プログラミング言語I

最小二乗法

20-413

北野正樹

1. 最小二乗法とは

最小二乗法とはあるデータ集合を直線で表したい時、誤差の二乗の和が最小になるすなわち誤差の平均値が最小である関係式を求める方法である。

例として一次の最小二乗法の正規方程式を求めてみる。

N個のデータに直線を当てはめてみる。

上記の式はa, bの関数であり、他変数の関数が最大値や最小値を取る点では、各変数に関する偏微分した関数に関しては0になる。したがって

これらを解いてa, bを定めれば良い。上記の直線の式を各変数で偏微分すると次の式を得る。

上記の式から次の連立一次方程式を得る

=

上記の連立一次方程式を正規方程式と呼び、これを解くことによってa, bが定まる。

左辺の(a, b)を残すため行列を移項すると

=

つまり

=

この式を解けばa, bが定まる。

1. 最小二乗法を適用する
   1. 以下の4点に最小二乗法を適用し、パラメータa, bを求めよ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | 1 | 2 | 3 | 5 |
| Y | 1 | 3 | 5 | 3 |

上の値を正規方程式に代入する。

=

おわり

* 1. 上記の4点に最小二乗法を適用し、パラメータa, bを求めよ。ただし、PythonとC言語の両方で実装すること

まず初めにPythonで実装し適用する。Pythonで最小二乗法を適用したソースコードを以下に示す。

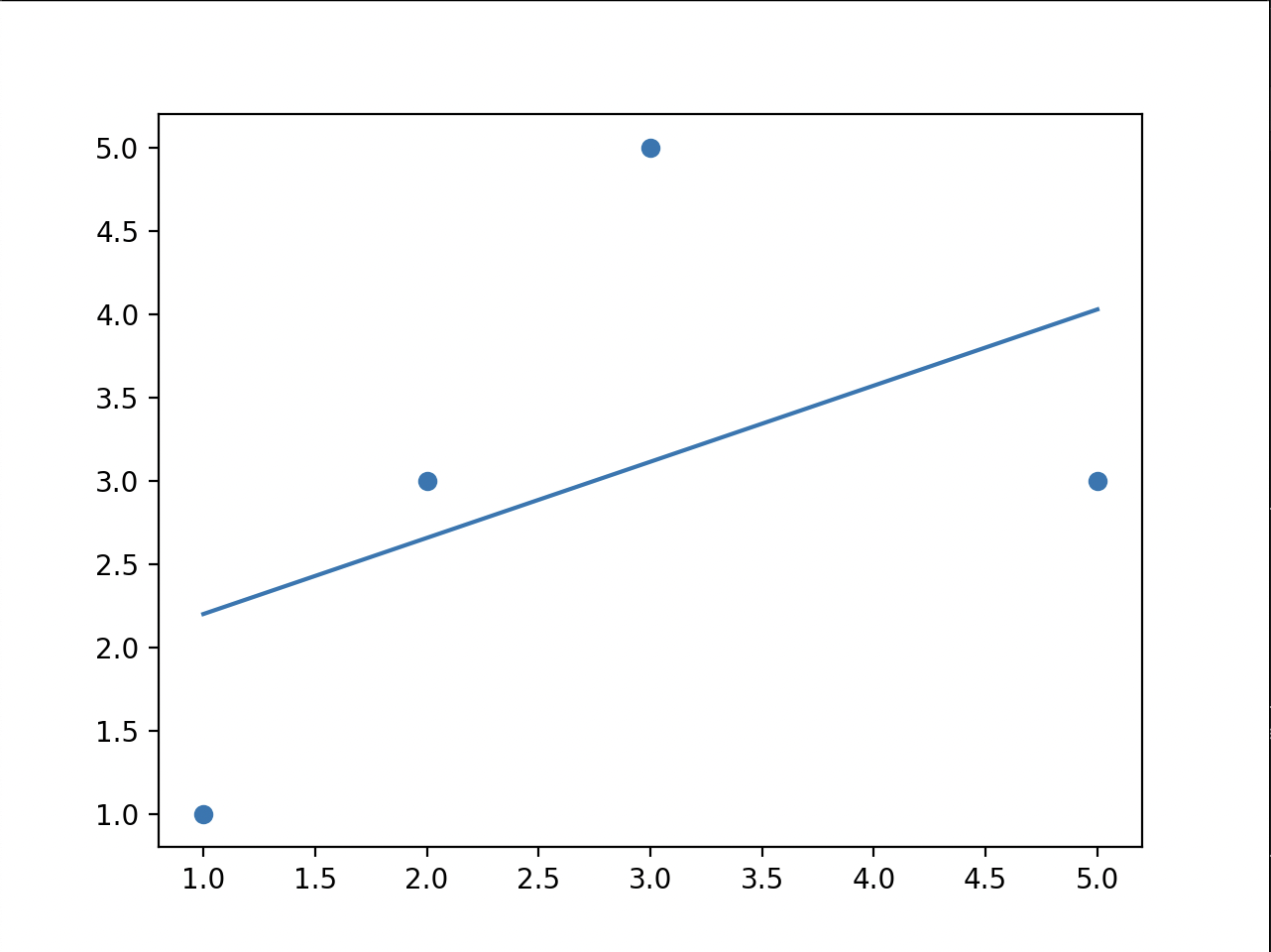
|  |
| --- |
| # 各ライブラリのインポート  import numpy as np  import matplotlib.pyplot as plt  import csv  # 各データを格納しておくための配列を宣言  x = []  y = []  t = []  # CSVファイルからデータをロードしてくる処理  filename = input("Please Enter CSVFileName: ")  with open(filename, encoding='utf8', newline='') as f:  csvreader = csv.reader(f)  for row in csvreader:  x.append(float(row[0]))  y.append(float(row[1]))  t.append(1)  # ロードしてきたデータからnumpyの配列を作成  X = np.array(x)  Y = np.array(y)  T = np.array(t)  # 作成した配列から各行列とベクトルと求める  A = np.array([  [np.sum(X\*X), np.sum(X)],  [np.sum(X), np.sum(T)]  ])  B = np.array([  [np.sum(X\*Y)],  [np.sum(Y)]  ])  # 計算するためにA行列の逆行列を求める  Ainv = np.linalg.inv(A)  # 各行列とベクトルとドット積を求める  Ans = np.dot(Ainv, B)  # 直線の方程式関数  def func(x):  y = Ans[0] \* x + Ans[1]  return y  # パラメータの表示  print(Ans)  # パラメータを関数に代入しグラフを出力する  plt.plot(x, func(x))  plt.scatter(x, y)  plt.show() |

上記のソースコードを実行した結果を以下に示す。

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % python3 LeastSquaresMethod.py example1.csv  [[0.45714286]  [1.74285714]] |

