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 32 番　平田 蓮

提出日　　：2020 年 6 月 18 日

実験日　　：2020 年 5 月 21 日, 5 月 28 日, 6 月 4 日, 6 月 11 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起訴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, 音声データの入出力プログラムを作成する.

2　周期関数の生成と可視化

正弦波のように一定周期ごとに同じ波形が繰り返される関数を**周期関数**と呼ぶ. よく知られている周期関数として, のこぎり波などがある. 本節では周期関数に関する演習を行う.

■**演習 1-1**　任意の孤度 r を区間 $[0:2\pi]$ に変換する関数 `rad` を作成せよ.

作成した関数をソースコード 1 に示す.

ソースコード 1　rad.c

```
1 double rad(double r) {
2     if (r >= 0) {
3         return fmod(r, 2 * PI);
4     } else {
5         return 2 * PI - fmod(-r, 2 * PI);
6     }
7 }
```

r が正のときは 2π との剰余を取る. r が負のときは, r を正数に変換し, 2π と剰余をとったものを 2π から引くことで変換している.

図 1 に横軸に r, 縦軸に `rad(r)` を取ったグラフを示す. この図から, 変換がうまく行われていることがわかる.

■**演習 1-2**　任意の孤度 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.

作成した関数をソースコード 2 に示す.

ソースコード 2　saw.c

```
1 double saw(double r) {
2     return 1 - rad(r) / PI;
3 }
```

まず与えられた孤度 r を演習 1-1 で実装した `rad` を使い $[0:2\pi]$ の範囲に変換する.

r が 2π の倍数の場合に 1 を取り, そこから傾き $-\frac{1}{\pi}$ の形を繰り返すので, 上記のように実装することができた.

図 2 にグラフを示す. うまくのこぎり波が現れていることがみてとれる.

