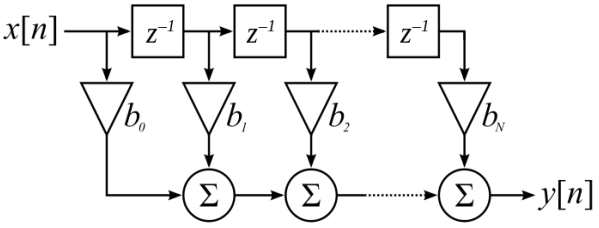
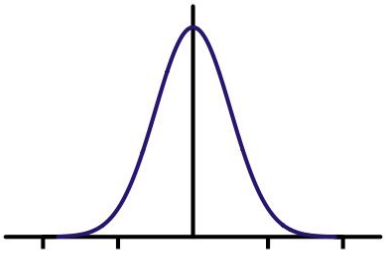
LEE IRINA **3407821**
FABRE WILLIAM **3410804**

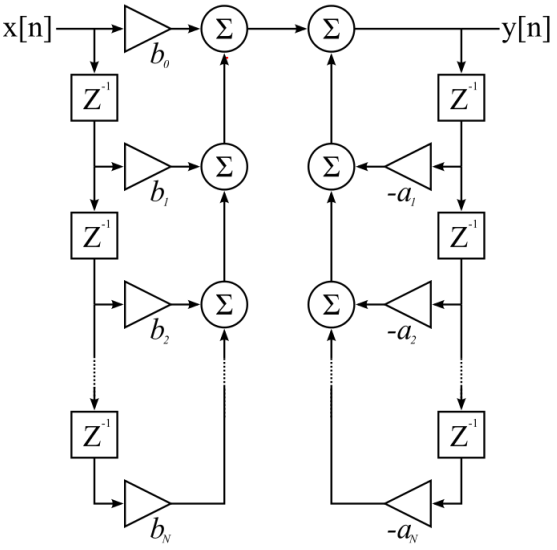
Compte Rendu du TP9 : Filtrage



Filtre FIR



Filtre de Gauss



Filtre IIR

Professeur : Lionel Lacassagne

Sorbonne Université
Faculté des Sciences et Ingénierie

Tables des matières

Objectifs 2

Exercice 1 : Filtrage non récursif 3

1. **Filtre `fir_average_i16`** 3
2. **Filtre `fir_average_q16`** 3
3. **Comparaison des 4 filtres** 4

Exercice 2 : Filtrage récursif 6

1. **Filtre `iir_q16`** 6
2. **Filtre `iir_q32`** 6

Tableau comparatif 7

Conclusion 8

Objectifs

- **Ecrire le code de différents types de filtres : FIR et IIR**
- **Comprendre la virgule fixe, la virgule flottante et le filtrage**
- **Observer et comparer les filtres entre eux**

Exercice 1 : Filtrage non récursif

1. Filtre `fir_average_i16`

Description :

Ce filtre FIR utilise des valeurs de type uint16 pour ses calculs et la division est réalisée avec une division entière.

La division entière est moins précise puisqu'au lieu d'avoir un résultat en flottant, le résultat y est tronqué car le dividende et le diviseur sont des entiers.

L'arrondi est fait dans la fonction flottante avec la fonction **lroundf** cependant cette fonction ne peut pas arrondir un entier dans le cas du `fir_average_i16`. Donc, pour résoudre cela, on a choisi d'arrondir le résultat dans le calcul de la variable **y** :

y = s / d devient y = (s + d/2) / d;

Conclusion :

Par rapport à un FIR utilisant des variables flottantes, ce FIR est moins précis à cause de la division où le résultat **y** n'est pas forcément la bonne valeur.
En revanche, le fait d'arrondir puis de tronquer ne différencie pas la fonction flottante de la fonction entière : les deux fonctions deviennent identiques.

2. Filtre `fir_average_q16`

Description :

Ce filtre FIR utilise des valeurs de type uint16 pour ses calculs et la division est réalisée avec une fraction équivalente. Ce filtre aura des variables en virgule fixe pour augmenter la précision.

Pour augmenter la précision, on a défini **lambda = (1 << q) / d** de telle sorte à avoir **y = s * ((1 << q) / d) >> q** ce qui revient à avoir **y = s / d** en virgule fixe. Mais lorsqu'on utilise ce principe, il y a une perte au niveau du **lambda** puisque la division est entière donc pour combler cette perte, on a arrondi en Q16 le résultat **y**, ce qui nous donne au final le calcul :

y = (s * lambda + (1 << (q-1))) >> q;

Conclusion :

Par rapport aux deux filtres précédents, ce filtre utilise la virgule fixe mais il est instable puisque la perte liée à la division peut ne pas être récupéré lors de l'arrondi du calcul. Le résultat peut être soit bon ou soit décalé de 1.

