磁気回路の解析手法に関する研究

98H160 室屋晋二

2002 年 2 月 8 日

目 次

1 はじめに **2**

2 研究対象機材 **3**
2.1 Electric Spanish Guitar 3
2.2 Magnetic Pickup 3
2.3 研究使用機 5

3 フーリエ変換 **6**
3.1 フーリエ級数 6
3.2 DFT 6
3.3 FFT 6
3.4 DFT と FFT の違い 8
3.5 直流成分 9
3.6 窓関数 10
3.7 振幅スペクトル 11
3.8 DAT 形式 12

4 解析の手順 **13**

5 解析 **14**
5.1 解析結果 14

6 考察 **15**

1　はじめに

現在、磁気回路や磁気特性の違う Replacement Pickup が多数存在している。磁気特性が違うため音色も違う。しかし音色を言葉として表現する際、人によって言葉が曖昧で確実に伝えることが難しい。視覚的に音色を表示し、様々な磁気特性を持つ Pickup の比較検討ができる状態にすれば音という見えないものを他人に伝え、音色というものの価値観を共有化できるのではないだろうかと考えた。Electric Spanish Guitar の音色の違いというものは主に Pickup の磁気回路の違いである。その磁気回路のの比較の方法として、一定の状態にセットアップした Electric Spanish Guitar の音を PC に MP3 形式で保存し後の処理の為にファイル形式を SOX アプリケーションを使用し DAT 形式へと変換し一連の FFT 処理を行う。さらに視覚的に比較検討しやすい情報にするために周波数特性に着目したスペクトルグラフと音色の判別を行いやすくする為に正規化したグラフの 2 つのグラフを出力するという方法を取り磁気回路の比較検討をしやすい状態にした。

本研究では Electric Spanish Guitar の Magnetic Pickup を対象とし、Pickup の磁気回路の解析手法を提案する。

2　研究対象機材

2.1　Electric Spanish Guitar

膝の上に置いて演奏するタイプの Guitar を Hawaiian Guitar と呼んでいるのに対して、腕で抱え込んで演奏する種類を Spanish Guitar と呼んでいる。他の Guitar には Hawaiian Guitar,Lap(膝) Steel Guitar、Pedal Steel Guitar などがある。そしてペダルには右足で操作し音量を変化させるヴォリューム・ペダルと左足か膝または複合で操作しチューニングを変化させるペダルとニー・レバーがある。このチューニングを変化させる機構はフットペダルまたはレバーを操作すると連動して動くチェンジャーと呼ばれるメカニズムを組み込んだ可動ブリッジがネックの裏側に仕掛けられたロッドを通してブリッジがわずかに動き、張られた弦を引っぱったり弛めたりする仕組みになっている。この様にして各ペダルとレバーを操作または複数組み合わせて操作することによってチューニングとボリュームを様々に変化させることができる。

表 1: 他の Guiter

Hawaiian Guitar	Guitar 本体に足がありペダルのない Guitar
Lap Steel Guitar	足とペダルが無く膝の上に置いて演奏する Guitar
Pedal Steel Guitar	ペダルも足も付いている Guitar

2.2　Magnetic Pickup

本研究の対象とする Magnetic Pickup 図 **(1)** とは一般的にコイルを 6000 から 8000 ターン巻いた金属製のポールピースとバルカンファイバまたはプラスティック製プレートの下にフェライト磁石を張り付けたものからなりたっており、金属弦の振動を電磁誘導作用で電気信へと変換するものである。この図 **(2)** のタイプ Pickup は Gibson 社が開発したのが始まりと言われている。Pickup は、小型交流発電機でモーターを手で回すと電力が生まれるのと同じ原理である。Pickup は Pickup の磁石によって弦が磁性体になり、その磁性体となった弦がコイルを通る磁束を変化させる事で、電磁誘導作用が起こり、電力を発生させる。ピックアップは磁石によって安定した磁界を保っており、自然界は平衡を保とうとする。そこを弦によって安定した磁界を乱されると、弦の動きとは正反対の磁界を発生させて平衡を保とうとする。そのとき電力が発生する。電線に電流を流すと磁界が発生する原理と、逆のことである。コイルの巻き数を増減させる、磁石の磁力をかえる、磁界を変化させるなどが磁気回路を変更したということになる。また Fender 社が開発した図 **(3)** のタイプの Pickup は、ポールピース自体が磁石である。このタイプの Pickup はポールピースの材質はアルミニウム、ニッケル、コバルトの合金であるアルニコという金属が使われるのが一般的で、これを磁化している。Rickenbacker 社が開発した図 **(4)** のような磁石自体が露出していて磁石の形状が馬蹄に似ているため Horseshoe と呼ばれる Pickup もある。

