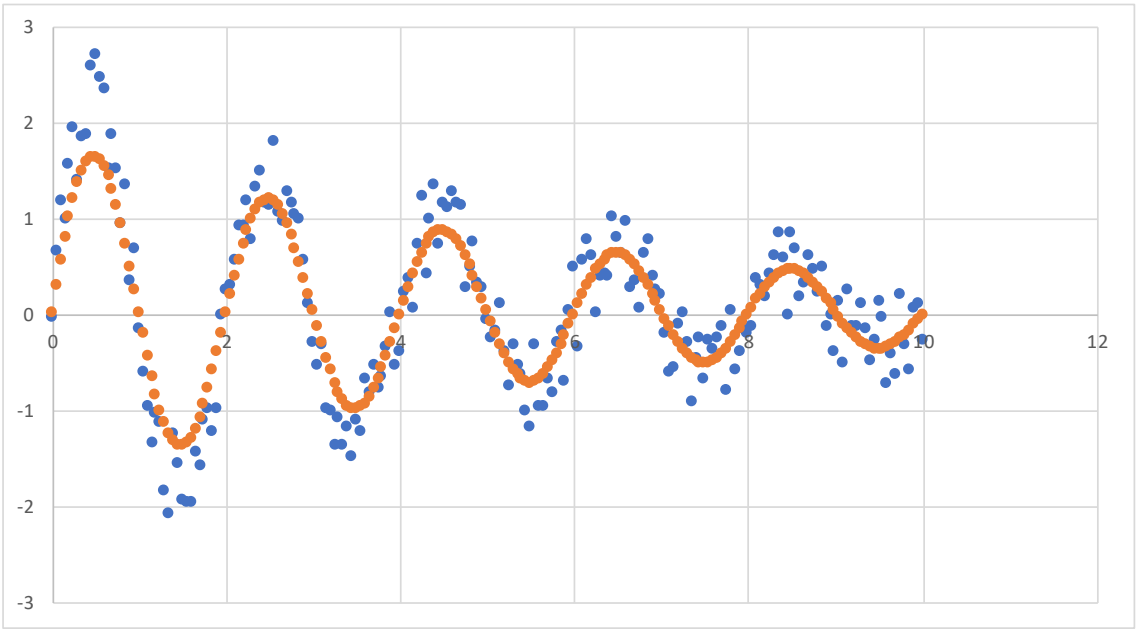
グラフ



実行後の値

A：1.766019

α ：0.160815

ω ：3.136629

ソース

```
#include <stdio.h>
#include <math.h>
#define VAR 3              //1変数だから
#define NUM (VAR + 1) // 配列の大きさは変数の数＋1
#define Dmax 400         // 読み込みファイルのデータ数の許容最大値（扱うデータ数よりも大きくしておくこと）

// プロトタイプ宣言の部分
void joretsu(double var[NUM][NUM]);                                    //降順に並べなおす
void f0c(double var[NUM][NUM], double f0[VAR]);                        //重心値の計算
```

```
void frd(double var[NUM][NUM]);                                    //
シンプレックスの縮小
void frc(double alpha, double f0[VAR], double var[NUM][NUM], double fr[NUM]); //fr の
計算
void fec(double gamma, double f0[VAR], double fr[NUM], double fe[NUM]);        //fe
の計算
void fcc(double beta, double f0[VAR], double var[NUM][NUM], double fc[NUM]);   //fc
の計算
double func(double var[NUM], double datax[Dmax], double datay[Dmax], int Dline);
// 関数の定義,2 乗誤差を計算するには、入力データの情報も必要なので、それを引数に追
加

int main()
{
    double datax[Dmax], datay[Dmax]; // 入力データは t の値と y の値で別の配列に保存
    int Dline = 0;                    // データファイルの行数に関する変数

    double var[NUM][NUM] = {{1.9906, 0.2010, 2.0, 0.0},
                            {2.0000, 0.2352, 3.0, 0.0},
                            {2.1000, 0.3000, 4.0, 0.0},
                            {2.3000, 0.4000, 5.0, 0.0}};

    //重心を入れる配列を用意(この例では、x,y の順で値が入る。)
    double f0[VAR];
    //反射, 拡張, 収縮値を入れる配列を用意(この例では、x,y,f(x,y)の順で値が入る。)
    double fr[NUM], fe[NUM], fc[NUM];
    //反射, 拡張, 収縮各係数を設定
    double alpha = 2.0, gamma = 2.0, beta = 0.5;
    //シンプレックスの大きさ fsize(初期値は fmin より大きくしておく)と収束判定値
fmin(適切に設定する)
    double fsize = 1000.0, fmin = 1.0e-5;

    int i, j, k = 0; //  k はループのカウンター

    /* データファイルから配列への保存、ここから */
    FILE *fp2;
```

```
if ((fp2 = fopen("data3.csv", "r")) == NULL)
{
    printf("入力用ファイル開けず¥n");
    return (1);
}

for (i = 0; i < Dmax; i++)
{
    if (fscanf(fp2, "%lf,%lf¥n", &datax[i], &datay[i]) == EOF)
    {
        break;
    }
}

fclose(fp2);
/* ここまで */

//// データ入力チェック用（動作チェックのときはコメントアウトをはずす）
// for (j = 0; j < i; j++)
// {
//     printf("%lf %lf¥n", datax[j], datay[j]);
// }

//Dline は、例えば、データファイルの最終行番号となるようにする（データを格納した配列の最後の要素番号は Dline ということ）
for (j = 0; j < i; j++)
    Dline++;

// func の引数が変わったので、例えば関数値 f(x,y)の計算の部分はこのようになる。
// （他にも func を呼び出しているところが有るので修正する。）
// for (i = 0; i < NUM; i++)
// {
//     var[i][VAR] = func(var[i], datax, datay, Dline); //ここが変更
// }
```

```
while (fsize > fmin)
{
    // 収束判定まで繰り返すループスタート
    printf("############################¥n");
    printf("%d 回目の試行¥n", k + 1);
    printf("############################¥n");

    //関数値 f(x,y)の計算
