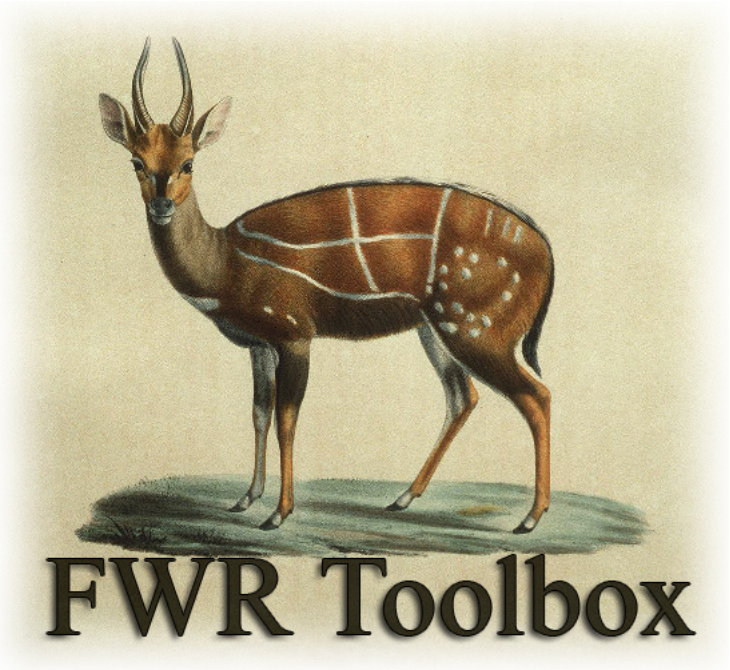

Finite Wordlength Realizations Toolbox User's Guide

v 0.99

http://fwrtoolbox.gforge.inria.fr/

Contents

Abstract

The FWR Toolbox is a MATLAB toolbox used to analyze the Finite Word Length effects of linear time-invariant digital filters/controllers implementations. When digital filter/controller are implemented in computing machines (micro-controller, DSP, FPGA, etc.) with finite precision, a degradation occurs. It comes from:

- the addition of roundoff noise after each arithmetic operation,
- the rounding of the embedded parameters.

The FWR Toolbox provides a general description of any possible realization (direct form, state-space, δ or shift operator, observer-state-feedback, cascad decomposition, etc...) in a form that allows a straightforward analysis of the FWL effects. Several tools are provided to compute open-loop/closed-loop sensitivity, related stability measure, roundoff noise gain, ..., and to find *optimal* realizations, according to these criteria. It is also possible to generate fixed-point code (C code, VHDL, ...).

License

Authors

- Thibault HILAIRE (main contributor):
  thibault.hilaire@nt.tuwien.ac.at, Institute of Communications and Radio-Frequency Engineering, Vienna University of Technology, Austria.
- Yu FENG:
  yu.feng@emn.fr, École des Mines de Nantes, France.

Acknowledgment

This toolbox is based on

- the work done during Thibault Hilaire's PhD thesis, (Philippe Chevrel and Yvon Trinquet as advisors) with the IRCCyN Lab (*Institut de Recherche en Communication et en Cybernétique de Nantes*, France) and PSA Peugeot-Citroën
- the work done at IRISA Lab (*nstitut de Recherche en Informatique et Systèmes Aléatoires*, France) in the Cairn (ex-R2D2) project.
- the work done by Yu Feng during his Master's thesis at IRCCyN Lab.

This toolbox is now maintained (and still enriched) during my postdoc position at the Institute of Communications and Radio-Frequency Engineering, Vienna University of Technology, Austria.

The logo comes from http://environnement.ecoles.free.fr.

The authors would like to thank people who help us for this toolbox, directly or indirectly:

- Philippe Chevrel – IRCCyN Lab and École des Mines de Nantes, France
- Olivier Sentieys, Daniel Ménard – INRIA-IRISA Cairn Team and Université de Rennes 1, France
- James Whidborne – Department of Aerospace Science, Cranfield University, UK

1 Installation

Instructions for installing the FWR Toolbox can be found in the section entitled *Installing Toolboxes* in the computer specific section of the MATLAB user's guide.
We recommend that you locate the files from this toolbox in a directory named `FWRToolbox` of the toolbox directory and add it in the MATLAB path.

This toolbox may require the *Adaptative Simulating Annealing* (`ASA`) software (`http://www.ingber.com/`) and its Matlab's gateway (`Asamin` `http://www.econ.ubc.ca/ssakata/public_html/software`) in order to find optimal realization with the global optimization algorithm ASA.

The latest version of the toolbox can be download with the anonymous Subversion access with the following command (or with your favorite SVN client):

```
svn checkout svn://scm.gforge.inria.fr/svn/fwrtoolbox
```

Warning : the `Control System Toolbox` is required in order to use the FWR Toolbox.

2 Introduction to FWL problem

When digital filters are implemented, they are implemented with finite precision due to the finite wordlength (FWL) of the representation of numbers within the computing machine. There are two FWL effects. The first is the addition of noise into the system resulting from the rounding of variables before and after each arithmetic operation - the "round-off noise". The second is the degradation in the performance and/or the stability resulting from rounding of the filter coefficients - the "coefficient sensitivity". The FWL problem is hence to analyze the effects to ensure that they do not cause significant deterioration in the performance of an implemented filter. The effects are obviously dependent upon the chosen wordlength and on the chosen arithmetic format (floating-point, fixed-point, etc.). Slightly less obvious is the fact that the FWL effects are very dependent upon the particular realization, (direct form, cascade, etc.), and upon the chosen operator (shift operator, δ operator, etc.). Thus in seeking to alleviate the FWL effects, the realization must also be considered.

The FWL effects have been studied for many years. Although many of the early works were motivated by problems in control systems [3, 45], the analysis of the effects were often considered in the open loop. See [35] for a comprehensive review of early work. Further reviews can be found in [52, 9, 26]. There has also been a large amount of work that considers the problem of round-off noise (e.g. [43, 42, 23]).

Early consideration of the transfer function sensitivity to rounding errors in the coefficients can be found in [29, 1]. The work of Thiele [46, 47, 48] is particularly important in defining a norm on the input-output sensitivity that is tractable. This sensitivity measure provides the foundation for much of the

subsequent work. Solutions for other similar measures can be found in [9, 55] and further developed in, for example, [33, 54, 22]. A related measure using a statistical analysis of the input-output sensitivity has been developed [27]. An extension to the multivariable system case is provided in [37]. The closed-loop control case has also been considered, for example in [38]. Methods for the simultaneous minimization of a sensitivity measure with round-off noise [36] and subject to scaling requirements [21] have also been developed recently.

The sensitivity of the poles (and zeros) is also a commonly used measure of the coefficient rounding effect. An early analysis appears in [28]. Mantey [39] showed that the poles/eigenvalues are dependent on the state-space realization. It is well-known that an eigenvalue sensitivity is minimized if the system is normal [44]. However Gevers and Li [9] subsequently determined the realization that would minimize a pole sensitivity measure combined with a zero sensitivity measure proposed in [51]. Much subsequent work (see [32, 50, 53, 30], for example) has considered various similar eigenvalue sensitivity measures for closed-loop control systems.

Most of the significant results have expressed the filter in the state space form. Although **most realizations can be transformed into the state-space form, this form is not completely general and has several limitations**. Firstly, the analysis of the rounding effect of a specific coefficient in a particular realization form can become very difficult after transformation to the state space form. Secondly, many realization forms require the computation of intermediate variables that cannot be expressed in the state-space form. Furthermore, the state space form is specific to the chosen operator. In reality all implementable operators are actually implemented using the shift operator. For example, a realization expressed in the form of a δ-operator is actually implemented using a shift operator in combination with an intermediate variable.

Thus a description that includes intermediate variables is required. The FWR Toolbox proposes a particular implicit state-space description that is not subject to these limitations. The proposed specialized implicit form provides a generalized description of any realization in a form that allows a straightforward analysis of the FWL effects as will be shown in section 3. The description is macroscopic in that it does not require coding details and is platform independent but gives a direct relationship between the description and the implementation algorithm. Note that the idea of representing the intermediate variables in the description has been considered previously [5] (see also [4, 41]), but the description form is less general than the implicit form considered in this paper. For example, δ-realizations cannot be described using this form.

3 A unifying framework

3.1 The Specialized Implicit Framework (SIF)

To show the utility of the implicit realization, we consider an example of the implementation of a δ-operator state-space realization. It is well-known [9,

40, 10] that the δ-operator is numerically superior to the usual shift operator generally resulting in less sensitive implementations with less rounding noise.

For a realization expressed with the δ-operator, the input/output relation is

$$\left\{\begin{array}{rcl} \delta[X(k)] & = & A_\delta X(k) + B_\delta U(k) \\ Y(k) & = & C_\delta X(k) + D_\delta U(k) \end{array}\right. \tag{1}$$

with $\delta = \frac{q-1}{\Delta}$, where Δ is a strictly positive constant and q is the delay operator [9]. This is equivalent, in infinite precision, to the classical state-space realization

$$\left\{\begin{array}{rcl} q[X(k)] & = & A_q X(k) + B_q U(k) \\ Y(k) & = & C_q X(k) + D_q U(k) \end{array}\right. \tag{2}$$

with $A_q = \Delta A_\delta + I$, $B_q = \Delta B_\delta$, $C_q = C_\delta$ and $D_q = D_\delta$.

With these two equivalent realizations, the parametrization is different, therefore when the parameters are subjected to FWL rounding, the two realizations are no longer equivalent, and the impact of the quantization is different. In addition, in order to implement the δ-operator, intermediate variables are necessary. These are also subject to FWL quantization. So the following algorithm

$$\begin{array}{rcl} T & \leftarrow & A_\delta X(k) + B_\delta U(k) \\ X(k+1) & \leftarrow & X(k) + \Delta T \\ Y(k) & \leftarrow & C_\delta X(k) + D_\delta U(k) \end{array} \tag{3}$$

implements (1) where T is an intermediate variable vector.

There are many other possible implementation forms, such as direct form I or II, cascade/parallel decomposition, lattice filters, mixed q/δ, etc., and many of these also require intermediate variables. In order to consider all of them within a general unifying framework, we propose a description, in a single equation, of the filter implementation. The equation provides an explicit description of the parametrization, and allows the analysis of the FWL effects, but is still a macroscopic description. Furthermore, the description is given within a formalism such that the description takes the form of an implicit state-space system. This specialized implicit framework (SIF) is given by

$$\begin{pmatrix} J & 0 & 0 \\ -K & I_n & 0 \\ -L & 0 & I_p \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & M & N \\ 0 & P & Q \\ 0 & R & S \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{4}$$

where

- $J \in \mathbb{R}^{l\times l}$, $K \in \mathbb{R}^{n\times l}$, $L \in \mathbb{R}^{p\times l}$, $M \in \mathbb{R}^{l\times n}$, $N \in \mathbb{R}^{l\times m}$, $P \in \mathbb{R}^{n\times n}$, $Q \in \mathbb{R}^{n\times m}$, $R \in \mathbb{R}^{p\times n}$, $S \in \mathbb{R}^{p\times m}$, $T(k) \in \mathbb{R}^l$, $X(k) \in \mathbb{R}^n$, $U(k) \in \mathbb{R}^m$ and $Y(k) \in \mathbb{R}^p$,
- matrix J is lower triangular with 1's on the diagonal, i.e.

$$J = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \star & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ \star & \star & \dots & 1 \end{pmatrix}, \tag{5}$$

- $T(k+1)$ is the intermediate variable in the calculations of step k (the column of 0's in the second matrix shows that $T(k)$ is not used for the calculation at step k – this characterizes the concept of an intermediate variable),
- $X(k+1)$ is the stored state-vector ($X(k)$ is effectively stored from one step to the next, in order to compute $X(k+1)$ at step k).

$T(k+1)$ and $X(k+1)$ form the descriptor-vector: $X(k+1)$ is stored from one step to the next, while $T(k+1)$ is computed and used within one time step.

It is implicitly assumed that the computations associated with the realization (4) are executed in row order giving the following algorithm:

$$\begin{aligned} &\text{[i]} && JT(k+1) \leftarrow MX(k) + NU(k) \\ &\text{[ii]} && X(k+1) \leftarrow KT(k+1) + PX(k) + QU(k) \\ &\text{[iii]} && Y(k) \leftarrow LT(k+1) + RX(k) + SU(k) \end{aligned}$$

Note that in practice, steps [ii] and [iii] could be exchanged to reduce the computational delay. Also note that because the computations are executed in row order and J is lower triangular with 1's on the diagonal, there is no need to compute J^{-1}.

Equation (4) is equivalent in infinite precision to the classical state-space form

$$\left(\begin{array}{c} T(k+1) \\ X(k+1) \\ \hline Y(k) \end{array}\right) = \left(\begin{array}{cc|c} 0 & J^{-1}M & J^{-1}N \\ 0 & A_Z & B_Z \\ \hline 0 & C_Z & D_Z \end{array}\right) \left(\begin{array}{c} T(k) \\ X(k) \\ \hline U(k) \end{array}\right) \tag{6}$$

with $A_Z \in \mathbb{R}^{n\times n}$, $B_Z \in \mathbb{R}^{n\times m}$, $C_Z \in \mathbb{R}^{p\times n}$ and $D_Z \in \mathbb{R}^{p\times m}$ where

$$A_Z = KJ^{-1}M + P, \qquad B_Z = KJ^{-1}N + Q, \tag{7}$$

$$C_Z = LJ^{-1}M + R, \qquad D_Z = LJ^{-1}N + S. \tag{8}$$

Note that (6) corresponds to a different parametrization than (4). The system transfer function is given by

$$H(z) = C_Z(zI_n - A_Z)^{-1}B_Z + D_Z. \tag{9}$$

3.2 definitions

To complete the framework, the following definitions are required.

Definition 1 *A **realization**, $\mathcal{R}$, is defined by the specific set of matrices J, K, L, M, N, P, Q, R and S used to describe a realization with the implicit form of (4) :*

$$\mathcal{R} :\triangleq (J, K, L, M, N, P, Q, R, S). \tag{10}$$

Remark 1 *$\mathcal{R}$ can also be defined by the matrix $Z \in \mathbb{R}^{(l+n+p)\times(l+n+m)}$*

$$Z \triangleq \begin{pmatrix} -J & M & N \\ K & P & Q \\ L & R & S \end{pmatrix} \tag{11}$$

and the dimensions l, m, n and p, so $\mathcal{R}$ could be defined by $\mathcal{R} := (Z, l, m, n, p)$.

Definition 2 *$\mathscr{R}_H$ denotes the set of realizations with transfer function H. These realizations are said to be equivalent.*

In order to encompass realizations with some special structure (q-operator state-space, δ-operator state-space, direct form, cascade, lattice filters, etc.), we define a set of realizations that possess a particular structure.

Definition 3 *A **structuration**[1] $\mathscr{S}$ is a set of realizations having a common structure: some coefficients or some dimensions are fixed* a priori.

Some examples of common structurations are given in the next section.

Definition 4 *$\mathscr{R}_H^{\mathscr{S}}$ is the set of equivalent structured realizations. Realizations from $\mathscr{R}_H^{\mathscr{S}}$ are structured according to $\mathscr{S}$ and have a transfer function H. Hence $\mathscr{R}_H^{\mathscr{S}} \triangleq \mathscr{R}_H \cap \mathscr{S}$.*

Definition 5 *A **parametrization** of a realization $\mathcal{R}$ is the set of coefficients of Z that are significant for the realization.*

3.3 Examples

3.3.1 Classical state-space

The classical state-space realization can, of course, be expressed with the SIF. The realization

$$\begin{cases} X(k+1) &= A_q X(k) + B_q U(k) \\ Y(k) &= C_q X(k) + D_q U(k) \end{cases} \tag{12}$$

correspond to the implicit form with $l = 0$, So

$$Z = \begin{pmatrix} . & . & . \\ . & A_q & B_q \\ . & C_q & D_q \end{pmatrix} \tag{13}$$

[1] This is a useful French word that we have purloined. It is also used in the field of social sciences. Here it means the set of structured realizations.

3.3.2 State-space with δ operator

The δ-state-space realization corresponds to

$$\left\{ \begin{array}{rcl} \delta[X(k)] & = & A_\delta X(k) + B_\delta U(k) \\ Y(k) & = & C_\delta X(k) + D_\delta U(k) \end{array} \right. \tag{14}$$

with $\delta = \frac{q-1}{\Delta}$ and $\Delta > 0$.
This is realized with

$$\begin{pmatrix} I_n & 0 & 0 \\ -\Delta I_n & I_n & 0 \\ 0 & 0 & I_p \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & A_\delta & B_\delta \\ 0 & I_n & 0 \\ 0 & C_\delta & D_\delta \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{15}$$

3.3.3 Cascade decomposition

Let's consider two systems $\mathcal{R}_1$ and $\mathcal{R}_2$ with the following SIF expression:

$$\begin{pmatrix} J_1 & 0 & 0 \\ -K_1 & I & 0 \\ -L_1 & 0 & I \end{pmatrix} \begin{pmatrix} T_1(k+1) \\ X_1(k+1) \\ Y_1(k) \end{pmatrix} = \begin{pmatrix} 0 & M_1 & N_1 \\ 0 & P_1 & Q_1 \\ 0 & R_1 & S_1 \end{pmatrix} \begin{pmatrix} T_1(k) \\ X_1(k) \\ U_1(k) \end{pmatrix} \tag{16}$$

$$\begin{pmatrix} J_2 & 0 & 0 \\ -K_2 & I & 0 \\ -L_2 & 0 & I \end{pmatrix} \begin{pmatrix} T_2(k+1) \\ X_2(k+1) \\ Y_2(k) \end{pmatrix} = \begin{pmatrix} 0 & M_2 & N_2 \\ 0 & P_2 & Q_2 \\ 0 & R_2 & S_2 \end{pmatrix} \begin{pmatrix} T_2(k) \\ X_2(k) \\ U_2(k) \end{pmatrix} \tag{17}$$

The two systems are put on cascade.

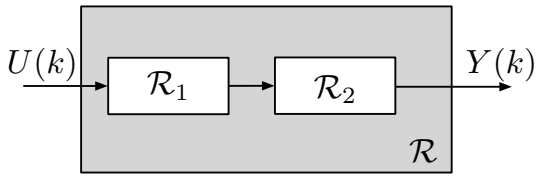

Figure 1: Two systems cascaded

The intermediate variables receive the result of the first system, so the cascaded system has the following SIF realization:

$$\left(\begin{array}{ccc:ccc} J_1 & 0 & 0 & 0 & 0 & 0 \\ -L_1 & I & 0 & 0 & 0 & 0 \\ 0 & -N_2 & J_2 & 0 & 0 & 0 \\ \hdashline -K_1 & 0 & 0 & I & 0 & 0 \\ 0 & -Q_2 & -K_2 & 0 & I & 0 \\ \hdashline 0 & -S_2 & -L_2 & 0 & 0 & I \end{array}\right) \begin{pmatrix} T_1(k+1) \\ T(k+1) \\ T_2(k+1) \\ X_1(k+1) \\ X_2(k+1) \\ Y_2(k) \end{pmatrix} = \left(\begin{array}{ccc:ccc} 0 & 0 & 0 & M_1 & 0 & N_1 \\ 0 & 0 & 0 & R_1 & 0 & S_1 \\ 0 & 0 & 0 & 0 & M_2 & 0 \\ \hdashline 0 & 0 & 0 & P_1 & 0 & Q_1 \\ 0 & 0 & 0 & 0 & P_2 & 0 \\ \hdashline 0 & 0 & 0 & 0 & R_2 & 0 \end{array}\right) \begin{pmatrix} T_1(k) \\ T(k) \\ T_2(k) \\ X_1(k) \\ X_2(k) \\ U_1(k) \end{pmatrix}$$

So

$$Z = \left(\begin{array}{ccc:ccc} -J_1 & 0 & 0 & M_1 & 0 & N_1 \\ L_1 & -I & 0 & R_1 & 0 & S_1 \\ 0 & N_2 & -J_2 & 0 & M_2 & 0 \\ \hdashline K_1 & 0 & 0 & P_1 & 0 & Q_1 \\ 0 & Q_2 & K_2 & 0 & P_2 & 0 \\ \hdashline 0 & S_2 & L_2 & 0 & R_2 & 0 \end{array}\right) \tag{18}$$

3.3.4 Others forms

A lot of other possible structurations are considered. Here are some of them:

- Direct Form I with q-operator (`DFIq2FWR`)
- ρ Direct Form II transposed (`rhoDFIIt2FWR`), that encompasses the Direct Form II with q and δ-operators
- cascade and parallel decomposition (`plus`, `mtimes`)
- classical state-space realizations (`SS2FWR`)
- δ-state-space realizations (`SSdelta2FWR`)
- Modal form with ρ-operator (`Modalrho2FWR`)
- ...

3.4 Equivalent classes

In order to exploit the potential offered by the specialized implicit form in improving implementations, it is necessary to describe sets of equivalent system realizations. However, non-minimal realizations may provide better implementations (the δ-form can be seen as a non-minimal realization when expressed in the implicit state-space form with the shift operator. Hence the notion of equivalence needs to be extended so that the system state dimension does not need to be preserved. The *Inclusion Principle*, introduced by Šiljak and Ikeda [24, 49] in the context of decentralized control, has been used to allows the formalization of the *equivalence* and *inclusion* relations between two realizations $\mathcal{R}$ and $\tilde{\mathcal{R}}$ (see [17]).

Although this extension of the *Inclusion Principle* gives the formal description of equivalent classes, it is of practical interest to consider realizations of the same dimensions ($\tilde{l} = l$ and $\tilde{n} = n$) where transformations from one realization to another is only a similarity transformation.

Proposition 1 *Consider a realization* $\mathcal{R} := (Z, l, m, n, p)$. *All the realizations* $\tilde{\mathcal{R}} := (\tilde{Z}, l, m, n, p)$ *with*

$$\tilde{Z} = \begin{pmatrix} \mathcal{Y} & & \\ & \mathcal{U}^{-1} & \\ & & I_p \end{pmatrix} Z \begin{pmatrix} \mathcal{W} & & \\ & \mathcal{U} & \\ & & I_m \end{pmatrix} \tag{19}$$

and $\mathcal{U}$, $\mathcal{W}$, $\mathcal{Y}$ are non-singular matrices, are equivalent to $\mathcal{R}$.
Eq. (19) defines a "$\mathcal{UYW}$-transformation.

It is also possible to just consider a subset of similarity transformations that preserve a particular structure, say cascade or delta. These are always a particular case of proposition 1.
For example, if an initial δ-structured realization $\mathcal{R} := (Z_0, n, m, n, p)$ is given, the subset of equivalent δ-structured realization is defined by

$$\mathscr{R}_H^{\mathscr{S}_\delta} = \left\{ \begin{array}{l} \mathcal{R} := (Z, n, m, n, p) \backslash \\ \quad Z = \begin{pmatrix} \mathcal{U}^{-1} & & \\ & \mathcal{U}^{-1} & \\ & & I_p \end{pmatrix} Z_0 \begin{pmatrix} \mathcal{U} & & \\ & \mathcal{U} & \\ & & I_m \end{pmatrix} \\ \quad \forall \mathcal{U} \in \mathbb{R}^{n \times n} \text{ non-singular} \end{array} \right\} \tag{20}$$

In addition to a description of the various existing realizations with the exact parametrization, this formalism gives an algebraic characterization of equivalent classes. These classes can be used to search for an optimal structured realization (see Section 3.5).

3.5 Finite Wordlength measures

3.5.1 Coefficient's quantization

A coefficient's quantization depends both on its value and its representation.

Firstly if the value of a coefficient is such that it will be quantized without error, then that parameter makes no contribution to the overall coefficient sensitivity. Hence we introduce weighting matrices W_J to W_S (and also W_Z) respectively associated with matrices J to S of a realization, such that

$$(W_X)_{i,j} \triangleq \begin{cases} 0 & \text{if } X_{i,j} \text{ is exactly implemented,} \\ 1 & \text{otherwise.} \end{cases} \tag{21}$$

In the toolbox, the notation (β, γ) is used for the fixed-point representation of a variable or coefficient (2's complement scheme), according to figure 2: β is the total wordlength in bits of the representation, whereas γ is the fractional part wordlength (it gives the binary-point position of the representation). They are fixed for each variable and coefficient, and implicit, unlike the floating-point representation. β and γ will be suffixed by the variable/coefficient they refer to. These parameters could be scalars, vectors or matrices, according to the variables they refer to.

To represent a value x without overflow, a fixed-point representation (β_x, γ_x) may satisfy:

$$\beta_x - \gamma_x - 1 \geqslant \left\lfloor \log_2 |x| \right\rfloor + 1 \tag{22}$$

where the $\lfloor a \rfloor$ operation rounds a to the nearest integer lower or equal to a (for positive numbers $\lfloor a \rfloor$ is the integer part).

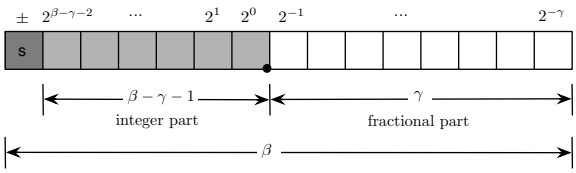

Figure 2: Fixed-point representation

In order to simplify the expressions that manipulate wordlengths or binary point positions, matrix extensions of $\log_2$, floor operator $\lfloor . \rfloor$ and power of 2 are used. For example, if $M \in \mathbb{R}^{p\times q}$, then $\log_2(M) \in \mathbb{R}^{p\times q}$ such as $(\log_2(M))_{i,j} \triangleq \log_2(M_{i,j})$.

3.5.2 Input-Output sensitivity

The open-loop transfer function sensitivity measure is defined by

$$M_{L_2}^W = \left\| \frac{\delta H}{\delta Z} \times r_Z \right\|_F^2 . \tag{23}$$

where $\frac{\delta H}{\delta Z} \in \mathbb{R}^{l+n+p\times l+n+q}$ is the *transfer function sensitivity matrix*. It is the matrix of the L_2-norm of the sensitivity of the transfer function H with respect to each coefficient $Z_{i,j}$. It is defined by

$$\left(\frac{\delta H}{\delta Z}\right)_{i,j} \triangleq \left\| \frac{\partial H}{\partial Z_{i,j}} \right\|_2 , \tag{24}$$

In SISO case, the $M_{L_2}^W$ measure is equal to

$$M_{L_2}^W = \left\| \frac{\partial H}{\partial Z} \times r_Z \right\|_2^2 . \tag{25}$$

and is then an extension to the SIF of the classical state-space sensitivity measure

$$M_{L_2} \triangleq \left\| \frac{\partial H}{\partial A} \right\|_2^2 + \left\| \frac{\partial H}{\partial B} \right\|_2^2 + \left\| \frac{\partial H}{\partial C} \right\|_2^2 . \tag{26}$$

This measure is implemented with the function `MsensH`.
See [13],[17] for more details.

3.5.3 Pole sensitivity

The pole sensitivity measure of $\mathcal{R}$ is defined by

$$\Psi = \sum_{k=1}^{n} \left\| \frac{\partial |\lambda_k|}{\partial Z} \times r_Z \right\|_F^2 . \tag{27}$$

(it is also possible to only consider the sensitivity of λ_k instead of the sensitivity of $|\lambda_k|$.

A *pole sensitivity matrix* can also be constructed to evaluate the overall impact of each coefficient. Let $\frac{\delta|\lambda|}{\delta Z}$ denote the pole sensitivity matrix defined by

$$\left(\frac{\delta\,|\lambda|}{\delta Z}\right)_{i,j} \triangleq \sqrt{\sum_{k=1}^{n}\left(\frac{\partial\,|\lambda_k|}{\partial Z_{i,j}}\right)^2}. \tag{28}$$

Then, The pole sensitivity measure is then given by:

$$\Psi = \left\|\frac{\delta\,|\lambda|}{\delta Z} \times r_Z\right\|_F^2. \tag{29}$$

This measure is implemented with the function `MsensPole`.
See [14],[17] for more details.

3.5.4 Output roundoff noise

When implemented a realization $\mathcal{R}$, the steps (i) to (iii) are modified by the add of noises $\xi_T(k)$, $\xi_X(k)$ and $\xi_Y(k)$:

$$\begin{aligned} J.T(k+1) &\leftarrow M.X(k) + N.U(k) + \xi_T(k) \\ X(k+1) &\leftarrow K.T(k+1) + P.X(k) + Q.U(k) + \xi_X(k) \\ Y(k) &\leftarrow L.T(k+1) + R.X(k) + S.U(k) + \xi_Y(k) \end{aligned} \tag{30}$$

These noises added depend on:

- the way the computations are organized (the order of the sums) and done,
- the fixed-point representation of the inputs, the outputs,
- and the fixed-point representation chosen for the states, the intermediate variables and the coefficients.

The add of these noises are equivalent to a noise $\xi'(k)$ added on the output. The Output Noise Power is defined as the power of $\xi'(k)$:

$$P \triangleq E\xi'(k)\xi'(k)^\top \tag{31}$$

where the $E.$ is the mean operator.

The Roundoff Noise Gain is the output noise power computed in a specific computational scheme : the noises are supposed to appear only after each multiplication and are modeled by centered white noise statistically independent. Each noise is supposed to have the same power σ_0^2 (determined by the wordlength chosen for all the variables and coefficients).

