解析用クラス ユーザマニュアル

アドバンスソフト(株)

2011 年 10 月

目次

1. 概要

ElementContainer のヒストグラムデータを解析するためのクラスを説明する．ElementContainer のデータに対して平滑化，ピーク探索，ピークフィットを行うことができる．これらを行うためのドライバーと，実際の計算アルゴリズムがクラスとして実装されている．

ドライバークラスの一覧を表 1.1 に示す．これらのクラスは共通の方法で使用できるように設計されている．平滑化，ピーク探索，ピークフィットそれぞれに固有の操作が必要になった場合のみ，その操作が拡張されている．

表 1.1 ドライバークラス

クラス	Smoothing	PeakSearch	PeakFit
用途	誤差を含む測定値の平滑化	誤差を含む測定値のピーク探索	誤差を含む測定値を任意の関数でピークフィット
使用可能なアルゴリズム	B-Spline，移動平均	B-Spline，移動平均	Levmar

表 1.2　計算アルゴリズムを実装したクラスの説明と用途.

クラス名	BSpline	MovingAverage	Levmar
用途	測定値に対する B-スプライン関数のあてはめ，および指定された座標点での関数の値，微分係数の計算.	各ビンでの移動平均の値，関数の一階および二階の微係数に相当する差分商の算出.	Levanberg-Marquiate 法による関数のあてはめ，および指定された座標点でのあてはめた関数の値，微係数の計算.
平滑化	可	可	
ピーク探索†	可	可	
ピークフィット			可
備考	GNU Scientific Library[1]を使用		Levmar-2.5[2] のラッパークラスであるため，Levmar-2.5 のライブラリが必須. Levmar-2.5 の全機能を利用するには，さらに Lapack かその互換ライブラリが必要.

†内部で前処理（データ平滑化）のために使用

ドライバークラスは，ソースデータとなる ElementContainer のインスタンスと計算アルゴリズムを実装したクラスのインスタンスを渡す必要がある．

これらのクラスには，実行に必要なデータを ElementContainer，Domain および ParamSet のインスタンスとして与える必要がある．ElementContainer は加工したい（ヒストグラム）データが入っている．Domain は ElementContainer のデータの加工したい範囲を設定するクラスである．ParamSet はその他必要なデータを持つ．

1.1. ElementContainer のヒストグラムの構造

ビンの総数が N であるヒストグラムを持つ ElementContainer を考える．全てのビンの境界に横軸の座標$\{X_i; 0 \le i \le N\}$が昇順に割り振られている．各ビンは閉開区間$I_i = [X_i, X_{i+1})$である．各ビンには測定値$\{Y_i; 0 \le i < N\}$とその誤差$\{E_i; 0 \le i < N\}$が割り当てられている．当然のことであるが，区間$I_{-1} = (-\infty, X_0)$と$I_N = [X_N, \infty)$はヒストグラムの範囲外である．

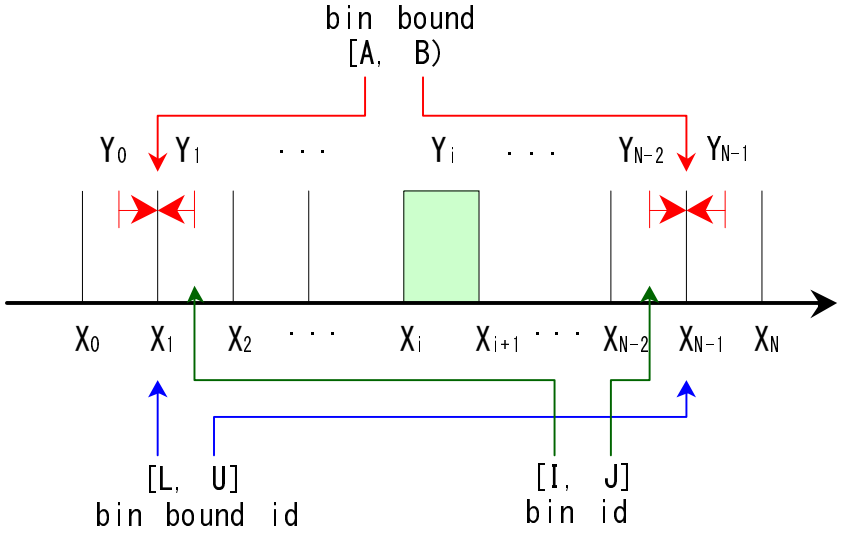


図 1.1 ElementContainer のヒストグラムの構造と Domain による範囲指定.