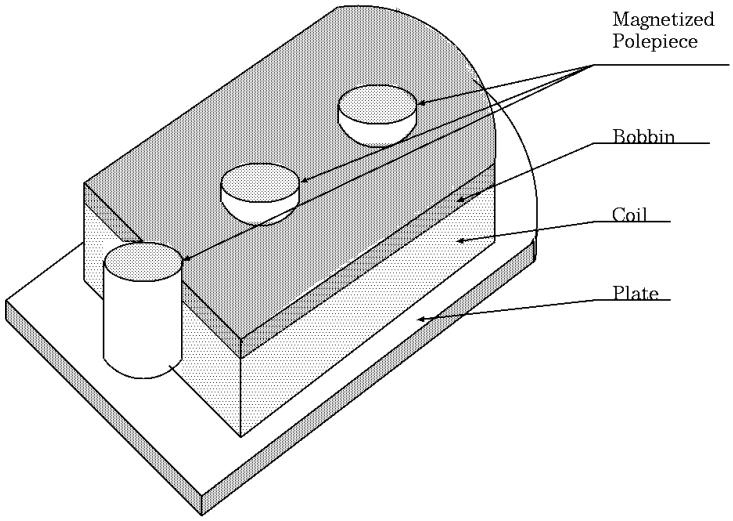

図 1: Pickup 断面図

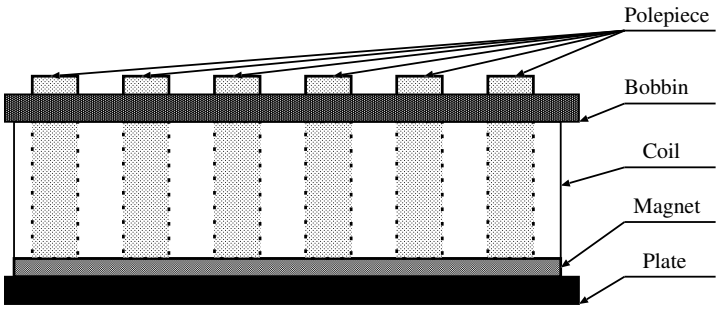

図 2: Gibson type Pickup

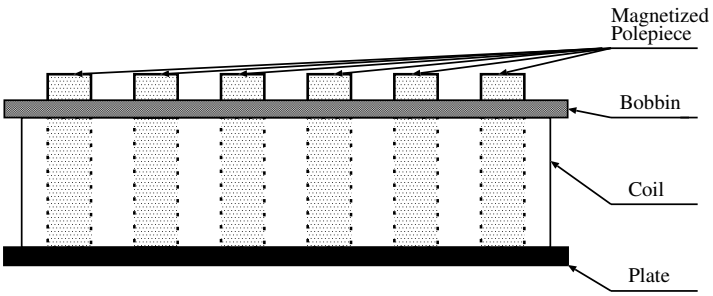

図 3: Fender type Pickup

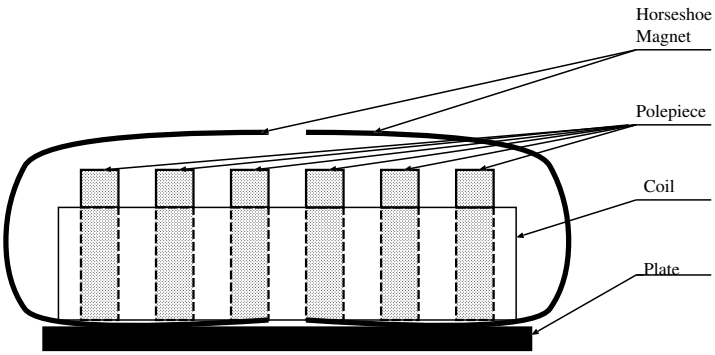

図 4: Ricken backer type HorseshoePickup

Magnetic Pickup 以外には Piezo Pickup、Condenser Pickup、$LightWave^{TM}$ (Optical Pickup System) という Pickup がある。

表 2: 他の Pickup

Piezo Pickup	ギターボディの振動を拾い、電気へと変換する Pickup
Condenser Pickup	カラオケマイクの性能を良くし、小型化したマイクの事
$LightWave^{TM}$	弦の振動を光学センサーで認識し、電気へと変換する Pickup

2.3 研究使用機

Electric Spanish Guitar を磁気特性の違う様々な Pickup を容易に付け換えを容易にした Guitar を卒業研究用 Guitar として製作した。

