Introduction

Cette documentation a pour but de présenter l'utilisation du patch non_linearite_rationnelle.jmax sous jMax.

Ce patch est disponible ainsi que les objets non_linearite_rationnelle∼ et miditofreq, les templates Filtre_additionnel_midi et control_midi, les fichiers d'aide, dans l'archive compréssée `non_linearite_package.tar.gz`.

Décompression de l'archive

On se place dans le répertoire contenant tous les packages, généralement représenté par la variable d'envirronnement jMax: `jmaxPkgPath`

On décompresse l'archive avec la commande:

```
tar xvzf non_linearite_package.tar.gz
```

La décompression de l'archive crée le répertoire: `non_linearite`

La liste des fichiers crée est affiché lors de l'execution de la commande. Il existe aussi un fichier README, contenant cette liste, dans le répertoire non_linearite créé.

Configuration de l'environnement

Au cas on l'on pas décompréssée l'archive dans le répertoire contenant le package, il est nécessaire de repositionner la variable d'environnement de jMax, `jmaxPkgPath` avec le chemin d'accès au répertoire créé:

```
set jmaxPkgPath chemin_acces_repertoire_cree
```

Si cette variable est mal positionnée, lors de l'ouverture du patch les objets apparaitront en rouge car le package non_linearite n'aura pas été chargé en mémoire. Ce package est appellé par le fichier de configuration (.env) du patch.

Remarque :

Les objets ont été compilés avec la version 2.3.19 de jMax. Il est possible qu'il y ait des dysfonctionnement avec des autres versions de jMax.

Chapitre 1

Documentation du patch non linéarité

1.1 Contenu du patch

Le contenu de ce patch (Figure 1.3) correspond à la contruction d'un instrument sommaire. Il est composé d'une excitation, d'un filtre résonnant, d'un filtre passe-tout et d'une ligne à retard de longueur variable (Figure 1.1).
Il est possible de fixer les valeurs de certains paramètres par le biais d'un contrôleur MIDI.

Remarque:

La prise en compte des messages MIDI s'effectue par des objets externes à l'objet **non_linearite_rationnelle∼**.

Tous les paramètres de contrôle sont modifiables par le biais de sliders. L'utilisation de ces sliders implique celle de l'objet **scale**, livré avec la distribution de jMax, qui permet d'effectuer des remises à l'échelle. En effet, par défaut les sliders livrent une valeur entière compris dans l'intervalle $[0; 127]$, il faut donc faire une transformation, généralement linéaire, pour se conformer aux valeurs attendues par l'objet, généralement de type réel.

1.2 Objet non_linearite_rationnelle∼

Il prend comme arguments (les arguments sont optionnels):

- La valeur de la pente à l'origine de la fonction d'excitation α (float).
- La valeur de la pente des asymptotes de la fonction d'excitation β

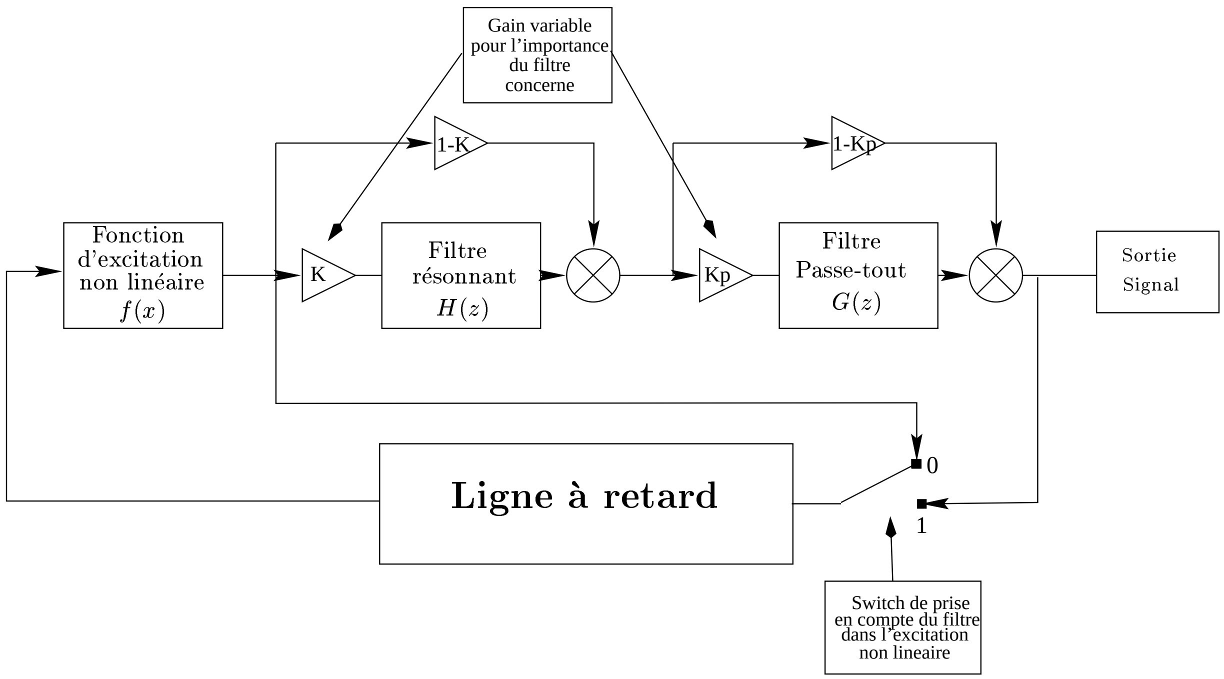

FIG. 1.1 – *Schéma fonctionnel du système*

(float).
- La valeur du coefficient de dissymétrie de la fonction d'excitation γ (float).
- La largeur de la bande passante en Hertz (float).
- La valeur de la fréquence de résonnance en Hertz (float).
- La taille maximale de la ligne de retard en nombre d'échantillons (int).
- La taille de la ligne de retard en nombre d'échantillons (int).

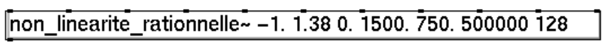

FIG. 1.2 – *Objet non_linearite_rationnelle~*

1.2.1 Limitation sur les valeurs des paramètres

Afin d'assurer la stabilité du système, les valeurs des différents paramètres sont limitées: α (pente à l'origine de la fonction non linéaire) doit être inférieur à 0.
β (pente des asymptotes de la fonction non linéaire) doit être supérieur à