The Roundoff Noise Gain is defined by

$$G \triangleq \frac{P}{\sigma_0^2} \tag{32}$$

These measures are implemented with the function `ONP` and `RNG`. See [19] and [20].

3.5.5 Closed-loop measures

These measures are also extended to the closed-loop context. They now concern the closed-loop transfer function or the closed-loop poles.

Consider the plant $\mathcal{P}$ together with the controller $\mathcal{C}$ according to the standard form shown in Figure 3, where $W(k) \in \mathbb{R}^{p_1}$ is the exogenous input, $Y(k) \in \mathbb{R}^{p_2}$ the control input, $Z(k) \in \mathbb{R}^{m_1}$ the controlled output and $U(k) \in \mathbb{R}^{m_2}$ the measured output.

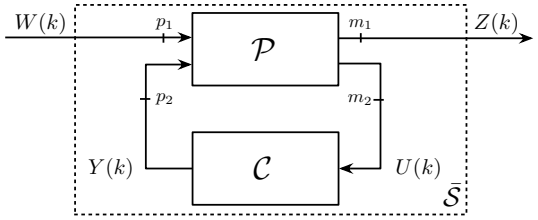

Figure 3: Closed-loop control system

The controller is defined as $\mathcal{C} := (Z, l, m_2, n, p_2)$ and the plant $\mathcal{P}$ as

$$\mathcal{P} := \left(\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & 0 \end{array}\right) \tag{33}$$

where $A \in \mathbb{R}^{n_\mathcal{P} \times n_\mathcal{P}}$, $B_1 \in \mathbb{R}^{n_\mathcal{P} \times p_1}$, $B_2 \in \mathbb{R}^{n_\mathcal{P} \times p_2}$, $C_1 \in \mathbb{R}^{m_1 \times n_\mathcal{P}}$, $C_2 \in \mathbb{R}^{m_2 \times n_\mathcal{P}}$, $D_{11} \in \mathbb{R}^{m_1 \times p_1}$, $D_{12} \in \mathbb{R}^{m_1 \times p_2}$, $D_{21} \in \mathbb{R}^{m_2 \times p_1}$ and $D_{22} \in \mathbb{R}^{m_2 \times p_2}$ is assumed to be zero only to simplify the mathematical expressions.

The closed-loop system $\bar{\mathcal{S}}$ is then given by

$$\bar{\mathcal{S}} = F_l(\mathcal{P}, \mathcal{C}) := \left(\begin{array}{c|c} \bar{A} & \bar{B} \\ \hline \bar{C} & \bar{D} \end{array}\right) \tag{34}$$

where $F_l(\cdot, \cdot)$ is the well-known lower linear fractional transform [56] and where $\bar{A} \in \mathbb{R}^{n_\mathcal{P}+n \times n_\mathcal{P}+n}$, $\bar{B} \in \mathbb{R}^{n_\mathcal{P}+n \times p_1}$, $\bar{C} \in \mathbb{R}^{m_1 \times n_\mathcal{P}+n}$ and $\bar{D} \in \mathbb{R}^{m_1 \times p_1}$ are such that

$$\bar{A} = \begin{pmatrix} A + B_2 D_Z C_2 & B_2 C_Z \\ B_Z C_2 & A_Z \end{pmatrix}, \qquad \bar{B} = \begin{pmatrix} B_1 + B_2 D_Z D_{21} \\ B_Z D_{21} \end{pmatrix}, \tag{35}$$

$$\bar{C} = \begin{pmatrix} C_1 + D_{12} D_Z C_2 & D_{12} C_Z \end{pmatrix}, \qquad \bar{D} = D_{11} + D_{12} D_Z D_{21}. \tag{36}$$

The closed-loop transfer function is

$$\bar{H} : z \mapsto \bar{C} \left(zI - \bar{A}\right)^{-1} \bar{B} + \bar{D}. \tag{37}$$

These notations will be used in the toolbox.
Then, the following measures are considered:

- the input-output sensitivity (`MsensH_cl`)

$$\bar{M}_{L_2}^W = \left\| \frac{\partial \bar{H}}{\partial Z} \times r_Z \right\|_2^2. \tag{38}$$

- the pole-sensitivity and related stability measure (`MsensPole_cl` and `Mstability`)

$$\bar{\Psi} = \sum_{k=1}^{n} \left\| \frac{\partial \left| \bar{\lambda}_k \right|}{\partial Z} \times r_Z \right\|_F^2. \tag{39}$$

- the roundoff noise gain (`RNG_cl`)

See [18].

4 The optimal realization problem

The problem of determining the best realization can be posed as follows:

Problem 1 (Optimal realization problem) *Consider a transfer function H and a sensitivity measure $\mathcal{J}$. The* optimal design problem *is to find the best realization $\mathcal{R}_{opt}$ with transfer function H according to the criteria $\mathcal{J}$, that is*

$$\mathcal{R}_{opt} = \arg \min_{\mathcal{R} \in \mathscr{R}_H} \mathcal{J}(\mathcal{R}). \tag{40}$$

Due to the size of $\mathscr{R}_H$, this problem cannot be solved practically. Indeed, a solution may even have infinite dimension. Hence the following problem is introduced to restrict the search to a particular structuration.

Problem 2 (Optimal structured realization problem) *The problem to find the optimal structured realization $\mathcal{R}_{opt}^{\mathscr{S}}$, that is*

$$\mathcal{R}_{opt}^{\mathscr{S}} = \arg \min_{\mathcal{R} \in \mathscr{R}_H^{\mathscr{S}}} \mathcal{J}(\mathcal{R}). \tag{41}$$

The *Inclusion Principle* (Proposition 1) provides the means to search over the structured realizations set $\mathscr{R}_H^{\mathscr{S}}$.
Since the measure $\mathcal{J}$ could be non-smooth and/or non-convex, the Adaptive Simulated Annealing (ASA) [25] method could be used to solve Problem 2. This method has worked well for other optimal realization problems [53].
The FWR Toolbox can also use simplex or Quasi-Newton algorithm to solve problem 2.

5 Tutorial

5.1 First example

Let's consider this discrete state-space system:

```
>> A = [ 1.4590 -0.91037 0.39565; 1 0 0; 0 0.5 0 ];
>> B = [ 0.5; 0; 0 ];
>> C = [ 0.28261 0.13244 0.15183 ];
>> D = 0.0031689;
>> Sys = ss(A,B,C,D, 1e-2);
```

Then, we can create a first Finite Wordlength Realization

```
R = SS2FWR(Sys);
```

First, let's display this FWR object

```
>> R
R has 1 input, 1 output, 3 states, and 0 intermediate variable.
Z=
   1.4590e+00  -9.1037e-01   3.9565e-01   5.0000e-01
   1.0000e+00            0            0            0
            0   5.0000e-01            0            0
   2.8261e-01   1.3244e-01   1.5183e-01   3.1689e-03
```

It is also possible to display, for example, the associated matrices Wo (observability grammian), P and W_Z:

```
>> R.Wo
ans =
   1.0232e+00  -6.1354e-01   3.9081e-01
  -6.1354e-01   5.5557e-01  -2.7112e-01
   3.9081e-01  -2.7112e-01   1.8322e-01

>> R.P
ans =
   1.4590e+00  -9.1037e-01   3.9565e-01
   1.0000e+00            0            0
            0   5.0000e-01            0

>> R.WZ
ans =
     1     1     1     1
     0     0     0     0
     0     1     0     0
     1     1     1     1
```

This last result shows that some coefficients will be considered as exactly implemented (coefficients where $(W_Z)_{ij}$ is null), whereas some will be modified during the quantization.
It is possible to ask for the input-output sensitivity and the sensitivity matrix

```
>> [M MZ] = MsensH(R)
M =
   1.4391e+01

MZ =
   1.9730e+00   1.9730e+00   9.8651e-01   1.0115e+00
            0            0            0            0
            0   8.4062e-01            0            0
   1.1358e+00   1.1358e+00   5.6788e-01   1.0000e+00
```

To have the sensitivity for all the coefficients, it is possible to set W_Z to 1

```
>> R.WZ=ones(4);
>> [M MZ] = MsensH(R);
>> MZ
MZ =
   1.9730e+00   1.9730e+00   9.8651e-01   1.0115e+00
   1.3716e+00   1.3716e+00   6.8582e-01   7.4536e-01
   8.4062e-01   8.4062e-01   4.2031e-01   4.2805e-01
   1.1358e+00   1.1358e+00   5.6788e-01   1.0000e+00
```

Now, it could be interesting to find, among the equivalent state-space realizations, one that minimize this input-output sensitivity. For this purpose, we need to create a Finite Wordlength Structuration:

```
>> S = SS2FWS( Sys)
 has 1 input, 1 output, 3 states, and 0 intermediate variable.
Z=
   1.4590e+00  -9.1037e-01   3.9565e-01   5.0000e-01
   1.0000e+00            0            0            0
            0   5.0000e-01            0            0
   2.8261e-01   1.3244e-01   1.5183e-01   3.1689e-03
T=
     1     0     0
     0     1     0
     0     0     1
```

In addition to a FWR object, a FWStructuration includes a parameter `T`, that allows to get all the state-space equivalent realizations $(T^{-1}AT, T^{-1}B, CT, D)$. To consider a new realization, deduced from the original one with this transformation, we simply give a new value for T:

```
>> S.T=rand(3)
```

```
 has 1 input, 1 output, 3 states, and 0 intermediate variable.
Z=
   3.7455e-01   1.3029e+00  -4.6071e+00  -2.0013e+00
   2.1170e-01  -9.4236e-01   5.5850e+00   5.0983e-01
  -1.2475e-02  -4.9193e-01   2.0268e+00   1.8077e+00
   3.2995e-01   1.8533e-01   3.9119e-01   3.1689e-03
T=
   3.8156e-01   1.8687e-01   6.4631e-01
   7.6552e-01   4.8976e-01   7.0936e-01
   7.9520e-01   4.4559e-01   7.5469e-01
```

The original realization is stored in `S.Rini`, whereas the actual realization is obtained with `S.R`.
It is important to remark that all the equivalent realizations deduced from the original one have the same W_Z matrix. So, in order to consider fully parameterized realizations, it is important to set W_Z to a matrix with all coefficients set to 1, with `S.Rini.WZ=ones(size(S.Rini.WZ));`.
It is possible to compare these two realizations and their I/O-sensitivity

```
>> MsensH(S.Rini)
ans =
   1.4391e+01

>> MsensH(S.R)
ans =
   5.8668e+02
```

Now, the most interesting thing is to search for the optimal realization. First, we need to set the options for the search

```
>> options = {'method','simplex','Display','Iter','MaxFunEvals',1e4};
```

Then, we can run the optimization:

```
>> S = optim( S, optins, @MsensH)
```

The options are cells of pairs. Some options concern the `optim` methods, while some are directly passed to the Matlab optimization algorithm used (`fminsearch`, `fminunc`, ...). {`'method','simplex'`} allows to use the `fminsearch` algorithm, whereas {`'Display','Iter'`} and {`'MaxFunEvals',1e4`} allow to display each iteration and set the maximum number of function evaluation to 10^4. See `optimset` for all the possible options and the method `optim` for all the possible options.

`S` has now the *optimal* value for T.

6 The classes

The the `FWR toolbox` is based on two classes:

- the `FWR` class to describe the Finite Wordlength Realizations
- the `FWS` class for the Finite Wordlength Structurations

6.1 The `FWR` class

The `FWR` class describes a realization expressed with the SIF (see def. 1). It contains the following fields:

- `l`, `m`, `n` and `p` : dimensions of the realization (l intermediate variables, m inputs, n states and p outputs)
- `J`, `K`, `L`, `M`, `N`, `P`, `Q`, `R` and `S` : correspond to matrices J to S
- `Z` : the Z matrix defined in (11) from J to S
- `WJ`, `WK`, `WL`, `WM`, `WN`, `WP`, `WQ`, `WR` and `WS` : the weighting matrices W_J to W_S (they imply which parameter is exactly implemented (0, ± 1, a power of two or any number that will not be changed during the quantization), see (21))
- `WZ` : the W_Z matrix
- `AZ`, `BZ`, `CZ` and `AZ` : matrices A_Z, B_Z, C_Z and D_Z (see equations (7) and (8))
- `Wc`, `Wo` : commandability and observability gramians W_c and W_o
- `FPIS` : a *Fixed-Point Implementation Scheme* (see section 6.1.1)
- `fp`, `block`, `rZ` : these fields are related to the coefficient representation
  - `fp` : sets if the implementation uses the fixed-point or the floating-point representation. `fp` can take the values `'fixed'` (1=default) or `'floating'` (2).
  - `block` : sets the coefficient's block. Coefficients in the same block share the same representation (binary-point position). `block` can take the values `'full'` (1), `'natural'` (2=default), or `'none'` (3).
  - `rZ` : gives how much Z is changed during the quantization process : Z is perturbed to $Z + r_Z \times \Delta$ where

$$r_Z \triangleq \begin{cases} W_Z & \text{for fixed-point representation,} \\ 2\eta_Z \times W_Z & \text{for floating-point representation,} \end{cases} \tag{42}$$

    and η_Z is such that

$$(\eta_Z)_{i,j} \triangleq \left\{ \begin{array}{l} \text{the largest absolute value of} \\ \text{the block in which } Z_{i,j} \text{ resides.} \end{array} \right. \tag{43}$$

These fields are only used for the sensitivities measure (`MsensH`, `MsensPole`, ...), and are independent from the FPIS. These fields will probably disapear.
See [17] for *block-fixed-point* and *block-floating-point* representation.

When a FWR object is created, it is not possible to change its dimensions (fields `l`, `m`, `n` and `p` or the size of the matrices) or the fields `AZ`, `BZ`, `CZ`, `AZ`, `Wc` and `Wo`.
Fields `Z` and `WZ` are redundant with fields `J` to `S` and `WJ` to `WS`, but they can both be usefull. Changing `Z` automatically changes fields `J` to `S` and reciprocally (the same with `WZ`). Fields `AZ`, `BZ`, `CZ`, `AZ`, `Wc` and `Wo` are deduced accordingly.

6.1.1 Fixed-Point Implementation Scheme

The FPIS is a structure to set the *Fixed-Point Implementation Scheme*. It is composed by:

- the fixed-point format of the input (β_U, γ_U) and its maximum magnitude value $\overset{\max}{U}$
- the fixed-point format of the intermediate variables (β_T, γ_T)
- the fixed-point format of the states (β_X, γ_X)
- the fixed-point format of the output (β_Y, γ_Y)
- the fixed-point format of the coefficients (β_Z, γ_Z)
- the fixed-point format of the accumulator $(\beta_{ADD} + \beta_G, \gamma_{ADD})$ (β_G guard bits)
- the right-shift bits after each scalar product d_{ADD} (`shiftADD`)
- the right-shift bits after each multiplication by a coefficient d_Z (`shiftZ`)
- the computational scheme : *Roundoff After Multiplication* (RAM) or *Roundoff Before Multiplication* (RBM)

The algorithm

$$\begin{aligned} &\text{[i]} && JT(k+1) \leftarrow MX(k) + NU(k) \\ &\text{[ii]} && X(k+1) \leftarrow KT(k+1) + PX(k) + QU(k) \\ &\text{[iii]} && Y(k) \leftarrow LT(k+1) + RX(k) + SU(k) \end{aligned}$$

requires to implement $l + n + p$ scalar products.
Each scalar product

$$S = \sum_{i=1}^{n} P_i E_i \tag{44}$$

where $(P_i)_{1 \leqslant i \leqslant n}$ are given coefficients and $(E_i)_{1 \leqslant i \leqslant n}$ some bounded variables, can be implemented according to the following algorithms 1 and 2, and where P'_i, E'_i and S'_i are the integer representations (according to their fixed-point format) to P_i,E_i and S_i.

$Add \leftarrow 0$
for i *from 0 to* n **do**
  $Add \leftarrow (P'_i * E'_i) >> d_i$
end
$S'_i \leftarrow Add >> d'_i$

Algorithm 1: *Roundoff After Multiplication* (RAM)

$Add \leftarrow 0$
for i *from 0 to* n **do**
  $Add \leftarrow (P'_i >> d_i) * E'_i$
end
$S'_i \leftarrow Add >> d'_i$

Algorithm 2: *Roundoff Before Multiplication* (RBM)

Of course, d_i represents the right-shift after each multiplication and d'_i represents the final shift. They respectively correspond to the d_Z and d_{ADD} shift in the SIF algorithm.

The user may specify all the wordlengths (β_U, β_T, β_X, β_Y, β_{ADD}, β_g and β_Z) and $\overset{\max}{U}$.

See [20] and the function `setFPIS` for more details.

6.1.2 Methods

The FWR's methods are :

`algorithmCfloat`	Return the algorithm associated to this realization.
`algorithmLaTeX`	Return the pseudocode algorithm described in LaTeX
`computationalCost`	Give the number of additions and multiplications implied in the realization
`computeW`	Compute (or update) the weighting matrices (W_J to W_S, and W_Z) of a FWR object
`display`	Display the realization (dimensions and Z)
`double`	Convert FWR object to double (return Z matrix)
`FWR`	FWR class' constructor
`FWRmat2LaTeX`	Display a matrix (Z or a sensitivity matrix) in LaTeX(with `pmat`
`get`	Get some properties of a FWR object
`implementLaTeX`	Return the associated fixed-point algorithm described in LaTeX
`implementMATLAB`	Create the associated fixed-point algorithm in Matlab language
`implementVHDL`	Create the associated fixed-point algorithm in VHDL.
`l2scaling`	Perform a L_2-scaling on the FWR

MsensH	Compute the open-loop transfer function sensitivity measure (and the
MsensH_cl	Compute the closed-loop transfer function sensitivity measure (and the
MsensPole	Compute the open-loop pole sensitivity measure
MsensPole_cl	Compute the closed-loop pole sensitivity measure
Mstability	Compute the closed-loop pole sensitivity stability related measure
mtimes	Multiply two FWR (put them in cascade)
ONP	Compute the Output Noise Power for a FWR object with Roundoff Before
plus	add two FWR object (put them in parallel)
quantized	Return the quantized realization, according to a fixed-point implementation scheme
realize	Numerically compute the outputs, states and intermediate variables with a given input U.
relaxedl2scaling	Perform a relaxed-L_2-scaling on the FWR.
RNG	Compute the open-loop Roundoff Noise Gain
RNG_cl	Compute the closed-loop Roundoff Noise Gain
set	Set some properties of a FWR object
setFPIS	Set the Fixed-Point Implementation Scheme (FPIS) of an FWR object
sigma_tf	Compute the open-loop transfer function error $\sigma^2_{\Delta H}$)
simplify	Simplify (if possible) a FWR, by removing the non-necessary intermediate variables and states
size	Return the size of the FW Realization
ss	Convert a FWR object into a ss (state-space) object (equivalent state-space)
subsasgn	Subscripted assign for FWR object
subsref	Subscripted reference for FWR object
tf	Convert a FWR object into a tf object (transfer function)
TradeOffMeasure_cl	Compute a (pseudo) tradeoff closed-loop measure
transform	Perform a UYW-transformation (similarity on Z)

6.2 The FWS class

The FWS class describes a structuration and the way to search over the set of equivalent realizations of this structuration. This is done by defining the parameters used to search over the equivalent set and how theses parameters give new realizations (via the $\mathcal{UYW}$-transformation, eq. (19), or directly).

It contains the following fields :

- Rini : the initial realization of this structuration (all the other equivalent structured realizations are computed *from* this realization),

- `R` : the actual considered realization,
- `paramsName` : name of the parameters used to search over the equivalent set,
- `paramsValue` : value of this parameters (they define the actual considered realization `R`),
- `paramsSize` : size of these parameters,
- `indices` : vector of indices. To do the optimization, the values of the parameters are put together in a row vector, and `indices` stores these indices, in that row, of each parameters (it is used when some parameters are fixed during the optimization). This is also internally used when some parameters are fixed,
- `UYWfun` : handle to a function that gives the transformation matrices $\mathcal{U}$, $\mathcal{Y}$ and $\mathcal{W}$ from the parameters,
- `Rfun` : handle to a function that gives the realization from the parameters (when the `UYWfun` cannot be defined),
- `dataMeasure` : cell of extra data used for the FWL measures (to store values that do not change from one realization to another equivalent) : every FWL measure can add what it needs inside,
- `dataFWS` : cell of extra data used to store internal values (can be used when defining a structuration).

The parameters (their names, values and sizes are stored in `paramsName`, `paramsValue` and `paramsSize`) can be used directly by their names, like any other fields : if `S` is a structuration, with parameters named `'T1'` and `'T2'`, expressions `S.T1` and `S.T2` allow to access the corresponding values.
All the fields of the `FWS` class are accessible with the `.fieldname` method (`get`), but it is only possible to change `Rini` and the parameters (it is not possible to change their sizes). And changing one parameter will automatically change `R`.

When a structuration `S` is defined, it is then possible to search over all the equivalent realizations with the same structuration by changing the values of the associated parameters (those where the names are in `paramsName`). `S.R` gives the new realization (automatically update when `paramsValue` change, with `S.`*`paramName`*`=...`). This can be done with the `optim` method, that uses quasi-Newton, simplex or ASA algorithm.

There is two ways to define how the parameters give a new realization. Only one of the two fields `UYWfun` and `Rfun` must be filled. The function `UYWfun` must be preferred, because it allows to compute more quickly the FWL measures whose comportment with the $\mathcal{UYW}$-transformation is defined.

6.2.1 `UYWfun` function

Purpose :
The `UYWfun` is a function that defines how to search over all the equivalent realizations. It transforms the parameters of the structuration (given by `args`) in transformation matrices $\mathcal{U}$, $\mathcal{Y}$ and $\mathcal{W}$. The `cost_flag` indicates if the transformation is valid.

Syntax :
`function [U,Y,W,cost_flag] = my_UYW_fun( Rini, paramsValue, dataFWS)`

Arguments :

`U,Y,W`	: matrices $\mathcal{U}$, $\mathcal{Y}$ and $\mathcal{W}$
`cost_flag`	: boolean that indicates if the parameters forms a valid transformation
`Rini`	: initial FW Realization
`paramsValue`	: cells of parameters values
`dataFWS`	: cell of extra data (that can be used to store internal values)

The parameters' values are accessible with `paramsValue` : the i^{th} parameter (in the order it is built) is given by `paramsValue{i}`. Most of the time, `Rini` and `dataFWS` are not useful (`U`, `Y` and `W` often directly depend on `paramsValue`).

6.2.2 `Rfun` function

Purpose :
The `Rfun` is a function that directly creates a new realization from the parameters' values, without returning $\mathcal{U}$, $\mathcal{Y}$ and $\mathcal{W}$ matrices.

Syntax :
`function [R,cost_flag] = my_R_fun( Rini, paramsValue, dataFWS)`

Arguments :

`R`	: new realization
`cost_flag`	: boolean that indicates if the parameters forms a valid transformation
`Rini`	: initial FW Realization
`paramsValue`	: cells of parameters' values
`dataFWS`	: cell of extra data (that can be used to store internal values)

Everyone who creates a FWS should follow theses definitions.

6.2.3 Methods

The FWS's methods are :

display	Display the realization (dimensions, Z and the parameters)

FWS	Constructor of the FWS class.
genCostFunction	G eneric cost function for optimization of a FWS
get	Get some properties of a FWS object (or list the properties if propName is ignored)
getValues	Return the parameters' value (cells of values)
MsensPole	Compute the open-loop pole sensitivity measure for a FWS object.
MsensPole_cl	Compute the closed-loop pole sensitivity measure for a FWS object.
Mstability	Compute the closed-loop pole sensitivity stability related measure for a FWS object.
optim	Find the optimal realization, according to the `measureFun` measure, in the set of structured equivalent realizations
RNG	Compute the open-loop Roundoff Noise Gain for a FWS.
RNG_cl	Compute the closed-loop Roundoff Noise Gain for a FWS.
set	Set some properties of a FWS object
setFPIS	Set the Fixed-Point Implementation Scheme (FPIS) of an FWS object
ss	Convert a FWS object into a ss (state-space) object (equivalent state-space)
subsasgn	Subscripted assign for FWS object.
subsref	Subscripted reference for FWS object.
tf	Convert a FWS object into a tf object (transfer function)

7 FWR Toolbox reference

7.1 create realizations and structurations

In additions to the FWR/FWS's methods, the following functions can be used to create classical realizations and structurations :

DFIq2FWR	Transform a transfer function (`tf` object) into a FWR object with the Direct Form I scheme
FFT2FWR	Build Fast Fourier Transform (size n) with a FWR object
implicitSS2FWS	Transform an implicit State-space system into a FWS object
Modaldelta2FWR	Transform a δ-based modal realization into a FWR object
Modalrho2FWR	Transform a ρ-modal realization into a FWR object, where the parameter Δ is reserved for relaxed L_2-scaling
Modalrho2FWS	Transform a ρ-modal realization into a FWS object, where the parameter Δ is reserved for relaxed L_2-scaling
Observer2FWR	Transform an Observer-State-Feedback form in FWR object
OpModalrho2FWR	Optimal ρ-based modal realization under FWR structure
rhoDFIIt2FWR	Build a ρ-DFIIt realization
rhoDFIIt2FWS	Build a ρ-DFIIt structuration
SS2FWR	Transform a classical state-space system (`ss` object) into a FWR object
SS2FWS	Transform a classical state-space system (`ss` object) into a FWS object
SSdelta2FWR	Transform a δ-state-space realization into a FWR object
SSdelta2FWS	Transform a δ-state-space realization into a FWS object
SSrho2FWR	Transform a ρ-based state-space realization into a FWR object
SSrho2FWS	Transform a ρ-state-space realization into a FWS object

Here is the detailed list:

7.1.1 DFIq2FWR

Purpose :
Transform a transfer function (`tf` object) into a FWR object with the Direct Form I scheme

Syntax :
`R = DFIq2FWR( H )`

Parameters :
`R` : FWR object
`H` : tf object

Description :
The system considered is described by the transfer function

$$H(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{\sum_{i=0}^{n} a_i z^{-i}} \tag{45}$$

and implemented with the recurrent equation

$$Y(k) = \frac{1}{a_0}\left(\sum_{i=0}^{n} b_i U(k-i) - \sum_{i=1}^{n} a_i Y(k-i)\right) \qquad \forall k > n \tag{46}$$

This could be described by the figure 4.
The (finite precision) equivalent system, in the implicit state-space formalism,

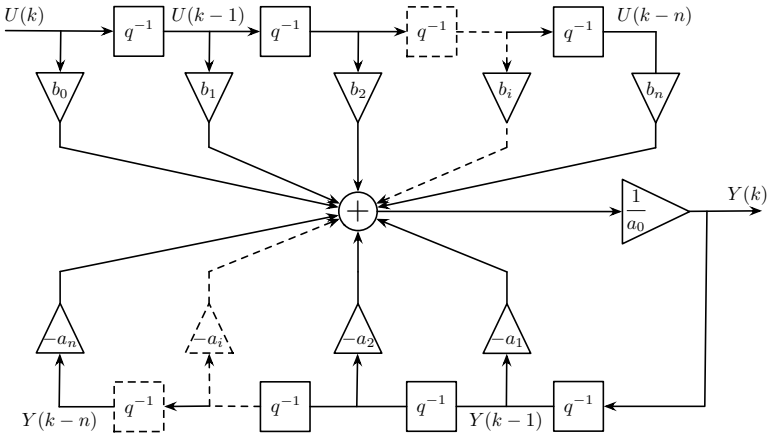

Figure 4: Direct Form I with q-operator

is given by

$$\begin{pmatrix} a_0 & 0 & 0 \\ -\Gamma_4 & I_n & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & \Gamma_1 & b_0 \\ 0 & \Gamma_2 & \Gamma_3 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{47}$$

where

$$\Gamma_1 = \left(\begin{array}{cccc|cccc} b_1 & \cdots & \cdots & b_n & -a_1 & \cdots & \cdots & -a_n \end{array} \right) \tag{48}$$

$$\Gamma_2 = \left(\begin{array}{cccc|cccc} 0 & & & & & & & \\ 1 & \ddots & & & & & & \\ & \ddots & \ddots & & & & & \\ & & 1 & 0 & & & & \\ \hline & & & & 0 & & & \\ & & & & 1 & \ddots & & \\ & & & & & \ddots & \ddots & \\ & & & & & & 1 & 0 \end{array} \right) \tag{49}$$

$$\Gamma_3 = \left(\begin{array}{cccc|cccc} 1 & 0 & \ldots & 0 & 0 & \ldots & \ldots & 0 \end{array} \right)^\top \tag{50}$$

$$\Gamma_4 = \left(\begin{array}{cccc|cccc} 0 & \ldots & \ldots & 0 & 1 & 0 & \ldots & 0 \end{array} \right)^\top \tag{51}$$

7.1.2 `FFT2FWR`

Purpose :
Build Fast Fourier Transform (size n) with a FWR object

Syntax :
```
R = FFT2FWR(n)
R = FFT2FWR(n, toSimplify)
```

Parameters :

`R`	: FWR object to represent the algorithm
`n`	: size of the FFT
`toSimplify`	: 1 (default) to make the simplification (level=1)
	: 0 to avoid simplifications

Description :
The general factorization of DFT matrices are given by the Cooley-Tukey algorithm [6, 7], also known as Fast Fourier Transform (FFT):

$$DFT_n = (DFT_r \otimes I_s)\, \mathscr{T}^n_s\, (I_r \otimes DFT_s)\, \mathscr{L}^n_r \tag{52}$$

where

- n is factorized in $n = rs$,
- $\otimes$ denotes the Kronecker product of matrices defined by $A \otimes B \triangleq (A_{k,l}B)$,

- $\mathscr{T}_s^n$ is the twiddle matrix with

$$\mathscr{T}_s^{rs} \triangleq \bigoplus_{j=0}^{r-1} \operatorname{diag}\left(\omega_n^0, \ldots, \omega_n^{s-1}\right)^{-j} \tag{53}$$

- $A \oplus B$ is the direct sum of A and B:

$$A \oplus B \triangleq \begin{pmatrix} A & \\ & B \end{pmatrix} \tag{54}$$

- and $\mathscr{L}_r^n$ is the stride permutation matrix that maps the vector elements indices j as:

$$\mathscr{L}_r^{rs} : j \mapsto \begin{cases} jr \mod rs-1 & \text{for } 0 \leqslant j \leqslant rs-2 \\ rs-1 & \text{for } j = rs-1 \end{cases} \tag{55}$$

Equation (52) depends on the factorization $n = rs$, and should be applied recursively until n is a prime number. Since DFT is a linear transform, it is possible to express it with the SIF. Let us find the SIF realization $\mathcal{R} := (J, K, L, M, N, P, Q, R, S)$ that realizes

$$Y(k) = DFT_n U(k) \tag{56}$$

with $U(k) \in \mathbb{R}^{n\times 1}$ and $Y(k) \in \mathbb{C}^{n\times 1}$.
The following proposition allows to express such a realization by applying Cooley-Tukey factorization.