グラフ



次にC言語で実装したソースコードを以下に示す。

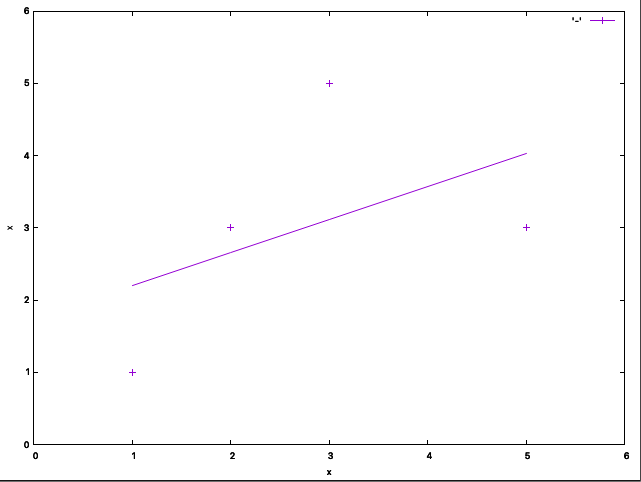
|  |
| --- |
| /\* 各ヘッダのインクルード \*/  #include <stdio.h> // 各入出力で使用  #include <stdlib.h> // 動的メモリ管理、exit関数で使用  #define GNUPLOT "gnuplot -persist"  /\* ペア型の定義 \*/  typedef struct Pair {  float n;  float m;  struct Pair\* nextPt;  } Pair;  /\* 単精度浮動点小数要素の定義 \*/  typedef struct Float {  float n;  struct Float\* nextPt;  } Float;  /\* 行列型の定義 \*/  typedef struct Matrix {  float a;  float b;  float c;  float d;  } Matrix;  /\* ベクトル型の定義 \*/  typedef struct Vector {  float a;  float b;  } Vector;  /\* 各関数のプロトタイプ宣言 \*/  Pair\* loadCSV(char\* filename);  void splitToFloat(Pair\* p, Float\*\* f1, Float\*\* f2);  float solveSigma(Float\* f1, Float\* f2);  int countFloat(Float\* f);  Pair findRange(Float\* f);  Matrix solveMatrixInverse(Matrix m);  Vector solveDotProduct(Vector v, Matrix m);  void freePair(Pair\* p);  void freeFloat(Float\* f);  float linerEquation(float x, Vector v);  void plot(Float\* x, Float\* y, Vector v);  /\* メイン関数 \*/  int main(int argc, char \*argv[]) {  // コマンドライン引数が合っているか判定  if (argc != 2) {  printf("IllegalArgumentsError\n");  printf("usage: ./LeastSquaresMethod <filename>\n");  return -1;  }  // CSVファイルからデータをロード  Pair\* points = loadCSV(argv[1]);  // 各座標の空のリストを宣言  Float \*X = NULL;  Float \*Y = NULL;  // 座標からX,Yのそれぞれのデータを抽出  splitToFloat(points, &X, &Y);  // 抽出してきたデータからそれぞれの行列・ベクトルを求める  Matrix A = {solveSigma(X, X), solveSigma(X, NULL)  ,solveSigma(X, NULL), countFloat(X)};  Vector B = {solveSigma(X, Y)  ,solveSigma(Y, NULL)};  // 演算のためにA行列の逆行列を求める  Matrix AInv = solveMatrixInverse(A);  // それぞれの行列とベクトルから答えを求める  Vector ans = solveDotProduct(B, AInv);  // 結果の表示  printf("(%f %f)\n", ans.a, ans.b);  // グラフの表示  plot(X, Y, ans);    // 各リストのメモリ解放  freePair(points);  freeFloat(X);  freeFloat(Y);  return 0;  }  /\*\*  FunctionName: plot    Argument : x, y, v  ArgumentType: ListOfFloat, ListOfFloat, Vector    Return : void  ReturnType : void    Description : 直線と点をプロットする関数  \*/  void plot(Float\* x, Float\* y, Vector v) {  FILE \*gp;  Float\* currentX = x;  Float\* currentY = y;  Pair rangeX = findRange(x);  Pair rangeY = findRange(y);  gp = popen(GNUPLOT, "w");  if (gp == NULL) {  printf("FileOpenError \"%s\"\n", GNUPLOT);  exit(-1);  }  fprintf(gp, "set multiplot\n");  fprintf(gp, "set xrange [%f:%f]\n", rangeX.n - 1, rangeX.m + 1);  fprintf(gp, "set yrange [%f:%f]\n", rangeY.n - 1, rangeY.m + 1);  fprintf(gp, "set xlabel \"x\"\n");  fprintf(gp, "set ylabel \"x\"\n");  fprintf(gp, "plot '-' with points pointtype 1\n");  while (currentX != NULL || currentY != NULL) {  fprintf(gp, "%f\t%f\n", currentX -> n, currentY -> n);  currentX = currentX -> nextPt;  currentY = currentY -> nextPt;  }  fprintf(gp, "e\n");  fprintf(gp, "plot '-' with lines linetype 1\n");  currentX = x;  while (currentX != NULL) {  fprintf(gp, "%f\t%f\n", currentX -> n, linerEquation(currentX -> n, v));  currentX = currentX -> nextPt;  }  fprintf(gp, "e\n");  fprintf(gp, "set nomultiplot\n");  fprintf(gp, "exit\n");  fflush(gp);  pclose(gp);  }  /\*\*  FunctionName: linerEquation    Argument : x, v  ArgumentType: float, Vector    Return : y  ReturnType : float    Description : xの値を代入するとyの値が返される直線方程式  \*/  float linerEquation(float x, Vector v) {  float y = v.a \* x + v.b;  return y;  }  /\*\*  FunctionName: freePair    Argument : f  ArgumentType: ListOfPair    Return : void  ReturnType : void    Description : Pairリストを全てメモリ解放する関数  \*/  void freePair(Pair\* p) {  Pair \*currentPair, \*nextPair;  currentPair = p;  while (currentPair != NULL) {  nextPair = currentPair -> nextPt;  free(currentPair);  currentPair = nextPair;  }  }  /\*\*  FunctionName: freeFloat    Argument : f  ArgumentType: ListOfFloat    Return : void  ReturnType : void    Description : Floatリストを全てメモリ解放する関数  \*/  void freeFloat(Float\* f) {  Float \*currentFloat, \*nextFloat;  while (currentFloat != NULL) {  nextFloat = currentFloat -> nextPt;  free(currentFloat);  currentFloat = nextFloat;  }  }  /\*\*  FunctionName: solveDotProduct    Argument : v, m  ArgumentType: Vector, Matrix    Return : DotProductVector  ReturnType : Vector    Description : ベクトルと行列のドット積を求める関数  \*/  Vector solveDotProduct(Vector v, Matrix m) {  Vector vector = {v.a\*m.a + v.b\*m.c  ,v.a\*m.b + v.b\*m.d};  return vector;  }  /\*\*  FunctionName: solveMatrixInverse    Argument : m  ArgumentType: Matrix    Return : MatrixInverse  ReturnType : Matrix    Description : 渡された行列の逆行列を求める関数  \*/  Matrix solveMatrixInverse(Matrix m) {  float scala = 1/(m.a\*m.d - m.b\*m.c);  Matrix matInv = {scala\*m.d, scala\*(-1)\*m.b  ,scala\*(-1)\*m.c, scala\*m.a};  //TODO N次元行列に対応させる  return matInv;  }  /\*\*  FunctionName: countFloat    Argument : f  ArgumentType: ListOfFloat    Return : ElementCount  ReturnType : integer    Description : 渡された単精度浮動点小数リストの要素数を数える関数  \*/  int countFloat(Float\* f) {  int count = 0;  Float\* currentFloat = f;  while (currentFloat != NULL) {  currentFloat = currentFloat -> nextPt;  count++;  }  return count;  }  /\*\*  FunctionName: findRange    Argument : f  ArgumentType: ListOfFloat    Return : range{min, max}  ReturnType : Pair    Description : 渡された単精度浮動点小数リストの総和を求める関数  \*/  Pair findRange(Float\* f) {  Float\* currentFloat = f;  Pair range = {f -> n, f -> n};  while (currentFloat != NULL) {  if (currentFloat -> n < range.n) {  range.n = currentFloat -> n;  }  if (currentFloat -> n > range.m) {  range.m = currentFloat -> n;  }  currentFloat = currentFloat -> nextPt;  }  return range;  }  /\*\*  FunctionName: solveSigma    Argument : f1, f2  ArgumentType: ListOfFloat, ListOfFloat    Return : SummationOfFloat  ReturnType : float    Description : 渡された単精度浮動点小数リストの総和を求める関数  \*/  float solveSigma(Float\* f1, Float\* f2) {  float sum = 0;  if (f1 == NULL && f2 == NULL) {  printf("IllegalArgumentsError\n");  exit(-1);  }  else if (f1 != NULL && f2 != NULL) {  Float \*currentFloat1 = f1;  Float \*currentFloat2 = f2;  while (currentFloat1 != NULL || currentFloat2 != NULL) {  sum += currentFloat1 -> n \* currentFloat2 -> n;  currentFloat1 = currentFloat1 -> nextPt;  currentFloat2 = currentFloat2 -> nextPt;  }  return sum;  }  else if (f1 != NULL || f2 != NULL) {  Float \*currentFloat;  if (f1 == NULL) {  currentFloat = f2;  }  else if (f2 == NULL) {  currentFloat = f1;  }  while (currentFloat != NULL) {  sum += currentFloat -> n;  currentFloat = currentFloat -> nextPt;  }  return sum;  }  return sum;  }  /\*\*  FunctionName: splitToFloat    Argument : p, f1, f2  ArgumentType: ListOfPair, PointerToListOfFloat, PointerToListOfFloat    Return : void  ReturnType : void    Description : ペアで渡された座標をX、Yに分ける関数  \*/  void splitToFloat(Pair\* p, Float\*\* f1, Float\*\* f2) {  Pair \*currentPair = p;  while (currentPair != NULL) {  Float \*newFloat = (Float\*) malloc(sizeof(Float));  if (newFloat == NULL) {  printf("MemoryAllocationError\n");  exit(-1);  }  newFloat -> n = currentPair -> n;  newFloat -> nextPt = NULL;  if (\*f1 == NULL) {  \*f1 = newFloat;  } else {  Float \*currentFloat = \*f1;  while (currentFloat -> nextPt != NULL) {  currentFloat = currentFloat -> nextPt;  }  currentFloat -> nextPt = newFloat;  }  newFloat = (Float\*) malloc(sizeof(Float));  if (newFloat == NULL) {  printf("MemoryAllocationError\n");  exit(-1);  }  newFloat -> n = currentPair -> m;  newFloat -> nextPt = NULL;  if (\*f2 == NULL) {  \*f2 = newFloat;  } else {  Float \*currentFloat = \*f2;  while (currentFloat -> nextPt != NULL) {  currentFloat = currentFloat -> nextPt;  }  currentFloat -> nextPt = newFloat;  }  currentPair = currentPair -> nextPt;  }  }  /\*\*  FunctionName: loadCSV  Argument : filename  ArgumentType: String    Return : ListOfCoordinate  ReturnType : ListOfPair    Description : 座標の入っているCSVファイルからデータをペアでロードしてくる関数  \*/  Pair\* loadCSV(char\* filename) {  FILE \*fp;  fp = fopen(filename, "r");  if (fp == NULL) {  printf("FileNotFoundError\n");  exit(-1);  }  Pair \*points = NULL;  float x, y;  while (fscanf(fp, "%f, %f\n", &x, &y) != EOF) {  Pair \*newPoint = (Pair\*) malloc(sizeof(Pair));  if (newPoint == NULL) {  printf("MemoryAllocationError\n");  exit(-1);  }  newPoint -> n = x;  newPoint -> m = y;  newPoint -> nextPt = NULL;  if (points == NULL) {  points = newPoint;  } else {  Pair \*currentPoint = points;  while (currentPoint -> nextPt != NULL) {  currentPoint = currentPoint -> nextPt;  }  currentPoint -> nextPt = newPoint;  }  }  fclose(fp);  return points;  } |

