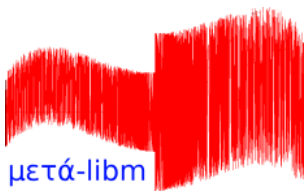
Algorithmique de l'implémentation fiable de filtres numériques

Quand le traitement de signal et l'arithmétique des ordinateurs se rencontrent

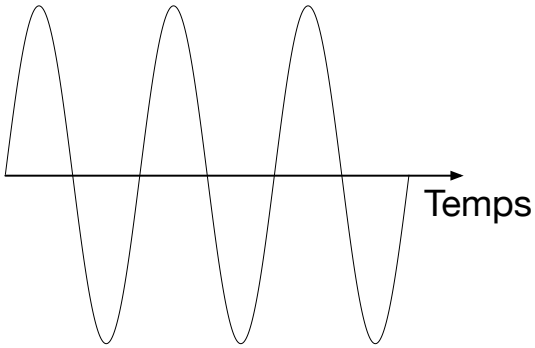
Anastasia Volkova

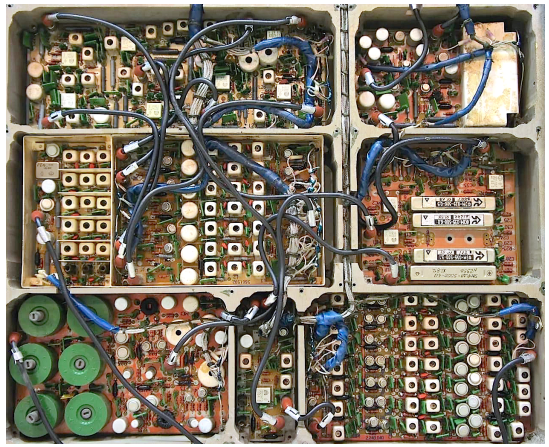
Directeur de thèse : Jean-Claude Bajard
Encadrants : Thibault Hilaire et Christoph Lauter



Des signaux partout

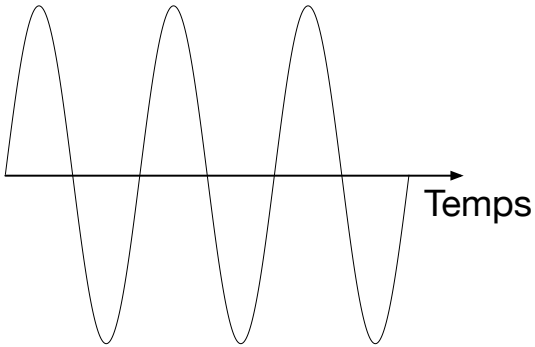
- Signaux analogiques : en temps continu

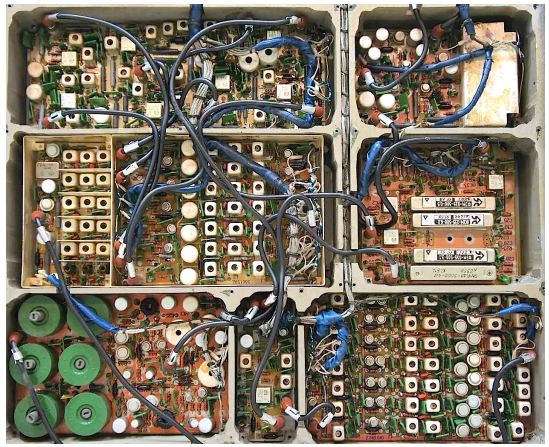




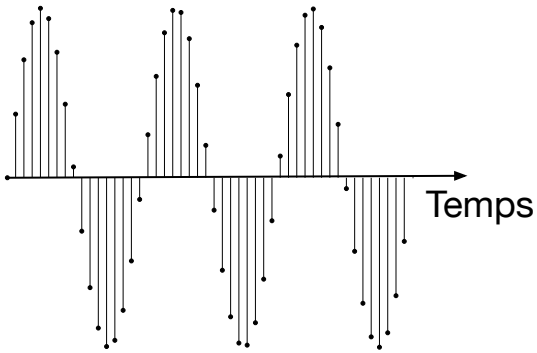
Des signaux partout

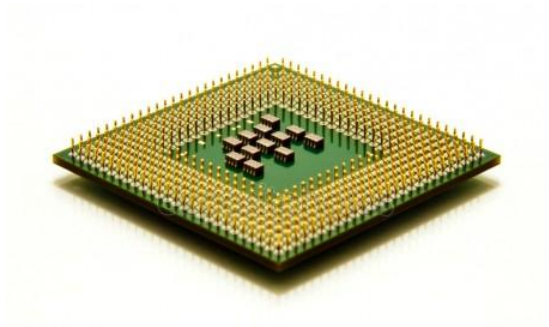
- Signaux analogiques : en temps continu





- Signaux numériques : en temps discret





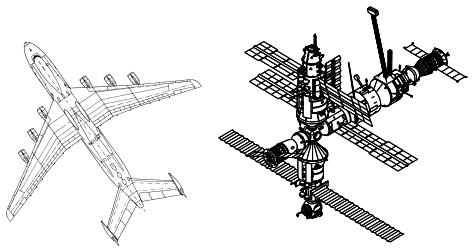
Applications : systèmes fiables

Filtres numériques :

Algorithmes de transformation de signaux numériques

- Pas besoin de garanties dans la majorité d'applications

- Une garantie est obligatoire pour d'autres applications.



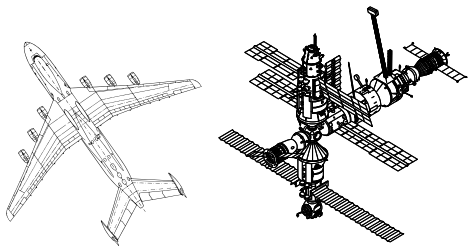
Applications : systèmes fiables

Filtres numériques :

Algorithmes de transformation de signaux numériques

- Pas besoin de garanties dans la majorité d'applications

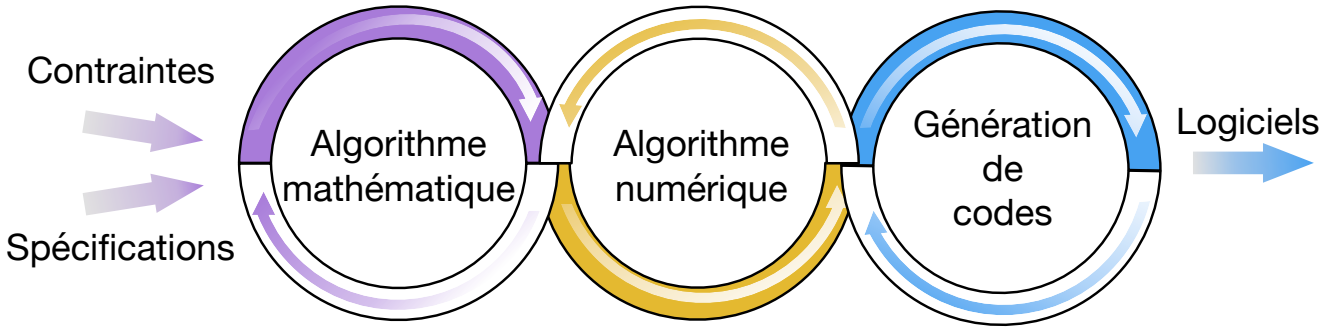
- Une garantie est obligatoire pour d'autres applications.



Nous nous intéressons aux garanties liées à l'implémentation d'algorithmes numériques, surtout dans les systèmes embarqués.

Automatisation

L'implémentation se fait en plusieurs étapes :



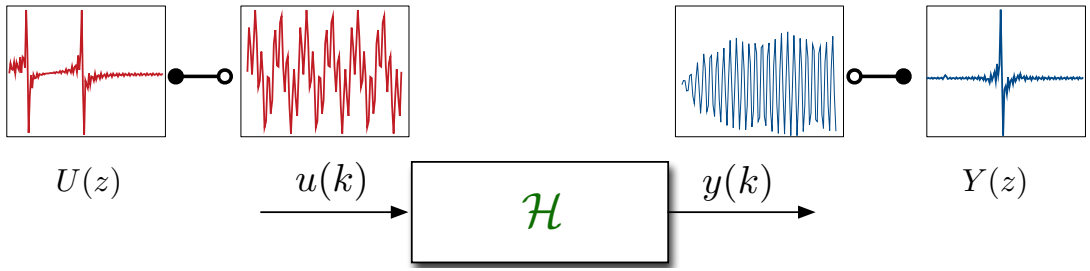
Plusieurs contraintes :

- performance
- surface
- précision
- consommation d'énergie
- mémoire
- etc.

Nous nous intéressons à un processus **automatique** d'implémentations **fiable**.

Filtres

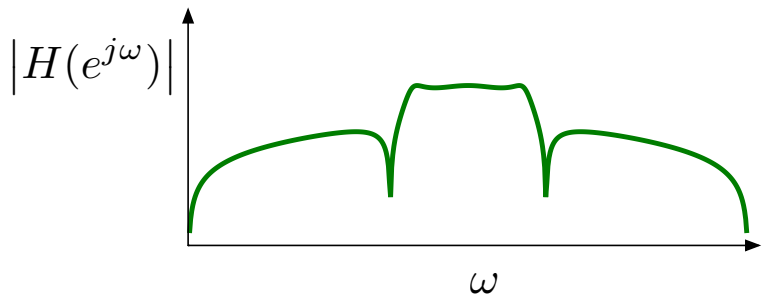
Domaine fréquentiel



Filtre linéaire $\mathcal{H}$: atténuation/amplification de propriétés **spectrales**

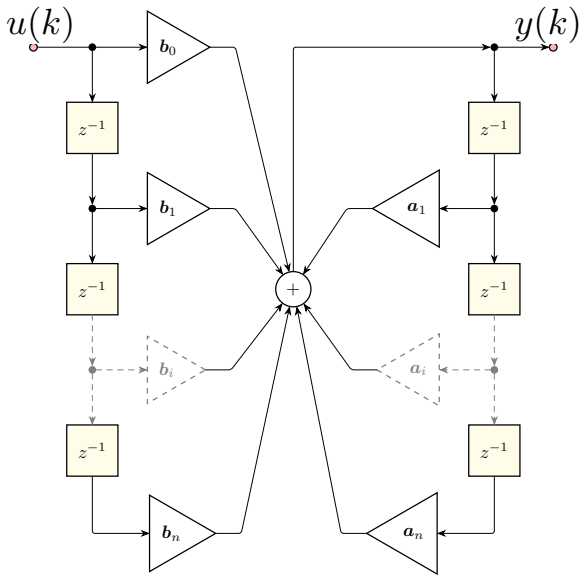
Fonction de transfert $H(z)$, $z \in \mathbb{C}$

$$H(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=1}^{n} a_i z^{-i}}$$



Filtres

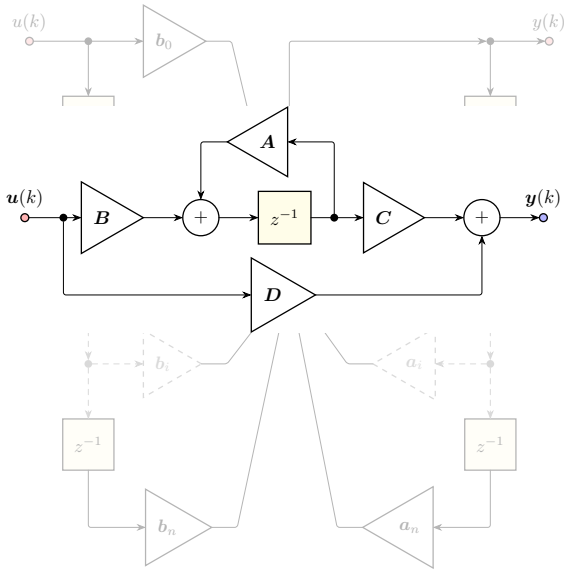
Domaine temporel



- $y(k) = \sum_{i=0}^{n} b_i\, u(k-i) - \sum_{i=1}^{n} a_i\, y(k-i)$

Filtres

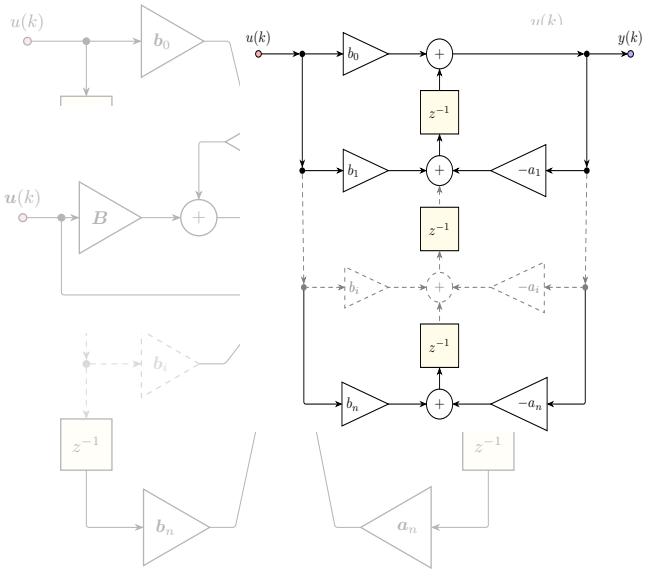
Domaine temporel



- $y(k) = \sum_{i=0}^{n} b_i\, u(k-i) - \sum_{i=1}^{n} a_i\, y(k-i)$
- $$\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Filtres

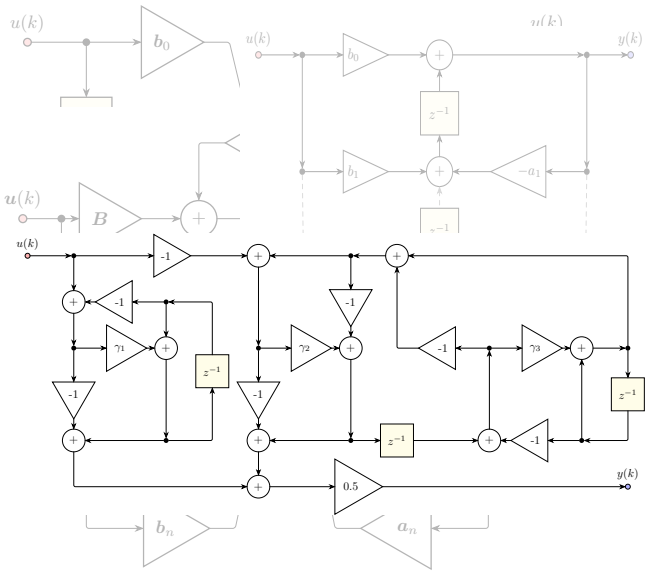
Domaine temporel



- $y(k) = \sum_{i=0}^{n} b_i\, u(k-i) - \sum_{i=1}^{n} a_i\, y(k-i)$
- $\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$
- ...

Filtres

Domaine temporel



- $y(k) = \sum_{i=0}^{n} b_i\, u(k-i) - \sum_{i=1}^{n} a_i\, y(k-i)$
- $\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$
- ...

Algorithme typique : entrée $u(k)$, état interne $\boldsymbol{x}(k)$, sortie $y(k)$

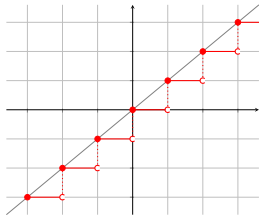
Mathématiquement, les différents algorithmes calculent la même sortie.

Algorithmes numériques

Le choix de l'arithmétique et de ses paramètres détermine la qualité numérique de l'implémentation de l'algorithme.

Pourquoi ?

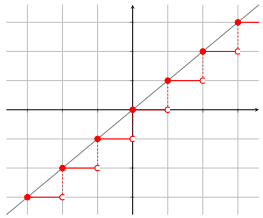
- les signaux sont discrets en valeur
- les instructions pour l'évaluation peuvent induire des erreurs
- la propagation et la compensation des erreurs dépendent des instructions



Algorithmes numériques

Le choix de l'arithmétique et de ses paramètres détermine la qualité numérique de l'implémentation de l'algorithme.

Pourquoi ?



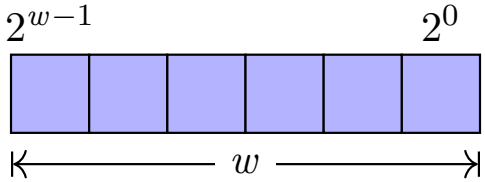
- les signaux sont discrets en valeur
- les instructions pour l'évaluation peuvent induire des erreurs
- la propagation et la compensation des erreurs dépendent des instructions

De plus, le choix de l'arithmétique et de ses paramètres influencent:

- la vitesse de calculs
- la surface
- ...

