電子制御工学実験報告書

実験題目　：　**信号処理プログラミング1**

報告者　：　4年37番　本間 三暉

提出日　：　2022年5月20日

実験日　：　2022年4月14日,4月21日,4月28日,5月19日

実験班　：

共同実験者　：

※ 指導教員記入欄

評価項目	配点	一次チェック ・　・	二次チェック ・　・
記載量	20		
図・表・グラフ	20		
見出し，ページ番号，その他体裁	10		
その他の減点	–		
合計	50		

コメント：

1　目的

アナログ信号をディジタルデータに変換し，ディジタル機器で処理するために必要な基礎事項について学習し，C 言語で基本的な信号処理プログラムを作成する．また，実用的な音声フォーマットの一つである WAVE ファイルの構造を理解し，音声データの出入力プログラムを作成する．

2　演習

今回の実験の演習では，特に明言されていない限りソースコード 1 に示すヘッダファイル，及びソースコード 2 に示す定義ファイルが読み込まれているものとする．なお，pi.h は付録 B のリスト 6 である．

ソースコード 1: ヘッダファイル

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <string.h>
5 #include <time.h>
6 #include <ctype.h>
7
8 #include "pi.h"
```

ソースコード 2: 定義ファイル

```
1 #define BUFSIZE 80
2 #define A_BIAS 0x80
3 #define DATANUM 101
```

また，それ以前の演習で示した自作関数はそれ以降のソースコードで断りなく使用する．

2.1　次の指示に従いプログラムを作成し，出力を gnuplot で可視化して動作確認を行え．

2.1.1　任意の弧度 r を $[0 : 2\pi]$ の範囲内に収まるように変換する関数 rad を作成せよ．

ソースコード 3 に関数 rad を示す．

ソースコード 3: double rad(double r)

```
1     double mod;
2     mod = fmod(r,2 * PI);
3     if(r < 0) mod = mod + 2 * PI;
4     return mod;
```

数学関数 fmod は r が負の場合 $[-2\pi : 0]$ の値を取る．このままでは目的の値は得られないので，r が負の場合は fmod が返した値に 2π[rad] を加えることで値を $[0 : 2\pi]$ の範囲に収める事ができる．

これを出力すると図 1 となる．

2.1.2 任意の弧度 r に対して，のこぎり波の振幅値を求める関数 saw を作成せよ．

ソースコード 4 に関数 saw を示す.

ソースコード 4: double saw(double r)

```
1    double saw_wave;
2    saw_wave = 1 - rad(r) / PI;
3    return saw_wave;
```

ソースコード 3 の rad において出力値の変化幅は 2π[rad] である．これに対してのこぎり波の変化幅は 2[rad] であるので，関数 rad の戻り値を π で割ることによって変化幅を 2[rad] にすることができる．また理想的なのこぎり波の範囲は $[-1:1]$ で，$0, 2\pi, 4\pi$ で -1 から 1 へ瞬時に変化するため，ソースコード 4 の 2 行目のように記述すれば良い.

これを出力すると図 2 となる.

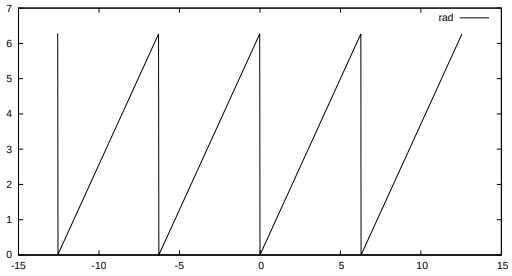

図 1: rad の出力波形

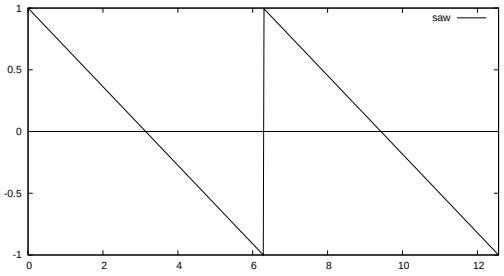

図 2: saw の出力波形

2.2 次の指示に従いプログラムを作成し，動作を確認せよ．

2.2.1 リスト 2 のコードを解析し，完成させよ．振幅 100，周波数 3[Hz] のデータを gnuplot でグラフ化し，サンプリングの効果を確認せよ．

リスト 2 のコードを完成させたものをソースコード 5 に示す.

ソースコード 5: リスト 2 変更部分のみ

