計測自動制御学会東北支部 第269回研究集会 (2011.12.12)
資料番号 269-5

固定小数点演算に基づく高速H_∞フィルタの実装に関する基礎的研究

Fundamental Study on Mounting Fast H_∞ Filter Based on Fixed-Point Arithmetic

○松塚 春男（岩手大・院），西山 清（岩手大）

○Haruo Matsuzuka, Kiyoshi Nishiyama

岩手大学, Iwate University

キーワード： 高速H_∞フィルタ (fast H_∞filter), 固定小数点数 (fixed-point number), Qフォーマット (Q format)

連絡先： 〒020-8551 盛岡市上田4–3–5 岩手大学 工学部 情報システム工学科 西山研究室
西山清，Tel.: (019)621-6475，Fax.: (019)621-6475，E-mail: nisiyama@cis.iwate-u.ac.jp

1. はじめに

通信や制御の分野においてシステム同定は広く用いられている。特に、エコーキャンセラは時変システム同定問題の典型的な応用例である。エコーキャンセラは適応フィルタを用いて構成されるが、この適応フィルタの基本的な性能はフィルタ係数を更新する適応アルゴリズムによって決まる。従来は、計算量が少なくアルゴリズムが単純なLMSアルゴリズムやその改良版のNLMSアルゴリズムが広く利用されてきた。しかし、収束性が非常に低く、追従性も十分でないため、携帯電話のようにエコー経路の変動が無視できない音響系の時変システムの同定には不十分であった。一方収束性が優れているRLSは計算量が$\boldsymbol{O}(N^2)$であり、タップ数Nと共に計算量が急激に増加してしまうため、実時間での処理が困難となる。RLSアルゴリズムを高速化したFTFや高速カルマンフィルタもあるが、忘却係数などのパラメータ調節が難しく、数値的な不安定さがしばしば指摘されて来た。

これらの従来の技術に対して、H_∞フィルタという新しい理論がある。まだ応用例は少ないが、カルマンフィルタと比べて外乱にロバストであることが知られている。

ところで、実数値の表現方法は、浮動小数点数と固定小数点数の2通りがあるが、実用的なデジタル信号処理システムでは、多くの場合固定小数点数を用いる。しかし、固定小数点数は表現できる数値の範囲が浮動小数点数に比べて非常に狭いため、浮動小数点演算でつくられたアルゴリズム固定小数点数では動かない場合も多い。この固定小数点演算への変換が可能かどうかで、そのアルゴリズムを実現できるかどうかが決まるため、浮動小数点数の固定小数点数化は非常に重要な課題である。

本研究では、高速H_∞フィルタのプログラムを固

定小数点数化し、シミュレーションを行い、正常に動作するかを検証する。

2. 高速H_∞フィルタ

ハイパーH_∞フィルタは$\hat{\boldsymbol{\Sigma}}_{k|k-1} \in \mathcal{R}^{N\times N}$の更新に$\boldsymbol{O}(N^2)$の計算量が必要である。しかし、観測行列$\boldsymbol{H}_k$が以下のようなシフト特性をもつとき、計算量が$\boldsymbol{O}(N)$となる高速アルゴリズムが存在する。

$$\boldsymbol{H}_k = [u_k, H_{k-1}(1), \cdots, H_{k-1}(N-1)]$$

ここで、u_kは時間kでの未知システムへの入力信号である。

[高速H_∞フィルタ]

高速H_∞フィルタは次のように計算量$\boldsymbol{O}(N)$で実行できる。

[Step 0]初期条件を設定する。

$$\begin{aligned}
\boldsymbol{K}_{-1} &= \boldsymbol{0}_{N\times 2}\ ,\ \ \boldsymbol{A}_{-1} = \boldsymbol{0}_{N\times 1}\ ,\ \ S_{-1} = \frac{1}{\varepsilon_0} \\
\boldsymbol{D}_{-1} &= \boldsymbol{0}_{N\times 1}\ ,\ \ \hat{\boldsymbol{x}}_{-1|-1} = \boldsymbol{0}_{N\times 1}
\end{aligned}$$

