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**

報告者　：　4 年 37 番　本間 三暉

提出日　：　2022 年 7 月 1 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：

2 演習

通常レポートで示した通りである．また，通常レポートに示した定義ファイルにソースコード 1 を追加した．

ソースコード 1: 定義ファイル追加

```
1 #define ROUND(x) ((x > 0) ? (x + 0.5) : (x - 0.5))
```

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

2.1.3 任意の弧度 r に対して，矩形波の振幅値を求める関数 squ を作成せよ

ソースコード 2 に関数 squ を示す．

ソースコード 2: double squ(double r)

```
1 double squ(double r) {
2   return ((saw(r) < 0) ? -1 : 1);
3 }
```

通常レポートで作成した saw 関数を流用し，saw(r) が正のときは 1，そうでないときは-1 を取るようにした．原理的には数学関数の sin 関数等を使用しても良いが，実行時間などの観点から，こっちの方が良いと判断した．

入力範囲を $[-2\pi : 2\pi]$ としたときの出力波形を図 1 に示す．

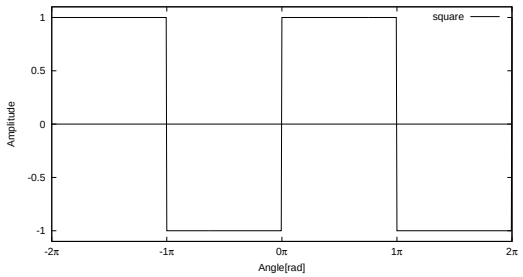

図 1: squ の出力波形

2.1.4 任意の弧度 r に対して，三角波の振幅値を求める関数 tri を作成せよ

ソースコード 3 に示す．

ソースコード 3: double tri(double r)

```
1  double tri(double r) {
2    int index = 1;
3    double triangle, mod;
4
5    mod = fmod(r, PI);
6    if (r < 0) mod = mod + PI;
7    triangle = mod * 2 / PI;
```

```
8
9       if (triangle > 1) triangle = 2 - triangle;
10      if (r < 0) {
11        r *= (-1);
12        index *= (-1);
13      }
14
15      while (1) {
16        r -= PI;
17        if (r < 0) break;
18        index *= (-1);
19      }
20      triangle *= index;
21      return triangle;
22    }
```

入力範囲を $[-2\pi : 2\pi]$ としたときの出力波形を図 2 に示す.

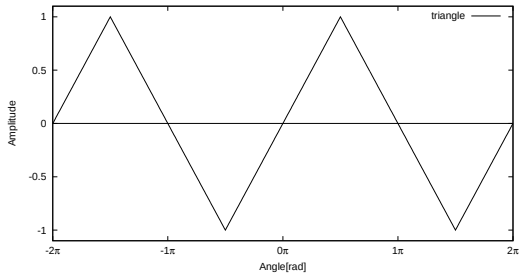

図 2: tri の出力波形

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．

2.2.3 量子化誤差を評価するため，sin10af1.c に量子化における二乗平均平方根誤差を計算する機能を追加せよ．

ソースコード 4 に main 部の変更点，ソースコード 5 に追加したユーザ定義関数を示す.

ソースコード 4: esum に関係する main 部の変更点

```
1  for (t = 0; t <= T_END; t += DT, n++) {
2    rad = t / (1000 / frq) * 2 * PI;
3    vin = amp * sin(rad) + A_BIAS;
4
5    vout = (vin < 0) ? 0 : ((vin > 255) ? 225 : vin);
6    esum = err_sum(vin, vout, esum);
7    printf("%4d,␣%4d\n", t, vout);
8  }
9  esum /= n;
10 esum = sqrt(esum);
11 printf("#E␣%g\n", esum);
```

ソースコード 5: double err_sum(double,unsigned char,double)

```
1 double err_sum(double true_value, unsigned char quantization, double e_rms) {
2   e_rms = (true_value - quantization) * (true_value - quantization);
3   return e_rms;
4 }
```

ソースコード 5 は量子化前の値 true_value と量子化後の値 quantization の差を二乗したものを戻り値とする関数である.

2.2.4 各時刻の振幅値の小数点以下第一位を四捨五入して量子化するプログラム sin10af2.c を作成し，量子化における RMS 誤差の軽減効果を定量的に調べよ.

sin10af2.c の sin10af1.c からの変更点をソースコード 6 に示す.

ソースコード 6: sin10af2.c

```
1   for (t = 0; t <= T_END; t += DT, n++) {
2     rad = t / (1000 / frq) * 2 * PI;
3     vin = amp * sin(rad) + A_BIAS;
4     vout = (vin < 0) ? 0 : ((vin > 255) ? 225 : ROUND(vin));
5     esum = err_sum(vin, vout, esum);
6     printf("%4d,␣%4d\n", t, vout);
7   }
```

ソースコード 6 は vin の値を vout に代入するときに四捨五入をするようにしたものである．sin10af1.c にソースコード 4 の変更を加えたファイルと sin10af2.c に振幅 100，周波数 3[Hz] の条件で時系列データの RMS 誤差を比べると，図 1 のようになる.

表 1: 四捨五入による RMS 誤差の比較

	sin10af1	sin100af2
E_{RMS}	0.0995037	7.07016×10^{-15}

これを見ると，四捨五入をすることで RMS 誤差が大幅に軽減されたことが確認できる.

2.2.6 正弦波の振幅，周波数，位相，標本化間隔 (ms 単位の実数) をコマンドライン引数で指定したとき，サンプリング結果を出力するプログラムを作成せよ.

位相の単位を ms 単位の実数としたときのソースコードをソースコード 7 に示す．なお，位相角の単位は SI 単位系ではラジアン (rad) であるが，今回は波形が正しいか確認しやすいため，単位は ms 単位の実数とする.