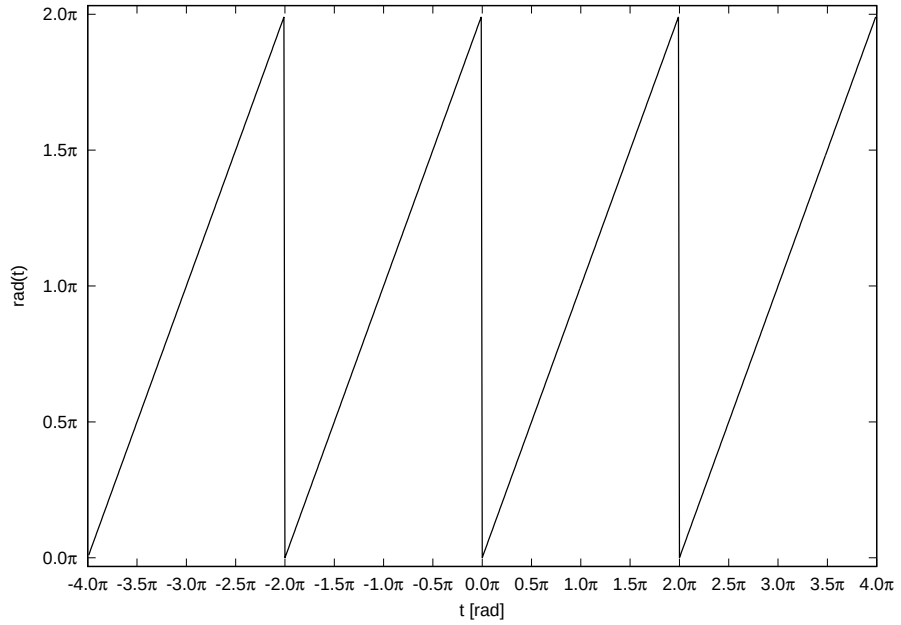

図 1　`rad` による変換

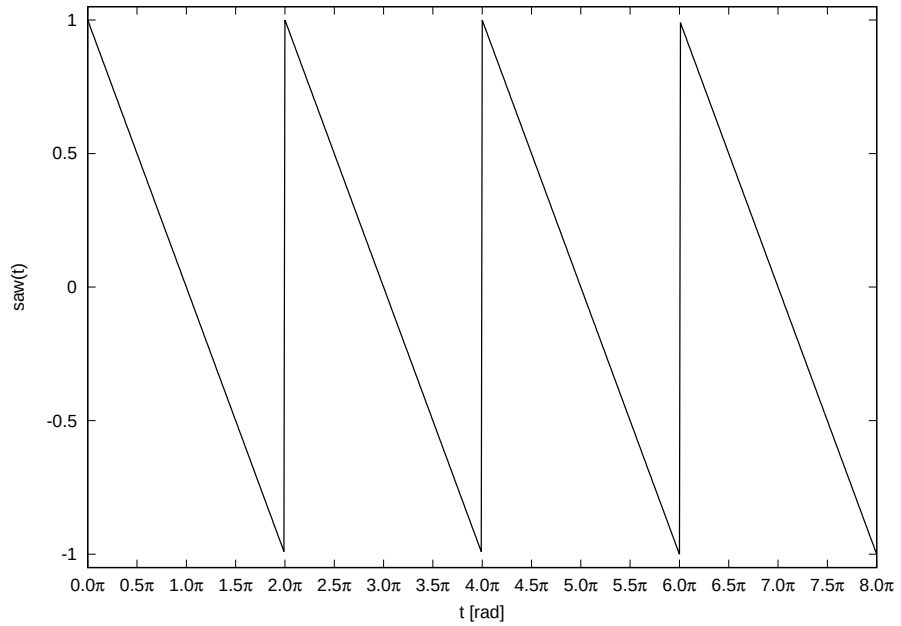

図 2　`saw` による変換

3　サンプリング

アナログ信号波形をディジタルデータに変換する手法に**サンプリング**がある．本節では演習問題を通して実際にサンプリングの様子を観察する．

■**演習 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 4Hz のデータを出力リダイレクションを使って sin100f4.csv に出力せよ. この sin100f4.csv を gnuplot でグラフ化し, サンプリングの効果を確認せよ.

まず, 完成させたリスト 2 のコード, sin10af1.c を出力部分を抜粋して掲載する.

ソースコード 3　sin10af1.c

```
1  int t;
2  double amp, frq, rad, vin;
3  unsigned char vout;
4
5  for (t = 0; t <= TEND; t += DT) {
6      rad = 2 * PI * frq * t / 1000.0;
7      vin = amp * sin(r) + BIAS;
8      if (vin > 255) {
9          vout = 255;
10     } else if (vin < 0) {
11         vout = 0;
12     } else {
13         vout = vin;
14     }
15     printf("%4f, %4d\n", t / 1000.0, vout);
16 }
```

4 行目から 10 行目に渡って, クリッピングという処理を施してある. クリッピングをすることで, 出力用の変数 (今回は 8bit の char 型) に収まらない値を切り捨て, 波形を維持することができる.

図 3 にサンプリングしたデータを示す. 実際に正弦波をサンプリングできていることがわかる.

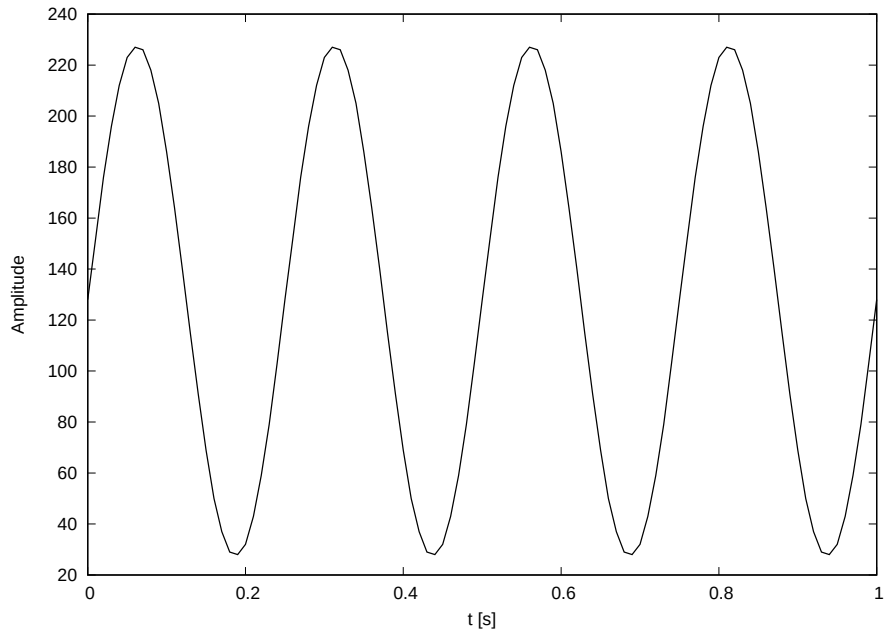

図 3　振幅 100, 周波数 4Hz の正弦波

■**演習 2-2**　振幅 150, 周波数 4Hz のデータ sin150f4.csv 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．