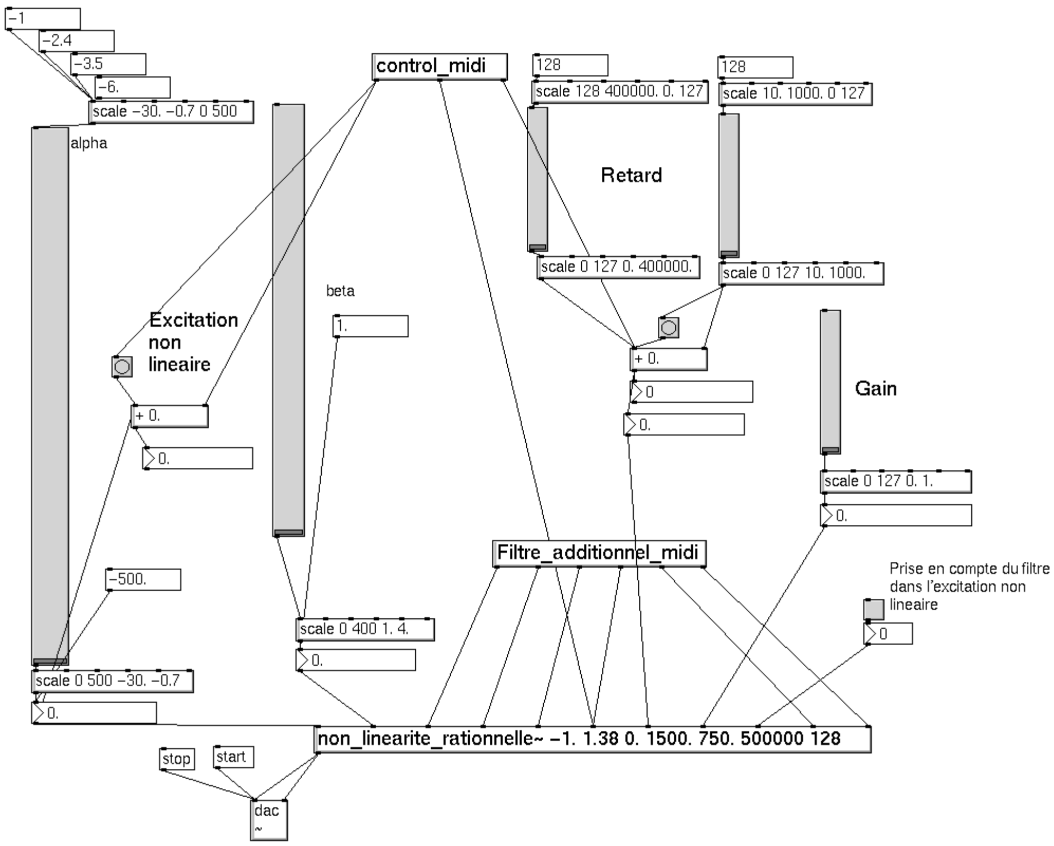

Fig. 1.3 – *Patch principal utilisant l'objet* **non_linearite_rationnelle~**

1.001 .
γ (facteur de dissymétrie) doit appartenir à l'intervalle: $[-2\sqrt{(1-\alpha)(\beta-1)}, 2\sqrt{(1-\alpha)(\beta-1)}]$
La bande passante du filtre résonnant doit être supérieure à 4Hz.
La fréquence de résonnance du filtre résonnant doit être supérieur à 1Hz.
Le paramètre du filtre passe-tout doit appartenir à l'intervalle: $[-1, 1]$.
La taille de la ligne à retard doit être supérieure à 10.
La taille maximale de la ligne à retard doit être supérieure à 10000.

Remarque:

> Un test sur la valeur des paramètres est effectué lors de chaque modification des dits paramètres. Si l'on se situe hors de l'intervalle valide, l'objet utilisera la valeur de la borne la plus proche comme nouvelle valeur de paramètre.

1.2.2 Descripition de la fonction d'excitation

La fonction d'excitation est la fonction non linéaire (Equation 1.1):

$$f(x) = \frac{x^3 + \gamma x^2 + \alpha x}{\beta x^2 + 1} \tag{1.1}$$

α Pente à l'origine.
β Pente des asymptotes.
γ Facteur de dissymétrie de la fonction.

L'oscillation commence lorsque le paramètre α de l'équation 1.1 est inférieur à -1. On entre en zone chaotique pour une valeur de ce même paramètre avoisinante de -6. Pour que le système soit stable, il est nécessaire que le coefficient β soit supérieur à 1.

1.2.3 Description des filtres

Nous utilisons un filtre résonnant et un filtre passe-tout.

Filtre résonnant

Le filtre résonnant est un filtre deux pôles, deux zéros:

$$H(z) = K\frac{c + dz^{-1} + ez^{-2}}{1 + az^{-1} + bz^{-2}} \tag{1.2}$$

Ces coefficients ne sont pas directement accessibles par l'utilisateur.Il peut modifier la valeur de la fréquence de résonnance et celle de la bande passante, le changement de ces valeurs entraînant le calcul des coefficients du filtre par l'objet lui-même. La modification des valeurs de ces paramètres s'effectue par les inlets 4 (fréquence de résonnance) et 3 (bande passante).

Filtre passe tout

C'est un filtre un pôle un zéro.

$$G(z) = \frac{a + z^{-1}}{1 + az^{-1}} \tag{1.3}$$

Nous pouvons constater que le système est stable lorsque a est compris dans l'intervalle $[-1; 1]$.
Le paramètre a est directement accessible par l'utilisateur par le biais de l'inlet 5.

"Importance" des filtres

Afin de pouvoir transformer le son, nous avons laissé la possibilité de modifier "l'importance" des filtres. Ce qui est appellé "importance" correspond à un gain situé avant l'entrée de chaque filtre. Il existe une branche parallèle au filtre de gain complémentaire à 1. Les gains sont bien sûr limités aux valeurs réelles comprises entre 0 et 1.

Le réglage de ces gains s'effectue par le biais des deux sliders situés en bas du template **"Filtre_additionnel_midi"** (Figure 1.4).

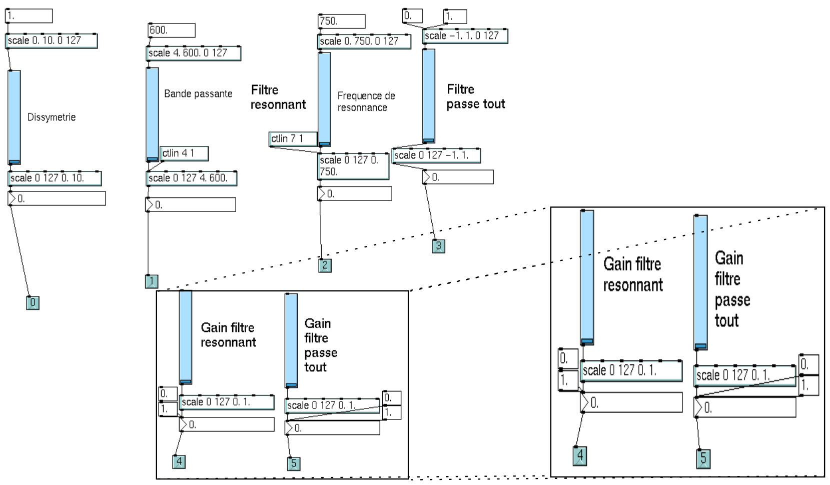

Fig. 1.4 – *Sliders de réglage de l' "importance" des filtres*

Remarque:

Si le switch de prise en compte de la sortie des filtres en entrée de la fonction d'excitation non linéaire (Figure 1.5) n'est pas activé, on ne discernera pas d'influence de la part du slider "**Gain filtre passe tout**" (Figure 1.4).

Bouclage du système

Il est possible, à l'utilisateur, de faire boucler le système avant ou après passage dans les filtres selon sa convenance. Ce choix se fait par le biais d'une case à cocher à l'intérieur du patch principal (Figure 1.5).
Le switch est considéré comme désactivé lorsque la valeur reçue sur l'inlet correspondante, l'inlet numéro 8, est nulle.