ソースコード 7: sinafpt.c

```
1   int main(int argc, char **argv) {
2     int t, n = 0;
3     double amp, frq, phf, rad, vin, esum = 0, dt;
```

```
4     unsigned char vout;
5
6     if (argc != 5) {
7       fprintf(stderr, "Usage:␣%s␣amplitude␣frequency\n", argv[0]);
8       return EXIT_FAILURE;
9     }
10
11    amp = atof(argv[1]);
12    frq = atof(argv[2]);
13    phf = atof(argv[3]);
14    dt = atof(argv[4]);
15
16    printf("#A␣%f\n", amp);
17    printf("#F␣%f\n", frq);
18    printf("#T␣%f\n", dt);
19    printf("#B␣%d\n", A_BIAS);
20    printf("#N␣%f\n", T_END / dt + 1);
21
22    for (t = 0; t <= T_END; t += dt) {
23      rad = (t + phf) / (1000 / frq) * 2 * PI;
24      vin = amp * sin(rad) + A_BIAS;
25      vout = (vin < 0) ? 0 : ((vin > 255) ? 225 : ROUND(vin));
26      printf("%4d,␣%4d\n", t, vout);
27    }
28    return EXIT_SUCCESS;
29  }
```

これを用いて，振幅 100，周波数 10[Hz]，位相 25[ms]，標本化間隔 3[ms] の正弦波を出力したものを図 3 に示す．波形を見やすくするため，実際には 1000[ms] まで測定したが，横軸は [0:600] の範囲で出力する．

入力した条件から，周波数 10[Hz] のとき 1 周期は 100[ms] となる．位相はこれの $\frac{1}{4}$ 倍の 25[ms] なので，振幅 100，周波数 10[Hz]，標本化間隔 3[ms] の余弦波が出力されていればよいはずである．図 3 を見ると，振幅 100，周波数 10[Hz]，標本化間隔 3[ms] の余弦波が出力されているため正しいことが確認できる．

2.2.7 振幅，周波数，位相，標本化間隔 (ms 単位の実数) をコマンドライン引数で指定したとき，のこぎり波，矩形波，三角波を出力するプログラムを作成せよ．

プログラム sin10af2 の主要部ソースコード 6 をソースコード 8，9，10 に書き換えることで，それぞれのこぎり波，矩形波，三角波を出力するプログラムを示す．

ソースコード 8: sawaft.c

```
1   dt = atof(argv[3]);
2   printf("#N␣%f\n", T_END / dt + 1);
3
4   for (t = 0; t <= T_END; t += dt) {
5     rad = t / (1000 / frq) * 2 * PI;
6     vin = amp * saw(rad) + A_BIAS;
7     vout = (vin < 0) ? 0 : ((vin > 255) ? 225 : ROUND(vin));
```

```
8     printf("%4f,␣%4d\n", t, vout);
9   }
```

ソースコード 9: squaft.c

```
1   dt = atof(argv[3]);
2   printf("#N␣%f\n", T_END / dt + 1);
3
4   for (t = 0; t <= T_END; t += dt) {
5     rad = t / (1000 / frq) * 2 * PI;
6     vin = amp * squ(rad) + A_BIAS;
7     vout = (vin < 0) ? 0 : ((vin > 255) ? 225 : ROUND(vin));
8     printf("%4f,␣%4d\n", t, vout);
9   }
```

ソースコード 10: triaft.c

```
1   dt = atof(argv[3]);
2   printf("#N␣%f\n", T_END / dt + 1);
3
4   for (t = 0; t <= T_END; t += dt) {
5     rad = t / (1000 / frq) * 2 * PI;
6     vin = amp * tri(rad) + A_BIAS;
7     vout = (vin < 0) ? 0 : ((vin > 255) ? 225 : ROUND(vin));
8     printf("%4f,␣%4d\n", t, vout);
9   }
```

これらのソースコードに振幅 100，周波数 3[Hz]，標本化間隔 3[ms] の条件で出力した波形を図 4，5，6 に示す.