上記のソースコードを実行した結果を以下に示す

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example1.csv  (0.457143 1.742857) |

グラフ



次にPythonとC言語で計算した結果を比較する。

Python

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % python3 LeastSquaresMethod.py example1.csv  [[0.45714286]  [1.74285714]] |

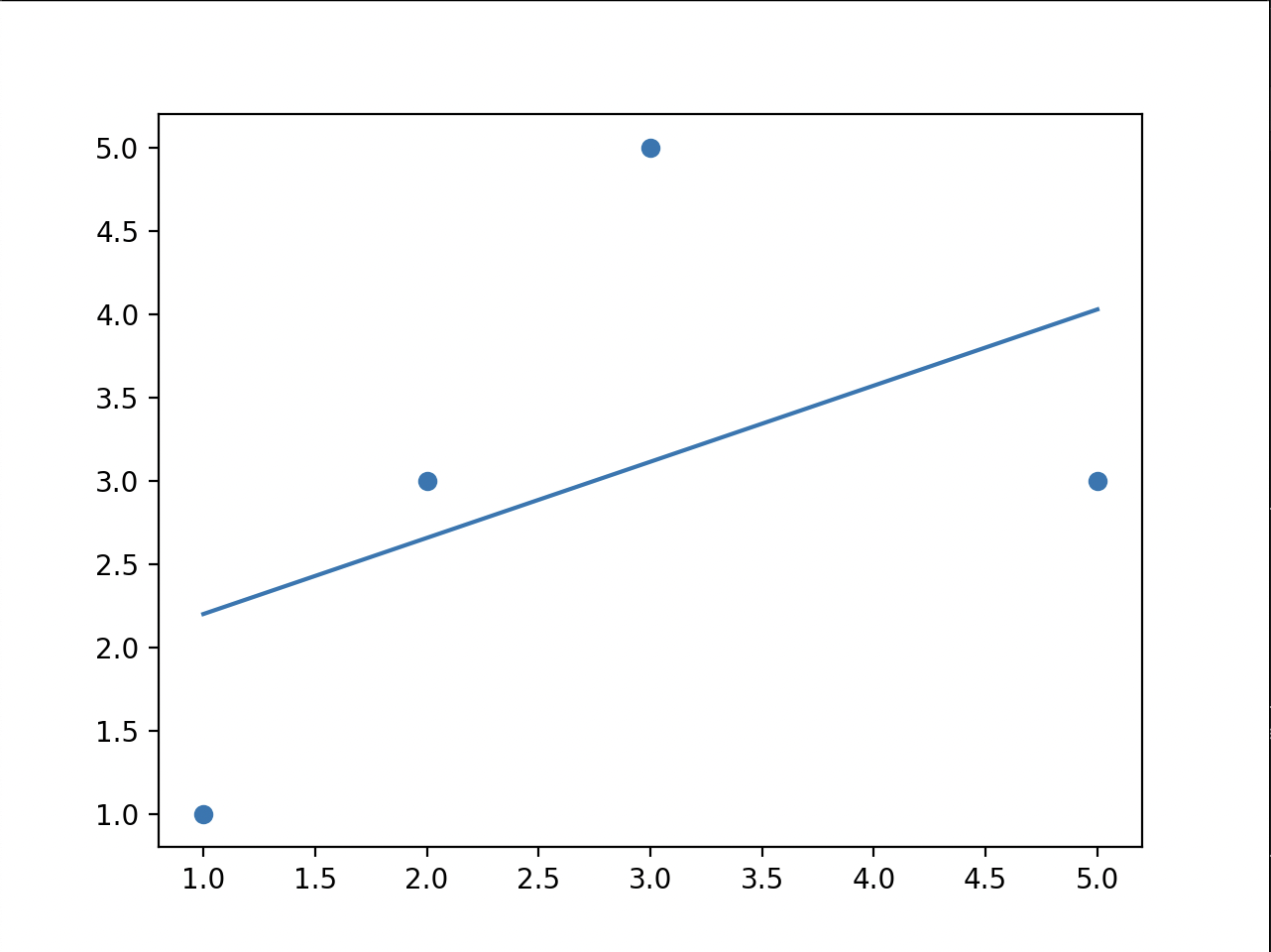
C言語

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example1.csv  (0.457143 1.742857) |

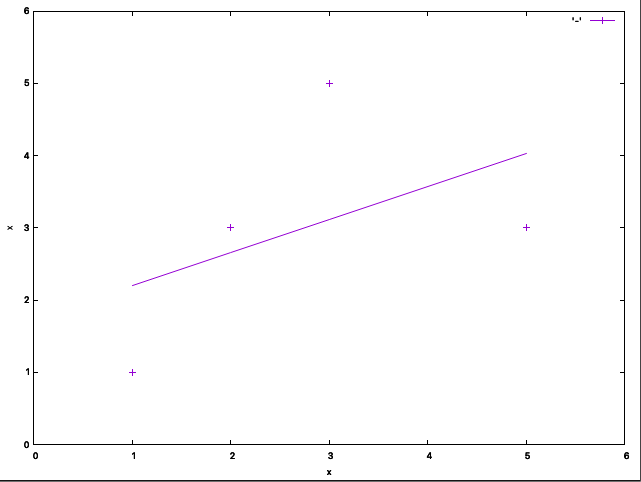
計算できる小数の桁に違いはあるが、同じような結果が出力されている

次にグラフの比較をする。

Python



C言語



プロットされた点の位置、グラフの傾き、切片が一致していることがわかる。

また、データ量が少なくデータに空白が大きいため直線での近似ではそれほど近似できていないように見える。

また、手計算で計算した結果と一致している。このことからこのPythonで書いたプログラムとC言語と書いたプログラムは信頼度が高いと考える。

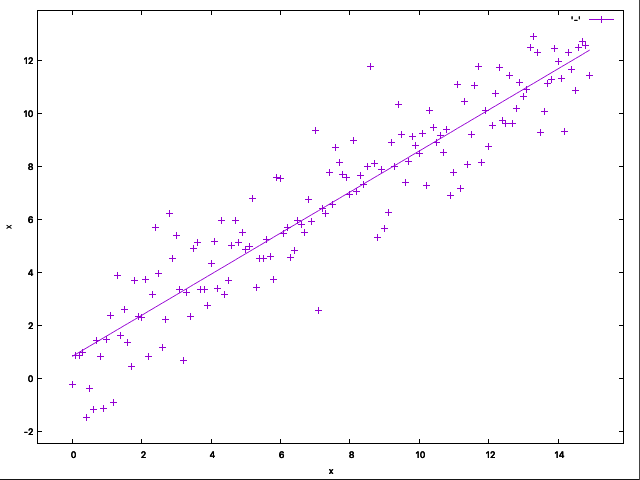
* 1. データexample2.csvに最小二乗法を適用し、直線y=ax+bのパラメータa, bを求めよ。

本問題はC言語で求める。実行した結果を以下に示す。

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example2.csv  (0.774297 0.845703) |

グラフ



このデータはデータ数が多く、外れ値が少ない。またデータが直線的な分布をしているため一次式で近似が可能だと考える。

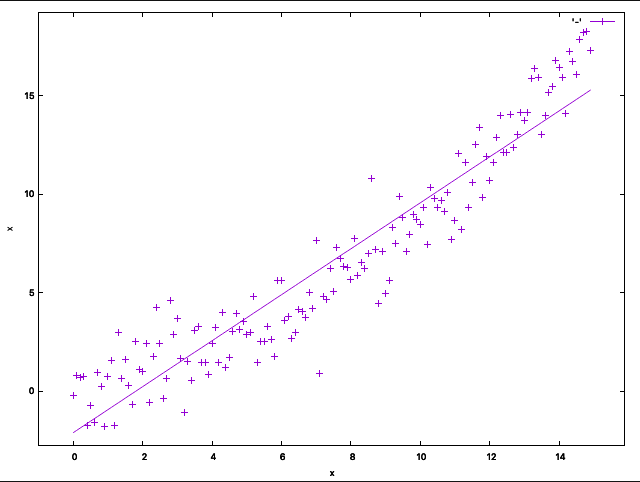
* 1. データexample3.csvに最小二乗法を適用し、直線y=ax+bのパラメータa,bを求めよ

本問題はC言語で求める。実行結果を以下に示す。

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example3.csv  (1.166298 -2.094570) |