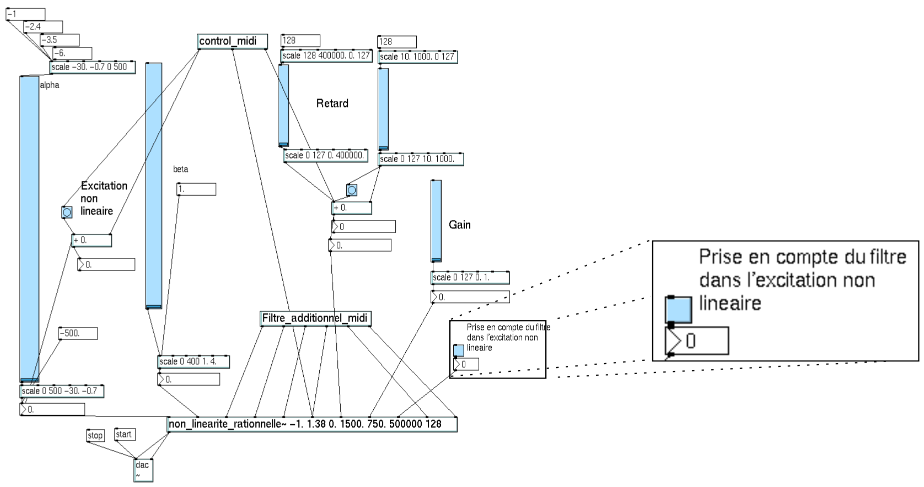

FIG. 1.5 – *Switch pour la prise en compte de la sortie des filtres en entrée de la fonction d'excitation non linéaire (Equation 1.1)*

1.2.4 Ligne à retard

Selon le type de la valeur reçue sur l'inlet correspondant à la longueur de retard, inlet numéro 6, l'objet utilisera une longueur de ligne à retard entière ou réelle. C'est à dire un nombre d'échantillons entier ou réel.

Lors de l'utilisation d'une ligne à retard réel, l'interpolation sur les valeurs se fait par la méthode de Lagrange. Une ligne à retard réel permet d'éviter les "clics" lors de variation de longueur de ligne à retard. Cependant l'utilisation de celle-ci entraîne une charge machine un peu plus importante, ainsi il semble intéressant de laisser la possibilité à l'utilisateur de choisir entre les avantages et désavantages de chacune des types de ligne à retard.

1.3 Gain global de l'instrument

Il existe un gain global de l'instrument situé à la droite du patch principal (Figure 1.6). Ce gain permet en particulier l'utilisation de la fonction d'excitation dans un régime très chaotique. En effet, si ce réglage du gain n'existait pas, on risquerait de saturer la sortie son de la machine pour une pente à l'origine α de la fonction d'excitation (Equation 1.1) trop importante.

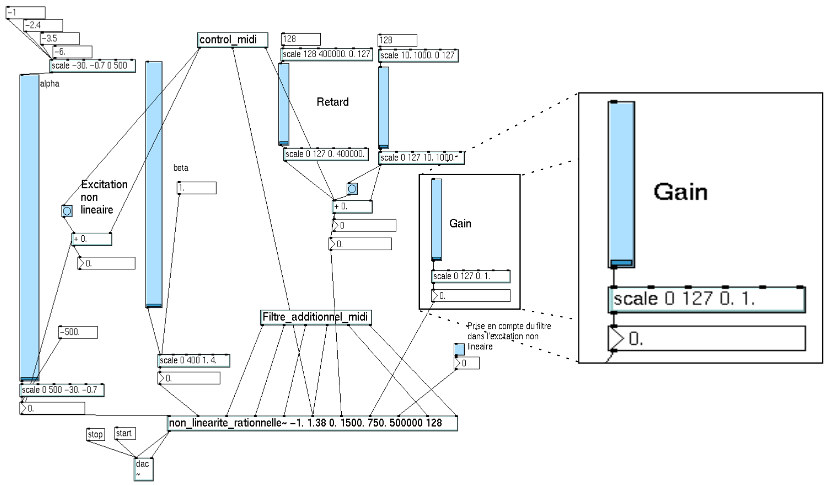

FIG. 1.6 – *Slider de réglage du gain global du système*

1.4 Template control_midi

Ce template (Figure 1.7) prend en compte certains contrôleurs midi pour permettre une plus grande jouabilité du patch. Il a été conçu sur la base de l'utilisation d'un saxophone MIDI de type **WX5**.

Sur cet instrument midi, il existe un capteur de pression qui mesure le souffle et envoie la valeur par le biais du contrôleur MIDI 2. On effectue alors une remise à l'échelle des valeurs possibles (on passe de l'écart [0;127] à [0;-10]). Cette remise à l'échelle correspond à notre souhait d'utiliser la pression du souffle comme paramètre α de notre fonction d'excitation non linéaire. Ainsi plus le souffle est fort, plus le volume sonore est important.

On utilise l'objet **bendin**, livré dans la distribution de jMax, pour prendre en compte la valeur du bender. Le bender est piloté par une molette située sous le contrôleur **WX5**. Après récupération de la valeur du bender, on remet celle-ci à l'échelle pour satisfaire notre souhait d'utiliser le bender comme contrôleur du filtre passe-tout. Il faut noter que l'objet **bendin** est relié à l'objet **miditofreq**. Une action sur le bender modifiera donc le paramètre du filtre passe tout et la longueur du tube.

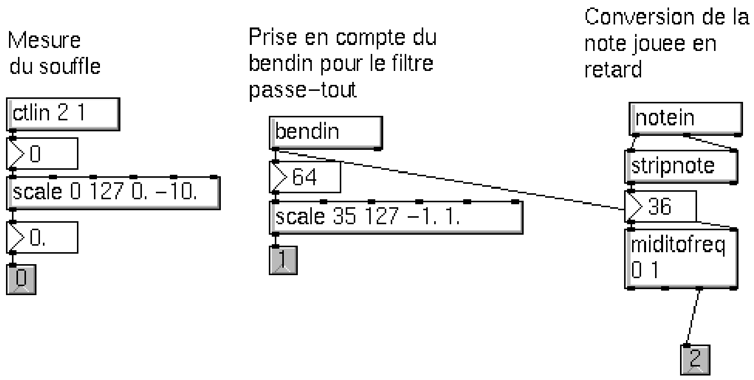

FIG. 1.7 – *Template utilisé pour le contrôle midi*

1.4.1 Objet miditofreq

Cet objet est l'implémentation sous jMax de l'objet de même nom, écrit par Geoffroy Peeters, sous Max/MSP lors de son stage DEA ATIAM.

Il permet de faire la conversion entre le numéro de la note lue sur le port midi et la taille du retard. Il faut remarquer que cet objet retourne la taille sous forme de valeur entière ou réelle. Ceci laisse le choix à l'utilisateur d'utiliser ou non l'interpolation de Lagrange. Ainsi s'il ne veut pas augmenter la charge de sa machine, il préférera se passer de l'interpolation.

On effectue ce choix en reliant l'entrée de la valeur de retard de l'objet **non_linearite_rationnelle∼** à la sortie entière ou réelle de l'objet **miditofreq**, de même pour les boîtes d'affichage des valeurs des sliders. Les sorties correspondantes aux valeurs entière et réelle sont les suivantes: l'outlet numéro 1 pour la valeur entière et l'outlet numéro 2 pour la valeur réelle.

