ハイパフォーマンス
コンピューティング 93−16
（2003．3．12）

補間を用いたFFTの実装と評価

前田 崇† 黒田久泰†† 金田康正††

一般にFFT(高速フーリエ変換)のアルゴリズムは基底を2とする長さの場合に最も計算効率が良く，混合基底の長さの場合においても，良い性能が発揮できるFFTのアルゴリズムは数多く考案されている．しかし，長さが素数の場合は最も計算効率が悪く，計算効率が良い場合と比較して計算時間は極端に長くなる．このようなFFTの一般的性質を踏まえ，任意の個数の標本値から補間によって2のべき乗個の標本値を生成し，これに対して独自に実装したFFTで処理を行った．その結果，元の標本値に対して代表的なFFTライブラリを用いてFFTを施した場合と比較して多くの場合において計算時間が短縮された．更に適切なサンプリング周波数の下で補間方法を工夫することで，同じ出力結果を得ることができた．

Implementation of FFT by Interpolation

TAKASHI MAEDA,† HISAYASU KURODA†† and YASUMASA KANADA††

General FFT(Fast Fourier Transform) algorithms are most efficient when proceeding the radix-2 length data. Other FFT algorithms which proceed the mixed radix length data faster are also devised. However most FFT algorithms are least efficient when the length is a prime number. In this case, the calculation time is much more longer than the most efficient case. In this reserch, we implemented the original FFT program which always proceeds the radix-2 length data created from any length ones by the interpolation. As a result, our FFT program was able to shorten the calculation time more than some famous FFT libraries which proceeded the original length data. Moreover, if the data were sampled in the proper sampling frequency, we were able to obtain the same outputs by the divice of the interpolation method.

1. はじめに

線形変換の中でも特にフーリエ変換法 (Fourier Transform) は，理工学の分野でデータの解析などに広く使われている．一般的なFFTのアルゴリズムでは，基底を2とする長さのデータに対してFFTを施した時に計算量が$O(n\log n)$となり最も効率が良い．更に，任意の長さのデータに対してFFTを施したときにおいても，良い性能が発揮できるFFTのアルゴリズムはこれまで数多く考案されてきた．しかし，長さNが素数のデータに対してFFTを施す場合には，DFT(離散フーリエ変換)の定義式

$$H(n)=\sum_{k=0}^{N-1}h(k)e^{-2\pi ikn/N}(n=0,\ldots,N-1)(1)$$

をそのまま計算しなければならないため，計算量は$O(n^2)$となり最も効率が悪い[1]．このようなFFTの一般的な性質を踏まえて，任意の長さのデータを効率よく基底を2とする長さのデータに変換することができれば，計算時間を短縮できることが分かる．よって多倍長演算にFFTを応用する場合においては，元々のデータの末尾に0を挿入し，強制的に長さを2のべき乗の形式に変換してからFFTを施すといった手法がとられている．しかし，本来のFFTの適用範囲である周波数解析の場合は，この手法では観測されたデータと異なるデータにFFTを施すことになってしまう．そこで本研究では，任意の長さのデータから補間によって基底を2とする長さのデータを生成し，これに対してFFTを施す手法を提案する．

本稿では，まず代表的なFFTライブラリを用いた場合に計算時間がデータの長さによって大きく変動する事実を示す．そして，今回独自に実装したFFTの処理の詳細を説明し，処理結果の妥当性について検証した結果を示す．

2. データの長さが計算時間に与える影響

ここでは代表的なFFTライブラリを用いた場合

† 東京大学大学院新領域創成科学研究科
the University of Tokyo, department of frontier informatics
†† 東京大学情報基盤センター
the University of Tokyo, Information Technology Center

に，計算時間が処理するデータの長さにどの程度依存しているのかを示したい．今回 FFT ライブラリとして FFTW(FFT in the West)[2),3)] と GSL(the GNU Scientific Library)[4)] の一機能として提供されている FFT(以後 GSLFFT とする) を使用した．GSLFFT は FFTPACK のアルゴリズムをほぼ踏襲している．

