練習問題解説 (コンピューターによるフーリエ級数)

山本昌志*

2003 年 11 月 17 日

1　フーリエ級数

連立方程式の最後の問題は、フーリエ級数に関係した問題です。そこで、その問題について簡単に解説しておきます。

1.1　フーリエ級数とは

フーリエ級数は、フーリエ (1768-1830 フランス) が熱伝導の方程式を研究しているときに発見した級数です。今は、もっぱら波 (振動も波と考える) の問題に適用されています。

フーリエ級数とは、つぎのように区間 $[-\pi, \pi]$ で定義された任意の関数を sin と cos で展開したものです。

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos nx + \sum_{n=1}^{\infty} b_n \sin nx \tag{1}$$

この式が正確に成り立つために $f(x)$ に課せられる条件は、$f(x)$ はたかだか有限個の有限な不連続点しか持たず、また有っても有限個の極値、極大値または極小値を持つことです。科学技術で用いられる普通の関数はこの条件を満足します。有限個の有限な不連続点も許されのです。これは、非常に驚くべきことです。今までほぼ無視してきた不連続点が取り扱えるというのは面白いことです。

このように関数を展開することはいろいろな場面で使われます。皆さんは、フーリエ級数以外にも冪乗に展開するテイラー展開というものをすでに学習しているはずです。場面に応じて都合の良い展開を使えばよいのです。

本当に任意の関数が、式 (1) に展開できることの証明は数学の教科書に譲ることにして、その展開の係数、フーリエ係数 a_n と b_n を求める方法を示します。これこそが、フーリエ級数やフーリエ変換、あるいはコンピューターによる離散フーリエ変換 (DFT) または高速フーリエ変換 (FFT) の実際の計算です。

この展開係数の理論的な式は、三角関数の次の性質を使いことにより求めることができます。ただし、m と n は 1 以上の整数とします。

$$\int_{-\pi}^{\pi} \cos mx \cos nx = \begin{cases} 0 & m \neq n \\ \pi & m = n \end{cases} \tag{2}$$

*国立秋田工業高等専門学校　電気工学科

$$\int_{-\pi}^{\pi} \sin mx \sin nx = \begin{cases} 0 & m \neq n \\ \pi & m = n \end{cases} \tag{3}$$

$$\int_{-\pi}^{\pi} \cos mx \sin nx = 0 \tag{4}$$

$$\int_{-\pi}^{\pi} \cos nx = 0 \tag{5}$$

$$\int_{-\pi}^{\pi} \sin nx = 0 \tag{6}$$

まずは、a_0 のを計算します。そのために、式 (1) の両辺を区間 $[-\pi, \pi]$ を x で積分すればよく、

$$\begin{aligned} \int_{-\pi}^{\pi} f(x)dx &= \int_{-\pi}^{\pi} \frac{a_0}{2} dx + \sum_{n=1}^{\infty} a_n \int_{-\pi}^{\pi} \cos nx dx + \sum_{n=1}^{\infty} b_n \int_{-\pi}^{\pi} \sin nx dx \\ &= \pi a_0 \end{aligned} \tag{7}$$

となります。これから、

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)dx \tag{8}$$

と計算できます。$\frac{a_0}{2}$ は、関数 $f(x)$ の区間 $[-\pi, \pi]$ の平均値になっていることに気が付いてほしい。これを技術者は直流成分と言います。

つぎに、a_n を求めよう。式 (1) の両辺に $\cos mx$ を書けて、区間 $[-\pi, \pi]$ を x で積分します。すると、

$$\begin{aligned} \int_{-\pi}^{\pi} f(x) \cos mx dx &= \int_{-\pi}^{\pi} \frac{a_0}{2} \cos mx dx + \sum_{n=1}^{\infty} a_n \int_{-\pi}^{\pi} \cos nx \cos mx dx + \sum_{n=1}^{\infty} b_n \int_{-\pi}^{\pi} \sin nx \cos mx dx \\ &= a_m \pi \end{aligned} \tag{9}$$