Proposition 2 *Let suppose that* $\mathcal{R}_1 := (J_1, ., L_1, ., N_1, ., ., ., S_1)$ *and* $\mathcal{R}_2 := (J_2, ., L_2, ., N_2, ., ., ., S_2)$ *respectively realize* DFT_r *and* DFT_s, i.e.*:*

$$DFT_r : \quad \begin{cases} J_1 T_1 &= N_1 U_1(k) \\ Y_1(k) &= L_1 T_1 + S_1 U_1(k) \end{cases} \tag{57}$$

$$DFT_s : \quad \begin{cases} J_2 T_2 &= N_2 U_2(k) \\ Y_2(k) &= L_2 T_2 + S_2 U_2(k) \end{cases} \tag{58}$$

Remark that no states are needed and that S_1 *(and* S_2*) are only required if* $r = 2$ *(*$s = 2$*).* $\mathcal{R}$ *is given by* $\mathcal{R} := (J, , L, , N, , , ,)$ *with*

$$J = \begin{pmatrix} (I_r \otimes J_2) & 0 & 0 & 0 \\ -(I_r \otimes L_2) & I_n & 0 & 0 \\ 0 & -\mathscr{T}_s^n & I_n & 0 \\ 0 & 0 & -(N_1 \otimes I_s) & (J_1 \otimes I_s) \end{pmatrix} \tag{59}$$

$$L = \begin{pmatrix} 0 & 0 & -(S_1 \otimes I_s) & -(L_1 \otimes I_s) \end{pmatrix} \tag{60}$$

$$N = \begin{pmatrix} (I_r \otimes N_2)\,\mathscr{L}_r^n \\ (I_r \otimes S_2)\,\mathscr{L}_r^n \\ 0 \\ 0 \end{pmatrix} \tag{61}$$

This proposition is realized with complexFFT, that returns a FFT realization with complex coefficients. Considering that DFT_2 can be realized with $\mathcal{R} := (,,,,,,,,\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix})$, the proposition 2 can be recursively applied to obtain the SIF of DFT_{2^l} (for example, with $r = s = 2^{\frac{l}{2}}$ if l is even and with $r = 2^{\frac{l-1}{2}}$ and $s = 2$ to obtain $DFT_{2^{\frac{l+1}{2}}}$ and with $r = 2^{\frac{l-1}{2}}$ and $s = 2^{\frac{l=11}{2}}$ is l is odd). Since the proposition 2 provides rules for DFT_n realization with complex coefficients, a transformation is then required in order to obtain the DFT algorithm with real coefficients. The output will now represent the real and imaginary parts of the complex DFT outputs.

Proposition 3 *Let suppose realization $\mathcal{R}' := (J',,L',,N',,,,S')$ realizes DFT_n with complex coefficients ($J' \in \mathbb{C}^{l\times l}$, $L' \in \mathbb{C}^{n\times l}$, $N \in \mathbb{C}^{l\times n}$ and $S \in \mathbb{C}^{n\times n}$) and real inputs. Then the realization $\mathcal{R} := (J,,L,,N,,,,S)$ with*

$$J = \mathscr{L}^{2l}_l \begin{pmatrix} \Re J' & -\Im J' \\ \Im J' & \Re J' \end{pmatrix} \left(\mathscr{L}^{2l}_l\right)^\top \tag{62}$$

$$L = \mathscr{L}^{2n}_n \begin{pmatrix} \Re L' & -\Im L' \\ \Im L' & \Re L' \end{pmatrix} \left(\mathscr{L}^{2l}_l\right)^\top \tag{63}$$

$$N = \mathscr{L}^{2l}_l \begin{pmatrix} \Re N' & \Im N' \end{pmatrix} \tag{64}$$

$$S = \mathscr{L}^{2n}_n \begin{pmatrix} \Re S' & \Im S' \end{pmatrix} \tag{65}$$

realize the DFT_n with real coefficients and real outputs (the $2n$ outputs alternate real and imaginary parts of the n complex outputs of $\mathcal{R}'$).
$\Re.$ and $\Im.$ denote respectively the real and imaginary parts.

Example :
The FFT_4 is given by the following realization (after simplification):

$$Z = \left(\begin{array}{cccc:c:cccc} -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 \\ \hdashline 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right)$$

See also :
simplify, complexFFT, strideM, twiddleM

7.1.3 `implicitSS2FWS`

Purpose :
Transform an implicit State-space system into a FWS object

Syntax :
```
S = implicitSS2FWS( Aq,Bq,Cq,Dq,Eq )
S = implicitSS2FWS( Sys, Eq)
```

Parameters :

`S`	: FWR object
`Aq,Bq,Cq,Dq`	: State-space (q-operator) matrices
`Eq`	: lower triangular matrix (default = identity)
`Sys`	: state-space system (`ss` object)

Description :
The system considered is described by the equations:

$$\left\{ \begin{array}{rcl} EX(k+1) & = & A_q X(k) + B_q U(k) \\ Y(k) & = & C_q X(k) + D_q U(k) \end{array} \right. \tag{66}$$

where E is a lower triangular matrix, with 1 on the diagonal (like the J matrix). The (finite precision) equivalent system, in the implicit state-space formalism, is given by

$$Z_0 = \left(\begin{array}{c|c|c} -E & A & B \\ \hline I_n & 0 & 0 \\ \hline 0 & C & D \end{array} \right) \tag{67}$$

and equivalent realizations can be searched with the similarity

$$Z = \left(\begin{array}{c|c|c} \mathcal{Y}^{-1} & & \\ \hline & \mathcal{U}^{-1} & \\ \hline & & I_p \end{array} \right) Z_0 \left(\begin{array}{c|c|c} \mathcal{U} & & \\ \hline & \mathcal{U} & \\ \hline & & I_p \end{array} \right) \tag{68}$$

where $\mathcal{U}$ is a non-singular matrix and $\mathcal{Y}$ is chosen so that $\mathcal{Y}E\mathcal{U}$ is still lower triangular (in practice, the coefficients of the new matrix E are chosen by the optimization algorithm, and $\mathcal{Y}$ is then deduced).

See also :
SS2FWS

References :
[17] : T. Hilaire, P. Chevrel, and J. Whidborne. A unifying framework for finite wordlength realizations. IEEE Trans. on Circuits and Systems, 8(54), August 2007.

7.1.4 `Modaldelta2FWR`

Purpose :
Transform a δ-based modal realization into a FWR object , and this realization can be relaxed L_2-scaled or not.

Syntax :
```
R = Modaldelta2FWR( SYS, Delta, isDeltaExact )
R = Modaldelta2FWR( Aq, Bq, Cq, Dq, Delta, isDeltaExact )
```

Parameters :

`R`	: FWR object
`SYS`	: `ss` object
`Aq,Bq,Cq,Dq`	: State-space (q-operator) matrices
`Delta`	: Vector of Δ_i. If they are not given, a relaxed L_2-scaling is performed (Δ_k then are induced)
`isDeltaExact`	: 1 if the vector of Δ_i is exactly implemented
	: 0 (default value) else

Description :
Let consider the following transfer function given and its related modal realization (Λ, B, C, D):

$$\begin{aligned} H(z) &= D + C(zI - \Lambda)^{-1}B \\ &= D + \sum_{i=1}^{n} \frac{c_i bi}{1 - \lambda_i z^{-1}} \end{aligned} \tag{69}$$

with $\lambda_i \neq \lambda_j$ for all $i \neq j$ so that Λ may be chosen as a diagonal matrix.
The modal representation is not unique since B and C may be scaled and the diagonal elements of Λ may be permuted in different ways so as to to produce the same transfer function. One invariant however is that is decouples the dynamic modes λ_i and is closely related to partial-fraction expansion of $H(z)$. Rather to diagonalize the A-matrix, it is preferred in the sequel to combine the complex-conjugate pole-pairs to form a real "block-diagonal" section in which Λ has two-by-two real matrices along its diagonal as follows:

$$\Lambda = \begin{pmatrix} \alpha_1 & \beta_1 & & & & & \\ \beta_2 & \alpha_2 & & & & & \\ & & \alpha_3 & \beta_3 & & & \\ & & \beta_4 & \alpha_4 & & & \\ & & & & \ddots & & \\ & & & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & & \beta_n & \alpha_n \end{pmatrix} \tag{70}$$

where α_i and β_i are linked to the real part and the imaginary part of the i^{th} pole, respectively. If the i^{th} pole is real, then $\beta_i = 0$; if the i^{th} and $(i+1)^{th}$

poles are complex-conjugate, then $\alpha_i = \alpha_{i+1}$ and $\beta_i = -\beta_{i+1} = Im(\lambda_i)$.
The system considered is described by the equations

$$\left\{ \begin{array}{rcl} \delta[X(k)] & = & A_\delta X(k) + B_\delta U(k) \\ Y(k) & = & C_\delta X(k) + D_\delta U(k) \end{array} \right. \tag{71}$$

where the δ-operator is defined by

$$\delta_i \triangleq \frac{q-1}{\Delta_i} \tag{72}$$

and Δ_i is a strictly positive constant.
The (finite precision) equivalent system, in the implicit state-space formalism, is given by

$$\begin{pmatrix} I_n & 0 & 0 \\ -\Delta & I_n & 0 \\ 0 & 0 & I_p \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & A_\delta & B_\delta \\ 0 & I_n & 0 \\ 0 & C_\delta & D_\delta \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{73}$$

where

$$\Delta = \mathrm{diag}(\Delta_1 \ \cdots \ \Delta_n)$$

If the system is given in classical state-space (`ss` object), the equivalent δ-realization is obtained with :

$$A_\delta = \Delta^{-1}(\Lambda - I_n), \quad B_\delta = \Delta^{-1}B, \quad C_\delta = C, \quad D_\delta = D \tag{74}$$

The `isDeltaExact` parameter determines W_K.

See also :
Modalrho2FWR

References :
[40] R. Middleton and G. Goodwin, Digital Control and Estimation, a unified approach, Prentice-Hall International Editions, 1990.
[8] Y. Feng, P. Chevrel, and T. Hilaire. A practival strategy of an efficient and sparse fwl implementation of lti filters. In submitted to ECC'09, 2009.

7.1.5 `Modalrho2FWR`

Purpose :
Transform a ρ-modal realization into a FWR object, where the parameter Δ is reserved for relaxed L_2-scaling

Syntax :
```
R = Modalrho2FWR( SYS, Gamma, Gamma, isGammaExact )
R = Modalrho2FWR( Aq, Bq, Cq, Dq, Gamma, isGammaExact )
```

Parameters :

`R`	: FWR object
`SYS`	: Initial classical q-state-space system to be converted
`Aq,Bq,Cq,Dq`	: State-space (q-operator) matrices
`Gamma`	: Vector of $gamma_i$ parameters
`isGammaExact`	: 1 (default value) if we consider that the vector of γ_i is exactly implemented
	: 0 else

Description :
The modal representation applied here is the same as that used in `Modaldelta2FWR` (see the details therein), while Δ is reserved for relaxed L_2-scaling. Define the following series of 1^{st} polynomial operators, named ρ-operators:

$$\rho_i = \frac{q - \gamma_i}{\Delta_i}, \quad \forall i = 1, 2, \cdots, n \tag{75}$$

with α_i and $\Delta_i > 0$ are two sets of constants to determine. The particular choice $\alpha_i = 0$ and $\Delta_i = 1$ (resp. $\alpha_i = 1$) leads to the shift operator (resp. the δ-operator). The specialized implicit form related to the ρ-operator has the particular structure:

$$\begin{pmatrix} I & 0 & 0 \\ -\Delta & I & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} T_{k+1} \\ X_{k+1} \\ Y_k \end{pmatrix} = \begin{pmatrix} 0 & A_\rho & B_\rho \\ 0 & \gamma & 0 \\ 0 & C_\rho & D_\rho \end{pmatrix} \begin{pmatrix} T_k \\ X_k \\ U_k \end{pmatrix} \tag{76}$$

The condition of keeping equivalece is given as below:

$$A_\rho = \Delta^{-1}(\Lambda - 1), B_\rho = \Delta^{-1}B, C_\rho = C \text{ and } D_\rho = D \tag{77}$$

$$\Delta = \text{diag}(\Delta_1 \ \cdots \ \Delta_n), \quad \gamma = \text{diag}(\gamma_1 \ \cdots \ \gamma_n) \tag{78}$$

The `isGammaExact` parameters determines W_P.

See also :
`Modalrho2FWS`, `SSrho2FWR`

7.1.6 `Modalrho2FWS`

Purpose :
Transform a ρ-modal realization into a FWS object, where the parameter Δ is reserved for relaxed L_2-scaling

Syntax :
```
S = Modalrho2FWS( SYS, Gamma, Gamma, isGammaExact )
S = Modalrho2FWS( Aq, Bq, Cq, Dq, Gamma, isGammaExact )
```

Parameters :

`S`	: FWS object
`SYS`	: Initial classical q-state-space system to be converted
`Aq,Bq,Cq,Dq`	: State-space (q-operator) matrices
`Gamma`	: Vector of $gamma_i$ parameters
`isGammaExact`	: 1 (default value) if we consider that the vector of γ_i is exactly implemented
	: 0 else

Description :
`Modaldelta2FWR` (see the details therein), while Δ is reserved for relaxed L_2-scaling. Define the following series of 1^{st} polynomial operators, named ρ-operators:

$$\rho_i = \frac{q - \gamma_i}{\Delta_i}, \quad \forall i = 1, 2, \cdots, n \tag{79}$$

with α_i and $\Delta_i > 0$ are two sets of constants to determine. The particular choice $\alpha_i = 0$ and $\Delta_i = 1$ (resp. $\alpha_i = 1$) leads to the shift operator (resp. the δ-operator). The specialized implicit form related to the ρ-operator has the particular structure:

$$\begin{pmatrix} I & 0 & 0 \\ -\Delta & I & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} T_{k+1} \\ X_{k+1} \\ Y_k \end{pmatrix} = \begin{pmatrix} 0 & A_\rho & B_\rho \\ 0 & \gamma & 0 \\ 0 & C_\rho & D_\rho \end{pmatrix} \begin{pmatrix} T_k \\ X_k \\ U_k \end{pmatrix} \tag{80}$$

The condition of keeping equivalece is given as below:

$$A_\rho = \Delta^{-1}(\Lambda - 1), B_\rho = \Delta^{-1}B, C_\rho = C \text{ and } D_\rho = D \tag{81}$$

$$\Delta = \text{diag}(\Delta_1 \ \cdots \ \Delta_n), \quad \gamma = \text{diag}(\gamma_1 \ \cdots \ \gamma_n) \tag{82}$$

The `isGammaExact` parameters determines W_P. All equivalent δ-state-space realizations (with same size) can be obtained by modification of operator which is achieved by choosing different γ_i. So there are only one parameter in the structuration's definition, namely γ.

See also :
`Modalrho2FWR`

References :
[8] Y. Feng, P. Chevrel, and T. Hilaire. A practival strategy of an efficient and sparse FWL implementation of LTI filters. Submitted to ECC'09, 2009.

7.1.7 `Observer2FWR`

Purpose :
Transform an Observer-State-Feedback form in FWR object

Syntax :
`R = Observer2FWR( Sysp, Kc, Kf, Q )`

Parameters :

`R`	: FWR object
`Sysp`	: plant to be controlled (`ss` object)
`Kf, Kc, Q`	: observer-state-feedback parameters

Description :
The system considered is described by the equations:

$$\left\{ \begin{array}{rcl} X(k+1) & = & A_p X(k) + B_p U(k) + K_f (Y(k) - C_p X(k)) \\ U(k) & = & -K_c X(k) + Q(Y(k) - C_p X(k)) \end{array} \right. \tag{83}$$

where A_p, B_p and C_p are the state-space matrices of the plant, $U(k)$ the p input of the plant and $Y(k)$ its m outputs.
K_f, K_c and Q are the parameters of the controller (see [2] for more details).
The (finite precision) equivalent system, in the implicit state-space formalism, is given by

$$Z_0 = \left(\begin{array}{cc:cc} I_p & 0 & -C_p & I_p \\ -Q & I_m & -K_c & 0 \\ \hdashline K_f & B_p & A_p & 0 \\ \hdashline 0 & I_m & 0 & 0 \end{array} \right) \tag{84}$$

References :
[15] T. Hilaire, P. Chevrel, and Y. Trinquet. Implicit state-space representation : a unifying framework for FWL implementation of LTI systems. Proc. of the 16th IFAC World Congress. Elsevier, July 2005.
[2] D. Alazard, C. Curres, P. Apkarian, M. Gauvrit, and G. Ferreses. Robustesse et Commande Optimale. Cepadues Edition, 1999.

7.1.8 `OpModalrho2FWR`

Purpose :
Optimal ρ-based modal realization under FWR structure , in which the parameter Δ is reserved for relaxed L_2-scaling and implemented exactly, while the Parameter γ is optimized analytically and is supposed to be coded exactly

Syntax :
`R = OpModalrho2FWR( SYS, Wcii )`
`R = OpModalrho2FWR( Aq, Bq, Cq, Dq, Wcii )`

Parameters :

`R`	: FWR object
`SYS`	: State-space object
`Aq,Bq,Cq,Dq`	: State-space (q-operator) matrices
`Wcii`	: diagonal controllabily gramian desired values. If `Wcii` is empty, no scaling is applied

Description :
The realization's structure applied here is the same as what is used for `Modaldelta2FWR`. Δ is reserved for relaxed L_2-scaling, and the choice of γ is given:

$$\gamma_i = \begin{cases} \alpha_i + \dfrac{\beta_i \left(W_{cX}\right)_{i+1,i}}{\left(W_{cX}\right)_{i,i}}, & \text{i is odd;} \\ \alpha_i + \dfrac{\beta_i \left(W_{cX}\right)_{i,i-1}}{\left(W_{cX}\right)_{i,i}}, & \text{i is even.} \end{cases} \tag{85}$$

where W_{cX} is the controllability gramian associated with the state such as:

$$W_{cX} = A_Z W_{cX} A_Z^T + B_Z B_Z^T \tag{86}$$

See also :
`Modalrho2FWR`

References :
[8] Y. Feng, P. Chevrel, and T. Hilaire. A practival strategy of an efficient and sparse FWL implementation of LTI filters. In submitted to ECC'09, 2009.

7.1.9 `OpModalrho2FWS`

Purpose :

Syntax :

7.1.10 `rhoDFIIt2FWR`

Purpose :
Build a ρ-DFIIt realization (ρ Direct Form II transposed, according to Li and Zhao's work). This realization can be L_2-scaled or not

Syntax :
```
[R1, R2] = rhoDFIIt2FWR( H, gamma, isGammaExact, delta, isDeltaExact)
```

Parameters :

`R1`	: ρDFIIt realization (FWR object)
`R2`	: equivalent q-state-space (sparse realization) FWR object
`H`	: `tf` object
`gamma`	: vector of parameters γ_k
`isGammaExact`	: (default: true) boolean to express if γ are exactly represented or not
`delta`	: vector or parameters Δ_k : if they are not given, a L_2-scaling is performed (Δ_k then are induced) : if they are negative, a relaxed-$L2$-scaling is performed (Δ_k then are induced)
`isDeltaExact`	: (default: false) boolean to express if Δ_k are exactly represented or not

Description :
The considered system `H` is re-parameterized as follows:

$$H(z) = \frac{\beta_0 + \beta_1 \varrho_1^{-1} + \ldots + \beta_{n-1}\varrho_{n-1}^{-1} + \beta_n \varrho_n^{-1}}{1 + \alpha_1 \varrho_1^{-1} + \ldots + \alpha_{n-1}\varrho_{-1}^{-n+1} + \alpha_n \varrho_n^{-1}} \tag{87}$$

where

$$\varrho_i(z) \triangleq \prod_{j=1}^{i} \rho_j(z) \qquad 1 \leqslant i \leqslant n \tag{88}$$

$$\rho_i(z) \triangleq \frac{z - \gamma_i}{\Delta_i} \qquad 1 \leqslant i \leqslant n \tag{89}$$

and $(\gamma_i)_{1\leqslant i\leqslant n}$ and $(\Delta_i > 0)_{1\leqslant i\leqslant n}$ are two sets of constants. Equation (87) can be, for example, implemented with a transposed direct form II with operators $\left(\rho_i^{-1}\right)_{1\leqslant i\leqslant n}$, see figures 5 and 6.
Clearly, when $\gamma_i = 0, \Delta_i = 1 \quad (1 \leqslant i \leqslant n)$, fig 5 is the conventional transposed direct form II, and, with $\gamma_i = 1, \Delta_i = \Delta \quad (1 \leqslant i \leqslant n)$, one gets the δ transposed direct form II.

L_2-scaling According to [34], the scaling is reached iff $d_k^2 (W_c)_{kk} = 1$, with d_k are the diagonal parameters of the transformation matrix, so such that $d_k = \pi_{i=1}^{k} \Delta_m$.
The conditions gives

$$\begin{cases} \Delta_1 &= \sqrt{(W_c)_{1,1}} \\ \Delta_k &= \sqrt{\frac{(W_c)_{k,k}}{(W_c)_{k-1,k-1}}}, \qquad k \geqslant 2 \end{cases}$$

L_2-scaling Now, the conditions are changed in $1 \leqslant d_k^2 (W_c)_{kk} < 4$.
So, d_k should satisfy

$$d_k = \frac{2^{\mathscr{F}_2(\sqrt{(W_c)_{k,k}})}}{\sqrt{(W_c)_{k,k}}}$$

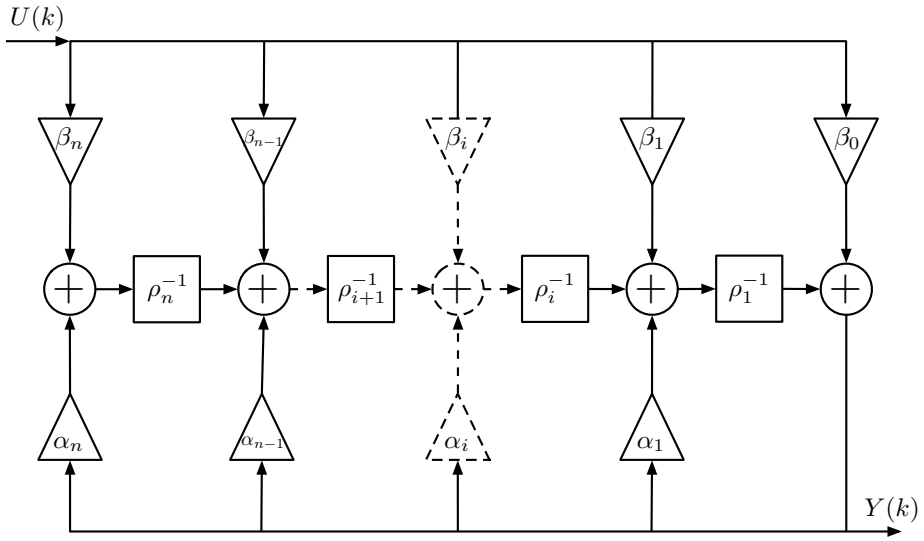

Figure 5: Generalized ρ Direct Form II

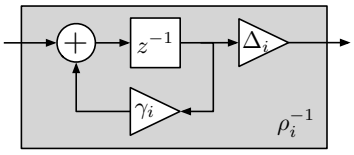

Figure 6: realization of operator ρ_i^{-1}

Since $\Delta_k = \frac{d_{k-1}}{d_k}$, then

$$\left\{ \begin{array}{rcl} \Delta_1 & = & \sqrt{(W_c)_{1,1}} 2^{-\mathscr{F}_2(\sqrt{(W_c)_{1,1}})} \\ \Delta_k & = & \sqrt{\frac{(W_c)_{k,k}}{(W_c)_{k-1,k-1}}} 2^{\mathscr{F}_2(\sqrt{(W_c)_{k-1,k-1}}) - \mathscr{F}_2(\sqrt{(W_c)_{k,k}})}, \quad k \geqslant 2 \end{array} \right.$$

See also :
rhoDFIIt2FWS

References :
[34] G. Li and Z. Zhao. On the generalized DFIIt structure and its state-space realization in digital filter implementation. IEEE Trans. on Circuits and Systems, 51(4):769–778, April 2004

7.1.11 rhoDFIIt2FWS

Purpose :
Build a ρ-DFIIt structuration (ρ Direct Form II transposed, according to Li and Zhao's work) this realization can be L_2-scaled or not

Syntax :
```
[S1, S2] = rhoDFIIt2FWS( H, gamma, isGammaExact, delta, isDeltaExact)
```

Parameters :

R1	: ρDFIIt structuration (FWS object)
R2	: equivalent q-state-space (sparse realization) FWS object
H	: tf object
gamma	: vector of parameters γ_k
isGammaExact	: (default: true) boolean to express if γ are exactly represented or not
delta	: vector or parameters Δ_k : if they are not given, a L_2-scaling is performed (Δ_k then are induced)
isDeltaExact	: (default: false) boolean to express if Δ_k are exactly represented or not

See also :
rhoDFIIt2FWR

References :
[34] G. Li and Z. Zhao. On the generalized DFIIt structure and its state-space realization in digital filter implementation. IEEE Trans. on Circuits and Systems, 51(4):769–778, April 2004

7.1.12 SS2FWR

Purpose :
Transform a classical state-space system (`ss` object) into a FWR object

Syntax :
```
R = SS2FWR( Aq,Bq,Cq,Dq )
R = SS2FWR( Sys)
```

Parameters :
`R` : realization (FWR object)
`Aq,Bq,Cq,Dq` : (classical) state-space matrices
`Sys` : state-space system (`ss` object)

Description :
The system considered is described by the equations

$$\left\{ \begin{array}{rcl} X(k+1) & = & A_q X(k) + B_q U(k) \\ Y(k) & = & C_q X(k) + D_q U(k) \end{array} \right. \tag{90}$$

The (finite precision) equivalent system, in the implicit state-space formalism, is given by

$$\begin{pmatrix} . & . & . \\ . & I_n & 0 \\ . & 0 & I_m \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} . & . & . \\ . & A_q & B_q \\ . & C_q & D_q \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{91}$$

or

$$Z = \begin{pmatrix} . & . & . \\ . & A_q & B_q \\ . & C_q & D_q \end{pmatrix} \tag{92}$$

This is a particular case without any intermediate variables ($l = 0$).