今回は振幅が 150 なので, char 型に収まらない数値をサンプリングすることになる．そこで, 演習 2-1 で実装したクリッピングが働く．

図 4, 5 にクリッピングを施した前と後の波形を示す．この図からクリッピングの効果がみて取れる．クリッピングをしていない波形では, オーバーフローした値が波形の逆側に飛んでしまい, 波形が崩れているが, クリッピングを施した方ではオーバーフローした値が切り捨てられ, 波形が維持できている．

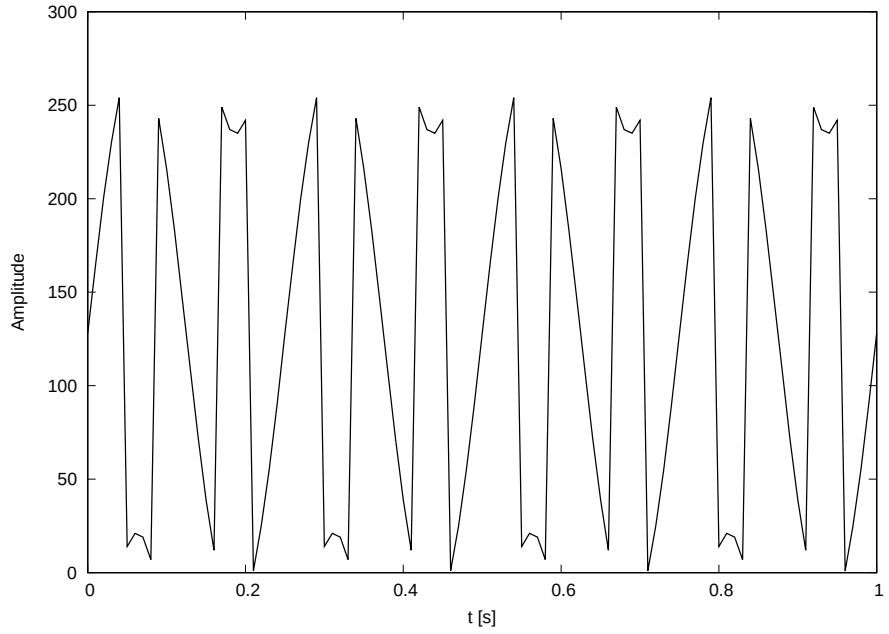

図 4　振幅 150, 周波数 4Hz の正弦波 (クリッピングなし)

■**演習 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形に見えることがある．この現象を実際に観察せよ．

今回はサンプリング周波数が 100Hz であるので, 50Hz を超える周波数の波はうまくサンプリングをすることができない．

図 6 に 98Hz の正弦波をサンプリングしたものを示す．実際には 98Hz であるはずが, 2Hz の波形のように見える．

参考に, 上の波形にサンプリング周波数を上げて計測した本来の 98Hz の波形を重ねたものを図 7 に示す．

4　信号処理プログラムの分類

信号処理プログラムは, オンライン型をオフライン型の二種類に分類することができる．データを一つ取り込むたびに逐次処理を行うオンライン型に対して, オフライン型はデータを一定数取り込んだの後に一括処理を行う．