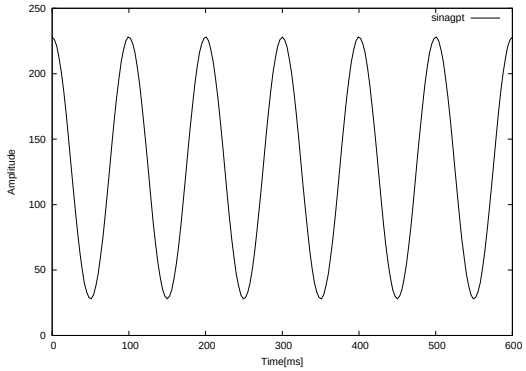

図 3: sinafpt.c で出力した正弦波

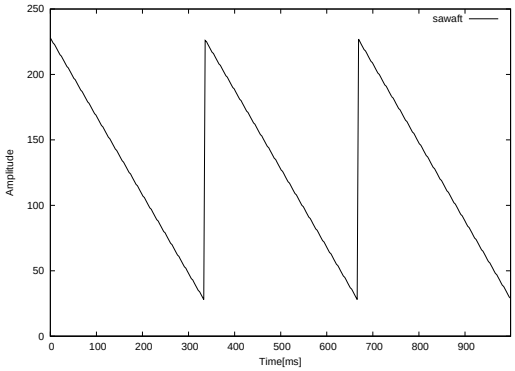

図 4: sawaft.c で出力したのこぎり波

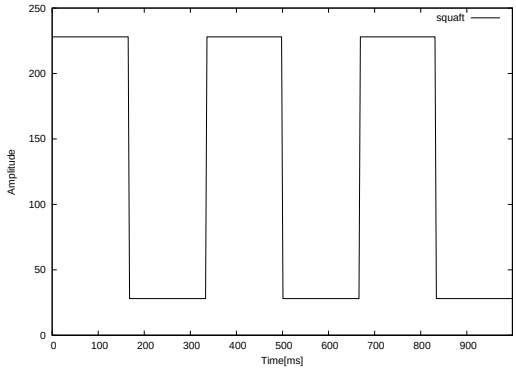

図 5: squaft.c で出力した矩形波

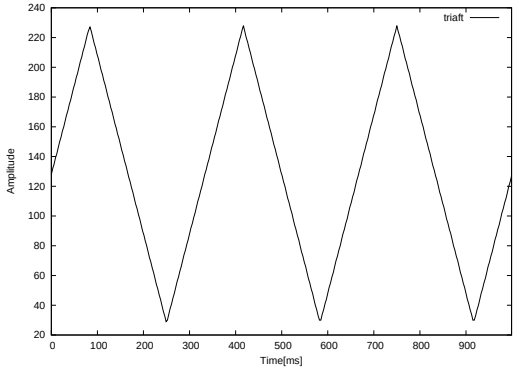

図 6: triaft.c で出力した三角波

グラフの範囲は [0:999] である．これらの波形を見る限り出力されていることが確認できる．

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．

2.3.3 5 点単純移動平均プログラムを作成せよ

5 点単純移動平均プログラムのオンライン型をソースコード 11 に，オフライン型をソースコード 12 に示す．

ソースコード 11: mvave5-1.c

```
1  #define MOVING_AVERAGE 5
2  int main(int argc, char **argv) {
3    int tm, ain, aout, nmax, n = 0;
4    double err_5add = 0.0, now;
5    char buf[BUFSIZE];
6    int err_before[MOVING_AVERAGE];
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    if ((fp = fopen(argv[1], "r")) == NULL) {
14      fprintf(stderr, "File:␣%s␣cannot␣open\n", argv[1]);
15      return EXIT_FAILURE;
16    }
17    while (fgets(buf, sizeof(buf), fp) != NULL) {
18      if (buf[0] == '#') {
19        printf("%s", buf);
20        continue;
21      }
22      tm = atoi(strtok(buf, ","));
23      ain = atoi(strtok(NULL, "\r\n\0"));
24      for (int count = MOVING_AVERAGE - 1; count > 0; count--) {
25        err_before[count] = err_before[count - 1];
26
27      }
28      err_before[0] = ain / MOVING_AVERAGE;
29      err_5add += err_before[0];
30      if (n < MOVING_AVERAGE - 1) {
31        n++;
32        continue;
33      }
34      aout = (err_5add < 0) ? 0 : (err_5add > 255) ? 225 : ROUND(err_5add);
35
36  #if defined TEST
37      printf("%4d,␣%4d,␣%4d\n", tm, ain, aout);
38  #else
39      printf("%4d,%4d\n", tm, aout);
40  #endif
41      err_5add -= err_before[MOVING_AVERAGE - 1];
42    }
43    fclose(fp);
44    return EXIT_SUCCESS;
45  }
```

ソースコード 12: mvave5-2.c

```
1   #define MOVING_AVERAGE 5
2   int main(int argc, char **argv) {
3     int n, i;
4     int tm[DATANUM], amp[DATANUM], aout[DATANUM] = {0}, editing[DATANUM - 1];
5     int nmax;
6     double err, err_5add = 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")) / MOVING_AVERAGE;
26      n++;
27    }
28    fclose(fp);
29    for (n = 0; n < MOVING_AVERAGE - 1; n++) {
30      err_5add += amp[n];
31    }
32    for (n = MOVING_AVERAGE / 2-1; n <= DATANUM - 1; n++) {
33      err_5add += amp[n + MOVING_AVERAGE / 2];
34      aout[n] = (err_5add < 0) ? 0 : (err_5add > 255) ? 225 : ROUND(err_5add);
35      err_5add -= amp[n - MOVING_AVERAGE / 2];
36    }
37    for (n = MOVING_AVERAGE / 2-1; n <= DATANUM - (MOVING_AVERAGE / 2); n++) {
38  #if defined TEST
39      printf("%4d,␣%4d,␣%4d\n", tm[n], amp[n] * MOVING_AVERAGE, aout[n]);
40  #else
41      printf("%4d,%4d\n", tm[n], aout[n]);
42  #endif
43    }
44    return EXIT_SUCCESS;
45  }
```

予めデータを 4 つの合計を求めておき，新しいデータを読み込むたびにそれに加算し平均を求める．その後加算した中で最も古いデータを引くことで一度のループに対して，一回の加算，一回の除算，一回の減算で計算を行うことができる．これらに，振幅 100，周波数 3[Hz] の正弦波に最大値を 10 に設定した白色雑音を加えた波形と，それに 5 点移動平均処理を施した波形を比較する．オンライン型によって処理したものを図 7 に，オフライン型によって処理したものを図 8 に示す.

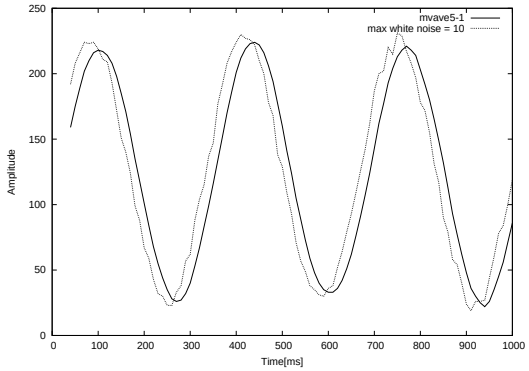

図 7: オンライン型で処理した振幅 100，周波数 3[Hz]，最大白色雑音 10 の正弦波

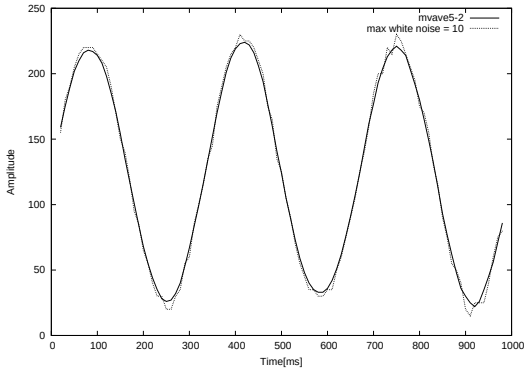

図 8: オフライン型で処理した振幅 100，周波数 3[Hz]，最大白色雑音 10 の正弦波

これを見ると，オンライン型はオフライン型に比べ値を出力するのに遅れが生じていることがわかる．また，最大振幅が小さくなっているように感じる．これを確かめるため，振幅 100，周波数 3[Hz] の正弦波に最大値を 10 に設定した白色雑音を加えた波形を 3 点移動平均と 5 点移動平均で処理したものを比較する．それを図 9 に示す．なお，オンライン型とオフライン型には出力時間のズレ以外の違いがないので，両方ともオフライン型の場合のみを比較する．