See also :
SS2FWS

7.1.13 SS2FWS

Purpose :
Transform a classical state-space system (`ss` object) into a FWS object

Syntax :
```
S = SS2FWS( Aq,Bq,Cq,Dq )
S = SS2FWS( Sys)
```

Parameters :

`S`	: structuration (FWS object)
`Aq,Bq,Cq,Dq`	: (classical) state-space matrices
`Sys`	: state-space system (`ss` object)

Description :
The system considered is described by the equations

$$\left\{ \begin{array}{rcl} X(k+1) & = & A_q X(k) + B_q U(k) \\ Y(k) & = & C_q X(k) + D_q U(k) \end{array} \right. \tag{93}$$

The (finite precision) equivalent system, in the implicit state-space formalism, is given by

$$\begin{pmatrix} . & . & . \\ . & I_n & 0 \\ . & 0 & I_m \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} . & . & . \\ . & A_q & B_q \\ . & C_q & D_q \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{94}$$

or

$$Z = \begin{pmatrix} . & . & . \\ . & A_q & B_q \\ . & C_q & D_q \end{pmatrix} \tag{95}$$

This is a particular case without any intermediate variables ($l = 0$).
All the state-space equivalent realizations (with same size) are given by the state-space systems $(T^{-1}A_qT, T^{-1}B_q, C_qT, D_q)$, where T is a nonsingular matrix. So there is one parameter in the structuration's definition, named `T`. The equivalent realizations are obtained with $\mathcal{U} = T$, $\mathcal{Y} = \mathcal{W} = I_l$. If Z_0 is the initial realization, the other realizations are obtained with

$$Z = \begin{pmatrix} I_l & & \\ & T^{-1} & \\ & & I_p \end{pmatrix} Z_0 \begin{pmatrix} I_l & & \\ & T & \\ & & I_m \end{pmatrix} \tag{96}$$

See also :
SS2FWR

7.1.14 `SSdelta2FWR`

Purpose :
Transform a δ-state-space realization into a FWR object

Syntax :

```
R = SSdelta2FWR( Ad,Bd,Cd,Dd, Delta, isDeltaExact )
R = SSdeltas2FWR( Sysq, Delta, isDeltaExact )
```

Parameters :

`R`	: FWR object
`Ad,Bd,Cd,Dd`	: State-space (delta-operator) matrices
`Sysq`	: initial q-state-space system, to be converted in δ-state-space
`Delta`	: Δ parameter of the δ-realization
`isDeltaExact`	: 1 (default value) if Δ is exactly implemented
	: 0 else

Description :
The system considered is described by the equations

$$\begin{cases} \delta[X(k)] &= A_\delta X(k) + B_\delta U(k) \\ Y(k) &= C_\delta X(k) + D_\delta U(k) \end{cases} \tag{97}$$

where the δ-operator is defined by

$$\delta \triangleq \frac{q-1}{\Delta} \tag{98}$$

and Δ is a strictly positive constant[2].
The (finite precision) equivalent system, in the implicit state-space formalism, is given by

$$\begin{pmatrix} I_n & 0 & 0 \\ -\Delta I_n & I_n & 0 \\ 0 & 0 & I_p \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & A_\delta & B_\delta \\ 0 & I_n & 0 \\ 0 & C_\delta & D_\delta \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{99}$$

If the system is given in classical state-space (`ss` object), the equivalent δ-realization is obtained with :

$$A_\delta = \frac{A_q - I_n}{\Delta}, \quad B_\delta = \frac{B_q}{\Delta}, \quad C_\delta = C_q, \quad D_\delta = D_q \tag{100}$$

The `isDeltaExact` parameter determines W_K.

See also :
SSdelta2FWS

References :
[40] R. Middleton and G. Goodwin, Digital Control and Estimation, a unified approach, Prentice-Hall International Editions, 1990.

7.1.15 `SSdelta2FWS`

Purpose :
Transform a δ-state-space realization into a FWS object

[2] In [40], Δ corresponds to the sampling period, but this constraint is removed in [9]

Syntax :
`S = SSdelta2FWR( Ad,Bd,Cd,Dd, Delta, isDeltaExact )`
`S = SSdeltas2FWR( Sysq, Delta, isDeltaExact )`

Parameters :

`S`	: FWR object (structuration)
`Ad,Bd,Cd,Dd`	: State-space (δ-operator) matrices
`Sysq`	: initial q-state-space system, to be converted in δ-state-space
`Delta`	: Δ parameter of the δ-realization
`isDeltaExact`	: 1 (default value) if Δ is exactly implemented
	: 0 else

Description :
The δ-based state-space structure is presented in SSdelta2FWR (see details therein). All the δ-state-space equivalent realizations (with same size) are given by the δ-state-space systems

$$(T^{-1}A_\delta T, T^{-1}B_\delta, C_\delta T, D_\delta),$$

where T is a nonsingular matrix. So there is one parameter in the structuration's definition, named `T`. The equivalent realizations are obtained with $\mathcal{U} = \mathcal{W} = T$, $\mathcal{Y} = T^{-1}$. If Z_0 is the initial realization, the other realizations are obtained with

$$Z = \begin{pmatrix} T^{-1} & & \\ & T^{-1} & \\ & & I_p \end{pmatrix} Z_0 \begin{pmatrix} T & & \\ & T & \\ & & I_m \end{pmatrix} \tag{101}$$

See also :
SSdelta2FWR

7.1.16 SSrho2FWR

Purpose :
Transform a ρ-based state-space realization into a FWR object

Syntax :
`R = SSrho2FWR( Arho, Brho, Crho, Drho, Gamma, isGammaExact, Delta, isDeltaExact )`
`R = SSrho2FWR( Sysq, Gamma, isGammaExact, Delta, isDeltaExact )`

Parameters :

`R`	: FWR object
`Arho, Brho, Crho, Drho`	: State-space (ρ-operator) matrices
`Sysq`	: Initial classical q-state-space system, to be converted in ρ-operator realization
`Gamma`	: Vector of γ_i parameters
`isGammaExact`	: 1 (default value) if we consider that the vector of γ_i is exactly implemented
	: 0 else
`Delta`	: Vector of Δ_i
`isDeltaExact`	: 1 (default value) if the vector of Δ_i is exactly implemented
	: 0 else

Description :
The system considered is described by the equations as follows:

$$\left\{ \begin{array}{rcl} \rho[X(k)] & = & A_\rho X(k) + B_\rho U(k) \\ Y(k) & = & C_\rho X(k) + D_\rho U(k) \end{array} \right. \tag{102}$$

where the ρ-operator is defined by

$$\rho_i \triangleq \frac{q - \gamma_i}{\Delta_i}, \quad 1 \leqslant i \leqslant n \tag{103}$$

with γ_i and $\Delta_i > 0$ are two constants to determine. The particular choice γ_i=0 and $\Delta_i = 1$ (resp. γ_i=1) leads to the shift operator q (resp. the δ-operator).
The (finite precision) equivalent system, in the implicit state-space formalism, is given by

$$\begin{pmatrix} I_n & 0 & 0 \\ -\Delta & I_n & 0 \\ 0 & 0 & I_p \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & A_\rho & B_\rho \\ 0 & \gamma & 0 \\ 0 & C_\rho & D_\rho \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix} \tag{104}$$

where

$$\Delta = \mathrm{diag}(\Delta_1 \ \cdots \ \Delta_n), \quad \gamma = \mathrm{diag}(\gamma_1 \ \cdots \ \gamma_n)$$

If the system is given in classical state-space (`ss` object), the equivalent ρ-realization is obtained with :

$$A_\rho = \Delta^{-1}(A_q - \gamma), \quad B_\rho = \Delta^{-1} B_q, \quad C_\rho = C_q, \quad D_\rho = D_q \tag{105}$$

The `isDeltaExact` and the `isGammaExact` parameters determine respectively W_K and W_P.

See also :
SSrho2FWS, SSdelta2FWR

7.1.17 `SSrho2FWS`

Purpose :
Transform a ρ-state-space realization into a FWS object

Syntax :
```
S = SSrho2FWS( Arho, Brho, Crho, Drho, Gamma, isGammaExact, Delta,
isDeltaExact )
S = SSrho2FWS( Sysq, Gamma, isGammaExact, Delta, isDeltaExact )
```

Parameters :

`S`	: FWR object (structuration)
`Arho, Brho, Crho, Drho`	: State-space (ρ-operator) matrices
`Sysq`	: initial classical state-space system, to be converted in ρ-operator realization
`Gamma`	: Vector of γ_i parameters
`isGammaExact`	: 1 (default value) if we consider that the vector of γ_i is exactly implemented
	: 0 else
`Delta`	: Vector of Δ_i
`isDeltaExact`	: 1 (default value) if the vector of Δ_i is exactly implemented
	: 0 else

Description :
The ρ-based state-space structure is presented in funcName[@FWR/SSrho2FWR]SSrho2FWR (see details therein). All equivalent δ-state-space realizations (with same size) can be obtained by two methods, namely, the similarity transformation which is specified by a nonsingular matrix T such that the corresponding ρ-state-space systems, after the coordinate transformation, can be described as $(T^{-1}A_\rho T, T^{-1}B_\rho, C_\rho T, D_\rho)$ and the modification of operator which is achieved by choosing different Δ_i and/or γ_i. So there are by consequence three parameters in the structuration's definition, `T`, Δ and γ.

See also :
SSrho2FWR

7.2 Private functions

Some functions internally used are described here:

complexFFT	Create FFT with complex coefficients (for FFT2FWR)
strideM	Create a stride permutation matrix (for FFT2FWR)
twiddleM	Create a diagonal matrix of twiddle factors (for FFT2FWR)

7.2.1 canon_modal

Purpose :

Syntax :

7.2.2 `complexFFT`

Purpose :
Create FFT with complex coefficients (for FFT2FWR) (matrices J,L,N and S are used)

Syntax :
```
[J,L,N,S] = complexFFT(n, toSimplify)
```

Parameters :
`J,L,N,S` : FWR matrices with complex coefficients
`n` : size of the FFT
`toSimplify` : 1 (default) if the simplification (level=1) is done at each recursive step

Description :
This function is called by `FFT2FWR`

See also :
`FFT2FWR`, `simplify`, `strideM`, `twiddleM`

7.2.3 `strideM`

Purpose :
Create a stride permutation matrix (for FFT2FWR)

Syntax :
```
L = strideM(r,n)
```

Parameters :
`L` : stride permutation matrix
`r,n` : parameters

Description :
Create a stride permutation matrix. This function is used by `FFT2FWR`.
This matrix $\mathscr{L}_r^n$ is the stride permutation matrix that maps the vector elements indices j as:

$$\mathscr{L}_r^{rs} : j \mapsto \begin{cases} jr \mod rs-1 & \text{for } 0 \leqslant j \leqslant rs-2 \\ rs-1 & \text{for } j = rs-1 \end{cases} \tag{106}$$

See also :
`FFT2FWR`

7.2.4 `twiddleM`

Purpose :
Create a diagonal matrix of twiddle factors (for FFT2FWR)

Syntax :
`T = twiddleM(r,n)`

Parameters :
`T` : twiddle factors matrix
`r,n` : n decomposition : $n = rs$

Description :
Create a diagonal matrix of twiddle factors. This function is used by FFT2FWR. This matrix $\mathscr{T}_s^n$ is defined by

$$\mathscr{T}_s^{rs} \triangleq \bigoplus_{j=0}^{r-1} \operatorname{diag}\left(\omega_n^0, \ldots, \omega_n^{s-1}\right)^{-j} \tag{107}$$

See also :
FFT2FWR

7.3 FWR class methods

algorithmCfloat	Return the algorithm associated to this realization.
algorithmLaTeX	Return the pseudocode algorithm described in LaTeX
computationalCost	Give the number of additions and multiplications implied in the realization
computeW	Compute (or update) the weighting matrices (W_J to W_S, and W_Z) of a FWR object
display	Display the realization (dimensions and Z)
double	Convert FWR object to double (return Z matrix)
FWR	FWR class' constructor
FWRmat2LaTeX	Display a matrix (Z or a sensitivity matrix) in LaTeX(with `pmat`
get	Get some properties of a FWR object
implementLaTeX	Return the associated fixed-point algorithm described in LaTeX
implementMATLAB	Create the associated fixed-point algorithm in Matlab language
implementVHDL	Create the associated fixed-point algorithm in VHDL.
l2scaling	Perform a L_2-scaling on the FWR
MsensH	Compute the open-loop transfer function sensitivity measure (and the
MsensH_cl	Compute the closed-loop transfer function sensitivity measure (and the
MsensPole	Compute the open-loop pole sensitivity measure
MsensPole_cl	Compute the closed-loop pole sensitivity measure
Mstability	Compute the closed-loop pole sensitivity stability related measure
mtimes	Multiply two FWR (put them in cascade)
ONP	Compute the Output Noise Power for a FWR object with Roundoff Before
plus	add two FWR object (put them in parallel)
quantized	Return the quantized realization, according to a fixed-point implementation scheme
realize	Numerically compute the outputs, states and intermediate variables with a given input U.
relaxedl2scaling	Perform a relaxed-L_2-scaling on the FWR.
RNG	Compute the open-loop Roundoff Noise Gain
RNG_cl	Compute the closed-loop Roundoff Noise Gain
set	Set some properties of a FWR object
setFPIS	Set the Fixed-Point Implementation Scheme (FPIS) of an FWR object
sigma_tf	Compute the open-loop transfer function error $\sigma^2_{\Delta H}$)

simplify	Simplify (if possible) a FWR, by removing the non-necessary intermediate variables and states
size	Return the size of the FW Realization
ss	Convert a FWR object into a ss (state-space) object (equivalent state-space)
subsasgn	Subscripted assign for FWR object
subsref	Subscripted reference for FWR object
tf	Convert a FWR object into a tf object (transfer function)
TradeOffMeasure_cl	Compute a (pseudo) tradeoff closed-loop measure
transform	Perform a UYW-transformation (similarity on Z)

7.3.1 `algorithmCfloat`

Purpose :
Return the algorithm associated to this realization. The algorithm is written in C-code with float

Syntax :
```
code = algorithmCfloat( R, funcName)
```

Parameters :
`code` : resulting C-code (with float)
`R` : FWR object
`funcName` : name of the C-function (default=`myFilter`)

Description :
Transform in `C`-code with `float` the algorithm of the realization:

$$\begin{aligned} &[\text{i}] && JT(k+1) \leftarrow MX(k) + NU(k) \\ &[\text{ii}] && X(k+1) \leftarrow KT(k+1) + PX(k) + QU(k) \\ &[\text{iii}] && Y(k) \leftarrow LT(k+1) + RX(k) + SU(k) \end{aligned}$$

All the operations with matrices are expanded, and null multiplications are removed, identity multiplications are simplified, etc.
The input or a pointer to the vector of inputs is given to the function. The function returns the output or a pointer to a vector of putputs.
The intermediate variables are stored in a variable `T`. The states are stored in `static` variables `xn` and `xnp` (`xnp` is not necessary if P is upper triangular), and a permutation of the vector (a permutation of the pointer to the vector) is performed for the next call.

Example :

```
>> algorithmCfloat(R)
ans =
float myFilter( float u)
{
// states
static float* xn = (float*) calloc( 8*sizeof(float));
// intermediate variables
float T = -0.6630104844*xn[0] + 2.9240526562*xn[1] +
    -4.8512758825*xn[2]
+ 3.5897338871*xn[3] + 0.0000312390*xn[4] + 0.0001249559*xn[5]
+ 0.0001874339*xn[6] + 0.0001249559*xn[7] + 0.0000312390*u ;
// output(s)
y = T   ;
// states
xn[0] = xn[1];
xn[1] = xn[2];
xn[2] = xn[3];
xn[3] = T   ;
xn[4] = xn[5];
xn[5] = xn[6];
xn[6] = xn[7];
xn[7] = u   ;
}
```

See also :
algorithmLaTeX

7.3.2 algorithmLaTeX

Purpose :
Return the pseudocode algorithm described in LaTeX(to be used with package algorithm2e)

Syntax :
```
code = algorithmLaTeX( R)
code = algorithmLaTeX( R, caption)
```

Parameters :
`R` : FWR object
`caption` : caption used to describe the algorithm
: (default=`'Pseudocode algorithm ...'`)

Description :
Return a LaTeX-code that describe the algorithm of the realization:

$$\begin{aligned}
&[\text{i}] && JT(k+1) \leftarrow MX(k) + NU(k) \\
&[\text{ii}] && X(k+1) \leftarrow KT(k+1) + PX(k) + QU(k) \\
&[\text{iii}] && Y(k) \leftarrow LT(k+1) + RX(k) + SU(k)
\end{aligned}$$

All the operations with matrices are expanded, and null multiplications are removed, identity multiplications are simplified, etc.
The package *algorithm2e* is used.
The file `@FWR/private/myFilter.tex.template` is used as a template.

Example :
The following LaTeX-code is produced by this function:

```
\begin{algorithm}[h]
\caption{Pseudocode algorithm ...}
\KwIn{$u$: real}
\KwOut{$y$: real}
\KwData{$xn, xnp$: array [1..13] of reals}
\SetLine
\Begin{
\tcp{\emph{Intermediate variables}}
$xnp(1) \leftarrow 0.5529838718*xn(1) + -0.5379265439*xn(2) +
    0.0291718129*xn(3) + 0.6715678041*u $\;
$xnp(2) \leftarrow 0.5379265439*xn(1) + 0.0971133953*xn(2) +
    -0.3562792507*xn(3) + -0.3237597311*u $\;
$xnp(3) \leftarrow 0.0291718129*xn(1) + 0.3562792507*xn(2) +
    -0.0728567423*xn(3) + 0.0792887747*u $\;
\tcp{\emph{Outputs}}
$y      \leftarrow 0.6715678041*xn(1) + 0.3237597311*xn(2) +
    0.0792887747*xn(3) + 0.0985311609*u $\;
\tcp{\emph{Permutations}}
$xn \leftarrow xnp$\;
}
\end{algorithm}
```

And it corresponds to the algorithm 3:

See also :
algorithmCfloat, implementLaTeX

7.3.3 computationalCost

Purpose :
Give the number of additions and multiplications implied in the realization

Input: u: real
Output: y: real
Data: xn, xnp: array [1..13] of reals
begin
  // Intermediate variables
  $xnp(1) \leftarrow 0.5529838718 * xn(1) + -0.5379265439 * xn(2) +$
  $0.0291718129 * xn(3) + 0.6715678041 * u;$
  $xnp(2) \leftarrow 0.5379265439 * xn(1) + 0.0971133953 * xn(2) +$
  $-0.3562792507 * xn(3) + -0.3237597311 * u;$
  $xnp(3) \leftarrow 0.0291718129 * xn(1) + 0.3562792507 * xn(2) +$
  $-0.0728567423 * xn(3) + 0.0792887747 * u;$
  // Outputs
  $y \leftarrow 0.6715678041 * xn(1) + 0.3237597311 * xn(2) + 0.0792887747 *$
  $xn(3) + 0.0985311609 * u;$
  // Permutations
  $xn \leftarrow xnp;$
end

Algorithm 3: Pseudocode algorithm ...

Syntax :
`[add, mul] = computationalCost( R, tol)`

Parameters :
`add` : number of additions
`mul` : number of multiplications
`R` : FWR object
`tol` : tolerance (default value = 1e-8)

Description :
The number of additions and multiplications is based on the number of trivial parameters and null parameters.
The evaluation is based on the following proposition, applied on the three steps [i], [ii] and [iii] of algorithm (4) :

Proposition 4 *Let $Y \in \mathbb{R}^{a \times b}$ be a constant, and $V \in \mathbb{R}^{b \times 1}$ a variable.*
The calculus YV needs $a(b-1) - n_Y^0$ additions and $ab - n_Y^1$ multiplications, where n_Y^0 is the number of null elements of Y and n_Y^1 is the number of trivial elements (0,1,−1) of Y (these elements don't imply a multiplication)

Then, the algorithm requires $(l+n+p)(l+m+n-1) - l - n_Z^0$ additions and $(l+n+p)(l+m+n) - n_Z^1$ multiplications.

7.3.4 `computeW`

Purpose :
Compute (or update) the weighting matrices (W_J to W_S, and W_Z) of a FWR

object

Syntax :
`R = computeW(R,tol)`

Parameters :
`R` : FWR object
`tol` : tolerance (default=1e-8) (maximal distance to 0, -1 or +1)

Description :
For X in $\{J, K, L, M, N, P, Q, R, S\}$ and $X = Z$, the weighting matrices W_J, W_K, W_L, W_M, W_N, W_P, W_Q, W_R, W_S and W_Z are computed according to

$$(W_X)_{ij} = \begin{cases} 0 & \text{if } |X_{ij}| < \epsilon \text{ or } |X_{ij}+1| < \epsilon \text{ or } |X_{ij}-1| < \epsilon \\ 1 & \text{else} \end{cases} \tag{108}$$

Where ϵ is the tolerance ($1e-8$ as default value). Here, the proximity to 0 and ± 1 are considered (the other power of 2 are not considered).

7.3.5 `display`

Purpose :
Display the realization (dimensions and Z)

Syntax :
`display(R)`

Parameters :
`R` : FWR object

Description :
Display the dimensions (inputs, outputs, states and intermediate variables) and Z.

Example :

```
R has 1 input, 1 output, 4 states, and 0 intermediate variable.
3.5897e+00  -1.2128e+00   3.6551e-01  -1.6575e-01   1.5625e-02
4.0000e+00            0            0            0            0
0 2.0000e+00            0            0            0
0            0   5.0000e-01            0            0
1.5174e-02   5.7416e-04   1.7304e-03   1.6844e-04   3.1239e-05
```

See also :
display

7.3.6 `double`

Purpose :
Convert FWR object to double (return Z matrix)

Syntax :
```
d = double(R)
```

Parameters :
`d` : double
`R` : FWR object

Description :
return the Z matrix.

See also :
display

7.3.7 `FWR`

Purpose :
FWR class' constructor

Syntax :
```
R = FWR()
R = FWR(R1)
R = FWR(J,K,L,M,N,P,Q,R,S, fp, block)
```

Parameters :

`R`	: FWR object created
`R1`	: FWR object to be copied
`J,K,...,S`	: matrices of the realization
`fp`	: fixed-point or floating-point representation
	: `'fixed'` (default value) or `'floating'`
`block`	: block-representation scheme. The coefficients in a same block share the same representation (same scale factor, etc...). Take the following values:
	: `'full'`: same representation for all coefficients of R
	: `'natural'` (default value): blocks are made of matrices J,K,L,M,N,P,Q,R,S
	: `'none'`: each coefficient has its own representation (according to its value)

Description :
Constructor of the `FWR` class.
It could create an empty object ($l = m = n = p = 0$), copy an object or create a `FWR` object from matrices J to S: in that case, the other parameters are deduced (only 0, 1 or -1 are considered as exactly implemented).

See also :
FWS

7.3.8 FWRmat2LaTeX

Purpose :
Display a matrix (Z or a sensitivity matrix) in LaTeX(with `pmat` package). The non trivial parameters (according to W_Z) are in bold.

Syntax :
```
S = FWRmat2LaTeX( R)
S = FWRmat2LaTeX( R, M, format,tol)
```

Parameters :
`S`	: string result (to be pasted in LaTeXsource)
`R`	: FWR object
`M`	: matrix to print in LaTeXformat
	: if M is omitted (or empty), Z is printed
	: M must have the same size as Z
`format`	: print format
	: default value : `'%0.5g'`
	: `'%.3e'` for short e format, etc...
`tol`	: tolerance to find trivial parameter (1,-1,0)
	: default value: 1e-10

Description :
Display a matrix (Z or a given matrix `M`, for example a sensitivity matrix) of a FWR object in a special LaTeXformat (with `pmat` package): the coefficients with $W_{Z(i,j)}$ non null are in bold.

Example :
The command `FWRmat2latex(R)`, where R is a FWR object, returns:

```
\begin{pmat}({|...|})
\cr\-
&\mathbf{3.7673} & \mathbf{-1.8552} & \mathbf{1.0013} & \mathbf
    {-0.91839} & \mathbf{2} \cr
&\mathbf{4} & 0 & 0 & 0 & 0 \cr
&0 & \mathbf{2} & 0 & 0 & 0 \cr
&0 & 0 & \mathbf{0.5} & 0 & 0 \cr\-
```

```
&\mathbf{0.90722} & \mathbf{-0.56715} & \mathbf{0.24114} & \
    mathbf{-0.16096} & \mathbf{0.48163} \cr
\end{pmat}
```

When compiled, this LaTeXcode produces

$$\left(\begin{array}{ccccc} & & & & \\ \hdashline \mathbf{3.7673} & \mathbf{-1.8552} & \mathbf{1.0013} & \mathbf{-0.91839} & \mathbf{2} \\ \mathbf{4} & 0 & 0 & 0 & 0 \\ 0 & \mathbf{2} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0.5} & 0 & 0 \\ \hdashline \mathbf{0.90722} & \mathbf{-0.56715} & \mathbf{0.24114} & \mathbf{-0.16096} & \mathbf{0.48163} \end{array}\right)$$

7.3.9 get

Purpose :
Get some properties of a FWR object (or list the properties if `propName` is ignored)

Syntax :
```
value = get(R, propName)
```

Parameters :

`value`	: value of the property
`R`	: FWR object
`propName`	: name of the property (string)

Description :
This function is most of the time called by `@FWR/subsref`.
The value of every field (`l`, `m`, `n` and `p` ; `J`, `K`, `L`, `M`, `N`, `P`, `Q`, `R` and `S`; `Z` ; `WJ`, `WK`, `WL`, `WM`, `WN`, `WP`, `WQ`, `WR` and `WS` ; `WZ` ; `AZ`, `BZ`, `CZ` and `AZ`) can be evaluated with this command, but `l`, `m`, `n`, `p`, `AZ`, `BZ`, `CZ` and `AZ` cannot be modified.
Changing `Z` changes fields `J` to `S`, and reciprocally (this is the same with `WZ`).

See also :
`set`, `subsref`, `subsasgn`, `get`

7.3.10 implementLaTeX

Purpose :
Return the associated fixed-point algorithm described in LaTeX(to be used with package `algorithm2e`)

Syntax :
```
code = implementLaTeX( R)
code = implementLaTeX( R, caption)
```

Parameters :

`R`	: FWR object
`caption`	: caption used to describe the algorithm
	: (default = `'Numerical fixed-point algorithm ...'`)

Description :

Return the associated fixed-point algorithm in LaTeX. It uses the package *algorithm2e*. All the wordlengths and the fixed-point positions should be first computed by adjusting the FPIS with `@FWR/setFPIS`.
The file `@FWR/private/myFilter.tex.template` is used as a template.

Example :

It returns LaTeX-code like this

```
\begin{algorithm}[h]
\caption{Numerical fixed-point algorithm ...}
\KwIn{$u$: 16 bits integer}
\KwOut{$y$: 16 bits integer}
\KwData{$xn, xnp$: array [1..13] of 16 bits integers}
\KwData{$Acc$: 32 bits integer}
\SetLine
\Begin{
\tcp{\emph{Intermediate variables}}
$Acc \leftarrow (xn(1) * 18120)$\;
$Acc \leftarrow Acc + (xn(2) * -8813)$\;
$Acc \leftarrow Acc + (xn(3) * 239)$\;
$Acc \leftarrow Acc + (u  * 11003)$\;
$xnp(1) \leftarrow Acc >> 15$\;
$Acc \leftarrow (xn(1) * 17627)$\;
$Acc \leftarrow Acc + (xn(2) * 1591)$\;
$Acc \leftarrow Acc + (xn(3) * -2919)$\;
$Acc \leftarrow Acc + (u  * -5304)$\;
$xnp(2) \leftarrow Acc >> 14$\;
$Acc \leftarrow (xn(1) * 3824)$\;
$Acc \leftarrow Acc + (xn(2) * 23349)$\;
$Acc \leftarrow Acc + (xn(3) * -2387)$\;
$Acc \leftarrow Acc + (u  * 5196)$\;
$xnp(3) \leftarrow Acc >> 15$\;
\tcp{\emph{Outputs}}
$Acc \leftarrow (xn(1) * 22006)$\;
$Acc \leftarrow Acc + (xn(2) * 5304)$\;
$Acc \leftarrow Acc + (xn(3) * 650)$\;
$Acc \leftarrow Acc + (u  * 1614)$\;
$y     \leftarrow Acc >> 14$\;
\tcp{\emph{Permutations}}
$xn \leftarrow xnp$\;
```

```
}
\end{algorithm}
```

That corresponds to the algorithm 4.

Input: u: 16 bits integer
Output: y: 16 bits integer
Data: xn, xnp: array [1..13] of 16 bits integers
Data: Acc: 32 bits integer
begin
  // Intermediate variables
  $Acc \leftarrow (xn(1) * 18120)$;
  $Acc \leftarrow Acc + (xn(2) * -8813)$;
  $Acc \leftarrow Acc + (xn(3) * 239)$;
  $Acc \leftarrow Acc + (u * 11003)$;
  $xnp(1) \leftarrow Acc >> 15$;
  $Acc \leftarrow (xn(1) * 17627)$;
  $Acc \leftarrow Acc + (xn(2) * 1591)$;
  $Acc \leftarrow Acc + (xn(3) * -2919)$;
  $Acc \leftarrow Acc + (u * -5304)$;
  $xnp(2) \leftarrow Acc >> 14$;
  $Acc \leftarrow (xn(1) * 3824)$;
  $Acc \leftarrow Acc + (xn(2) * 23349)$;
  $Acc \leftarrow Acc + (xn(3) * -2387)$;
  $Acc \leftarrow Acc + (u * 5196)$;
  $xnp(3) \leftarrow Acc >> 15$;
  // Outputs
  $Acc \leftarrow (xn(1) * 22006)$;
  $Acc \leftarrow Acc + (xn(2) * 5304)$;
  $Acc \leftarrow Acc + (xn(3) * 650)$;
  $Acc \leftarrow Acc + (u * 1614)$;
  $y \leftarrow Acc >> 14$;
  // Permutations
  $xn \leftarrow xnp$;
end

Algorithm 4: Numerical fixed-point algorithm ...

See also :
algorithmLaTeX, implementMATLAB, implementVHDL

7.3.11 implementMATLAB

Purpose :
Create the associated fixed-point algorithm in Matlab language (it uses integer to simulate fixed-point). The algorithm is written in a file (by default, in 'myFilter.m' file)

Syntax :
`implementMATLAB( R,fileName)`

Parameters :
`R` : FWR object
`fileName` : filename of the created function (default=`'myFilter'`)

Description :
Generate a Matlab file (named `'myFilter.m'` by default) that emulates the fixed-point algorithm corresponding to the realization. The rounding operations are realized by the `floor` function. All the wordlengths and the fixed-point positions should be first computed by adjusting the FPIS with `@FWR/setFPIS`. The file `@FWR/private/myFilter.m.template` is used as a template.