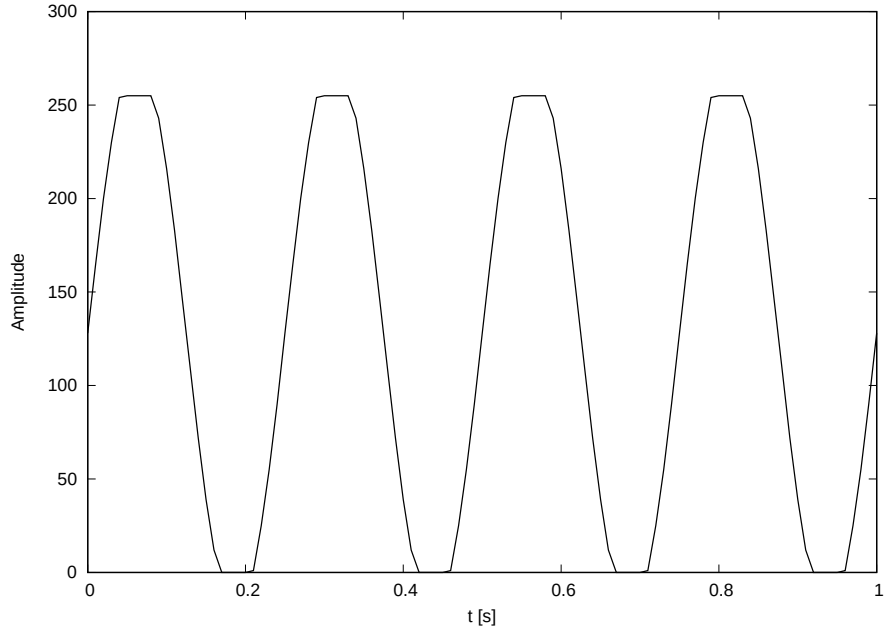

図 5　振幅 150, 周波数 4Hz の正弦波 (クリッピングあり)

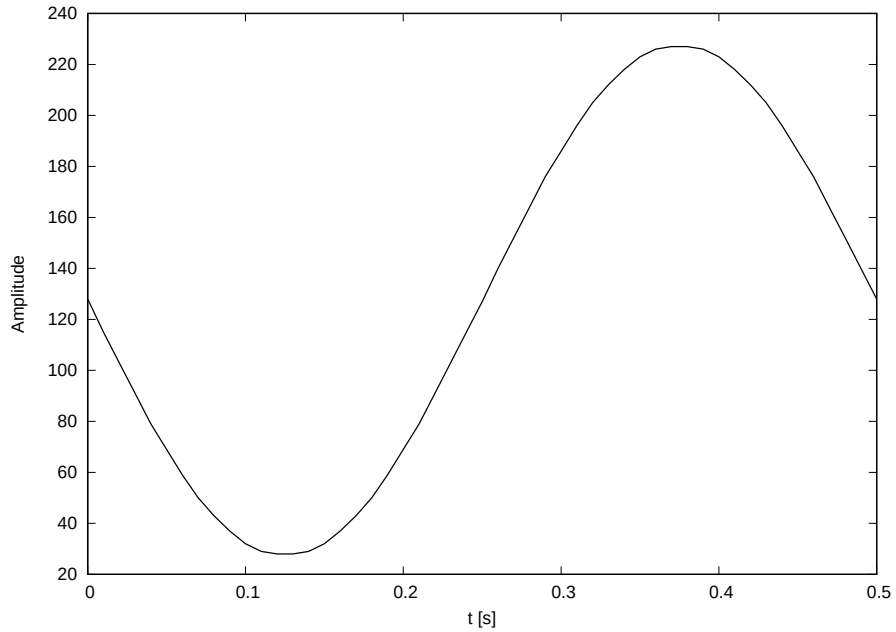

図 6　周波数 98Hz の正弦波 (サンプリング周波数:100Hz)

5　雑音除去

信号に重畳された雑音を取り除く処理は, 極めて基本的な信号処理である. 演習 3 では雑音の処理に関する演習を行う.