となります。これから、直ちに a_n は計算できて、

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx dx \qquad (n = 1, 2, 3, \cdots) \tag{10}$$

となります。ここで、この式と式 (8) を比べます。すると、

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx dx \qquad (n = 0, 1, 2, 3, \cdots) \tag{11}$$

となることが分かります。式がひとつ節約できたことになります。同様に、b_n は、両辺に $\sin mx$ をかけて、積分を行います。すると、

$$\begin{aligned} \int_{-\pi}^{\pi} f(x) \sin mx dx &= \int_{-\pi}^{\pi} \frac{a_0}{2} \sin mx dx + \sum_{n=1}^{\infty} a_n \int_{-\pi}^{\pi} \cos nx \sin mx dx + \sum_{n=1}^{\infty} b_n \int_{-\pi}^{\pi} \sin nx \sin mx dx \\ &= b_m \pi \end{aligned} \tag{12}$$

となります。したがって、b_n は、

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin mx dx \qquad (n = 1, 2, 3, \cdots) \tag{13}$$

と求めることができます。この式と式 (11) からフーリエ係数が計算でき、式 (1) の展開が可能となるわけです。

これまでは、関数の区間 $[-\pi, \pi]$ としてきましたが、式 (1) はその区間の外側でも計算できます。外側の値が気になるところですが、これは三角関数の性質により直ちにわかります。フーリエ級数を形成する三角関数は、2π の周期を持ちますので、フーリエ級数も同じ周期を持つことになります。したがって、区間 $[-\pi, \pi]$ の形がその外側で周期的に現れることになります。波と一緒です。

1.2 対称性

フーリエ級数は、あらゆる関数は三角関数からできていると言っています。その三角関数は、原点をを境に対称な cos と反対称な sin に分けることができます。このことから、どんな形の関数でも対称関数と反対称な関数に分解できることを意味しています。

もし、元の関数が原点を境に対象な形をしているならば、そのフーリエ級数は $cos(nx)$ の和のみで書かれることになります[1]。一方、原点を中心に反対照な関数であれば、$sin(nx)$ の和のみで表現できます。これは、a_n や b_n を求める式からも理解できます。

2 計算機によるフーリエ級数

2.1 離散フーリエ変換 (フーリエ cos 展開)

ここでは、計算機によりフーリエ変換を行う方法を説明します。フーリエ変換と言っても、実際はフーリエ級数の係数を求めているだけです。今までは数学だったので、関数 $f(x)$ は連続的な値でした。しかし、実際の測定量、例えば電圧など連続的に測定してそのデータが蓄えられるわけでは有りません。連続ではなく離散的なデータとなります。これを上手に扱いフーリエ変換する方法を考えましょう。とは言っても、上手にフーリエ変換する FFT まではたどり着きませんが、そのさわりを説明します。

ここでも話を簡単にするために、周期を 2π とします。そして、原点を境に対称な場合を考えます。これは、連立方程式の練習問題がそうなっていたからで、一般的な問題に拡張することは容易です。一般化についても、後に示す私の講義ノートを参照してください。

その、周期の半分 $[0, \pi]$ 中で N 個の等間隔でデータが得られたとします。当然、原点を境に対称という仮定がありますので、$[-\pi, 0]$ の値も分かっていますが、ここでは使いません。データが等間隔に並ぶということは重要です[2]。すなわち、

$$x_j = \frac{\pi}{N} j \qquad (j = 0, 1, 2, \cdots, N-1) \tag{14}$$

[1] 直流成分は、$n = 0$ の cos の一部と考えます。
[2] ここでは重要でありませんが、DFT や FFT の計算では重要になります。

の点でデータが得られたものとします。ここで得られデータを

$$f_j = f(x_j) \tag{15}$$

とします。

さて、準備ができたので、実際のフーリエ級数の式 (1) を評価してみます。測定量である f_j と x_j はそれぞれ N 個しかありません。従って、フーリエ係数の c_n も N 個しか決めることはできません。関数は対称と仮定していますので、反対称を示す b_n はゼロとなり、a_n の $n = 0, 1, 2, \cdots, N-1$ を求めることになります。この a_n を積分の式 (11) を使わないで、連立方程式から計算しようというのです。