    for (i = 0; i < NUM; i++)
    {
        var[i][VAR] = func(var[i], datax, datay, Dline);
    }

    // 最大値、2 番目に大きい値、最小値を探索(実際には降順にソート)
    joretsu(var);

    printf(" * ソート後¥n");
    for (i = 0; i < NUM; i++)
    {
        for (j = 0; j < NUM; j++)
        {
            printf("変数%d %d¥t%lf¥t", i, j, var[i][j]);
        }
        printf("¥n");
        printf("関数値%lf¥n¥n", var[i][VAR]);
    }

    //最高値を除外した残りの点で重心 X0 を算出
    f0c(var, f0);

    //反射 Xr の計算する
    frc(alpha, f0, var, fr);
    fr[VAR] = func(fr, datax, datay, Dline);
    printf("反射  %d    %lf¥n¥n", VAR, fr[VAR]);

    if (fr[VAR] <= var[1][VAR])
```

```
{ //var[1][*]（Fs のこと）と比較
    if (fr[VAR] < var[VAR][VAR])
    { //var[VAR][VAR]（Fl のこと）と比較
        //拡張 Xe,Fe を計算する
        fec(gamma, f0, fr, fe);                    //拡張 Xe を計算する
        fe[VAR] = func(fe, datax, datay, Dline); //拡張 Xe から Fe を計算する
        printf("拡張 %d   %lf¥n¥n", VAR, fe[VAR]);

        //フローチャートの下方に進む
        if (fe[VAR] < fr[VAR])
        { //Fe と Fr の比較
            for (i = 0; i < VAR; i++)
            {
                var[0][i] = fe[i]; //var[0][*] は xh のこと。 fe[i]と入替え
                printf("表 1,2 の場合 h と置き換わるのは   %lf¥n", var[0][i]);
            }
        }
        else
        {
            for (i = 0; i < VAR; i++)
            {
                var[0][i] = fr[i]; //var[0][*] は xh のこと。 fr[i]と入替え
                printf("表 3 の場合 h と置き換わるのは   %lf¥n", var[0][i]);
            }
        }
    }
    else
    {
        for (i = 0; i < VAR; i++)
        {
            var[0][i] = fr[i]; //var[0][*] は xh のこと。 fr[i]と入替え
            printf("表 4 の場合 h と置き換わるのは   %lf¥n", var[0][i]);
        }
    }
}
else
```

```
{ //フローチャートの右下に来ている
    if (fr[VAR] < var[0][VAR])
    { //var[0][*] は xh のこと。 fr[i]と比較
        for (i = 0; i < VAR; i++)
        {
            var[0][i] = fr[i]; //var[0][*] は xh のこと。 xr と入替え
        }
    }
    fcc(beta, f0, var, fc); //収縮のポリトープを求める
    fc[VAR] = func(fc, datax, datay, Dline);
    printf("収縮 %d  %lf¥n¥n", VAR, fc[VAR]);

    if (fc[VAR] < var[0][VAR])
    { //var[0][*] は xh のこと。 fc と比較
        for (i = 0; i < VAR; i++)
        {
            var[0][i] = fc[i]; //var[0][*] は xh のこと。 xc と入替え
            printf("表 5 の場合 h と置き換わるのは  %lf¥n", var[0][i]);
        }
    }
    else
    {
        //              シンプレックスの縮小化
        frd(var);
    }
}

//シンプレックスの大きさ
fsize = 0.0;
for (j = 0; j < VAR; j++)
{
    for (i = 0; i < VAR; i++)
    {
        fsize += fabs(var[j][i] - var[j + 1][i]);
    }
}
```

```
        for (i = 0; i < VAR; i++)
        {
            fsize += fabs(var[0][i] - var[VAR][i]);
