SuperCon2017 本選課題

オーディオデータを近似的に圧縮しよう

1　背景

CD などのデジタルオーディオでは、元となるアナログ信号を PCM(Pulse Code Modulation) という方式によって離散的なデジタル信号に変換する。ここでサンプリング周波数とビット深度という概念を説明しよう。サンプリング周波数とはアナログ信号を一定時間間隔ごとに数値データ化するその時間間隔の逆数である。たとえば CD ではサンプリング周波数が 44100Hz だが、これは元のアナログ信号を 1 秒間に 44100 回数値データ化していることを意味している。ビット深度は本来連続値である音量データを何ビットの整数で表現するかを表す。CD では 16 ビットである。これは音の最大値と最小値の間を 2^{16} に分割することを意味している (つまり、音の大きさは-32768 から +32767 までの整数で表現され、無音が 0 である)。以下の問題では CD と同じサンプリング周波数 44100Hz、ビット深度 16 ビットの PCM データを扱おう。つまり、1 秒間の音のデータは 44100 個の 16 ビット整数で表現される。

さて、PCM データをさらに圧縮することを考えたい。そこでここではビット深度の代わりに信号の変化を「前の時刻に比べて数値が増えるか減るか」の 1 ビット情報で表すことを考えてみよう。元のサンプリング周波数よりも充分に細かい時間分解能でこれを行うと、元の PCM データをほぼ完全に再現できる。実際にこれは一部のデジタル・オーディオ・アンプで使われている方法である。しかし、ここではデータ圧縮を目的として、時間間隔は CD と同じままとする。この場合、データ量は 1/16 に減る代わりに、元の PCM データを完全には再現できないことは明らかである。このように完全には元のデータを再現できないデータ圧縮法を「不可逆圧縮」と呼ぶ。

今回の目的は、与えられた PCM データを上で説明した 1 ビットのデータ列でなるべくよく近似することである。元の PCM データをもとに、数値を増やすときは 1、減らす時は 0 となる 1 ビットの数列を生成する。これを「エンコード」(符号化) と呼ぼう。増減の 1 ステップがどれだけの数値変化に対応するかは自由に決めるが、この数値はデータ全体を通して固定である。

近似の「よさ」の判断基準が必要である。そのためにエンコードされた 1 ビット・データ列から音のデータに変換する。これを「デコード」(復号) と呼ぼう。単純にデコードしただけでは、元データに比べてぎざぎざしたものが得られるのは明らかだろう。これは「高周波ノイズ」として音に影響する。そこで、デコードに際しては「平滑化」を行うことにする。つまり、適当な時間幅で音量を平均してしまうのである。これは高周波をカットする「ローパスフィルター」の一種である。近似のよさを評価するために、デコードされた音と元の PCM の音の両方を平滑化し、ふたつの「同時刻での差の絶対値」を全時刻に渡って足したものが小さければ小さいほどよいものとしよう。

2 問題

問題: エンコード:
元の PCM データ列を

$$P_i, i = 0, \ldots, N-1$$

とする。i はデータの番号であり、各データは 1/44100 秒間隔でサンプルされた音量を表す。またエンコードされた 1 ビットデータ列を

$$B_i, i = 0, \ldots, N-2$$

としよう (変化なので一個少ないことに注意)。B_i は 0 または 1 のいずれかである。
デコード:
B からデコードされる平滑化前の音のデータは

$$S_i = P_0 + A\sum_{k=0}^{i-1}(2B_k - 1)$$

である。A は任意に設定した「増減幅」を表す整数である。ただし、S_i が-32768 以下になる場合は-32768、+32767 以上になる場合は +32767 となり (数値の「頭打ち」であり、一般にクリッピングと呼ばれる)、それ以前の履歴は引き継がれない。たとえば、$A = 100$ として、クリッピングなしにデコードしたときに 32700,32800,32900,32800,32700,32600 となるはずの 1 ビット・データは、実際にはクリッピングによって、32700,32767,32767,32667,32567,32467 となる。
平滑化は注目している時刻とその前後 5 点ずつの計 11 点のデータを平均する (おおむね、4000Hz 以上の高周波をカットすることに相当する)。ただし、ここでは近似のよさを表せればいいだけなので、平均ではなく単なる和でよいことにしよう。つまり、PCM とデコードされたデータそれぞれの平滑化された i 番目の音を

$$\bar{P}_i = \sum_{k=i-5}^{i+5} P_k$$

$$\bar{S}_i = \sum_{k=i-5}^{i+5} S_k$$

とする。このとき

$$\Delta = \sum_{k=5}^{N-6} |\bar{P}_k - \bar{S}_k|$$

を「近似のよさ」とする (話を簡単にするために最初と最後の 5 点は扱わないことにする)。この Δ がなるべく小さくなるような 1 ビットデータ列 B_j と増減幅 A を求めるためのプログラムを作るのが問題である。