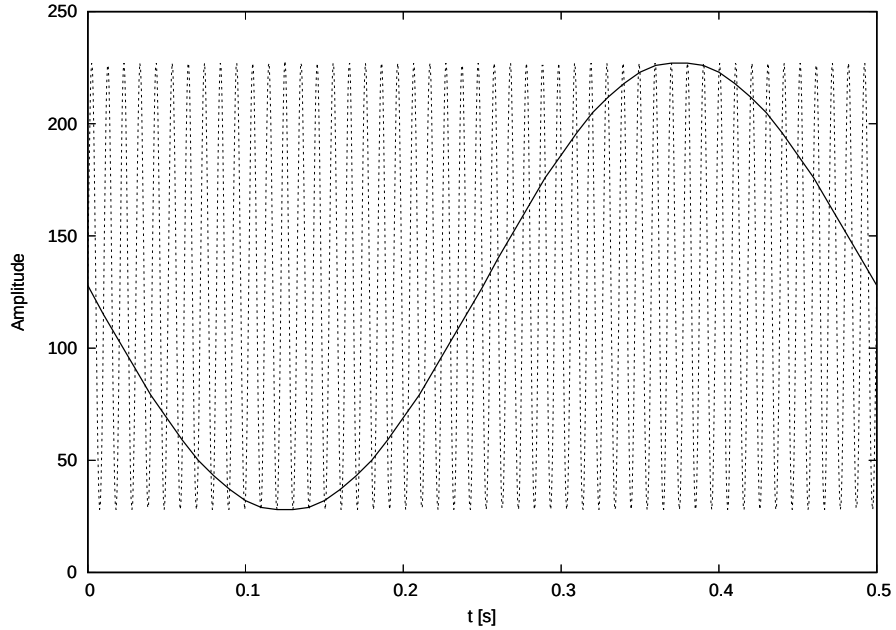

図 7　周波数 98Hz の正弦波 (サンプリング周波数:100Hz, 1000Hz)

■**演習 3-1**　リスト 3 を参考に, コマンドライン引数で元信号の CSV ファイルと, 白色雑音の最大振幅を与えたとき, 白色雑音を加えるオフライン型のプログラム add-wn2.c を作成せよ.

作成したプログラムの信号処理部分を抜粋してソースコード 4 に示す.

ソースコード 4　add-wn2.c

```
1  double tm[DATANUM];
2  int amp[DATANUM], nmax;
3  double err;
4
5  for (int n = 0; n <= DATANUM; n++) {
6      err = nmax * (2.0 * rand() / RAND_MAX - 1.0);
7      amp[n] += ROUND(err);
8
9      printf("%4d, %4d\n", tm[n], amp[n]);
10 }
```

通常の sin 波に最大振幅 10 の白色雑音を加えたものを図 8 に示す. 図から, 適当な雑音が加えられていることがわかる.

■**演習 3-2**　3 点単純移動平均プログラム mvave3-?.c を作成せよ (?はオンライン型は 1, オフライン型は 2).

図 5, 6 にそれぞれオンライン型とオフライン型のソースコードを抜粋して示す.

