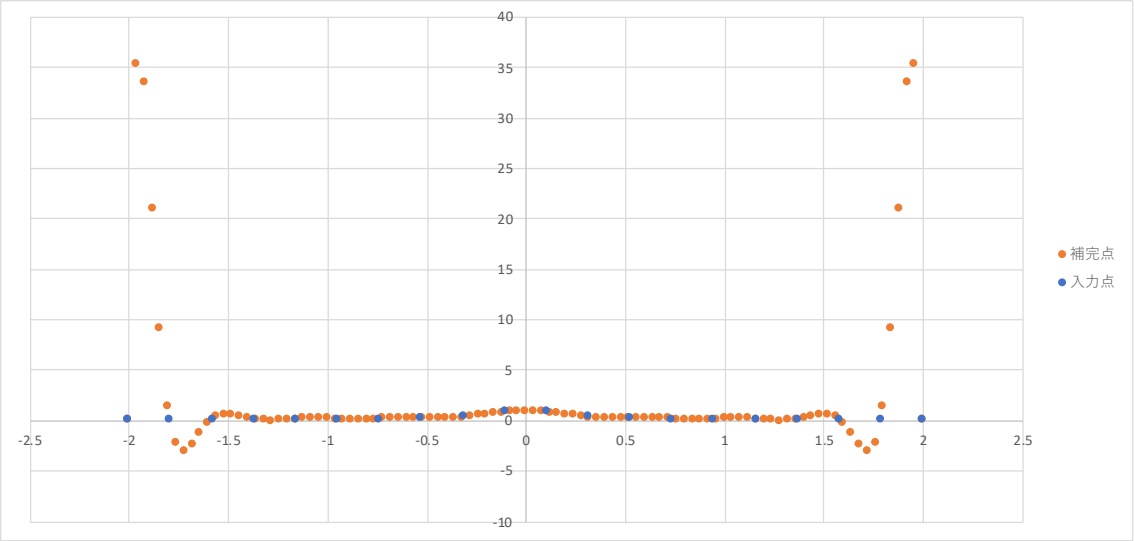
課題 3

・変更前のソースコード

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#define N 20            // データ点数（変更しないこと）
#define xmin (-2.0) //  （変更しないこと）
#define xmax (2.0)  //  （変更しないこと）
#define L 100           // 全区間の分割数（きちんと"補間"がされてさえすれば変数名ごと変更してもよい）
#define MAX_ORDER 3 // 補間時の最大の次数

double calLagrange(double sub_x[MAX_ORDER], double sub_y[MAX_ORDER], double a);

int main()
{
    int i, k;
    double a, sum;
    double x[N];
    double y[N];
    FILE *fp;

    if ((fp = fopen("output3_1.csv", "w")) == NULL)
    {
        printf("出力用ファイル開けず¥n");
        return (1);
    }

    /* データ点の準備（変更しないこと）*/
    for (i = 0; i < N; i++)
    {
        x[i] = xmin + i * (xmax - xmin) / (N - 1.0);
        y[i] = 1.0 / (36.0 * x[i] * x[i] + 1.0);
    }
```

```
    /* 補間計算（ここを完成させる。大幅に書き換えても良いが、x[i]やy[i]に適当な変換を施したものに対して補間を行い、その結果を逆変換するというのはダメ。）*/
    for (k = 0; k < L + 1; k++)
    {
        a = x[0] + k * (x[N - 1] - x[0]) / L;
        sum = calLagrange(x, y, a);
        fprintf(fp, "%lf,%lf¥n", a, sum);
    }

    fclose(fp);
    return 0;
}

double calLagrange(double sub_x[MAX_ORDER], double sub_y[MAX_ORDER], double a)
{
    int i, j;
    double tmp, sum;
    sum = 0;

    for (i = 0; i < N; i++)
    {
        tmp = 1.0;
        for (j = 0; j < N; j++)
        {
            if (i != j)
                tmp *= (a - sub_x[j]) / (sub_x[i] - sub_x[j]);
        }
        sum += sub_y[i] * tmp;
    }