グラフ



このデータは先ほどのデータのように直線的なデータに見えるが最小二乗法を適用した直線を引いてみると多少近似ができていないところがあるように考える。

このことからこのデータは一次式でもある程度は近似できるが、二次式で最小二乗法を適用するのが最適だと考える。

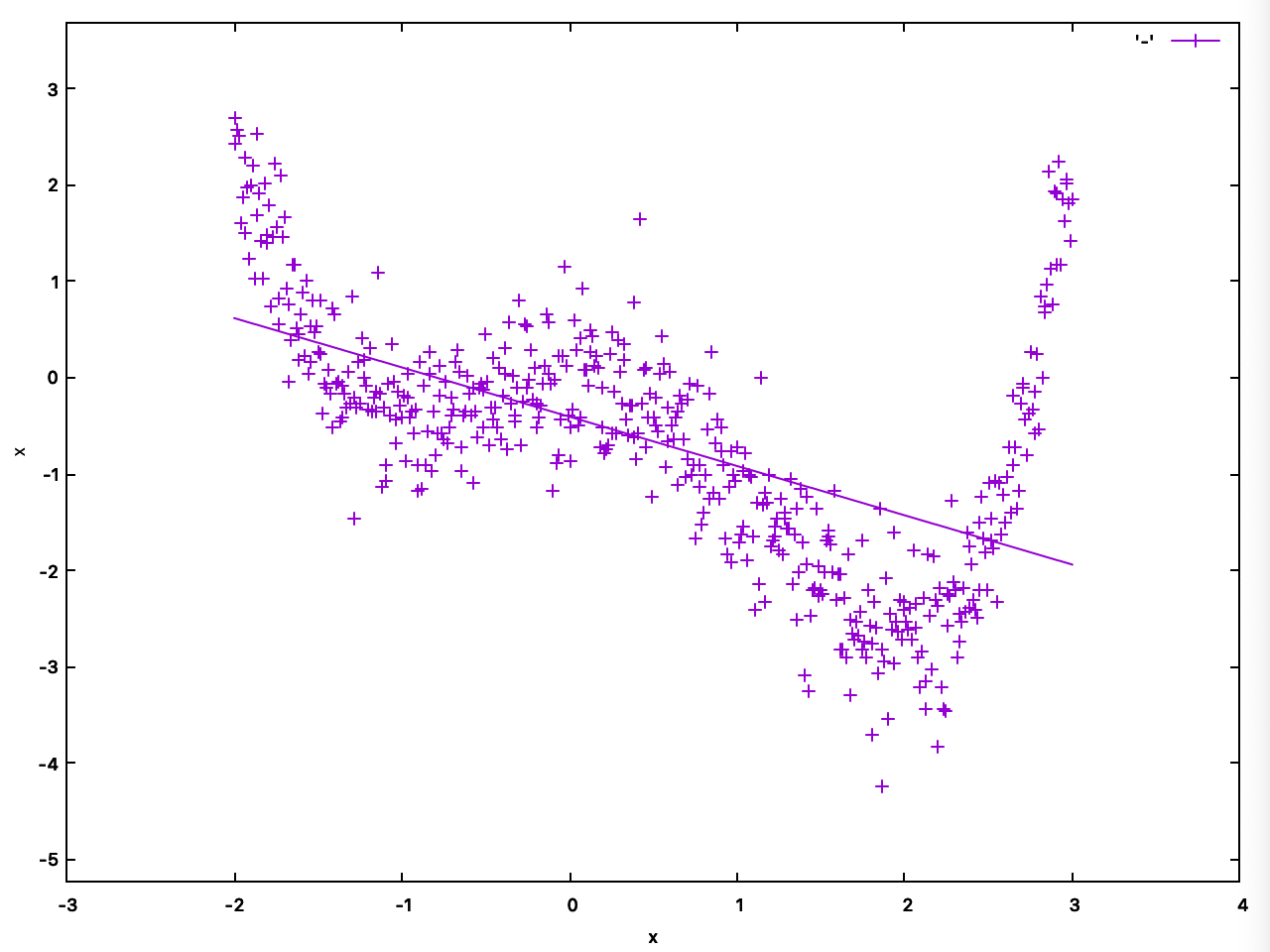
* 1. データexample4.csvに最小二乗法を適用し、直線y=ax+bのパラメータa, bを求めよ。

本問題はC言語で求める。実行した結果を以下に示す。

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example4.csv  (-0.511643 -0.404821) |

グラフ



このデータは曲線的な分布をしている。1次式で最小二乗法を適用すると確かに近似はできているように見えるが大まかな分布の傾きしかわからない。この分布は偶関数のグラフの分布をしており、4次式などで近似をするのが最適だと考える。

* 1. example3.csv, example4.csvに対して、1次式の最小二乗法y=ax+bを適用した時の誤差を求めよ

誤差はRMSE(root meat squared error, 二乗平均平方根誤差)を用いることにする。RMSEの導出式を以下に示す。

は予測した値、は真の値を示す。nはデータ数を表している。

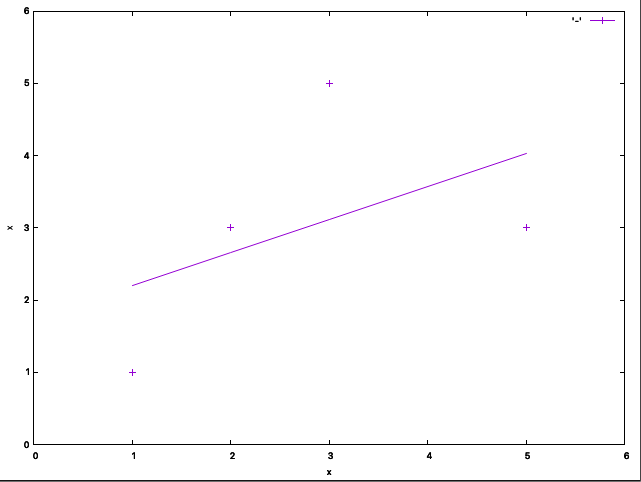
上記の式を今回の直線の方程式に適用した実行結果を以下に示す。

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example1.csv  (0.457143 1.742857)  RMSE=1.242118  ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example2.csv  (0.774297 0.845703)  RMSE=1.323579  ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example3.csv  (1.166298 -2.094570)  RMSE=1.703980.  ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example4.csv  (-0.511643 -0.404821)  RMSE=1.067494  ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example5.csv  (-0.007910 25.592649)  RMSE=0.056124 |

RMSEは0に近いほど誤差がないという指標である。

この値だけ見ていてもどうにもならないのでグラフと比較して考察していく。

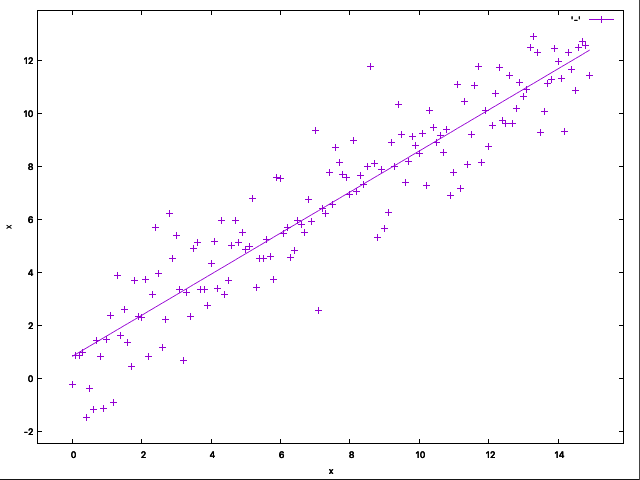
* + 1. example1.csv



RMSE = 1.242118

データ数が少なく、データに空きがあるのでRMSEの値は比較的大きくなっていると考える。

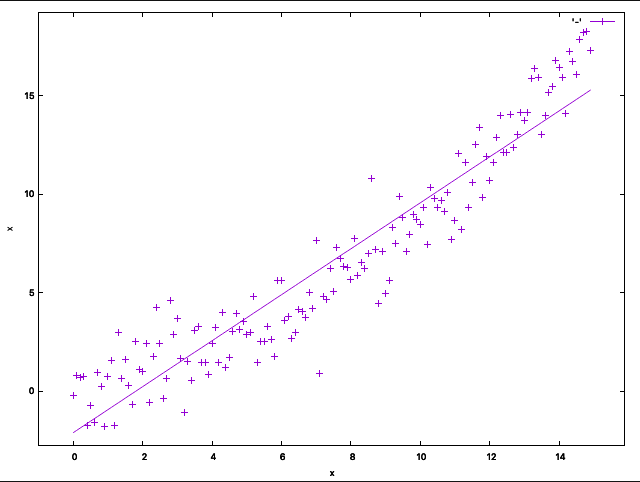
* + 1. example2.csv



RMSE=1.323579

データは多く全体的には線形的な分布になっているが分布の濃度が一定でY軸の±方向にかなり幅広い分布をしているためRMSEの値の大きさも比較的大きくなっているということが考えられる。