`http://andrew.d-pad.co.jp/GTR/A2/`

3　フーリエ変換

スペクトル解析の基本理論として、多くの分野で用いられている、フーリエ変換はフランスの数学、物理学者 Fourier (Jean Baptiste Joseph Fourier ,1768-1830) によって発表された。フーリエ変換とは、一見して不規則に見える波形でも、周期性のある波形であれば、その周期の周波数である基本波周波数と、その整数倍の高調波周波数の正弦波と余弦波の組み合わせによって成り立っているというフーリエ級数の原理を元に、それぞれの成分の含まれている割合を求めるものである。それぞれの成分の含まれている割合 (フーリエ係数) を求める事により、一見不規則でつかみどころのない波形が持っている特徴を見い出すことができる。

3.1　フーリエ級数

フーリエ級数とは、ある時間信号 $x(t)$ は直流成分及び周期 T の逆数 $1/T$(Hz) で与えられる周波数成分と、その整数倍の周波数成分 n/T(Hz) の和として与えられるということであり、このことを式で表すと、式 **(1)** の様になる。これがフーリエ級数に展開すると呼ばれるものである。

$$\begin{aligned} x(t) \text{ のフーリエ級数展開} &= (\text{直流成分}) + (\text{係数 } 1 \times \text{基本波}) \\ &\quad +(\text{係数 } 2 \times 2 \text{ 次高調波}) + \cdots + (\text{係数 } n \times n \text{ 次高調波}) \\ &= a_0 + a_1 \cos 2\pi t/T + a_2 \cos 2\pi 2t/T + a_3 \cos 2\pi 3t/T + \cdots + a_n \cos 2\pi nt/T \\ &\quad + b_1 \sin 2\pi t/T + b_2 \sin 2\pi 2t/T + b_3 \sin 2\pi 3t/T + \cdots + b_n \sin 2\pi nt/T \\ &= a_0 + \sum_{n=1}^{\infty} (a_n \cos 2\pi nt/T + b_n \sin 2\pi nt/T) \end{aligned} \tag{1}$$

3.2　DFT

サンプリングされた有限個のディジタル信号に帯するフーリエ変換は、離散フーリエ変換 (DFT : Discrete Fourier Transformer) と呼ばれる。フーリエ変換は、フーリエ級数の係数を求める処理で、フーリエ変換の対象となるデータは、式 **(2)** で表される。(jx_i = 虚数)

$$x(n) = x_r(n) + jx_i(n) \tag{2}$$

複素数で表現するので、DFT による変換結果も複素数式 **(3)** で表される。

$$\begin{aligned} X(k) &= \Sigma(x_r(n) + jx_i(n))(\cos(2\pi nk/N) - j\sin(2\pi nk/N)) \\ &= \Sigma\{(x_r(n)\cos(2\pi nk/N) + x_i(n)\sin(2\pi nk/N) \\ &\quad +j(x_i(n)\cos(2\pi nk/N) - x_r(n)\sin(2\pi nk/N))\} \end{aligned} \tag{3}$$

この DFT 計算プログラムを付録に掲載している。このプログラムは実数部を虚数部に分けて計算し、処理が済んだ順に出力している。

3.3　FFT

FFT は高速フーリエ変換 (Fast Fourier Transform) の事で Cooley.J.W と Tukey.J.W. の 2 人によって、1965 年に"An Algorithm for the Machine Calculation of Complex Fourier Series"という題で Mathematics of Computation 誌上に発表された。この FFT アルゴリズムを用いると、これまで膨大な時間を要していた DFT(離散フーリエ変換) の演算時間が大幅に短縮できる。高速化の仕掛けは、ビット逆転処理やバタフライ演算と呼ばれるコンピュータ処理に適した基本演算処理にある。プログラムリストの長さは DFT プログラムに比べるとかなり長くなっているが、特に乗算の演算回数が激減している。

- ビット逆転処理

  演算の過程で、ビット並びが逆転する現象が起きてしまう為に、正常な順に並び替える処理。

- バタフライ処理

  FFT ぼ基本演算におけるデータの流れが、羽を広げた蝶々(butterfly) の形に似ている事からそう呼ばれる。FFT では、このようなデータの流れの処理を繰り返して行っているが、そうする事によって複素数の乗算処理を共用できるようになるため、乗算回数が大幅に減少し、高速演算処理が可能となっている。

この FFT プログラムを使用するに当たっての、フーリエ変換の基本的な性質を示す。

1. 処理する信号は周期信号と仮定

   コンピュータ処理ではでは、対象となる信号の一部分を切り出して処理する事になるが、対象となる信号は切り出した信号が繰り返しつながっている長い周期信号であると仮定している。実際には、切り出した信号の最初と最後に不連続が生じている場合が多く、スペクトルの誤差となって現れることになるので後に示す窓関数を使用する。