    printf("%lf, %lf, %lf, %lf¥r¥n", sub_x[0], sub_y[0], a, sum);
    return sum;
}
```

・**実行結果**



・**問題点の分析とそれを解決するために加えた工夫に関する説明**

ラグランジュ補間では補間する際に生成される関数の次数が「補間点の数-1」となり，点の数が多くなるほど，関数の次数が大きくなる．関数の次数が大きいということはその分だけ変曲点の数の増加につながり，結果として今回のようなデータセットでは両端が振動するような波形となった．

関数の次数が大きいことが問題なら次数を下げれば良い．そこで今回は関数の次数を一定値となるようにした．具体的な手法としては，注目している点（コードないの変数 a）に一番近い x を取得し，その点周りの数点のみで補間するようにした．

・プログラム

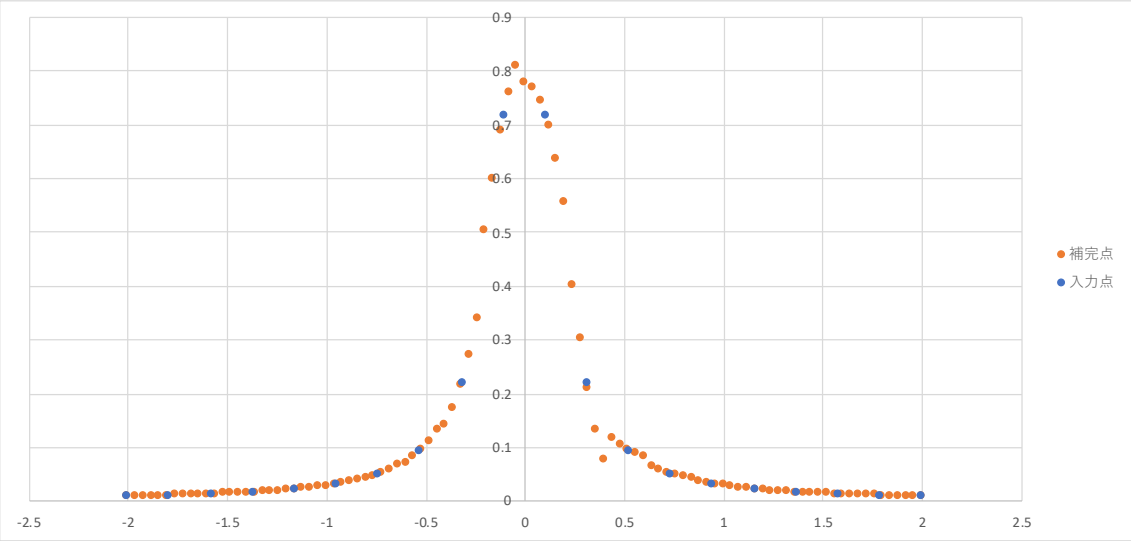
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <math.h>

#define N 20            // データ点数（変更しないこと）
#define xmin (-2.0) //  （変更しないこと）
#define xmax (2.0)   //  （変更しないこと）
#define L 100           // 全区間の分割数（きちんと"補間"がされてさえすれば変数名ごと変更してもよい）
#define MAX_ORDER 4 // 補間時の最大の次数

int findNearPointIndex(double x[N], double a);
```

```
double calLagrange(double sub_x[MAX_ORDER], double sub_y[MAX_ORDER], double a);

int main()
{
    int i, k;
    int near_point;
    double a, sum;
    double x[N];
    double y[N];
    double sub_x[MAX_ORDER], sub_y[MAX_ORDER];
    FILE *fp;

    if ((fp = fopen("output3_1.csv", "w")) == NULL)
    {
        printf("出力用ファイル開けず¥n");
        return (1);
    }

    /* データ点の準備（変更しないこと）*/
    for (i = 0; i < N; i++)
    {
        x[i] = xmin + i * (xmax - xmin) / (N - 1.0);
        y[i] = 1.0 / (36.0 * x[i] * x[i] + 1.0);
    }

    /* 補間計算（ここを完成させる。大幅に書き換えても良いが、x[i]や y[i]に適当な変換を施したものに対して補間を行い、その結果を逆変換するというのはダメ。）*/
    for (k = 0; k < L + 1; k++)
    {
        a = x[0] + k * (x[N - 1] - x[0]) / L;

        // a に一番近い x を求める
        near_point = findNearPointIndex(x, a);
        near_point++;
        // インデックス内に抑える
        if (near_point - MAX_ORDER / 2 < 0)
```