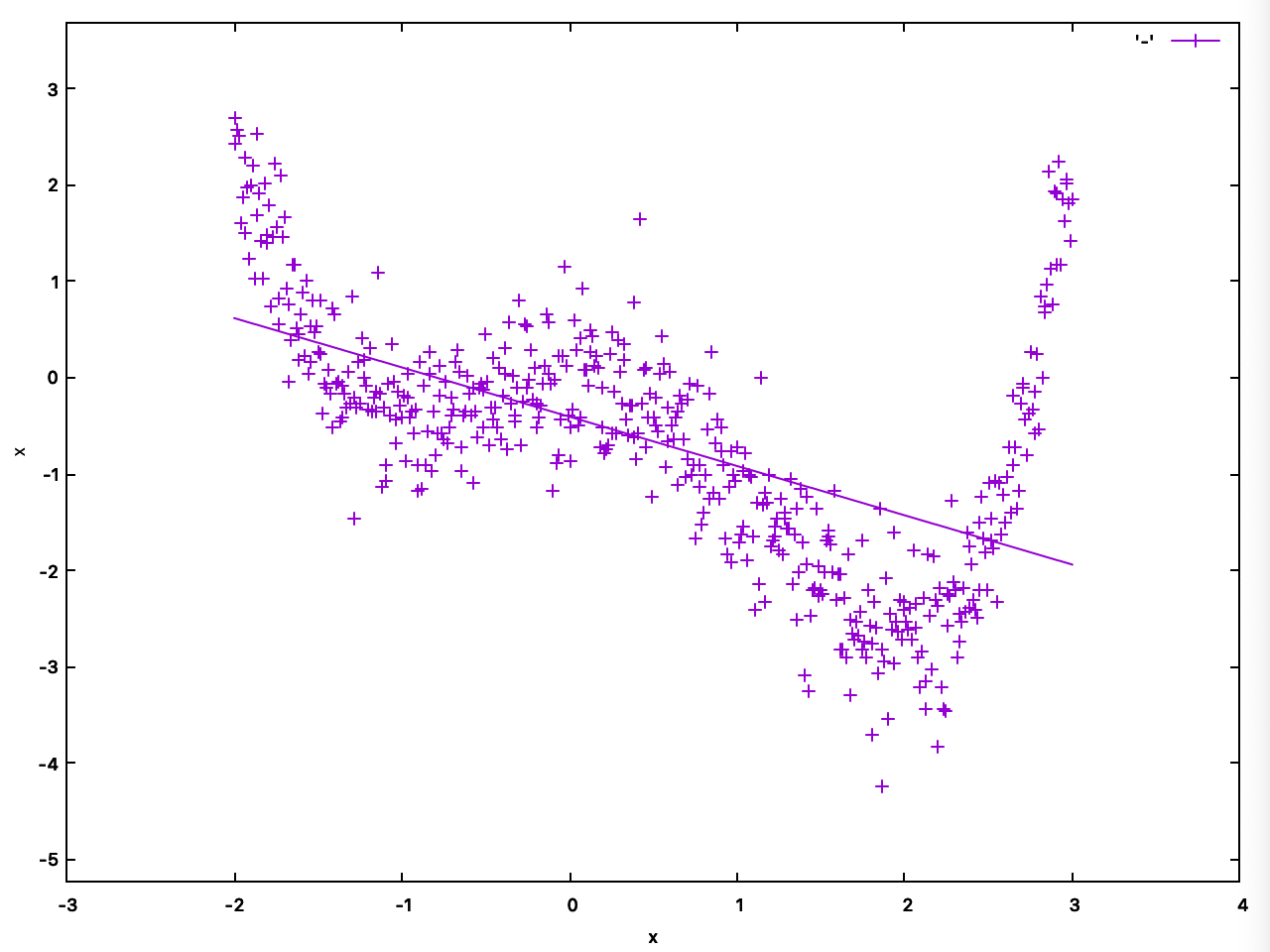
* + 1. example3.csv



RMSE=1.703980.

データの濃度は先ほどよりは濃いと考えるがデータが曲線的に分布しておりその曲線を直線で近似しようとしているためRMSEの値はこのCSVのデータの中で一番大きな誤差になっていると考える。

* + 1. example4.csv

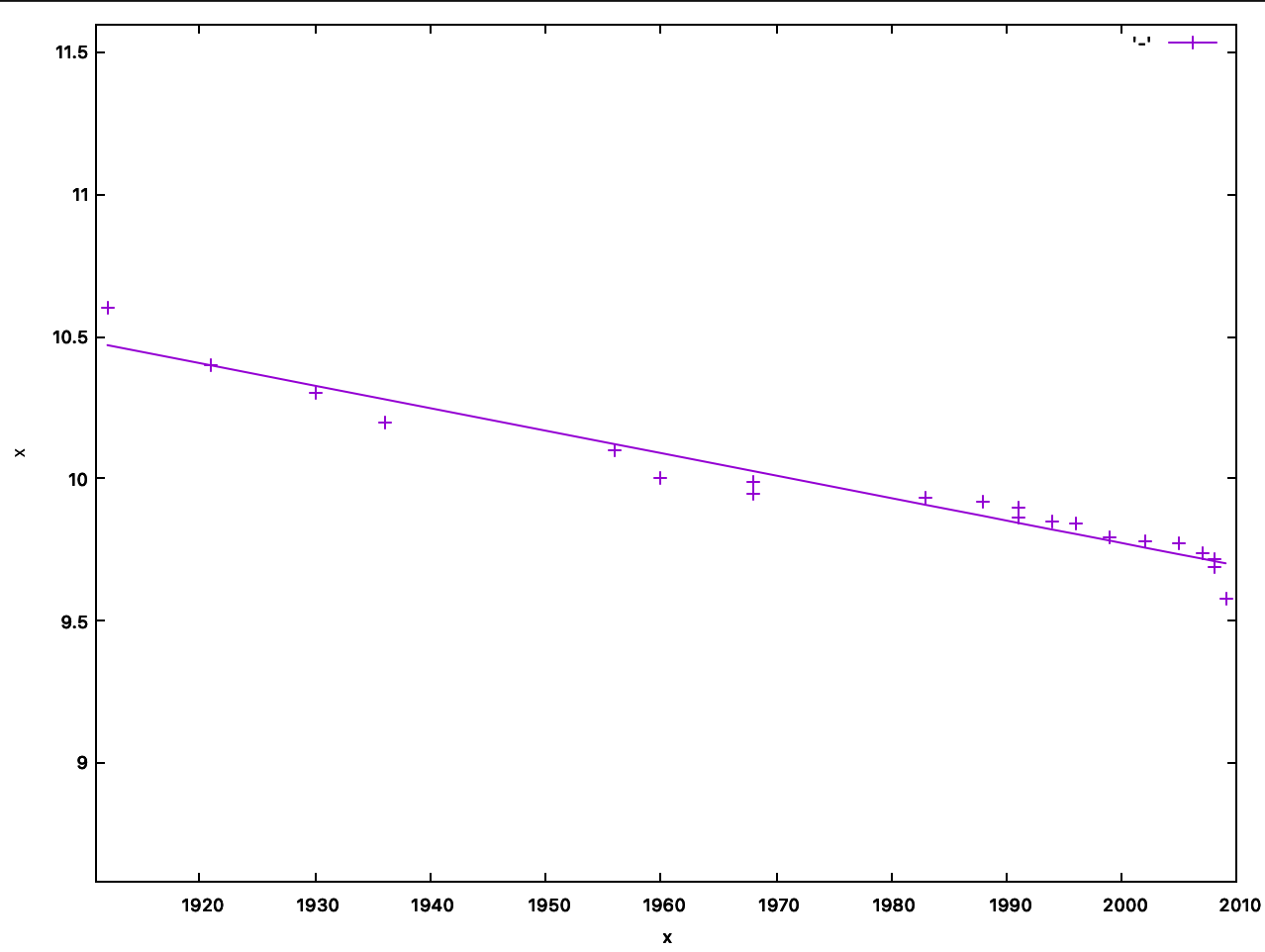


RMSE=1.067494

このデータは見たところ曲線的で直線では近似できないのかと考えていたがRMSEの値を見ると以上３つの値よりも小さいことがわかった。これによりこのデータ群は直線でも近似がかなりできていると考える。

また、なぜこのような結果になったか考察してみた。このデータは以上３つのデータと比べグラフの高低差が大きいということがわかるがデータの濃度が他３つに比べ多いことがわかる。また、データがX座標－１から2が一番濃度が高くその範囲の分布はあまりグラフの高低差がない。そのため以上３つのデータよりも誤差は少なくなったのだと考える。

* + 1. example5.csv



RMSE=0.056124

このデータはかなり直線的な分布になっており、高低差もあまりないグラフになっている。そのためRMSEの値もかなり低くなっており、正確な近似ができていると考える。

* 1. データexample5.csvに最小二乗法を適用し、直線y=ax+bのパラメータa, bを求めよ

本データはC言語で求める。実行した結果を以下に示す。

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % ./a.out example5.csv  (-0.007910 25.592649)  RMSE=0.056124  (1) xが2050の時のyの値を推定せよ  Ans=9.377478  (2) yが9.0になるxを推定せよ  Ans=2097.722412 |

今の所陸上100mのタイムは比例的に縮まってはいるが、これからも比例して縮まると仮定できるのかはわからない。しかし統計的にはこの直線に代入することでタイムと年数を推定することができる。