ここで、ε_0は大きな正の数であり、$\boldsymbol{0}_{N\times M}$は$N\times M$の零行列とする。

[Step 1]$\boldsymbol{A}_k$とS_kを決定する。

$$\begin{aligned}
\tilde{\boldsymbol{e}}_k &= \boldsymbol{c}_k + \boldsymbol{C}_{k-1}\boldsymbol{A}_{k-1} && \in \mathcal{R}^{2\times 1} \\
\boldsymbol{A}_k &= \boldsymbol{A}_{k-1} - \boldsymbol{K}_{k-1}\boldsymbol{W}\tilde{\boldsymbol{e}}_k && \in \mathcal{R}^{N\times 1} \\
\boldsymbol{e}_k &= \boldsymbol{c}_k + \boldsymbol{C}_{k-1}\boldsymbol{A}_k && \in \mathcal{R}^{2\times 1} \\
S_k &= \rho S_{k-1} + \boldsymbol{e}_k^T\boldsymbol{W}\tilde{\boldsymbol{e}}_k && \in \mathcal{R}
\end{aligned}$$

ここで、

$$\begin{aligned}
\boldsymbol{C}_k &= \begin{bmatrix} \boldsymbol{H}_k \\ \boldsymbol{H}_k \end{bmatrix} \in \mathcal{R}^{2\times N} \\
\boldsymbol{W} &= \rho\boldsymbol{R}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & -\gamma_f^{-2} \end{bmatrix} \in \mathcal{R}^{2\times 2}
\end{aligned}$$

また、$0 < \rho = 1 - \chi(\gamma_f) \leqq 1$と$\gamma_f > 1$という条件が存在する。

[Step 2]$\breve{\boldsymbol{K}}_k$を計算する。

$$\breve{\boldsymbol{K}}_k = \begin{bmatrix} S_k^{-1}\boldsymbol{e}_k^T \\ \boldsymbol{K}_{k-1} + \boldsymbol{A}_k S_k^{-1}\boldsymbol{e}_k^T \end{bmatrix} \in \mathcal{R}^{(N+1)\times 2}$$

[Step 3]$\breve{\boldsymbol{K}}_k$を分ける。

$$\breve{\boldsymbol{K}}_k = \begin{bmatrix} \boldsymbol{m}_k \\ \boldsymbol{\mu}_k \end{bmatrix}\ ,\ \ \boldsymbol{m}_k \in \mathcal{R}^{N\times 2}\ ,\ \boldsymbol{\mu}_k \in \mathcal{R}^{1\times 2}$$

[Step 4]$\boldsymbol{m}_k$と$\boldsymbol{\mu}_k$を用いて、次の時間kでのゲイン行列$\boldsymbol{K}_k$を決定する。それにはフィルタゲイン$\boldsymbol{K}_{s,k}$が含まれている。

$$\begin{aligned}
\boldsymbol{\eta}_k &= \boldsymbol{c}_{k-N} + \boldsymbol{C}_k\boldsymbol{D}_{k-1} && \in \mathcal{R}^{2\times 1} \\
\boldsymbol{D}_k &= [\boldsymbol{D}_{k-1} - \boldsymbol{m}_k\boldsymbol{W}\boldsymbol{\eta}_k][1 - \boldsymbol{\mu}_k\boldsymbol{W}\boldsymbol{\eta}_k]^{-1} && \in \mathcal{R}^{N\times 1} \\
\boldsymbol{K}_k &= \boldsymbol{m}_k - \boldsymbol{D}_k\boldsymbol{\mu}_k && \in \mathcal{R}^{N\times 2} \\
\tilde{\boldsymbol{K}}_k(i) &= \rho\boldsymbol{K}_k(i,1) \\
G_k &= \rho + \gamma_f^{-2}\boldsymbol{H}_k\tilde{\boldsymbol{K}}_k \\
\boldsymbol{K}_{s,k} &= G_k^{-1}\tilde{\boldsymbol{K}}_k && \in \mathcal{R}^{N\times 1}
\end{aligned}$$

[Step 5]フィルタ方程式を更新する。

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{K}_{s,k}(y_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_{k-1|k-1})$$

[Step 6]時間kを1増加させ、Step 1に戻る。

[スカラー存在条件]