2. 有効周波数範囲

   サンプル数 N の離散フーリエ変換は X_k は、周期 N で繰り返す。時間 T(秒) を間隔 Δt(秒) でサンプリングし、N 個のデータを得たとすると、X_k は $1/T(=1/\Delta t \cdot N)$Hz ごとの N 個の離散周波数点の値となり、それが有効な周波数範囲は $N/T(=1/\Delta t + f_s)$Hz となる。ここで f_s はサンプリング周波数である。

1, **2** のような 2 つの性質を持つ為、周波数スペクトルのうち有効な周波数範囲は $0 \sim f_s/2$Hz の帯域内だけである。そのため、有効なサンプリングを行う為には、入力データの 1 周期内に 2 個以上のサンプリング点が存在しなければならないことを意味している。すなわち周波数 f_s 正弦波に対しては、サンプリング周波数 f_0 を f_s の 2 倍以上にしなければならない。

3.4 DFT と FFT の違い

DFT と FFT の演算回数の比較すると通常では、データ数 n 個のフーリエ変換を行うのに、n^2 回の積和演算が必要となるが、FFT アルゴリズムを用いると、$n \log 2n$ 回と飛躍的に減らす事が出来る。DFT に比べ FFT はビット逆転処理とバタフライ処理を行うことによって高速演算を可能にしている。このように、FFT ではデータ数が多くなるにしたがって、その威力を発揮される事となる。DFT で、サンプル数 $N = 4$ の時の計算回数は、データ数が 4 個なので、16 回の複素乗算と 12 回の複素加算が必要となる。一般に、データ数が N 個では、N^2 回の複素乗算と、$N(N-1)$ 回の複素加算が必要となる。一方、FFT の場合、サンプル数 $N = 4$ の時の計算回数は、複素乗算が 4 回、複素加算が 8 回である。一般にデータ数 N=2^r(r は正の整数) では、複素乗算回数は $N \times r/2$ 回、複素加算回数は $N \times r$ となる。参考に DFT と FFT の演算回数を比較したものを表 **(3)** に示す。この演算速度の違いから本実験では DFT 処理ではなく FFT 処理を使用した。

表 3: DFT と FFT の演算回数の比較

データ数	乗算回数		
	DFT	FFT	DET/FFT
128	16,384	448	37
256	65,536	1,024	64
512	262,114	2,304	114
1,024	1,048,578	5,120	205
2,048	4,194,304	11,264	32
4,096	16,777,216	24,576	683

3.5　直流成分

入力データの平均値を表す。振り幅 0 を中心に + 側と − 側が + 側に片寄っていると図図 (5) バランスが取れていると図 (6)、逆に − 側にずれていると図 (7) の値となる。本実験では最大振幅スペクトルの大きさで正規化したグラフとして表示するのだが、その際、処理対象データの直流成分が+側や-側に大きく偏っていると、その大きく偏った値で正規化されてしまうため、振幅スペクトルが相対的に小さく表示されてしまうことになる。これを避けるために直流成分を 0 にする。