* 1. データexample3.csv, example4.csvに対して当てはまりを良くするために、1じ式ではなく、2次、3次、4次の最小二乗法を適用してみよ

この時RMSEを計測し、評価せよ

まずN次の最小二乗法の正規方程式を導出する

N個のデータにn次式 を当てはめる。

上記の式のパラメータを最小二乗法

が最小になるように定める。そこで各々の変数について偏微分をしてあげれば良い

これを解いてを定めれば良い。上記の式をで偏微分すると次式を得る。

これを０と置いてk＝0, 1,…, nに対する式を並べると次の正規方程式を得られる

これをプログラムに実装して適用させてみる。最初は2次式で近似させる。example3.csvに適用させる。

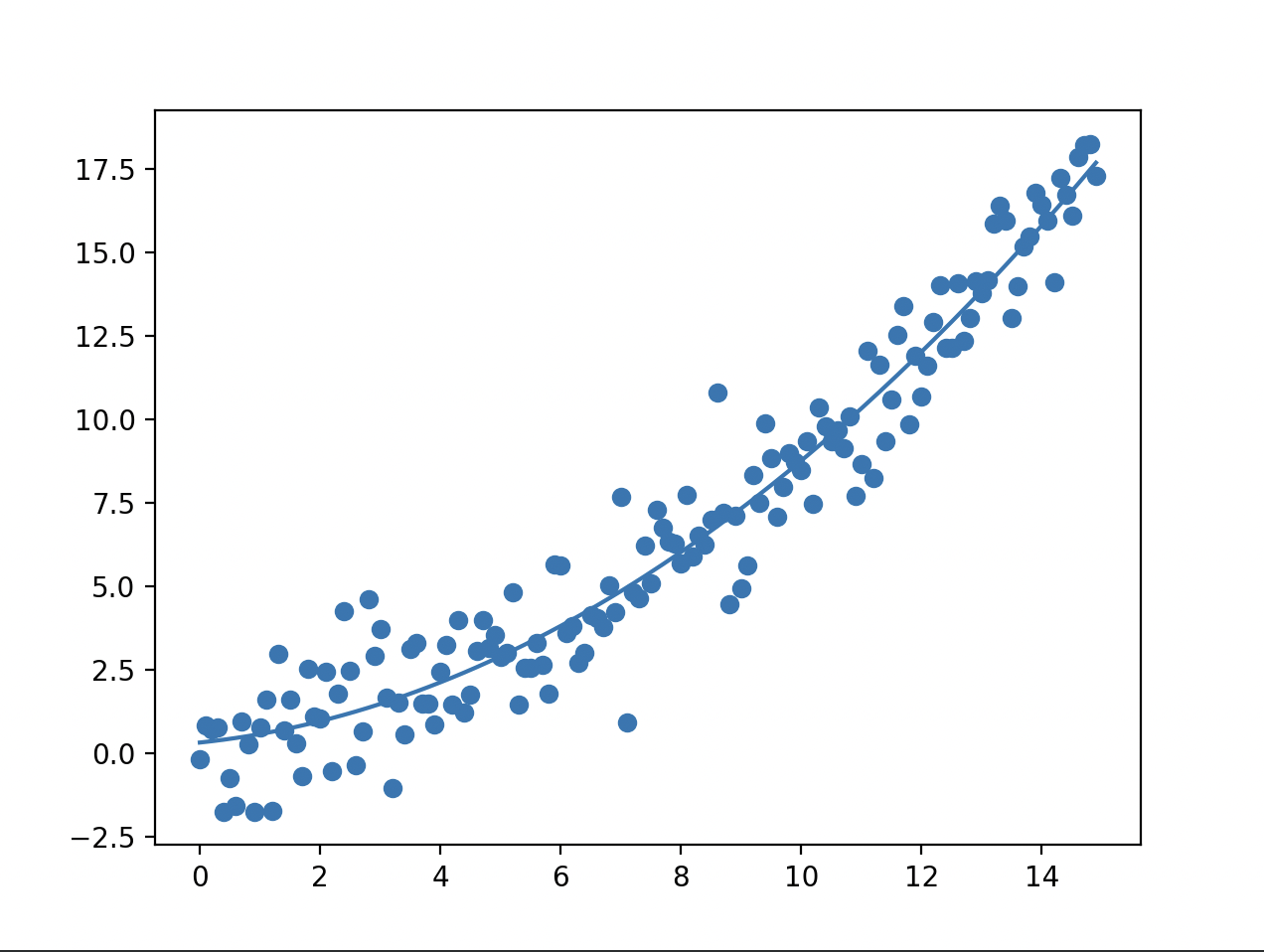
本プログラムはPythonで実装する。実行した結果を以下に示す。

・2次式

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2j4pro0623% python3 LeastSquaresMethod.py example3.csv  [[0.06559876]  [0.18887581]  [0.31640386]]  rmse=[1.30136226] |

グラフ



RMSE=1.30136226

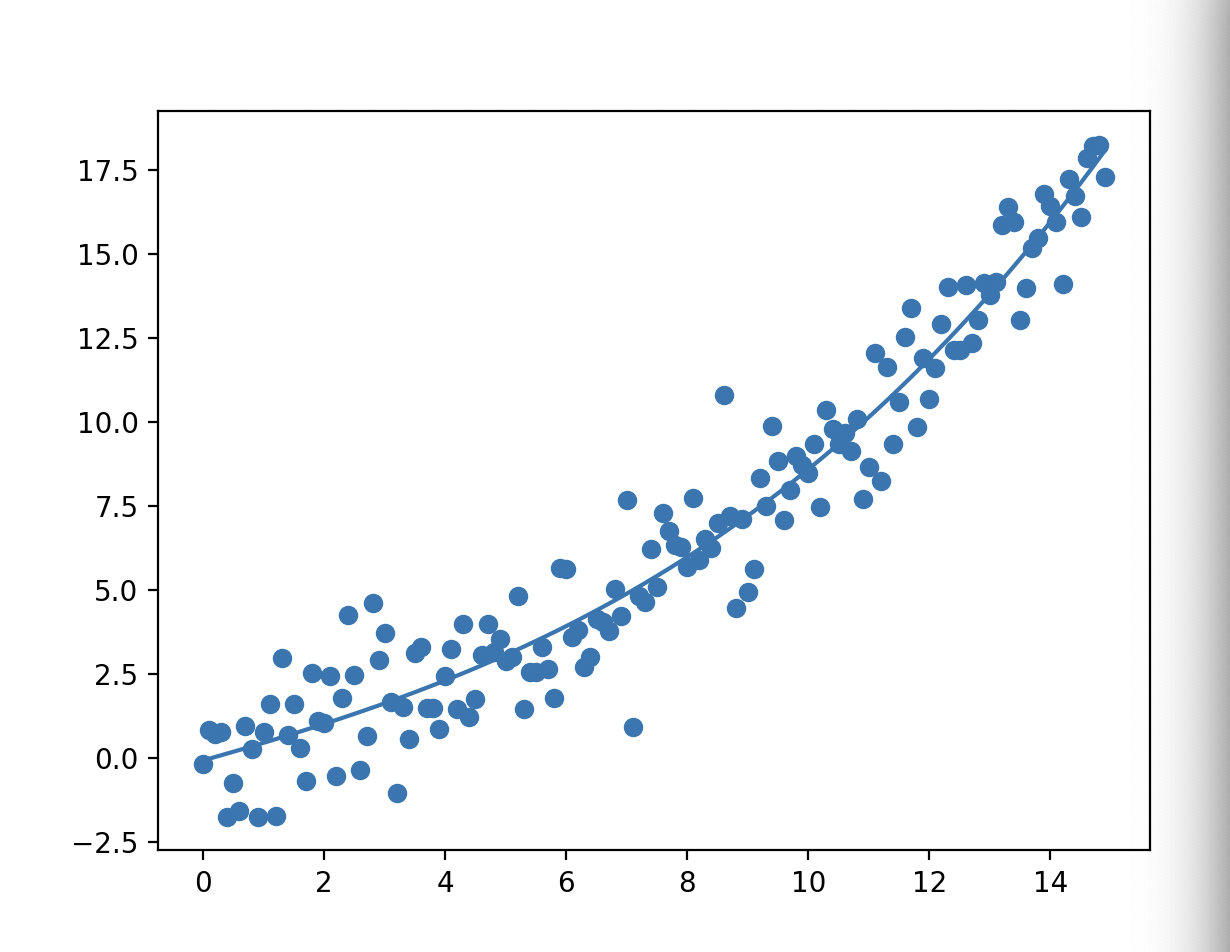
次は3次式に対応させる。実行した結果を以下に示す。

・3次式

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2j4pro0623% python3 LeastSquaresMethod.py example3.csv  [[ 0.00243287]  [ 0.01122408]  [ 0.51186629]  [-0.07792134]]  rmse=[1.2920837] |