```
1  for (t = 0; t <= T_END; t += DT) {
2    rad = t * (frq / 1000) * 2 * PI;
3    vin = amp * sin(rad) + A_BIAS;
4    vout = vin;
5    printf("%4d,␣%4d\n", t, vout);
6  }
```

これを出力すると 3 となる.

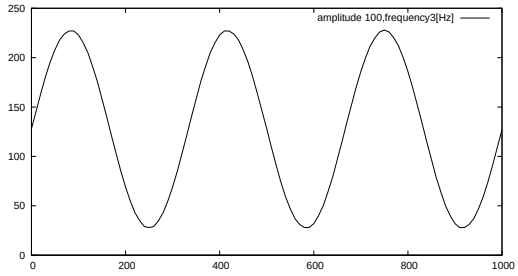

図 3: 振幅 100, 周波数 3[Hz]

2.2.2 振幅 140，周波数 3[Hz] のデータを生成し，その波形を確認せよ．波形に不具合があれば，その原因を考えて不具合を軽減するような修正を行え．

振幅 140，周波数 3[Hz] の条件で出力した波形は図 4 となった．これは vout が unsigned char 型であり，[0 : 225] の範囲の値しか取らないため，この範囲に入らない値はオーバーフローしてしまい別の値として出力されてしまうためだと考えられる．

これを修正するために，範囲を超えてしまったものは範囲内の値に修正すれば良い．これを実装したものをソースコード 6 に示す．

ソースコード 6: 修正後

```
1  for (t = 0; t <= T_END; t += DT) {
2    rad = t * (frq / 1000) * 2 * PI;
3    vin = amp * sin(rad) + A_BIAS;
4    if (vin < 0) vin = 0;
5    if (vin > 255) vin = 255;
6    vout = vin;
7    printf("%4d, %4d\n", t, vout);
8  }
```

また，これを出力したものを図 5 に示す．

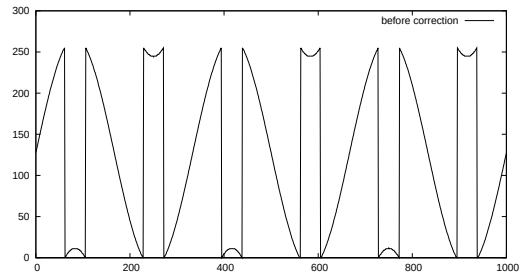

図 4: 振幅 140, 周波数 3[Hz] 修正前

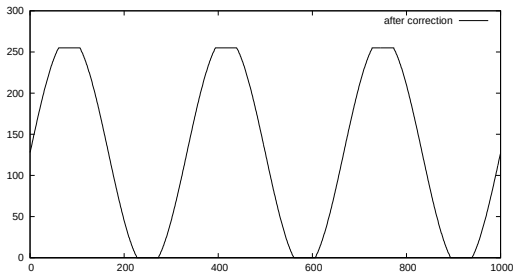

図 5: 振幅 140, 周波数 3[Hz] 修正後

2.2.5 サンプリング定理を満たさないような高い周波数の正弦波を PCM によりディジタルデータ化すると，周波数や位相が全く異なる波のように見えることがある．この現象を実際に観測せよ．

この現象をまとめたものを図 6 に示す．なお，DT は標本化間隔のことである．

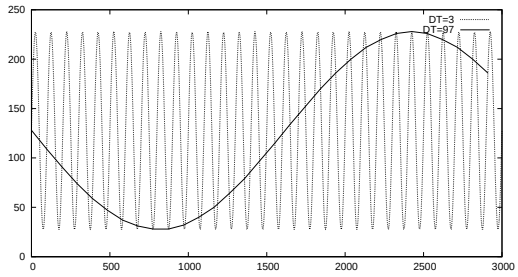

図 6: 振幅 100, 周波数 10[Hz]

このように同じ振幅同じ周波数のはずなのに，全く違う波形に見えてしまうことを観測した．この現象は高速回転するタイヤなどでも起きるものなのでかなり身近なものである．

2.3 次のプログラムを作成して動作を確認せよ．出力の形式はプログラム例 3 を参考にして，加工後の振幅値を一列加えることにしておくと良い．

2.3.1 白色雑音を加えるオフライン型プログラムを作成せよ．

白色雑音を加えるオフライン型プログラムをソースコード 7 に示す．

ソースコード 7: add-wn2.c