すなわち、

$$f(x) = \sum_{n=0}^{N-1} a_n \cos(nx) \tag{16}$$

を満たす a_n を求めることになります。ここで残された問題は、測定量の x_j と f_j から a_n を決めることになります。これは比較的簡単で、

$$f_j = \sum_{n=0}^{N-1} a_n \cos(kx_j) \qquad (j = 0, 1, 2, \cdots, N-1) \tag{17}$$

の連立方程式を解けば良いことが分かります。この式の形が分かりにくい。それではもう少し分かり易く書いてみましょう。

$$\begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \cos(x_1) & \cos(2x_1) & \dots & \cos\left((N-1)x_1\right) \\ 1 & \cos(x_2) & \cos(2x_2) & \dots & \cos\left((N-1)x_2\right) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(x_{N-1}) & \cos(2x_{N-1}) & \dots & \cos\left((N-1)x_{N-1}\right) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{pmatrix} \tag{18}$$

f_j と x_j は既知なので、この連立方程式を解いて a_j を計算することは可能です。その解がフーリエ係数 a_n となります。高速フーリエ変換 (First Fourier Transform 略して FFT) は、この a_n をある工夫されたアルゴリズムで高速に計算しているだけです。

「なるほど、連立方程式ならば計算機は得意なので、式 (18) を解けば話は終わり」と思ってはいけません。ここでの学習はこれを実際に計算してみることですが、実際には高速に計算するためにいろいろと工夫ができます。何しろ、$N = 1000$ 位になるとこの式を計算するのに膨大な時間がかかりますので、計算時間の短縮が必要になります。

この計算を高速で行うように工夫したアルゴリズムが、離散フーリエ変換 (DFT) であったり高速フーリエ変換 (FFT) と呼ばれるものです。これは、データが等間隔で並んでいるという性質を利用する方法です。もう少し詳しい説明は、私の 5M 実験の講義ノートの「フーリエ変換とその周辺」を見てください。URL は以下の通りです。

http://www.ipc.akita-nct.ac.jp/ yamamoto/lecture/2003/5M_Exp/lecture_5M_Exp/fourier_transform.pdf

2.2　練習問題について

式が分かったので練習問題の内容について説明します。練習問題は、図 1 に示すデータから、フーリエ級数を求めようとするものです。周期は 2π で、原点を中心に対称としています。図からも分かるように三角波をフーリエ *cos* 展開することになります。

この波形のフーリエ級数は、積分ではなく式 (18) の連立方程式を解くことにより求めることができます。図 1 を表す式は、以下の通りです。

$$\begin{pmatrix} 0 \\ \frac{1}{N} \\ \frac{2}{N} \\ \vdots \\ \frac{N-1}{N} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \cos\left(\frac{\pi}{N}\right) & \cos\left(\frac{2\pi}{N}\right) & \cdots & \cos\left(\frac{(N-1)\pi}{N}\right) \\ 1 & \cos\left(\frac{2\pi}{N}\right) & \left(\frac{4\pi}{N}\right) & \cdots & \cos\left(\frac{2(N-1)\pi}{N}\right) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \cos\left(\frac{(N-1)\pi}{N}\right) & \cos\left(\frac{2(N-1)\pi}{N}\right) & \cdots & \cos\left(\frac{(N-1)(N-1)\pi}{N}\right) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{pmatrix} \tag{19}$$