Example :
It creates a matlab file like

```
% initialize
u = round(2.^11.*u);
y = zeros( size(u,1), 1 );
xn = zeros(3,1);
xnp = zeros(3,1);
for i=1:size(u,1)
% intermediate variables
Acc0 = xn(1) * 18120;
Acc0 = Acc0 + xn(2) * -8813;
Acc0 = Acc0 + xn(3) * 239;
Acc0 = Acc0 + u(i) * 11003;
xnp(1) = floor( Acc0/2^15 );
Acc1 = xn(1) * 17627;
Acc1 = Acc1 + xn(2) * 1591;
Acc1 = Acc1 + xn(3) * -2919;
Acc1 = Acc1 + u(i) * -5304;
xnp(2) = floor( Acc1/2^14 );
Acc2 = xn(1) * 3824;
Acc2 = Acc2 + xn(2) * 23349;
Acc2 = Acc2 + xn(3) * -2387;
Acc2 = Acc2 + u(i) * 5196;
xnp(3) = floor( Acc2/2^15 );
% output(s)
Acc3 = xn(1) * 22006;
Acc3 = Acc3 + xn(2) * 5304;
Acc3 = Acc3 + xn(3) * 650;
Acc3 = Acc3 + u(i) * 1614;
y(i)  = floor( Acc3/2^14 );
%permutations
```

```
xn = xnp;
end
y = 2.^-11.*y;
```

It could be used to compute the fixed-point output of the associated realization, with the fixed-point algorithm.

```
>> u=10*rand(1000,1);
>> y=myFilter(u);
```

See also :
implementLaTeX, implementVHDL

7.3.12 implementVHDL

Purpose :
Create the associated fixed-point algorithm in VHDL. Two files are generated 'xxxx_entity.vhd' and 'xxx_types.vhd' where xxxx is the name given ('myFilter' by default)

Syntax :
`implementVHDL( R,fileName)`

Parameters :
`R` : FWR object
`fileName` : name of the function (default=`myFilter`)

Description :
Generate two VHDL files (named "myFilter_entity.vhd" and "myFilter_types.vhd" by default) that realizes the fixed-point algorithm corresponding to the realization.
All the wordlengths and the fixed-point positions should be first computed by adjusting the FPIS with `@FWR/setFPIS`.
The files `@FWR/private/FP_types.vhd.template` and `@FWR/private/FP_entity.vhd.template` are used as a template.

Example :
This function produces a `myFilter_types.vhdl` file

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_SIGNED.all;
-- purpose: filtering (generic fixed-point specificatin)
-- type  : sequential/arithmetic
-- inputs : u(n)
-- output : y(n)
```

```
-- author : automatically generated by
-- date  : 08-Dec-2008 18:41:00
package FP_types is
-- input data with FP format (16,4,11)
subtype datain is integer range -2**15 to 2**15-1;
-- filtered output data with FP format (16,4,11)
subtype dataout is integer range -2**15 to 2**15-1;
-- states
subtype state1 is integer range -2**15 to 2**15-1; -- format
    (16,5,10)
subtype state2 is integer range -2**15 to 2**15-1; -- format
    (16,4,11)
subtype state3 is integer range -2**15 to 2**15-1; -- format
    (16,3,12)
-- intermediate variables
end FP_types;
```

It also produces a `myFilter_entity.vhdl` file

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_arith.all;
use IEEE.STD_LOGIC_SIGNED.all;
library work;
use work.FP_types.all;
entity myFilter is
port (
rstb   : in std_logic; -- asynchronous reset asynchrone active
    low
clk    : in std_logic; -- global clock
u      : in datain; -- input data
y      : out dataout); -- filtered output
end myFilter;
architecture RTL of myFilter is
-- states
signal xn1 : state1 := 0;
signal xn2 : state2 := 0;
signal xn3 : state3 := 0;
-- intermediate variables
begin
-- output(s)
y <= (   xn1 * 22006 + xn2 * 5304 + xn3 * 650 + u * 1614) /
    2**14;
S1: process(rstb,clk)
begin
if rstb = '0' then              -- asynchronous reset
xn1 <= 0;
```

```
xn2 <= 0;
xn3 <= 0;
elsif clk'event and clk = '1' then -- rising clock edge
-- states
xn1 <= (   xn1 * 18120 + xn2 * (-8813) + xn3 * 239 + u * 11003) /
     2**15;
xn2 <= (   xn1 * 17627 + xn2 * 1591 + xn3 * (-2919) + u * (-5304)
    ) / 2**14;
xn3 <= (   xn1 * 3824 + xn2 * 23349 + xn3 * (-2387) + u * 5196) /
     2**15;
end if;
end process S1;
end RTL;
```

See also :
implementLaTeX, implementMATLAB

7.3.13 l2scaling

Purpose :
Perform a L_2-scaling on the FWR

Syntax :
```
R = l2scaling(R, Wcii)
[U,Y,W] = l2scaling(R)
```

Parameters :

`R`	: FWR object
`U,Y,W`	: transformation matrices applied on R
`Wcii`	: vector (size $(1, l+n)$) of controllability gramians desired
	: if `Wcii` is omitted, strict L_2-scaling is applied (`Wcii=ones(1,n+l)`)

Description :
Perform a L_2-scaling.
The scaling forces the transfer functions from the inputs to the states and the intermediate variables to have a unitary L_2-norm. Theses norms are given by the diagonal terms of W_c and $J^{-1}\left(NN^\top + MW_cM^\top\right)J^{-\top}$.
The L_2-scaling is a $\mathcal{UYW}$-transformation where $\mathcal{U}$ and $\mathcal{W}$ are diagonal with:

$$(\mathcal{U})_{ii} = \sqrt{(W_c)_{ii}} \tag{109}$$

$$(\mathcal{W})_{ii} = \sqrt{\left(J^{-1}\left(NN^\top + MW_cM^\top\right)J^{-\top}\right)_{ii}} \tag{110}$$

It is also possible to assign some particular values for the diagonal terms of the two gramians.

See also :
relaxedl2scaling

References :
[8] Y. Feng, P. Chevrel, and T. Hilaire. A practival strategy of an efficient and sparse FWL implementation of LTI filters. In submitted to ECC'09, 2009.

7.3.14 MsensH

Purpose :
Compute the open-loop transfer function sensitivity measure (and the sensitivity matrix)

Syntax :
```
[M MZ] = MsensH(R)
```

Parameters :
M : sensitivity measure
MZ : sensitivity matrix
R : FWR object

Description :
The open-loop transfer function sensitivity measure is defined by

$$M_{L_2}^W = \left\| \frac{\delta H}{\delta Z} \times r_Z \right\|_F^2 . \tag{111}$$

where $\frac{\delta H}{\delta Z} \in \mathbb{R}^{l+n+p \times l+n+q}$ is the *transfer function sensitivity matrix*. It is the matrix of the L_2-norm of the sensitivity of the transfer function H with respect to each coefficient $Z_{i,j}$. It is defined by

$$\left(\frac{\delta H}{\delta Z} \right)_{i,j} \triangleq \left\| \frac{\partial H}{\partial Z_{i,j}} \right\|_2 , \tag{112}$$

In SISO case, the $M_{L_2}^W$ measure is equal to

$$M_{L_2}^W = \left\| \frac{\partial H}{\partial Z} \times r_Z \right\|_2^2 \tag{113}$$

and is then an extension to the SIF of the classical state-space sensitivity measure

$$M_{L_2} \triangleq \left\| \frac{\partial H}{\partial A} \right\|_2^2 + \left\| \frac{\partial H}{\partial B} \right\|_2^2 + \left\| \frac{\partial H}{\partial C} \right\|_2^2 . \tag{114}$$

The $M_{L_2}^W$ measure can be evaluated by the following propositions

Proposition 5

$$\frac{\partial H}{\partial Z} = H_1 \circledast H_2 \tag{115}$$

where H_1 and H_2 are defined by

$$H_1 : z \mapsto C_Z(zI_n - A_Z)^{-1}M_1 + M_2 \tag{116}$$
$$H_2 : z \mapsto N_2 + N_1(zI_n - A_Z)^{-1}B_Z \tag{117}$$

with

$$M_1 \triangleq \begin{pmatrix} KJ^{-1} & I_n & 0 \end{pmatrix}, \qquad M_2 \triangleq \begin{pmatrix} LJ^{-1} & 0 & I_{p_2} \end{pmatrix}, \tag{118}$$

$$N_1 \triangleq \begin{pmatrix} J^{-1}M \\ I_n \\ 0 \end{pmatrix}, \qquad N_2 \triangleq \begin{pmatrix} J^{-1}N \\ 0 \\ I_{m_2} \end{pmatrix}. \tag{119}$$

Proposition 6 *The transfer function sensitivity matrix $\frac{\delta H}{\delta Z}$ can be computed as*

$$\left(\frac{\delta H}{\delta Z}\right)_{i,j} = \|H_1 E_{i,j} H_2\|_2 \tag{120}$$

with

$$H_1 E_{i,j} H_2 := \left(\begin{array}{cc|c} A_Z & 0 & B_Z \\ M_1 E_{i,j} N_1 & A & M_1 E_{i,j} N_2 \\ \hline M_2 E_{i,j} N_1 & C_Z & M_2 E_{i,j} N_2 \end{array}\right) \tag{121}$$

and $E_{i,j}$ is the matrix of appropriate size with all elements being 0 except the (i,j)th element which is unity.

Remark 2 *In the SISO case, the problem becomes simpler by noting that*

$$\left(\frac{\delta H}{\delta Z}\right)_{i,j} = \|(H_2 H_1)_{i,j}\|_2 \tag{122}$$

$$= \left\| \left(\begin{array}{cc|c} A_Z & 0 & B_Z \\ M_1 N_1 & A_Z & M_1 N_2 \\ \hline M_2 N_1 & C_Z & M_2 N_2 \end{array}\right)_{i,j} \right\|_2 \tag{123}$$

The $(l+n+1) \times (l+n+1)$ H_2-norm evaluations here require only $l+n+1$ Lyapunov equations to be solved .

See also :
MsensH_cl, w_prod_norm

References :
[12] T. Hilaire and P. Chevrel. On the compact formulation of thederivation of a transfer matrix with respect to another matrix. Technical Report RR-6760, INRIA, 2008.
[13] T. Hilaire, P. Chevrel, and J.-P. Clauzel. Low parametric sensitivity realization design for FWL implementation of MIMO controllers : Theory and application to the active control of vehicle longitudinal oscillations. In Proc. of Control Applications of Optimisation CAO'O6, April 2006.
[17] T. Hilaire, P. Chevrel, and J. Whidborne. A unifying framework for finite wordlength realizations. IEEE Trans. on Circuits and Systems, 8(54), August 2007.

7.3.15 `MsensH_cl`

Purpose :
Compute the closed-loop transfer function sensitivity measure (and the sensitivity matrix)

Syntax :
```
[M MZ] = MsensH_cl( R, Sysp)
```

Parameters :
`M` : sensitivity measure
`MZ` : sensitivity matrix
`R` : FWR object
`Sysp` : plant system (ss object)

Description :
The closed-loop transfer function sensitivity measure is very similar to the open-loop transfer function sensitivity. It is defined by

$$\bar{M}_{L_2}^W = \left\| \frac{\delta \bar{H}}{\delta Z} \times r_Z \right\|_F^2 . \tag{124}$$

where $\frac{\delta \bar{H}}{\delta Z} \in \mathbb{R}^{l+n+p \times l+n+q}$ is the *transfer function sensitivity matrix*. It is the matrix of the L_2-norm of the sensitivity of the transfer function $\bar{H}$ with respect to each coefficient $Z_{i,j}$. It is defined by

$$\left(\frac{\delta \bar{H}}{\delta Z} \right)_{i,j} \triangleq \left\| \frac{\partial \bar{H}}{\partial Z_{i,j}} \right\|_2 , \tag{125}$$

In SISO case, the $\bar{M}_{L_2}^W$ measure is equal to

$$\bar{M}_{L_2}^W = \left\| \frac{\partial \bar{H}}{\partial Z} \times r_Z \right\|_2^2 . \tag{126}$$

The $M_{L_2}^W$ measure can be evaluated by the following propositions[12]

Proposition 7

$$\frac{\partial \bar{H}}{\partial Z} = \bar{H}_1 \circledast \bar{H}_2 \tag{127}$$

where H_1 and H_2 are defined by

$$\bar{H}_1 : z \mapsto \bar{C}\left(zI - \bar{A}\right)^{-1} \bar{M}_1 + \bar{M}_2 \tag{128}$$

$$\bar{H}_2 : z \mapsto \bar{N}_1\left(zI - \bar{A}\right)^{-1} \bar{B} + \bar{N}_2 \tag{129}$$

with

$$\bar{M}_1 = \begin{pmatrix} B_2 L J^{-1} & 0 & B_2 \\ K J^{-1} & I_n & 0 \end{pmatrix} \qquad \bar{N}_1 = \begin{pmatrix} J^{-1} N C_2 & J^{-1} M \\ 0 & I_n \\ C_2 & 0 \end{pmatrix} \tag{130}$$

$$\bar{M}_2 = \begin{pmatrix} D_{12} L J^{-1} & 0 & D_{12} \end{pmatrix} \qquad \bar{N}_2 = \begin{pmatrix} J^{-1} N D_{21} \\ 0 \\ D_{21} \end{pmatrix} \tag{131}$$

See also :
MsensH, w_prod_norm

References :
[12] T. Hilaire and P. Chevrel. On the compact formulation of the derivation of a transfer matrix with respect to another matrix. Technical Report RR-6760, INRIA, 2008.
[16] T. Hilaire, P. Chevrel, and J. Whidborne. Low parametric closed-loop sensitivity realizations using fixed-point and floating-point arithmetic. In Proc. European Control Conference (ECC'07), July 2007.
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.3.16 MsensPole

Purpose :
Compute the open-loop pole sensitivity measure (and the pole sensitivity matrix) for a FWR object

Syntax :
```
[M, dlambda_dZ, dlk_dZ] = MsensPole( R, moduli)
```

Parameters :

`M`	: pole sensitivity measure
`dlambda_dZ`	: the pole sensitivity matrix
`dlk_dZ`	: pole sensitivity matrices for each pole
`R`	: FWR object
`moduli`	: 1 (default value) : compute $\frac{\partial\|\lambda\|}{\partial Z}$ (the sensitivity of the moduli of the eigenvalues)
	: 0 : compute $\frac{\partial\lambda}{\partial Z}$ (without the moduli)

Description :
The pole sensitivity measure of $\mathcal{R}$ is defined by

$$\Psi = \sum_{k=1}^{n} \left\| \frac{\partial |\lambda_k|}{\partial Z} \times r_Z \right\|_F^2. \tag{132}$$

(it is also possible to only consider the sensitivity of λ_k instead of the sensitivity of $|\lambda_k|$. This measure can be evaluated with

$$\frac{\partial |\lambda_k|}{\partial Z} = \begin{pmatrix} KJ^{-1} & I & 0 \end{pmatrix}^\top \frac{\partial |\lambda_k|}{\partial A} \begin{pmatrix} J^{-1}M \\ I \\ 0 \end{pmatrix}^\top \tag{133}$$

and the following lemma[53]:

Lemma 1 *Let $M \in \mathbb{R}^{n\times n}$ be diagonalizable. Let $(\lambda_k)_{1\leqslant k\leqslant n}$ be its eigenvalues, and $(x_k)_{1\leqslant k\leqslant n}$ the corresponding right eigenvectors. Denote $M_x \triangleq \begin{pmatrix} x_1 x_2 \dots x_n \end{pmatrix}$ and $M_y = \begin{pmatrix} y_1 y_2 \dots y_n \end{pmatrix} \triangleq M_x^{-H}$. Then*

$$\frac{\partial \lambda_k}{\partial M} = y_k^* x_k^\top \quad \forall k = 1, \dots, n \tag{134}$$

and

$$\frac{\partial |\lambda_k|}{\partial M} = \frac{1}{|\lambda_k|} Re\left(\lambda_k^* \frac{\partial \lambda_k}{\partial M} \right) \tag{135}$$

where $\cdot^*$ denotes the conjugate operation, $Re(\cdot)$ the real part and $\cdot^H$ the transpose conjugate operator.

A *pole sensitivity matrix* can also be constructed to evaluate the overall impact of each coefficient. Let $\frac{\delta|\lambda|}{\delta Z}$ denote the pole sensitivity matrix defined by

$$\left(\frac{\delta |\lambda|}{\delta Z} \right)_{i,j} \triangleq \sqrt{\sum_{k=1}^{n} \left(\frac{\partial |\lambda_k|}{\partial Z_{i,j}} \right)^2}. \tag{136}$$

Then, The pole sensitivity measure is then given by:

$$\Psi = \left\| \frac{\delta |\lambda|}{\delta Z} \times r_Z \right\|_F^2. \tag{137}$$

See also :
MsensPole_cl, Mstability, deigdZ, MsensPole

References :
[14] T. Hilaire, P. Chevrel, and J.-P. Clauzel. Pole sensitivity stability related measure of FWL realization with the implicit state-space formalism. In 5th IFAC Symposium on Robust Control Design (ROCOND'06), July 2006.
[17] T. Hilaire, P. Chevrel, and J. Whidborne. A unifying framework for finite wordlength realizations. IEEE Trans. on Circuits and Systems, 8(54), August 2007.

7.3.17 MsensPole_cl

Purpose :
Compute the closed-loop pole sensitivity measure (and the pole sensitivity matrix) for a FWR object

Syntax :
```
[M, dlambdabar_dZ, dlbk_dZ] = MsensPole_cl( R, Sysp, moduli)
```

Parameters :

M	: pole sensitivity measure
dlambdabar_dZ	: the pole sensitivity matrix
dlbk_dZ	: pole sensitivity matrices for each pole
R	: FWR object
Sysp	: plant system (ss object)
moduli	: 1 (default value) : compute $\frac{\partial\lvert\bar{\lambda}\rvert}{\partial Z}$ (the sensitivity of the moduli of the eigenvalues)
	: 0 : compute $\frac{\partial\bar{\lambda}}{\partial Z}$ (without the moduli)

Description :
This measure is similar to the pole sensitivity measure in open-loop case (see MsensPole), but the closed-loop poles are now considered (the eigenvalues of $\bar{A}$).
The pole sensitivity measure of $\mathcal{R}$ is defined by

$$\bar{\Psi} = \sum_{k=1}^{n} \left\| \frac{\partial \left|\bar{\lambda}_k\right|}{\partial Z} \times r_Z \right\|_F^2 . \tag{138}$$

This measure can be evaluated with

$$\frac{\partial \left|\bar{\lambda}_k\right|}{\partial Z} = \bar{M}_1^t op \frac{\partial \left|\bar{\lambda}_k\right|}{\partial A} \bar{N}_1^\top \tag{139}$$

with

$$\bar{M}_1 = \begin{pmatrix} B_2 L J^{-1} & 0 & B_2 \\ K J^{-1} & I_n & 0 \end{pmatrix} \qquad \bar{N}_1 = \begin{pmatrix} J^{-1} N C_2 & J^{-1} M \\ 0 & I_n \\ C_2 & 0 \end{pmatrix} \tag{140}$$

and the following lemma[53]:

Lemma 2 *Let $M \in \mathbb{R}^{n\times n}$ be diagonalizable. Let $(\lambda_k)_{1\leqslant k\leqslant n}$ be its eigenvalues, and $(x_k)_{1\leqslant k\leqslant n}$ the corresponding right eigenvectors. Denote $M_x \triangleq \begin{pmatrix} x_1 x_2 \dots x_n \end{pmatrix}$ and $M_y = \begin{pmatrix} y_1 y_2 \dots y_n \end{pmatrix} \triangleq M_x^{-H}$. Then*

$$\frac{\partial \lambda_k}{\partial M} = y_k^* x_k^\top \quad \forall k = 1, \dots, n \tag{141}$$

and

$$\frac{\partial \left|\lambda_k\right|}{\partial M} = \frac{1}{\left|\lambda_k\right|} Re\left(\lambda_k^* \frac{\partial \lambda_k}{\partial M}\right) \tag{142}$$

where $\cdot^*$ denotes the conjugate operation, $Re(\cdot)$ the real part and $\cdot^H$ the transpose conjugate operator.

A *pole sensitivity matrix* can also be constructed to evaluate the overall impact of each coefficient. Let $\frac{\delta|\bar{\lambda}|}{\delta Z}$ denote the pole sensitivity matrix defined by

$$\left(\frac{\delta \left|\bar{\lambda}\right|}{\delta Z}\right)_{i,j} \triangleq \sqrt{\sum_{k=1}^{n} \left(\frac{\partial \left|\bar{\lambda}_k\right|}{\partial Z_{i,j}}\right)^2}. \tag{143}$$

Then, The pole sensitivity measure is then given by:

$$\Psi = \left\| \frac{\delta \left|\bar{\lambda}\right|}{\delta Z} \times r_Z \right\|_F^2. \tag{144}$$

See also :
`MsensPole`, `Mstability`, `deigdZ`, `MsensPole`

References :
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.3.18 `Mstability`

Purpose :
Compute the closed-loop pole sensitivity stability related measure for a FWR object

Syntax :
`M = Mstability( R, Sysp, moduli)`

Parameters :
`M` : pole sensitivity measure
`R` : FWR object
`Sysp` : plant system (ss object)
`moduli` : 1 (default value) : compute $\frac{\partial|\bar{\lambda}|}{\partial Z}$ (the sensitivity of the moduli of the eigenvalues)
: 0 : compute $\frac{\partial\bar{\lambda}}{\partial Z}$ (without the moduli)

Description :
The Pole Sensitivity Stability related Measure (PSSM) is defined by

$$\mu_1(Z) \triangleq \min_{1\leqslant k\leqslant n_{\mathcal{P}}+n} \frac{1-\left|\bar{\lambda}_k\right|}{\|W_Z\|_F \left\|\frac{\partial|\bar{\lambda}_k|}{\partial Z} \times W_Z\right\|_F}. \tag{145}$$

This measure evaluates how a perturbation, Δ, of the parameters, Z, can cause instability. It is determined by how close the eigenvalues of $\bar{A}$ are to the unit circle and by how sensitive they are to the controller parameter perturbation.
See `MsensPole_cl` for the computation of $\frac{\partial|\bar{\lambda}_k|}{\partial Z}$.

See also :
`MsensPole_cl`, `Mstability`

References :
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.3.19 `mtimes`

Purpose :
Multiply two FWR (put them in cascade)

Syntax :
```
R = R1*R2
R = mtimes(R1,R2)
```

Parameters :
`R` : FWR result
`R1` : first operand (FWR)
`R2` : second operand (FWR)

Description :
Put two realization in cascade (see figure 7).

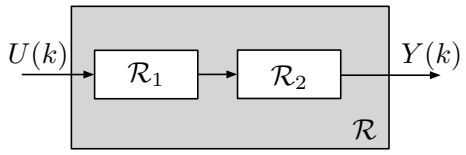

Figure 7: Two realizations in cascade

We consider two realizations $\mathcal{R}_1 := (J_1, K_1, L_1, M_1, N_1, P_1, Q_1, R_1, S_1)$ and $\mathcal{R}_2 := (J_2, K_2, L_2, M_2, N_2, P_2, Q_2, R_2, S_2)$ (with compatible size, *i.e.* $p_1 = m_2$).
By introducing an intermediate variable T equal to the output of $\mathcal{R}_1$, the resulting realization $\mathcal{R}$ can be expressed in the implicit form by :

$$\left(\begin{array}{ccc:c:c} J_1 & 0 & 0 & 0 & 0 \\ -L_1 & I & 0 & 0 & 0 \\ 0 & -N_2 & J_2 & 0 & 0 \\ \hdashline -K_1 & 0 & 0 & I & 0 \\ 0 & -Q_2 & -K_2 & 0 & I \\ \hdashline 0 & -S_2 & -L_2 & 0 & 0 \end{array}\right) \left(\begin{array}{c} T_1(k+1) \\ T(k+1) \\ T_2(k+1) \\ X_1(k+1) \\ X_2(k+1) \\ Y_2(k) \end{array}\right) = \left(\begin{array}{ccc:c:c} 0 & 0 & 0 & M_1 & N_1 \\ 0 & 0 & 0 & R_1 & S_1 \\ 0 & 0 & 0 & 0 & M_2 \\ \hdashline 0 & 0 & 0 & P_1 & Q_1 \\ 0 & 0 & 0 & P_2 & 0 \\ \hdashline 0 & 0 & 0 & R_2 & 0 \end{array}\right) \left(\begin{array}{c} T_1(k) \\ T(k) \\ T_2(k) \\ X_1(k) \\ X_2(k) \\ U_1(k) \end{array}\right)$$

See also :
`plus`

7.3.20 `ONP`

Purpose :
Compute the Output Noise Power for a FWR object with Roundoff Before Multiplication (RBM) computational scheme

Syntax :
```
M = ONP( R, roundingMode)
```

Parameters :
`M` : output noise power measure
`R` : FWR object
`mode` : indicates the truncation mode: `'truncation'` (default) or `'nearest'`

Description :
This function computes the Output Noise Power.
Let us consider a realization $\mathcal{R}$ described with the implicit form (4), with transfer

function H. When implemented, the steps (i) to (iii) are modified by the add of noises $\xi_T(k)$, $\xi_X(k)$ and $\xi_Y(k)$:

$$\begin{aligned} J.T(k+1) &\leftarrow M.X(k) + N.U(k) + \xi_T(k) \\ X(k+1) &\leftarrow K.T(k+1) + P.X(k) + Q.U(k) + \xi_X(k) \\ Y(k) &\leftarrow L.T(k+1) + R.X(k) + S.U(k) + \xi_Y(k) \end{aligned} \tag{146}$$

These noises added depend on:

- the way the computations are organized (the order of the sums) and done,
- the fixed-point representation of the inputs, the outputs,
- and the fixed-point representation chosen for the states, the intermediate variables and the coefficients.

They are modeled as independent white noise, characterized by their first and second order moments. Denote ξ the vector with all the added noise sources:

$$\xi(k) \triangleq \begin{pmatrix} \xi_T(k) \\ \xi_X(k) \\ \xi_Y(k) \end{pmatrix} \tag{147}$$

Proposition 8 *It is then possible to express the implemented system as the initial system with a noise $\xi'(k)$ added on the output(s) (see figure 8).*

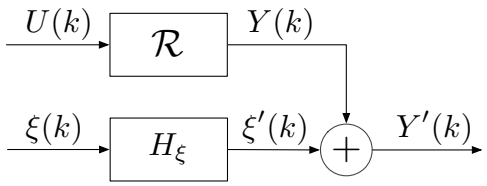

Figure 8: Equivalent system, with noises extracted

$\xi'(k)$ is the noise $\xi(k)$ through the transfer function H_ξ defined by:

$$H_\xi : z \rightarrow C_Z \left(zI_n - A_Z\right)^{-1} M_1 + M_2 \tag{148}$$

with

$$M_1 \triangleq \begin{pmatrix} KJ^{-1} & I_n & 0 \end{pmatrix} \tag{149}$$

$$M_2 \triangleq \begin{pmatrix} LJ^{-1} & 0 & I_{p_2} \end{pmatrix} \tag{150}$$

The Output Noise Power is defined as the power of the noises added on the output

$$P \triangleq E\xi'(k)\xi'(k)^\top \tag{151}$$

where the $E.$ is the mean operator.
It is evaluated by [20]:

$$P = tr\left(\psi_\xi \left(M_2^\top M_2 + M_1^\top W_o M_1\right)\right) + \mu_{\xi'}^\top \mu_{\xi'} \tag{152}$$

where $\mu_{\xi'} = (C_Z(I - A_Z)^{-1}M_1 + M_2)\mu_\xi$ and W_o is the observability gramian of the system $\mathcal{R}$. Then, in *Roundoff Before Multiplication* (RBM) scheme, the quantizations only occur at the end of the additions, when the accumulator result is stored in intermediate variables, states or output, and a right-shift of d_{ADD} bits is applied. The lemma 3 recalls the noise produced during shift:

Lemma 3 *Let $x(k)$ be a signal with fixed-point format $(\beta + d, \alpha + d)$. Right shifting $x(k)$ of d bits is similar to add to $x(k)$ the independent white noise $e(k)$.*
The right shift could round $x(k)$ towards $-\infty$ (truncation: default behaviour) or toward the nearest integer (nearest rounding: possible with some additional hardware/software operations [31]). If $d > 0$, the moments of $e(k)$ are given by:

	truncation	*best roundoff*
μ_e	$2^{-\gamma-1}(1-2^{-d})$	$2^{-\gamma-d-1}$
σ_e^2	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$	$\frac{2^{-2\gamma}}{12}(1-2^{-2d})$

(153)

else ($d \leqslant 0$) $e(k)$ is null.

It is now possible to define the moments of $\xi(k)$: Denote $\bar{\gamma} \triangleq \begin{pmatrix}\gamma_T \\ \gamma_X \\ \gamma_Y\end{pmatrix}$ and define s by

$$s_i \triangleq \begin{cases} 1 & \text{if } d_{ADDi} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{154}$$

Then μ_ξ is given by:

$$(\mu_\xi)_i = \begin{cases} s_i 2^{-\bar{\gamma}_i - 1} & \text{truncation} \\ s_i 2^{-\bar{\gamma}_i - 1 - d_{ADD}} & \text{nearest rounding} \end{cases} \tag{155}$$

and, since these noises are independent, ψ_ξ is diagonal with:

$$(\psi_\xi)_{i,i} = s_i \frac{2^{-2\bar{\gamma}_i}}{12}\left(1 - 2^{-d_{ADD}}\right) \tag{156}$$

See also :
`setFPIS`, `RNG`

References :
[20] T. Hilaire, D. Ménard, and O. Sentieys. Bit accurate roundoff noise analysis of fixed-point linear controllers. In Proc. IEEE International Symposium on Computer-Aided Control System Design (CACSD'08), September 2008.

