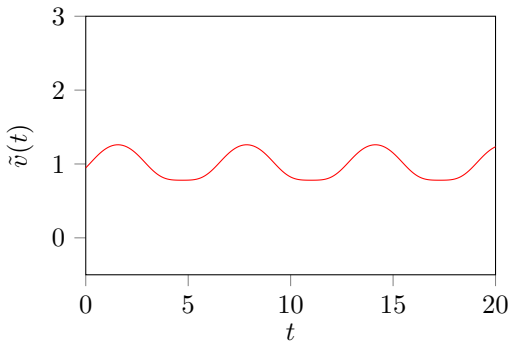

Figure 5: Estimation of $v(t)$ obtained with WNR.

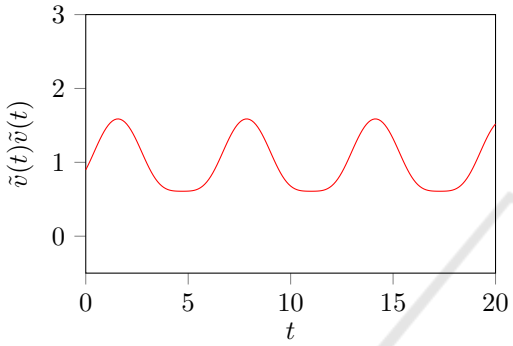

Figure 6: Plot showing that $\tilde{v}(t)\tilde{v}(t) = u(t)$.

10^{-6}, we obtain the new estimation $\tilde{v}(t)$ of $v(t)$, with parameters:

$$P_{v,0} = 0.98394\imath\,, \quad P_{v,1} = 0.24967\,, \\ P_{v,2} = -0.03621\imath\,, \quad P_{v,3} = 0.00968 \tag{32}$$

so that the resulting signal is depicted in Figure 5.

As correctness proof, Figure 6 shows that the product $\tilde{v}(t)\tilde{v}(t)$ is a signal equal to $u(t)$. WNR reaches the correct estimation $\tilde{v}(t)$ in 8 steps, as shown in Figure 7, that describes the behavior until convergence of the error of each step of WNR, given by the infinite norm of $\mathbf{f}(\mathbf{z})$.

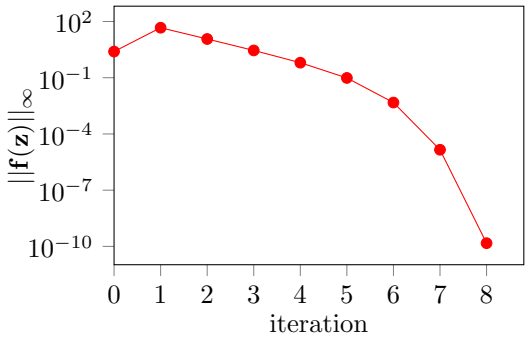

Figure 7: Plot in logaritmic scale of the descending error in WNR iterations.

5 CONCLUSIONS AND FUTURE WORK

When numerically computing the physical part of a CPS state, it could be needed to solve complex non-holomorphic equations. This paper addressed this class of equations and developed: i) a novel semiautomatic procedure to transform their original formulation into a new and more tractable form, and ii) a solution method for this new formulation, implemented in MATLAB.

The major principles the new solution is based on are: Wirtinger calculus application and centrohermitian structure exploitation. The idea is to define a general procedure that can be followed by the modeler, without concentrating on mathematical details except for the definition of function $\mathbf{f}$ and derivatives computation, that can in many cases be performed symbolically by MATLAB itself.

The advantage of the new proposed approach is that the evolution of the states of the modeled CPS can be described by changing the values of the parameters of $\mathbf{f}$, thus avoiding the complicated redefinition of the solver.

Extensions of the presented study are foreseen in several directions, including:

- performance of the new solution method has to be investigated in comparison with other, ad-hoc, strategies;
- automatic symbolic derivatives in the context of Wirtinger calculus is a promising line of research and can further automatize the approach presented in this paper;
- the centrohermitian symmetry is not the only structure that the Wirtinger Jacobian matrix presents, thus a careful investigation of the properties of this matrix can lead to improved performance of the proposed approach;
- the new solver can be customized to fulfill particular implementation needs, such as a reduced memory consumption. For example, the discussed case study offers a sparse Jacobian matrix with a structure that can be further exploited using the native sparse representation capabilities of MATLAB.

REFERENCES

Arrillaga, J. (2003). *Power system harmonics*. J. Wiley & Sons, West Sussex, England Hoboken, NJ.

Avargel, Y. and Cohen, I. (2010). Modeling and identification of nonlinear systems in the short-time fourier

transform domain. *IEEE Transactions on Signal Processing*, 58(1):291–304.

Bejarano, J. D. and Sánchez, A. M. (1989). Generalized fourier transforms for nonlinear systems. *Journal of Mathematical Physics*, 30(8):1871–1876.

Broyden, C. G. (1965). A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92):577–593.

Kelley, C. (1995). *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics.

Khalil, H. (2002). *Nonlinear Systems*. Pearson Education. Prentice Hall.

Kreutz-Delgado, K. (2009). The Complex Gradient Operator and the CR-Calculus. *ArXiv e-prints*.

Lee, A. (1980). Centrohermitian and skew-centrohermitian matrices. *Linear Algebra and its Applications*, 29(Supplement C):205 – 210. Special Volume Dedicated to Alson S. Householder.

MathWorks (2017). *MATLAB version 9.3.0.713579 (R2017b)*. The Mathworks, Inc., Natick, Massachusetts.

NIST (2015). *Framework for Cyber-Physical Systems*.

Rizano, T., Passerone, R., Macii, D., and Palopoli, L. (2011). Model-based design of embedded control software for hybrid vehicles. In *2011 6th IEEE International Symposium on Industrial and Embedded Systems*, pages 75–78.

Tasse, C. (2014). Applying wirtinger derivatives to the radio interferometry calibration problem.

Wang, Z., Cui, B., and Wang, J. (2017). A necessary condition for power flow insolvability in power distribution systems with distributed generators. *IEEE Transactions on Power Systems*, 32(2):1440–1450.