1.5 Template Filtre_additionnel_midi

Ce template (Figure 1.8) comporte les possibilités de réglages des paramètres suivants:

- La dissymétrie de l'excitation non linéaire.
- Les valeurs des importances des filtres (cf schéma).

- La fréquence de résonnance et la bande passante du filtre résonnant.
- Le paramètre du filtre passe-tout.

Remarque:

Nous pouvons remarquer que ce template utilise deux contrôleurs MIDI pour affecter les valeurs de la fréquence de résonnance et celle de la bande passante du filtre résonnant. Ces contrôleurs correspondent, actuellement, à deux pédales branchées sur le clavier **YAMAHA DX7**.

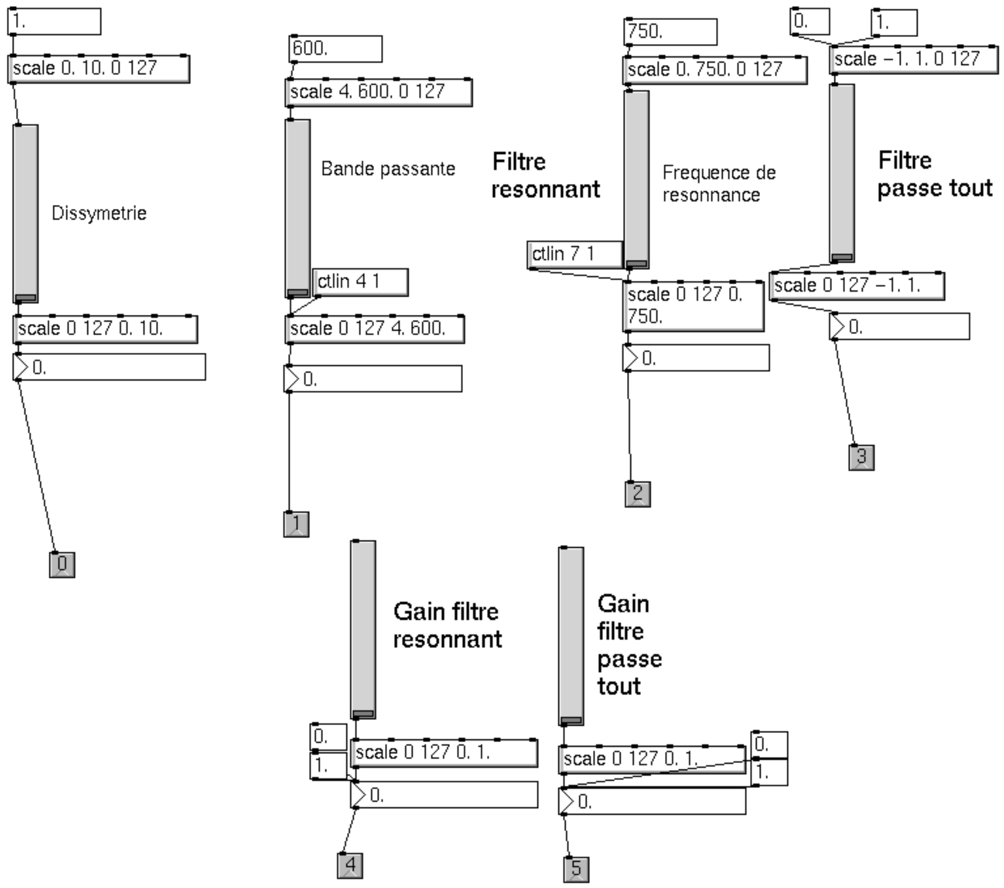

FIG. 1.8 – *Template utilisé pour le contrôle des paramètres des filtres*

1.6 Aides fournies avec le patch

Il est possible de consulter une aide sommaire sur l'objet **non_linearite_rationnelle∼**. Pour cela il suffit de cliquer avec le bouton du milieu sur l'objet. Une nouvelle fenêtre, créée par jMax, contient le patch d'aide suivant (Figure 1.9):

Objet
non_linearite_rationnelle~

Cet objet est compose d'une fonction non lineaire, de 2
filtres en cascade et d'une ligne de retard.
La fonction non lineaire est la fonction (X**3 + d X**2 +a X)/(1 + b
X**2). a est la pente a l'origine, b la pente des asymptotes de cette
fonction et d un parametre de dissymetrie.
Les 2 filtres en cascade sont : un filtre resonant (2 poles 2 zeros) et un filtre
passe−tout.
L' "importance" de ces filtres est parametrable par les inlets 9 et 10
Pour plus de details sur les inlets, voir dans
Help->Inlets&Outlets_non_linearite1 summary

Les arguments de l'objet sont :
- La pente a l'origine de l'excitation non lineaire (float)
- La pente des asymptotes de l'excitation non lineaire (float)
- La dissymetrie de l'excitation non lineaire (float)
- La valeur de la bande passante (float)
- La valeur de la frequence de resonance (float)
- La taille maximale de la ligne de retard (int)
- La taille de la ligne de retard (int)

FIG. 1.9 – *Patch d'aide de l'objet* **non_linearite_rationnelle∼**

Comme il l'est indiqué dans ce patch d'aide, il existe aussi une aide accessible depuis le menu **help** de jMax (Figure 1.10). Cette aide nous décrit les caractéristiques des inlets (Figure 1.11).

Help Patch
Inlets&Outlets_non_linearite1 summary
ISPW control summary
ISPW signal summary
Explode summary
Lists summary

FIG. 1.10 – *Menu* **help** *de jMax*

1.7 Action de la dissymétrie

Afin d'illustrer l'action de la dissymétrie sur le système, voici quelques signaux de sortie (Figures 1.12 à 1.15), pour différentes valeurs de α, β et le coefficient de dissymétrie γ. Ces exemples ont été réalisés avec un retard de 128 échantillons, en bouclant après la sortie des filtres. L' "importance" du filtre résonnant est de 1, celle du filtre passe tout de 0. La fréquence de résonnance du filtre résonnant est de 300 Hz, sa bande passante de 300 Hz.

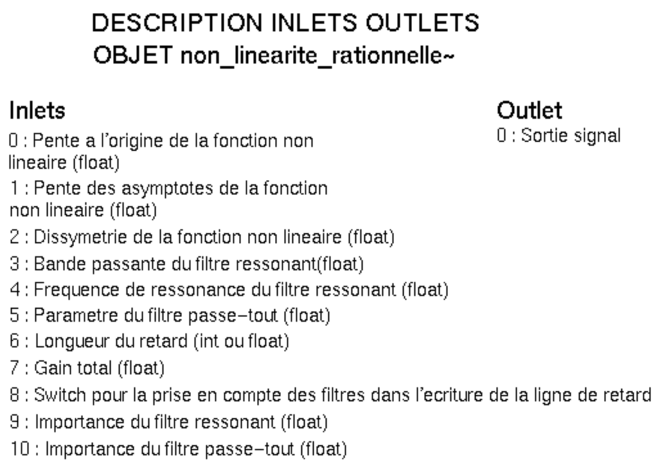

FIG. 1.11 – *Patch d'aide de l'objet* **non_linearite_rationnelle∼** *accessible depuis le menu* **help** *de jMax*

La visualisation du signal de sortie a été réalisée à l'aide de l'objet **graph_signal∼** qui utilise la librairie **X11**.