高速H_∞フィルタの存在条件は次式となる。

$$-\varrho\hat{\Xi}_i + \rho\gamma_f^2 > 0\ ,\ \ i = 0, \cdots, k$$

ここで、$\varrho, \hat{\Xi}_i$は次のように定義される。

$$\varrho = 1 - \gamma_f^2\ ,\ \hat{\Xi}_i = \frac{\boldsymbol{H}_i\tilde{\boldsymbol{K}}_i}{1 - \boldsymbol{H}_i\tilde{\boldsymbol{K}}_i}$$

以上のように高速H_∞フィルタは、スカラー存在条件を含めて計算量$\boldsymbol{O}(N)$で実行できる。これらの式を整理すると、高速H_∞フィルタは以下のように表される。ただし、Nは状態ベクトルの次元(タップ数)である。

[高速H_∞フィルタ]

$$\begin{aligned}
\hat{\boldsymbol{x}}_{k|k} &= \hat{\boldsymbol{x}}_{k-1|k-1} + \boldsymbol{K}_{s,k}(y_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_{k-1|k-1}) \\
\boldsymbol{K}_{s,k} &= \frac{\boldsymbol{K}_k(:,1)}{1 + \gamma_f^{-2}\boldsymbol{H}_k\boldsymbol{K}_k(:,1)} && \in \mathcal{R}^{N\times 1}
\end{aligned}$$

ここで、ゲイン行列$\boldsymbol{K}_k$は次式で更新される。

$$
\begin{aligned}
\boldsymbol{K}_k &= \boldsymbol{m}_k - \boldsymbol{D}_k\boldsymbol{\mu}_k \quad \in \mathcal{R}^{N\times 2} \\
\boldsymbol{D}_k &= [\boldsymbol{D}_{k-1} - \boldsymbol{m}_k\boldsymbol{W}\boldsymbol{\eta}_k][1 - \boldsymbol{\mu}_k\boldsymbol{W}\boldsymbol{\eta}_k]^{-1} \\
\boldsymbol{\eta}_k &= \boldsymbol{c}_{k-N} + \boldsymbol{C}_k\boldsymbol{D}_{k-1} \\
\begin{bmatrix} \boldsymbol{m}_k \\ \boldsymbol{\mu}_k \end{bmatrix} &= \begin{bmatrix} S_k^{-1}\boldsymbol{e}_k^T \\ \boldsymbol{K}_{k-1} + \boldsymbol{A}_k S_k^{-1}\boldsymbol{e}_k^T \end{bmatrix} \\
\boldsymbol{m}_k &\in \mathcal{R}^{N\times 2}\ ,\ \boldsymbol{\mu}_k \in \mathcal{R}^{1\times 2} \\
S_k &= \rho S_{k-1} + \boldsymbol{e}_k^T\boldsymbol{W}\tilde{\boldsymbol{e}}_k\ ,\ \boldsymbol{e}_k = \boldsymbol{c}_k + \boldsymbol{C}_{k-1}\boldsymbol{A}_k \\
\boldsymbol{A}_k &= \boldsymbol{A}_{k-1} - \boldsymbol{K}_{k-1}\boldsymbol{W}\tilde{\boldsymbol{e}}_k \\
\tilde{\boldsymbol{e}}_k &= \boldsymbol{c}_k + \boldsymbol{C}_{k-1}\boldsymbol{A}_{k-1}
\end{aligned}
$$