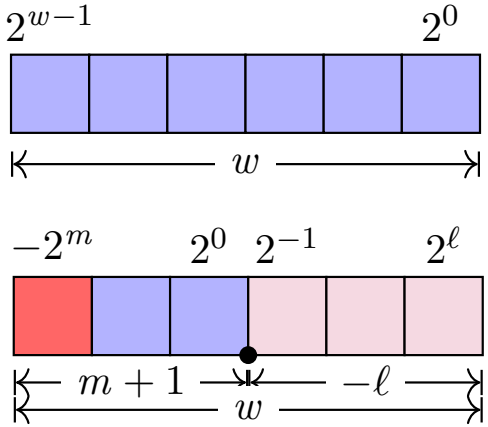
Arithmétiques

- Arithmétique entière :
  $y = Y$



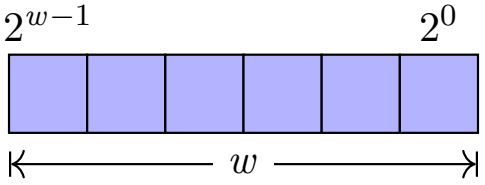
Arithmétiques

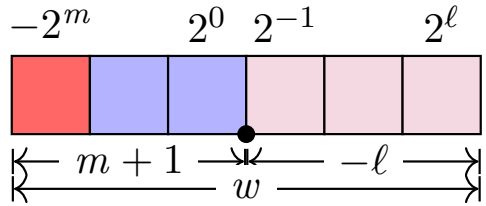
- Arithmétique entière :
  $y = Y$
- Arithmétique virgule fixe :
  $y = Y \cdot 2^{\ell}$
  où ℓ est un facteur *implicite*

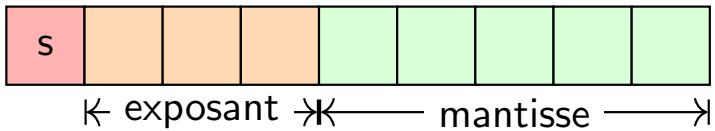


Arithmétiques

- Arithmétique entière :
  $y = Y$
- Arithmétique virgule fixe :
  $y = Y \cdot 2^{\ell}$
  où ℓ est un facteur *implicite*
- Arithmétique virgule flottante :
  $y = (-1)^s \cdot Y \cdot 2^e$
  où e est un facteur *explicite*

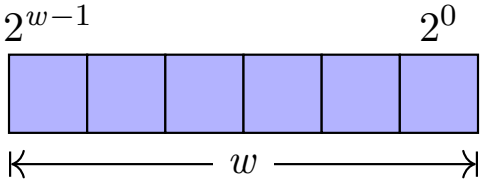


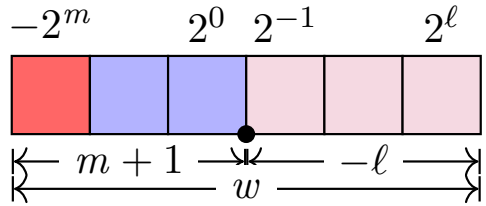


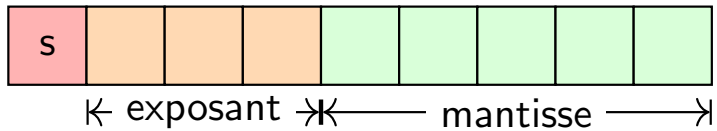


Arithmétiques

- Arithmétique entière :
  $y = Y$
- Arithmétique virgule fixe :
  $y = Y \cdot 2^{\ell}$
  où ℓ est un facteur *implicite*
- Arithmétique virgule flottante :
  $y = (-1)^s \cdot Y \cdot 2^e$
  où e est un facteur *explicite*
- Arithmétique d'intervalles :
  $[\underline{y}, \overline{y}] = \left\{ y \in \mathbb{R} \mid \underline{y} \leq y \leq \overline{y} \right\}$

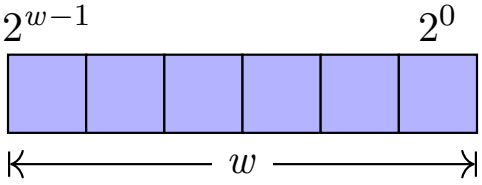


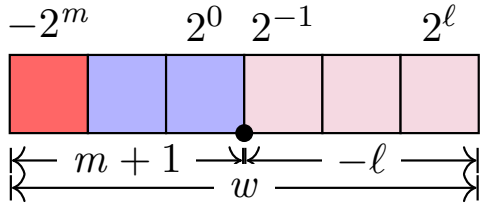


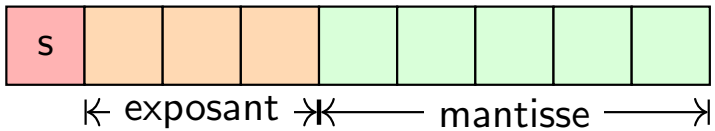


Arithmétiques

- Arithmétique entière :
  $y = Y$
- Arithmétique virgule fixe :
  $y = Y \cdot 2^{\ell}$
  où ℓ est un facteur *implicite*
- Arithmétique virgule flottante :
  $y = (-1)^s \cdot Y \cdot 2^e$
  où e est un facteur *explicite*
- Arithmétique d'intervalles :
  $[\underline{y}, \overline{y}] = \left\{ y \in \mathbb{R} \mid \underline{y} \leq y \leq \overline{y} \right\}$
- Arithmétique multi-précision : la taille de la mantisse varie dynamiquement







Arithmétiques

- Arithmétique virgule fixe :
  $y = Y \cdot 2^{\ell}$
  où ℓ est un facteur *implicite*

  Pour l'implémentation

- Arithmétique virgule flottante :
  $y = (-1)^s \cdot Y \cdot 2^e$
  où e est un facteur *explicite*

  Pour l'analyse d'erreur

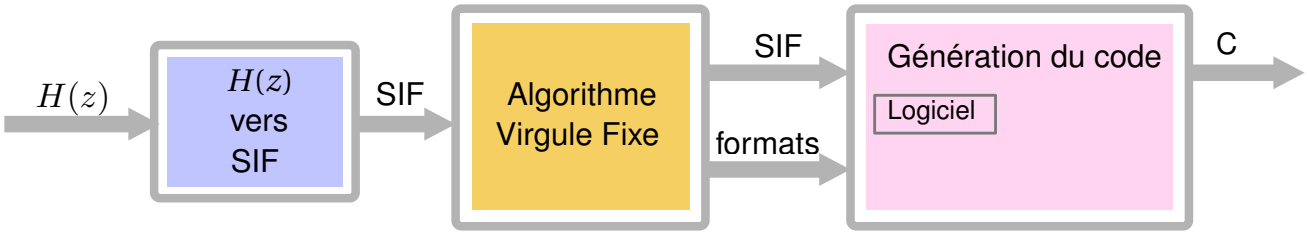
- Arithmétique d'intervalles :
  $[\underline{y}, \overline{y}] = \{y \in \mathbb{R} \mid \underline{y} \leq y \leq \overline{y}\}$

  Pour l'analyse d'erreur

- Arithmétique multi-précision : la taille de la mantisse varie dynamiquement

  Pour l'analyse d'erreur

Algorithmique de l'implémentation fiable de filtres

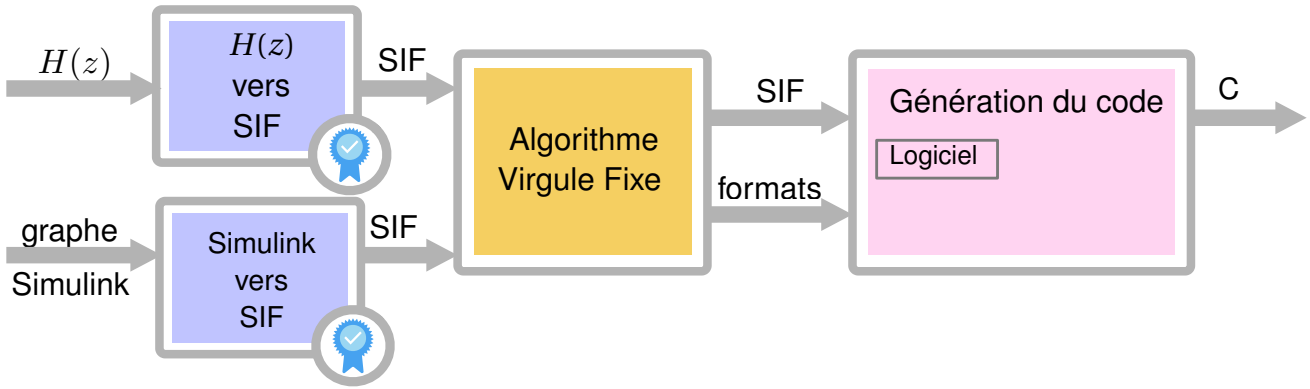


Avant cette thèse

- Une unification de la représentation des filtres (SIF)
  - Plusieurs algorithmes sont déjà décrits manuellement en SIF.

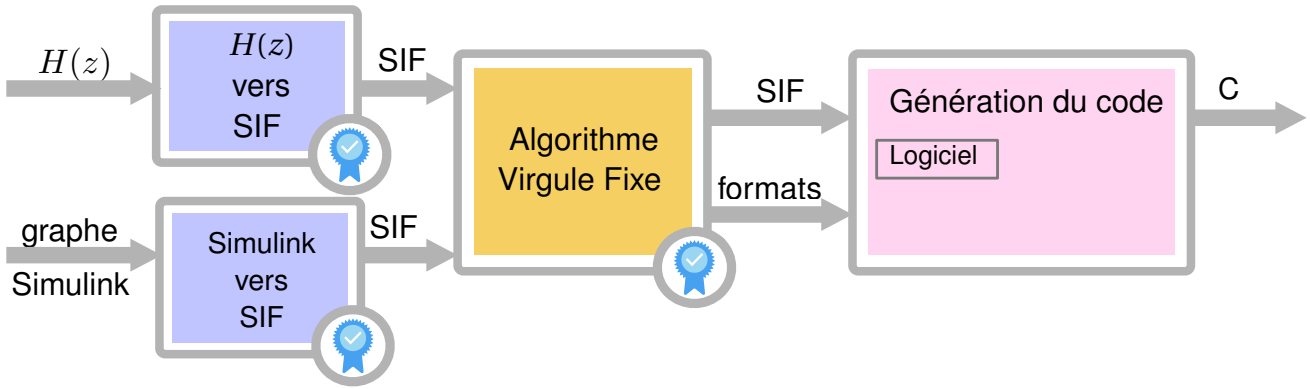
- Implémentation virgule fixe unifiée, mais pas fiable
- Génération de C pour le logiciel

Algorithmique de l'implémentation fiable de filtres



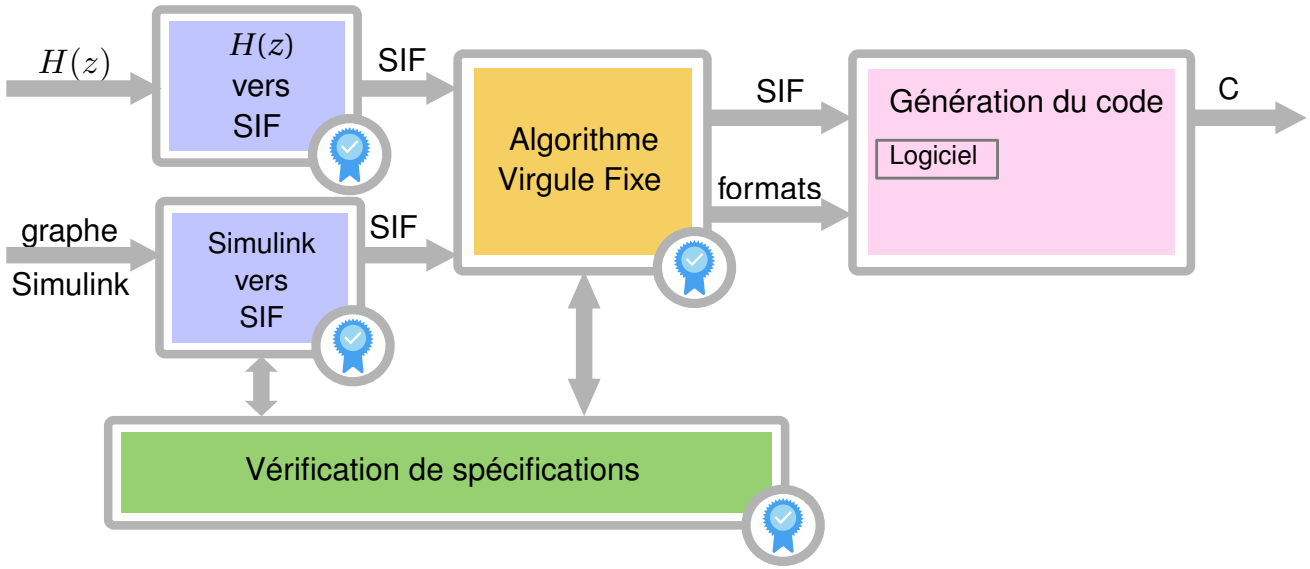
- Une unification de la représentation des filtres (SIF)
  - Plusieurs algorithmes sont déjà décrits manuellement en SIF.
  - **Formats d'entrée supplémentaires**
- Implémentation virgule fixe unifiée, mais pas fiable
- Génération de C pour le logiciel

Algorithmique de l'implémentation fiable de filtres



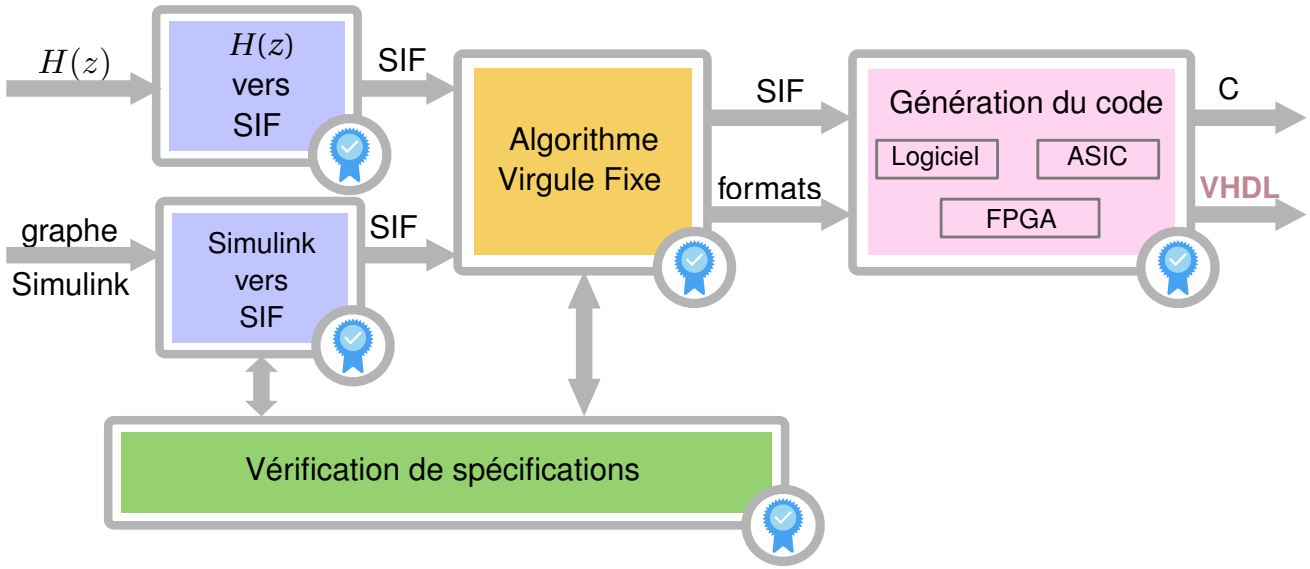
- Une unification de la représentation des filtres (SIF)
  - Plusieurs algorithmes sont déjà décrits manuellement en SIF.
  - **Formats d'entrée supplémentaires**
- Implémentation virgule fixe unifiée, **fiable**
- Génération de C pour le logiciel

Algorithmique de l'implémentation fiable de filtres



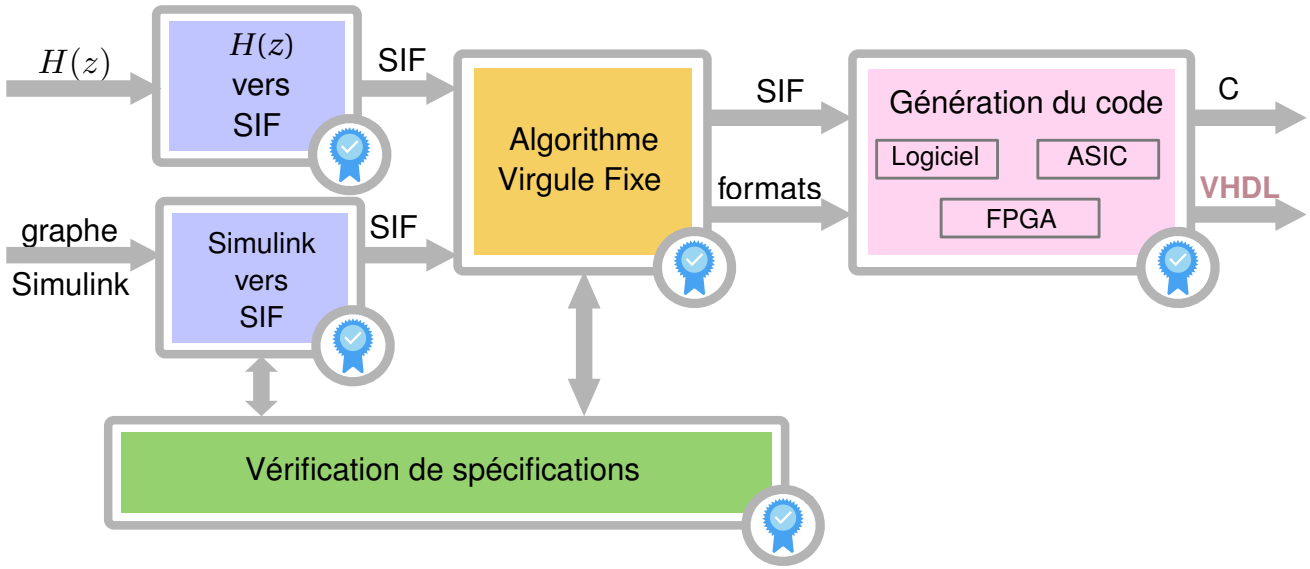
- Une unification de la représentation des filtres (SIF)
  - Plusieurs algorithmes sont déjà décrits manuellement en SIF.
  - **Formats d'entrée supplémentaires**
- Implémentation virgule fixe unifiée, **fiable** et **vérifiée**
- Génération de C pour le logiciel

Algorithmique de l'implémentation fiable de filtres



- Une unification de la représentation des filtres (SIF)
  - Plusieurs algorithmes sont déjà décrits manuellement en SIF.
  - **Formats d'entrée supplémentaires**
- Implémentation virgule fixe unifiée, **fiable** et **vérifiée**
- Génération de C pour le logiciel **et de VHDL pour des FPGA**