この連立方程式を解いて、式 (16) に代入すれば、離散的な点を通る式を求めることができます。最終的には、後述の 11 ページの図 2 のようになります。

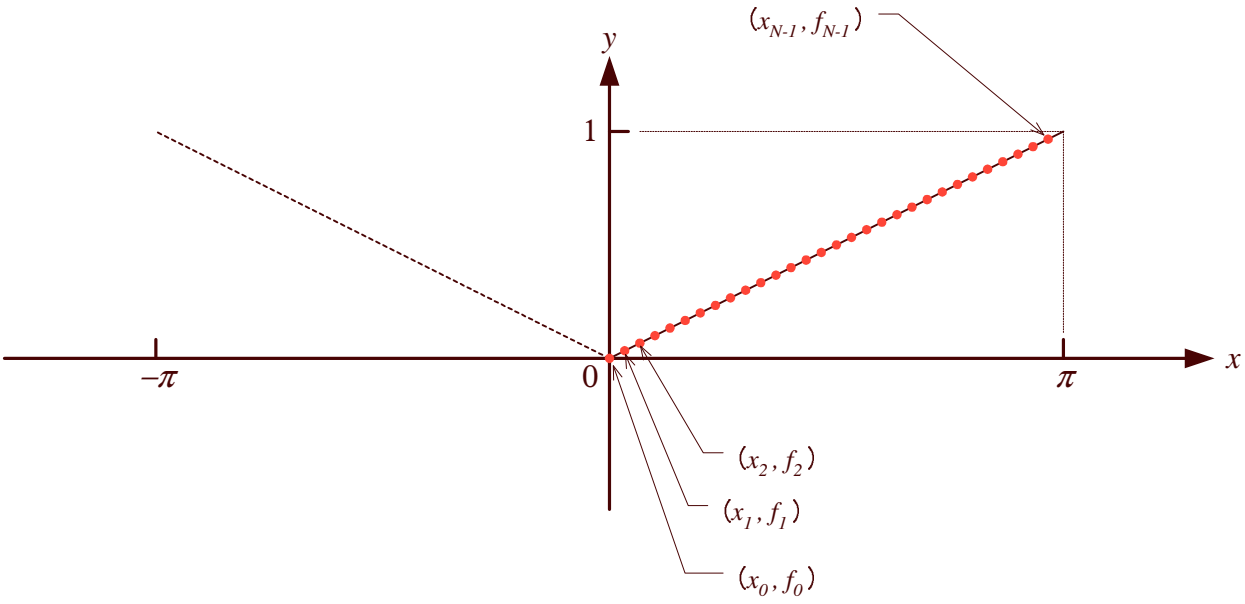

図 1: 練習問題のデータ

3　C 言語でフーリエ変換

3.1　プログラム

実際のプログラムを以下に示します。このプログラムは、

1. 式 (19) を解くために、配列のデータを作成する部分
2. 連立方程式を計算する部分
3. 連立方程式から、フーリエ級数の関数を計算する部分
4. 関数をグラフにする部分