実験に使用した計算機の仕様は表 1 の通りである．FFTW のバージョンは 2.1.3，GSLFFT のバージョンは 1.3 である．

表 1 使用した計算機の仕様

CPU	Intel Pentium IV 1.5 GHz
RAM	512 MB
Operating System	FreeBSD 4.5
C Compiler	gcc version 2.9.5.3
Floating-point Precision	double
Floating-point Size	8 Bytes

ここでは，長さ $N_0 = 8192(= 2^{13})$，及び $N_0 = 16384(= 2^{14})$ を基準として，それらの近傍 (長さを N として $-20 \leq N - N_0 \leq 20$ の範囲) において FFT を施した際の計算時間を測定した．各長さについては入力データとして 1 次元の実数値をランダムに発生させて FFT を施す処理を 10 回行い，FFT の部分に要した計算時間の平均値を測定値として採用した．

FFTW での測定結果を図 1 に，GSLFFT での測定結果を図 3 に示すが，どちらも長さによって計算時間が激しく変動していることが分かる．

また，図 1 の縦軸を対数表示した図 2 によると，$N_0 = 8192$ の近傍，$N_0 = 16384$ の近傍いずれの場合でも $N = N_0$ の場合に計算時間が最短になっており，更に $N = 8192$ と $N = 16384$ の近傍での計算時間を比較すると，後者における全 41 個の測定結果のうち，36 個もの測定結果が前者のそれよりも長くなっている．この特徴を詳しく調べるために，$8192 < N \leq 16384$ の場合の計算時間を測定し，そのうちいくつが $N = 16384$ の場合の計算時間よりもどの程度長くなるかを調べた．その結果を表 2 に示す．$N = 16384$ の場合の計算時間 (10 回繰り返した場合の平均値) t_0 は 4.425ms，この範囲で最も計算時間を要したのは $N = 16361$ の場合の 6.818s であり，t_0 の 1540 倍にもなった．

表 2 より，$8192 < N \leq 16384$ の場合の 95%以上で $N = 16384$ の場合よりも計算時間が長くなってしまうことが分かる．

以上の結果は次のようにまとめられる．

- FFT の計算時間は，処理するデータの長さに大きく依存する．
- 長さ N の FFT の計算時間は，多くの場合において，N 以上の 2 のべき乗のうち最小の数を長さとする FFT の計算時間よりも長くなる．

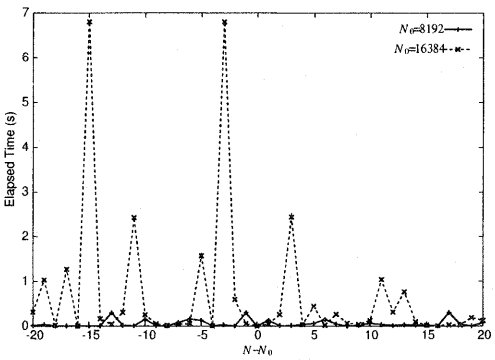

図 1 $N_0 = 8192$(実線) と $N_0 = 16384$(点線) の各近傍における FFTW 計算時間の推移 (横軸：標本数 N，縦軸：経過時間 (s))

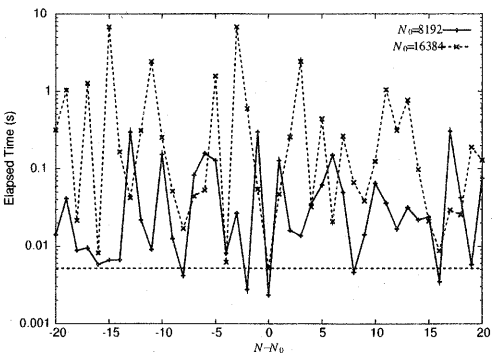

図 2 $N_0 = 8192$(実線) と $N_0 = 16384$(点線) の各近傍における FFTW 計算時間の推移 (横軸：標本数 N，縦軸：経過時間 (s)(対数表示))，水平に引かれた点線は $N_0 = 16384$ の場合の測定結果を示す．