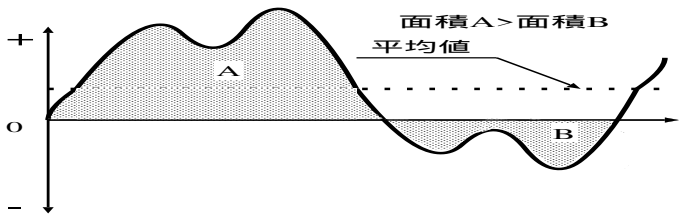

図 5: 直流成分がプラス

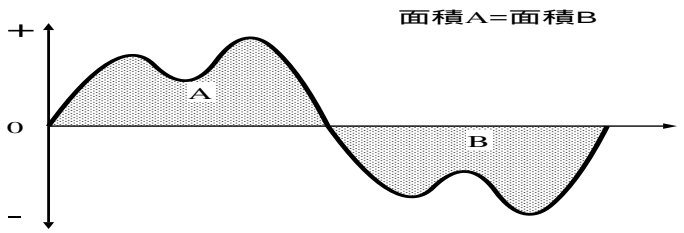

図 6: 直流成分が 0

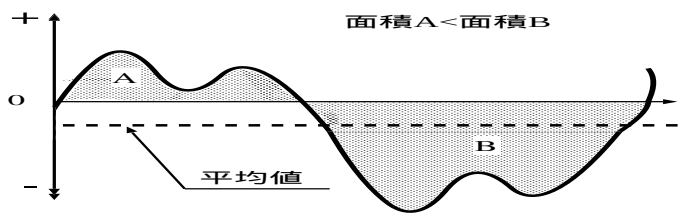

図 7: 直流成分がマイナス

3.6 窓関数

FFT 処理を行うための入力された波形データは、基本波のちょうど整数倍の周期分であることが必要である。どのような位相で始まっていても 1 周期分の波形であれば処理では同じ結果が得られる。これは、繰り返し (周期) 波形フーリエ変換の処理対象の前提条件なので、波形のどの部分を切り出しても波形が整数倍の周期分であればよいのである。ただし、現実の問題となるとちょうど整数倍周期の波形データを取り出すのは簡単ではない。整数倍周期分のデータを取り出すには基本波の周波数が分かっていることが前提となる。しかしそれは、周波数成分がわからないから振幅スペクトルを求めることと矛盾する。どんな周波数が成分が含まれているのかを知りたいからこそ FFT 変換を行うのであって、基本周波数や高調波がわかっているなら処理する必要がない。しかし、ある程度 FFT 変換をするのに適した入力データでないと正しい処理結果が出ないのも事実である。そこである程度 FFT 処理に適した入力データにするために窓関数というものがある。

- 窓関数の効果

  フーリエ変換処理誤差をできるだけ少なくするためには入力データから切り出した信号の最初の値と最後の値を等しくする処理を行うのが窓関数である。図 (8) の様な波形データにおいて整数倍のサイクルから少し遅めでサンプリングしてしまうと、正しく振幅スペクトルが算出されないために窓関数を掛けて、図 (9) の様に波形データの最初と最後の部分の振幅を極端に小さくする。これは繰り返し波形と見なすというフーリエ変換の前提条件に従って、解析に影響の出そうな継目部分の振幅を小さくし不自然な解析結果が出るのを防ぐ効果がある。

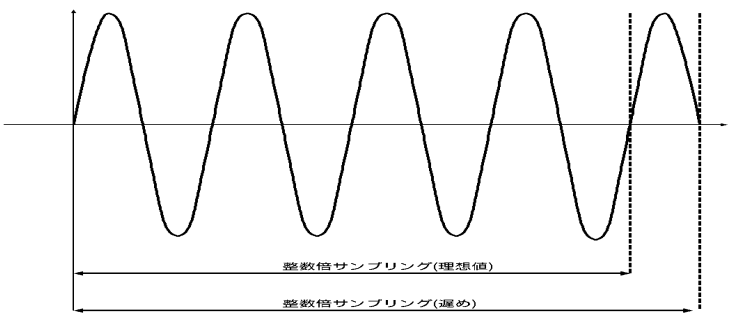

図 8: 窓関数処理前

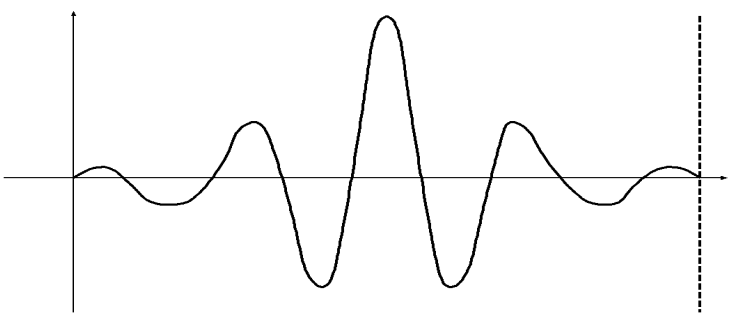

図 9: 窓関数処理後

- 窓関数の種類
  - hamming 窓

    周波数分解能がよいので、近接した 2 つの周波数成分を検出するのに適している。音声関係の解析などによく使用させる。本研究でも音を解析するので、この hamming 窓関数を使用している。付録にプログラムを掲載している。
  - hanning 窓,blackmn 窓

    主な周波数成分から少し離れた周波数の、小さな成分を検出するのに適している。そのため計測器には比較適によく使用される。

3.7 振幅スペクトル

入力波形に含まれている各周波数成分は、それぞれ、大きさが異なるだけでなく、それぞれの位相も異なっている。周波数分析においては、成分の大きさを調べると同時に、この位相の具合を調べることが重要になる。

同じ周波数の正弦波と余弦波では、位相が 90 ° 異なるので、ある周波数 (n/T)Hz の成分の大きさ (振幅) は、余弦波成分の係数 a_n と正弦波成分の係数 b_n のベクトルの比を取ることによってわかる。これらのフーリエ係数を周波数軸上に離散的に分布させた状態のものを一般的に線スペクトルという。その際に、振幅に着目するならば振幅スペクトル、位相に着目するならば位相スペクトル、パワーに着目するならば、パワースペクトルを求める。