ただし、$\boldsymbol{C}_k = \begin{bmatrix} \boldsymbol{H}_k \\ \boldsymbol{H}_k \end{bmatrix}$, $\boldsymbol{W} = \begin{bmatrix} 1 & 0 \\ 0 & -\gamma_f^{-2} \end{bmatrix}$であり、$\boldsymbol{c}_k \in \mathcal{R}^{2\times 1}$は$\boldsymbol{C}_k = [\boldsymbol{c}_k, \ldots, \boldsymbol{c}_{k-N+1}]$の第一列ベクトルである。ここで、$\boldsymbol{c}_{k-i} = \boldsymbol{0}_{2\times 1}$, $k - i < 0$と仮定した。また、$0 < \rho = 1 - \chi(\gamma_f) \leqq 1$, $\chi(1) = 1$, $\chi(\infty) = 0$, $\gamma_f > 1$とし、初期値は$\boldsymbol{K}_{-1} = \boldsymbol{0}_{N\times 2}$, $\boldsymbol{A}_{-1} = \boldsymbol{0}_{N\times 1}$, $S_{-1} = \frac{1}{\varepsilon_0}$, $\boldsymbol{D}_{-1} = \boldsymbol{0}_{N\times 1}$, $\hat{\boldsymbol{x}}_{-1|-1} = \boldsymbol{0}_{N\times 1}$と設定した。ここで、$\varepsilon_0$は十分に大きな正の値とし、$\boldsymbol{0}_{N\times M}$は$N \times M$零行列を表す。

また、与えられたγ_fに対する高速H_∞フィルタの存在条件は次式となる。

$$
-\varrho\hat{\Xi}_i + \rho\gamma_f^2 > 0\ ,\ i = 0, \ldots, k
$$

ここで、ϱ, $\hat{\Xi}_i$はそれぞれ次式で定義される。

$$
\varrho = 1 - \gamma_f^2\ ,\ \hat{\Xi}_i = \frac{\rho\boldsymbol{H}_i\boldsymbol{K}_{s,i}}{1 - \boldsymbol{H}_i\boldsymbol{K}_{s,i}}
$$

3. 固定小数点数とその演算

3.1 固定小数点数

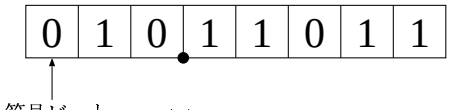
実数値を表現する方法は、浮動小数点数と固定小数点数の2通りがある。固定小数点数と比べると、浮動小数点数の表現できる値の範囲はとても広いため、通常計算機で数値計算を行う場合は、浮動小数点数を用いる。しかし、実用的なデジタル信号処理システムでは、多くの場合固定小数点数を用いる。その理由として、浮動小数点数と比べて、固定小数点数の方がハードウェアの規模が小さくてすみ、コストを軽減できることと、固定小数点数での演算の方が計算速度が速いことがあげられる。また、浮動小数点数より表現できる値の範囲が狭い点についても、扱う数値の範囲に注意すれば、固定小数点数でも動作することが多いことが知られている。

よって、浮動小数点演算でつくられたアルゴリズムもビット長の割り当てを工夫することにより、固定小数点演算へと変換することができる。この固定小数点演算への変換が可能かどうかで、そのアルゴリズムを実現できるかどうかが決まるため、浮動小数点数の固定小数点数化は非常に重要な課題である。

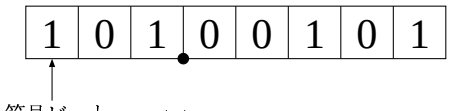
3.2 Qフォーマット

固定小数点数は、Qフォーマットと呼ばれる表記方法で表される。小数部のビット数をnビットとして固定小数点数を表現することを、「Qnフォーマットで表現する」という。

また、小数部だけでなく整数部のビット数も表記する場合は、Q(m.n)のように表記する。ここで、mは整数部のビット数であり、nは小数部のビット数である。整数部のビット数mには、符号ビットを含まない。このmとnを変化させることによって、固定小数点数の演算を実現する。



(a)2.84375



(b)-2.84375

Fig. 1　Q(2.5)の例

Q(2.5)(またはQ5)の例を図1に示す。図で最上位ビットは符号ビットであるため、ビット数には含まない。この例では、10進数整数値94が

$$1*2^{1}+0*2^{0}+1*2^{-1}+1*2^{-2}$$

$$+0*2^{-3}+1*2^{-4}+1*2^{-5}=2.84375$$

を表している。

本論文では、以下Qフォーマットは$Q(m.n)$として表記する。

次にこのQ(2.5)フォーマットで浮動小数点数を固定小数点数化する方法を示す。例として、円周率 π を固定小数点数化する。まず、Q(2.5)フォーマットの整数値'1'の値を求める。Q(2.5)での整数値'1'は、図2であり、これは、10進数整数値では32である。