Récapitulatif des valeurs des paramètres:

- Longueur de la ligne à retard : 128 échantillons.
- Importance du filtre résonnance : 1.
- Importance du filtre passe tout : 0.
- Switch de prise en compte de la sortie des filtres en entrée de la fonction non linéaire: 1.
- Fréquence de résonnance : 300 Hz
- Bande passante : 300 Hz

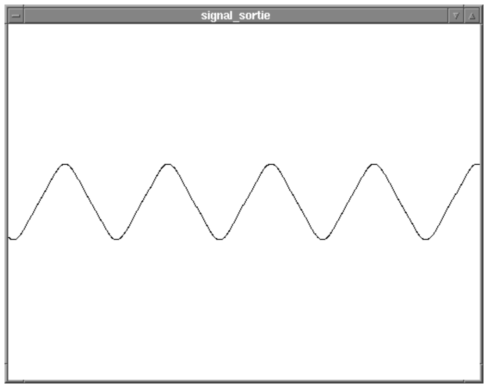

FIG. 1.12 – *Signal de sortie avec :* $\alpha = -2$, $\beta = 3$, $\gamma = 0$

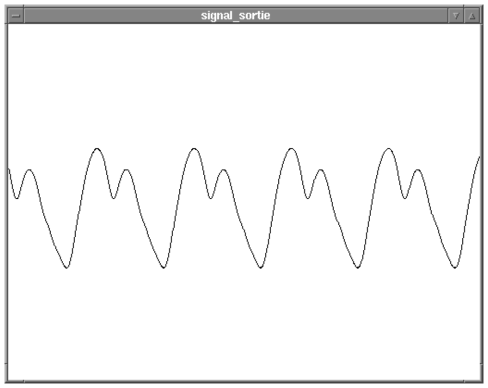

FIG. 1.13 – *Signal de sortie avec :* $\alpha = -2$, $\beta = 3$, $\gamma = 5$

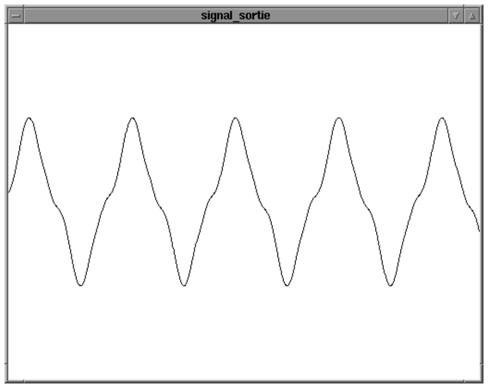

FIG. 1.14 – *Signal de sortie avec :* $\alpha = -4$, $\beta = 3$, $\gamma = 0$

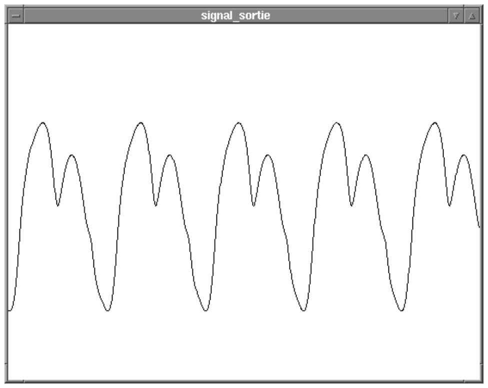

FIG. 1.15 – *Signal de sortie avec :* $\alpha = -4$, $\beta = 3$, $\gamma = 5$

Chapitre 2

Partie Développement

2.1 Fichiers de compilation

2.1.1 Makefile

Dans le Makefile, nous pouvons visualiser le nom des librairies créées lors de la compilation du package.

```
DISTDIR = /usr/LOCAL/max/fts
PACKAGEROOT = ..

include $(DISTDIR)/Makefiles/Makefile.lib
include Sources

MODULE_CFLAGS=
MORESOLIBS=

all: $(LIBDIR)/libnonlinearite.a $(LIBDIR)/libnonlinearite.so

$(LIBDIR)/libnonlinearite.a: $(OBJECTS)
$(AR) $(ARFLAGS) $(LIBDIR)/libnonlinearite.a $(OBJECTS)

TOCLEAN = $(LIBDIR)/libnonlinearite.a $(LIBDIR)/libnonlinearite.so

include $(DISTDIR)/Makefiles/Makefile.lib.end
```

2.1.2 Sources

Comporte la liste des fichiers à compiler pour pouvoir utiliser les objets du package.

```
SOURCES = \
non_linearite_fichier_config.c \
miditofreq.c \
non_linearite_fonction_auxiliaire.c \
non_linearite_rationnelle.c \
non_linearite_segment.c \
non_linearite_totale.c \
```

Remarque:

Dans ce package, nous avons définis quatre objets. Ceux-ci correspond aux fichiers sources: miditofreq.c, non_linearite_rationnelle.c, non_linearite_segment.c, non_linearite_segment.c .
Les noms des objets créés sont :
- miditofreq
- non_linearite_rationnelle~
- non_linearite_segment~
- non_linearite_totale~

Il est à noter que l'objet non_linearite_totale~ peut se comporter comme l'objet non_linearite_rationnelle~ ou non_linearite_segment~, selon la position d'un switch.

2.2 Fichiers à compiler

2.2.1 non_linearite_fichier_config.c

Ce fichier contient toutes les méthodes de configuration des objets du package non_linearite. C'est dans ce fichier que l'on trouve le nom du module, à savoir: **nonlinearite**.

```
#include "fts.h"

extern void test_config(void);

static void
nonlinearite_config_init(void)
{
  miditofreq_config();
  non_linearite_rationnelle_config();
```

```
  non_linearite_segment_config();
  non_linearite_config();
}

fts_module_t nonlinearite_module = {"nonlinearite", "nonlinearite synthesis classes",nonlinearite_config_init};
```

2.2.2 miditofreq.c

Ce fichier est le code de l'objet qui permet de faire la conversion d'une note rentrée par le port midi en une longueur de tube en nombre d'échantillons. il est à nter que ce nombre d'échantilons n'est pas forcément entier.

2.2.3 non_linearite_fonction_auxiliare.c

Regroupe toutes les fonctions auxiliaires communes aux trois objets de non linéarité. Plus particulièrement, il comporte les déclarations des méthodes sur lesquelles on est succeptible de pointer.