- パワースペクトル

  各周波数の正弦波、余弦葉の係数を 2 乗して加えた値で表される、振幅スペクトルの 2 乗に相当する。各周波数成分の持っているエネルギー値の比を得ることが出来る。グラフに表した場合、振幅スペクトルよりも周波数成分の割合が強調されて表示される。

$$ {X_n}^2 = {a_n}^2 + {b_n}^2 \tag{4} $$

- 振幅スペクトル

  パワースペクトルの平方根を取った値で表され、 各周波数成分の振幅を意味する。本研究では振幅を算出するため、振幅スペクトル求めるプログラムを付録に掲載している。

$$ {X_n}^2 = \sqrt{{a_n}^2 + {b_n}^2} \tag{5} $$

- 位相スペクトル

  $\tan^{-1}(b_n/a_n)$ は、その位相を表している。

$$ \theta_n = \tan^{-1}(b_n/a_n) \tag{6} $$

3.8 DAT 形式

MP3 ファイル形式から SOX アプリケーション (SoundeXchange `http://www.spies.com/Sox/`) のテキストデータである DAT ファイルに変換したデータが表 (4) である。最初の一行目がサンプリングレートを表しており続く数値データの左側が時間、右側サンプル値を表している。そのテキストデータをグラフにしたのが 図 (10) が実時間波形で縦軸がサンプル値、横軸が時間を示している。

表 4: dat ファイル

Sample Rate 44100	
0.29473923	9.1552734e-05
0.2947619	7.6293945e-05
0.29478458	4.5776367e-05
0.29480726	3.0517578e-05
0.29482993	3.0517578e-05
≀	≀

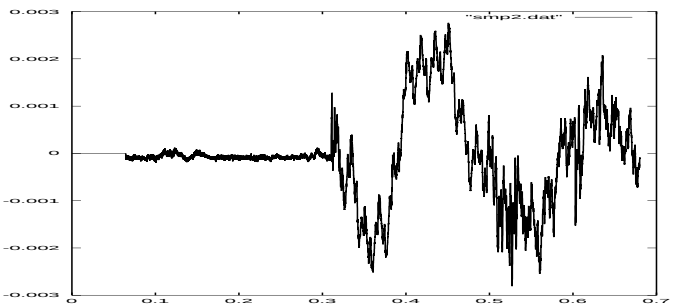

図 10: 実時間波形

4　解析の手順

概要での一連の FFT 処理を以下に示す。3 ~ 7 までのデータ処理は付録に掲載しているプログラムで標準入出力を使用し、一括で行えるようにしている。

1. 入力音を PC に MP3 形式のファイルとして入力し保存する。
2. DAT 形式に変換する。
3. データ個数 N を 2^r 個 (r は正の整数) にする。
4. 平均値を 0 にする。
5. 窓関数を掛ける。
6. FFT 処理をする。
7. 振幅スペクトルを求める。
8. グラフ化をする。

上記の作業により視覚的に磁気回路の特性を検討出来るようになる。

5　解析

解析では一連の FFT 処理が正しく行えるかどうかを確かめるため、Electric Spanish Guitar の 5 弦の開放音である A 音のデータを解析した。

5.1　解析結果

解析の結果図 **(11)**, 図 **(12)** と言う結果が出力された。しかし、結果の図は Electric Spanish Guitar の 5 弦の開放音には主に 440Hz の周波数成分が多く含まれているという、思った通りの結果が得られなかった。予想される原因は、あらかじめ CD-ROM に納められていた音源をさらに MP3 形式に圧縮変換しサンプリングしたデータをさらに SOX アプリケーションでサンプリングしたことによって、サンプリングレートの計算が狂ったのが原因だと思われる。

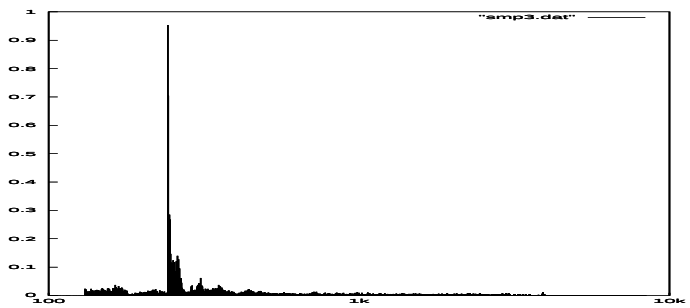

図 11: 線スペクトル (周波数表示)