3 詳細

1. 問題として与えられるのはサンプリング周波数 44.1kHz, ビット深度 16 ビットの PCM データ列である (もっとも、問題を解くに当たって必要なのはデータ点の数 N であってサンプリング周波数は必要ない)。
2. データ点の数 N は問題ごとに違うが、最大は $N = 5292000$(2 分以内) である。
3. 与えられる PCM データは-32768 から +32767 の範囲に収まっている。
4. 増減幅 A は 32767 以下の正の整数である。この値は問題ごとに最適なものを求める。

4 勝利条件

1. 提出されたプログラムで問題を 5 題解く。なお、計算時間の上限は 5 題合計で 10 分とする。第一問から順に計算していって、計算時間の上限に達した時点で、計算途中であっても終了とする。なお、入出力に要した時間は計算時間から除かれる。
2. 各問について、解答から Δ を計算し、Δ が小さい順に 1 位から 5 位まで順位をつける。なお、Δ が同じであれば (最適解が求まればそうなる)、その計算の実行時間が短いほど上位とする (並列化を使うので CPU 時間ではなく、経過時間である)。なお、秒単位まで実行時間が一致する場合は同順位とする。
3. 各問ごとに 1 位から 5 位に 5 点から 1 点の点数を与え、点数の合計で総合順位を決定する。

5 ヘッダーファイル

配布するコンテスト専用ヘッダーファイル sc17.h には以下のように定数・配列・関数が定義されている。

```
#define MAXN 882000
void SC_input();
void SC_output(int a);
int SC_p[MAXN];
int SC_s[MAXN];
int SC_n;
void SC_input()
{ int n,data;
n=0;
while(scanf("%d",&data) != EOF){SC_p[n]=data; ++n;}
SC_n=n;
}
void SC_output(int a)
{ int i;
printf("%d\n",a);
for(i=0;i<SC_n-1;++i)printf("%d ",SC_s[i]);
```

```
printf("\n");
}
```

このヘッダーファイルをプログラム冒頭で

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "sc17.h"
```

などのようにインクルードして使う。

1. 関数 SC_input() は必ずプログラム中で最初の実行文とすること。これ以前に実行文を書いてはいけない。SC_input を実行することにより、整数配列 SC_p に PCM データが格納され、整数変数 SC_n にデータの個数が格納される。なお、データは標準入力から読み込まれる。データの読み込みには必ずこの SC_input 関数を使うこと。
2. 配列 SC_p と SC_s は 1 次元配列で、サイズは定数 MAXN を使って MAXN に固定されている。なお、SC_p には入力データが格納されるので、この内容を変更してはならない。
3. MAXN は 882000(最大の PCM データ数) である。
4. 解答の 1 ビット列を SC_s に格納し、関数 SC_output(int a) を呼ぶことにより、結果が出力される。引数は増減幅を表す整数 A である。SC_output 関数以外の方法で出力してはならない。

なお、プログラム提出後の審査には出題側で用意したヘッダーファイル (本質的には上と同じだが、審査用のコードが含まれる) を使うので、ヘッダーファイルを独自に改変しても反映されない。

6 ベクトル化乱数生成

大量の乱数が必要な場合にはベクトル化された乱数発生関数を用いる。そのために

```
#include "vrand.h"
```

とヘッダーファイル vrand.h をインクルードする。これには

```
void RN_ranset(int RN_raninit);
void RN_rnd(int RN_r[], int RN_n);
```

のふたつの関数と定数 RN_MAX が定義されている。

RN_ranset は乱数の初期化関数であり、乱数を使う前に必ず一度だけこの関数を呼ぶ。引数は任意の整数で、この数値によって発生する乱数列が決まる。RN_rnd が乱数を発生する関数である。RN_n に発生したい乱数の個数を入れてこの関数を呼ぶと、整数配列 RN_r[] に整数乱数が格納される。なお、配列 RN_r[] はプログラム中で定義しておかなくてはならない (任意の名前でよい)。

このようにして得られた乱数は 0 から RN_MAX までの 31 ビットの正の整数である。これを 0 から 1 までの乱数に変換したければ、RN_r[i]/(double)RN_MAX などとすればよい。また、0 から 100 までの整数乱数が欲しければ、RN_r[i]% 100 でよい。