グラフ



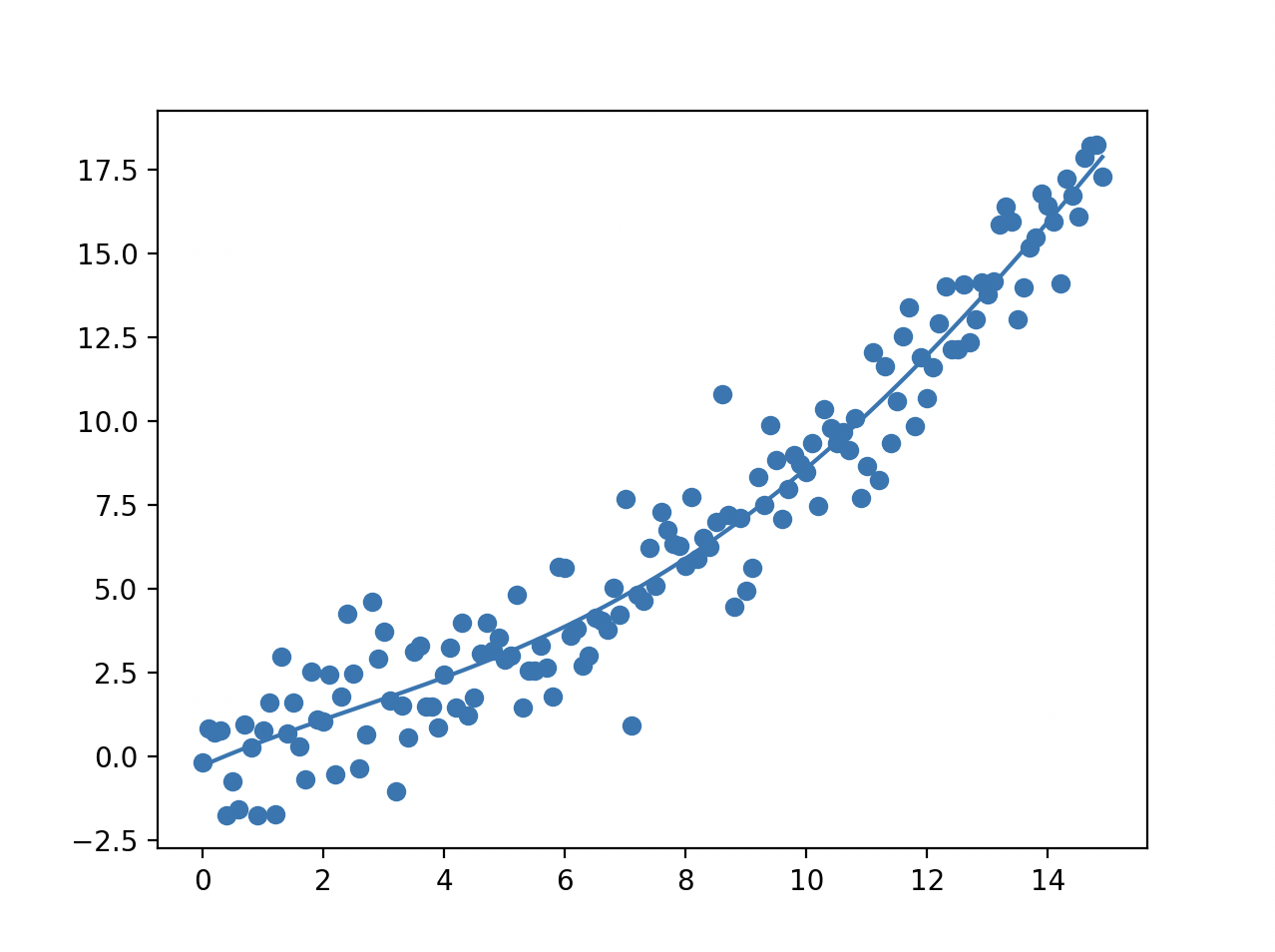
RMSE=1.2920837

・4次式

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2j4pro0623% python3 LeastSquaresMethod.py example3.csv  [[-3.01412526e-04]  [ 1.14149650e-02]  [-7.46212226e-02]  [ 7.93904021e-01]  [-2.81710665e-01]]  rmse=[1.29004169] |

グラフ



RMSE=1.29004169

次はexample4.csvに適用させる。

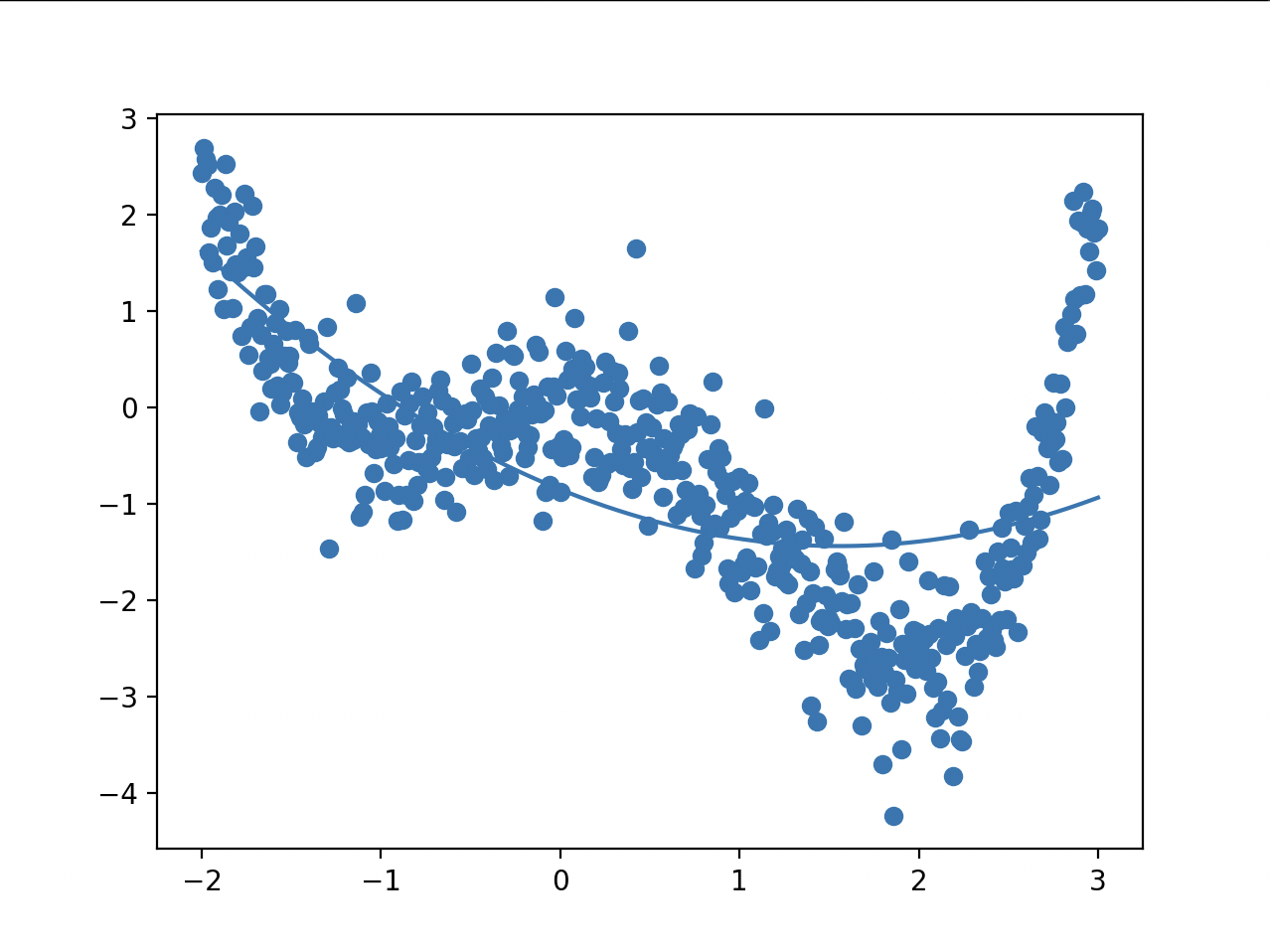
本プログラムはPythonで実装する。実行した結果を以下に示す。

・2次式

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2j4pro0623% python3 LeastSquaresMethod.py example4.csv  [[ 0.24170879]  [-0.75335176]  [-0.84996811]]  rmse=[0.96698597] |