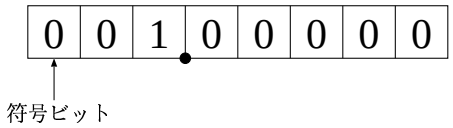


Fig. 2　Q(2.5)での整数値'1'

この32を求めたい浮動小数点数に掛けることにより、固定小数点数を求めることができる。つまり、

$$32*3.141592=100.530944\approx 101$$

と固定小数点数は101となる(図3)。これを小数に戻すと円周率πは

$$1*2^{1}+1*2^{0}+1*2^{-1}+0*2^{-2}+1*2^{-3}$$

$$+0*2^{-4}+1*2^{-5}=3.15625$$

となり、近似できていることがわかる。

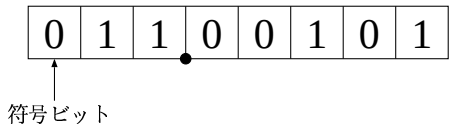


Fig. 3　Q(2.5)での円周率π

よって、浮動小数点数に$Q(m.n)$フォーマットの整数値'1'を掛けることにより、固定小数点数(10進数整数値)へと変換できることが確認できた。逆に、固定小数点数を$Q(m.n)$フォーマットの整数値'1'で割ることにより、固定小数点数から浮動小数点数を求めることができる。

先ほどの円周率πの例で言えば、固定小数点数'101'をQ(2.5)フォーマットの整数値'1'つまり'32'で割ることにより、

$$101/32=3.15625$$

浮動小数点数を求めることができる。

したがって、固定小数点数と浮動小数点数は図4のような関係がある。つまり、ビットの配列はそ

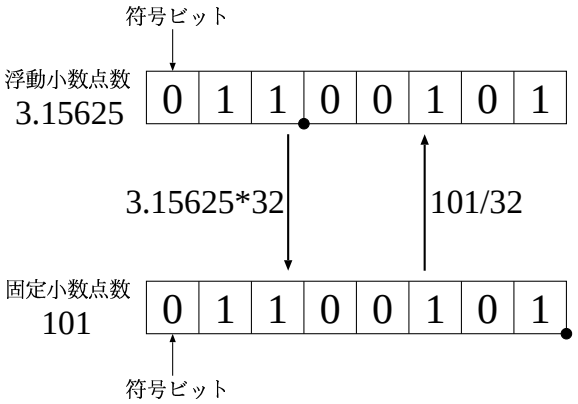


Fig. 4　浮動小数点数と固定小数点数の関係

のままで小数点の位置だけが移動していると考えることができる。

3.3 固定小数点数の四則演算

固定小数点数の四則演算の例を示す。

加減算については、ビット数に変化はないため、整数での演算結果をそのまま固定小数点数の演算結果としている。

整数ビット数をmビット、小数ビット数をnビットとして考えると、乗算の場合は、ビット数が2倍の大きさ、つまり$2(m+n)$ビットになる。このとき小数部と整数部はそれぞれ2倍となっている。そのため、まず小数部のビット数を合わせるために、nビット右にシフトして下位nビットを切り捨てる。この際、$n-1$ビットシフトした後に1を加えてから残りの1ビットシフトすることにより誤差を小さくしている(10進数でいうと四捨五入)。その後、下位$(m+n)$ビットを演算結果とすることで、整数部のビット数も合わせる。

除算の場合は、小数点以下nビットまでの商が整数として現れるように、被除数を前もって2^n倍する必要がある。当然$(m+n)$ビットに収まらないので、一時的に$2(m+n)$ビットとする。乗算と違い、除算では演算結果のビット数は被除数のビット数と一致するため、商も$2(m+n)$ビットとなる。これにより整数型のままで計算できるようになっている。

4. 高速H_∞フィルタの固定小数点数化

C言語により、単精度浮動小数点数(float)を用いてプログラミングされた高速H_∞フィルタ(FHF)のプログラムを固定小数点数化して動作を確認する。

プログラム内でのfloat型変数をlong long int型の64ビットの変数で定義し、固定小数点数で演算を行う。よって演算の途中では64ビットまで使用することができる。