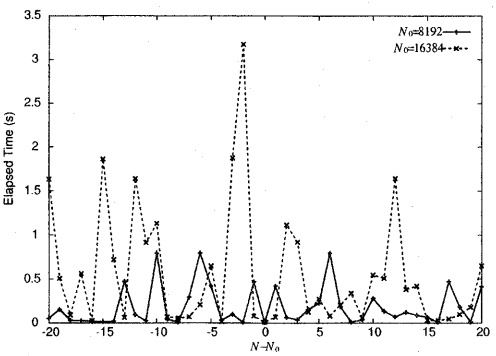

図 3 $N_0 = 8192$(実線) と $N_0 = 16384$(点線) の各近傍における GSLFFT 計算時間の推移 (横軸：標本数 N，縦軸：経過時間 (s))

表 2　$N = 16384$ での FFT 計算時間 (t_0) に対する $8192 < N \leq 16384$ での FFT 計算時間 (t)

計算時間比	個数	構成比
$\frac{t}{t_0} < 1$	323 個	4.0%
$1 \leq \frac{t}{t_0} < 10$	3714 個	45.3%
$10 \leq \frac{t}{t_0} < 10^2$	2720 個	33.2%
$10^2 \leq \frac{t}{t_0} < 10^3$	1329 個	16.2%
$10^3 \leq \frac{t}{t_0}$	106 個	1.3%
合計	8192 個	100%

3. 補間を用いた FFT の実装

長さ N に対して N 以上の 2 のべき乗のうち最小の数を新しい長さ N_1 とする．前節の結果を踏まえると，長さ N のデータを長さ N_1 のデータに効率よく変換することができれば，長さ N_1 のデータに FFT を施すことで実質的に長さ N の FFT をより高速に計算できることが考えられる．

3.1 補間によるデータの変換

いかにして長さ N のデータを長さ N_1 のデータに変換するのかということであるが，ここでは各データを 1 次元の実数値として 3 次スプライン補間で N 個のデータを全て通るような連続関数を推定し，これを N_1 個に等分割して新しいデータを取得した．尚，ここで適用した 3 次スプライン補間は，N 個のデータの両方の端点で 2 階導関数の値が 0，すなわち自然 3 次スプライン条件を満たすとした[5),6)]．

3.2 FFT の実装

補間の後で独自に実装した FFT で処理を行った．この FFT ルーチンは基底を 2 とする長さのデータのみ処理できれば十分である．よって単純に Cooley-Tukey 型アルゴリズムを適用して実装した[7)]．長さ N_1 の 1 次元実数値データを FFT するとパワースペクトルとして $N_1/2$ 個の成分が出力されるが，このうち低い周波数成分から $N/2$ 個の成分を出力とした．

全体の処理の流れを示すと図 4 のようになる．

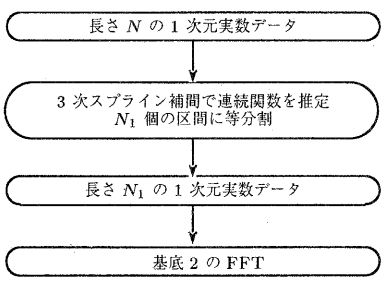

図 4　実装した FFT の処理の流れ

4. 補間を用いた FFT の評価

実装した補間 FFT(以後 IFFT とする) の動作検証に用いた計算機の仕様は 2 節で示した表 1 の通りである．動作検証ではある長さのデータに対して FFTW で計算を行った場合との間で計算時間，出力結果を比較した．