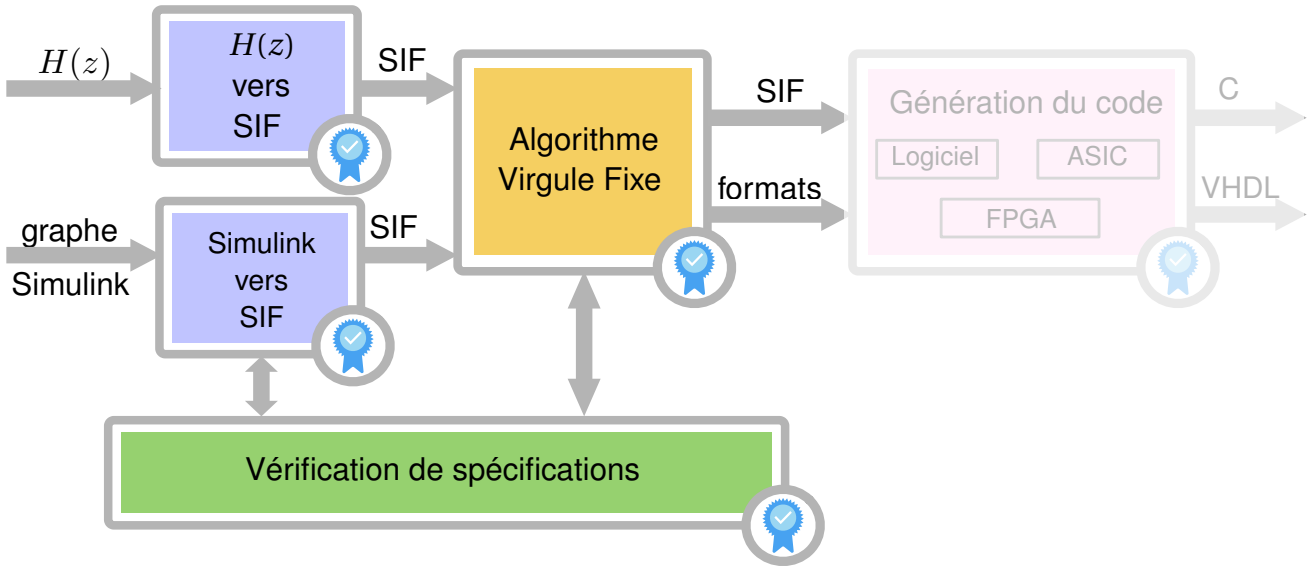
Algorithmique de l'implémentation fiable de filtres



- Une unification de la représentation des filtres (SIF)
  - Plusieurs algorithmes sont déjà décrits manuellement en SIF.
  - **Formats d'entrée supplémentaires**
- Implémentation virgule fixe unifiée, **fiable** et **vérifiée**
- Génération de C pour le logiciel **et de VHDL pour des FPGA**

Production scientifique : 9 publications

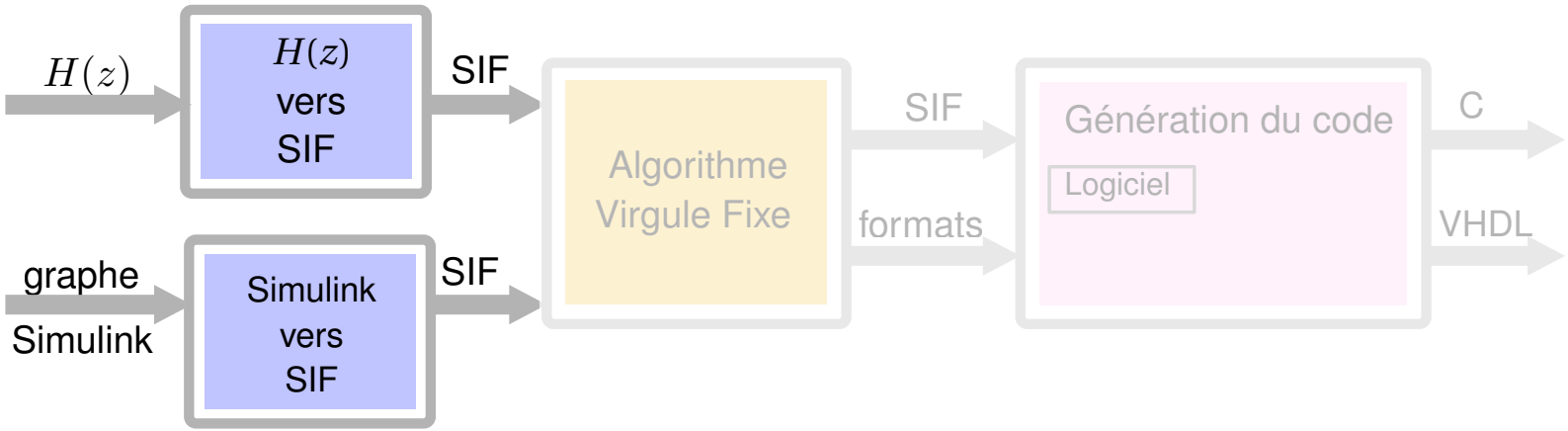
Algorithmique de l'implémentation fiable de filtres



- Une unification de la représentation des filtres (SIF)
  - Plusieurs algorithmes sont déjà décrits manuellement en SIF.
  - **Formats d'entrée supplémentaires**
- Implémentation virgule fixe unifiée, **fiable** et **vérifiée**
- Génération de C pour le logiciel **et de VHDL pour des FPGA**

Production scientifique : 9 publications

Formats d'entrée supplémentaires



SIF : *Forme Implicite Spécialisée*

- Forme matricielle basée sur un système d'équations linéaires
  - ☞ Différente de celle par graphes
- L'ordre de calcul est exprimé dans les équations.
- Tous les systèmes linéaires sont exprimables en SIF.

SIF : *Forme Implicite Spécialisée*

- Forme matricielle basée sur un système d'équations linéaires
  - ☞ Différente de celle par graphes
- L'ordre de calcul est exprimé dans les équations.
- Tous les systèmes linéaires sont exprimables en SIF.

Idée de base :

$$y = m_2 m_1 u$$

SIF : *Forme Implicite Spécialisée*

- Forme matricielle basée sur un système d'équations linéaires
  - ☞ Différente de celle par graphes
- L'ordre de calcul est exprimé dans les équations.
- Tous les systèmes linéaires sont exprimables en SIF.

Idée de base :

$$y \longleftarrow m_2(m_1\, u)$$

1: $t \longleftarrow m_1\, u$

2: $y \longleftarrow m_2\, t$

SIF : *Forme Implicite Spécialisée*

- Forme matricielle basée sur un système d'équations linéaires
  - ☞ Différente de celle par graphes
- L'ordre de calcul est exprimé dans les équations.
- Tous les systèmes linéaires sont exprimables en SIF.

Idée de base :

$y \longleftarrow m_2(m_1 u)$

1: $t \longleftarrow m_1 u$

2: $y \longleftarrow m_2 t$

$$\begin{pmatrix} 1 & 0 \\ -m_2 & 1 \end{pmatrix} \begin{pmatrix} t \\ y \end{pmatrix} = \begin{pmatrix} m_1 \\ 0 \end{pmatrix} u$$

SIF : *Forme Implicite Spécialisée*

- Forme matricielle basée sur un système d'équations linéaires
  - ☞ Différente de celle par graphes
- L'ordre de calcul est exprimé dans les équations.
- Tous les systèmes linéaires sont exprimables en SIF.

Idée de base :

$y \longleftarrow m_2(m_1 u)$

1: $t \longleftarrow m_1 u$

2: $y \longleftarrow m_2 t$

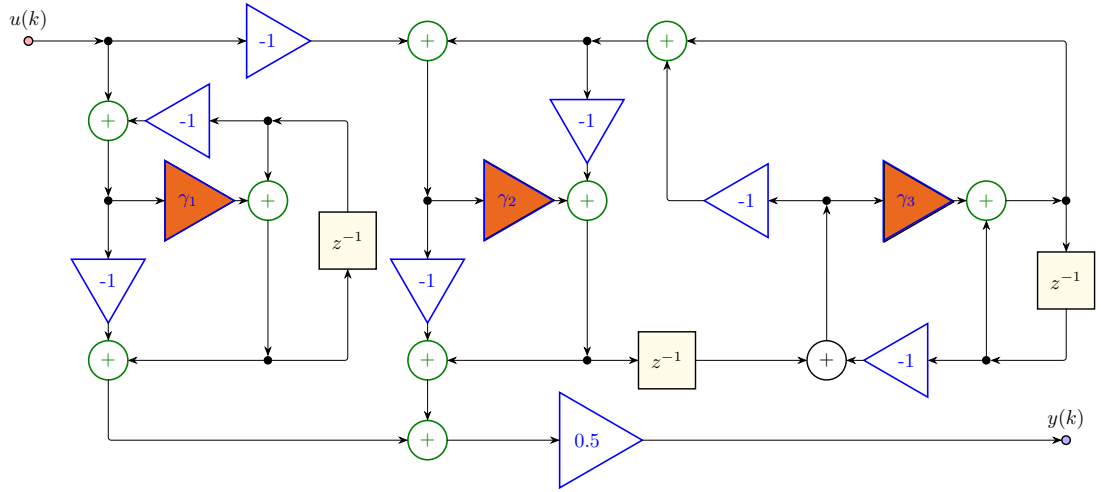
$$\begin{pmatrix} 1 & 0 \\ -m_2 & 1 \end{pmatrix} \begin{pmatrix} t \\ y \end{pmatrix} = \begin{pmatrix} m_1 \\ 0 \end{pmatrix} u$$

Contributions :

- Conversion des Lattice Wave Digital Filters vers SIF
- Conversion automatique de graphes Simulink vers SIF

Conversion à partir de Simulink

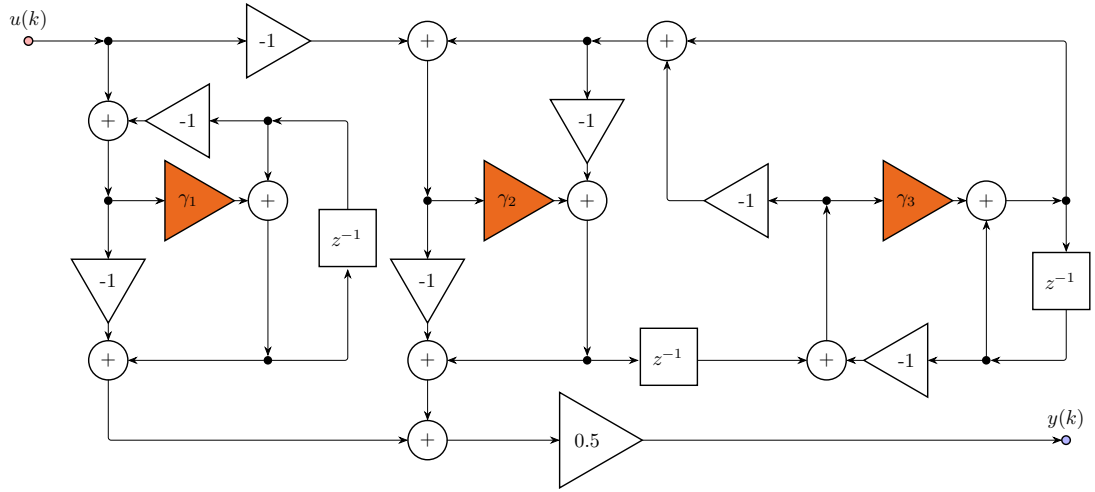
Simulink :



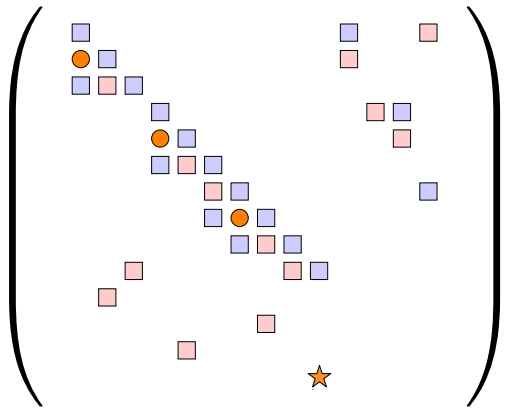
$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$

Conversion à partir de Simulink

Simulink :



SIF :



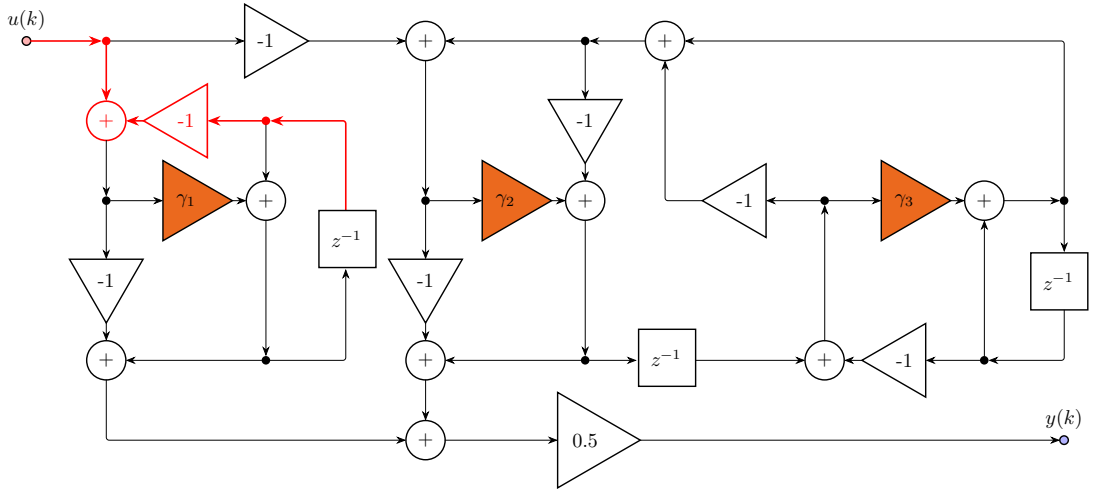
$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$

Idée clé de notre algorithme de conversion :

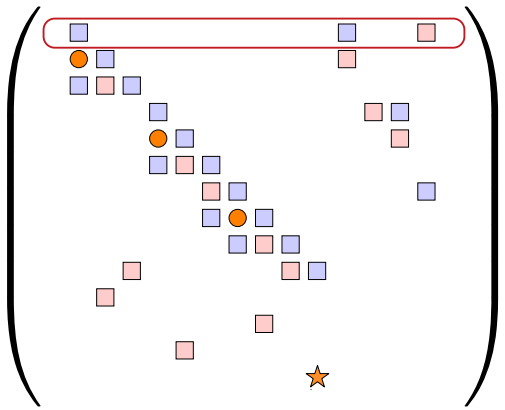
- Identification des entrées, sorties, états, variables temporaires
- Construction des équations
- Tri topologique
- Recopie exacte des coefficients

Conversion à partir de Simulink

Simulink :



SIF :



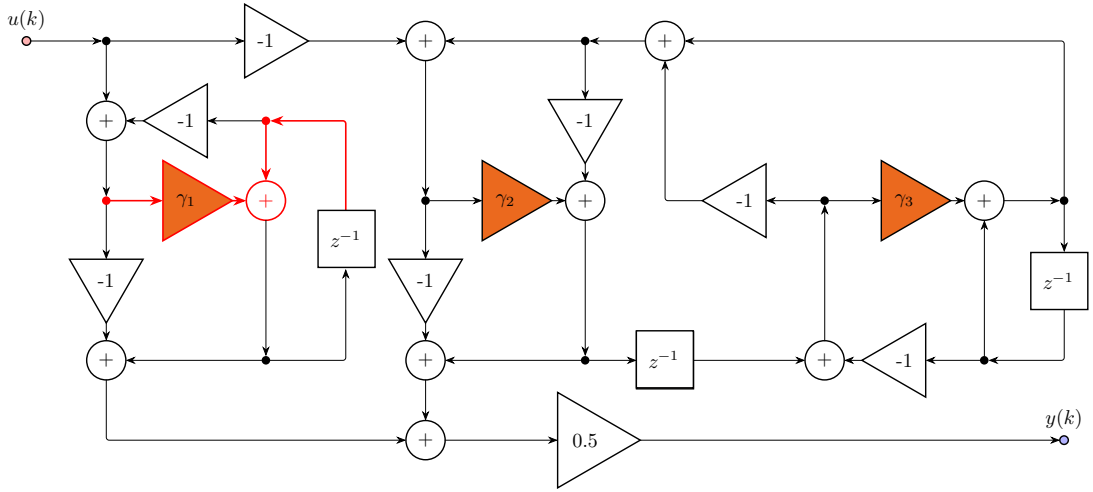
$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$

Idée clé de notre algorithme de conversion :

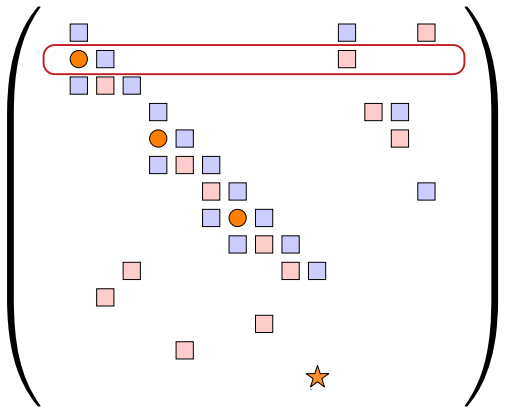
- Identification des entrées, sorties, états, variables temporaires
- Construction des équations
- Tri topologique
- Recopie exacte des coefficients

Conversion à partir de Simulink

Simulink :



SIF :