        }
        printf("ポリトープの大きさ　%lf¥n", fsize);
        k++;
    }

    printf("¥n");
    for (i = 0; i < VAR; i++)
    {
        printf("最終変数 0 の%d 番目のパラメータの結果は　%lf¥n", i, var[0][i]);
    }

    printf("最終変数 0 の%d 番目のパラメータの結果は　%lf¥n", VAR, func(var[i], datax,
datay, Dline));
}

// 関数値の計算
double func(double var[VAR], double datax[Dmax], double datay[Dmax], int Dline)
{
    //引数が変わったので、ここが変更
    double sum = 0.0;
    double x, y; //y は理論式の値で、y の計算のために x を使う。
    int i;

    // 各データ点に対して 2 乗誤差を計算して、全て足し合わせたい。Dline を使って表
す。
    for (i = 0; i < Dline; i++)
    {
        x = datax[i];                                    //x の値はデータファイルの
x と同じ値にしないといけない。
        y = var[0] * exp(-var[1] * x) * sin(var[2] * x); //y は x とフィッティングパラメー
タ（var[0], var[1]）で計算する。

        sum += pow(y - datay[i], 2); // 各データ点に対して 2 乗誤差を計算して、それら
```

```
をどんどん足していく。
    }

    return sum;
}

// 降順にソートする
void joretsu(double var[NUM][NUM])
{

    int i, j, k;
    double dummy[NUM];

    for (i = 0; i < NUM; i++)
    {
        for (j = i + 1; j < NUM; j++)
        {
            if (var[i][NUM - 1] < var[j][NUM - 1])
            {
                for (k = 0; k < NUM; k++)
                {
                    dummy[k] = var[i][k];
                    var[i][k] = var[j][k];
                    var[j][k] = dummy[k];
                }
            }
        }
    }
}

//重心値の計算
void f0c(double var[NUM][NUM], double f0[VAR])
{
    int i, j;

    //配列の初期化
```

```
	for (i = 0; i < VAR; i++)
	{
		f0[i] = 0.0;
	}

	//最高値を除いた総和の算出
	for (i = 0; i < VAR; i++)
	{
		for (j = 1; j < NUM; j++)
		{
			f0[i] += var[j][i];
		}
	}

	//平均の算出
	for (i = 0; i < VAR; i++)
	{
		f0[i] = f0[i] / (double)(NUM - 1);
		printf("重心  %d    %lf¥n", i, f0[i]);
	}
	printf("¥n");
}

//シンプレックスの縮小
void frd(double var[NUM][NUM])
{
	int i, j;
	for (j = 0; j < VAR; j++)
	{
		for (i = 0; i < VAR; i++)
		{
			var[j][i] = (var[j][i] + var[VAR][i]) * 0.5;
		}
	}
	printf("シンプレックスを縮小¥n");
}
```

```
//fr の計算
void frc(double alpha, double f0[VAR], double var[NUM][NUM], double fr[NUM])
{
	int i;

	for (i = 0; i < VAR; i++)
	{
		fr[i] = f0[i] + alpha * (f0[i] - var[0][i]);
		printf("反射  %d    %lf¥n", i, fr[i]);
	}
}

//fe の計算
void fec(double gamma, double f0[VAR], double fr[NUM], double fe[NUM])
{
	int i;

	for (i = 0; i < VAR; i++)
	{
		fe[i] = f0[i] + gamma * (fr[i] - f0[i]);
		printf("拡張  %d    %lf¥n", i, fe[i]);
	}
}

//fc の計算
void fcc(double beta, double f0[VAR], double var[NUM][NUM], double fc[NUM])
{
	int i;
	for (i = 0; i < VAR; i++)
	{
		fc[i] = f0[i] + beta * (var[0][i] - f0[i]);
		printf("収縮  %d    %lf¥n", i, fc[i]);
	}
}
```