```
1  #define BUFSIZE 80
2  #define DATANUM 101
3  #define ROUND(x) ((x > 0) ? (x * 0.5) : (x - 0.5))
4
5  int main(int argc, char **argv) {
6    int n, n_keep;
7    int tm[DATANUM], amp[DATANUM], aout[DATANUM];
8    int nmax;
9    double err;
10   char buf[BUFSIZE];
11   FILE *fp;
12
13   if (argc != 3) {
14     fprintf(stderr, "Usage:␣%s␣infile␣max_noise\n", argv[0]);
15     return EXIT_FAILURE;
16   }
17   if ((fp = fopen(argv[1], "r")) == NULL) {
18     fprintf(stderr, "File:␣%s␣cannot␣open\n", argv[1]);
19     return EXIT_FAILURE;
20   }
```

```
21
22    for (n = 0; n < DATANUM;) {
23      if (fgets(buf, sizeof(buf), fp) == NULL) break;
24      if (buf[0] == '#') {
25        printf("%s", buf);
26        continue;
27      }
28      tm[n] = atoi(strtok(buf, ","));
29      amp[n] = atoi(strtok(NULL, "\r\n\0"));
30      n++;
31    }
32    fclose(fp);
33
34    n_keep = n;
35    nmax = atoi(argv[2]);
36    printf("# WN %d\n", nmax);
37    srand((unsigned)time(NULL));
38    for (n = 0; n < DATANUM; n++) {
39      err = nmax * (2.0 * rand() / RAND_MAX - 1.0);
40      aout[n] = amp[n] + ROUND(err);
41      if (aout[n] < 0) aout[n] = 0;
42      if (aout[n] > 255) aout[n] = 255;
43    }
44    for (n = 0; n < n_keep; n++) {
45  #if defined TEST
46      printf("%4d, %4d, %4d\n", tm[n], amp[n], aout[n]);
47  #else
48      printf("%4d,%4d\n", tm[n], aout[n]);
49  #endif
50    }
51    return EXIT_SUCCESS;
52  }
```

ソースコード 7 において 22 行目の for 文の条件式に n++ がなく 30 行目に n++ がある理由は，for 文の条件式に n++ があると continue の処理をした際に n++ が実行されてしまい，出力用の配列に代入するのがうまくいかないからである．

振幅 100，周波数 3[Hz] の正弦波に最大振幅を 10 とした白色雑音を加えた波形と元の波形を重ねたものを図 7 に示す．

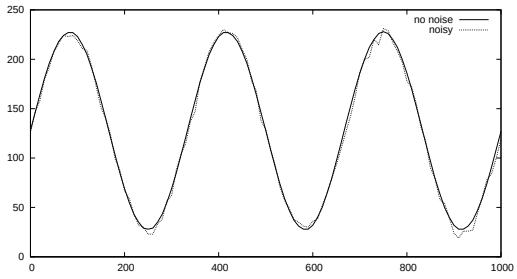

図 7: ノイズなしとノイズあり

最大振幅をそこまで大きくしなかったが，目に見えるほどずれてしまっているので，本当にノイズを除去できるのか不安である．

2.3.2 3 点単純平均移動プログラムを作成せよ．

オンライン型をソースコード 8，オフライン型をソースコード 9 に示す．

ソースコード 8: mwave3-1.c