ソースコード 5　mvave3-1.c

```
1  int tm, __amp = 0, _amp = 0, amp, _tm;
2
```

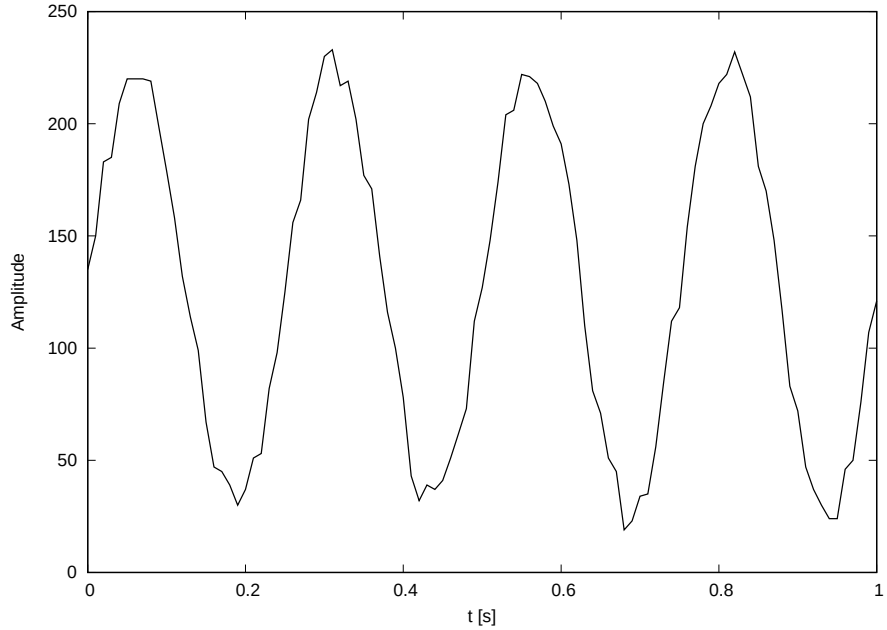

図 8　最大振幅 10 の正弦波を加えた sin 波

```
3  while (fgets(buf, sizeof(buf), fp) != NULL) {
4      if (buf[0] == '#') continue;
5
6      tm = atoi(strtok(buf, ","));
7      amp = atoi(strtok(NULL, "\r\n\0"));
8
9      if (_amp == 0) {
10         _amp = amp;
11         continue;
12     }
13     if (__amp == 0) {
14         __amp = _amp;
15         _tm = tm;
16         continue;
17     }
18
19     vout = ROUND((__amp + _amp + amp) / 3.0);
20     printf("%4d, %4d\n", _tm, (int)vout);
21
22     __amp = _amp;
23     _amp = amp;
24     _tm = tm;
25 }
```

ソースコード 6　mvave3-2.c

```
1  for (int n = 1; n < DATANUM; n++) {
2      vout = ROUND((amp[n - 1] + amp[n] + amp[n + 1]) / 3.0);
```

```
3
4       printf("%4d, %4d\n", tm[n], (int)vout);
5   }
```

オンライン型のソースコードでは, 3 点平均を計算するために 2 個前までのデータ, また, 出力用に一個前の t を毎回更新している.

これらのプログラムで図 8 から雑音を取り除いた波形を図 9 に示す.

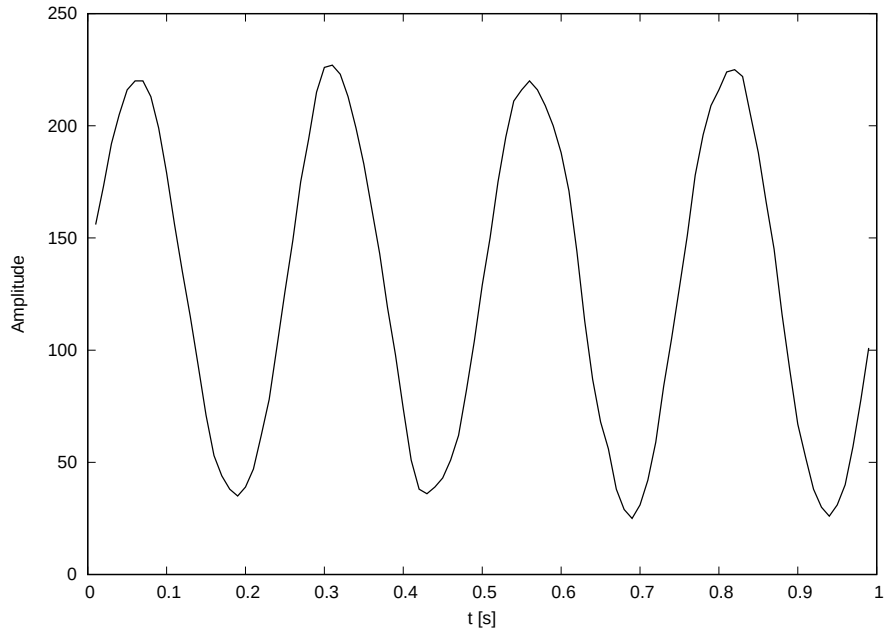

図 9　雑音を取り除いた波形

3 点平均を取る過程で始点と終点のデータが一つずつ欠けてしまっているが, 波形は概ね元のものに近づいている.

6　時系列データの解析

周期的なアナログ信号 $x(t)$ に由来する時系列データ $x_i; i = 1, 2, \ldots, N$ が与えられたとき, 元の信号の基本周波数, 振幅, 位相などを求めることを信号解析を呼ぶ. 本実験では信号の**最小値**, **最大値**, **平均値**, **標準偏差**, **最大振幅**, **実効値**を求める.

■平均値と標準偏差　時系列データ $x_i; i = 1, 2, \ldots, N$ の平均値 $\bar{x}$ は次式で定義される:

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

標準偏差 σ は分散 σ^2 の正の平方根として定義される:

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

この式では平均値が既知である必要があり, オンライン処理に適用することができない. オンライン処理で標準偏差や分散を計算したい場合は, 変形した次式を用いる:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - \bar{x}^2 \tag{3}$$

■**演習 4-1**　式 (2) を式 (3) に変形する過程を示せ.