7.3.21 `plus`

Purpose :
add two FWR object (put them in parallel)

Syntax :
```
R = R1+R2
R = plus(R1,R2,generalform)
```

Parameters :

`R`	: FWR result
`R1`	: first FWR
`R2`	: second FWR
`generalform`	: (default is true) to use the general form (or a particular form)

Description :
Put two realization in parallel (see figure 9).

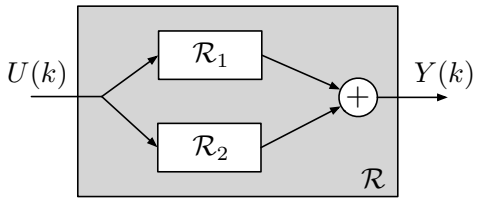

Figure 9: Two realizations in parallel

We consider two realizations $\mathcal{R}_1 := (J_1, K_1, L_1, M_1, N_1, P_1, Q_1, R_1, S_1)$ and $\mathcal{R}_2 := (J_2, K_2, L_2, M_2, N_2, P_2, Q_2, R_2, S_2)$ (with compatible size, *i.e.* $m_1 = m_2$ and $p_1 = p_2$).
By introducing the intermediate variables T_1' and T_2' equal to the output of the two realizations, the resulting realization $\mathcal{R}$ (*general* form) can be expressed in

the implicit form by

$$\left(\begin{array}{ccccc:cc:c} J_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -L_1 & I_{p_1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & J_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -L_2 & I_{p_2} & 0 & 0 & 0 \\ \hdashline -K_1 & 0 & 0 & 0 & I_{n_1} & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & I_{n_2} & 0 \\ \hdashline 0 & -I_p & 0 & -I_p & 0 & 0 & I_p \end{array}\right) \left(\begin{array}{c} T_1(k+1) \\ T_1'(k+1) \\ T_2(k+1) \\ T_2'(k+1) \\ X_1(k+1) \\ X_2(k+1) \\ Y_2(k) \end{array}\right) = \left(\begin{array}{cccc:cc:c} 0 & 0 & 0 & 0 & M_1 & 0 & N_1 \\ 0 & 0 & 0 & 0 & R_1 & 0 & S_1 \\ 0 & 0 & 0 & 0 & 0 & M_2 & N_2 \\ 0 & 0 & 0 & 0 & 0 & R_2 & S_2 \\ \hdashline 0 & 0 & 0 & 0 & P_1 & 0 & Q_1 \\ 0 & 0 & 0 & 0 & 0 & P_2 & Q_2 \\ \hdashline 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right) \left(\begin{array}{c} T_1(k) \\ T_1'(k) \\ T_2(k) \\ T_2'(k) \\ X_1(k) \\ X_2(k) \\ U_1(k) \end{array}\right)$$

If we allow to regroup S_1 and S_2 in one term S (and changing a bit the parametrization if S_1 and S_2 are both non-zero), the resulting realization can be expressed in a *compact* form :

$$\left(\begin{array}{cccc:c} J_1 & 0 & 0 & 0 & 0 \\ 0 & J_2 & 0 & 0 & 0 \\ \hdashline -K_1 & 0 & I_{n_1} & 0 & 0 \\ 0 & -K_2 & 0 & I_{n_2} & 0 \\ \hdashline -L_1 & -L_2 & 0 & 0 & I_p \end{array}\right) \left(\begin{array}{c} T_1(k+1) \\ T_2(k+1) \\ X_1(k+1) \\ X_2(k+1) \\ Y_2(k) \end{array}\right) = \left(\begin{array}{cc:cc:c} 0 & 0 & M_1 & 0 & N_1 \\ 0 & 0 & 0 & M_2 & N_2 \\ \hdashline 0 & 0 & P_1 & 0 & Q_1 \\ 0 & 0 & 0 & P_2 & Q_2 \\ \hdashline 0 & 0 & R_1 & R_2 & (S_1+S_2) \end{array}\right) \left(\begin{array}{c} T_1(k) \\ T_1'(k) \\ T_2(k) \\ T_2'(k) \\ X_1(k) \\ X_2(k) \\ U_1(k) \end{array}\right)$$

If S_1 and/or S_2 are null, the two forms are equivalent (in finite precision).

See also :
mtimes

7.3.22 quantized

Purpose :
Return the quantized realization, according to a fixed-point implementation scheme

Syntax :
```
[Rqt, DeltaZ] = quantized( R);
```

Parameters :
`Rqt` : quantized realization (FWR object)
`DeltaZ` : approximation on Z (it is NOT $Z - Z^{\dagger}$)
`R` : FWR object

Description :
According to a fixed-point implementation scheme, this function returns the realization with quantized coefficients. The binary point position of the coefficient depends on the computational scheme (*Roundoff Before Multiplication* or

Roundoff After Multiplication) and is given by:

$$\tilde{\gamma}_Z = \begin{cases} \gamma_Z & \text{if } RAM \\ \gamma_{ADD}.\mathbb{1}_{1,l+n+m} - \mathbb{1}_{l+n+p,1}.\begin{pmatrix}\gamma_T \\ \gamma_X \\ \gamma_U\end{pmatrix}^\top & \text{if } RBM \end{cases} \tag{157}$$

(γ_{ADD} is the binary point position of the adders, see [20] and `setFPIS`).
The new coefficients $Z^\dagger$ are then given by

$$Z^\dagger \triangleq 2^{-\gamma_Z} \lfloor Z \times 2^{\gamma_Z} \rceil \tag{158}$$

and the approximation Δ_Z on Z is

$$\Delta_Z = \frac{2^{-\gamma_Z}}{2} \times W_Z \tag{159}$$

See also :
`setFPIS`

References :
[20] T. Hilaire, D. Ménard, and O. Sentieys. Bit accurate roundoff noise analysis of fixed-point linear controllers. In Proc. IEEE International Symposium on Computer-Aided Control System Design (CACSD'08), September 2008.

7.3.23 `realize`

Purpose :
Numerically compute the outputs, states and intermediate variables with a given input U. Floating-point is used for the computations.

Syntax :
```
[Y, X, T] = realize( R, U, X0)
```

Parameters :
`Y` : outputs
`X` : states
`T` : intermediate variables
`R` : FWR object
`U` : inputs
`X0` : initial states (default=0)

Description :
This function could be useful to evaluate the magnitude values of the intermediate variables and the states.

It could also be useful to compare two different realizations, for example a realization and its quantized one.
It is important to notice that the computations are done in floating-point (the fixed-point implementation is not considered here).

7.3.24 `relaxedl2scaling`

Purpose :
Perform a relaxed-L_2-scaling on the FWR. The wordlength are deduced from the FPIS if it is defined.

Syntax :
```
R = lrelaxedl2scaling(R,Umax,delta)
[U,Y,W] = relaxedl2scaling(R,Umax,delta)
```

Parameters :
`R` : FWR object
`U,Y,W` : transformation matrices applied on R
`Umax` : input maximum magnitude (default: Umax = a power of 2)
`delta` : security parameter (default: delta=1)

Description :
Perform a relaxed-L_2-scaling.
The scaling forces the transfer functions from the inputs to the states and the intermediates variables to have a L_2-norm between 1 and 4. Theses norms are given by the diagonal terms of W_c and $J^{-1}\left(NN^\top + MW_cM^\top\right)J^{-\top}$.
Denote W_{cX} the controllability gramian of the realization (W_c) and W_{cT} the controllability gramian related to the intermediate variables. It is given by

$$W_{cX} = \sqrt{\left(J^{-1}\left(NN^\top + MW_{cX}M^\top\right)J^{-\top}\right)} \tag{160}$$

The relaxed-L_2-scaling is a transformation that make the realization satisfy the constraints ($\forall 1 \leqslant i \leqslant n$)

$$\frac{2^{2\alpha_{X\,i}}}{\delta^2} \leqslant \quad (W_{cX})_{i,i} \quad < 4\frac{2^{2\alpha_{X\,i}}}{\delta^2} \tag{161}$$

$$\frac{2^{2\alpha_{T\,i}}}{\delta^2} \leqslant \quad (W_{cT})_{i,i} \quad < 4\frac{2^{2\alpha_{T\,i}}}{\delta^2} \tag{162}$$

where

$$\alpha_{X\,i} \quad \triangleq \quad \beta_{X_i} - \beta_U - \mathscr{F}_2\begin{pmatrix}\max\\ U\end{pmatrix} \tag{163}$$

$$\alpha_{T\,i} \quad \triangleq \quad \beta_{T_i} - \beta_U - \mathscr{F}_2\begin{pmatrix}\max\\ U\end{pmatrix} \tag{164}$$

and $\mathscr{F}_2(x)$ is defined as the fractional value of $\log_2(x)$:

$$\mathscr{F}_2(x) \triangleq \log_2(x) - \lfloor \log_2(x) \rfloor \tag{165}$$

If the wordlength β_X and β_T are not defined by the FPIS, they are also supposed to be equal. Moreover, if $\delta = 1$ (default case) and $\overset{\max}{U}$ is a power of 2, then the realization satisfies the following constraints

$$1 \leqslant (W_{cX})_{ii} < 4, \qquad 1 \leqslant (W_{cT})_{ii} < 4, \qquad \forall 1 \leqslant i \leqslant n \tag{166}$$

The L_2-scaling is achieved by a $\mathcal{UYW}$-transformation where $\mathcal{U}$ and $\mathcal{W}$ are diagonal with:

$$(\mathcal{U})_{ii} = \delta \sqrt{(W_{cX})_{i,i}} 2^{-\mathscr{F}_2(\delta\sqrt{(W_{cX})_{i,i}}) - \alpha_{X\,i}} \tag{167}$$

$$(\mathcal{W})_{ii} = \delta \sqrt{(W_{cT})_{i,i}} 2^{-\mathscr{F}_2(\delta\sqrt{(W_{cT})_{i,i}}) - \alpha_{T\,i}} \tag{168}$$

See also :
l2scaling

References :
[8] Y. Feng, P. Chevrel, and T. Hilaire. A practical strategy of an efficient and sparse fwl implementation of lti filters. In submitted to ECC?09, 2009.
[11] T. Hilaire. Low parametric sensitivity realizations with relaxed l2-dynamic-range-scaling constraints. submitted to IEEE Trans. on Circuits & Systems II, 2009.

7.3.25 RNG

Purpose :
Compute the open-loop Roundoff Noise Gain

Syntax :
```
G = RNG(R)
[G, dZ] = RNG(R,tol)
```

Parameters :
`G` : roundoff noise gain
`dZ` : number of non-trivial parameters (used by @FWS/RNG)
`R` : FWR object
`tol` : tolerance on trivial parameters (default=1e-8);

Description :
This function computes the Roundoff Noise Gain (in open-loop context).
The Roundoff Noise Gain is the output noise power computed in a specific

computational scheme : the noises are supposed to appear only after each multiplication and are modeled by centered white noise statistically independent. Each noise is supposed to have the same power σ_0^2 (determined by the wordlength chosen for all the variables and coefficients).
The Roundoff Noise Gain is defined by

$$G \triangleq \frac{P}{\sigma_0^2} \tag{169}$$

where P is the output roundoff noise power. It could be computed by

$$G = tr\left(dZ(M_2^\top M_2 + M_1^\top W_o M_1)\right) \tag{170}$$

with

$$M_1 \triangleq \begin{pmatrix} KJ^{-1} & I_n & 0 \end{pmatrix} \tag{171}$$

$$M_2 \triangleq \begin{pmatrix} LJ^{-1} & 0 & I_{p_2} \end{pmatrix} \tag{172}$$

and the matrix d_Z is a diagonal matrix defined by

$$(d_Z)_{i,i} \triangleq \text{number of non-trivial parameters in the i}^{\text{th}} \text{ row of } Z \tag{173}$$

The trivial parameters considered are 0, 1, -1 and powers of 2 because they do not imply a multiplication.

See also :
RNG_cl, RNG

References :
[19] T. Hilaire, D. Ménard, and O. Sentieys. Roundoff noise analysis of finite wordlength realizations with the implicit state-space framework. In 15th European Signal Processing Conference (EUSIPOC'07), September 2007.

7.3.26 RNG_cl

Purpose :
Compute the closed-loop Roundoff Noise Gain

Syntax :
```
G = RNG( R, Sysp)
[G, dZ] = RNG(R, Sysp, tol)
```

Parameters :
`G` : roundoff noise gain
`dZ` : number of non-trivial parameters (used by @FWS/RNG)
`R` : FWR object
`Sysp` : plant (to be controlled)
`tol` : tolerance on trivial parameters (default=1e-8)

Description :
This function computes the Roundoff Noise Gain (in closed-loop context).
The Roundoff Noise Gain is the output noise power computed in a specific computational scheme : the noises are supposed to appear only after each multiplication and are modeled by centered white noise statistically independent. Each noise is supposed to have the same power σ_0^2 (determined by the wordlength choosen for all the variables and coefficients).
The Roundoff Noise Gain is defined by

$$\bar{G} \triangleq \frac{\bar{P}}{\sigma_0^2} \tag{174}$$

where $\bar{P}$ is the output roundoff noise power (the global noise added on the output of the plant). It could be computed by

$$\bar{G} = tr\left(d_Z(\bar{M}_2^\top \bar{M}_2 + \bar{M}_1^\top \bar{W}_o \bar{M}_1)\right) \tag{175}$$

with

$$\bar{M}_1 = \begin{pmatrix} B_2 L J^{-1} & 0 & B_2 \\ K J^{-1} & I_n & 0 \end{pmatrix}, \tag{176}$$

$$\bar{M}_2 = \begin{pmatrix} D_{12} L J^{-1} & 0 & D_{12} \end{pmatrix} \tag{177}$$

and the matrix d_Z is a diagonal matrix defined by

$$(d_Z)_{i,i} \triangleq \text{number of non-trivial parameters in the i}^{\text{th}} \text{ row of } Z \tag{178}$$

The trivial parameters considered are 0, 1 and -1 because they do not imply a multiplication.

See also :
RNG, RNG_cl

References :
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.3.27 set

Purpose :
Set some properties of a FWR object

Syntax :
```
R = set(R, propName, value)
```

Parameters :

`R`	: FWR object
`propName`	: name of the property
`value`	: new value for this property

Description :
This function is most of the time called by `subsasgn`.
The value of every field (`l`, `m`, `n` and `p` ; `J`, `K`, `L`, `M`, `N`, `P`, `Q`, `R` and `S`; `Z` ; `WJ`, `WK`, `WL`, `WM`, `WN`, `WP`, `WQ`, `WR` and `WS`, `WZ`, `AZ`, `BZ`, `CZ` and `AZ`) can be evaluated, but `l`, `m`, `n`, `p`, `AZ`, `BZ`, `CZ` and `AZ` cannot be modified.
Changing `Z` changes fields `J` to `S`, and reciprocally (this is the same with `WZ`).

See also :
`get`, `subsasgn`, `set`

7.3.28 `setFPIS`

Purpose :
Set the Fixed-Point Implementation Scheme (FPIS) of an FWR object (the wordlength may be matrices or scalar. The scalar case is used to set all the wordlength to the same length)

Syntax :
```
R = setFPIS( R, betaU, Umax, betaZ, betaT, betaX, betaY, betaADD, betaG,
method )
R = setFPIS( R, FPIS)
R = setFPIS( R, FPISname,Umax)
```

Parameters :

`R`	: FWR object
`FPIS`	: an other Fixed-Point Implementation Scheme (a structure with betaU, Umax, betaZ, betaT, betaX, betaY, betaADD, betaG and method)
`FPISname`	: `'DSP8'` or `'DSP16'`
`betaU`	: wordlength of U (inputs)
`Umax`	: maximum value of U (necessary to set γ_U)
`betaZ`	: wordlength of the coefficients
`betaT, betaX, betaY`	: wordlength of the intermediate variables T, the states X and the outputs
`betaADD`	: wordlength of the accumulators
`betaG`	: nb of guard bits in the accumulators
`method`	: `'RBM'` (default) Roundoff Before Multiplication
	: `'RAM'` Roundoff After Multiplication

Description :
This function sets the Fixed-Point Implementation Scheme (FPIS). This structure is composed by:

- the fixed-point format of the input (β_U, γ_U) and its maximum magnitude value $\overset{\max}{U}$
- the fixed-point format of the intermediate variables (β_T, γ_T)
- the fixed-point format of the states (β_X, γ_X)
- the fixed-point format of the output (β_Y, γ_Y)
- the fixed-point format of the coefficients (β_Z, γ_Z)
- the fixed-point format of the accumulator $(\beta_{ADD} + \beta_G, \gamma_{ADD})$ (β_G guard bits)
- the right-shift bits after each scalar product d_{ADD} (`shiftADD`)
- the right-shift bits after each multiplication by a coefficient d_Z (`shiftZ`)
- the computational scheme : *Roundoff After Multiplication* (RAM) or *Roundoff Before Multiplication* (RBM)

The algorithm

$$\begin{aligned} &\text{[i]} && JT(k+1) \leftarrow MX(k) + NU(k) \\ &\text{[ii]} && X(k+1) \leftarrow KT(k+1) + PX(k) + QU(k) \\ &\text{[iii]} && Y(k) \leftarrow LT(k+1) + RX(k) + SU(k) \end{aligned}$$

requires to implement $l + n + p$ scalar products.
Each scalar product

$$S = \sum_{i=1}^{n} P_i E_i \tag{179}$$

where $(P_i)_{1 \leqslant i \leqslant n}$ are given coefficients and $(E_i)_{1 \leqslant i \leqslant n}$ some bounded variables, can be implemented according to the algorithms 5 and 6, and where P'_i, E'_i and S'_i are the integer representation (according to their fixed-point format) to P_i,E_i and S_i.

$Add \leftarrow 0$
for *i from 0 to n* **do**
  $Add \leftarrow (P'_i * E'_i) >> d_i$
end
$S'_i \leftarrow Add >> d'_i$

Algorithm 5: *Roundoff After Multiplication* (RAM)

$Add \leftarrow 0$
for *i from 0 to n* **do**
  $Add \leftarrow (P'_i >> d_i) * E'_i$
end
$S'_i \leftarrow Add >> d'_i$

Algorithm 6: *Roundoff Before Multiplication* (RBM)

Of course, d_i represents the right-shift after each multiplication and d'_i represents the final shift. They respectively correspond to the d_Z and d_{ADD} shift in the SIF algorithm. The user may specify all the wordlengths (β_U, β_T, β_X, β_Y, β_{ADD}, β_g and β_Z) and $\overset{\max}{U}$. The binary-point positions are deduced by:

$$\gamma_U = \beta_U - 2 - \left\lfloor \log_2 \overset{\max}{U} \right\rfloor \tag{180}$$

$$\begin{pmatrix} \gamma_T \\ \gamma_X \\ \gamma_Y \end{pmatrix} = \begin{pmatrix} \beta_T \\ \beta_X \\ \beta_Y \end{pmatrix} - 2.\mathbb{1}_{l+n+p,1} - \left\lfloor \log_2 \left(\|H_{\max}\|_{l_1} \overset{\max}{|U|} \right) \right\rfloor \tag{181}$$

where $\mathbb{1}_{k,l}$ represents the matrix of $\mathbb{R}^{k\times l}$ with all coefficients set to 1, $\|.\|_{l_1}$ the l_1-norm and

$$H_{\max} : z \rightarrow N_1 \left(zI_n - A_Z\right)^{-1} B_Z + N_2, \tag{182}$$

$$N_1 \triangleq \begin{pmatrix} J^{-1}M \\ I_n \\ C_Z \end{pmatrix}, N_2 \triangleq \begin{pmatrix} J^{-1}N \\ 0 \\ D_Z \end{pmatrix} \tag{183}$$

The binary point position[3] γ_Z of the coefficients Z are given by:

$$\gamma_Z = \beta_Z - 2.\mathbb{1}_{l+n+p,l+n+m} - \left\lfloor \log_2 |Z| \right\rfloor \tag{184}$$

The fixed-point formats of the additions are given by:

$$\gamma_{ADD} = \beta_{ADD} - \max_{row} \left(\begin{pmatrix} \beta_T \\ \beta_X \\ \beta_Y \end{pmatrix} - \beta_g - \begin{pmatrix} \gamma_T \\ \gamma_X \\ \gamma_Y \end{pmatrix}, \alpha \right) \tag{185}$$

where

$$\alpha = \max_{row} \left(\beta_Z - \gamma_Z + \mathbb{1}_{l+n+p,1} \left(\begin{pmatrix} \beta_T \\ \beta_X \\ \beta_U \end{pmatrix} - \begin{pmatrix} \gamma_T \\ \gamma_X \\ \gamma_U \end{pmatrix} \right)^\top \right) \tag{186}$$

and $\max_{row}(M)$ returns a column vector with the maximum value of each row of M.
The final alignments are right shifts of d_{ADD} bits, with:

$$d_{ADD} = \gamma_{ADD} - \begin{pmatrix} \gamma_T \\ \gamma_X \\ \gamma_Y \end{pmatrix} \tag{187}$$

Denote $\tilde{\gamma}_Z$ the final binary point position of the coefficients Z, according to *RAM* or *RBM* scheme, and d_Z the shifts needed after each multiplication ($(d_Z)_{i,j}$ is

[3] $(\gamma_Z)_{i,j}$ could be $-\infty$ for null coefficients, but it is not a problem because such coefficients are not implemented

the right shift needed after the multiplication by $Z_{i,j}$) in order to align the format after each multiplication. Then:

$$\tilde{\gamma}_Z = \begin{cases} \gamma_Z & \text{if } RAM \\ \gamma_{ADD}.\mathbb{1}_{1,l+n+m} - \mathbb{1}_{l+n+p,1}.\begin{pmatrix}\gamma_T\\ \gamma_X\\ \gamma_U\end{pmatrix}^\top & \text{if } RBM \end{cases} \tag{188}$$

and

$$d_Z = \tilde{\gamma}_Z + \mathbb{1}_{l+n+p,1}.\begin{pmatrix}\gamma_T\\ \gamma_X\\ \gamma_U\end{pmatrix}^\top - \gamma_{ADD}.\mathbb{1}_{1,l+n+m} \tag{189}$$

(d_Z is a null matrix in RBM case).
With d_Z, $\tilde{\gamma}_Z$, γ_{ADD}, d_{ADD}, γ_T, γ_X and γ_Y, the fixed-point implementation of the controller is entirely defined.

See also :
`quantized`, `setFPIS`

References :
[20] T. Hilaire, D. Ménard, and O. Sentieys. Bit accurate roundoff noise analysis of fixed-point linear controllers. In Proc. IEEE International Symposium on Computer-Aided Control System Design (CACSD'08), September 2008.

7.3.29 `sigma_tf`

Purpose :
Compute the open-loop transfer function error $\sigma^2_{\Delta H}$)

Syntax :
```
[e, eZ] = sigma_tf(R)
```

Parameters :
`e` : transfer function error
`e` : transfer function error matrix
`R` : FWR object

Description :
The open-loop transfer function error is defined by

$$\sigma^2_{\Delta H} \triangleq \frac{1}{2\pi}\int_0^{2\pi} E\left\{\left|\Delta H\left(e^{j\omega}\right)\right|^2\right\} d\omega \tag{190}$$

where $E\{.\}$ is the mean operator.

See also :
MsensH

7.3.30 simplify

Purpose :
Simplify (if possible) a FWR, by removing the non-necessary intermediate variables and states

Syntax :
```
Rs = simplify( R, level, tol)
```

Parameters :
Rs	: simplified FWR object
R	: FWR to be simplified
level	: level of simplification
	: 0 : simplify only null terms
	: 1(default) : simplify lines with only one term (substitution)
	: 2 : simplify lines with 2 terms, etc...
	: level > 1 may increase the complexity
tol	: tolerance (default=1e-14)

Description :
Due to the sparsity of some realizations (for example, the FFT ones created by FFT2FWR), some simplifications in the realization can be provided. Null intermediate variables can appear and must be propagated, and intermediate variables that are sometimes set equal to an other value (without any other computations) must be removed. Let consider a realization $\mathcal{R} := (Z, l, m, n, p)$. Computing $T(k+1)$, $X(k+1)$ and $Y(k)$ for each step according to the associated algorithm is equivalent to compute

$$Z'.\begin{pmatrix} T(k+1) \\ X(k) \\ U(k) \end{pmatrix} \text{ with } Z' \triangleq Z + \begin{pmatrix} I_l & \\ & 0 \end{pmatrix} \tag{191}$$

This function simplifies (if possible) the realization by applying two transformations.
The first transform removes the intermediate variables that are null (they could appear for the FFT realization, because of the real or imaginary parts that are null). It can be described by the following algorithm:

while *it exists* $i \leqslant l+n$ *such as* $Z'_{i,\bullet}$ *is a null vector* **do**
  // Remove i-th intermediate variable or state
  remove i-th row and i-th column of Z';
  decrease l or n;
end

Algorithm 7: Remove the null values

In some cases, various intermediate variables are only equal to another intermediate variable.
For example, if $T_2 \leftarrow aT_1$ and $T_3 \leftarrow bT_2$, when it is possible to substitute T_3 to T_2 if a or $b \in \{-1, 1\}$ (in order to preserve the parametrization).
The algorithm 8 allows the substitution of an intermediate variable by an other one.

while *it exists* $i \leqslant l$ *such as* $Z'_{i,\bullet}$ *has only one non-null element* $Z'_{i,j}$ **do**
  `// then we have something like` $T_i \leftarrow aT_j$
  if *for all* $k \neq i$, $Z'_{k,i} \neq 0$ *implies* $\left(Z'_{k,j} = 0 \text{ and } \left(Z'_{i,j} = \pm 1 \text{ or } Z'_{k,i} = \pm 1\right)\right)$ **then**
    `// Substitution`
    for $1 \leqslant k \leqslant l + n + p$ *such as* $Z'_{k,i} \neq 0$ **do**
      $Z'_{k,j} \leftarrow \varepsilon Z'_{k,i}.Z'_{i,j}$
      with $\varepsilon = \text{sign}(k,i).\text{sign}(i,j).\text{sign}(k,j)$
      and $\text{sign}(p,q) = \begin{cases} -1 & \text{if } p \leqslant l \text{ and } q \leqslant l \\ 1 & \text{otherwise} \end{cases}$
    end
    `// Remove` i`-th intermediate variable`
    remove i-th row and i-th column of Z';
    decrease l;
  end
end

Algorithm 8: Substitute the intermediate variables

The term ε in that algorithm is introduced to taking in consideration the $-J$ in the definition of Z (eq. (11))(this $-J$ was introduced in [17] in order to simplify the sensitivities and roundoff measure).

Remark 3 *It is also possible to consider the substitution when* T_i *is composed various terms. For example,* $T_3 \leftarrow aT_1 + bT_2$ *and* $T_4 \leftarrow cT_3$ *become* $T_4 \leftarrow acT_1 + bcT_2$ *if* $c = \pm 1$ *or* ($a = \pm 1$ *and* $b = \pm 1$). *In that case, this transformation can increase the complexity of the computations (since an intermediate variable that is substituted is used twice or more).*

The input `level` can set the level of the substitution. `0` means that only the null terms are removed, and `1` only the terms like $T_2 \leftarrow T_3$ are removed. With a greater value, the function considers the substitution if an intermediate variable is composed from various terms, and `level` gives the maximum number of terms.

Example :
The FFT_4 transform first corresponds to the following algorithm (see `FFT2FWR`)
When only the null terms are removed (with `level`=0), the algorithm becomes:
Then, with a complete substitution (`level`=1), the final algorithm is:

See also :
`FFT2FWR`

7.3.31 `size`

Purpose :
Return the size of the FW Realization

Syntax :
```
lmnp = size(R)
[l,m,n,p] = size(R)
```

Parameters :
`lmnp` : vector [l,m,n,p]
`l` : nb of intermediate variables
`m` : nb of inputs
`n` : nb of states
`p` : nb of outputs
`R` : FWR object

Description :
Overload of the classical `size` function.

7.3.32 `ss`

Purpose :
Convert a FWR object into a ss (state-space) object (equivalent state-space)

Syntax :
```
S = ss(R,Te)
```

Parameters :
`S` : ss object
`R` : FWR object
`Te` : period (default=1)

Description :
Give the equivalent state-space (`ss` object). It is defined with matrices A_Z, B_Z, C_Z and D_Z (see eq. (7) and (8)).

Example :
```
>>bode(ss(R));
```
where `R` is a FWR object, allows to plot its Bode frequency response.