```
1  #define MOVING_AVERAGE 3
2  int main(int argc, char **argv) {
3    int tm,tm_1, ain, aout, nmax, n = 0;
4    double err_3add = 0.0, err_2before, err_1before, now;
5    char buf[BUFSIZE];
6    FILE *fp;
7
8    if (argc != 2) {
9      fprintf(stderr, "Usage:␣%s␣infile␣max_noise\n", argv[0]);
10     return EXIT_FAILURE;
11   }
12   if ((fp = fopen(argv[1], "r")) == NULL) {
13     fprintf(stderr, "File:␣%s␣cannot␣open\n", argv[1]);
14     return EXIT_FAILURE;
15   }
16   while (fgets(buf, sizeof(buf), fp) != NULL) {
17     if (buf[0] == '#') {
18       printf("%s", buf);
19       continue;
20     }
21     tm = atoi(strtok(buf, ","));
22     tm_1 = tm;
23     ain = atoi(strtok(NULL, "\r\n\0"));
24     err_2before = err_1before;
25     err_1before = now;
26     now = ain;
27     err_3add += ain;
28     if (n < MOVING_AVERAGE - 1) {
29       n++;
30       continue;
31     }
32
33     aout = err_3add / MOVING_AVERAGE;
34     if (aout < 0) aout = 0;
35     if (aout > 255) aout = 255;
36
37 #if defined TEST
38     printf("%4d,␣%4d,␣%4d\n", tm_1, ain, aout);
39 #else
40     printf("%4d,%4d\n", tm_1, aout);
41 #endif
```

```
42
43      err_3add -= err_2before;
44
45    }
46    fclose(fp);
47    return EXIT_SUCCESS;
48  }
```

ソースコード 9: mwave3-2.c

```
1   int main(int argc, char **argv) {
2     int n, i;
3     int tm[DATANUM], amp[DATANUM], aout[DATANUM], editing[DATANUM - 1];
4
5     int nmax;
6     double err, err_3add = 0;
7     char buf[BUFSIZE];
8     FILE *fp;
9
10    if (argc != 2) {
11      fprintf(stderr, "Usage:␣%s␣infile␣max_noise\n", argv[0]);
12      return EXIT_FAILURE;
13    }
14    if ((fp = fopen(argv[1], "r")) == NULL) {
15      fprintf(stderr, "File:␣%s␣cannot␣open\n", argv[1]);
16      return EXIT_FAILURE;
17    }
18    for (n = 0; n < DATANUM;) {
19      if (fgets(buf, sizeof(buf), fp) == NULL) break;
20      if (buf[0] == '#') {
21        printf("%s", buf);
22        continue;
23      }
24      tm[n] = atoi(strtok(buf, ","));
25      amp[n] = atoi(strtok(NULL, "\r\n\0"));
26      n++;
27    }
28    fclose(fp);
29    for (n = 0; n < MOVING_AVERAGE / 2 + 1; n++) {
30      err_3add += amp[n];
31    }
32
33    for (n = MOVING_AVERAGE / 2; n < DATANUM - 1; n++) {
34      err_3add += amp[n + MOVING_AVERAGE / 2];
35      aout[n] = err_3add / MOVING_AVERAGE;
36      err_3add -= amp[n - MOVING_AVERAGE / 2];
37      if (aout[n] < 0) aout[n] = 0;
38      if (aout[n] > 255) aout[n] = 255;
39    }
40    for (n = 1; n < DATANUM; n++) {
41  #if defined TEST
```