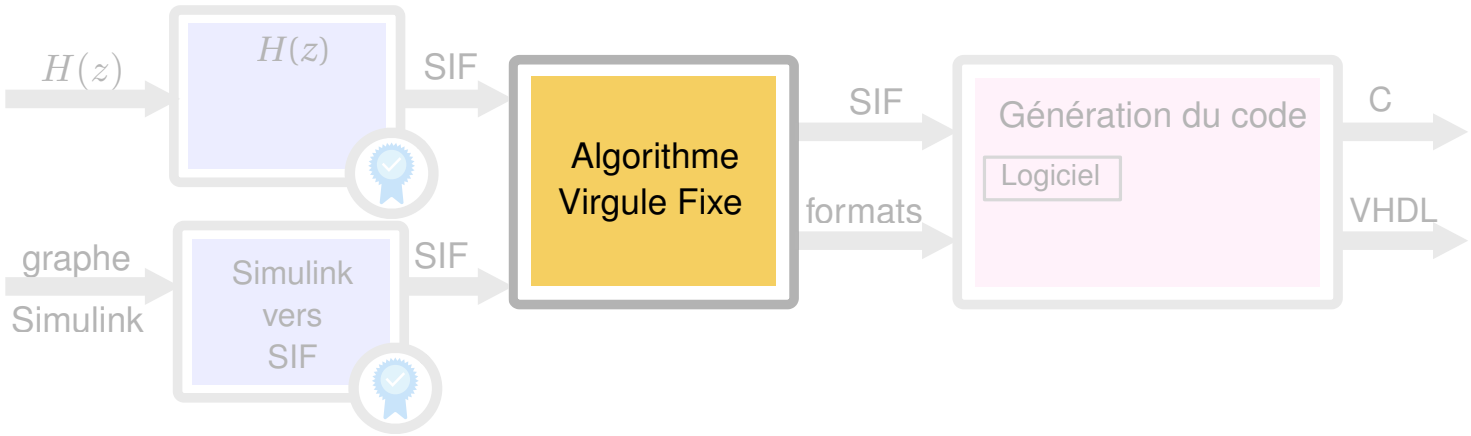


$$\gamma_1 = 89 \cdot 2^{-8}, \gamma_2 = 43 \cdot 2^{-7}, \gamma_3 = 11 \cdot 2^{-7}$$

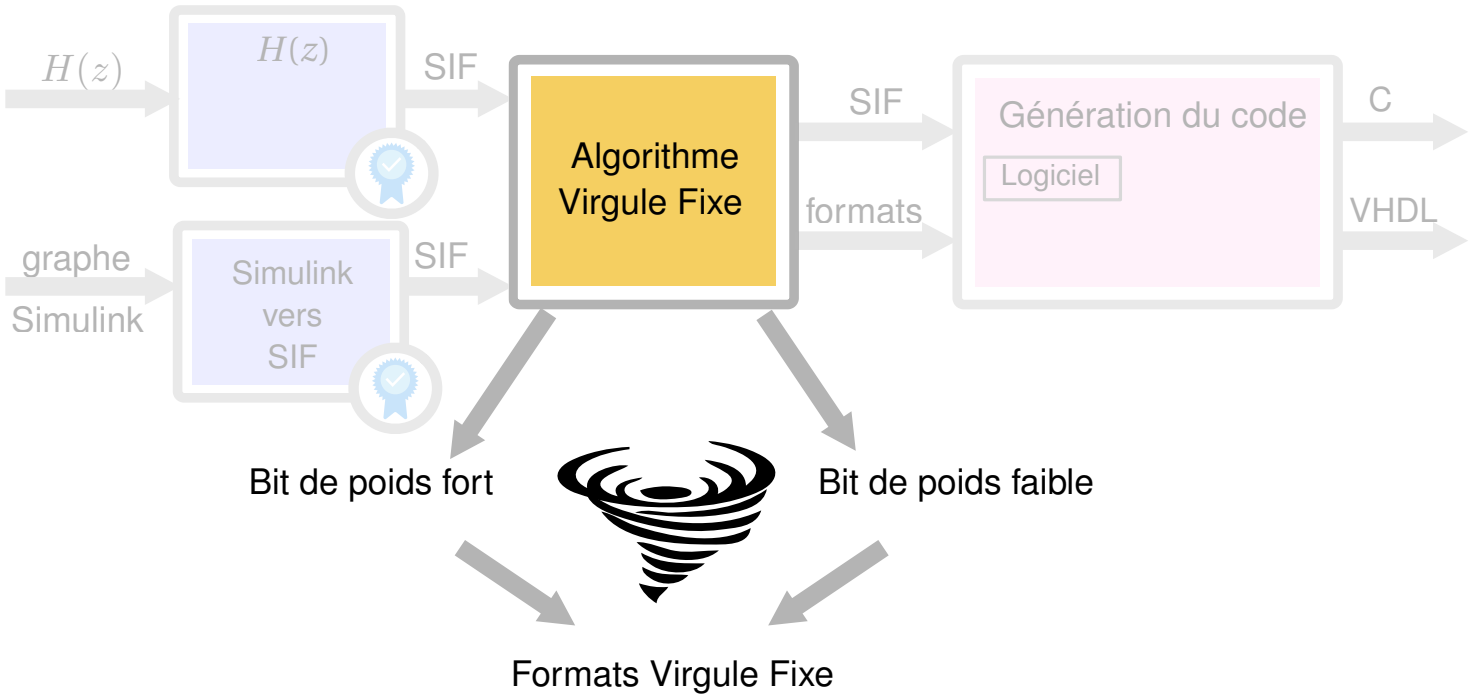
Idée clé de notre algorithme de conversion :

- Identification des entrées, sorties, états, variables temporaires
- Construction des équations
- Tri topologique
- Recopie exacte des coefficients

Algorithmes numériques fiables



Algorithmes numériques fiables

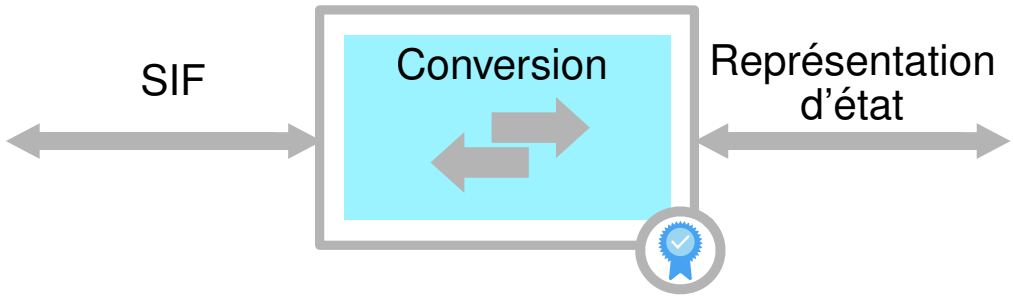


Représentation d'état

Représentation d'état d'un filtre linéaire $\mathcal{H}$:

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

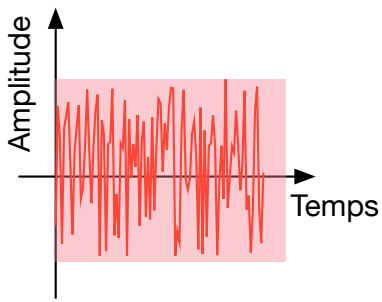
Les conversions entre SIF et représentation d'état sont exactes.

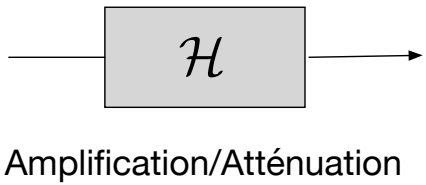


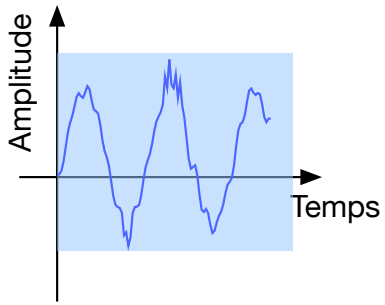
Dynamique des variables

Worst-Case Peak Gain

Entrée $u(k)$ | Filtre stable | Sortie $y(k)$







$\forall k, |u(k)| \leq \bar{u}$

$\forall k, |y(k)| \leq \langle\langle \mathcal{H} \rangle\rangle \bar{u}$

Worst-Case Peak Gain : $\langle\langle \mathcal{H} \rangle\rangle = \|h\|_1 = |d| + \sum_{k=0}^{\infty} \left| \boldsymbol{c} \boldsymbol{A}^k \boldsymbol{b} \right|$

Problème du choix des formats

- **Contraintes** : largeur w_y fixée pour la variable de sortie y
- **But** : pas de dépassements, borne rigoureuse sur les erreurs
- **Bonus** : minimiser les erreurs

Problème du choix des formats

- **Contraintes** : largeur w_y fixée pour la variable de sortie y
- **But** : pas de dépassements, borne rigoureuse sur les erreurs
- **Bonus** : minimiser les erreurs

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Problème : trouver le plus petit m_y tel que pour tout k

$$|y(k)| \leq 2^{m_y}\left(1 - 2^{-w_y+1}\right)$$

Problème du choix des formats

- **Contraintes** : largeur w_y fixée pour la variable de sortie y
- **But** : pas de dépassements, borne rigoureuse sur les erreurs
- **Bonus** : minimiser les erreurs

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Problème : trouver le plus petit m_y tel que pour tout k

$$\langle\langle\mathcal{H}\rangle\rangle\bar{u} = |y(k)| \leq 2^{m_y}\left(1 - 2^{-w_y+1}\right)$$

Solution mathématique :

$$m_y = \left\lceil \log_2\left(\langle\langle\mathcal{H}\rangle\rangle\,\bar{u}\right) - \log_2\left(1 - 2^{1-w_y}\right)\right\rceil$$

Problème du choix des formats

- **Contraintes** : largeur w_y fixée pour la variable de sortie y
- **But** : pas de dépassements, borne rigoureuse sur les erreurs
- **Bonus** : minimiser les erreurs

$$\mathcal{H}\begin{cases} \boldsymbol{x}(k+1) &= \boldsymbol{A}\boldsymbol{x}(k) + \boldsymbol{b}u(k) \\ y(k) &= \boldsymbol{c}\boldsymbol{x}(k) + du(k) \end{cases}$$

Problème : trouver le plus petit m_y tel que pour tout k

$$\langle\langle\mathcal{H}\rangle\rangle\bar{u} = |y(k)| \leq 2^{m_y}\left(1 - 2^{-w_y+1}\right)$$

Solution mathématique :

$$m_y = \left\lceil \log_2\left(\langle\langle\mathcal{H}\rangle\rangle\,\bar{u}\right) - \log_2\left(1 - 2^{1-w_y}\right)\right\rceil$$

Solution pratique : contrôler la précision du WCPG pour que

$$0 \leq \widehat{m}_y - m_y \leq 1$$

Rebouclage des erreurs de calcul

Filtre exact $\mathcal{H}$ est :

$$\mathcal{H} \begin{cases} \boldsymbol{x}\ (k+1) &= & \boldsymbol{A}\boldsymbol{x}\ (k) + \boldsymbol{b}\,u(k) \\ y\ (k) &= & \boldsymbol{c}\boldsymbol{x}\ (k) + du(k) \end{cases}$$

Rebouclage des erreurs de calcul

Filtre implémenté $\mathcal{H}^{\Diamond}$ est :

$$\mathcal{H}^{\Diamond} \left\{ \begin{array}{rcl} \boldsymbol{x}^{\Diamond}(k+1) & = & \Diamond_{m_x}(\boldsymbol{A}\boldsymbol{x}^{\Diamond}(k) + \boldsymbol{b}\,u(k)) \\ y^{\Diamond}(k) & = & \Diamond_{m_y}(\boldsymbol{c}\boldsymbol{x}^{\Diamond}(k) + du(k)) \end{array} \right.$$

ou $\Diamond_m$ est un opérateur qui garantit l'arrondi fidèle :

$$|\Diamond_m(x) - x| \leq 2^{m-w+1}$$

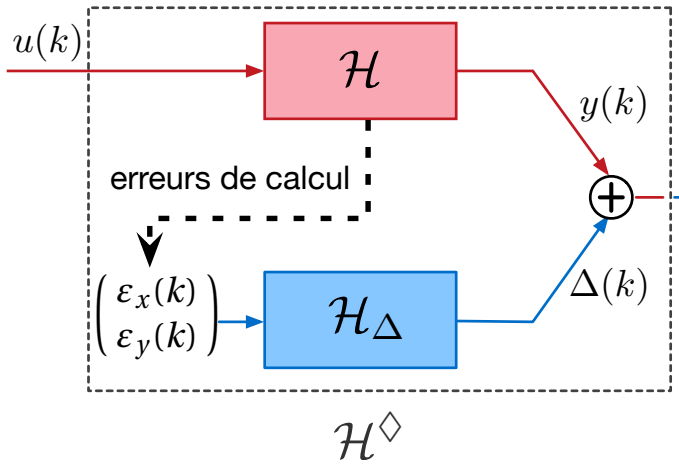
Rebouclage des erreurs de calcul

Filtre implémenté $\mathcal{H}^\Diamond$ est :

$$\mathcal{H}^\Diamond \begin{cases} \boldsymbol{x}^\Diamond(k+1) &= \boldsymbol{A}\boldsymbol{x}^\Diamond(k) + \boldsymbol{b}u(k) + \boldsymbol{\varepsilon}_x(k) \\ y^\Diamond(k) &= \boldsymbol{c}\boldsymbol{x}^\Diamond(k) + du(k) + \varepsilon_y(k) \end{cases}$$

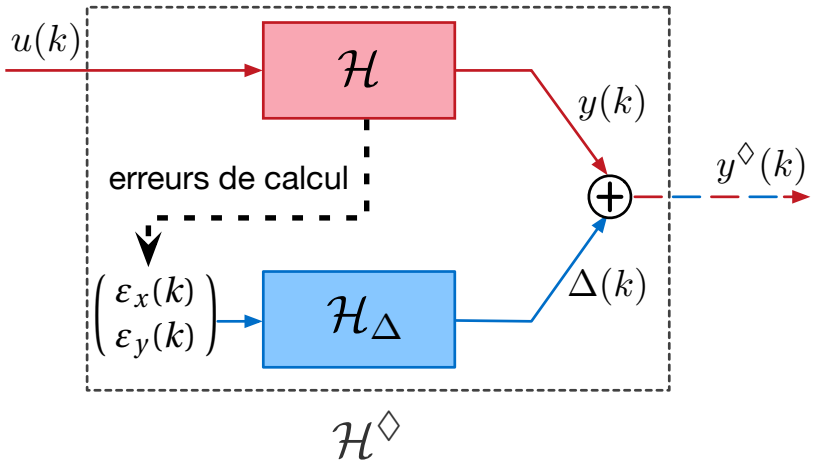
avec

$$|\boldsymbol{\varepsilon}_x(k)| \leq 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1} \quad \text{et} \quad |\varepsilon_y(k)| \leq 2^{m_y - w_y + 1}$$



$$|\Delta(k)| \leq \langle\langle \mathcal{H}_\Delta \rangle\rangle \begin{pmatrix} 2^{\boldsymbol{m}_x - \boldsymbol{w}_x + 1} \\ 2^{m_y - w_y + 1} \end{pmatrix}$$

Algorithme fiable



1. Estimation initiale de la position du bit de poids fort m_y pour le filtre exact $\mathcal{H}$
2. Prise en compte des erreurs induites par le format choisi à l'aide du filtre $\mathcal{H}_\Delta$, puis calcul du bit de poids fort $m_y^{\Diamond}$
3. Si $m_y^{\Diamond} = m_y$, renvoyer $m_y^{\Diamond}$
   sinon $m_y \leftarrow m_y + 1$ et retourner à l'étape 2

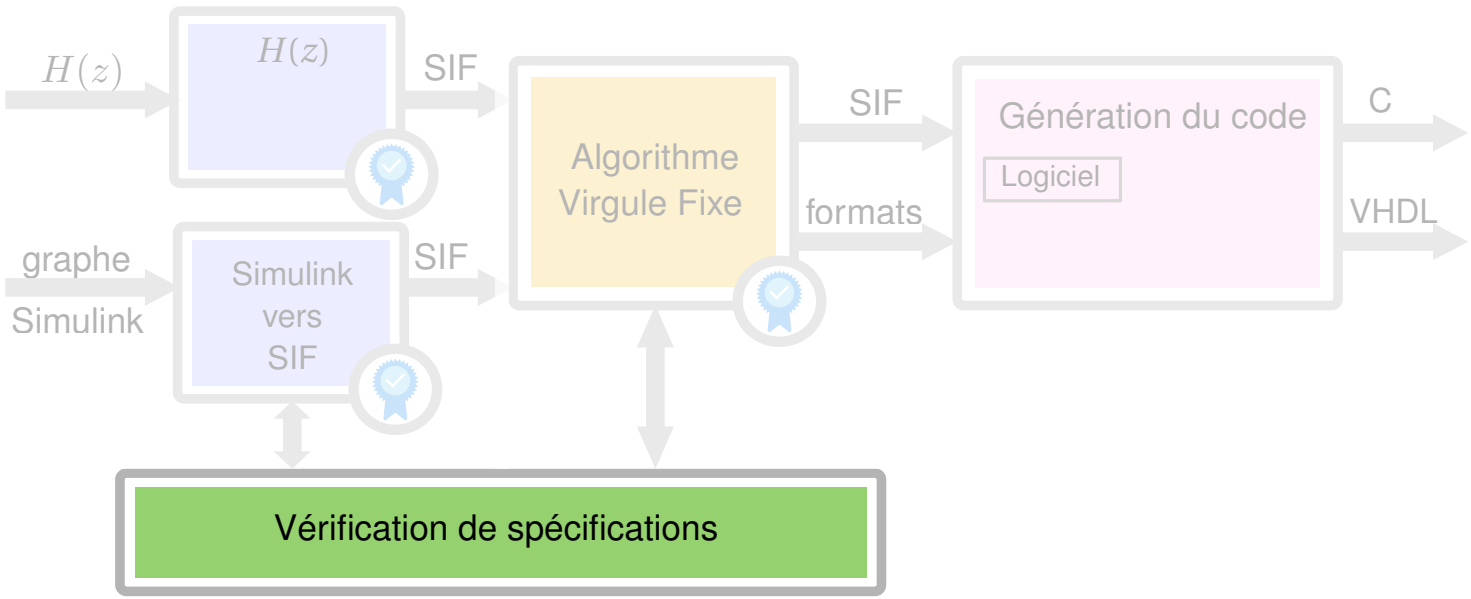
Exemple numérique

Largeurs : 16 bits
Entrées : $\forall k,\ -1 \leq u(k) < 1$
Formats virgule fixe :

	u	x_1	x_2	x_3	y
bit de poids fort	0	5	5	5	1
bit de poids faible	-15	-10	-10	-10	-14