See also :
`tf`, `ss`

7.3.33 subsasgn

Purpose :
Subscripted assign for FWR object here, `R.prop=value` is equivalent to `set(R,'prop',value)`

Syntax :
```
R = subsasgn(R,Sub,value)
```

Parameters :
`R` : FWR object
`Sub` : subassignment layers
`value` : value of the assignation

Description :
These functions are called internally when operators `[]`, `()` and `.` are applied on a FWR object.
Only the operator `.` is valid, and links to `set` and `get` functions. The command `R.field` returns the field `field` of `R` (internally, `get(R,'field')` is called), and `R.field=vaue` set the field `field` of `R`

Example :
```
R.P .* R.WP
R.P(1,:)
R.Z(3,3) = 0;
R.WZ = zeros( size(R.WZ) );
```

See also :
`set`, `get`, `subsref`, `subsasgn`

7.3.34 subsref

Purpose :
Subscripted reference for FWR object here, R.prop is equivalent to get(R,`prop`)

Syntax :
```
value = subsref(R,Sub)
```

Parameters :
`value` : returned value
`R` : FWR object
`Sub` : layers of subreferencing

Description :
These functions are called internally when operators `[]`, `()` and `.` are applied on a FWR object.
Only the operator `.` is valid, and links to `set` and `get` functions. The command `R.field` returns the field `field` of `R` (internally, `get(R,'field')` is called), and `R.field=vaue` set the field `field` of `R`

Example :
```
R.P .* R.WP
R.P(1,:)
R.Z(3,3) = 0;
R.WZ = zeros( size(R.WZ) );
```

See also :
`set`, `get`, `subsasgn`, `subsref`

7.3.35 `tf`

Purpose :
Convert a FWR object into a tf object (transfer function)

Syntax :
```
H = tf(R,Te)
```

Parameters :
`H` : tf object
`R` : FWR object
`Te` : period (default=1)

Description :
Give the transfer function of the realization. It is defined with matrices A_Z, B_Z, C_Z and D_Z (see eq. (7) and (8)) by:

$$H : z \mapsto C_Z \left(zI_n - A_Z\right)^{-1} B_Z + D_Z \tag{192}$$

See also :
`ss`, `tf`

7.3.36 `TradeOffMeasure_cl`

Purpose :
Compute a (pseudo) tradeoff closed-loop measure with the MsensH_cl, MsensPole_cl, RNG_cl

Syntax :
```
M = TradeOffMeasure_cl( R, Sysp, MH, MP, MRNG)
```

Parameters :

`M`	: measure value
`R`	: FWR object
`Sysp`	: plant system (`ss` object)
`MH,MP,MRNG`	: optimal value of the MsensH_cl, MsensPole_cl and RNG_cl measure

Description :
Even if a tradeoff measure like this one is not the best solution to multi-objective optimal realization, it is interesting to look for a realization that is *good enough* for the three measures $\bar{M}_{L_2}^W$, $\bar{\Psi}$ and $\bar{G}$. This (pseudo) tradeoff measure is defined by

$$\bar{TO}(Z) \triangleq \frac{\bar{M}_{L_2}^W(Z)}{\bar{M}_{L_2}^{W\ opt}} + \frac{\bar{\Psi}(Z)}{\bar{\Psi}^{opt}} + \frac{\bar{G}(Z)}{\bar{G}^{opt}} \tag{193}$$

where $\bar{M}_{L_2}^{W\ opt}$, $\bar{\Psi}^{opt}$ and $\bar{G}^{opt}$ are the optimal values for these respective measures.

See also :
`MsensH_cl`, `MsensPole_cl`, `RNG_cl`

References :
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.3.37 `transform`

Purpose :
Perform a UYW-transformation (similarity on Z)

Syntax :
```
R = transform(R, U, Y, W)
```

Parameters :

`R`	: FWR object
`U,Y,W`	: transformation matrices

Description :
The $\mathcal{UYW}$-transformation is defined as a particular similarity on Z:

$$\tilde{Z} = \begin{pmatrix} \mathcal{Y} & & \\ & \mathcal{U}^{-1} & \\ & & I_p \end{pmatrix} Z \begin{pmatrix} \mathcal{W} & & \\ & \mathcal{U} & \\ & & I_m \end{pmatrix} \tag{194}$$

where $\mathcal{U}$, $\mathcal{W}$, $\mathcal{Y}$ are non-singular matrices, are equivalent to $\mathcal{R}$.

7.4 FWR private functions

compute_rZ	Compute the matrix r_Z
computeAZBZCZDZWcWo	Compute the A_Z, B_Z, C_Z and D_Z matrices and the gramians of a FWR object
computeJtoS	Compute the J, K, L, M, N, P, Q, R, S, W_J, ..., W_J, of a FWR object
computelmnp	Compute the dimension l, m, n and p of a FWR object (from its parameters J, K, ..., S) and check the dimensions
computeZ	Compute the Z and W_Z of a FWR object from its parameters J, ..., S, W_J, ..., W_S.
deigdZ	Compute $M_1^\top \frac{\partial \lambda}{\partial A} M_2^\top$.
mylyap	Solve the continuous-time Lyapunov equations
scalprodCfloat	Return the C-code corresponding to a fixed-point scalar product
scalprodMATLAB	Write the MATLAB code corresponding to a fixed-point scalar product
scalprodVHDL	Write the VHDL code corresponding to a fixed-point scalar product
w_prod_norm	Compute the weighting L_2-norm of the system composed by $G \circledast H = Vec(G).(Vec(H^\top))^\top$
w_prod_norm_SISO	Compute the weighting L_2-norm of the system composed by $G \circledast H = Vec(G).(Vec(H^\top))^\top$

7.4.1 compute_rZ

Purpose :
Compute the matrix r_Z

Syntax :
```
R = compute_rZ(R)
```

Parameters :
R : FWR object

Description :

Internal function

During the quantization process, Z is perturbed to $Z + r_Z \times \Delta$ where

$$r_Z \triangleq \begin{cases} W_Z & \text{for fixed-point representation,} \\ 2\eta_Z \times W_Z & \text{for floating-point representation,} \end{cases} \tag{195}$$

and η_Z is such that

$$(\eta_Z)_{i,j} \triangleq \left\{ \begin{array}{l} \text{the largest absolute value of} \\ \text{the block in which } Z_{i,j} \text{ resides.} \end{array} \right. \tag{196}$$

This function considers the `fp` and `block` records of the FWR class, and r_Z is built according to the fixed-point or floating-point and the possible block representations [17].

References :
[17] T. Hilaire, P. Chevrel, and J. Whidborne. A unifying framework for finite wordlength realizations. IEEE Trans. on Circuits and Systems, 8(54), August 2007.

7.4.2 `computeAZBZCZDZWcWo`

Purpose :
Compute the A_Z, B_Z, C_Z and D_Z matrices and the gramians of a FWR object

Syntax :
`R=updateAZBZCZDZWoWc(R)`

Parameters :
`R` : FWR object

Description :

Internal function

Compute the matrices A_Z, B_Z, C_Z, D_Z and the controllability and observability gramians associated with a FWR object.
$A_Z \in \mathbb{R}^{n\times n}$, $B_Z \in \mathbb{R}^{n\times m}$, $C_Z \in \mathbb{R}^{p\times n}$ and $D_Z \in \mathbb{R}^{p\times m}$ are given by:

$$A_Z = KJ^{-1}M + P, \qquad B_Z = KJ^{-1}N + Q, \tag{197}$$
$$C_Z = LJ^{-1}M + R, \qquad D_Z = LJ^{-1}N + S. \tag{198}$$

and the gramians are the solutions to the Lyapunov equations

$$W_c = A_Z W_c A_Z^\top + B_Z B_Z^\top \tag{199}$$
$$W_o = A_Z^\top W_o A_Z + C_Z^\top C_Z \tag{200}$$

7.4.3 `computeJtoS`

Purpose :
Compute the J, K, L, M, N, P, Q, R, S, W_J, ..., W_J, of a FWR object from the Z and W_Z matrices of this object

Syntax :
`R=computeJtoS(R)`

Parameters :
`R` : FWR object

Description :

Internal function

This function updates the matrices J, K, L, M, N, P, Q, R, Q, W_J, W_K, W_L, W_M, W_N, W_P, W_Q, W_R and W_S from the matrices Z and W_Z.
It is called when a new value for Z or W_Z is given.

See also :
computeZ

7.4.4 computelmnp

Purpose :
Compute the dimension l, m, n and p of a FWR object (from its parameters J, K, ..., S) and check the dimensions

Syntax :
`R=updatelmnp(R)`

Parameters :
`R` : FWR object

Description :

Internal function

Compute the size l, m, n and p of a FWR realization.
It also checks the concordance of the size of matrices J, K, L, M, N, P, Q, R, S .

7.4.5 computeZ

Purpose :
Compute the Z and W_Z of a FWR object from its parameters J, ..., S, W_J, ..., W_S.

Syntax :
`R = updateZ(R)`

Parameters :
`R` : FWR object

Description :

Internal function

This function updates the matrices Z and W_Z from the matrices J, K, L, M, N, P, Q, R, Q, W_J, W_K, W_L, W_M, W_N, W_P, W_Q, W_R and W_S.
It is called when a new value for one of these matrices is given.

See also :
computeJtoS

7.4.6 deigdZ

Purpose :
Compute $M_1^\top \frac{\partial \lambda}{\partial A} M_2^\top$. This function used to compute the pole sensitivity (open-loop and closed-loop)

Syntax :
```
[dlambda_dZ, dlk_dZ] = dleigdZ( A, M1, M2, Z, moduli)
```

Parameters :

`dlambda_dZ`	: the pole sensitivity matrix
`dlk_dZ`	: pole sensitivity matrices for each pole
`A`	: matrix from whom the eigenvalues are taken
`M1,M2`	: such that $\frac{\partial \lambda}{\partial Z} = M1^\top \frac{\partial \lambda}{\partial A} M2^\top$
`moduli`	: 1 (default value) : compute $\frac{\partial \lvert\lambda\rvert}{\partial Z}$ (the sensitivity of the moduli of the eigenvalues)
	: 0 : compute $\frac{\partial \lambda}{\partial Z}$ (without the moduli)

Description :

Internal function

This function computes

$$M_1^\top \frac{\partial \lambda_k}{\partial A} M_2^\top \tag{201}$$

where the λ_k are the eigenvalues of A.
This is done by the followin lemma[53]:

Lemma 4 *Let $M \in \mathbb{R}^{n\times n}$ be diagonalisable. Let $(\lambda_k)_{1\leqslant k\leqslant n}$ be its eigenvalues, and $(x_k)_{1\leqslant k\leqslant n}$ the corresponding right eigenvectors. Denote $M_x \triangleq \big(x_1, x_2, \ldots, x_n\big)$ and $M_y = \big(y_1, y_2, \ldots, y_n\big) \triangleq M_x^{-H}$. Then*

$$\frac{\partial \lambda_k}{\partial M} = y_k^* x_k^\top \quad \forall k = 1, \ldots, n \tag{202}$$

and

$$\frac{\partial\,|\lambda_k|}{\partial M} = \frac{1}{|\lambda_k|} Re\left(\lambda_k^* \frac{\partial \lambda_k}{\partial M}\right) \tag{203}$$

where $\cdot^*$ denotes the conjugate operation, $Re(\cdot)$ the real part and $\cdot^H$ the transpose conjugate operator.

See also :
MsensPole, MsensPole_cl, Mstability

7.4.7 mylyap

Purpose :
Solve the continuous-time Lyapunov equations adapted from `lyap.m` (S.N. Bangert The MathWorks, Inc.)

Syntax :
```
X = mylyap(A, B, C, ua, ta)
```

Parameters :
`X` : solution of the continuous-time Lyapunov equation
`A,B,C` : parameters of the equation
`ua,ta` : precomputed values

Description :

Internal function

This function executes the same algorithm described in `lyap.m` (S.N. Bangert The MathWorks, Inc.), except that the values `ua` and `ta` are already computed. It permits to save some computational time when a lot of Lyapunov equations have to be solved with the same value A.

See also :
w_prod_norm_SISO

7.4.8 scalprodCfloat

Purpose :
Return the C-code corresponding to a fixed-point scalar product (the vector of coefficient `P` by the vector of variables `name`). Ex: P(1)*name(1) + P(2)*name(2) + ... + P(n)*name(n)

Syntax :
```
S = scalprodCfloat( P, name)
```

Parameters :
`S` : returned string
`P` : vector of coefficients used in the scalar product
`name` : name of the variables

Description :

Internal function

This function is called by `algorithmCfloat` for each scalar product to be done. It returns the C-code corresponding.
`P` correspond to the vector of coefficients to use, and `name` to the vector of variables' name to use.

See also :
`algorithmCfloat`, `scalprodVHDL`, `scalprodMATLAB`

7.4.9 scalprodMATLAB

Purpose :
Write the MATLAB code corresponding to a fixed-point scalar product (the vector of coefficient `P` by the vector of variables `name`). Ex: P(1)*name(1) + P(2)*name(2) + ... + P(n)*name(n)

Syntax :
```
scalprodMATLAB( fiel, P, name, gamma, shift, strAcc)
```

Parameters :
`file` : file id, where the scalar product is written
`P` : vector of coefficients used in the scalar product
`name` : name of the variables
`gamma` : fractional part of the coefficients P
`shift` : shift to apply after each multiplication
`strAcc` : name of the accumulator

Description :

Internal function

This function is called by implementMATLAB for each scalar product to be done.
It writes the corresponding MATLAB code in a file.
`P` corresponds to the vector of coefficients to use, and `name` to the vector of variables' name to use.

See also :
implementMATLAB, scalprodCfloat, scalprodVHDL

7.4.10 `scalprodVHDL`

Purpose :
Write the VHDL code corresponding to a fixed-point scalar product (the vector of coefficient `P` by the vector of variables `name`). Ex: P(1)*name(1) + P(2)*name(2) + ... + P(n)*name(n)

Syntax :
```
S = scalprodVHDL( file, P, name, gamma, shift, strAcc)
```

Parameters :

`S`	: returned string
`P`	: vector of coefficients used in the scalar product
`name`	: name of the variables
`gamma`	: fractional part of the coefficients P
`shift`	: shift to apply after each multiplication
`finalshift`	: shift to apply at the end

Description :

Internal function

This function is called by implementVHDL for each scalar product to be done.
It writes the corresponding VHDL code in a file.
`P` corresponds to the vector of coefficients to use, and `name` to the vector of variables' name to use.

See also :
implementVHDL, scalprodCfloat, scalprodMATLAB

7.4.11 w_prod_norm

Purpose :
Compute the weighting L_2-norm of the system composed by $G \circledast H = Vec(G).(Vec(H^\top))^\top$ Each transfer function is weighted by the weighting matrix W. G and H are defined by their state-space matrices Ag,Bg,Cg,Dg and Ah,Bh,Ch,Dh.

Syntax :
`[N, MX]= w_prod_norm( Ag,Bg,Cg,Dg, Ah,Bh,Ch,Dh, W)`

Parameters :

`N`	: weighted norm
`MX`	: sensibility matrix of $G \circledast H$
`Ag,Bg,Cg,Dg`	: state-space matrices of G
`Ah,Bh,Ch,Dh`	: state-space matrices of H
`W`	: weighting matrix

Description :

Internal function

From two MIMO state-space system G and H, this function computes the weighting L_2-norm of the system composed by

$$G \circledast H = Vec(G).(Vec(H^\top))^\top \tag{204}$$

Each transfer function is weighted by the weighting matrice W G and H are defined by their state-space matrices $G := (A_G, B_G, C_G, D_G)$ and $H := (A_H, B_H, C_H, D_H)$ The results N and MX are given by:

$$MX_{ij} \triangleq \left\| (G \circledast H)_{ij} W_{ij} \right\|_2 \tag{205}$$

$$\begin{aligned} N &\triangleq \|(G \circledast H) \times W\|_2^2 && (206)\\ &= \|MX\|_F^2 && (207) \end{aligned}$$

This function is used by `MsensH` and `MsensH_cl`.

See also :
`MsensH`, `MsensH_cl`, `w_prod_norm_SISO`

7.4.12 w_prod_norm_SISO

Purpose :
Compute the weighting L_2-norm of the system composed by $G \circledast H = Vec(G).(Vec(H^\top))^\top$ Each transfer function is weighted by this weighting matrix W. G and H are SIMO and MISO state-space system defined by their state-space matrices Ag,Bg,Cg,Dg and Ah,Bh,Ch,Dh.

Syntax :
`[N, MX]= w_prod_normSISO( Ag,Bg,Cg,Dg, Ah,Bh,Ch,Dh, W)`

Parameters :

`N`	: weighted norm
`MX`	: sensibility matrix of $G \circledast H$
`Ag,Bg,Cg,Dg`	: state-space matrices of G
`Ah,Bh,Ch,Dh`	: state-space matrices of H
`W`	: weighting matrix

Description :

Internal function

From two SIMO and MISO state-space system G and H, this function compute the weighting L_2-norm of the system composed by

$$G \circledast H = Vec(G).(Vec(H^\top))^\top \tag{208}$$

Each transfer function is weighted by the weighting matrix W G and H are defined by their state-space matrices $G := (A_G, B_G, C_G, D_G)$ and $H := (A_H, B_H, C_H, D_H)$ The results N and MX are given by:

$$MX_{ij} \triangleq \left\| (G \circledast H)_{ij} W_{ij} \right\|_2 \tag{209}$$

$$\begin{aligned} N &\triangleq \|(G \circledast H) \times W\|_2^2 && (210)\\ &= \|MX\|_F^2 && (211) \end{aligned}$$

This function is used by `MsensH` and `MsensH_cl`. In that case, due to the SISO size of the transfer function $(G \circledast H)_{ij}$, it is possible to use that

$$\begin{aligned} MX_{ij} &= \|(GH)_{i,j}\|_2 && (212)\\ &= \left\| \left(\begin{array}{cc|c} A_G & 0 & (B_G)_i \\ B_H C_G & A_H & B_H D_G \\ \hline D_H C_G & C_H & D_H D_G \end{array} \right)_{i,j} \right\|_2 && (213) \end{aligned}$$

See also :
`MsensH`, `MsensH_cl`, `w_prod_norm`

7.5 FWS class methods

display	Display the realization (dimensions, Z and the parameters)
FWS	Constructor of the FWS class.
genCostFunction	G eneric cost function for optimization of a FWS
get	Get some properties of a FWS object (or list the properties if propName is ignored)
getValues	Return the parameters' value (cells of values)
MsensPole	Compute the open-loop pole sensitivity measure for a FWS object.
MsensPole_cl	Compute the closed-loop pole sensitivity measure for a FWS object.
Mstability	Compute the closed-loop pole sensitivity stability related measure for a FWS object.
optim	Find the optimal realization, according to the measureFun measure, in the set of structured equivalent realizations
RNG	Compute the open-loop Roundoff Noise Gain for a FWS.
RNG_cl	Compute the closed-loop Roundoff Noise Gain for a FWS.
set	Set some properties of a FWS object
setFPIS	Set the Fixed-Point Implementation Scheme (FPIS) of an FWS object
ss	Convert a FWS object into a ss (state-space) object (equivalent state-space)
subsasgn	Subscripted assign for FWS object.
subsref	Subscripted reference for FWS object.
tf	Convert a FWS object into a tf object (transfer function)

7.5.1 display

Purpose :
Display the realization (dimensions, Z and the parameters)

Syntax :
`display(S)`

Parameters :
`S` : FWS object

Description :
Display the dimensions (inputs, outputs, states and intermediate variables), Z

and the associated parameters.

Example :

```
has 1 input, 1 output, 3 states, and 0 intermediate variable.
Z=
5.5298e-01  -5.3793e-01   2.9172e-02   6.7157e-01
5.3793e-01   9.7113e-02  -3.5628e-01  -3.2376e-01
2.9172e-02   3.5628e-01  -7.2857e-02   7.9289e-02
6.7157e-01   3.2376e-01   7.9289e-02   9.8531e-02
T=
1     0     0
0     1     0
0     0     1
```

See also :
display

7.5.2 FWS

Purpose :
Constructor of the FWS class. A structuration is characterized by an initial realization and a `way` to transform this realization

Syntax :
`S = FWS(Rini, UYWfun, Rfun, dataFWS, param1Name, param1Value, param2Name, param2Value, ...)`

Parameters :

`S`	: FWS object
`Rini`	: initial realization (FWR object)
`UYWfun`	: handle to a function that links the parameters to the transformation matrices U,Y and W
`Rfun`	: handle to a function that links the parameters to the new realization
	: only ONE of these two functions must be provided
`dataFWS`	: cells of extra datas
`paramName`	: parameters' names
`paramValue`	: initial value for the parameters

Description :
This function is called to construct a FWS object.
Only one of the two functions `UYWfun` `Rfun` must be given (a handle to a function is defined by `@` + `name of the function` - see Matlab's documentation for more informations on function's handle).

These functions must satisfy the specifications explain in section 6.2.
The names and values of each parameter are given by pair.

Example :
Let us consider a state-space realization (A, B, C, D).
The equivalent realizations are given by the state-space $(T^{-1}AT, T^{-1}B, CT, D)$.
This correspond to the following SIF

$$Z = \begin{pmatrix} . & . & . \\ . & A_q & B_q \\ . & C_q & D_q \end{pmatrix} \tag{214}$$

and the $\mathcal{YUW}$ transformation with $\mathcal{U} = T$, $\mathcal{Y} = \mathcal{W} = I_l$. Then, to create a state-space structuration, from matrices `A`, `B`, `C` and `D`, with a parameter `T`, one should create a UYWfun like

```
% UYW function for the classical state-space structuration
function [U,Y,W,cost_flag] = UYW_SS( Rini, paramsValue, dataFWS)
%test if T is singular
if (cond(paramsValue{1})>1e10)
cost_flag=0;
paramsValue{1} = eye(size(paramsValue{1}));
else
cost_flag=1;
end
% compute U,W,Y
Y = eye(0);
W = eye(0);
U = paramsValue{1};
```

The `cost_flag` could return 0 if the paramsValue proposed is not acceptable (here a non-invertible matrix).
Then the structuration is created by

```
Rini =  SS2FWR(A,B,C,D);
S = FWS( Rini, @UYW_SS, [], [], 'T', eye(R.n));
```

(there is no need for a `dataFWS`). Even if it is not preferrable, it is also possible to create this FWS with a `Rfun` function.
So a function that creates a new state-space realization from the `paramsValue` is needed

```
% UYW function for the classical state-space structuration
function [R,cost_flag] = Rfun_SS( Rini, paramsValue, dataFWS)
%test if T is singular
if (cond(paramsValue{1})>1e10)
cost_flag=0;
paramsValue{1} = eye(size(paramsValue{1}));
```

```
else
cost_flag=1;
end
% compute the new realization
T = paramsValue{1};
R = SS2FWR( inv(T)*Rini.P*T, inv(T)*Rini.Q, Rini.R*T, Rini.S );
```

and the FWS object is defined by

```
Rini =  SS2FWR(A,B,C,D);
S = FWS( Rini, [], @Rfun_SS, [], 'T', eye(R.n));
```

In that case, the optimization process will have to compute for each iteration a new realization (with the `Rfun` function) and then compute the associated FWL measure; whereas in the first case, the FWL measure is directly compute from the $\mathcal{U}$, $\mathcal{Y}$, $\mathcal{W}$ matrices.

See also :
FWR

7.5.3 genCostFunction

Purpose :
G eneric cost function for optimization of a FWS

Syntax :
`[cost_value, cost_flag] = genCostFunction( x, S, freeparams, isaFWSMethod, l2scaling,Umax,delta, measureFun, ...)`

Parameters :

`cost_value`	: value of the measure for parameter x
`cost_flag`	: 0 if parameter x is incorrect
	: 1 otherwise
`x`	: vector of parameter used by optimization (ASA, fminsearch, ...)
`S`	: FWS object
`freeparams`	: vector that indicates which parameters are free to be optimized
`isaFWSMethod`	: boolean that indicates that if the measure is a FWS's method
`l2scaling`	: boolean - tell if R is L_2-scaled
`Umax`	: magnitude value for the input - used for the L_2-scaling
`delta`	: δ parameter for L_2-scaling
`measureFun`	: handle of the measure function
`...`	: the extra parameters are given to the measure function

Description :
This function is used internally only, *but* should be visible for fminsearch, ASA, ...
It just calls `genCostFunctionS` (or `genCostFunctionR`) that are real FWS's methods (fminsearch or ASA needs a cost function with x as first argument, but we wanted this function to be a FWS's method).

See also :
optim, genCostFunctionR, genCostFunctionS

7.5.4 get

Purpose :
Get some properties of a FWS object (or list the properties if propName is ignored)

Syntax :
```
value = get(S, propName)
```

Parameters :
`value`	: value of the property
`S`	: FWS object
`propName`	: name of the property (string)

Description :
This function is most of the time called by `subsref`.
All the fields are actually changeable (may change).

See also :
get

7.5.5 getValues

Purpose :
Return the parameters' value (cells of values)

Syntax :
```
v = getValues(S)
```

Parameters :
`v` : cells of values
`S` : FWS object

Description :
This function returns the parameters value that are encapsuled in the `paramsValue` field.
It is used by UYWfunctions.

See also :
FWS

7.5.6 `MsensPole`

Purpose :
Compute the open-loop pole sensitivity measure for a FWS object. The computation is based on the UYW-transform

Syntax :
```
M = MsensPole( S, U,Y,W, moduli)
```

Parameters :

`M`	: pole sensitivity measure
`S`	: FWS object
`U,Y,W`	: transformation matrices
`moduli`	: 1 (default value) : compute $\frac{\partial \lvert\lambda\rvert}{\partial Z}$ (the sensitivity of the moduli of the eigenvalues)
	: 0 : compute $\frac{\partial \lambda}{\partial Z}$ (without the moduli)

Description :
This function computes the open-loop pole sensitivity measure for a FWS object. It is based on the $\mathcal{UYW}$-transform. If we consider $\mathcal{T}_1$ and $\mathcal{T}_2$ such that

$$Z_1 = \mathcal{T}_1 Z_0 \mathcal{T}_2 \tag{215}$$

$$\mathcal{T}_1 = \begin{pmatrix} \mathcal{Y} & & \\ & \mathcal{U}^{-1} & \\ & & I_p \end{pmatrix}, \quad \mathcal{T}_2 = \begin{pmatrix} \mathcal{W} & & \\ & \mathcal{U} & \\ & & I_m \end{pmatrix} \tag{216}$$

then the sensitivity measure for Z_1 can be computed from the sensitivity for Z_0 with

$$\left.\frac{\partial \lvert\lambda_k\rvert}{\partial Z}\right|_{Z_1} = \mathcal{T}_1^{-\top} \left.\frac{\partial \lvert\lambda_k\rvert}{\partial Z}\right|_{Z_0} \mathcal{T}_2^{-\top} \tag{217}$$

These matrix $\left.\frac{\partial \lvert\lambda_k\rvert}{\partial Z}\right|_{Z_0}$ are stored in the `dataMeasure` field.

See also :
MsensPole_cl, MsensPole

References :
[14] T. Hilaire, P. Chevrel, and J.-P. Clauzel. Pole sensitivity stability related measure of FWL realization with the implicit state-space formalism. In 5th IFAC Symposium on Robust Control Design (ROCOND'06), July 2006.
[17] T. Hilaire, P. Chevrel, and J. Whidborne. A unifying framework for finite wordlength realizations. IEEE Trans. on Circuits and Systems, 8(54), August 2007.

7.5.7 `MsensPole_cl`

Purpose :
Compute the closed-loop pole sensitivity measure for a FWS object. The computation is based on the UYW-transform

Syntax :
`M = MsensPole_cl( S, U,Y,W, Sysp,moduli)`

Parameters :

`M`	: pole sensitivity measure
`S`	: FWS object
`U,Y,W`	: transformation matrices
`Sysp`	: plant system (ss object)
`moduli`	: 1 (default value) : compute $\frac{\partial\lvert\bar{\lambda}\rvert}{\partial Z}$ (the sensitivity of the moduli of the eigenvalues)
	: 0 : compute $\frac{\partial\bar{\lambda}}{\partial Z}$ (without the moduli)

Description :
This function computes the closed-loop pole sensitivity measure for a FWS object. It is based on the $\mathcal{UYW}$-transform.
If we consider $\mathcal{T}_1$ and $\mathcal{T}_2$ such that

$$Z_1 = \mathcal{T}_1 Z_0 \mathcal{T}_2 \tag{218}$$

$$\mathcal{T}_1 = \begin{pmatrix} \mathcal{Y} & & \\ & \mathcal{U}^{-1} & \\ & & I_p \end{pmatrix}, \quad \mathcal{T}_2 = \begin{pmatrix} \mathcal{W} & & \\ & \mathcal{U} & \\ & & I_m \end{pmatrix} \tag{219}$$

then the sensitivity measure for Z_1 can be computed from the sensitivity for Z_0 with

$$\left.\frac{\partial\lvert\bar{\lambda}_k\rvert}{\partial Z}\right|_{Z_1} = \mathcal{T}_1^{-\top} \left.\frac{\partial\lvert\bar{\lambda}_k\rvert}{\partial Z}\right|_{Z_0} \mathcal{T}_2^{-\top} \tag{220}$$

These matrix $\left.\frac{\partial\lvert\bar{\lambda}_k\rvert}{\partial Z}\right|_{Z_0}$ are stored in the `dataMeasure` field.