```
42      printf("%4d,␣%4d,␣%4d\n", tm[n], amp[n], aout[n]);
43  #else
44      printf("%4d,%4d\n", tm[n], aout[n]);
45  #endif
46    }
47    return EXIT_SUCCESS;
48  }
```

オンライン型が最新のデータから 3 つを保持するのに対し，オフライン型はすべてのデータが既知であるため配列をうまく活用して処理することが求められる．両方とも同じやり方で 3 点の平均値を求めているが上記のやり方が一番計算量が少ないと思われる．

雑音を含んだ波形を 3 点単純移動平均プログラムで復元したものと雑音を含んだ波形を図 8 に示す．また，復元したものともとの波形を図 9 に示す．

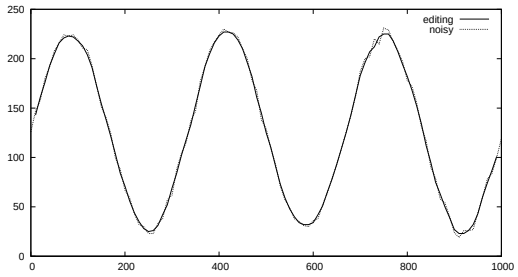

図 8: ノイズありとノイズ除去後

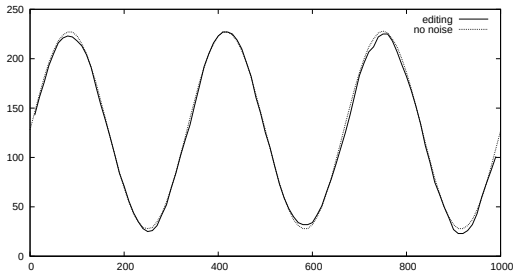

図 9: ノイズなしとノイズ除去後

この結果を見ると，やや大きくずれた部分は完全に雑音を取り去るのは難しそうであるが，概ねもとの波形に近づけることができていると考えられる．

2.4 次の指示に従い，式変形及びプログラム作成と動作確認を行え．

2.4.1 式 (7) を変形して式 (8) を導出する過程を示せ．

テキストの式 (7) は以下の通りである．

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

中身を展開してそれぞれの項に分解し，定数を前に出す．

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i) - \frac{2\bar{x}}{N}\sum_{i=1}^{N}x_i + \frac{1}{N}\sum_{i=1}^{N}\bar{x}^2$$

このとき，

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N}x_i$$

なので,

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i) - 2\bar{x}^2 + \bar{x}^2$$
$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i) - \bar{x}^2$$

これは式 (8) であるので，式 (7) を変形して式 (8) を導出することができた.

2.4.2 コマンドライン引数で CSV ファイル名と解析対象の列番号を指定し，最小値，最大値，平均値，標準偏差，最大振幅を求めるプログラムを示せ.

オンライン型をソースコード 10，オフライン型をソースコード 11 に示す.

ソースコード 10: stat1.c

```
1  int main(int argc, char **argv) {
2    int tm, ain, column = 2, ncolumn, add_square, a_rms_add;
3    int add_n = 0, max, min, n = 0;
4    double average, standard_deviation, max_amplitude, a_rms;
5    char buf[BUFSIZE];
6    FILE *fp;
7
8    if (argc < 2) {
9      fprintf(stderr, "Usage:␣%s␣infile␣max_noise\n", argv[0]);
10     return EXIT_FAILURE;
11   }
12
13   if ((fp = fopen(argv[1], "r")) == NULL) {
14     fprintf(stderr, "File:␣%s␣cannot␣open\n", argv[1]);
15     return EXIT_FAILURE;
16   }
17   if (argc > 2) {
18     column = atoi(argv[2]);
19   }
20
21   while (fgets(buf, sizeof(buf), fp) != NULL) {
22     if (buf[0] == '#') {
23       printf("%s", buf);
24       continue;
25     }
26     tm = atoi(strtok(buf, ","));
27     for (ncolumn = 1; ncolumn < column; ncolumn++) {
28      ain = atoi(strtok(NULL, ",\r\n\0"));
29     }
30     add_n = ain;
31     max = ain;
32     min = ain;
33     add_square = ain * ain;
```

```
34      a_rms_add = (ain - A_BIAS) * (ain - A_BIAS);
35      n++;
36      break;
37    }
38
39    while (fgets(buf, sizeof(buf), fp) != NULL) {
40      if (buf[0] == '#') {
41        printf("%s", buf);
42        continue;
43      }
44      tm = atoi(strtok(buf, ","));
45      for (ncolumn = 1; ncolumn < column; ncolumn++) {
46        ain = atoi(strtok(NULL, ",\r\n\0"));
47      }
48
49      add_n += ain;
50
51      if (max < ain) {
52        max = ain;
53      }
54
55      if (min > ain) {
56        min = ain;
57      }
58
59      add_square += ain * ain;
60      a_rms_add += (ain - A_BIAS) * (ain - A_BIAS);
61      n++;
62    }
63
64    average = (double)add_n / n;
65
66    standard_deviation =
67        sqrt(add_square / n - average * average);
68
69    max_amplitude = A_BIAS - min;
70    if (max - A_BIAS > A_BIAS - min) {
71      max_amplitude = max - A_BIAS;
72    }
73
74    a_rms = sqrt(a_rms_add / n);
75
76    printf(
77        "最小値␣␣:%d\最大値 n␣␣:%d\平均値 n␣␣"
78        ":%.4f\標準偏差 n:%.4f\最大振幅 n:%.4f\実効値 n␣␣:"
79        "%.4f\n",
80        min, max, average, standard_deviation, max_amplitude, a_rms);
81
82    fclose(fp);
83    return EXIT_SUCCESS;
84  }
```

ソースコード 11: stat2.c