Borne sur l'erreur : $|\Delta(k)| \leq 2^{-9}$
Temps : 0.012 s

Retour vers le générateur

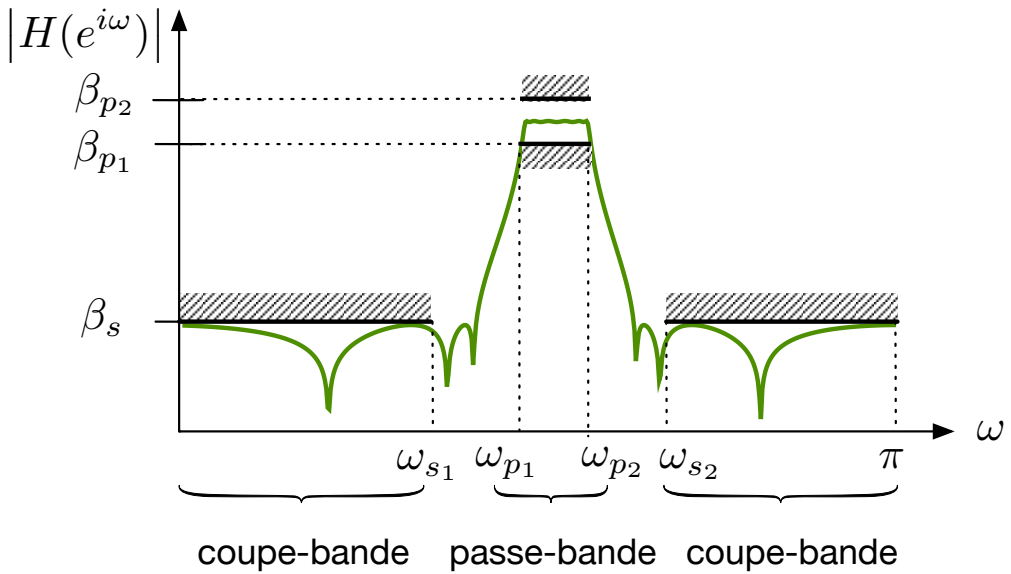


Spécifications fréquentielles

Réponse fréquentielle :

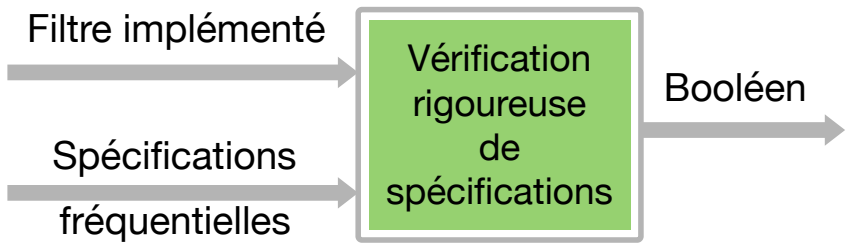
$$H\left(e^{j\omega}\right) = \underbrace{\left|H\left(e^{j\omega}\right)\right|}_{\text{module}} e^{\overbrace{\measuredangle H\left(e^{j\omega}\right)}^{\text{phase}}}$$

Spécifications :

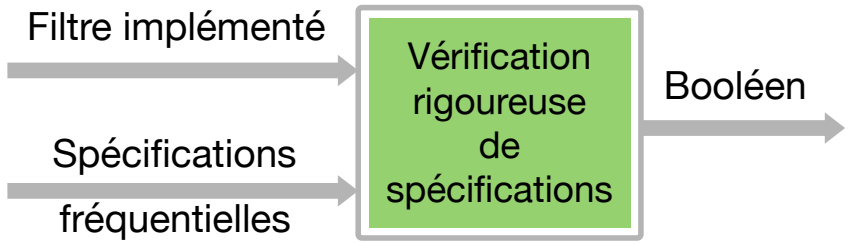


$$\underline{\beta} \leq \left|H(e^{j\omega})\right| \leq \overline{\beta}, \quad \forall \omega \in [\omega_1, \omega_2] \subseteq [0, \pi]$$

Vérification d'une implémentation



Vérification d'une implémentation



Approches existantes :

- par simulations
- réponse fréquentielle approchée

Notre approche fiable :

- pas de simulation, seulement preuves
- arithmétiques rationnelle et d'intervalles

But :

- Garantie d'implémentation
- Calcul rapide de cette garantie

Vérification d'une fonction de transfert

Nous avons besoin de montrer que $\forall z = e^{j\omega}, \omega \in \Omega \subseteq [0, \pi]$

$$\underline{\beta} \leq |H(z)| \leq \overline{\beta}$$

Vérification d'une fonction de transfert

Nous avons besoin de montrer que $\forall z = e^{j\omega}, \omega \in \Omega \subseteq [0, \pi]$

$$\underline{\beta}^2 \leq |H(z)|^2 \leq \overline{\beta}^2$$

Vérification d'une fonction de transfert

Nous avons besoin de montrer que $\forall z = e^{j\omega}, \omega \in \Omega \subseteq [0, \pi]$

$$\underline{\beta}^2 \leq |H(z)|^2 \leq \overline{\beta}^2$$

où

$$|H(z)|^2 = \frac{|b(z)|^2}{|a(z)|^2} = \frac{b(z)\overline{b(\overline{z})}}{a(z)\overline{a(\overline{z})}} = \frac{b(z)b(\frac{1}{z})}{a(z)a(\frac{1}{z})} =: \frac{v(z)}{w(z)},$$

$v(z)$ et $w(z)$ sont des polynômes à coefficients réels.

Simplification du problème

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$

Simplification du problème

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$

Pas besoin de gérer les complexes

Changement de variable : $t = \tan \frac{\omega}{2}$

$$z = e^{j\omega} = \frac{1 - t^2}{1 + t^2} + j \frac{2t}{1 + t^2}$$

Simplification du problème

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{v(\frac{1-t^2}{1+t^2} + j\frac{2t}{1+t^2})}{w(\frac{1-t^2}{1+t^2} + j\frac{2t}{1+t^2})} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan\frac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$

Simplification du problème

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \underbrace{\frac{r(t) + j\text{ж}(t)}{s(t) + j\text{щ}(t)}}_{\in \mathbb{R} \text{ car } |H(z)|^2} \leq \overline{\beta}^2$$

Polynômes $r, s, \text{ж}, \text{щ} \in \mathbb{R}[t]$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \frac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$

Simplification du problème

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(t)}{s(t)} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \tfrac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$

Maintenant, on ne travaille qu'avec des réels.

$t = \tan \frac{\omega}{2}$ envoie ω sur tout $\mathbb{R}$

Changement de variable : $\xi = \frac{t+2-\sqrt{t^2+4}}{2t}$

Simplification du problème

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(t)}{s(t)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(\frac{1-2\xi}{\xi(1-\xi)})}{s(\frac{1-2\xi}{\xi(1-\xi)})} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \tfrac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$
$$\downarrow$$
$$\xi = \frac{t+2-\sqrt{t^2+4}}{2t}$$
$$\xi \in \Xi \subseteq [0, 1]$$

Simplification du problème

$$\underline{\beta}^2 \leq \frac{v(z)}{w(z)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{r(t)}{s(t)} \leq \overline{\beta}^2$$

$$\underline{\beta}^2 \leq \frac{p(\xi)}{q(\xi)} \leq \overline{\beta}^2$$

$$z = e^{j\omega}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$\downarrow$$
$$t = \tan \frac{\omega}{2}$$
$$\forall \omega \in \Omega \subseteq [0, \pi]$$
$$t \in [-\infty, \infty]$$
$$\downarrow$$
$$\xi = \frac{t+2-\sqrt{t^2+4}}{2t}$$
$$\xi \in \Xi \subseteq [0, 1]$$

Même plus simple

Le problème se résume à montrer que $\forall \xi \in \Xi \subseteq [0,1]$

$$f(\xi) \geq 0$$

avec $f \in \mathbb{R}[\xi]$ donné par

$$f(\xi) = q(\xi)^2 \left(\overline{\beta}^2 - \underline{\beta}^2\right)^2 - \left(p(\xi) - \left(\underline{\beta}^2 + \overline{\beta}^2\right) q(\xi)\right)^2$$

Même plus simple

Le problème se résume à montrer que $\forall \xi \in \Xi \subseteq [0,1]$

$$f(\xi) \geq 0$$

avec $f \in \mathbb{R}[\xi]$ donné par

$$f(\xi) = q(\xi)^2 \left(\overline{\beta}^2 - \underline{\beta}^2\right)^2 - \left(p(\xi) - \left(\underline{\beta}^2 + \overline{\beta}^2\right) q(\xi)\right)^2$$

Vérification que f est positif ou nul

Notre algorithme est basé sur :

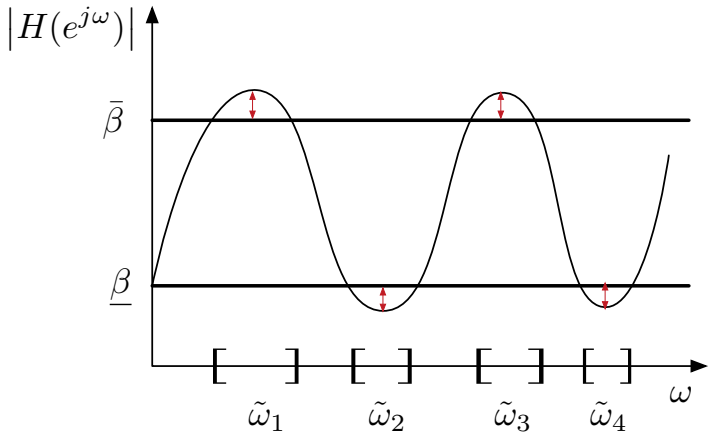
- la technique de Sturm qui donne
  le nombre de zéros réels d'un polynôme
- une subdivision en sous-intervalles
- des évaluations en arithmétique d'intervalles multi-précision

Algorithme de vérification

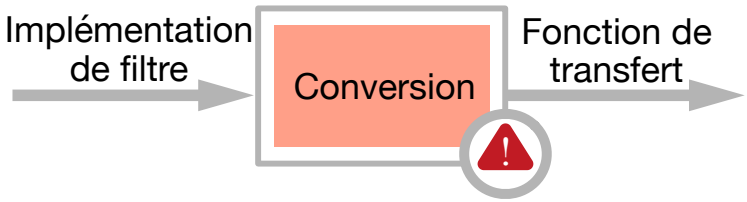
La fonction de transfert vérifie-t-elle les spécifications fréquentielles?

Oui

Non



Calcul de la fonction de transfert



Calcul de la fonction de transfert

Fonction de transfert de la représentation d'état :

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d$$

Calcul de la fonction de transfert

Fonction de transfert de la représentation d'état :

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1})^{-1}\boldsymbol{b} + d$$

Nous calculons une approximation $\widehat{H}(z)$ de $H(z)$:

$$\widehat{H}(z) = \frac{\sum_i \widehat{b}_i z^{-i}}{\sum_i \widehat{a}_i z^{-i}}$$

☞ L'erreur $\left|\left(H - \widehat{H}\right)(e^{j\omega})\right|$ peut être rendue arbitrairement petite.

Calcul de la fonction de transfert

Fonction de transfert de la représentation d'état :

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1})^{-1}\boldsymbol{b} + d$$

Nous calculons une approximation $\widehat{H}(z)$ de $H(z)$:

$$\widehat{H}(z) = \frac{\sum_i \widehat{b}_i z^{-i}}{\sum_i \widehat{a}_i z^{-i}}$$

☞ L'erreur $\left|\left(H - \widehat{H}\right)(e^{j\omega})\right|$ peut être rendue arbitrairement petite.

Problème

Il nous faut une borne rigoureuse sur l'erreur $\left|\left(H - \widehat{H}\right)(e^{j\omega})\right|$.

Comment borner l'erreur

Calcul de la fonction de transfert $H(z)$ d'une représentation d'état $\mathcal{S}$

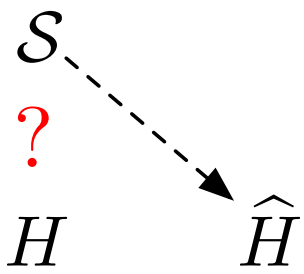
$\mathcal{S}$

?

H

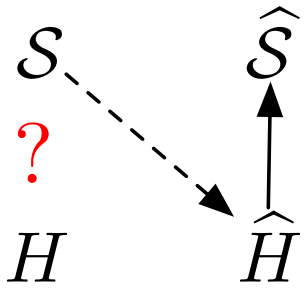
Comment borner l'erreur

Calcul de la fonction de transfert $H(z)$ d'une représentation d'état $\mathcal{S}$



Comment borner l'erreur

Calcul de la fonction de transfert $H(z)$ d'une représentation d'état $\mathcal{S}$



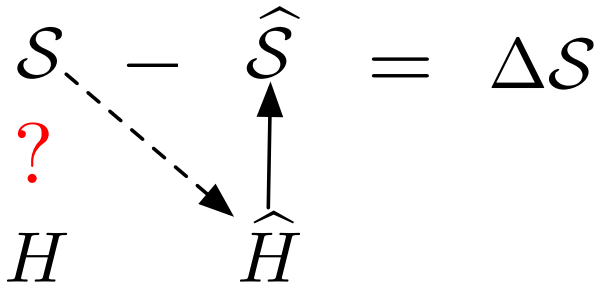
Transformation de $\widehat{H}$ vers $\widehat{\mathcal{S}}$ est exacte :

$$\widehat{\boldsymbol{A}} = \begin{pmatrix} -\widehat{a}_1 & 1 & & \\ \vdots & & \ddots & \\ \vdots & & & 1 \\ -\widehat{a}_n & 0 & \dots & 0 \end{pmatrix} \quad \widehat{\boldsymbol{b}} = \begin{pmatrix} \widehat{b}_1 - \widehat{a}_1\widehat{b}_0 \\ \vdots \\ \vdots \\ \widehat{b}_n - \widehat{a}_n\widehat{b}_0 \end{pmatrix}$$

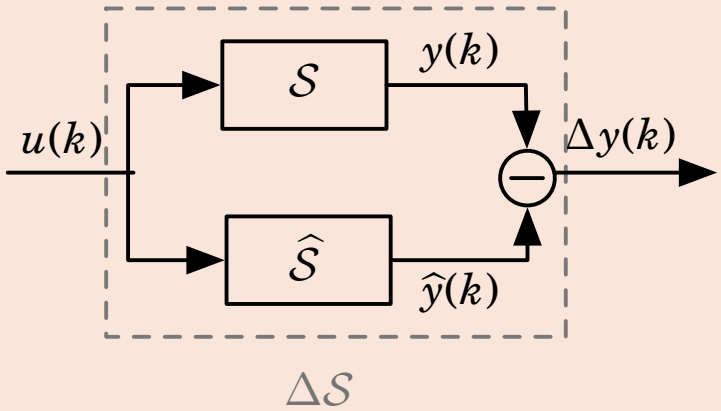
$$\widehat{\boldsymbol{c}} = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \qquad \widehat{d} = \widehat{b}_0$$

Comment borner l'erreur

Calcul de la fonction de transfert $H(z)$ d'une représentation d'état $\mathcal{S}$

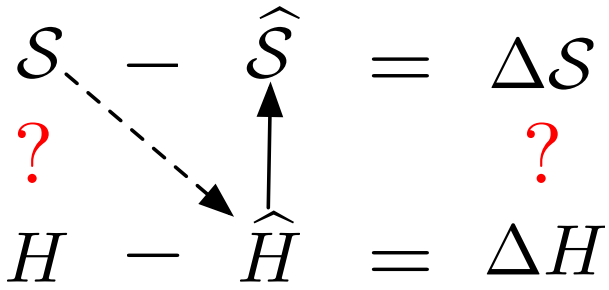


La différence de deux filtres est définie comme :



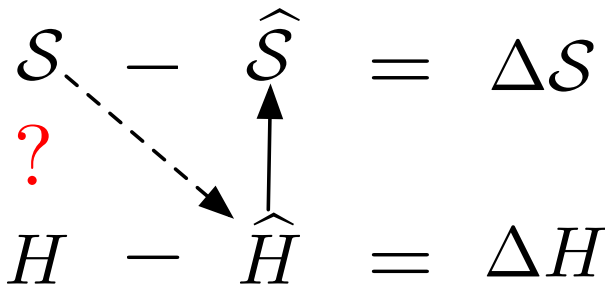
Comment borner l'erreur

Calcul de la fonction de transfert $H(z)$ d'une représentation d'état $\mathcal{S}$



Comment borner l'erreur

Calcul de la fonction de transfert $H(z)$ d'une représentation d'état $\mathcal{S}$



Relation entre $\Delta\mathcal{S}$ et ΔH :

$$\left|\left(H - \widehat{H}\right)(e^{j\omega})\right| \leq \langle\langle \Delta\mathcal{S} \rangle\rangle, \quad \forall\omega \in [0, 2\pi]$$

où $\langle\langle \Delta\mathcal{S} \rangle\rangle$ est encore une fois le WCPG du système $\Delta\mathcal{S}$.