パラメータの初期値は浮動小数点数で設定される。使用可能なビット数は、全ての固定小数点数で一律であり、整数部のビット数と小数部のビット数がそれぞれ設定されている。また、整数部のビット数には符号ビットを含んでいる。

四則演算については、各演算の処理をする関数を定義し、関数の呼び出しにより行う。演算結果がオーバーフローを起こした場合は、その値が正数であれば割り当てられたビット数で表現できる最大値を、負数であれば割り当てられたビット数で表現できる最小値を代入する。ここで、割り当てられたビット数をbビット(符号ビットを含む)としたときの上記の最大値と最小値の絶対値は、ともに$2^{(b-1)}-1$とした。実際には、負数の表現に2の補数を用いているため、最小値の絶対値は$2^{(b-1)}$となるが、今回は絶対値の大きさをそろえている。

指数の計算は、パラメータの初期値と同様に前もって浮動小数点数で計算し、固定小数点数に変換している。

以下に四則演算の処理の流れと具体例を示す。整数部のビット数mには符号ビットを含むが、$Q(m,n)$フォーマットについては符号ビットを除いたビット数を表記している。

4.1 四則演算

4.1.1 加減算(整数部mビット,小数部nビット)

固定小数点数を表す整数値について、そのまま加減算を行う。オーバーフローを起こしている場合は、割り当てられたビット数で表現出来る最大値(最小値)を格納する。

(例)整数部のビット数8ビット、小数部のビット数を24ビットとして$a+b$の加算結果をcとする場合

(i) 加算結果をcに格納する

```
c = a+b;
```

(ii)オーバーフロー時は32ビットで
表現できる最大値(最小値)をcに格納する

```
if(c>=2^31)        c=2^31-1;
```

```
else if(c<=-2^31) c=-(2^31-1);
```

4.1.2 乗算(整数部mビット,小数部nビット)

乗算結果を一時的に$2(m+n)$ビットで表す。本来の小数部ビット数に一致させるように乗算結果の小数部を調整する(乗算結果をnビット右シフトする)。オーバーフローを起こしている場合は、割り当てられたビット数で表現できる最大値(最小値)を格納する。

(例)整数部のビット数8ビット、小数部のビット数を24ビットとして$a*b$の乗算結果をcとする場合

(i) 乗算結果をcに格納する

```
c = a*b;
```

```
Q(7.24)*Q(7.24)
-> Q(1+(7+7).24+24) = Q(15.48)
```

(ii)cの小数点位置を合わせるために24ビット右シフト

```
c = c>>24;
```

```
Q(15+24.48-24) = Q(39.24)
```

(iii)オーバーフロー時は32ビットで
表現できる最大値(最小値)をcに格納する

```
if(c>=2^31)       c=2^31-1;
else if(c<=-2^31) c=-(2^31-1);
```

```
Q(39-32.24) = Q(7.24)
```

4.1.3 除算(整数部mビット,小数部nビット)

商が整数として現われるように被除数を2^n倍する。除算結果を一時的に$(m+2n)$ビットで表す。オーバーフローを起こしている場合は、割り当てられたビット数で表現できる最大値(最小値)を格納する。

(例)整数部のビット数8ビット、小数部のビット数を24ビットとしてa/bの除算結果をcとする場合

(i) aに2^24を掛ける計算をシフト演算子で実行

```
a = a<<24;
```

```
Q(7.24+24) = Q(7.48)
```

(ii)除算結果をcに格納する

```
c = a/b;
```

```
Q(7.48)/Q(7.24) -> Q(7+24.48-24) = Q(31.24)
```

(iii)オーバーフロー時は32ビットで
表現できる最大値(最小値)をcに格納する

```
if(c>=2^31)       c=2^31-1;
else if(c<=-2^31) c=-(2^31-1);
```

```
Q(31-24.24) = Q(7.24)
```

4.2 アルゴリズム全体の固定小数点数化

固定小数点数化したプログラムによるインパルス応答の推定値の結果を図5 に示す。データ長$L=8000$、タップ数$N=256$、$\gamma_f=28.0$としてプログラムを動かした。