グラフ

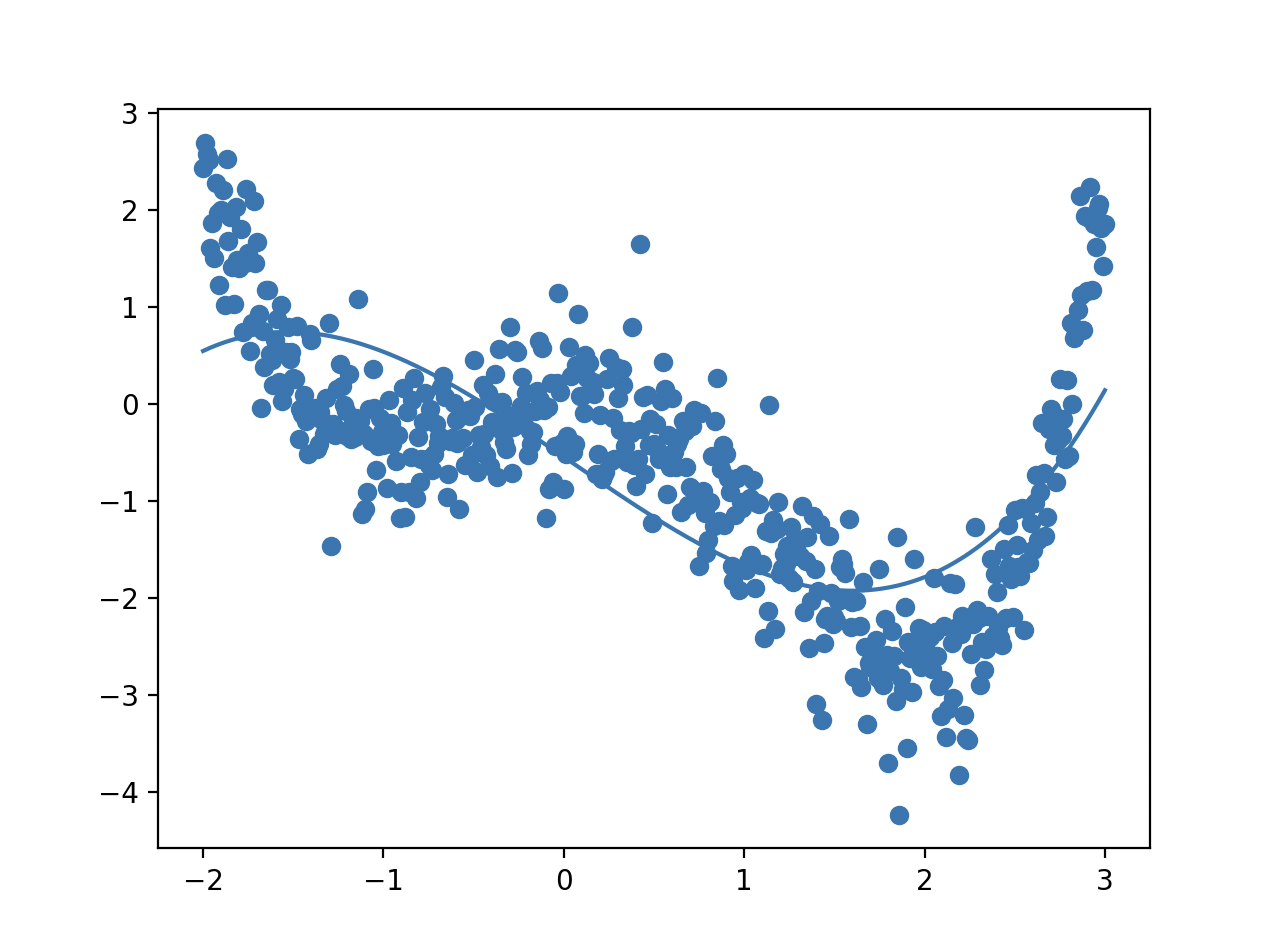


RMSE=0.96698597

・3次式

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2j4pro0623% python3 LeastSquaresMethod.py example4.csv  [[ 0.17328105]  [-0.01821279]  [-1.27579067]  [-0.5454284 ]]  rmse=[0.87492454] |



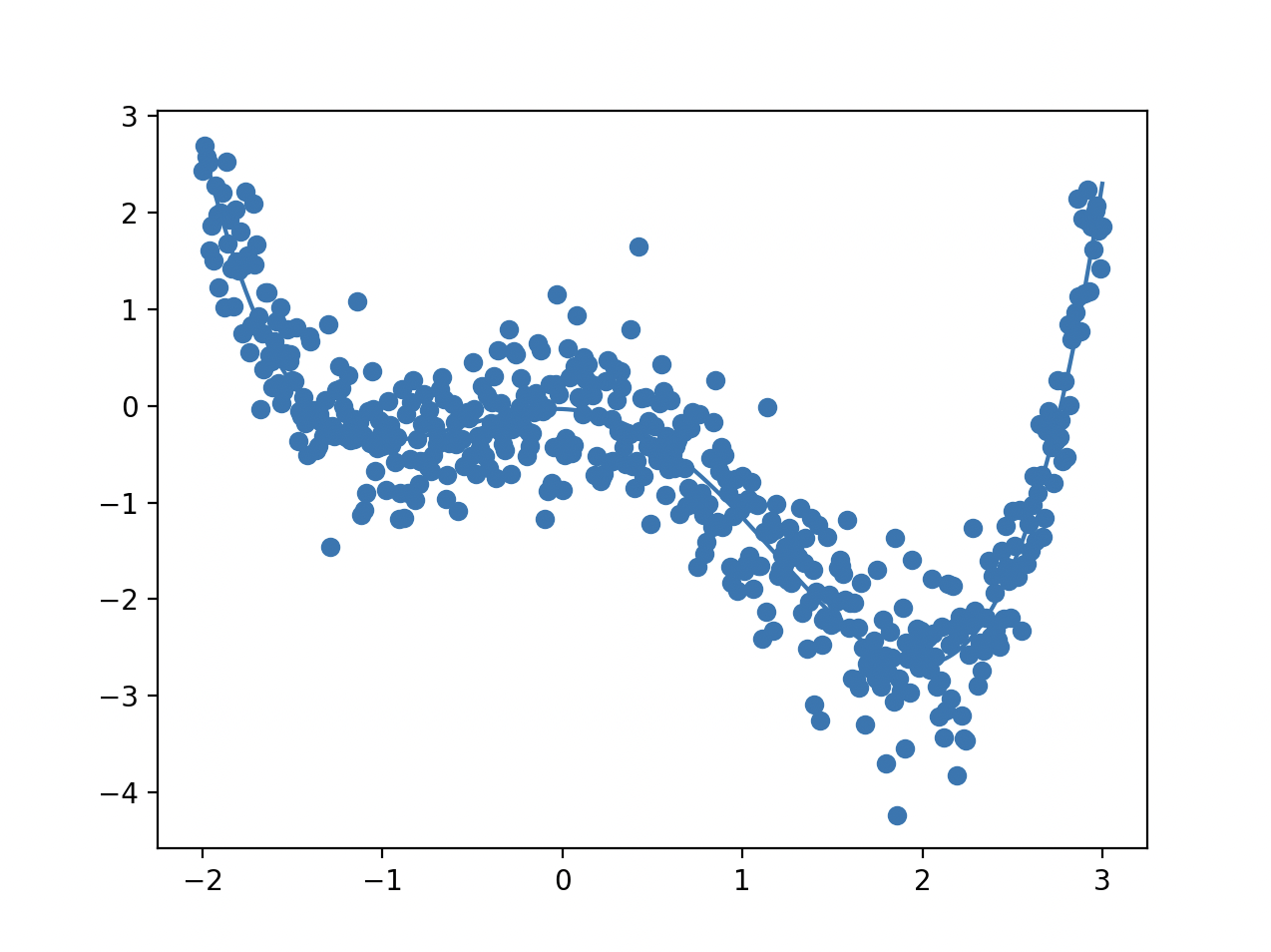
RMSE＝0.87492454

・4次式

コンソール

|  |
| --- |
| ji1wxs@kitanomasakinoMacBook-Pro-2 j4pro0623 % python3 LeastSquaresMethod.py example4.csv  [[ 0.24420137]  [-0.3151217 ]  [-0.96534782]  [-0.08445427]  [-0.03435863]]  rmse=[0.47835416] |

グラフ



RMSE=0.47835416

各々の結果を表にまとめた。

表2.9.1 example3.csvの最小二乗法比較

|  |  |  |
| --- | --- | --- |
| 次数 | RMSE | グラフ |
| 1次 | 1.703980 |  |
| 2次 | 1.30136226 |  |
| 3次 | 1.2920837 |  |
| 4次 | 1.29004169 |  |

これらの結果からexample3.csvのデータの近似式の次数が上がると誤差は小さくなることがわかった。また、2次以上の近似式に代入しても誤差率はあまり変わらないことがわかった。

表2.9.2 example4.csvの最小二乗法比較

|  |  |  |
| --- | --- | --- |
| 次数 | RMSE | グラフ |
| 1次 | 1.067494 |  |
| 2次 | 0.96698597 |  |
| 3次 | 0.87492454 |  |
| 4次 | 0.47835416 |  |

これらの結果からexample3.csvのデータの近似式の次数が上がると誤差は小さくなることがわかった。また3次までの近似はそこまで誤差が縮まるということはなかったが4次の近似にした際は大幅に誤差が縮まった。

1. 考察

これらの結果から最小二乗法の近似式の次数を上げると誤差は少なくなることがわかった。

また、データの分布から近似させる式の次数を決める必要があると考える。

最初から高次元の式に適用することも可能だとは考えるが、計算量が多くなることに対して誤差があまり変わらないため最適な次数を指定してあげることが大事だと考える。

適切な次数の近似式でほとんど理想の値が取れるためその都度変えていくのが大事だと考える。

また、誤差はデータの濃度によっても変化し、平面上の直線で近似する場合、データ濃度が高い場合は誤差が少なくなりデータ濃度が低い場合は誤差が大きくなると考える。

この場合濃度の軸をZ方向に伸ばし立体のグラフで表すと濃度も考慮したグラフができると考えた。