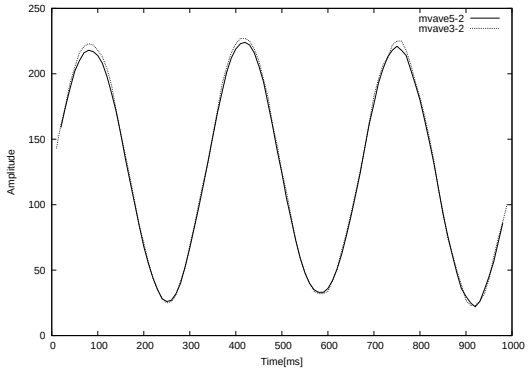

図 9: 3 点移動平均及び 5 点移動平均の比較

図 9 を見ると波形の振幅値が大きくなるにつれ，3 点移動平均は 5 点移動平均に比べ振幅値は小さくなっている事がわかる．この事実は，3 点移動平均と 5 点移動平均のどちらが優れているかを判断するときの一つの指標になると考えられる．

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．

2.4.3 雑音が重畳された CSV ファイルを引数として与えたときに，SNR[dB] を求めるプログラムを作成せよ．

SNR[dB] を求めるプログラムの主要部をソースコード 13 に示す．

ソースコード 13: snr1.c の主要部

```
while (fgets(buf, sizeof(buf), fp) != NULL) {
  if (buf[0] == '#') {
    printf("%s", buf);
    continue;
  }
  tm[n] = atoi(strtok(buf, ","));
  amp[n] = atoi(strtok(NULL, ","));
  aerr[n] = atoi(strtok(NULL, "\r\n\0"));
  add_s += (amp[n] - A_BIAS) * (amp[n] - A_BIAS);
  add_n += (aerr[n] - amp[n]) * (aerr[n] - amp[n]);
  n++;
}
snr_db = 10 * log10(add_s / add_n);
```

振幅 100，周波数 3[Hz]，白色雑音の最大値を 10 とした正弦波を基本波形として，その波形から振幅，周波数，白色雑音の最大値の値の内それぞれ一つだけ変えた波形を作成する．それぞれの値を振幅 140，周波数 5[Hz]，白色雑音の最大値 15 としたとき 4 つのデータを比較したものを表 2 に示す．

表 2: 白色雑音を加えた波形の S/N 比

	基本波形	振幅 140	周波数 5[Hz]	白色雑音の最大値 15
S/N 比	22.929221	26.120139	23.190504	19.768701

毎回シード値がランダムな疑似乱数を使用しているため．多少の誤差は発生してしまうため許容しなくてはならない．しかし，表 2 を見ると，白色雑音の最大値を変えた場合の S/N 比の変化量が比較的大きい．そのため主に S/N 比は白色雑音の最大値によって変化量が変化すると考えられる．

2.4.4 3 点移動平均や 5 点移動平均を行った後の CSV ファイルを引数として与えたときに，SNR[dB] を求めるプログラムを作成せよ．

もとの波形を出力した CSV ファイルと，3 点移動平均，及び 5 点移動平均を行った後のファイルをコマンドライン引数によって与えることとする．SNR[dB] を求めるプログラムをソースコード 14 に示す．このソースコードをオフライン型にした理由として，移動平均プログラムを行うことでデータ数が減ってしまうため，一つのループで完結させるのが難しい点が挙げられる．

ここで注意しなければならないのが，もとの波形に比べ，3 点移動平均や 5 点移動平均処理を行った後ではデータ数が少なくなっている点である．今回は 2 つのファイルからそれぞれ雑音を加える前の波形データと，移動平均によって雑音を抑える処理をした波形データを読み取ったが，コメント行に波形の情報があり生成する波のおおよその形がわかっていれば，ソースコード 15 のようにして波形を生成することもできる．

ソースコード 14: snr2.c の主要部

```
adjustment = atoi(argv[3]);
while (fgets(buf2, sizeof(buf2), fp2) != NULL) {
  if (buf2[0] == '#') {
```

```
4       printf("%s", buf2);
5       continue;
6     }
7     tm[sn] = atoi(strtok(buf2, ","));
8
9     aerr[sn] = atoi(strtok(NULL, ","));
10    amp[sn] = atoi(strtok(NULL, "\r\n\0"));
11    add_s += (amp[sn] - A_BIAS) * (amp[sn] - A_BIAS);
12    sn++;
13  }
14  nn = adjustment / 2;
15  while (fgets(buf1, sizeof(buf1), fp1) != NULL) {
16    if (buf1[0] == '#') {
17      printf("%s", buf1);
18      continue;
19    }
20    tm[nn] = atoi(strtok(buf1, ","));
21
22    aerr[nn] = atoi(strtok(NULL, ","));
23    edit[nn] = atoi(strtok(NULL, "\r\n\0"));
24    add_n += (edit[nn] - amp[nn]) * (edit[nn] - amp[nn]);
25    nn++;
26  }
27  snr_db = 10 * log10((add_s / sn) / (add_n / (nn - adjustment / 2)));
28  printf("%f", snr_db);
```

ソースコード 15: 波形情報が既知の場合の snr2.c の主要部

```
1   while (fgets(buf, sizeof(buf), fp) != NULL) {
2   if (buf[0] == '#') {
3     printf("%s", buf);
4     if (buf[1] == 'A') {
5       ampli = atoi(strtok((buf + 3), "\r\n\0"));
6     } else if (buf[1] == 'F') {
7       frq = atoi(strtok((buf + 3), "\r\n\0"));
8     }
9     continue;
10  }
11  tm[n] = atoi(strtok(buf, ","));
12
13  rad = tm[n] / (1000 / (double)frq) * 2 * PI;
14
15  amp[n] = (double)ampli * sin(rad) + A_BIAS;
16
17  aerr[n] = atoi(strtok(NULL, ","));
18  edit[n] = atoi(strtok(NULL, "\r\n\0"));
19
20  add_s += (amp[n] - A_BIAS) * (amp[n] - A_BIAS);
21  add_n += (edit[n] - amp[n]) * (edit[n] - amp[n]);
22  n++;
23 }
```

2.4.3 で作成した雑音が重畳された CSV ファイルを 3 点移動平均，及び 5 点移動平均を取り，それぞれの S/N 比を求める．求めた S/N 比をまとめたものを表 3 に示す.