に分かれています。実際のプログラムは、以下の通りで、次節に注意事項が載っているのでそれを見ながら理解してください。

```
#include <stdio.h>
#include <math.h>
#define MAXN 200

int gauss_jordan(int n, double a[][MAXN+10], double b[]);
double func(double x, int n, double b[]);
void mk_graph(int n, double x[], double x1, double x2,
                            double y[], double y1, double y2);

/*============================================================*/
/*    main function                                           */
/*============================================================*/
main(){

   double a[MAXN+10][MAXN+10], b[MAXN+10];
   double pi=4.0*atan(1);
   int n, i, j, k;
   int nf;
   double x1, x2, dx, x[1010], y[1010];

   n=100;

   /* -------- 係数行列と同次項の設定 ---------- */
   for(i=1 ; i<= n ; i++){
      for(j=1 ; j<=n ; j++){
         a[i][j]=cos((double)pi*(i-1)*(j-1)/n);
      }
      b[i]=(double)(i-1)/n;
   }

   /* -------- 連立方程式の計算 ---------------- */
   if(gauss_jordan(n, a, b) == 0){
      printf("singular matrix !!!\n");
      exit(0);
   };

   /* -------- 元の関数の計算   ---------------- */
   nf=1000;
   x1=-2*pi;
```

```
   x2=2*pi;

   dx=(x2-x1)/nf;

   for(i=0 ; i<=nf ; i++){
      x[i]=x1+dx*i;
      y[i]=func(x[i], n, b);
   }

   /* -------- グラフ作成 ----------------------- */
   mk_graph(nf,x,x1,x2,y,-0.5,1.5);

}

/*============================================================*/
/*    Gauss-Jordan method                                     */
/*============================================================*/
int gauss_jordan(int n, double a[][MAXN+10], double b[]){

   int ipv, i, j;
   double inv_pivot, temp;
   double big;
   int pivot_row, row[MAXN+10];

   for(ipv=1 ; ipv <= n ; ipv++){

      /* ---- 最大値探索 ---------------------------- */
      big=0.0;
      for(i=ipv ; i<=n ; i++){
         if(fabs(a[i][ipv]) > big){
            big = fabs(a[i][ipv]);
            pivot_row = i;
         }
      }
      if(big == 0.0){return(0);}
      row[ipv] = pivot_row;

      /* ---- 行の入れ替え -------------------------- */
      if(ipv != pivot_row){
         for(i=1 ; i<=n ; i++){
            temp = a[ipv][i];
            a[ipv][i] = a[pivot_row][i];
            a[pivot_row][i] = temp;
         }
         temp = b[ipv];
         b[ipv] = b[pivot_row];
         b[pivot_row] = temp;
      }
```

```
      /* ---- 対角成分=1(ピボット行の処理) ---------- */
      inv_pivot = 1.0/a[ipv][ipv];
      a[ipv][ipv]=1.0;
      for(j=1 ; j <= n ; j++){
         a[ipv][j] *= inv_pivot;
      }
      b[ipv] *= inv_pivot;

      /* ---- ピボット列=0(ピボット行以外の処理) ---- */
      for(i=1 ; i<=n ; i++){
         if(i != ipv){
            temp = a[i][ipv];
    a[i][ipv]=0.0;
            for(j=1 ; j<=n ; j++){
               a[i][j] -= temp*a[ipv][j];
            }
            b[i] -= temp*b[ipv];
         }
      }

   }

   /* ---- 列の入れ替え (逆行列) -------------------------- */
   for(j=n ; j>=1 ; j--){
      if(j != row[j]){
         for(i=1 ; i<=n ; i++){
            temp = a[i][j];
            a[i][j]=a[i][row[j]];
            a[i][row[j]]=temp;
         }
      }
   }

   return(1);
}

/*============================================================*/
/*    function                                                */
/*============================================================*/
double func(double x, int n, double b[]){

   int i;
   double y;

   y=0;
   for(i=1 ; i<=n ; i++){
      y += b[i]*cos((i-1)*x);
   }
```

```
   return(y);
}

/*============================================================*/
/*   make a graph                                             */
/*============================================================*/
void mk_graph(int n, double x[], double x1, double x2,
                     double y[], double y1, double y2){

   int i;
   char *data_file;
   FILE *out;
   FILE *gp;

   /* ------- make a data file ------------ */
   data_file="gpdata.txt";
   out = fopen(data_file, "w");
   for(i=0; i<=n; i++){
      fprintf(out, "%e\t%e\n", x[i], y[i]);
   }
   fclose(out);

   /* == flowing lines make a graph by using gnuplot == */

   gp = popen("gnuplot -persist","w");

   fprintf(gp, "reset\n");
   fprintf(gp, "set terminal postscript eps color\n");
   fprintf(gp, "set output \"graph.eps\"\n");
   fprintf(gp, "set grid\n");

   /* -------------  set x axis ----------------*/
   fprintf(gp, "set xtics 1\n");
   fprintf(gp, "set mxtics 10\n");
   fprintf(gp, "set xlabel \"%s\"\n", "x");
   fprintf(gp, "set nologscale x\n");
   fprintf(gp, "set xrange[%e:%e]\n", x1, x2);

   /* -------------  set y axis ---------------*/
   fprintf(gp, "set ytics 1\n");
   fprintf(gp, "set mytics 10\n");
   fprintf(gp, "set ylabel \"%s\"\n", "y");
   fprintf(gp, "set nologscale y\n");
   fprintf(gp, "set yrange[%e:%e]\n", y1, y2);

   /* -------------  plat graphs ---------------*/
   fprintf(gp, "plot \"%s\" using 1:2 with line\n", data_file);

   fprintf(gp, "set terminal x11\n");
```

```
    fprintf(gp, "replot\n");

    pclose(gp);
}
```