1.2. 計算対象範囲の指定

Domain は，ElementContainer の処理したい範囲を保持するクラスである．処理したい範囲は，ここからここまでという区間で指定する．場所は横軸の実数でも，ビンの番号でも指定できる．実数で区間$I = [A, B)$を指定したとする．実際の区間は A と B のそれぞれに最も近いビン境界が探索され，処理対象はこれらの間にある全てのビンとなる．ビンの番号で指定した場合は，指定した番号がヒストグラムの範囲内にあるかどうかをチェック

される．

リスト 1.1　計算する範囲の指定

```
ElementContainer src;    //  ソースデータ
Double A, B;
Double I, J;

// ドライバーの生成,
//    Opeation は Smoothing, PeakSearch, PeakFit のいずれか
//    Method クラスのインスタンスをソースデータの同時に登録する．
Opearation op = *(new Operarion(src, (new Method())));

op.setRange(A, B);    // 計算対象を区間 I = [A,B) に設定，A と B は実数
op.setRange(I, J);    // 計算対象を bin 番号で指定
```

1.3.　計算アルゴリズムに必要なパラメータの設定

ピークサーチ，ピークフィットなどで使用する計算アルゴリズムに必要なパラメータについては，デフォルト値が利用できるものもあれが，ユーザは必ず設定しなければならないものもある．使用できるパラメータの詳細は，使用するアルゴリズムの項を参照して欲しい．

設定可能なパラメータの型は表 1.3 に示す通りである．パラメータの値を設定するためには，パラメータを区別するための文字列キーとパラメータの値の組を set メンバー関数に与える．既存のパラメータの値の書き換えも同じ方法で可能である．つまり，文字列キーと別の値を set メンバー関数に与える．使用例をリスト 1.2 に示す．

表 1.3　パラメータとして使用できるデータ型.

データ型	説明
Bool	論理値，スカラー
UInt4	符号なし整数．スカラー
Int4	整数．スカラー
Double	倍精度実数．スカラー
vector<Double>	倍精度実数のベクトル（配列）
vector< vector<Double> >	倍精度実数の行列（二次元配列）
vector<FuncBase*>	パラメータフィッティングを行う関数（を指すポインタ）のリスト(配列)．ピークフィット用.

リスト 1.2　パラメータの設定

```
ElementContainer src;    //  ソースデータ
Double A, B;
Double I, J;

// ドライバーの生成,
//    Operation は Smoothing, PeakSearch, PeakFit のいずれか
//    Method クラスのインスタンスをソースデータの同時に登録する.
Opearation op = *(new Operarion(src, (new Method()));

op.setRange(A, B);   // 計算対象を区間 I = [A,B) に設定, A と B は実数
op.setRange(I, J);      // 計算対象を bin 番号で指定

op.setParam(key, value);              // 値の設定
op.setParam(key, another_value);     // 値の再設定
```

2.　ドライバークラス

2.1.　概要

平滑化，ピーク探索，ピークフィット等のインターフェースを提供するクラスである．実際の計算アルゴリズムは，あるインターフェースを持つクラスとして分離されており，同一のインターフェースを持つクラスであるならば，差し替えることができる．

2.2.　共通メソッド

ドライバークラスに共通のメンバー関数を表 2.1 に示す．

表 2.1　ドライバークラスに共通なメンバー関数

関数	説明
void setSource(ElementContainer *src)	データソースとして ElementContainer を登録
void setMethod(Method *method)	計算アルゴリズムを登録

void setMethod(MethodType &methodType)	計算アルゴリズムを登録
void setMethod(string &method)	計算アルゴリズムを登録
void setDomain(ElementContainer *src, Double xLower, Domain xUpper);	計算範囲を指定
void setDomain(ElementContainer *src, UInt4 lower, UInt4 upper)	計算の対象範囲を指定
void setDomain(Double xLower, Double xUpper)	計算範囲を指定，データソースは指定済みでなければならない.
void setDomain(UInt4 lower, UInt4 upper)	計算範囲を指定，データソースは指定済みでなければならない.
Domain getDomain()	計算範囲を取得.
Double getLowerBound()	計算対象範囲の下限（である bin 境界値）
Double getUpperBound()	計算対象範囲の上限（である bin 境界値）
UInt4 getLowerBoundID()	計算対象範囲の下限（である bin 境界値）の配列番号
UInt4 getUpperBoundID()	計算対象範囲の上限（である bin 境界値）の配列番号
void setDefaultParamSet()	計算アルゴリズム用パラメータのうちデフォルト値があるものにデフォルト値に設定
void setParam(ParamSet param)	パラメータに与えられた値を設定
void setParam(string key, Bool value)	
void setParam(string key, UInt4 value)	
void setParam(string key, Int4 value)	
void setParam(string key, Double value)	
void setParam(string key, vector<Double> value)	
void setParam(string, key, UInt4 I, Double value)	指定されたキーを持つベクトルの i 番目の成分を value

	にする.
void setParam(string key, vector< vector<Double> >)	パラメータに与えられた値を設定
void setParam(string key, UInt4 i, UInt4 j, Double value)	指定されたキーを持つパラメータの(i, j)成分の値を設定する.
Bool checkParam()	計算アルゴリズムのパラメータが，ソースデータ，計算対象範囲と整合しているかを調べる．整合してば true を，そうでなければ false を返す.
ParamSet getParam()	アルゴリズム用のパラメータを取得する.
ParamSet getFittedParam()	フィットした後のパラメータの値を取得する.
void execute()	計算アルゴリズムを実行.
ElementContainer getResult()	計算アルゴリズムでデータソースを処理した結果を返す.