表 3: 3 点移動平均及び 5 点移動平均を取った値の S/N 比とその変化量

	基本波形	振幅 140	周波数 5[Hz]	白色雑音の最大値 15
3 点移動平均	36.767451	35.558365	29.260603	36.767451
5 点移動平均	27.264763	27.111184	19.574977	27.264763
3 点移動平均による変化量	13.83823	9.438226	6.070099	16.99875
5 点移動平均による変化量	4.335542	0.991045	-3.615527	7.496062

これを見ると，おおよそ 5 点移動平均より 3 点移動平均のほうが S/N 比が大きくなることがわかる．つまり，3 点移動平均の方がより雑音を取り除けていると考えられる．移動平均処理を行うと，もとの波形と比べ振幅値が小さくなる．これはもとの波形の振幅が大きくなるほど影響を受けやすい．周波数 5[Hz] のときの S/N 比が小さいが，これは振幅値が最も大きくなる地点のサンプルをより多く採取してしまっているからではないかと考えられる．このデータを見ると，3 点移動平均の方が 5 点移動平均より S/N 比が大きくなっているため 3 点移動平均のほうが優れていると考えることができる.

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．

2.5.3 ステレオ音声・量子化ビット数 16 の WAVE ファイルの波形データを CSV ファイルに出力するダンププログラムを作成せよ．

ステレオ音声・量子化ビット数 16 の WAVE ファイルの波形データを CSV ファイルに出力するダンププログラムをソースコード 16 に示す.

ソースコード 16: wav2txt-s16.c

```
1  uLong read_head(FILE *fp, uShort *ch, uShort *qbit) {
2    char str[H_LEN];
3    uLong riffsize, fmtsize, smprate, datasize, bytepersec;
4    uShort fid, blksize;
5
6    fread(str, sizeof(char), H_LEN, fp);
7    if (memcmp("RIFF", str, H_LEN) != 0) return 0;
8    fread(&riffsize, sizeof(uLong), 1, fp);
9    printf("# RIFF size: %ld\n", riffsize);
10   fread(str, sizeof(char), H_LEN, fp);
11   if (memcmp("WAVE", str, H_LEN) != 0) return 0;
12   fread(str, sizeof(char), H_LEN, fp);
13   if (memcmp("fmt ", str, H_LEN) != 0) return 0;
14   fread(&fmtsize, sizeof(uLong), 1, fp);
15   printf("# fmt size: %ld\n", fmtsize);
16   fread(&fid, sizeof(uShort), 1, fp);
17   printf("# fmt ID: %d\n", fid);
```

```
18    if (fid != ID_LPCM) return 0;
19    fread(ch, sizeof(uShort), 1, fp);
20    printf("# CH: %d\n", *ch);
21    fread(&smprate, sizeof(uLong), 1, fp);
22    printf("# Sampling rate: %ld\n", smprate);
23    fread(&bytepersec, sizeof(uLong), 1, fp);
24    printf("# Bytes per sec: %ld\n", bytepersec);
25    fread(&blksize, sizeof(uShort), 1, fp);
26    printf("# Block size: %d\n", blksize);
27    fread(qbit, sizeof(uShort), 1, fp);
28    printf("# Q bit: %d\n", *qbit);
29    fread(str, sizeof(char), H_LEN, fp);
30    if (memcmp("data", str, H_LEN) != 0) return 0;
31    fread(&datasize, sizeof(uLong), 1, fp);
32    printf("# datasize: %ld\n", datasize);
33    return smprate;
34  }
35
36  int main(int argc, char **argv) {
37    double count;
38    double tm = 0;
39    short datR, datL;
40    long start_num = 0L, end_num;
41    FILE *fp;
42    uLong dsize, sampling_rate;
43    uShort ch, qbit;
44
45    if (argc < 2) {
46      fprintf(stderr, "Usage: %s file page\n", argv[0]);
47      return EXIT_FAILURE;
48    }
49    if ((fp = fopen(argv[1], "rb")) == NULL) {
50      fprintf(stderr, "File (%s) cannot open\n", argv[1]);
51      return EXIT_FAILURE;
52    }
53
54    sampling_rate = read_head(fp, &ch, &qbit);
55    if (argc > 3) {
56      start_num = atol(argv[2]) * (qbit / 8) * ch;
57      fseek(fp, start_num, SEEK_CUR);
58    }
59    if (argc == 4) {
60      end_num = atoi(argv[3]);
61    } else {
62      end_num = sampling_rate;
63    }
64    tm = (double)start_num / sampling_rate * 1000.0;
65    count = 1000.0 / sampling_rate;
66
67    while (fread(&datL, sizeof(short), 1, fp)) {
68      if (!(fread(&datR, sizeof(short), 1, fp))) break;
```

```
69
70        tm += count;
71        printf("%0.3f,%4d,%4d\n", tm, datL, datR);
72        if (tm > end_num) break;
73      }
74      fclose(fp);
75      return EXIT_SUCCESS;
76    }
```

ここで注意しなければならないのが，今回量子化ビット数が 16 なので，取り出した波形は short 型に格納しなければいけない．

仮に unsigned char 型で取り出すと図 10，int 型に格納すると図 11，unsigned short 型に格納すると図 12 のようになる．基本的に L の値も R の値も 2 バイトでないと表せない値が入っているためデータ数が 1 バイトしかない char 型に合わせ 1 バイトずつ値を取得すると元の値が負の値を取る場合や 255 以下の値を持つときに 0 や 255 などの振り切れた値を出力してしまう．

2 バイトで取り出した値を 4 バイトで表される int 型に格納する際，最下位ビットから数えて必要な分しか書き換えないため，今回だと最上位ビットから数えて 8 ビット分の値がわからず N/A となる．

また，unsigned short 型の変数に値を格納する場合，本来負の値を取るはずのデータが別の正の値を取ってしまうため負の値の部分が上にスライドしたような形になっている．