2.3 Système comportant une fonction d'excitation non linéaire

Les fichiers sources suivants permettent de créer des objets qui se comportent comme le système suivant (Figure 2.1)

2.3.1 non_linearite_rationnelle.c

La fonction non linéaire de cet objet est la suivante (Equation 2.1):

$$f(x) = \frac{x^3 + \gamma x^2 + x}{\beta x^2 + \alpha} \tag{2.1}$$

2.3.2 non_linearite_segment.c

La fonction non linéaire de cet objet est la suivante (Equation 2.4):

$$thr = 1 + \frac{1}{s_1} \tag{2.2}$$

$$excur = 1 - (s_2 thr) \tag{2.3}$$

$$f(x) = \begin{cases} excur + (s_2 x) & , \quad x < thr \\ s_1 - (x - 1) & , \quad x > thr \end{cases} \tag{2.4}$$

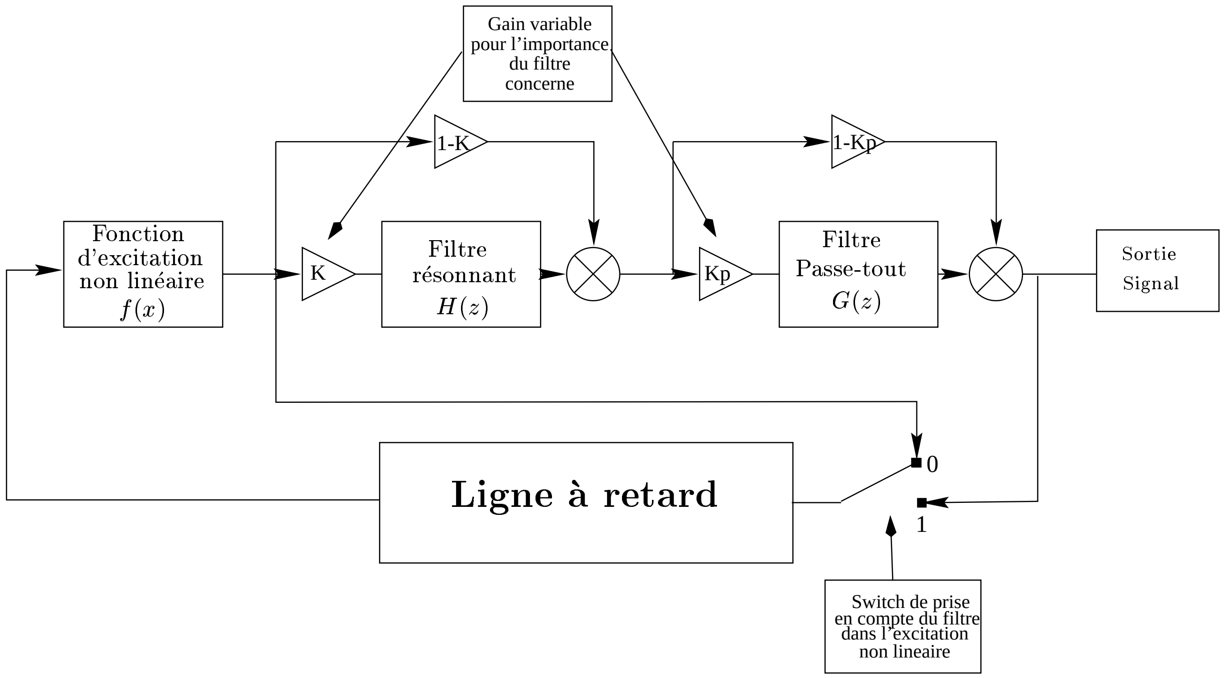

FIG. 2.1 – *Schéma fonctionnel du système, commun à tous les objets de type:* **non_linearite**

Les variables modifiables sont $s1$ et $s2$, elles correspondent aux pentes de deux segments utilisés pour la fonction non linéaire.

Cet objet ne permet pas de boucler à la sortie des filtres, le switch de prise en compte des filtres n'influe donc pas sur le comportement de l'objet.

2.3.3 non_linearite_totale.c

Cet objet regroupe le comportement des deux précédents en un seul objet. En envoyant un message sur la dernière inlet, on choisit la fonction d'excitation. Il est possible d'utiliser les deux fonctions précédentes (2.1 et 2.4) comme fonction d'excitation.

Chapitre 3

Descripition des fichiers inclus

3.1 prototype.h

Ce fichier regroupe tous les prototypes des fonctions auxiliaires utilisées dans les objets.
Le listage des prototypes dans ce fichier permet l'utilisation des pointeurs sur fonctions (cf 5.1.1) ce qui apporte une réduction du temps de calcul.

3.2 non_linearite_define.h

Regroupe toutes les définitions des constantes utlisées dans les objets de type **nonlinearite**. C'est dans ce fichier que l'on trouvera les valeurs minimales et maximales que les paramètres peuvent prendre.

3.3 non_linearite_description.h

Définit la structuredes objets de type **non_linearite**. Tous ces objets ont la même structure, la différence entre eux s'effectue lors de l'instantiation (il y a un inlet de plus pour l'objet **non_linearite_totale~** afin de pouvoir switcher entre la fonction non linéaire rationnelle (Equation 2.1) et celle par segment (Equation 2.4).

3.4 non_linearite_update.h

Regroupe toutes les fonctions dites d' "update" commune aux trois objets de type **non_linearite**.

Chapitre 4

Description des fichiers sources

4.1 Fichier non_linearite_auxiliaire.c

4.1.1 initialisation_interpolation

Cette fonction initialise la matrice permettant d'effectuer l'interpolation de Lagrange pour l'implémentation du retard fractionnaire.

4.1.2 ajout_bruit_constant

Ajoute un bruit constant.

4.1.3 ajout_bruit_aleatoire

Ajoute un bruit aléatoire.

4.1.4 ajout_bruit_nul

N'ajoute pas de bruit.

4.1.5 interpolation_linearite

Utiliser pour le retard entier.

4.1.6 interpolation_lagrange

Fonction permettant d'utiliser le retard fractionnaire. Est basée, comme son nom l'indique, sur la méthode de Lagrange.

4.1.7 action_dissymétrie

Obsolète.

4.1.8 action_dissymetrie_nulle

Obsolète.

4.1.9 coeff_filtre_additionnel

Calcule les coefficients du filtre résonnant selon les valeurs de la fréquence de résonnance, de la bande passante et de la fréquence d'échnatillonnage.
La fonction de transfert du filtre résonnant est la suivante (Equation 4.1):

$$H(z) = K\frac{c + dz^{-1} + ez^{-2}}{1 + az^{-1} + bz^{-2}} \tag{4.1}$$

4.1.10 nonlinear1

Fonction qui permet l'utilisation de la fonction d'excitation non linéaire rationelle (Equation 2.1).

4.1.11 nonlinear2

Fonction qui permet d'utiliser la fonction d'excitation non linéaire par segments (Equation 2.4).

4.1.12 fonction_filtre_resonnant_nulle

Evite de calculer la réponse du filtre résonnant lorsque le gain d' "importance" de ce filtre est nul.

4.1.13 fonction_filtre_passe_tout_nulle

Evite de calculer la réponse du filtre passe tout lorsque le gain d' "importance de ce filtre est nul.

4.1.14 fonction_filtre_resonnant1

Calcule la réponse du filtre résonnant lorsque la fonction non linéaire utilisée est la rationnelle (Equation 2.1).

4.1.15 fonction_filtre_resonnant2

Calcule la réponse du filtre résonnant lorsque la fonction non linéaire utilisée celle par segments (Equation 2.4).

4.1.16 filtre_passe_tout

Calcule la réponse du filtre passe tout.

4.1.17 ecriture_ligne_retard_non_linearite

Ecrit la sortie de la fonction non linéaire dans la ligne à retard lorsque le switch de prise en compte des filtres dans l'excitation non linéaire est positionnée à 0 (Figure 2.1).

4.1.18 ecriture_ligne_retard_filre

Ecrit la sortie des filtres dans la ligne à retard lorsque le switch de prise en compte des filtres dans l'excitation non linéaire est positionnée à 1 (Figure 2.1).