4.1 計算時間短縮への寄与

まず，長さを N として $8192 < N \leq 16384$ の範囲内で，FFTW を適用した場合と IFFT を適用した場合で計算時間がどの程度変化するかを調べた．IFFT ではこの範囲の長さは全て補間によって 16384 に変換される．各長さについて入力とする 1 次元実数データはランダムに生成し，計算時間の測定値は 10 回計算を行った各計算時間の平均値とした．

$8192 < N \leq 16384$ における FFTW の計算時間 (τ_0) と IFFT の計算時間 (τ) 比の確率密度分布を図 5 に，τ_0 と τ の推移を図 6 に示す．

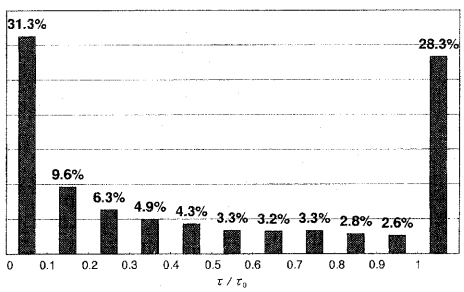

図 5　$8192 < N \leq 16384$ における FFTW の計算時間 (τ_0) と IFFT の計算時間 (τ) 比 (横軸：τ/τ_0，縦軸：確率密度)

図 5 によると $8192 < N \leq 16384$ の範囲での 8192 回の FFT 処理のうち実に 71.7%で計算時間が短縮され，31.3%で計算時間は 1/10 以下に短縮された．尚，IFFT の性能が FFTW の性能に比べて最も良くなるのは $N = 15227$ の場合で

$$\frac{\tau}{\tau_0} = \frac{0.017961\text{s}}{7.139274\text{s}} \approx 2.5 \times 10^{-3}$$

逆に最も悪くなるのは $N = 10240$ の場合で

$$\frac{\tau}{\tau_0} = \frac{0.032835\text{s}}{0.003094\text{s}} \approx 1.1 \times 10$$

であった．

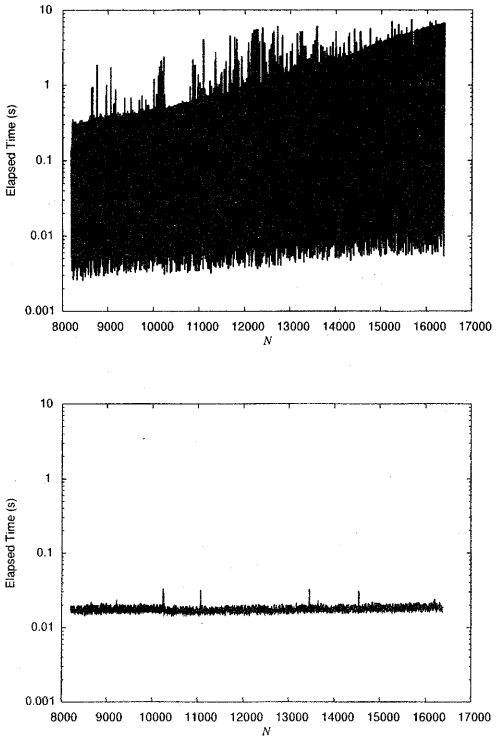

図 6 $8192 < N \leq 16384$ における FFTW の計算時間の推移 (上段), IFFT の計算時間の推移 (下段)(横軸：N, 縦軸：計算時間 (対数表示))

また，図 6 によると FFTW では計算時間が長さに大きく依存し，計算時間は最も速い場合と遅い場合の間で 1000 倍程度の違いがあるのに対し，IFFT では計算時間は長さによらずほぼ安定している．これは IFFT での計算時間の大半が補間後の長さ 16384 のデータに FFT を施すために要した時間であり，3 次スプライン補間の計算時間がデータの長さに依存していないことを示しているといえる．