3. Comparaison des 4 filtres

On cherche à comparer les quatre filtres : `fir_average_f32`, `fir_average_i16`, `fir_average_q16` et `fir_gauss_f32`.

On prend un échantillon de plusieurs valeurs pour chaque paramètre à faire varier :

- **sigma_noise :** 5, 10
- **radius :** 2, 6 et 10
- **q :** 4, 8, 12 et 16
- **sigma_gauss :** 0.5, 1.0 et 1.5

Filtre `fir_average_f32` et `fir_average_i16` :

sigma_noise	radius	PSNR average_f32	PSNR average_I16
5	2	85,15	85,15
5	6	140,56	140,56
5	10	119,65	119,65
10	2	42,37	42,37
10	6	55,47	55,47
10	10	57,59	57,59

On observe que pour un bruit faible, une largeur du filtre faible convient mais pour un bruit élevé, une largeur du filtre élevée convient.

On en conclut que la largeur du filtre **radian** doit être proportionnel au niveau de bruit **sigma_noise** pour bien filtrer le bruit du signal.

Filtre `fir_average_q16` :

sigma_noise	radius	q	PSNR average_q16
5	6	4	18,73
5	6	8	37,83
5	6	12	140,56
5	6	16	140,56
10	10	4	12,03
10	10	8	43,34
10	10	12	57,49
10	10	16	55.47

On a vu dans le filtre précédent que pour un bruit faible, une largeur du filtre autour de 6 convenait et que pour un bruit haut, une largeur du filtre autour de 10 convenait.

En observant ces deux niveaux et en faisant varier **q**, on peut en conclure qu'une précision haute influe sur le PSNR et donne un meilleur filtrage.

En revanche, il y a une limite sur le nombre de bit de précision à cause du nombre de bits limité lors d'un calcul : **uint16**. Comme on peut le voir, pour un niveau de bruit élevé, le PSNR est élevé pour une précision **q** autour de 12 mais au delà, le PSNR diminue. Contrairement à cela, un niveau de bruit bas, le PSNR reste élever pour un nombre de bit plus large : autour de 12 à 16.

On en conclut que le PSNR peut atteindre sa valeur maximum sur un intervalle de nombre de bit de précision **q** variant en fonction du niveau de bruit **sigma_noise**. Un bruit faible a un intervalle de **q** plus grand alors qu'un bruit élevé à un intervalle de **q** plus petit.

Filtre `fir_gauss_f32` :

sigma_noise	sigma_gauss	PSNR GAUSS_F32
5	0.5	44,34
5	1.0	70,42
5	1.5	89,49
10	0.5	29
10	1.0	38,98
10	1.5	43,67

En général, si on utilise le filtre de Gauss pour atténuer le bruit d'un signal, il faut régler **sigma_gauss ≤ 1** mais seulement lorsque le bruit n'est de forme gaussienne, ce qui n'est le cas ici puisqu'on utilise le bruit blanc gaussien. C'est pour cela que lorsque **sigma_gauss** augment, le PSNR augmente.

On en conclut qu'un filtre de Gauss va augmenter le contraste de tout signal gaussien et réduire celui de tout signal qui ne l'est pas pour un **sigma_gauss ≤ 1**.

Exercice 2 : Filtrage récursif

1. Filtre `iir_q16`

Description:

Le filtre `iir_q16` est codé en virgule fixe et utilise des variables sint16.
Le résultat $\mathbf{y(n) = b_0x(n) + a_1y(n-1) + a_2y(n-2)}$ est alors codé en virgule fixe grâce aux coefficients qui sont multipliés par 2^q, le résultat devient alors : $\mathbf{y_0 = B_0x_0 + A_1y_1 + A_2y_2}$.

Pour obtenir le résultat en sint16, il faut arrondir le résultat et faire un décalage à droite de 2^q pour normaliser le résultat.

2. Filtre `iir_q32`

Description :

Le filtre `iir_q32` est une amélioration de la précision sur les échantillons x et y et il utilise un des variables sint32.

Soit Q un coefficient multiplicateur en fonction de la précision q et soient X0, Y0 et Y1 correspondent respectivement à x0, y0 et y1 multiplié par Q.

Le calcul du résultat du filtre IIR devient : $\mathbf{Y_0 = B_0X_0 + A_1Y_1 + A_2Y_2}$.

Étant donné qu'on décale à gauche de 2^q les coefficients multiplicateurs dans la formule du IIR, on doit décaler une première fois de 2^q à droite pour corriger le décalage de B_0, A_1 et A_2 et une deuxième fois par 2^q pour X_0, Y_0 et Y_1 pour corriger le coefficient vu plus haut.