ソースコード 16 のダンププログラムを使用して，出力した chimes.wav の波形を図 13 に示す．

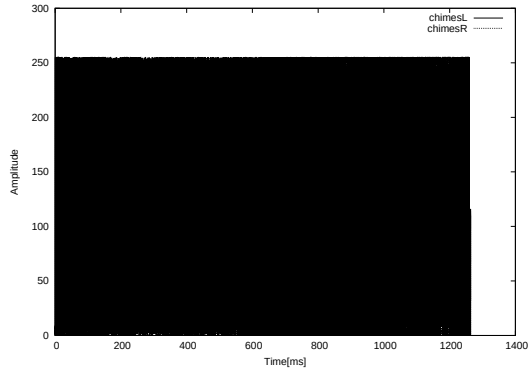

図 10: unsigned char 型に収まるように波形データを取り出した chimes.wav の波形

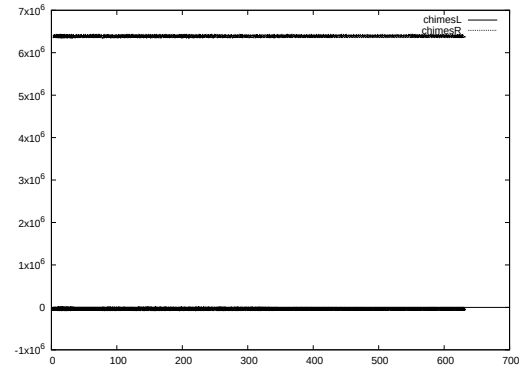

図 11: 取り出した波形データを int 型の変数に格納した chimes.wav の波形

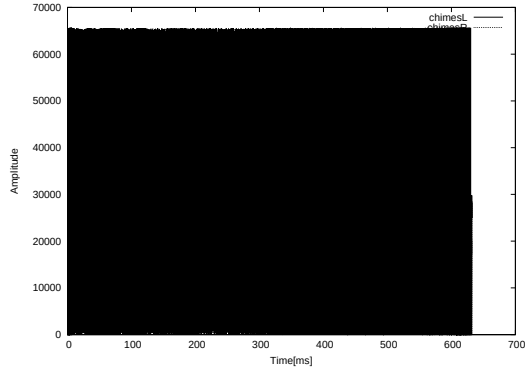

図 12: 取り出した波形データを unsigned short 型の変数に格納 chimes.wav の波形

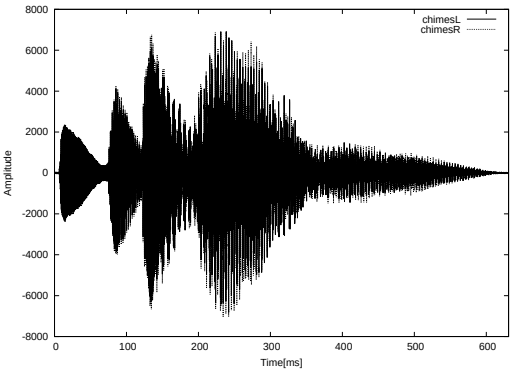

図 13: chimes.wav の出力波形

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．また，図 13 を見ると，振幅の大きさが L と R でほんの少し違うことがわかる．これは L と R で少し違う大きさの音を出し，音に立体感を出しているのだと思われる．

3 課題

ここではソースコード 17 に示すようなヘッダファイルが読み込まれているものとする．

ソースコード 17: 課題のヘッダファイル

```
1  #define T_END 1000
2  #define DT 1
3  #define A_BIAS 0x80
4  #define FRQ 3
```

周期関数 f_1，f_2，f_3 を実装したプログラムをそれぞれソースコード 18，19，20 に示す．

ソースコード 18: 周期関数 $f_1(t)$ の主要部

```
amp = atoi(argv[1]);

m = atoi(argv[2]);
for (int t = 0; t <= T_END; t += DT) {
  rad = t * FRQ * 2 * PI / 1000;
  f_1 = 0;
  for (count = 1; count <= m; count++) {
    f_1 += sin(count * rad) / count;
  }
  f_1 = 2 / PI * amp * f_1 + A_BIAS;
  printf("%4d,%4d\n", t, (int)f_1);
}
```

ソースコード 19: 周期関数 $f_2(t)$ の主要部

```
amp = atoi(argv[1]);

m = atoi(argv[2]);
for (int t = 0; t <= T_END; t += DT) {
  rad = t * FRQ * 2 * PI / 1000;
  f_2 = 0;
  for (count = 1; count < m; count++) {
    f_2 += sin((2 * count - 1) * rad) / (2 * count - 1);
  }
  f_2 = 4 / PI * amp * f_2 + A_BIAS;
  printf("%f,%f\n", rad, f_2);
}
```

ソースコード 20: 周期関数 $f_3(t)$ の主要部

```
amp = atoi(argv[1]);

m = atoi(argv[2]);
for (int t = 0; t <= T_END; t += DT) {
  rad = t * FRQ * 2 * PI / 1000;
  f_3 = 0;
  for (count = 1; count < m; count++) {
    f_3 += sin(count * PI / 2) * sin(count * rad) / (count * count);
  }
  f_3 = 2 / (PI * PI) * amp * f_3 + A_BIAS;
  printf("%f,%f\n", rad, f_3);
}
```

これらのソースコードに振幅 100 として，試行回数 m が 1，10，100，10000 の場合について波形を出力する．周期関数 $f_1(t)$，$f_2(t)$，$f_3(t)$ の波形をそれぞれ 14，15，16 に示す．