Évaluation fiable du WCPG

WCPG fiable est nécessaire pour :

- déterminer la dynamique des variables
- analyser l'erreur induite par les calculs en précision finie
- borner l'erreur d'approximation d'une fonction de transfert

Évaluation fiable du WCPG

WCPG fiable est nécessaire pour :

- déterminer la dynamique des variables
- analyser l'erreur induite par les calculs en précision finie
- borner l'erreur d'approximation d'une fonction de transfert

Problème dans le cas de filtres MIMO

Calculer $\boldsymbol{S}$, approximation de la matrice

$$\langle\langle \mathcal{H} \rangle\rangle = |\boldsymbol{D}| + \sum_{k=0}^{\infty} \left| \boldsymbol{C}\,\boldsymbol{A}^k\,\boldsymbol{B} \right|,$$

telle que pour un ε donné *a priori*

$$|\langle\langle \mathcal{H} \rangle\rangle - \boldsymbol{S}| < \varepsilon$$

☞ Nous utilisons une arithmétique en *multi-précision dynamique.*

Borne de troncature

Nous cherchons N tel que l'erreur de troncature vérifie

$$\left| \sum_{k=0}^{\infty} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right| \quad - \quad \sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right| \right| \leq \varepsilon_1$$

Solution mathématique :

$$N \geq \left\lceil \frac{\log_2 \frac{\varepsilon_1}{\|\boldsymbol{M}\|_{min}}}{\log_2 \rho(\boldsymbol{A})} \right\rceil$$

où

$$\boldsymbol{M} = \sum_{l=1}^{n} \frac{|\boldsymbol{R}_l|}{1-|\boldsymbol{\lambda}_l|} \frac{|\boldsymbol{\lambda}_l|}{\rho(\boldsymbol{A})}$$

$$(\boldsymbol{R}_l)_{i,j} = \boldsymbol{C}_{i,l}\boldsymbol{B}_{l,j}$$

Solution pratique :

- Valeurs propres calculées avec LAPACK
- Arithmétique d'intervalles
- Théorie des Inclusions Vérifiées (Rump)

Évaluation : mise à la puissance

$$\sum_{k=0}^{N} \left| C A^k B \right|$$

Nous proposons :

$$A = X E X^{-1}$$

d'où $A^k = X E^k X^{-1}$

où

X sont les vecteurs propres

E sont les valeurs propres

Évaluation : mise à la puissance

$$\left| \sum_{k=0}^{N} \left| C\,A^k\,B \right| \quad - \quad \sum_{k=0}^{N} \left| C\,V\,T^k\,V^{-1}\,B \right| \right| \leq \varepsilon_2$$

Nous proposons :

$$A = X\,E\,X^{-1}$$
$$\text{d'où } A^k = X\,E^k\,X^{-1}$$

où

X sont les vecteurs propres

E sont les valeurs propres

En pratique :

$$V \approx X \quad \text{et} \quad T = V^{-1}\,A\,V + \Delta$$

- Opérations en multi-précision avec la borne de l'erreur donnée *a priori*
- Δ est contrôlée pour satisfaire la borne ε_2 sur l'erreur propagée
- $\|T\|_2 \leq 1$ vérifié rigoureusement avec le théorème de Gershgorin

Évaluation : sommation

Nous continuons par analogie

$$\left|\sum_{k=0}^{N} |\boldsymbol{C}\,\boldsymbol{V}\,\boldsymbol{T}^k\,\boldsymbol{V}^{-1}\boldsymbol{B}| - \sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{T}^k\,\boldsymbol{B}'|\right| \leq \varepsilon_3$$

$$\left|\sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{T}^k\,\boldsymbol{B}'| - \sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{P}_k\,\boldsymbol{B}'|\right| \leq \varepsilon_4$$

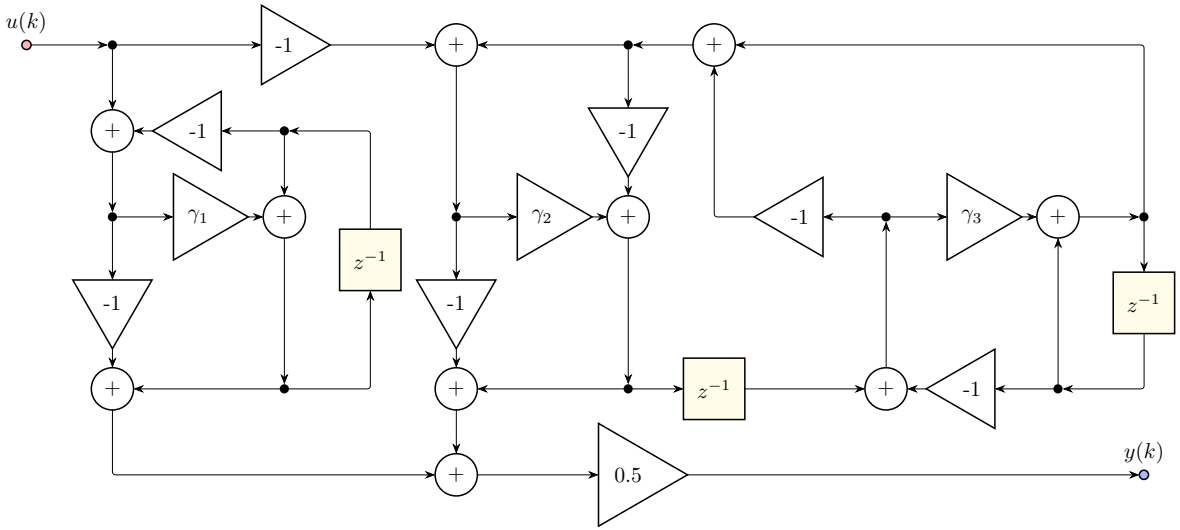
$$\left|\sum_{k=0}^{N} |\boldsymbol{C}'\,\boldsymbol{P}_k\,\boldsymbol{B}'| - \sum_{k=0}^{N} |\boldsymbol{L}_k|\right| \leq \varepsilon_5$$

$$\left|\sum_{k=0}^{N} |\boldsymbol{L}_k| - \boldsymbol{S}_N\right| \leq \varepsilon_6$$

☞ **Trois briques de base :** $\boldsymbol{X}\,\boldsymbol{Y} + \boldsymbol{Z}$, $\boldsymbol{X} + |\boldsymbol{Y}|$, $\boldsymbol{X}^{-1}$

Exemple de vérification en fréquence

Une implémentation de notre filtre-exemple :



$$\gamma_1 = 89 \cdot 2^{-8}$$
$$\gamma_2 = 43 \cdot 2^{-7}$$
$$\gamma_3 = 11 \cdot 2^{-7}$$

Les spécifications :

$$\begin{cases} 10^{\frac{1}{20}} \leq \left|H(e^{j\omega})\right| \leq 10^{\frac{3}{20}} & \forall \omega \in [0, \frac{1}{10}\pi] \quad \text{(passe-bande)} \\ \left|H(e^{j\omega})\right| \leq 10^{-\frac{20}{20}} & \forall \omega \in [\frac{3}{10}\pi, \pi] \quad \text{(coupe-bande)} \end{cases}$$

Résultat : ✓ le filtre implémenté respecte les spécifications.

Temps de vérification : 1.9 s

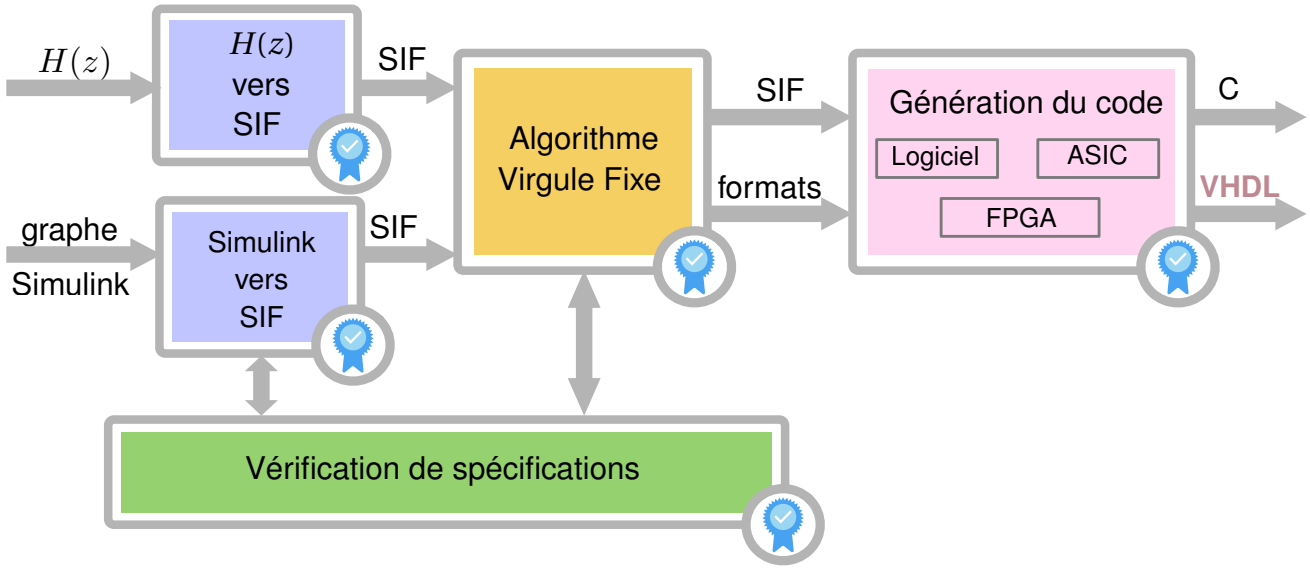
Exemple de vérification

Pendant ces 1.9 secondes...

- Arithmétique rationnelle
  - Réécriture de $H(z)$ en $f(\xi)$
  - Calcul de suites de Sturm
- Arithmétique d'intervalles
  - Inclusions vérifiées de valeurs propres
  - Borne de troncature N
- Arithmétique multi-précision dynamique
  - Calcul de cercles de Gershgorin
  - Évaluation du WCPG
- Arithmétique virgule flottante IEEE 754
  - Calcul de valeurs propres avec LAPACK

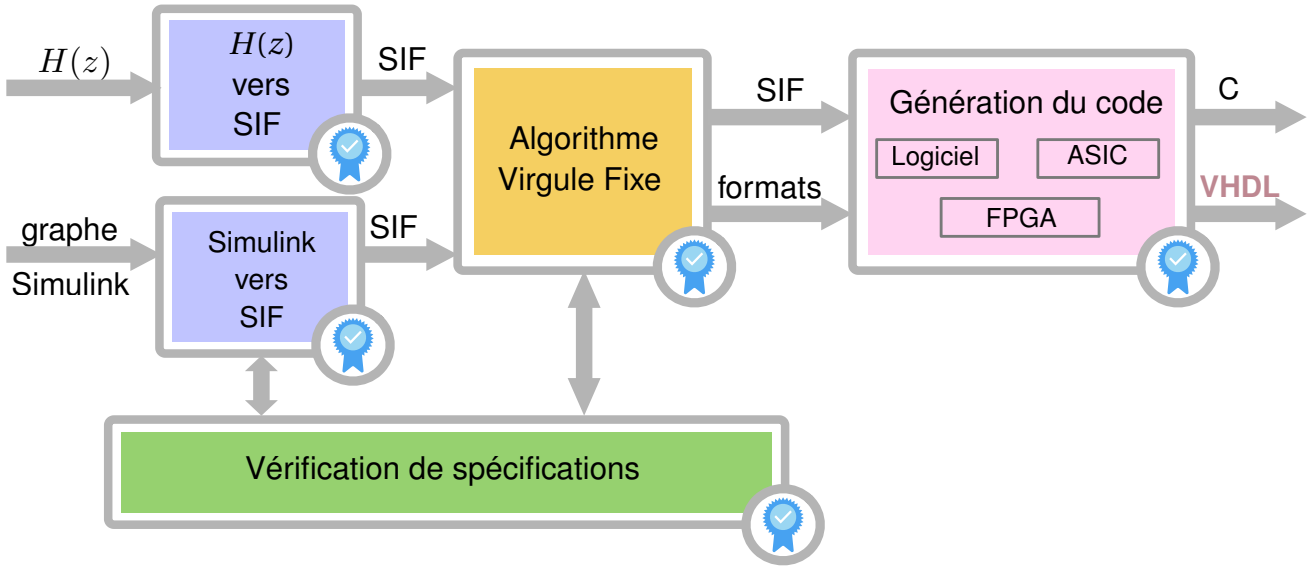
. . . pour vérifier une implémentation en virgule fixe !

En résumé



- Nous avons rendu l'implémentation de filtres fiable et automatique
  - Utilisation rigoureuse d'arithmétiques diverses
  - Implémentations virgule fixe vérifiant des bornes d'erreur a priori et en fréquentiel et en temporel
- Nous avons étendu la chaîne d'outils du générateur de filtres
  - Nouvelles conversions depuis d'autres formats d'entrée
  - Branchement avec FloPoCo pour l'implémentation FPGA

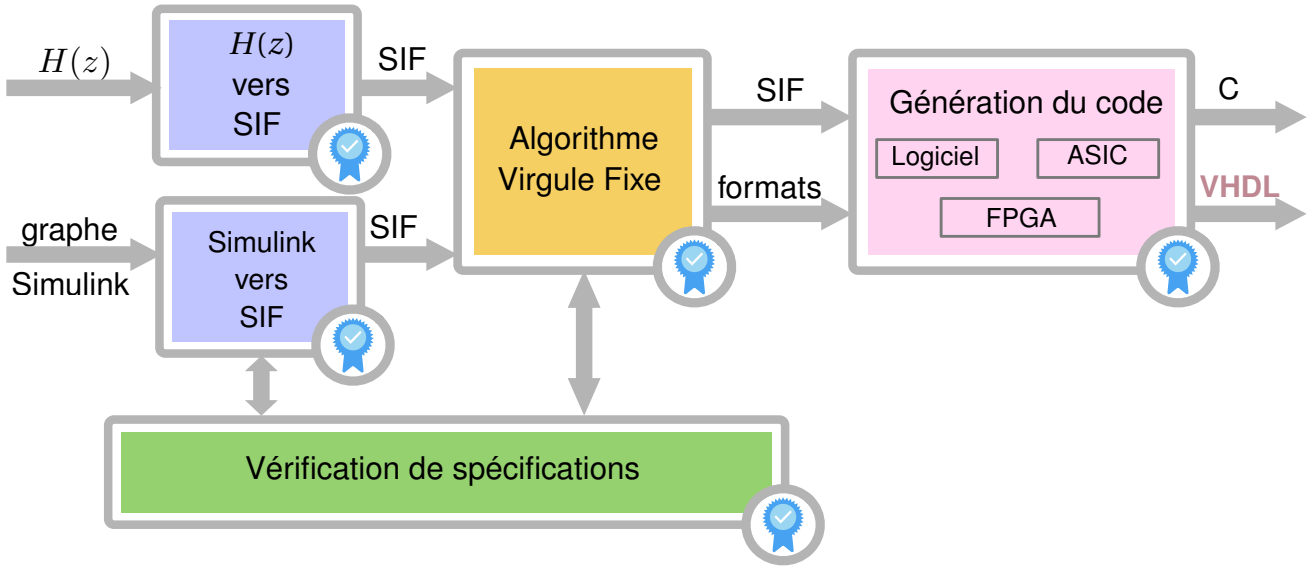
Développement de logiciels



- Outils open-source
- Implémentation en C/C++, Sollya, Python et Matlab
- $\approx$ 15000 lignes de code
- Génération de VHDL avec FloPoCo

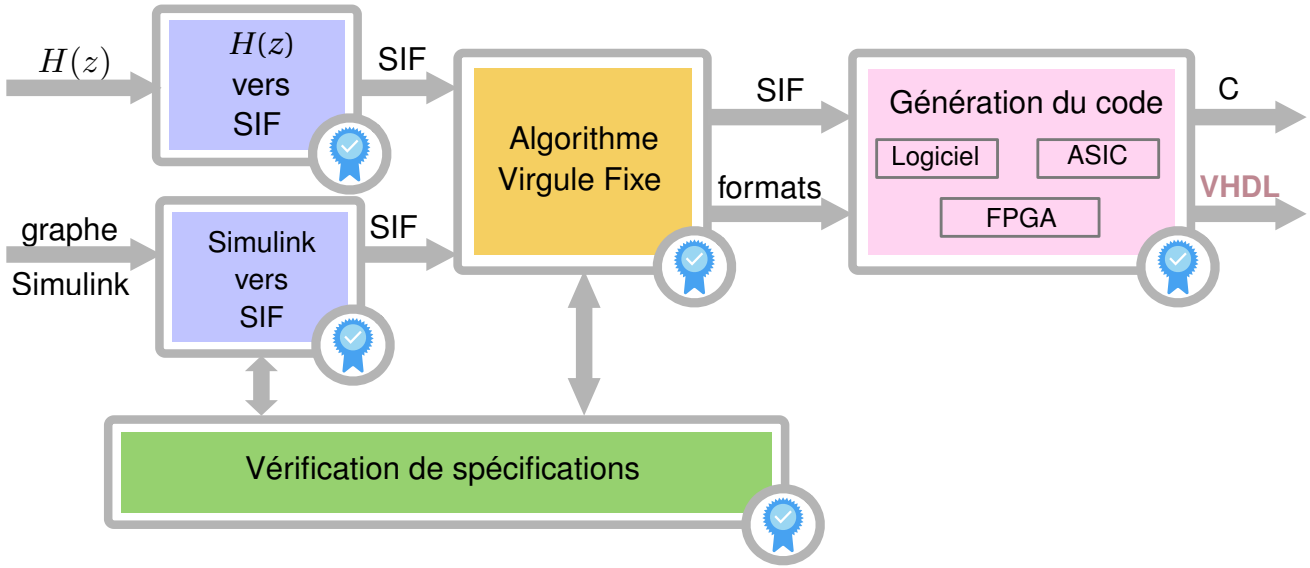
FIXIF TOOLBOX

Perspectives : demain à 9h00...