Soit i allant de 1 à n (nombre d'échantillons), on obtiendra chaque valeur de y(i) qu'après le cast de y_0 en uint8.

Conclusion des deux filtres :

Les tests de la partie **tableau comparatif** nous amène à dire qu'il y a très peu de gain entre les deux versions pour 16 bits de plus en précision. Il ne vaut peut être pas le coup de passer sur le `iir_q32`.

Tableau comparatif

Méthode :

Pour faire ce tableau, nous avons utilisé les fonctions d'affichage du fichier test_filterNR.c donné dans l'énoncé. Nous l'avons modifié pour faire des tests en boucle sur plusieurs valeurs :

1. **sigma_noise** : bruit ajouté au signal.
2. **sigma_gauss** : écart-type de la gaussienne de paramètre (sigma, nu).
3. **q** : précision en nombre de bits.
4. **radius** : longueur du filtre
5. **alpha** : lissage

Nous avons fait varier :

- **alpha** entre 0.4 et 0.8 avec un pas de 0.2
- **q** entre 8 et 12 avec un pas de 2
- **radius** de 4 à 10 avec un pas de 4
- **sigma_gauss** de 0.5 à 1.5 avec un pas de 0.5
- **sigma_noise** de 5 à 15 avec un pas de 5

Nous avons affiché toutes ces valeurs et écrit tout dans un fichier texte avec la commande ./tp &> tmp.txt, qui va exécuter le programme et rediriger le flux de sortie standard dans un fichier texte.

Nous avons ensuite uploaded tout dans un fichier google excel et benchmark les algorithmes colonne par colonne en utilisant la moyenne AVERAGE, le minimum MAX, le maximum MIN, la déviation standard STANDEV.

Observations :

NOISE=5	FIR_F32	FIR_I16	FIR_Q16	G	IIR_F32	IIR_Q16	IIR_Q32
AVERAGE	132,225	132,225	94,93333	68,08333	91,94	99,93333	82,88333
MAX	151,82	151,82	112,63	89,49	100,12	123,68	88,97
MIN	112,63	112,63	58,73	44,34	81,05	73,68	76,91
STANDEV	19,778994	19,778994	17,837747	18,680088	8,092944	20,681616	4,7782123

NOISE=10	FIR_F32	FIR_I16	FIR_Q16	G	IIR_F32	IIR_Q16	IIR_Q32
AVERAGE	54,79	54,79	52,526666	37,216666	44,723333	43,803333	44,432222
MAX	60,47	60,47	57,81	43,67	47,92	47,18	47,77
MIN	49,11	49,11	43,84	29	41,58	40,18	40,75
STANDEV	5,7333345	5,7333345	5,6035554	6,1748613	2,6128752	2,8899396	2,8626354

NOISE=15	FIR_F32	FIR_I16	FIR_Q16	G	IIR_F32	IIR_Q16	IIR_Q32
AVERAGE	40,82	40,82	39,205	29,12333	34,166666	33,95	34,361111
MAX	44,6	44,6	41,71	33,46	36,39	35,81	36,71
MIN	37,04	37,04	36,11	23,9	31,93	31,76	31,91
STANDEV	3,8154937	3,8154937	2,5461296	3,9900092	1,8379089	1,6854695	1,9180440

Conclusion

Le filtre IIR a des avantages puisqu'il ne requiert que trois coefficients alors que le filtre FIR requiert autant de coefficients que la largeur du filtre. Donc, pour une largeur du filtre supérieur à 3, il est préférable d'utiliser le filtre IIR. Il faudra par contre faire attention, la sensibilité aux erreurs (arrondi et quantification) est beaucoup plus forte à cause de la récursion.

En revanche, le filtre FIR est plus stable car il est linéaire et possède un nombre fini de sorties alors que le filtre IIR est récursif et possède un nombre infini de sorties donc il est nettement moins stable.

On en conclut que les deux filtres ont leurs propres avantages et désavantages donc pour choisir un bon filtre, il faut voir les caractéristiques de chaque filtre pour avoir un filtre qui réduit au maximum le bruit du signal.

La virgule fixe permet d'utiliser seulement 16 bits lors de ses calculs contre 32 bits pour la virgule flottante, ce qui rend la virgule fixe moins gourmande en terme d'énergie et de temps de calcul.

Aujourd'hui le plus utilisé reste la virgule flottante grâce à la puissance et à la taille des processeurs mais le problème se pose lorsqu'on fait de l'embarqué et que la taille est extrêmement réduite et les calculs entre mantis exposant pourraient être gênants face à la simplicité de la virgule fixe, moins grande précision mais plus simple et plus compacte.