```
1    int main(int argc, char **argv) {
2  int column = 2, ncolumn, add_square, a_rms_add;
3  int add_n = 0, max, min, n = 0, keep_n;
4  int tm[DATANUM], amp[DATANUM];
5  double average, standard_deviation, max_amplitude, a_rms;
6  char buf[BUFSIZE];
7  FILE *fp;
8
9  if (argc < 2) {
10   fprintf(stderr, "Usage:␣%s␣infile␣max_noise\n", argv[0]);
11   return EXIT_FAILURE;
12 }
13
14 if ((fp = fopen(argv[1], "r")) == NULL) {
15   fprintf(stderr, "File:␣%s␣cannot␣open\n", argv[1]);
16   return EXIT_FAILURE;
17 }
18 if (argc > 2) {
19   column = atoi(argv[2]);
20 }
21
22 for (n = 0; n < DATANUM;) {
23   if (fgets(buf, sizeof(buf), fp) == NULL) break;
24   if (buf[0] == '#') {
25     printf("%s", buf);
26     continue;
27   }
28   tm[n] = atoi(strtok(buf, ","));
29   for (ncolumn = 1; ncolumn < column - 1; ncolumn++) {
30     amp[n] = atoi(strtok(NULL, ",\r\n\0"));
31   }
32
33   n++;
34 }
35 keep_n = n;
36 n--;
37 add_n = amp[n];
38 max = amp[n];
39 min = amp[n];
40 add_square = amp[n] * amp[n];
41 a_rms_add = (amp[n] - A_BIAS) * (amp[n] - A_BIAS);
42
43 n--;
44
45 for (; n >= 0; n--) {
46   add_n += amp[n];
47   if (max < amp[n]) {
```

```
48        max = amp[n];
49      }
50
51      if (min > amp[n]) {
52        min = amp[n];
53      }
54      add_square += amp[n] * amp[n];
55      a_rms_add +=
56          (amp[n] - A_BIAS) * (amp[n] - A_BIAS);
57    }
58
59    average = (double)add_n / keep_n;
60
61    standard_deviation =
62        sqrt(add_square / keep_n - average * average);
63
64    max_amplitude = A_BIAS - min;
65    if (max - A_BIAS > A_BIAS - min) {
66      max_amplitude = max - A_BIAS;
67    }
68
69    a_rms = sqrt(a_rms_add / keep_n);
70
71    printf(
72        "最小値␣␣:%d\最大値 n␣␣:%d\平均値 n␣␣"
73        ":%.4f\標準偏差 n:%.4f\最大振幅 n:%.4f\実効値 n␣␣:"
74        "%.4f\n",
75        min, max, average, standard_deviation, max_amplitude, a_rms);
76    fclose(fp);
77    return EXIT_SUCCESS;
78  }
```

コマンドライン引数で列番号を指定するのはソースコード 10 の 27 行目の for 文で実装している．関数 strtok の第二引数に入れたすべての文字で区切るので，カンマ (,) を付け足して指定した列数まで回せば良い．

各値を求める方法はコードを見れば明らかであるが，各値について初期化を行う時定数ではなく各データの最初の値を使用しなければならない点について気をつけなければならない．

これらのデータが正しく求められているかを確認するために理論値と比較したものを表 1 に示す．

表 1: 理論値との比較

	理論値	オンライン型	オフライン型
最小値	28	28	28
最大値	228	228	228
平均値	127	127.5	127.5
標準偏差	70.3	71	70
最大振幅	100	100	100
実効値	70.71	70	70

これを見ると，大幅に外れた値はないので大丈夫そうである．

2.5 以下の指示に従って，プログラミングと動作確認を行え．

基本的に長岡高専 4 年電子制御工学実験の信号処理プログラミングのページ
(https://www2.st.nagaoka-ct.ac.jp/ãtaka/ec4exp/#Prog)
にある WAVE ファイルを用いるものとする．

2.5.1 リスト 5 を完成させ，もよりの WAVE ファイルのいくつかについてヘッダ情報を調べ，表に整理せよ．

調べたヘッダファイルをまとめたものを表 2 に示す．

表 2: WAVE ヘッダファイル

	ringout.wav	ringin.wav	chimes.wav	timetone.wav
RIFF 情報サイズ	5254	10018	55768	77609
fmt チャンクサイズ	16	16	16	16
FormatID	1	1	1	1
チャンネル数	1	1	2	1
サンプリングレート	11025	11025	22050	32000
データ速度	11025	11025	88200	32000
ブロックサイズ	1	1	4	1
量子化ビット数	8	8	16	8
データサイズ	5167	9981	55684	77573

共通している項目が多い WAVE ファイル同士があるので，この結果からある程度の規格などがあると考えても良さそうである．

2.5.2 モノラル音声・量子化ビット数 8 の WAVE ファイルの波形データを CSV ファイルに吐き出すダンププログラムを作成せよ．

ここで使用する関数 read_head は，テキストのリスト 5 の戻り値をサンプリング周波数としたものである．

ソースコード 12: wav2txt-m8.c