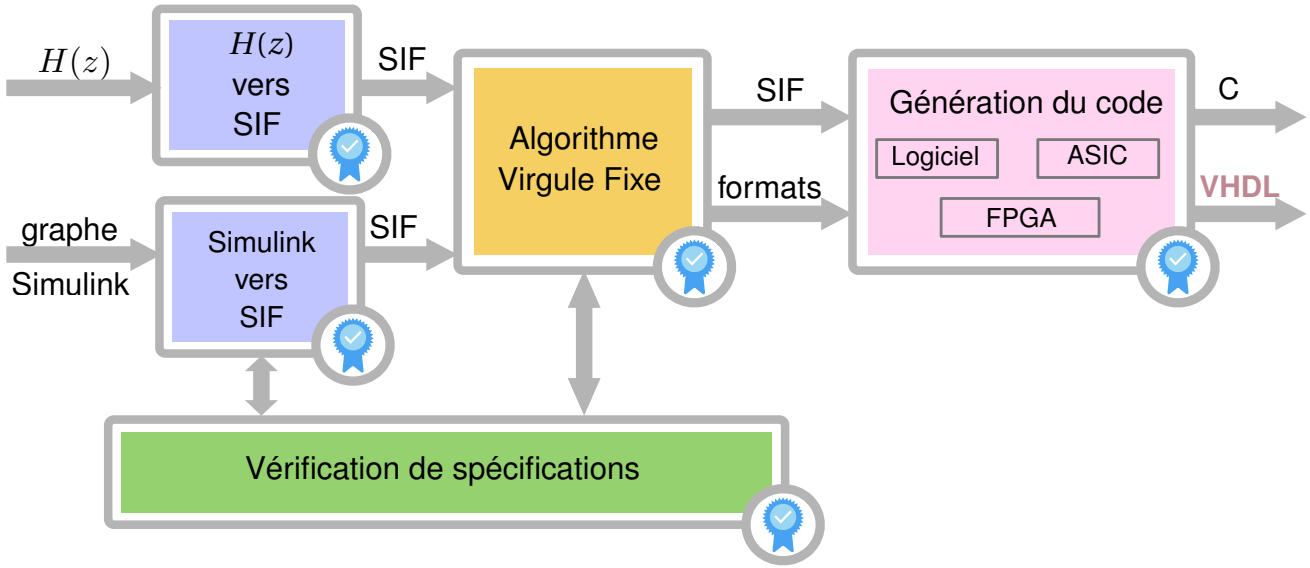
- Gagner un peu de place
  - Résoudre le problème « off-by-one » $(0 \leq \widehat{m}_y - m_y \leq 1)$
- Gagner plus de place
  - Prendre en compte des propriétés supplémentaires des signaux d'entrée (propriétés spectrales)
- Étudier divers algorithmes selon plusieurs métriques
  - LUTs, surface
  - précision

Perspectives : moyen terme



- Implémentation de filtres fiables
  - Intégrer les méthodes de conception des fonctions de transfert
  - Relâcher les contraintes de spécifications fréquentielles
- Outils arithmétiques
  - Calculer des valeurs propres d'une matrice en précision variée
  - Résoudre des équations de Lyapunov, etc.
- Optimisation
  - Entourer le générateur avec des boucles d'optimisation

Perspectives : long terme



- Côté arithmétique
  - Outils numériques pour l'algèbre linéaire
  - Évaluation des approximations rationnelles
- Côté arithmétique pour le traitement du signal
  - Concevoir la fonction de transfert avec les meilleurs coefficients quantifiés
  - Considérer les filtres non-linéaires, Kalman, etc.

Merci

Thank you

Спасибо

Дякую

Liste de publications

- A.V., C. Lauter., T. Hilaire and M. Mezzarobba Rigorous Determination of Recursive Filter Fixed-Point Implementation with Input Signal Frequency Specifications In *ASILOMAR'51*, Nov 2017
- T. Hilaire, A.V. Error analysis methods for the fixed-point implementation of linear systems. In *IEEE SiPS 2017*, Oct 2017
- F. Qureshi, A. V., J. Takala and T. Hilaire. Multiplierless Unified Architecture for Mixed Radix-2/3/4 FFTs. In *EUSIPCO'25*, Aug 2017
- A.V., C. Lauter and T. Hilaire. Reliable verification of digital implemented filters against frequency specifications. In *IEEE ARITH'24*, July 2017
- T. Hilaire, A.V. and M. Ravoson. Reliable fixed-point implementation of linear data-flows. In *IEEE SiPS 2016*, Oct 2016
- A.V., T. Hilaire and C. Lauter. Determining fixed-point formats for a digital filter implementation using the worst-case peak gain measure. In *ASILOMAR'49*, Nov 2015
- A.V. and T. Hilaire. Fixed-point implementation of lattice wave digital filter: Comparison and error analysis. In *EUSIPCO'23*, Sep 2015
- A.V., T. Hilaire, and C. Lauter. Reliable evaluation of the worst-case peak gain matrix in multiple precision. In *IEEE ARITH'22*, Jun 2015
- F. de Dinechin, T. Hilaire, M. Istoan, A.V. LTI Filters Computed Just Right Technical report

SIF

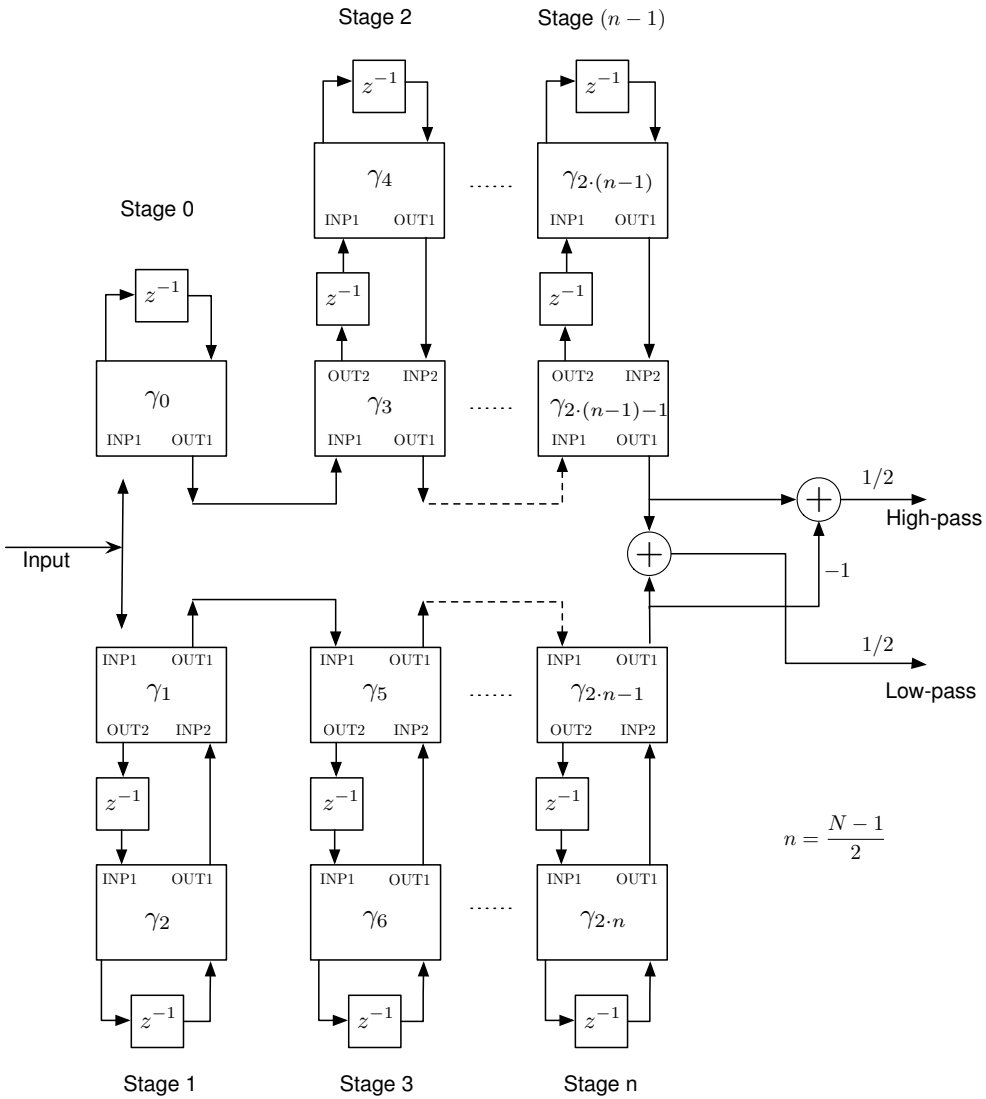
A MIMO LTI system in SIF representation is characterized by

$$\begin{pmatrix} \boldsymbol{J} & \boldsymbol{0} & \boldsymbol{0} \\ -\boldsymbol{K} & \boldsymbol{I}_n & \boldsymbol{0} \\ -\boldsymbol{L} & \boldsymbol{0} & \boldsymbol{I}_p \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k+1) \\ \boldsymbol{x}(k+1) \\ \boldsymbol{y}(k) \end{pmatrix} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{0} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{t}(k) \\ \boldsymbol{x}(k) \\ \boldsymbol{u}(k) \end{pmatrix}$$

By rewriting we obtain

$$\begin{cases} \boldsymbol{J}\boldsymbol{t}(k+1) = \boldsymbol{M}\boldsymbol{x}(k) + \boldsymbol{N}\boldsymbol{u}(k) \\ \boldsymbol{x}(k+1) = \boldsymbol{K}\boldsymbol{t}(k+1) + \boldsymbol{P}\boldsymbol{x}(k) + \boldsymbol{Q}\boldsymbol{u}(k) \\ \boldsymbol{y}(k) = \boldsymbol{L}\boldsymbol{t}(k+1) + \boldsymbol{R}\boldsymbol{x}(k) + \boldsymbol{S}\boldsymbol{u}(k) \end{cases}$$

Lattice Wave Digital Filters



The error filter $\mathcal{H}_\Delta$

Denote

$$\boldsymbol{\varepsilon}(k) = \begin{pmatrix} \boldsymbol{\varepsilon}_x(k) \\ \varepsilon_y(k) \end{pmatrix}$$

$$\boldsymbol{\Delta}_x(k) = \boldsymbol{x}^\Diamond(k) - \boldsymbol{x}(k)$$

$$\boldsymbol{\Delta}_y(k) = \boldsymbol{y}^\Diamond(k) - \boldsymbol{y}(k)$$

Then, the filter $\mathcal{H}_\Delta = \mathcal{H} - \mathcal{H}^\Diamond$ is

$$\mathcal{H}_\Delta \begin{cases} \boldsymbol{\Delta}_x(k+1) &= \boldsymbol{A}\boldsymbol{\Delta}_x(k) &+ \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \boldsymbol{\varepsilon}(k) \\ \boldsymbol{\Delta}_y(k) &= \boldsymbol{C}\boldsymbol{\Delta}_x(k) &+ \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \boldsymbol{\varepsilon}(k) \end{cases}$$

Worst-Case Peak Gain

Sensitive real-life filter

Input: A highly sensitive 5^{th} order transfer function from industrial application[1].
Problem: given inputs from interval $[-1.125; 1.125]$, determine the interval for output variables.
Result:

- Naive WCPG, i.e. summing 1000 terms in double precision: $\bar{y} = 1.6238497 \times 1.125 = 0.8693\ldots$
- Our WCPG with $\varepsilon = 2^{-53}$: $\bar{y} = 1.9997191 \times 1.125 = 1.1697\ldots$

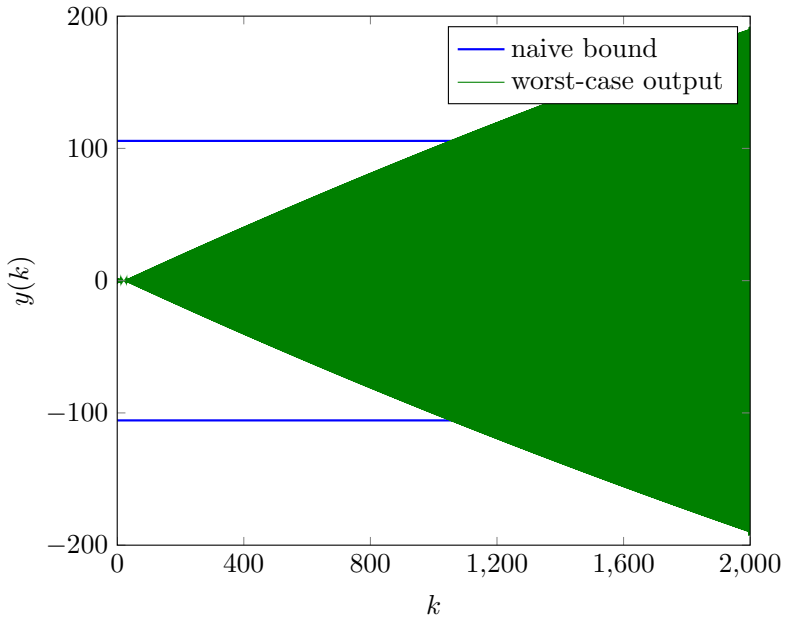
[1] Obtained from Xilinx

Worst-Case Peak Gain

Random filter

Input: Consider a stable 5^{th} order random SISO filter
Problem: given inputs from interval $[-1; 1)$, determine the interval for output variables.
Result:

- Naive WCPG (1000 terms in double precision) $\bar{y} = 105.66...$
- Our WCPG with $\varepsilon = 2^{-53}$: $\bar{y} = 772.48...$

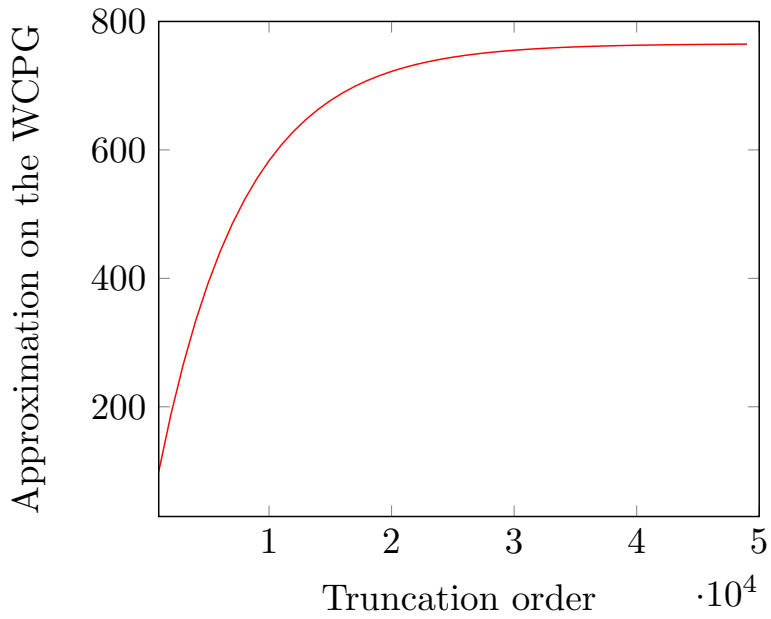


Worst-Case Peak Gain

Random filter

Input: Consider a stable 5^{th} order random SISO filter
Problem: given inputs from interval $[-1; 1)$, determine the interval for output variables.
Result:

- Naive WCPG (1000 terms in double precision) $\bar{y} = 105.66...$
- Our WCPG with $\varepsilon = 2^{-53}$: $\bar{y} = 772.48...$



Worst-Case Peak Gain

Examples

Example 1: comes from Control Theory, describes a controller of vehicle longitudinal oscillation

Example 2: 12th-order Butterworth filter

	Example 1			Example 2		
sizes n, p and q	$n = 10,$	$p = 11,$	$q = 1$	$n = 12,$	$p = 1,$	$q = 25$
$1 - \rho(\boldsymbol{A})$	1.39×10^{-2}			8.65×10^{-3}		
$\max(\boldsymbol{S}_N)$	3.88×10^{1}			5.50×10^{9}		
$\min(\boldsymbol{S}_N)$	1.29×10^{0}			1.0×10^{0}		
ε	2^{-5}	2^{-53}	2^{-600}	2^{-5}	2^{-53}	2^{-600}
N	220	2153	29182	308	4141	47811
Inversion iterations	0	2	4	2	3	5
overall max precision (bits)	212	293	1401	254	355	1459
$\boldsymbol{V}^{-1}$ max precision (bits)	106	173	727	148	204	756
$\boldsymbol{P}_N$ max precision (bits)	64	84	639	64	86	640
$\boldsymbol{S}_N$ max precision (bits)	64	79	630	64	107	658
Overall execution time (sec)	0.11	1.53	60.06	0.85	11.54	473.20

Transfer Function of a State-Space

Transfer function of a single-input single-output state-space $\mathcal{S}$:

$$H(z) = \boldsymbol{c}(z\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{b} + d$$

Using the eigendecomposition $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{-1}$:

$$H(z) = \frac{P(z)}{Q(z)} + d$$

$$P(z) = \sum_{i=1}^{n} (\boldsymbol{c}\boldsymbol{X})_i (\boldsymbol{X}^{-1}\boldsymbol{b})_i \prod_{j \neq i} (z - \lambda_j)$$

$$Q(z) = \prod_{j=1}^{n} (z - \lambda_j)$$

We compute an approximation $\widehat{H}(z)$ in Multiple Precision arithmetic with `mpmath`.

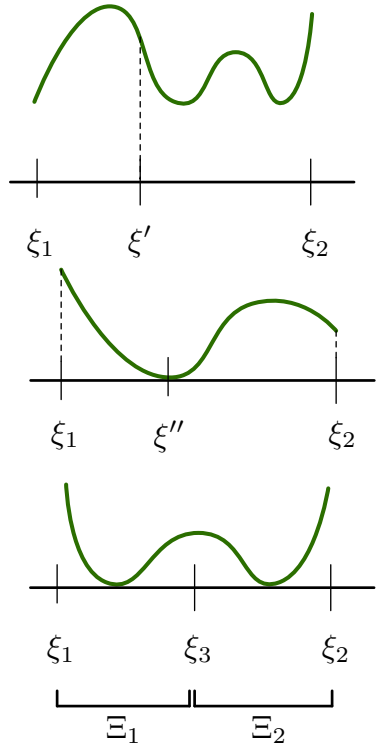
Verification of non-negativity

To verify $f(\xi) \geq 0$, $\forall \xi \in \Xi = [\xi_1, \xi_2] \subseteq [0, 1]$ we check if

(i) $f(\xi)$ has no zeros
$f(\xi') > 0$ for some $\xi' \in [\xi_1, \xi_2]$

(ii) $f(\xi)$ has one zero
$f(\xi_1) > 0$ and $f(\xi_2) > 0$

(iii) interval Ξ can be split into subintervals s.t. (i) or (ii) are satisfied for every subinterval



We use Sollya tool for the implementation

- Number of zeros: Sturm's theorem
- Evaluations: interval multiple precision arithmetic

Verification of specifications

Sturm's technique

Sturm's sequence is a sequence of polynomials $p_0(x), \ldots, p_m(x)$:

$$
\begin{aligned}
p_0(x) &= p(x) \\
p_1(x) &= p'(x) \\
p_2(x) &= -rem(p_0, p_1) = p_1(x)q_0(x) - p_0(x), \\
p_3(x) &= -rem(p_1, p_2) = p_2(x)q_1(x) - p_1(x), \\
&\ldots \\
0 &= -rem(p_{m-1}, p_m)
\end{aligned}
$$

Verification of specifications

Numerical examples

Input: four realizations of the same filter

Problem: verify realizations after coefficient quantization to 32/16/8 bits

Results:

	wordlength	32	16	8
DFIIt	margin	✓	unstable	unstable
	time	12.49s	-	-
ρ DFIIt	margin	✓	✓	4.68e-3 dB
	time	13.12s	4.19s	104.01s
State-Space Balanced	margin	6.16e-10 dB	✓	6.71e-1 dB
	time	12.27s	18.18s	92.05s
Lattice Wave	margin	3.80e-10 dB	✓	1.73e-2 dB
	time	920.88s	4.58s	200.83s

Verification of specifications

Numerical examples

Input: four simple frequency specifications

Problem: Verify and compare transfer function design methods.