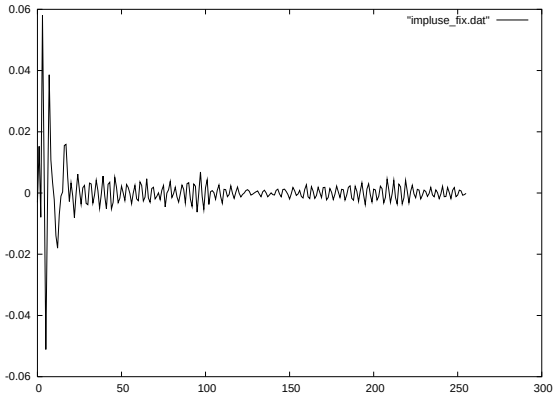
整数部のビット数が4ビット以下のときは、オーバーフローを起こし、γ_fの値が28.0ではなくなってしまう。そのため、整数部のビット数が5ビット以上の場合についてのみ考える。図5 より、小数部のビット数が少なくとも20ビット必要だということがわかる。また(c)より、整数部のビット数は5ビットあればよいことがわかる。

5. おわりに

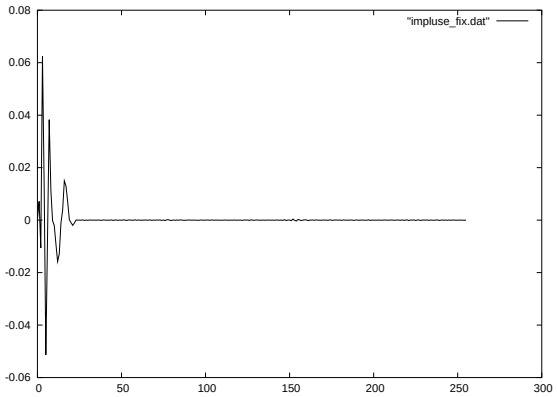
本研究では、高速H_∞フィルタのアルゴリズムの固定小数点数化に関して、具体的な演算方法を述べ、固定小数点数化したプログラムでシミュレーションを行い、その結果と単精度浮動小数点演算の場合とを比較した。結果として、固定小数点数化した高速H_∞フィルタのアルゴリズムが正常に動作することが確認できた。

さらに、ビット数による変化も調べ、小数部のビット数は最低20ビット必要であることがわかった。

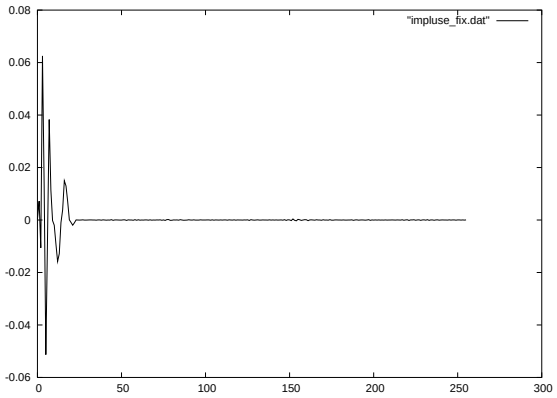
今回は、すべての変数に対して一律でビット数を割り当てたが、変数ごとに整数部と小数部のビット数を割り当てれば、数値を表現できる効率が上



(a) Q(13.18)



(b) Q(11.20)



(c) Q(5.26)

Fig. 5　32ビット固定小数点演算によるインパルス応答の推定値($L = 8000, \gamma_f = 28.0, N = 256$)

がるため、より少ないビット数でアルゴリズムが動作するかもしれない。このような工夫をして使用するビット数が最小で何ビット必要なのかを調べることが今後の課題となる。

参考文献

1) 勝俣 友紀: 「ハンズフリー携帯電話システムのためのH_∞エコーキャンセラに関する研究」, 博士論文, 岩手大学大学院, 2010.

2) 国沢 千寿: 「DSPを用いた信号波形の比較・検討」, 卒業研究論文, 高知工科大学, 2006.

3) 土井 伸洋: 「高位合成におけるデータパス最適化に関する研究」, 博士論文, 早稲田大学大学院, 2006.

4) 西山 清: 最適フィルタリング, 培風館, 2001.