```
1  int main(int argc, char **argv) {
2    double count;
3    double tm = 0;
4    int dat;
5    long start_num = 0L, end_num;
6    FILE *fp;
7    uLong sampling_rate;
8    uShort ch, qbit;
9
10   if ((argc < 3) || (argc > 4)) {
11     fprintf(stderr, "Usage:␣%s␣file␣page\n", argv[0]);
12     return EXIT_FAILURE;
13   }
14   if ((fp = fopen(argv[1], "rb")) == NULL) {
15     fprintf(stderr, "File␣(%s)␣cannot␣open\n", argv[1]);
16     return EXIT_FAILURE;
17   }
18
19   sampling_rate = read_head(fp, &ch, &qbit);
20   if (argc > 3) {
21     start_num = atol(argv[2]);
22     fseek(fp, start_num, SEEK_CUR);
23   }
24   if (argc == 4) {
25     end_num = atoi(argv[3]);
26   }
27   tm = (double)start_num / sampling_rate * 1000.0;
28   printf("%f", tm);
29   count = 1000.0 / sampling_rate;
30   while ((dat = fgetc(fp)) != EOF) {
31     if (count > end_num) break;
32     tm += count;
33     printf("%0.3f,%4d\n", tm, dat);
34   }
35   fclose(fp);
36   return EXIT_SUCCESS;
37 }
```

これに ringin.wav を通したものを図 10 に示す．

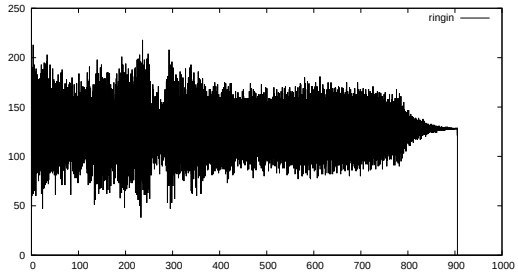

図 10: ringin.wav

細かく振動することで波による上下差をなくし，比較的音のゆらぎの少ないファイルを作っているのだと思われる．

2.5.3 ユーザが指定した周波数と振幅の正常波を 1 秒の長さだけ記録 (録音) するプログラムを作成し，動作を確かめよ．

ユーザが指定した周波数と振幅の正弦波を，モノラル音声で量子化ビット数 8，サンプリング周波数 11,025[Hz] で一秒の長さ記録するプログラムをソースコード 13 に示す.

ソースコード 13: rec-m8.c