3.2 注意点

このプログラムの内容について、以下に注意してください。

- 結果は実数を得たいのですが、整数同士の演算、特に除算は注意が必要です。FORTRAN で学習したように、整数同士の演算はたとえ除算であろうが整数になります。それを防ぐために、C 言語では強制型変換 (キャスト) というものがあります。(型) 式のように書くと、式の内容が示された型に変換されます。例えば、以下のように使います。

```
a[i][j]=cos((double)pi*(i-1)*(j-1)/n);
```

- グラフ作成の部分は関数にしました。その関数は、

```
mk_graph(データ点数,x 座標の配列,x 軸左端,x 軸右端,y 座標の配列,y 軸下端,y 軸上端);
```

の形になっています。これを呼び出せば、グラフを書きます。これをコピーペーストして使いましょう。

3.3 実行結果

このプログラムを実行すると、図 2 に示す結果が得られます。

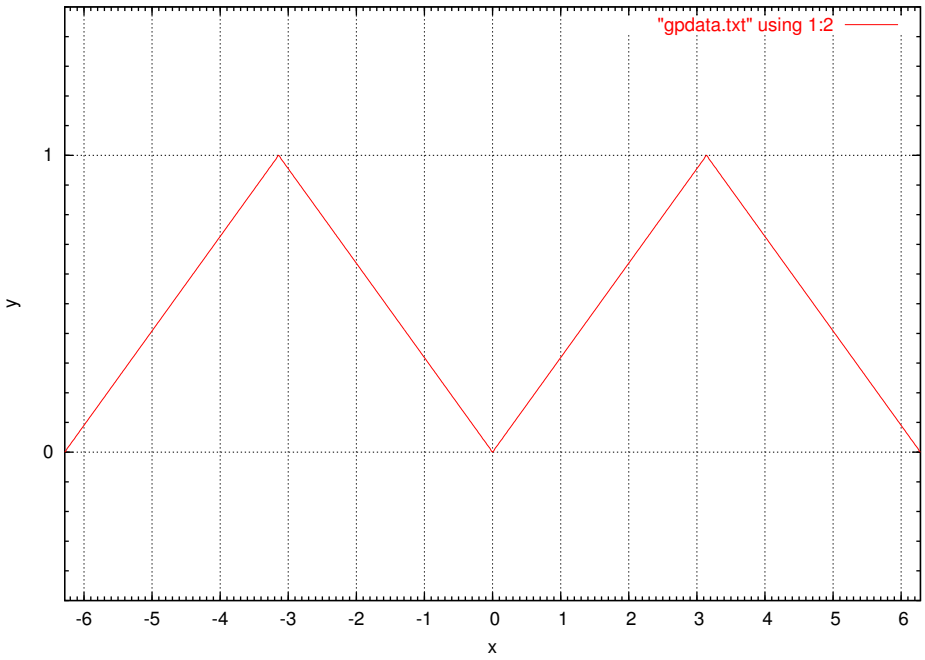

図 2: 実行結果。フーリエ係数は 100 個