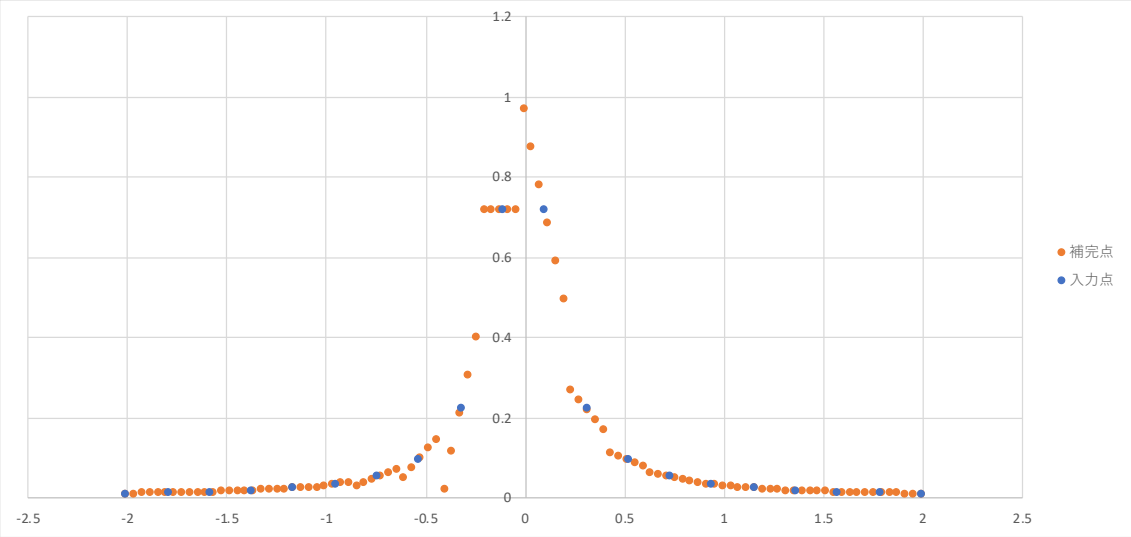
```
            near_point = MAX_ORDER / 2;
        if (near_point + MAX_ORDER / 2 + 1 > N)
            near_point = N - MAX_ORDER / 2 - 1;

        // a に近い MAX_ORDER 点だけの点群配列を作成
        for (i = 0; i < MAX_ORDER; i++)
        {
            sub_x[i] = x[near_point - MAX_ORDER / 2 + i];
            sub_y[i] = y[near_point - MAX_ORDER / 2 + i];
        }

        printf("%lf, %d, %lf, %lf, %lf¥n", a, near_point, sub_x[0], sub_y[0], sum);

        sum = calLagrange(sub_x, sub_y, a);
        fprintf(fp, "%lf, %lf¥n", a, sum);
    }

    fclose(fp);
    return 0;
}

int findNearPointIndex(double x[N], double a)
{
    int i;
    double dx = fabs(x[N - 1] - x[0]);
    double min_i = 0;
    for (i = 0; i < N; i++)
    {
        if (dx > fabs(a - x[i]))
        {
            dx = fabs(a - x[i]);
            min_i = i;
        }
    }
    return min_i;
}
```

```
double calLagrange(double sub_x[MAX_ORDER], double sub_y[MAX_ORDER], double a)
{
	int i, j;
	double tmp, sum;
	sum = 0;

	for (i = 0; i < MAX_ORDER; i++)
	{
		tmp = 1.0;
		for (j = 0; j < MAX_ORDER; j++)
		{
			if (i != j)
				tmp *= (a - sub_x[j]) / (sub_x[i] - sub_x[j]);
		}
		sum += sub_y[i] * tmp;
	}

	return sum;
}
```

・補間の次数を 2 とした時



・補間の次数を 1 とした時の実行結果



・補間の次数を 4 とした時の実行結果