See also :
`MsensPole`, `MsensPole_cl`

References :
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.5.8 `Mstability`

Purpose :
Compute the closed-loop pole sensitivity stability related measure for a FWS object. The computation is based on the UYW-transform

Syntax :
```
M = Mstability( S, U,Y,W, Sysp, moduli)
```

Parameters :

`M`	: pole sensitivity measure
`S`	: FWS object
`U,Y,W`	: transformation matrices
`Sysp`	: plant system (ss object)
`moduli`	: 1 (default value) : compute $\frac{\partial\lvert\bar{\lambda}\rvert}{\partial Z}$ (the sensitivity of the moduli of the eigenvalues)
	: 0 : compute $\frac{\partial\bar{\lambda}}{\partial Z}$ (without the moduli)

Description :
This function computes the closed-loop pole sensitivity stability related measure for a FWS object. It is based on the UYW-transform.
If we consider $\mathcal{T}_1$ and $\mathcal{T}_2$ such that

$$Z_1 = \mathcal{T}_1 Z_0 \mathcal{T}_2 \tag{221}$$

$$\mathcal{T}_1 = \begin{pmatrix} \mathcal{Y} & & \\ & \mathcal{U}^{-1} & \\ & & I_p \end{pmatrix}, \quad \mathcal{T}_2 = \begin{pmatrix} \mathcal{W} & & \\ & \mathcal{U} & \\ & & I_m \end{pmatrix} \tag{222}$$

then the sensitivity measure for Z_1 can be computed from the sensitivity for Z_0 with

$$\left.\frac{\partial\lvert\bar{\lambda}_k\rvert}{\partial Z}\right|_{Z_1} = \mathcal{T}_1^{-\top} \left.\frac{\partial\lvert\bar{\lambda}_k\rvert}{\partial Z}\right|_{Z_0} \mathcal{T}_2^{-\top} \tag{223}$$

These matrix $\left.\frac{\partial\lvert\bar{\lambda}_k\rvert}{\partial Z}\right|_{Z_0}$ are stored in the `dataMeasure` field.

See also :
`Mstability`, `MsensPole_cl`

References :
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.5.9 `optim`

Purpose :
Find the optimal realization, according to the `measureFun` measure, in the set of structured equivalent realizations

Syntax :
```
S = optim( S, options, measureFun, ...)
```

Parameters :

`S`	: FWS object
`options`	: cells of pairs (string/value) to define the options of the optimization
	: `'param1'`, value1, `'param2'`, value2
	: - `method` : `'newton'` (default), `'simplex'` or `'ASA'`
	: `'newton'` refers to the Quasi-Newton algorithm used by fminunc and fmincon
	: `'simplex'` refers to the simplex algorithm used by fminsearch
	: and `'ASA'` refers to the Annealed Simulated Algorithm
	: - `'l2scaling'` : `'no'` (default) ,`'yes'` or `'relaxed'`
	: - `'useFWSmeasure'` : `'yes'` (default) or `'no'`
	: - `'fixedParameter'` : a parameter to be fixed during optimization
	: - `'matFileName'` : filename of the mat-file created to store the result
	: (default) : measureFun + funName + date
	: [ ] : no mat-file is created
	: if ASA method is used, the name is also used for the log-file
	: - `'MinMax'` : scalar that fix the maximum value for the
	: parameters (min=-max)
	: - `'Min'` and `'Max'` : used to fix individually the min and max values. The values of `'Min'` and `'Max'` may have the same size as the parameters
	: - the other string/values are given to the fminsearch/ASA function
`measureFun`	: handle to the measure function
`...`	: the extra parameters are given to the measure function

Description :
This method is the main method of the FWS class. It allows to search over the

set of the structured realizations (defined with the FWS object), by running some optimization algorithms, like `fminsearch` or `ASA`.
Some options could be passed to this function, in order to parametrize the optimization:

- the `method` option can take the following values
  - `'newton'` (default): to use the Quasi-Newton algorithm (`fminunc` or `fmincon` functions)
  - `'simplex'`: to use the simplex algorithm (`fminsearch`)
  - `'ASA'`: to use the Annealed Simulated Algorithm[4]
- the `l2scaling` option indicates if a L_2-scaling should be applied. It can take the following values:
  - `'no'` (default): no L_2-scaling constraints is applied
  - `'yes'`: the classical L_2-scaling constraints are applied (see `l2scaling`)
  - `'relaxed'`: the relaxed-L_2-scaling constraints are applied (see `relaxedl2scaling`)
- the `useFWSmeasure` option (`'yes'` (default) or `'no'`) forces the use of the $\mathcal{UYW}$-function
- the `fixedParameter` allows to fix some parameters during the optimization. The value associated should be the name of the parameters. Several parameters coudld be fixed.
- the `matFileName` sets the filename of the .mat file that is created at the end of the optimization process to store the final optimized result. By default, this filename is defined by the name of the measure function plus the name of the structuration (given by the name of the UYWfun or the Rfun) plus the date. If the filename is empty, no .mat file is storned.
  In case of ASA method, this name is also used for the log-file created by asamin.
- the `MinMax` allows to set the minimum and maximum values for the parameters. Here only a scalar is required (the maximum value is set to this scalar, the minimum values to the opposite). In order to set individually the min and max values, uses `Min` and `Max` options, where you have to pass a vector this the same size as the parameters.
- the extra options are given to the optimization algorihtm (`fminunc`, `fmincon`, `fminsearch`, `ASA`. See Matlab's`optimset` or `asamin` documentation). Classical option (for quasi-Newton and convex algorithm) is {`'Display','Iter'`}

[4] In that case, `ASA` and its Matlab gateway `asamin` should be correctly installed, and added in the Matlab's path)

Example :

```
>>S = SS2FWS(A,B,C,D);
>>options = {'method','newton','Display','Iter', 'l2scaling','yes');
>>Sopt = optim( S, options, @MsensH_cl, Plant);
```

The `Plant` value is given to the function `MsensH_cl` as a supplementary parameter. The pair of options `'Display','Iter'` is passed to the newton algorithm (`fminunc` here), so as the iterations are displayed.
So, these commands define a state-space structuration, and search for the optimal L_2-scaled realization according to the closed-loop input-output sensitivity (with `Plant` as a plant to be controlled, see section 3.5.5).
`Sopt` contains now the structuration `S` with the optimal realization in `R` and the optimal parameter in `T`.

7.5.10 RNG

Purpose :
Compute the open-loop Roundoff Noise Gain for a FWS. The computation is based on the UYW-transform

Syntax :
`G = RNG(S,U,Y,W)`

Parameters :

`G`	: roundoff noise gain
`S`	: FWS object
`U,Y,W`	: transformation matrices

Description :
This function computes the open-loop Roundoff Noise Gain for a FWS. The computation is based on the $\mathcal{UYW}$-transform.
It is based on the `RNG` function.
The matrix d_Z is computed once and stored in the `dataMeasure` field.

See also :
`RNG`

References :
[19] T. Hilaire, D. Ménard, and O. Sentieys. Roundoff noise analysis of finite wordlength realizations with the implicit state-space framework. In 15th European Signal Processing Conference (EUSIPOC'07), September 2007.

7.5.11 RNG_cl

Purpose :
Compute the closed-loop Roundoff Noise Gain for a FWS. The computation is based on the UYW-transform

Syntax :
```
G = RNG_cl(S,U,Y,W, Plant, tol)
```

Parameters :

`G`	: roundoff noise gain
`S`	: FWS object
`U,Y,W`	: transformation matrices
`Plant`	: ss of the plant
`tol`	: tolerance on trivial parameters (default=1e-8)

Description :
This function computes the closed-loop Roundoff Noise Gain for a FWS. The computation is based on the UYW-transform.
It is based on the RNG_cl function.
The matrix d_Z is computed once and stored in the `dataMeasure` field.

See also :
RNG, RNG_cl

References :
[18] T. Hilaire, P. Chevrel, and J. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

7.5.12 set

Purpose :
Set some properties of a FWS object

Syntax :
```
S = set(S, propName, value)
```

Parameters :

`S`	: FWR object
`propName`	: name of the property
`value`	: new value for this property

Description :
This function is most of the time called by `subsasgn`.
The initial realization `Rini` can be changed (if the size doesn't change), so can the parameters of the structuration. It is not allowed to modified other fields.

See also :
`get`, `subsasgn`, `set`

7.5.13 `setFPIS`

Purpose :
Set the Fixed-Point Implementation Scheme (FPIS) of an FWS object (the wordlength may be a matrix or scalar. The scalar case is used to set all the wordlength to the same length)

Syntax :
```
S = setFPIS( S, betaU, Umax, betaZ, betaT, betaX, betaY, betaADD, betaG,
method );
S = setFPIS( S, FPIS);
S = setFPIS( S, FPISname,Umax);
```

Parameters :

`S`	: FWS object
`FPIS`	: an other Fixed-Point Implementation Scheme (a structure with betaU, Umax, betaZ, betaT, betaX, betaY, betaADD, betaG and method)
`FPISname`	: `'DSP8'` or `'DSP16'`
`betaU`	: wordlength of U (inputs)
`Umax`	: maximum value of U (necessary to set gammaU)
`betaZ`	: wordlength of the coefficients
`betaT, betaX, betaY`	: wordlength of the intermediate variables T, the states X and the outputs
`betaADD`	: wordlength of the accumulators
`betaG`	: nb of guard bits in the accumulators
`method`	: `'RBM'` (default) Roundoff Before Multiplication
	: `'RAM'` Roundoff After Multiplication

Description :
This function sets the Fixed-Point Implementation Scheme (FPIS) of an FWS object. It means that all the associated realization (`Rini` and `R`) whill have the same FPIS (the values can changed).

See also :
`setFPIS`

References :
[20] T. Hilaire, D. Ménard, and O. Sentieys. Bit accurate roundoff noise analysis of fixed-point linear controllers. In Proc. IEEE International Symposium on Computer-Aided Control System Design (CACSD'08), September 2008.

7.5.14 ss

Purpose :
Convert a FWS object into a ss (state-space) object (equivalent state-space)

Syntax :
```
Sys = ss(S,Te)
```

Parameters :
`Sys` : ss object
`S` : FWS object
`Te` : period (default=1)

Description :
Give the equivalent state-space (`ss` object). The current realization (`R`) is considered.
It is defined with matrices A_Z, B_Z, C_Z and D_Z (see eq. (7) and (8)).

See also :
ss

7.5.15 subsasgn

Purpose :
Subscripted assign for FWS object. Here, `'S.prop=value'` is equivalent to `set(S,'prop',value)`

Syntax :
```
S = subsasgn(S,Sub,value)
```

Parameters :
`S` : FWS object
`Sub` : subassignment layers
`value` : value of the assignation

Description :
These functions are called internally when operators `[]`, `()` and `.` are applied on a FWR object.
Only the operator `.` is valid, and links to `set` and `get` functions. The command `S.field` returns the field `field` of `S` (internally, `get(S,'field')` is called), and `S.field=vaue` set the field `field` of `S`

Example :
```
S.Rini = R
S.R
S.R.Z(3,3) = 0;
S.Rini.WZ = zeros(n);
```

See also :
set, get, subsref, subsasgn

7.5.16 subsref

Purpose :
Subscripted reference for FWS object. Here, `'S.prop'` is equivalent to `'get(S,prop)'`

Syntax :
```
value = subsref(S,Sub)
```

Parameters :
`value` : returned value
`S` : FWS object
`Sub` : layers of subreferencing

Description :
These functions are called internally when operators `[]`, `()` and `.` are applied on a FWR object.
Only the operator `.` is valid, and links to `set` and `get` functions. The command `S.field` returns the field `field` of `S` (internally, `get(S,'field')` is called), and `S.field=vaue` set the field `field` of `S`

Example :
```
S.Rini = R
S.R
S.R.Z(3,3) = 0;
S.Rini.WZ = zeros(n);
```

See also :
set, get, subsasgn, subsref

7.5.17 tf

Purpose :
Convert a FWS object into a tf object (transfer function)

Syntax :
```
H = tf(S,Te)
```

Parameters :
`H` : tf object
`S` : FWS object
`Te` : period (default=1)

Description :
Give the transfer function of the realization. The current realization (`R`) is considered.
It is defined with matrices A_Z, B_Z, C_Z and D_Z (see eq. (7) and (8)) by:

$$H : z \mapsto C_Z \left(zI_n - A_Z\right)^{-1} B_Z + D_Z \tag{224}$$

See also :
`ss`, `tf`

7.6 `FWS` private functions

`genCostFunctionR`	Generic cost function for optimization of a FWS.
`genCostFunctionS`	Generic cost function for optimization of a FWS.
`myoptions_asamin`	Set of options for the ASA algorithm (through asamin)
`updateR`	Update the R value from the (new) paramsValue

7.6.1 `genCostFunctionR`

Purpose :
Generic cost function for optimization of a FWS. Here the measure is computed from the new realization R

Syntax :
```
[cost_value, cost_flag] = genCostFunctionR( S, x, freeparams, l2scaling,
Umax, delta, measureFun, ...)
```

Parameters :

`cost_value`	: value of the measure for parameter x
`cost_flag`	: 0 if parameter x is incorrect
	: 1 otherwise
`S`	: FWS object
`x`	: vector of parameter used by optimiser (ASA, fminsearch, ...)
`freeparams`	: vector that indicates which parameters are free to be optimized
`l2scaling`	: boolean - tell if R is L_2-scaled
`Umax`	: magnitude value for the input - used for the L_2-scaling
`delta`	: delta parameter for L_2-scaling
`measureFun`	: handle of the measure function
`...`	: the extra parameters are given to the measure function

Description :

Internal function

This function is a generic cost function. From the vector x, it rebuilds the parameters' value, a new realization (with the `updateR` method), and then compute the associated cost value.
In `genCostFunctionS`, a new realization is not directly computed, only the $\mathcal{U}$, $\mathcal{Y}$ and $\mathcal{W}$ matrices are used to compute the cost value.

See also :
`optim`, `genCostFunction`, `genCostFunctionS`, `updateR`

7.6.2 genCostFunctionS

Purpose :
Generic cost function for optimization of a FWS. Here the measure is computed direclty from the U,Y,W (and other values computed one time and stored in data) in order to decrease the computational time (in `genCostFunctionR`, it's computed from the new realization R)

Syntax :
```
[cost_value, cost_flag] = genCostFunctionS( S, x, freeparams, l2scaling,
Umax, delta, measureFun, ...)
```

Parameters :

`cost_value`	: value of the measure for parameter x
`cost_flag`	: 0 if parameter x is incorrect
	: 1 otherwise
`S`	: FWS object
`x`	: vector of parameter used by optimiser (ASA, fminsearch, ...)
`freeparams`	: vector that indicates which parameters are free to be optimized
`l2scaling`	: boolean - tell if R is L_2-scaled
`Umax`	: magnitude value for the input - used for the L_2-scaling
`delta`	: delta parameter for L_2-scaling
`measureFun`	: handle of the measure function
`...`	: the extra parameters are given to the measure function

Description :

Internal function

This function is a generic cost function. From the vector x, it rebuilds the matrices $\mathcal{U}$, $\mathcal{Y}$ and $\mathcal{W}$ (thanks to the `UYWfun` function) and then compute the associated cost value.
In `genCostFunctionS`, a new realization is directly computed from the parameters' value (thanks to the `Rfun` function), and used to compute the cost value.

See also :
`optim`, `genCostFunction`, `genCostFunctionR`

7.6.3 `myoptions_asamin`

Purpose :
Set of options for the ASA algorithm (through asamin)

Syntax :
`myoptions_asamin`

Description :
One can change this script and put its preferred parameters.
It is important to remark that these parameters could, of course, be passed to asamin through the options of the `optim` method

See also :
`optim`

7.6.4 `updateR`

Purpose :
Update the R value from the (new) paramsValue

Syntax :
```
[R, cost_flag] = updateR(S)
```

Parameters :
`R` : FWR object
`S` : FWS object

Description :

Internal function

This function updates the `R` value from the (new) parameters' values (`paramsValue`). This is done via the $\mathcal{U}$, $\mathcal{Y}$ and $\mathcal{W}$ matrices or directly, depending on the FWS object and the `UYWfun` and `Runf` functions.

8 Bibliography

References

[1] R.C. Agarwal and C.S. Burrus. New recursive digital filter structures having very low sensitivity and roundoff noise. *IEEE Trans. Circuits & Syst.*, 22(12):921–927, December 1975.

[2] D. Alazard, C. Curres, P. Apkarian, M. Gauvrit, and G. Ferreses. *Robustesse et Commande Optimale*. Cepadues Edition, 1999.

[3] J.E Bertram. The effects of quantization in sampled-feedback systems. *Trans. of the American Institute of Electrical Engineers*, 77:177–182, 1958.

[4] D.S.K. Chan. Constrained minimization of roundoff noise in fixed-point digital filters. In *Proc. IEEE Int. Conf. On Acoust., Speech, and Signal Processing (ICASSP '79)*, pages 335–339, Washington, DC, April 1979.

[5] D.S.K. Chan. The structure of recursible multidimensional discrete systems. *IEEE Trans. Autom. Control*, 25(4):663–673, August 1980.

[6] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. of Computation*, 19:297–301, 1965.

[7] Sebastian Egner, Jeremy Johnson, David Padua, Jianxin Xiong, and Markus Püschel. Automatic derivation and implementation of signal processing algorithms. *ACM SIGSAM Bull. Communications in Computer Algebra*, 35(2):1–19, 2001.

[8] Y. Feng, P. Chevrel, and T. Hilaire. A practival strategy of an efficient and sparse fwl implementation of lti filters. In *submitted to ECC'09*, 2009.

[9] M. Gevers and G. Li. *Parametrizations in Control, Estimation and Filtering Probems*. Springer-Verlag, 1993.

[10] R. Goodall. Perspectives on processing for real-time control. *Annual Reviews in Control*, 25:123–131, 2001.

[11] T. Hilaire. Low parametric sensitivity realizations with relaxed l2-dynamic-range-scaling constraints. *submitted to IEEE Trans. on Circuits & Systems II*, 2009.

[12] T. Hilaire and P. Chevrel. On the compact formulation of the derivation of a transfer matrix with respect to another matrix. Technical Report RR-6760, INRIA, 2008.

[13] T. Hilaire, P. Chevrel, and J-P. Clauzel. Low parametric sensitivity realization design for FWL implementation of MIMO controllers : Theory and application to the active control of vehicle longitudinal oscillations. In *Proc. of Control Applications of Optimisation CAO'O6*, April 2006.

[14] T. Hilaire, P. Chevrel, and J-P. Clauzel. Pole sensitivity stability related measure of FWL realization with the implicit state-space formalism. In *5th IFAC Symposium on Robust Control Design (ROCOND'06)*, July 2006.

[15] T. Hilaire, P. Chevrel, and Y. Trinquet. Implicit state-space representation : a unifying framework for FWL implementation of LTI systems. In P. Piztek, editor, *Proc. of the 16th IFAC World Congress*. Elsevier, July 2005.

[16] T. Hilaire, P. Chevrel, and J. Whidborne. Low parametric closed-loop sensitivity realizations using fixed-point and floating-point arithmetic. In *Proc. European Control Conference (ECC'07)*, July 2007.

[17] T. Hilaire, P. Chevrel, and J.F. Whidborne. A unifying framework for finite wordlength realizations. *IEEE Trans. on Circuits and Systems*, 8(54), August 2007.

[18] T. Hilaire, P. Chevrel, and J.F. Whidborne. Finite wordlength controller realizations using the specialized implicit form. Technical Report RR-6759, INRIA, 2008.

[19] T. Hilaire, D. Ménard, and O. Sentieys. Roundoff noise analysis of finite wordlength realizations with the implicit state-space framework. In *15th European Signal Processing Conference (EUSIPOC'07)*, September 2007.

[20] T. Hilaire, D. Ménard, and O. Sentieys. Bit accurate roundoff noise analysis of fixed-point linear controllers. In *Proc.* IEEE International Symposium on Computer-Aided Control System Design *(CACSD'08)*, September 2008.

[21] T. Hinamoto, H. Ohnishi, and W.-S. Lu. Minimization of L_2 sensitivity for state-space digital filters subject to L_2-dynamic-range scaling constraints. *IEEE Trans. Circuits & Syst. II*, 52(10):641–645, 2005.

[22] T. Hinamoto, S. Yokoyama, T. Inoue, W. Zeng, and W. Lu. Analysis and minimization of L_2-sensitivity for linear systems and two-dimensional state-space filters using general controllability and observability gramians. In *IEEE Transactions on Circuits and Systems, Fundamental Theory and Applications*, volume 49, september 2002.

[23] S.Y. Hwang. Minimum uncorrelated unit noise in state-space digital filtering. *IEEE Trans. on Acoust., Speech, and Signal Processing*, 25(4):273–281, August 1977.

[24] M. Ikeda, D. Šiljak, and D. White. An inclusion principle for dynamic systems. *IEEE Trans. Automatic Control*, 29(3):244–249, March 1984.

[25] L. Ingber. Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.

[26] R.S.H. Istepanian and J.F. Whidborne. Finite-precision computing for digital control systems: Current status and future paradigms. In R.S.H. Istepanian and J.F. Whidborne, editors, *Digital Controller Implementation and Fragility: A Modern Perspective*, chapter 1, pages 1–12. Springer-Verlag, London, UK, 2001.

[27] M. Iwatsuki, M. Kawamata, and T. Higuchi. Statistical sensitivity and minimum sensitivity structures with fewer coefficients in discrete time linear systems. *IEEE Trans. Circuits & Syst.*, 37(1):72–80, January 1989.

[28] J.F. Kaiser. Digital filters. In F.F. Kuo and J.F. Kaiser, editors, *System Analysis by Digital Computer*, pages 218–285. Wiley, New York, 1966.

[29] J. Knowles and E. Olcayto. Coefficient accuracy and digital filter response. *IEEE Trans. Circuits & Syst.*, 15(1):31–41, mar 1968.

[30] H-J. Ko and WS Yu. Guaranteed robust stability of the closed-loop systems for digital controller implementations via orthogonal hermitian transform. *IEEE Trans. on Systems, Man, and Cybernetics*, 34(4):1923–1932, August 2004.

[31] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee. *DSP Processor Fundamentals: Architectures and Features*. Berkeley Design Technology, Inc, Fremont, CA, 1996.

[32] G. Li. On the structure of digital controllers with finite word length consideration. *IEEE Trans. on Autom. Control*, 43(5):689–693, May 1998.

[33] G. Li, B.D.O. Anderson, M. Gevers, and J.E. Perkins. Optimal FWL design of state space digital systems with weighted sensitivity minimization and sparseness consideration. *IEEE Trans. Circuits & Syst. II*, 39(5):365–377, May 1992.

[34] G. Li and Z. Zhao. On the generalized DFIIt structure and its state-space realization in digital filter implementation. *IEEE Trans. on Circuits and Systems*, 51(4):769–778, April 2004.

[35] B. Liu. Effects of finite word-length on the accuracy of digital filters — a review. *IEEE Trans. Circuit Theory*, 18(6):670–677, 1971.

[36] W.-S. Lu and T. Hinamoto. Jointly optimized error-feedback and realization for roundoff noise minimization in state-space digital filters. *IEEE Trans. Sig. Proc.*, 53(6):2135–2145, June 2005.

[37] W.J. Lutz and S.L. Hakimi. Design of multi-input multi-output systems with minimum sensitivity. *IEEE Trans. Circuits & Syst.*, 35(9):1114–1122, September 1988.

[38] A.G. Madievski, B.D.O. Anderson, and M. Gevers. Optimum realizations of sampled-data controllers for FWL sensitivity minimization. *Automatica*, 31(3):367–379, 1995.

[39] P.E. Mantey. Eigenvalue sensitivity and state-variable selection. *IEEE Trans. Autom. Control*, 13(3):263–269, 1968.

[40] R. Middleton and G. Goodwin. *Digital Control and Estimation, a unified approach*. Prentice-Hall International Editions, 1990.

[41] P. Moroney, A.S. Willsky, and P.K. Houpt. The digital implementation of control compensators: the coefficient wordlength issue. *IEEE Trans. Autom. Control*, 25(4):621–630, August 1980.

[42] C. Mullis and R. Roberts. Synthesis of minimum roundoff noise fixed point digital filters. In *IEEE Transactions on Circuits and Systems*, volume CAS-23, September 1976.

[43] I.W. Sandberg. Floating-point-roundoff accumulation in digital-filter realizations. *Bell Syst. Tech. J.*, 46(8):1775–1791, 1967.

[44] R. Skelton and D. Wagie. Minimal root sensitivity in linear systems. *Journal of Guidance, Control and Dynamics*, 7(5):570–574, 1984.

[45] J.B. Slaughter. Quantization errors in digital control systems. *IEEE Trans. Autom. Control*, 9:70–74, 1964.

[46] V. Tavşanoğlu and L. Thiele. Optimal design of state-space digital filters by simultaneous minimization of sensibility and roundoff noise. In *IEEE Trans. on Acoustics, Speech and Signal Processing*, volume CAS-31, October 1984.

[47] L. Thiele. Design of sensitivity and round-off noise optimal state-space discrete systems. *Int. J. Circuit Theory Appl.*, 12:39–46, 1984.

[48] L. Thiele. On the sensitivity of linear state space systems. *IEEE Trans. Circuits & Syst.*, 33(5):502–510, 1986.

[49] D. Šiljak. *Decentralized Control of Complex Systems*. Academic Press, 1991.

[50] J.F. Whidborne, R. Istepanian, and J. Wu. Reduction of controller fragility by pole sensitivity minimization. *IEEE Trans. Automatic Control*, 46:320–325, 2001.

[51] D. Williamson. Roundoff noise minimization and pole-zero sensibivity in fixed-point digital filters using residue feedback. In *IEEE Trans. on Acoustics, Speech and Signal Processing*, volume ASSP-43, October 1986.

[52] D. Williamson. *Digital Control and Implementation, Finite Wordlength Considerations*. Prentice-Hall International Editions, 1992.

[53] J. Wu, S. Chen, G. Li, R. Istepanian, and J. Chu. An improved closed-loop stability related measure for finite-precision digital controller realizations. *IEEE Trans. Automatic Control*, 46(7):1162–1166, 2001.

[54] C. Xiao. Improved L_2-sensitivity for state-space digital system. *IEEE Trans. Sig. Proc.*, 45(4):837–840, April 1997.

[55] W.-Y. Yan and J.B. Moore. On L^2-sensitivity minimization of linear state-space systems. *IEEE Trans. Circuits & Syst. I-Fundamental Theory & Appl.*, 39(8):641–648, August 1992.

[56] K. Zhou, J. Doyle, and K. Glover. *Robust and Optimal Control*. Prentice-Hall, 1996.

Input: u: array [1..4] of reals
Output: y: array [1..8] of reals
Data: T: array [1..16] of reals
begin
  // Intermediate variables
  $T_1 \leftarrow u(1) + u(3)$;
  $T_2 \leftarrow 0$;
  $T_3 \leftarrow u(1) + -u(3)$;
  $T_4 \leftarrow 0$;
  $T_5 \leftarrow u(2) + u(4)$;
  $T_6 \leftarrow 0$;
  $T_7 \leftarrow u(2) + -u(4)$;
  $T_8 \leftarrow 0$;
  $T_9 \leftarrow T_1$;
  $T_{10} \leftarrow T_2$;
  $T_{11} \leftarrow T_3$;
  $T_{12} \leftarrow T_4$;
  $T_{13} \leftarrow T_5$;
  $T_{14} \leftarrow T_6$;
  $T_{15} \leftarrow T_8$;
  $T_{16} \leftarrow -T_7$;
  // Outputs
  $y(1) \leftarrow T_9 + T_{13}$;
  $y(2) \leftarrow T_{10} + T_{14}$;
  $y(3) \leftarrow T_{11} + T_{15}$;
  $y(4) \leftarrow T_{12} + T_{16}$;
  $y(5) \leftarrow T_9 + -T_{13}$;
  $y(6) \leftarrow T_{10} + -T_{14}$;
  $y(7) \leftarrow T_{11} + -T_{15}$;
  $y(8) \leftarrow T_{12} + -T_{16}$;
end

Algorithm 9: FFT_4 without any simplification

Input: u: array [1..4] of reals
Output: y: array [1..8] of reals
Data: T: array [1..8] of reals
begin
    // Intermediate variables
    $T_1 \leftarrow u(1) + u(3)$;
    $T_2 \leftarrow u(1) + -u(3)$;
    $T_3 \leftarrow u(2) + u(4)$;
    $T_4 \leftarrow u(2) + -u(4)$;
    $T_5 \leftarrow T_1$;
    $T_6 \leftarrow T_2$;
    $T_7 \leftarrow T_3$;
    $T_8 \leftarrow -T_4$;
    // Outputs
    $y(1) \leftarrow T_5 + T_7$;
    $y(2) \leftarrow 0$;
    $y(3) \leftarrow T_6$;
    $y(4) \leftarrow T_8$;
    $y(5) \leftarrow T_5 + -T_7$;
    $y(6) \leftarrow 0$;
    $y(7) \leftarrow T_6$;
    $y(8) \leftarrow -T_8$;
end

Algorithm 10: FFT_4 with null terms removed

Input: u: array [1..4] of reals
Output: y: array [1..8] of reals
Data: T: array [1..5] of reals
begin
    // Intermediate variables
    $T_1 \leftarrow u(1) + u(3)$;
    $T_2 \leftarrow u(1) + -u(3)$;
    $T_3 \leftarrow u(2) + u(4)$;
    $T_4 \leftarrow u(2) + -u(4)$;
    // Outputs
    $y(1) \leftarrow T_1 + T_3$;
    $y(2) \leftarrow 0$;
    $y(3) \leftarrow T_2$;
    $y(4) \leftarrow -T_4$;
    $y(5) \leftarrow T_1 + -T_3$;
    $y(6) \leftarrow 0$;
    $y(7) \leftarrow T_2$;
    $y(8) \leftarrow T_4$;
end

Algorithm 11: FFT_4 with substitutions (1 term)