4.2 non_linearite_rationnelle.c, non_linearite_segment.c, non_linearite_totale,c

4.2.1 Méthode d'instanciation

Les objets non_linrearite_rationnelle~ et non_linearite_segment~ ont le même nombre d'inlets et outlets. L'objet non_linearite_totale~ a une inlet de plus. Cet inlet permet de switcher entre une fonction d'excitation non linéaire rationnelle et une par segment. La liste des méthodes définies dans ces objets est la suivante:

```
non_linearite_set_bande_passante
non_linearite_set_freq_res
non_linearite_ser_filtre_pass
non_linearite_set_retard
non_linearite_set_retard_fractionnaire
non_linearite_set_gain_filtre_res
non_linearite_set_gain_filtre_passe_tout
non_linearite_set_gain
non_linearite_set_ligne_retard
```

```
#Pour la fonction non lineaire de type rationnelle
non_linearite_set_alpha
non_linearite_set_beta
non_linearite_set_dissy

#Pour la fonction non lineaire par segment
non_lineaire_set_s1
non_lineaire_set_s2
```

Remarque:

Pour l'objet non_linearite_totale∼, les valeurs des paramètres des deux fonctions non linéaires sont modifiées par les mêmes inlets.

4.2.2 Méthode d'initialisation

Les objets peuvent être initialisé avec les paramètres rentrés par l'utilisateur. Nous testons donc la présence de ces paramètres dans cette méthode. Nous effectuons les allocations mémoires des pointeurs et affectons les pointeurs sur fonction. On initialise la matrice qui servira à effectuer l'interpolation selon la méthode de Lagrange. Et on insert l'objet DSP dans la liste dsp.

Remarque:

On peut constater que le pointeur d'écriture et le pointeur de lecture ne sont pas positionné en tête de zone mémoire allouée. Lorsque le pointeur d'écriture arrive en fin de cette zone mémoire, on recopie `taille_lecture` échantillons en début de zone mémoire et on positionne le pointeur à la suite de ceux-ci. Le fait de recopier ces échantillons permet d'éviter d'effectuer des tests lors de l'interpolation. En effet, lorsque le pointeur de lecture est repositionné à sa place initiale, on sait qu'il existe un nombre suffisant d'échantillons, le précédant, pour effectuer l'interpolation.

4.2.3 Méthode delete

On libère les espaces mémoires réservés lors de la méthode d'initialisation puis on enlève l'objet dsp de la liste dsp.

4.2.4 Méthode put

Dans cette méthode, on récupère la fréquence d'échantillonnage ce qui nous permet de calculer les coefficients du filtre résonnant. On appelle aussi la fonction dsp.

4.2.5 Traitement DSP

Au début de la fonction dsp, nous effectuons des test afin de calculer les nombres d'itérations nécessaires à amener les pointeurs d'écriture et de lecture, sur la ligne à retard, en fin de zone mémoire allouée. Ceci permet d'éviter d'effectuer deux tests à chaque passage dans la boucle (un pour chaque pointeur). Les résultats de ces tests positionnent des variables qui serviront qui serviront à décomposer le traitement en plusieurs boucles. Si le pointeur d'écriture arrive en fin de zone mémoire alloué, on recopie le nombre d'échantillons nécessaires à l'interpolation puis on repostionne le pointeur d'écriture à sa position initiale. Si le pointeur de lecture arrive en fin de zone mémoire, on le repositionne lui aussi à sa position initiale, sans effectuer de recopie de zone mémoire.

Les appels de fonctions sont les mêmes dans chaque boucles, on appelle les pointeurs sur fonction.

Test effectué en début de traitement

On teste la valeur de la variable `ecart`. Cette variable correspond à la variation de la longueur de tube. Ensuite selon le signe de cette variable, on teste les débordements possibles de la zone mémoire liée à la modification de la longueur de la ligne à retard. On vérifie que l'on pourra effectué l'interpolation (même si on ne fait pas d'interpolation, on prévoit la place nécessaire quand même). On vérifie aussi que l'on ne d'epasse pas la fin de la zone mémoire allouée (cas envisageable si l'on utilise une longueur de ligne à retard plus courte). Une fois ces tests effectués , on positionne le pointeur de lecture à sa nouvelle position.

4.3 miditofreq.c

4.3.1 Méthode instantiate

Les objets de la classe comportent 2 inlets et 4 outlets. Un des outlets est de type signal. Il permet de connaître le taux d'échantillonnage lors de l'appel de la méthode **put**. Les méthodes associées aux inlets sont:

`miditofreq_set_keyboards_input`
`miditofreq_set_bender_input`

Ces deux méthodes prennent en compte les messages de type entier.

4.3.2 Init

Dans cette méthode, on récupère les arguments passés à l'objet. Ceux-ci sont un paramètre pour régler la tonalité, et un pour la course du bender **A VERIFIER**. On ajoute aussi l'objet à la liste dsp, afin de pouvoir effectuer l'appel de la méthode **put**. Cet appel nous permet de connaître le taux d'échantillonnage et ainsi effectuer la conversion entre la fréquence d'une note et la longueur en nombre d'échantillons de la ligne à retard correspondante.

4.3.3 Méthode delete

On enlève cet objet de la liste DSP.

4.3.4 Méthode put

On récupère dans cette méthode le taux d'échantillonnage.

4.3.5 Méthode dîtes d' "update"

miditofreq_set_keyboards_input

Cette méthode récupère le numéro MIDI de la note jouée. Elle le convertit en fréquence puis en nombre d'échantillons. Le nombre d'échantillons est envoyé sur les outlets de type entier et réel. Ainsi en reliant l'un ou l'autre des outlets de sortie, on choisit en retard en nombre entier ou réel d'échantillons.

Méthode miditofreq_set_bender_input

Cette méthode permet de modifier la taille du retard selon les valeurs du bender. Le comportement est le même que la méthode précédente, seule la valeur récupérée par la méthode est différente. Ici on récupère la valeur du bender.

Chapitre 5

Index

5.1 Les variables

5.1.1 Les pointeurs sur fonction

AJOUT_BRUIT :

```
float (*AJOUT_BRUIT)(void)
```

Ce pointeur permet de choisir le type de bruit que l'on ajoute au signal entrant de la fonction non linéaire. Le bruit peut-être de type: constant, aléatoire ou nul.
Ce pointeur est positionnable sur les fonctions suivantes:

```
float ajout_bruit_constant(void);
float ajout_bruit_aleatoire(void);
float ajout_bruit_nul(void);
```

INTERPOLATION :

```
float (*INTERPOLATION)(fts_word_t *)
```

Ce pointeur permet de choisir le type d'interpolation que l'on désire effectuer sur la valeur lue sur la ligne à retard. On peut ne pas effectuer d'interpolation ou celle selon la méthode de Lagrange. On positionne ce pointeur lors de l'initialisation.

```
float interpolation_linearite(fts_word_t *);
float interpolation_lagrange(fts_word_t *);
```

DISSYMETRIE :

```
float (*DISSYMETRIE)(float, non_linearite_t *)
```

Ce pointeur permet de prendre en compte, ou non, la dissymétrie. Il peut pointer sur les fonctions suivantes:

```
float action_dissymetrie(float, non_linearite_t *);
float action_dissymetrie_nulle(float, non_linearite_t *);
```