Results: comparison of SciPy in Python and Matlab

		Butterworth	Chebyshev	Elliptic
		margin (dB)	margin (dB)	margin (dB)
lowpass	Matlab	1.29e-17	7.93e-17	✓
	SciPy	2.14e-15	4.48e-2	4.48e-2
highpass	Matlab	2.77e-16	6.94e-17	4.48e-2
	SciPy	3.02e-15	2.29e-16	4.48e-2
bandpass	Matlab	3.04e-17	✓	✓
	SciPy	✓	4.48e-2	4.48e-2
bandstop	Matlab	4.59e-16	3.09e-15	✓
	SciPy	✓	6.36e-15	7.02e-6

Verification of specifications

Numerical examples

Filter implementation: 14^{th} order bandpass filter
Specifications:

$$\begin{cases} 0\text{dB} \leq & |H(e^{i\omega})| \leq & -80\text{dB} & \forall\omega \in [0, 17\text{kHz}] & \text{(stopband)} \\ & |H(e^{i\omega})| \leq & 1 - 10^{-4}\text{dB} & \forall\omega \in [21\text{kHz}, 25\text{kHz}] & \text{(passband)} \\ & |H(e^{i\omega})| \leq & -80\text{dB} & \forall\omega \in [27\text{kHz}, 30\text{kHz}] & \text{(stopband)} \end{cases}$$

Verification result: implemented filter *does not* pass the verification against frequency constraints
Verification time: 53 s

Verification of specifications

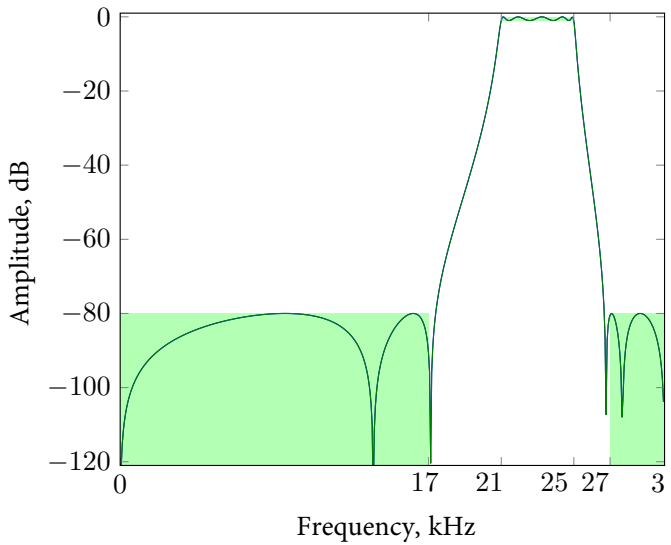
Numerical examples

Filter implementation: 14^{th} order bandpass filter

Verification result: implemented filter *does not* pass the verification against frequency constraints

Verification time: 53 s

Frequency response:



Verification of specifications

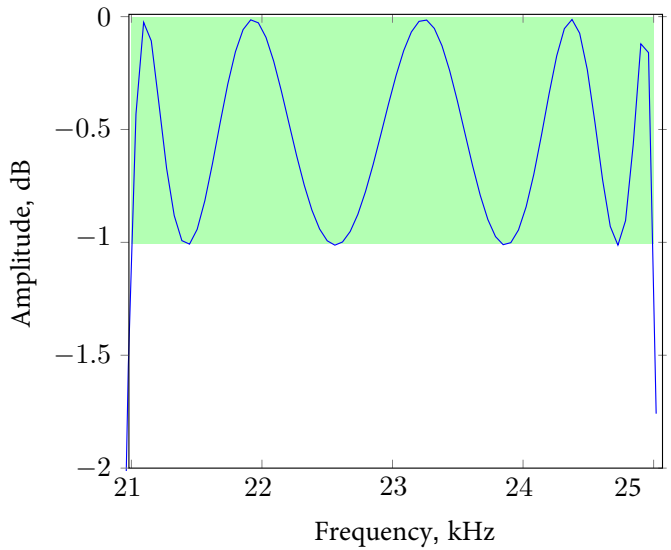
Numerical examples

Filter implementation: 14^{th} order bandpass filter

Verification result: implemented filter *does not* pass the verification against frequency constraints

Verification time: 53 s

Frequency response:



Verification of specifications

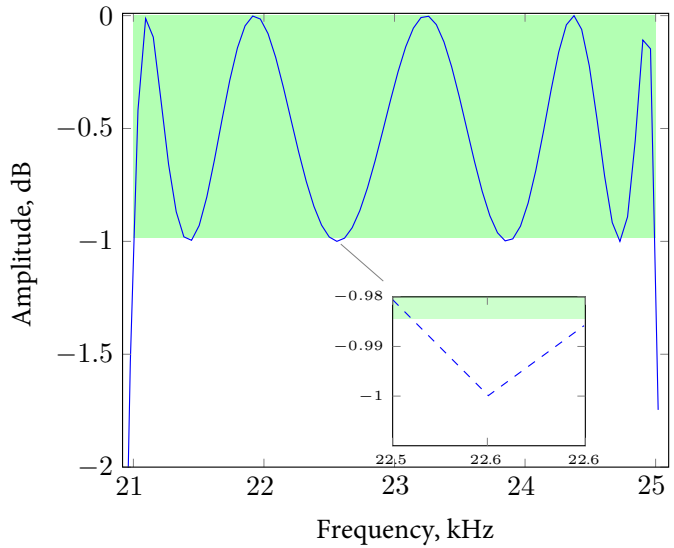
Numerical examples

Filter implementation: 14^{th} order bandpass filter

Verification result: implemented filter *does not* pass the verification against frequency constraints

Verification time: 53 s

Frequency response:



Binding with FloPoCo

FloPoCo:

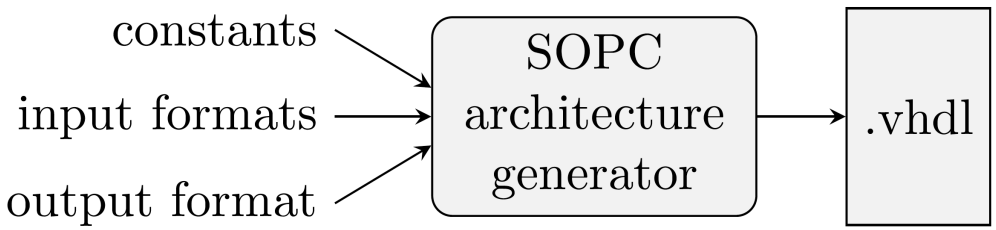


Figure: Interface to a Sum-Of-Product-by-Constant generator

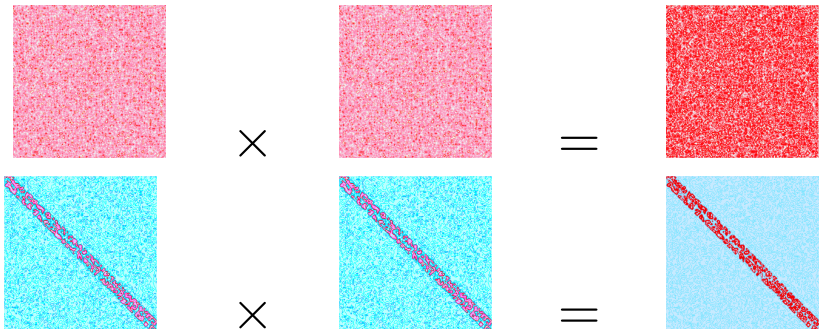
Our tool: we deduce a lower bound on the error of computation of each Sum-of-Product s.t. the error-bound *on the filter's output* is respected.

- Push-button
- Can implement any structure
- No need to quantize coefficients
- Reliable

Worst-Case Peak Gain

Powering matrix A

$$\sum_{k=0}^{N} \left| C A^k B \right|$$



cancellation

less cancellation

$$A = XEX^{-1} \qquad V \approx X \text{ and } T \approx E$$

$$T \approx V^{-1} \times A \times V$$

$$A^k \approx V \times T^k \times V^{-1}$$

Worst-Case Peak Gain

Step 3

$$\left| \sum_{k=0}^{N} \left| \boldsymbol{C}\,\boldsymbol{V}\,\boldsymbol{T}^k\,\boldsymbol{V}^{-1}\,\boldsymbol{B} \right| \quad - \quad \sum_{k=0}^{N} \left| \boldsymbol{C'}\,\boldsymbol{T}^k\,\boldsymbol{B'} \right| \right| \leq \varepsilon_3$$

Step 3 Compute the products $\boldsymbol{C}\,\boldsymbol{V}$ and $\boldsymbol{V}^{-1}\boldsymbol{B}$ such that the propagated error of matrix multiplications is bounded by ε_3.

$$\sum_{k=0}^{\infty} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\,\boldsymbol{V}\,\boldsymbol{T}^k\,\boldsymbol{V}^{-1}\,\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C'}\,\boldsymbol{T}^k\,\boldsymbol{B'} \right|$$

Worst-Case Peak Gain

Step 4

$$\left| \sum_{k=0}^{N} |C' T^k B'| \quad - \quad \sum_{k=0}^{N} |C' P_k B'| \right| \leq \varepsilon_4$$

$$P_0 := I$$
$$P_k := T \otimes P_{k-1}$$

$$\sum_{k=0}^{\infty} |C A^k B|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C A^k B|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C V T^k V^{-1} B|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C' T^k B'|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |C' P_k B'|$$

Step 4 Compute the powers P_k of matrix T such that the propagated error of matrix multiplications is bounded by ε_4.

Worst-Case Peak Gain

Step 5

$$\left| \sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}| \quad - \quad \sum_{k=0}^{N} |\boldsymbol{L}_k| \right| \leq \varepsilon_5$$

$$\boldsymbol{L}_k := \boldsymbol{C'} \otimes (\boldsymbol{P}_k \otimes \boldsymbol{B'})$$

Step 5 Compute on each step the matrix product $\boldsymbol{C'T}^k\boldsymbol{B'}$ such the overall error of these multiplications on each step is bounded by ε_5.

$$\sum_{k=0}^{\infty} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CVT}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{C'T}^k\boldsymbol{B'}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}|$$
$$\downarrow$$

Worst-Case Peak Gain

Step 5

$$\left| \sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}| \quad - \quad \sum_{k=0}^{N} |\boldsymbol{L}_k| \right| \leq \varepsilon_5$$

$$\boldsymbol{L}_k := \boldsymbol{C'} \otimes (\boldsymbol{P}_k \otimes \boldsymbol{B'})$$

Step 5 Compute on each step the matrix product $\boldsymbol{C'T}^k\boldsymbol{B'}$ such the overall error of these multiplications on each step is bounded by ε_5.

$$\sum_{k=0}^{\infty} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CA}^k\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{CVT}^k\boldsymbol{V}^{-1}\boldsymbol{B}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left|\boldsymbol{C'T}^k\boldsymbol{B'}\right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{C'P}_k\boldsymbol{B'}|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{L}_k|$$

Worst-Case Peak Gain

Step 6

$$\left| \sum_{k=0}^{N} |\boldsymbol{L}_k| \quad - \quad \boldsymbol{S}_N \right| \leq \varepsilon_6$$

$$\boldsymbol{S}_k := \boldsymbol{S}_{k-1} \oplus |\boldsymbol{L}_k|$$

Step 6 Compute the absolute value of matrix and accumulate it in the result such that the error is bounded by ε_6.

$$\sum_{k=0}^{\infty} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{A}^k\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}\boldsymbol{V}\boldsymbol{T}^k\boldsymbol{V}^{-1}\boldsymbol{B} \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} \left| \boldsymbol{C}'\boldsymbol{T}^k\boldsymbol{B}' \right|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{C}'\boldsymbol{P}_k\boldsymbol{B}'|$$
$$\downarrow$$
$$\sum_{k=0}^{N} |\boldsymbol{L}_k|$$
$$\downarrow$$
$$\boldsymbol{S}_N$$

Worst-Case Peak Gain

Algorithm

Taking $\varepsilon_i = \frac{1}{6}\varepsilon$ we obtain that $\varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_6 \leq \varepsilon$ hence the overall error bound is satisfied.

A floating-point evaluation of the WCPG:

Step 1: Compute N

Step 2: Compute $\boldsymbol{V}$

$\boldsymbol{T} \leftarrow \mathsf{inv}(\boldsymbol{V}) \otimes (\boldsymbol{A} \otimes \boldsymbol{V})$

Step 3: $\boldsymbol{B'} \leftarrow \mathsf{inv}(\boldsymbol{V}) \otimes \boldsymbol{B}$

$\boldsymbol{C'} \leftarrow \boldsymbol{C} \otimes \boldsymbol{V}$

$\boldsymbol{S}_{-1} \leftarrow |\boldsymbol{D}|$, $\boldsymbol{P}_{-1} \leftarrow \boldsymbol{I}_n$

for k **from** 0 **to** N **do:**

Step 4: $\quad \boldsymbol{P}_k \leftarrow \boldsymbol{T} \otimes \boldsymbol{P}_{k-1}$

Step 5: $\quad \boldsymbol{L}_k \leftarrow \boldsymbol{C'} \otimes (\boldsymbol{P}_k \otimes \boldsymbol{B'})$

Step 6: $\quad \boldsymbol{S}_k \leftarrow \boldsymbol{S}_{k-1} \oplus \mathsf{abs}(\boldsymbol{L}_k)$

end for

Worst-Case Peak Gain

Basic bricks

- `multiplyAndAdd`$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \delta)$: for $\boldsymbol{A} \in \mathbb{C}^{p \times n}$, $\boldsymbol{B} \in \mathbb{C}^{n \times q}$, $\boldsymbol{C} \in \mathbb{C}^{p \times q}$, computes a matrix $\boldsymbol{D} \in \mathbb{C}^{p \times q}$ such that

$$\boldsymbol{D} = \boldsymbol{A} \cdot \boldsymbol{B} + \boldsymbol{C} + \boldsymbol{\Delta},$$

  where the error-matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

The algorithm performs an error-free scalar multiplication and uses a modified software-implemented Kulisch-like accumulator.

Worst-Case Peak Gain

Basic bricks

- $\texttt{sumAbs}(\boldsymbol{A}, \boldsymbol{B}, \delta)$: for $\boldsymbol{A} \in \mathbb{R}^{p \times n}$, $\boldsymbol{B} \in \mathbb{C}^{p \times n}$, computes a matrix $\boldsymbol{C} \in \mathbb{R}^{p \times n}$ such that

$$\boldsymbol{C} = \boldsymbol{A} + |\boldsymbol{B}| + \boldsymbol{\Delta},$$

where the error matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

Worst-Case Peak Gain

Basic bricks

- $\texttt{inv}(V, \delta)$: for a complex square matrix $\boldsymbol{V} \in \mathbb{C}^{n \times n}$, computes a matrix $\boldsymbol{U} \in \mathbb{C}^{n \times n}$ such that

$$\boldsymbol{U} = \boldsymbol{V}^{-1} + \boldsymbol{\Delta},$$

  where the error matrix $\boldsymbol{\Delta}$ is bounded by $|\boldsymbol{\Delta}| < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

The algorithm is based on Newton-Raphson matrix iteration, requires a seed matrix in argument and works on certain conditions, easily verified in our case.

Worst-Case Peak Gain

Basic bricks

- `frobeniusNormUpperBound`$(\boldsymbol{A}, \delta)$: for $\boldsymbol{A} \in \mathbb{C}^{p \times n}$ computes f an upper bound on the Frobenius norm of $\boldsymbol{A}$ such that

$$f = \|\boldsymbol{A}\|_F + \gamma$$

where $0 \leq \gamma < \delta$, for a certain scalar absolute error bound δ, given in argument to the algorithm.

Interval Worst-Case Peak Gain

Given a state-space system $\mathcal{H} = ([\boldsymbol{A}], [\boldsymbol{B}], [\boldsymbol{C}], [\boldsymbol{D}])$, compute an approximation $\boldsymbol{S}$ on the WCPG

$$\langle\langle \mathcal{H} \rangle\rangle = |[\boldsymbol{D}]| + \sum_{k=0}^{\infty} \left| [\boldsymbol{C}][\boldsymbol{A}]^k [\boldsymbol{B}] \right|$$

such that two properties are ensured:

- bound property: $\langle\langle \mathcal{H} \rangle\rangle \leq \boldsymbol{S}$ element-by-element;
- if coefficients' radii $\to 0$ and precision $\to \infty$ then the exact $\langle\langle \mathcal{H} \rangle\rangle$ is contained in an ε neighborhood of the approximation $\boldsymbol{S}$ for an a priori given small $\varepsilon > 0$.

Interval Worst-Case Peak Gain

Computing the eigensystem of interval matrix

Eigenvalues of interval matrix

Compute enclosures $\boldsymbol{\lambda}^{\mathcal{I}}$ such that $\forall \boldsymbol{A} \in \boldsymbol{A}^{\mathcal{I}}, \boldsymbol{\lambda}(\boldsymbol{A}) \in \boldsymbol{\lambda}^{\mathcal{I}}$

Approach

Following the works of Xu and Rachid (1996) and Rohn(1998), use the Generalized Gershgorin's Circles theorem.

Eigenvectors of interval matrix

Given the enclosures on eigenvalues $\boldsymbol{\lambda}^{\mathcal{I}}$, compute enclosures $\boldsymbol{V}^{\mathcal{I}}$ such that $\forall \boldsymbol{\lambda} \in \boldsymbol{\lambda}^{\mathcal{I}}, \forall \boldsymbol{A} \in \boldsymbol{A}^{\mathcal{I}}$ if $\boldsymbol{A}\boldsymbol{\lambda} = \boldsymbol{A}\,\boldsymbol{V}$, then $\boldsymbol{V} \in \boldsymbol{V}^{\mathcal{I}}$.

Approach

Use Rump's theory of Verified Inclusions.