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

式 (1) より, $\frac{2}{N}\bar{x}\sum_{i=1}^{N} x_i = 2\bar{x}^2$ なので,

$$\begin{aligned}
\sigma^2 &= \frac{1}{N}\sum_{i=1}^{N} x_i^2 - \frac{2}{N}\bar{x}\sum_{i=1}^{N} x_i + \frac{\bar{x}^2}{N}\sum_{i=1}^{N} 1 \\
&= \frac{1}{N}\sum_{i=1}^{N} x_i^2 - 2\bar{x}^2 + \bar{x}^2 \\
&= \frac{1}{N}\sum_{i=1}^{N} x_i^2 - \bar{x}^2
\end{aligned}$$

■**演習 4-2**　コマンドライン引数で与えられた CSV ファイルについて, 最小値, 最大値, 平均値, 標準偏差, 最大振幅, 実効値を求めるプログラム stat?.c を作成せよ (?はオンライン型は 1, オフライン型は 2).

作成したプログラムをソースコード 7, 8 に示す.

ソースコード 7　stat1.c

```
1  int tm, amp;
2  char buf[BUFSIZE];
3  double
4      min_val = INF, max_val = -INF,
5      sum = 0, mean, sum_squ = 0,
6      std_dev, data_num = 0, max_amp, rms = 0;
7  FILE *fp;
8
9  while (fgets(buf, sizeof(buf), fp) != NULL) {
10     if (buf[0] == '#') continue;
11
```

```
12      data_num += 1;
13
14      tm = atoi(strtok(buf, ","));
15      amp = atoi(strtok(NULL, "\r\n\0"));
16
17      if (min_val > amp) min_val = amp;
18      if (max_val < amp) max_val = amp;
19
20      sum += amp;
21
22      sum_squ += amp * amp;
23
24      rms += amp - BIAS;
25  }
26  fclose(fp);
27
28  mean = sum / data_num;
29  std_dev = sqrt(sum_squ / data_num - mean * mean);
30
31  if (BIAS - min_val > max_val - BIAS) {
32      max_amp = BIAS - min_val;
33  } else {
34      max_amp = max_val - BIAS;
35  }
36
37  rms = sqrt(sum_squ / data_num - 2 * mean * max_amp + max_amp * max_amp);
38
39  printf(
40      "Min: %f, Max: %f, Mean: %f, Std Deviation: %f, Max Amplitude: %f, RMS: %f\n",
41          min_val, max_val, mean, std_dev, max_amp, rms
42  );
```

ソースコード 8　stat2.c

```
1   int tm[DATANUM], amp[DATANUM];
2   char buf[BUFSIZE];
3   double
4       min_val = INF, max_val = -INF,
5       sum = 0, mean, sum_squ = 0, std_dev,
6       data_num = 0, max_amp, rms;
7   FILE *fp;
8
9   for (int n = 0; n <= DATANUM; n++) {
10      if (min_val > amp[n]) min_val = amp[n];
11      if (max_val < amp[n]) max_val = amp[n];
12
13      sum += amp[n];
14
```

```
    sum_squ += amp[n] * amp[n];

    rms += amp - BIAS;
}

mean = sum / data_num;
std_dev = sqrt(sum_squ / data_num - mean * mean);

if (BIAS - min_val > max_val - BIAS) {
    max_amp = BIAS - min_val;
} else {
    max_amp = max_val - BIAS;
}

rms = sqrt(sum_squ / data_num - 2 * mean * max_amp + max_amp * max_amp);

printf(
    "Min: %f, Max: %f, Mean: %f, Std Deviation: %f, Max Amplitude: %f, RMS: %f\n",
        min_val, max_val, mean, std_dev, max_amp, rms
);
```

7 Windows WAVE ファイルの解析 · 加工

この節では, Windows 標準のサウンドフォーマットである WAVE ファイルを, 自作の C プログラムで読み書きをする.

参考文献

[1] Electro-ワンチップマイコン http://laboratory.sub.jp/ele/13.html/