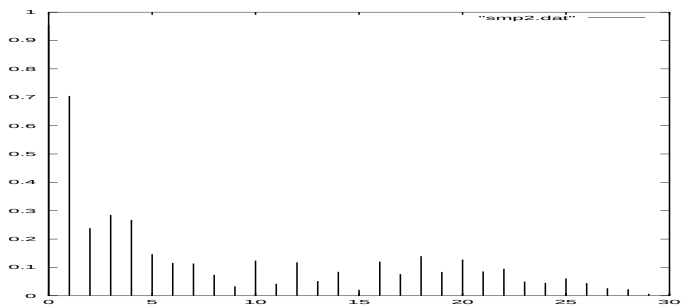

図 12: 線スペクトル (正規化表示)

6　考察

一般的な周波数解析アプリケーションでは図 (11) のような周波数分布でしか表示出来ないが、本研究では図 (12) のような最大振幅スペクトルの大きさで正規化したグラフを表示できるようにした。これにより入力音の高低に関わらず正規化して表示できる。

本研究での周波数解析プログラムのテストはサンプリングにサンプリングを行っていたデータを使用した為に失敗したが、Electric Spanish Guitar の音を直接 PC に取り込む方法からの FFT 処理の実験を今後行いたいと思う。この FFT 処理のテストが成功し、本研究の解析のプログラムの精度や計算速度を高める事で今後は Pickup の解析に留まらず、弦のゲージや素材を変えた時やボディやネックの材を変えた時のサウンドの変化の解析にもこの解析手法は利用できると思われる。またサンプリングレートを高くして解析できれば、いろんな媒体 (Radio, AudioCD, MP3, 電話、など) を通したときに原音をできるだけ忠実に伝えるにはどういう音色にすればいいか、という解析もできると思われる。

謝辞

本研究を行うにあたって、多大な御指導・御協力を頂いた大垣 斉講師、藤井 信夫教授、中村 孝講師、そして各研究生の方々に深く感謝の意を表します。

参考文献

[1] 坂巻佳壽美, 見てわかるディジタル信号処理

[2] 秋月影雄, 松山安男, 吉江修, C 言語 ディジタル信号処理

[3] 周波数解析

http://www.nano.pe.u-tokyo.ac.jp/ kota/ennsyuu/syuuha.htm

[4] フーリエ変換

http://gtr01.adin.hamamatsu-u.ac.jp:8080/Apl70.html

[5] サウンドフォーマット

http://hp.vector.co.jp/authors/VA015468/prog/wav/format.html

[6] linux コマンド

http://linux.nikkeibp.co.jp/command/index-alpha.html

[7] 翻訳

http://babelfish.altavista.com/tr

[8] gnuplot

http://ruby2.scphys.kyoto-u.ac.jp/ inakura/man/gnuplot/gnuplot.html

[9] ギターギャラリー

http://www5a.biglobe.ne.jp/ a-seta/museo.htm

[10] シェルスクリプト

http://web.kanazawa-u.ac.jp/univ/center/ma5-6.html

付録

直流成分を 0 にする

```
#include<stdio.h>
#include<malloc.h>

main ()
{
  int i, N;
  double *d, sum = 0.0;

  scanf ("%d", &N);
  printf ("%d\n", N);
  d = (double *) malloc (sizeof (double) * N);
/*
 * 平均値を計算
 */
  for (i = 0; i < N; i++)
    {
      scanf ("%lf", &d[i]);
      sum += d[i];
    }
  sum /= N;
/*
 * 平均値を計算ここまで
 * 平均値を引く計算
 */
  for (i = 0; i < N; i++)
    {
      printf ("%lf\n", d[i] - sum);
    }
/*
 * 平均値を引く計算ここまで
 */
  free (d);
}
```

窓関数

```
#include<stdio.h>
#include<math.h>
#define PAI 3.14159265358979
main ()
{
  int n, N;
  float d;

  scanf ("%d", &N);
  printf ("%d\n", N);

  for (n = 0; n < N; n++)
    {
      scanf ("%f", &d);
      d = d * (0.54 - 0.46 * cos (2 * PAI * n / (N - 1)));
      printf ("%f\n", d);
    }
}
```

DFT

```
#include<stdio.h>
#include<math.h>
#include<malloc.h>
#define Pai 3.14159265358

main ()
{
  int n, k, N;
  typedef struct
    {
      double r;
      double i;
    }
  complex;
  complex *x, X;
  scanf ("%d", &N);
  printf ("%d\n", N);
  x = (complex *) malloc (sizeof (complex) * N);
/*
 *DFT処理
 */
  for (k = 0; k < N; k++)
    {
      scanf ("%lf", &x[k].r);
      x[k].i = 0.0;
    }
  for (n = 0; n < N; n++)
    {
      X.r = 0.0;
      X.i = 0.0;
      for (k = 0; k < N; k++)
    {
      X.r += x[k].r * cos (2.0 * Pai * n * k / N)+ x[k].i * sin (2.0 * Pai * n * k / N);
      X.i += x[k].i * cos (2.0 * Pai * n * k / N)- x[k].r * sin (2.0 * Pai * n * k / N);
    }
      printf ("%1f %1f\n", X.r, X.i);
    }
/*
 *DFT処理ここまで
 */
  free (x);
}
```