4.2 出力結果の妥当性

出力結果の妥当性を検証するにあたり，まず FFTW によるパワースペクトルの各成分と IFFT によるそれとの差の 2 乗平均を誤差 ε として定義した．

まず 4.1 節の各長さで IFFT の計算時間が FFTW のそれと比べて最も短縮された $N = 15227$ のパワースペクトルの出力結果を比較した．誤差は $\varepsilon \approx 4.1 \times 10^{-2}$ と大きな値になった．

FFTW によるパワースペクトルの各成分と IFFT によるそれの差の推移を図 7 に示す．FFTW，IFFT いずれの場合もパワースペクトルの各成分を系列内

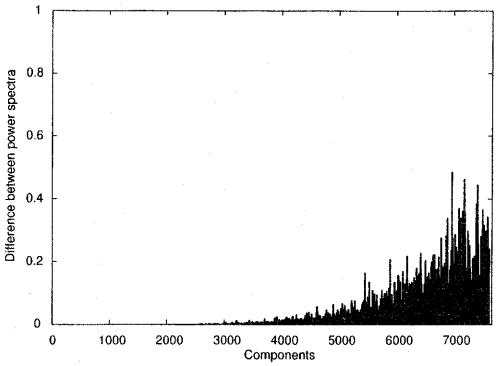

図 7 $N = 15227$ における FFTW，IFFT によるパワースペクトルの各成分の差の推移 (横軸：成分，縦軸：FFTW によるパワースペクトルから IFFT によるそれを引いた差)

の最大値で正規化したため，図 7 では縦軸の最大値は 1 になっているが，高い周波数成分ほど FFTW と IFFT の出力結果は大きく異なっている．この現象の原因を考察する．ここでは入力としてランダムに生成したデータを使用した．このため入力データと標本化間隔の間でナイキスト条件が満たされている保証は全くなく，隣接するデータには何の相関もない．すなわち，FFT を計算することはできるものの，入力の物理的意味を考えると標本化そのものが正しく行われていないと解釈できる．従ってこのような入力データに対して FFTW と IFFT の出力の比較を行ってもそれには物理的な意味はないことになる．

そこで，適切なサンプリング周波数で標本化された実際のデータを用いて FFTW と IFFT の出力比較を行うことにした．ここでは性質の異なる 3 種類の音波を使用した．

表 3 FFTW と IFFT の出力比較に使用した音波

音色名	サンプリング周波数	量子化ビット数	標本点数	補間後の標本点数
(A) ピアノ	23810 Hz	16	25528	32768
(B) ベル	15625 Hz	8	6658	8192
(C) シンバル	31250 Hz	8	48050	65536

一般的に楽器の波形は音程を決める基本振動と，音色を決める倍振動によって構成されている．各々の特徴は以下のように挙げられる．

(A) ピアノ
基本振動が最も強く，整数次倍振動を含む．高次の整数次倍振動程弱くなる傾向にある．

(B) ベル
倍振動の方が基本振動よりも強くなる場合や，非整数次倍振動を含む場合がある．

(C) シンバル
基本振動，倍振動の区別がつかず，任意の振動を不規則に含む．

実験ではこの3つの音波に対して以下の3つの手法でFFTを施した場合の計算時間の測定，並びに出力結果の比較を行った．

① FFTWを施した場合

② (3次スプライン補間を適用した)IFFT

③ 線形補間を適用したIFFT
(②と区別するために以後IFFT-Lとする．)

まず計算時間の測定結果を表4に示す．

表4 計算時間の測定結果

音色名	① FFW	② IFFT	③ IFFT-L
(A) ピアノ	0.382613 s	0.106646 s	0.106646 s
(B) ベル	0.111002 s	0.005944 s	0.005944 s
(C) シンバル	0.04076 s	0.247923 s	0.247923 s

IFFTを施した場合はこれまでの実験結果と同様に，FFTWを施した結果と比較して(A)では72%，(B)では46%の計算時間を減少させることができたが，(C)では逆に計算時間が増加した．また，補間法の違いによる計算時間短縮の効果は得られなかった．これは線形補間も3次スプライン補間も理論的な計算量が$O(n)$で同じであるためと考えられる．加えて，線形補間を用いると後に示すように出力結果に問題が生じる．