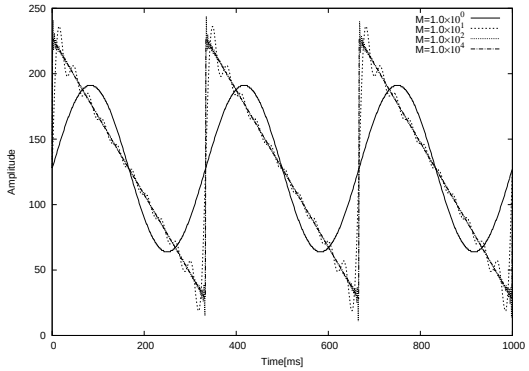

図 14: 振幅 100，周波数 3[Hz] の周期関数 $f_1(t)$

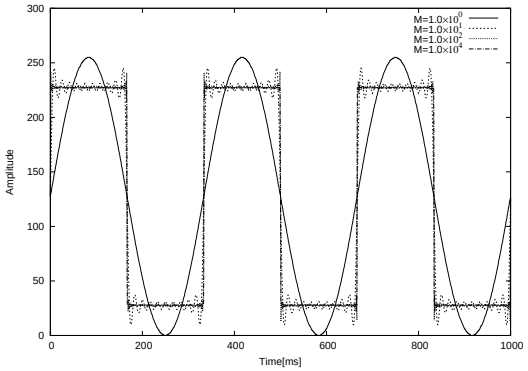

図 15: 振幅 100，周波数 3[Hz] の周期関数 $f_2(t)$

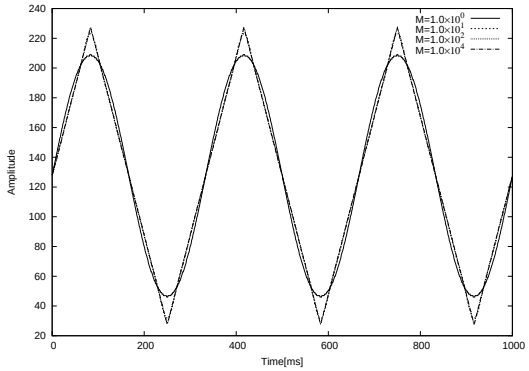

図 16: 振幅 100，周波数 3[Hz] の周期関数 $f_3(t)$

これらを見ると，$m = 10$ のときは振動している様子がよく分かる．また，$m = 100$ のときは殆どの場所で値が収束していて，極値付近のみ振動している様子が確認できる．極値の波形を観察するため，$[0 : 10]$ の範囲を切り出した波形をそれぞれ図 17，18，19 に示す．

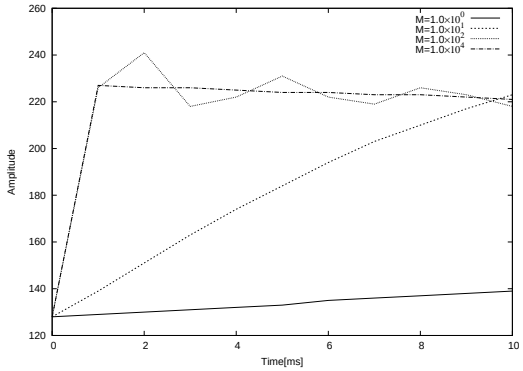

図 17: [0 : 10] の振幅 100，周波数 3[Hz] の周期関数 $f_1(t)$

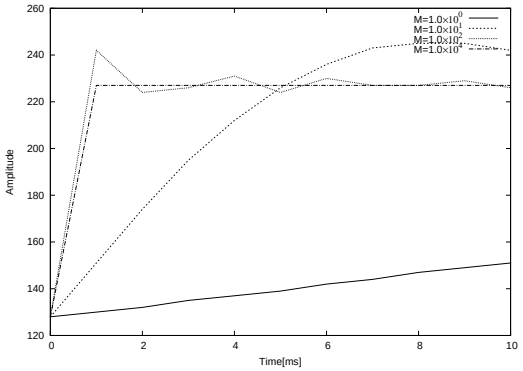

図 18: [0 : 10] の振幅 100，周波数 3[Hz] の周期関数 $f_2(t)$

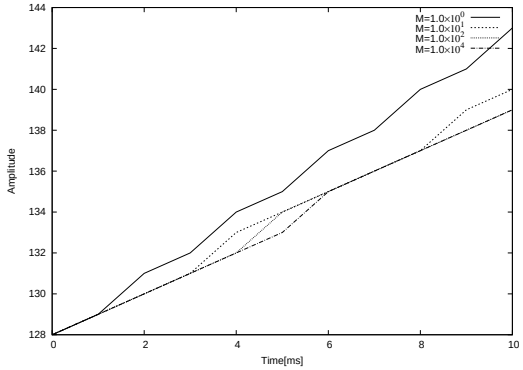

図 19: [0 : 10] の振幅 100，周波数 3[Hz] の周期関数 $f_3(t)$

これらを見ると $m = 100$ のとき，極値付近で減衰振動を行っていることがよくわかる．m の値が増えるほど，理想的な波形に近づいていることがわかる．$m = 10000$ の時のそれぞれの波形に最大値を 10 に設定した白色雑音を加え S/N 比を求める．また，雑音を加えた波形を 3 点移動平均と 5 点移動平均プログラムを用いて雑音を除去し，これらと元の波形の S/N 比を求める．求めた S/N 比を表 4 にまとめる．

表 4: 周期関数 $f_i(t)$ の S/N 比

	f_1	f_2	f_3
S/N 比	20.835381	25.806504	21.125344
3 点移動平均	24.380427	29.52961	41.946036
5 点移動平均	25.66529	30.426056	28.962362
3 点移動平均による変化量	3.545046	3.723106	20.820692
5 点移動平均による変化量	4.829909	4.619552	7.837018

これを見ると，f_1，f_2 は 3 点移動平均と 5 点移動平均の除去性能に大きな差は見られないが，f_3 では圧倒的に 3 点移動平均の値が大きくなっている．そのため，2.4.4 の結果もと合わせて考えると，3 点移動平均のほうが優れていると考えることができる．

4 発展課題

モノラル音声の WAVE ファイルをステレオ音声の WAVE ファイルに変換するプログラムを作成した．このプログラムをソースコード 21 に示す．

なお，入力として量子化ビット数 8 のモノラル音声の WAVE ファイルとそれをステレオ音声に変換して出力するための WAVE ファイルを与え，L から R に対して遅らせる位相の大きさはマクロ定義によって与えるものとする．また，変換後の量子化ビット数は 8 とし，ずらした後の空いてしまった R の値は 0 とする．