FFT

```
#include<stdio.h>
#include<math.h>
#include<malloc.h>
#define Pai  3.14159265358

typedef struct
  {
    double r;
    double i;
  }
complex;

int
bitr (int bit, int r)
{
  int i, bitr;

  bitr = 0;
  for (i = 0; i < r; i++)
    {
      bitr ≪= 1;
      bitr |= (bit & 1);
      bit ≫= 1;
    }
  return (bitr);
}



void
swap (complex * d0, complex * d1)
{
  double a;

  a = d0→r;
  d0→r = d1→r;
  d1→r = a;
  a = d0→i;
  d0→i = d1→i;
  d1→i = a;
}

main ()
```

```
{
  int n, k, N, r, i, j, j1, j2, p, k1, bit;
  double a, b;
  complex *x;

  scanf ("%d", &N);
  printf ("%d\n", N);
  x = (complex *) malloc (sizeof (complex) * N);

  for (k = 0; k < N; k++)
    {
      scanf ("%lf", &x[k].r);
      x[k].i = 0.0;
    }

  r = 0;
  i = N >> 1;
  while (i > 0)
    {
      i >>= 1;
      r++;
    }
  n = 1 << r;
  j2 = n;
  k = 0;
  j1 = r - 1;
  for (j = 0; j < r; j++)
    {
/*
 * FFT 処理
 */
      j2 >>= 1;
      for (;;)
    {
      for (i = 0; i < j2; i++)
        {
          p = k >> j1;
          k1 = k + j2;
          a = x[k1].r * cos (2.0 * Pai / n * bitr (p, r)) + x[k1].i * sin (2.0 * Pai / n * bitr (p, r));
          b = x[k1].i * cos (2.0 * Pai / n * bitr (p, r)) - x[k1].r * sin (2.0 * Pai / n * bitr (p, r));
/*
 * バタフライ演算
 */
```

```
        x[k1].r = x[k].r − a;
        x[k1].i = x[k].i − b;
        x[k].r = x[k].r + a;
        x[k].i = x[k].i + b;
/*
 * バタフライ演算ここまで
 */
        k++;
       }
     k += j2;
     if (k ≥ n)
       break;
   }
     k = 0;
     j1−−;
   }
/*
 * ビット反転処理
*/
  for (k = 0; k < n; k++)
    {
      bit = bitr (k, r);
      if (bit > k)
    swap (&x[k], &x[bit]);
    }
/*
 * ビット反転処理,FFT 処理ここまで
*/
  for (k = 0; k < N; k++)
    printf ("%1f %1f\n", x[k].r, x[k].i);

  free (x);
}
```

振幅スペクトル

```
#include<stdio.h>
#include<math.h>

main ()
{
  int i, N;

  typedef struct
    {
      double r;
      double i;
    }
  complex;

  complex data0;

  scanf ("%d", &N);
  printf ("%d\n", N / 2);
/*
 * 振幅スペクトル
 */
  for (i = 0; i < N / 2; i++)
    {
      scanf ("%lf %lf", &data0.r, &data0.i);
      printf ("%lf\n", sqrt (data0.r * data0.r + data0.i * data0.i));
    }
/*
 * 振幅スペクトルここまで
 */
}
```

一連の数値処理

```
#! /bin/sh

TEMP=/tmp/X−$$
trap "/bin/rm -f ${TEMP}*;exit 0" 0 1 2 3 15

# データの入力

cat  > $TEMP

# サンプリングレートの表示をとる

export SF=‘head −1 $TEMP | awk ’{print $NF}’‘

# ここからデータ個数を変換

tail +2 $TEMP | awk ’{print $2}’ > ${TEMP}−1
W=‘cat ${TEMP}−1 | wc −l‘

C=0

while [ $W −gt 1 ]
do
   W=‘expr ${W} / 2‘
   C=‘expr ${C} + 1‘
done

while [ $C −gt 0 ]
do
   W=‘expr ${W} \∗ 2‘
   C=‘expr ${C} − 1‘
done

echo $W              > ${TEMP}−2
head −$W ${TEMP}−1 ≫ ${TEMP}−2

# データ個数変換ここまで
# "|"を使用し順次データ処理プログラムにデータを渡している

cat ${TEMP}−2 | ./choku0 | ./hamming | ./fft | ./amp | tail +2

exit 0
```