NONLINEAR :

```
float (*NONLINEAR)(float, fts_word_t *)
```

Calcul la réponse de la fonction non linéaire.

```
float nonlinear1(float, fts_word_t *);
float nonlinear1bis(float, fts_word_t *);
float nonlinear2(float, fts_word_t *);
```

FILTRE_RESONNANT :

```
float (*FILTRE_RESONNANT)(float, non_linearite_t *)
```

Calcule la réponse du filtre résonnant selon le type de la non linéarité utilisée.

```
float fonction_filtre_resonnant_nulle(float, non_linearite_t *);
float fonction_filtre_resonnant1(float, non_linearite_t *);
float fonction_filtre_resonnant2(float, non_linearite_t *);
float fonction_filtre_passe_tout_nulle(float, non_linearite_t *);
float filtre_passe_tout(float, non_linearite_t *);
```

FILTRE_PASSE_TOUT :

```
float (*FILTRE_PASSE_TOUT)(float, non_linearite_t *)
```

Permet de calculer ou non la réponse du filtre passe tout.

```
float fonction_filtre_passe_tout_nulle(float, non_linearite_t *);
float filtre_passe_tout(float, non_linearite_t *);
```

ECRITURE_LIGNE_RETARD :

```
void (*ECRITURE_LIGNE_RETARD)(float, float, non_linearite_t *)
```

Permet de choisir la valeur à écrire dans la ligne à retard, soit celle de sortie de la fonction non linéaire, soit celle de sortie des filtres.
Peut pointer sur les fonctions suivantes:

```
void ecriture_ligne_retard_non_linearite(float, float, non_linearite_t *);
void ecriture_ligne_retard_filtre(float, float, non_linearite_t *);
void ecriture_ligne_retard_filtre1(float, float, non_linearite_t *);
void ecriture_ligne_retard_filtre2(float, float, non_linearite_t *);
```

5.1.2 Tableaux utilisés

q :

```
float **q
```

Ce tableau à deux entrées sert au stockage de la matrice utilisée pour effectuer l'interpolation de Lagrange.
L'initialisation de celui-ci, méthode `void initialisation_trompette (non_linearite_t *this)` s'effectue lors de la méthode d'initialisation de l'objet.

ck :

`float *ck`

Ce vecteur colonne permet de calculer le résultat de l'interpolation de Lagrange.

buffer :

`float *buffer`
`float *buffer_old`
`float *buf`

Ces pointeurs permettent de calculer les réponses du filtre résonnant et du filtre passe tout. Ils conservent les anciennes valeurs des entrées et sorties de ces filtres.

retard :

`float *retard`
`float *write`
`float *read`

Ce pointeur `retard` représente la ligne à retard. Nous utilisons deux pointeurs sur celui-ci, un pointeur d'écriture `write` et un pointeur de lecture `read`. L'utilisation de ces deux pointeurs supplémentaires peut paraître surprenant.

5.1.3 Liste des variables

Variables entières

Type de non linéarité :

`int type_non_linearite`

Ce flag permet de connaître la fonction non linéaire utilisée comme fonction d'excitation, le traitement dépendant de cette fonction. Par exemple, dans le cas de la fonction non linéaire par segments, le système est forcément bouclé avant le passage dans les filtres. Cette variable est bien sûr utilisé par l'objet **non_linearite_totale~**. Elle permet de positionner le pointeur sur fonction **NONLINEAR**.

Type d'interpolation :

`int type_interpolation`

Ce flag permet de connaître le type d'interpolation à effectuer sur la valeur de l'échantillon lut dans la ligne à retard. On effectue soit une interpolation selon la méthode de Lagrange, soit aucune interpolation. La position de cette variable implique celle du pointeur sur fonction **INTERPOLATION**.

Ordre de l'interpolation :

`int ordre_interpolation`

Cette variable est utilisée dans le cas de l'interpolation selon la méthode de Lagrange. Elle permet d'effectuer le produit matriciel qui nous donnera la valeur interpolée,

Taille maximale de la ligne à retard :

`int taille_max`

Comme son nom l'indique, elle représente la taille maximale de la ligne à retard. Elle évite de se positionner hors de la zone mémoire réservée lorsque l'on augmente la longueur de la ligne à retard.

Taille de la ligne à retard :

`int taille_retard`

Cette variable correspond à l'ancienne valeur de la longueur de la ligne à retard en nombre entier d'échantillons. Elle permet de connaître l'écart, la variation de position du pointeur de lecture en nombre d'échantillons.

Nombre d'échantillons à lire :

`int taille_lecture`

Cette variable représente le nombre de valeurs à lire pour effectuer l'interpolation. Il est important de la positionner correctement, pour éviter tout risque de sortie de la zone mémoire allouée pour la ligne à retard. Lorsque l'on replace le pointeur de lecture en début de zone mémoire, on le positionne à `taille_lecture` échantillons du début afin de pouvoir effectuer l'interpolation.

Variation du retard :

`int ecart`

Cette variable permet de connaître l'écart entre l'ancienne longueur de la ligne à retard et la nouvelle.

Choix du bruit :

`int noise`

Flag de prise en compte de la sortie des filtres :

`int sortie_filtre`

Variables réelles

Paramètres des fonctions non linéaires :

```
float alpha
float beta
float dissy
float s1
float s2
```

Ces cinq variables permettent de paramètrer la fonction non linéaire qui sert d'excitation au système. Les limites sur ces variables sont (les valeurs minimales et maximales sont définies dans le fichier **non_linearite_define.h**:
- alpha doit être inférieure à 0 .
- beta doit être supérieure à 1.001 .
- les valeurs limites de dissy dépendent de alpha et beta.
- s1 doit être inférieur à -0.01 .
- s2 doit être inférieur à 1 et supérieur à 0.01 .

Importance des filtres :

`float gain_non_linear`
`float gain_filtre_res`
`float gain_non_linear_filtre_res`
`float gain_filtre_passe_tout`

Ces quatres variables sont complémentaires deux à deux. Elles correspondent aux gains K, $1-K$, Kp et $1-Kp$ de la figure 1.1. Elles sont toutes les quatres limitées à l'intervalle de valeur $[-1;1]$.

Paramètres du filtre résonnant :

`float freq_res`
`float bande_passante`
`float freq_echan`

Les deux premières variables sont directement modifiables par l'utilisateur. Elles correspondent, comme leurs noms l'indiquent, à la fréquence de résonnance et la bande passante du filtre. La variable `freq_echan` correspond à la fréquence d'échantillonnage. Elle est affecté lors de la méthode **put**.
Ces trois variables sont utilisées par la fonction `coeff_filtre_additionnel` qui calculent en fonction de celles-ci les paramètres du filtres deux pôles deux zéros (Equation 4.1).

Paramètres de la fonction de transfert discrétisée du filtre résonnant :

`float K`
`float a`
`float b`
`float c`
`float d`
`float e`

Ces variables correspondent aux paramètres de la fonction de transfert du filtre deux pôles deux zéros considéré (Equation 4.1).

Paramètre du filtre passe tout :

`float ap`

Cette variable représente la valeur du paramètre du filtre passe tout. Elle est positionné par la méthode `non_linearite_set_filtre_pass`.

Variable de stockage pour les anciennes valeurs :

`float bufbis`

Cette variable permet de conserver l'ancienne valeur de sortie du filtre passe tout, ainsi il est possible d'effectuer la transformation de l'échantillon par ce filtre.