ソースコード 21: 発展課題

```
1  #include <ctype.h>
2  #include <math.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6
7  #define H_LEN 4
8  #define ID_LPCM 1
9  #define CH 2
10 #define DT 4
11 typedef unsigned short uShort;
12 typedef unsigned long uLong;
13
14 uLong header_data(FILE *fp, FILE *fp_write, short *q_bit, int *data_size) {
15   char str[H_LEN];
16   unsigned int h_len = H_LEN;
17   uLong riff_size, fmt_size, sampling_rate, data_speed;
18   uShort fid, block_size, ch;
19
20   fread(str, sizeof(char), h_len, fp);
21   if (memcmp("RIFF", str, h_len) != 0) return 0;
22   fread(&riff_size, sizeof(uLong), 1, fp);
23   printf("# RIFF size: %ld\n", riff_size);
24   fread(str, sizeof(char), h_len, fp);
25   if (memcmp("WAVE", str, h_len) != 0) return 0;
26   fread(str, sizeof(char), h_len, fp);
27   if (memcmp("fmt ", str, h_len) != 0) return 0;
28   fread(&fmt_size, sizeof(uLong), 1, fp);
29   printf("# fmt size: %ld\n", fmt_size);
30   fread(&fid, sizeof(uShort), 1, fp);
31   printf("# fmt ID: %d\n", fid);
32   if (fid != ID_LPCM) return 0;
33   fread(&ch, sizeof(uShort), 1, fp);
34   printf("# CH: %d\n", ch);
35   fread(&sampling_rate, sizeof(uLong), 1, fp);
36   printf("# Sampling rate: %ld\n", sampling_rate);
37   fread(&data_speed, sizeof(uLong), 1, fp);
38   printf("# Bytes per sec: %ld\n", data_speed);
39   fread(&block_size, sizeof(uShort), 1, fp);
40   printf("# Block size: %d\n", block_size);
```

```
41    fread(q_bit, sizeof(uShort), 1, fp);
42    printf("# Q bit: %d\n", *q_bit);
43    fread(str, sizeof(char), h_len, fp);
44    if (memcmp("data", str, h_len) != 0) return 0;
45    fread(data_size, sizeof(uLong), 1, fp);
46    printf("# datasize: %ld\n", *data_size);
47    *data_size *= 2;
48    ch = CH;
49    fwrite("RIFF", sizeof(char), 4, fp_write);
50    fwrite(&riff_size, sizeof(int), 1, fp_write);
51    fwrite("WAVE", sizeof(char), 4, fp_write);
52    fwrite("fmt ", sizeof(char), 4, fp_write);
53    fwrite(&fmt_size, sizeof(int), 1, fp_write);
54    fwrite(&fid, sizeof(short), 1, fp_write);
55    fwrite(&ch, sizeof(short), 1, fp_write);
56    fwrite(&sampling_rate, sizeof(int), 1, fp_write);
57    fwrite(&data_speed, sizeof(int), 1, fp_write);
58    fwrite(&block_size, sizeof(short), 1, fp_write);
59    fwrite(q_bit, sizeof(short), 1, fp_write);
60    fwrite("data", sizeof(char), 4, fp_write);
61    fwrite(data_size, sizeof(int), 1, fp_write);
62    return sampling_rate;
63  }
64
65  int main(int argc, char **argv) {
66    int sampling_rate, count = 0, data_size;
67    unsigned short data_dt[DT] = {0};
68    unsigned short datL, datR;
69    uShort q_bit;
70    FILE *fp;
71    FILE *fp_write;
72    if (argc < 2) {
73      fprintf(stderr, "Usage: %s file page\n", argv[0]);
74      return EXIT_FAILURE;
75    }
76    if ((fp = fopen(argv[1], "rb")) == NULL) {
77      fprintf(stderr, "File (%s) cannot open\n", argv[1]);
78      return EXIT_FAILURE;
79    }
80    if ((fp_write = fopen(argv[2], "wb")) == NULL) {
81      fprintf(stderr, "File (%s) cannot open\n", argv[2]);
82      return EXIT_FAILURE;
83    }
84    int countp = 0;
85    sampling_rate = header_data(fp, fp_write, &q_bit, &data_size);
86    while ((data_dt[0] = fgetc(fp)) != EOF) {
87      if (countp * 2 >= data_size) break;
88      datL = data_dt[0];
89      datR = data_dt[DT - 1];
90      fwrite(&datL, sizeof(unsigned char), 1, fp_write);
91      fwrite(&datR, sizeof(unsigned char), 1, fp_write);
```

```
92        printf("%d,%d\n", datL, datR);
93        for (count = 1; count < DT; count++) {
94          data_dt[DT - count] = data_dt[DT - count - 1];
95        }
96        countp++;
97      }
98      fclose(fp);
99      fclose(fp_write);
100     return EXIT_SUCCESS;
101   }
```

これに，2.5.3 で録音した振幅 100，周波数 100[Hz] 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] で一秒の長さ記録した波形を入力として用いる．出力する WAVE ファイルの量子化ビット数を 8，L から R に対して遅らせる位相の大きさを 4 とした WAVE ファイルを出力した．聞いてみたところ，程よい重低音が聞こえ最初の一瞬だけ右耳からプツッと言った音がした．これで良いと思われるが，ソースコード 16 を使用して波形を確認する．ソースコード 21 で作成した波形を図 20 に示す．

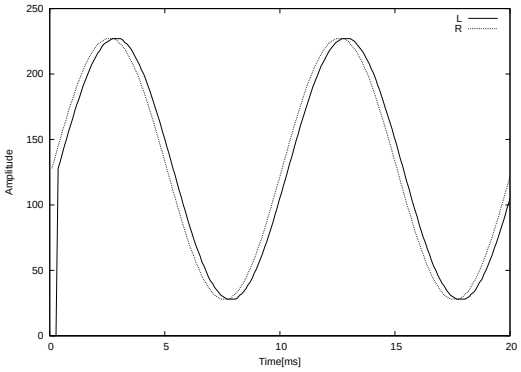

図 20: モノラル音声を変換したステレオ音声の波形

この通り振幅 100，周波数 100[Hz] の正弦波が L に，L から 4 データ分遅れて同様の正弦波が R に出力されていることがわかる．今回はしなかったが，WAVE ファイルから取り出す際 WAVE ファイルのヘッダ方法から取り出した量子化ビット数を用いることで，元の WAVE ファイルの量子化ビット数によらず処理を行うことができる．

5　あとがき

5.1　感想

今回の実験は前回の実験を基礎としてより発展的なことを行った．これらの実験を通して，実験内容だけでなくファイル操作に関係する関数や変数の型について理解を深めることができてよかったと思う．

5.2 要望

変数名や関数名を決めているが，自分が決めている変数名や関数名がどの程度正しいかを知りたいため，ある程度の指標としてコーディング規約についての記述があるとなお良いと思った.

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
2. 国際単位系（SI）第 9 版（2019）日本語版について p.120 (https://unit.aist.go.jp/nmij/public/report/SI_9th/pdf/SI_9th_%
3. 音ファイル（拡張子：WAV ファイル）のデータ構造について (https://www.youfit.co.jp/archives/1418)