```
1  #define BIAS 0x80
2  #define RIFF_SIZE 11061
3  #define FMT_SIZE 16
4  #define FORMAT_ID 1
5  #define SAMPLING_RATE 11025
6  #define CHANNEL_N 1
7  #define BLOCK_SIZE 1
8  #define Q_BIT 8
9  #define WAVE_SECOND 1
10 int data_size = SAMPLING_RATE * WAVE_SECOND;
11
12 void header_data(FILE *fp) {
13   int riff_size = RIFF_SIZE;
14   int fmt_size = FMT_SIZE;
15   int format_id = FORMAT_ID;
16   int channel_n = CHANNEL_N;
17   int sampling_rate = SAMPLING_RATE;
18   int block_size = BLOCK_SIZE;
19   int q_bit = Q_BIT;
20   int data_speed = sampling_rate * block_size;
21
22   fwrite("RIFF", sizeof(char), 4, fp);
23   fwrite(&riff_size, sizeof(int), 1, fp);
24   fwrite("WAVE", sizeof(char), 4, fp);
25   fwrite("fmt␣", sizeof(char), 4, fp);
26   fwrite(&fmt_size, sizeof(int), 1, fp);
```

```
27   fwrite(&format_id, sizeof(short), 1, fp);
28   fwrite(&channel_n, sizeof(short), 1, fp);
29   fwrite(&sampling_rate, sizeof(int), 1, fp);
30   fwrite(&data_speed, sizeof(int), 1, fp);
31   fwrite(&block_size, sizeof(short), 1, fp);
32   fwrite(&q_bit, sizeof(short), 1, fp);
33   fwrite("data", sizeof(char), 4, fp);
34   fwrite(&data_size, sizeof(int), 1, fp);
35 }
36
37 int main(int argc, char **argv) {
38   int t;
39   double amp, frq, rad, vin;
40   unsigned char vout[SAMPLING_RATE];
41
42   FILE *fp;
43   if (argc < 4) {
44     fprintf(stderr, "Usage:␣%s␣file␣page\n", argv[0]);
45     return EXIT_FAILURE;
46   }
47   if ((fp = fopen(argv[3], "wb+")) == NULL) {
48     fprintf(stderr, "File␣(%s)␣cannot␣open\n", argv[1]);
49     return EXIT_FAILURE;
50   }
51
52   amp = atof(argv[1]);
53   frq = atof(argv[2]);
54
55   for (t = 0; t <= data_size; t++) {
56     rad = t * (frq / SAMPLING_RATE) * 2 * PI;
57     vin = amp * sin(rad) + BIAS;
58     if (vin < 0) vin = 0;
59     if (vin > 255) vin = 255;
60     vout[t] = vin;
61   }
62
63   header_data(fp);
64   fwrite(vout, sizeof(unsigned char), data_size, fp);
65   fclose(fp);
66   return EXIT_SUCCESS;
67 }
```

これを用いて振幅 100，周波数 100[Hz] の正弦波を記録した WAVE ファイルを作成した．再生してみて程よい重低音が聞こえたためこれでいいとおもわれるが，念のためソースコード 12 を使用して波形を観測してみた．その波形を図 11 に示す.

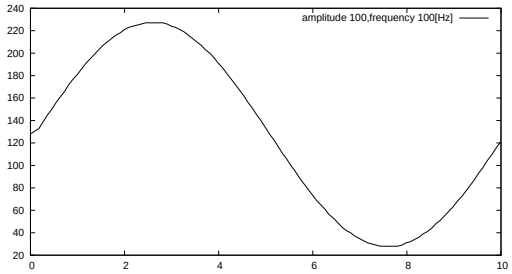

図 11: WAVE ファイルの波形

これを見る限り，しっかりと振幅 100，周波数 100[Hz] の正弦波が書きこまれていることがわかる．

仮にここで vout の型をデータサイズが 2Byte 以上の型にしてしまうと，全く違う波形が生成されてしまうため，注意が必要である．

3　あとがき

3.1　感想

今回は波に関する実験を行ったが，色々な波形の作り方や WAVE ファイルの構造や作り方など興味深い内容が多くあった．波について知ることはいろいろな場面で役に立つと思うので，今後もこれらについての理解を深めて行きたい．

3.2　要望

今回レポートを作成する際に気になった点として，gnuplot についての情報が従来の方法では手に入りづらい点が挙げられる．様々なエラーが発生するが，どこが悪いのかが分かりづらく関係するファイルもやや多いため，修正作業に時間がかかる．

ここについて情報が多く集まっているサイト等の記載があればもう少し楽になると考えられる．

参考文献

1. 令和 4 年度電子制御工学実験 · 4 年前期テキスト