次に誤差の測定結果を表5に示す．ここでも誤差εはFFTWによるパワースペクトルの各成分とIFFT，およびIFFT-Lによるそれとの差の2乗平均とした．

表5 誤差の測定結果

音色名	② IFFT	③ IFFT-L
(A) ピアノ	2.8×10^{-7}	3.0×10^{-6}
(B) ベル	1.3×10^{-5}	3.8×10^{-5}
(C) シンバル	4.7×10^{-4}	3.5×10^{-3}

誤差の値はランダムな入力データの場合と比べてはるかに低く抑えられ，IFFTとIFFT-Lの誤差を比較すると，後者の方が前者よりも誤差が大きくなった．これは，線形補間では補間によって得られた新しい標本値と元々の標本値が1本の滑らかな曲線上に乗るとは限らないため，補間後のデータが元のデータと異なるものになってしまったためだと考えられる．よって補間法としては線形補間よりも3次スプライン補間の方が優れているといえる．

以上を踏まえて，(B)と(C)の場合についてFFTWとIFFTの間の誤差の分布を調べた．尚，(A)では誤差の値が示すようにFFTWの出力結果とIFFTのそれはほぼ同じになった．(B)，(C)についてFFTWによるパワースペクトルの各成分とIFFTによるそれの差の推移を図8，図10に示す．参考のため，IFFTによる出力結果を図9，図11に示す．尚，全てのパワースペクトルの値は各系列の最大値で正規化している．図9，図11より，(B)は基本振動よりも倍振動の方が強く，(C)は不規則に任意の周波数成分を含んでいることが分かり，(B)，(C)ともそれぞれの音波の一般的特徴をよく反映している．そして図8，図10より，誤差は高い周波数成分ほど大きくなっている．これは(B)のベルや(C)のシンバルではナイキスト周波数以上の成分が無視できない程度に含まれているために誤差が大きくなってしまったと考えられる．従って，誤差を小さくするにはサンプリング周波数を大きくすればよいことが推測される．

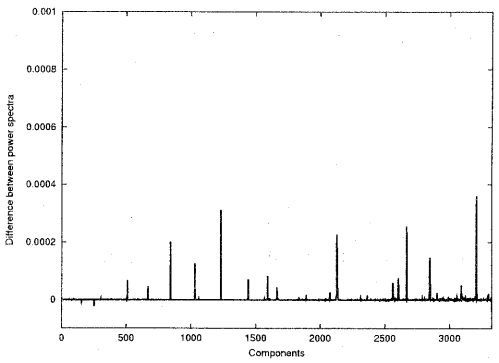

図8 (B) ベルのFFTW，IFFTによるパワースペクトルの各成分の差の推移 (横軸：成分，縦軸：FFTWによるパワースペクトルからIFFTによるそれを引いた差)

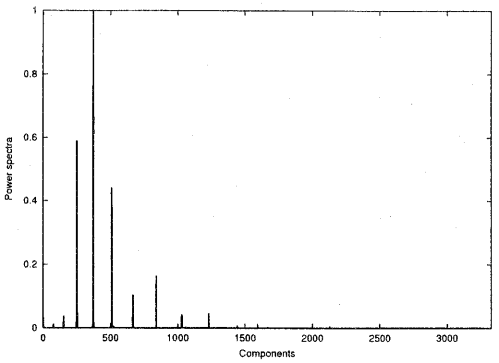

図9 (B) ベルのIFFTによるパワースペクトル (横軸：成分，縦軸：IFFTによるパワースペクトル)

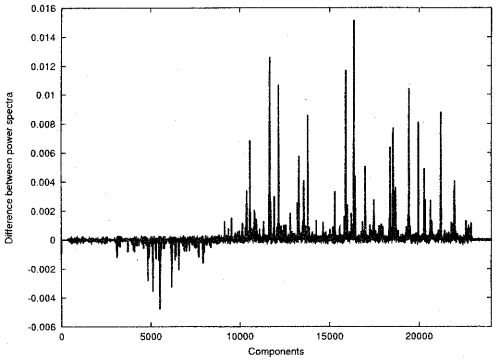

図 10 (C) シンバルの FFTW, IFFT によるパワースペクトルの各成分の差の推移 (横軸：成分，縦軸：FFTW によるパワースペクトルから IFFT によるそれを引いた差)

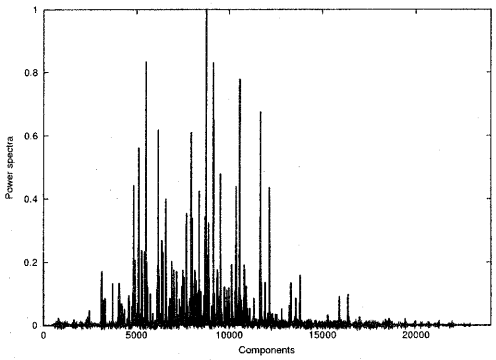

図 11 (C) シンバルの IFFT によるパワースペクトル (横軸：成分，縦軸：IFFT によるパワースペクトル)

5. ま と め

本稿では代表的な FFT ライブラリである FFTW, GSLFFT を用いて，一般的な FFT アルゴリズムの計算時間が入力されたデータの長さに大きく依存することを示した．そして更に長さ N の FFT の計算時間は，多くの場合において，N 以上の 2 のべき乗のうち最小の数 (N_1) を長さとする FFT の計算時間よりも長くなることを示した．これを踏まえ，任意の長さ N のデータを 3 次スプライン補間で長さ N_1 のデータに変換して Cooley-Tukey 型 FFT アルゴリズムを適用する IFFT を考案・実装し，FFTW による出力と IFFT によるそれを比較することで IFFT の動作検証を行った．その結果，IFFT の計算時間は多くの場合において FFTW のそれよりも短くなり，長さが計算時間に及ぼす影響を大幅に減らすことができた．そして IFFT の出力結果も，適切なサンプリング周波数で標本化されたデータを入力した場合であれば FFTW の出力結果との間の誤差を小さく抑えることができた．

6. 今後の予定

今回，IFFT の具体的な入力データとして音波を用いたが，最終的には GHz 帯の電波にまで IFFT を適用し，動作検証を行っていきたい．GHz 帯の電波を正しく標本化するためには，ナイキストの定理より最低 2×10^9Hz 以上のサンプリング周波数が必要である．このような膨大な数のデータを処理するためには IFFT の並列化も行うべきであると考える．また，今回の IFFT の動作検証は入力として 1 次元の実数データを用いたが，さらに複素数や多次元のデータにまで IFFT の概念を拡張していくことも重要であると考えている．

参 考 文 献

1) Brigham, E. Oran: The Fast Fourier Transform. Prentice Hall, (1974).
2) FFTW ホームページ: `http://www.fftw.org/`
3) M. Frigo and S. G. Johnson: FFTW, An Adaptive Software Architecture for the FFT, ICASSP conference proceedings, vol. 3, pp. 1381-1384 (1998).
4) GSL(the GNU Scientific Library) ホームページ: `http://sources.redhat.com/gsl/`
5) 高橋大輔：理工系の基礎数学 8 数値計算, 岩波書店 (1996).
6) 黒瀬能聿，松島勇雄，松尾俊彦：C 言語による科学技術計算サブルーチンライブラリ, 啓学出版 (1986).
7) W. H. Press, S. A. Teulsky, W. T. Vetterling, B. P. Flannery: Numerical Recipes in C, Cambridge Univ. Press (1988). 丹慶勝市，奥村晴彦，佐藤俊郎，小林誠 訳：Numerical Recipes in C, 技術評論社 (1993).