2.3. 平滑化ドライバー Smoothing クラス

2.3.1. 概要

平滑化を行うためのドライバークラスである．平滑化を実際に行うアルゴリズムを実装した BSplline クラスや MovingAvreage クラスを使用できる．このクラスに固有のメソッドはない.

2.3.2. 使用例

Smoothing クラスの C++での使用例をリスト 2.1 に示す.

リスト 2.1　C++での Smoothing クラスの使用例.

```
 1: #include “EhementContainer.hh”
 2: #include  “PeakData.hh”          // ピークデータのヘッダー
 2: #include  “BSpline.hh”           // 平滑化法のヘッダー
 3: #include  “Smoothing.hh”         // ドライバーのヘッダー
 4:
 5: ElementContainer src;       // ソースデータ
 6: ElementContainer result;    // 平滑化の結果
 7:
 8: Smoothing *smoothing = new Smoothing(&src, new BSpline);    // ドライバーの生成
 9: smoothing->execute();                              // 平滑化の実行
10: result=smoothing->getResult();                     // 結果の取り出し.
11: ParamSet param=smoothing->getFittedParam();    //  計算パラメータの取り出し
12: param.dump();                                      // パラメータの表示
13:
14: smoothing->setParam(BSpline::ORDER, 3);                              // 多項式の次数
15: smoothing->setParam(BSpline::USE_UNIFORM_BREAK_POINTS, true);  // 均等な区切り点
16: smoothing->setParam(BSpline::NUMBER_OF_BREAK_POINTS, 10);      // 計算領域を 9 分割
17: smoothing->setDomain(1.0, 14.0);      // 計算する範囲の指定:実数
18: if ( smoothing->checkParam() ) {      // 設定したパラメータの整合性の検査
19:     smoothing->execute();                 // 平滑化の実行
20: }
21: result=smoothing->getResult();  // 平滑化後のデータの取得
```

リスト 2.2　Python での Smoothing クラスの使用例．緑色の文字 Smoothing はテスト用モジュール名である．実際のモジュール名に読み替えること．

```
 1: src=Manyo.ElementContainer()       # ソースデータ
 2:
 3: smoothing=Smoothing.Smoothing(src, Smoothing.BSPLINE)    # ドライバーの生成
 4: smoothing.exececute()                          # 平滑化の実行
 5: result=smoothing.getResult()                   # 結果の取り出し.
 6: param=smoothing.getParam()                     #  計算パラメータの取り出し
 7: param.dump()                                   # パラメータの表示
 8:
 9: smoothing.setParam(Smoothing.BSpline.ORDER, 3)  #多項式の次数
10: smoothing.setParam(Smoothing.BSpline.USE_UNIFORM_BREAK_POINTS, True) # 均等区切り点
11: smoothing.setParam(Smoothing.BSpline.NUMBER_OF_BREAK_POINTS, 10) # 計算領域を 9 分割
12: smoothing.setDomain(1.0, 14.0)     #  計算する範囲
13: if smoothing.checkParam():         # 設定したパラメータの整合性の検査
14:     smoothing.execute()            # 平滑化の実行
15: result=smoothing.getResult()       # 平滑化された結果の取り出し
```

1～3 行目　ヘッダーファイルのインクルード

8 行目　平滑化ドライバーの生成．ソースデータの平滑化に使用する方法として B-spline を指定している．ソースデータも計算法のインスタンスもポインターで渡している．

9 行目　平滑化の実行.
10 行目　平滑化を行った結果を取り出す．結果は ElementContaiter として返す.
14～16 行目　平滑化のために BSpline クラスのパラメータを設定している.
17 行目　平滑化を行う範囲を設定している.
18 行目　設定したパラメータの整合性を検査する.
19 行目　平滑化の実行
21 行目　平滑化の結果を取り出す.
同様に Python での使用例をリスト 2.2 に示す.

2.4. ピーク探索ドライバー PeakSearch

2.4.1. 概要

このクラスに固有のメソッドを表 2.2 に示す.

表 2.2　ピーク探索に固有のメソッド

メソッド	説明
PeakData getPeaks()	探索したピークデータを取り出す.

2.4.2. 使用例

PeakSearch クラスの C++での使用法をリスト 2.3 に示す.
1～4 行目 ヘッダーファイル
10 行目　ドライバーの生成，ソースデータと平滑化の方法として移動平均（MovingAverage）を使用する．ソースデータと計算法のインスタンスの両方ともポインターで渡してしる.
11 行目　ピーク探索の実行.
12 行目　ピークデータの取り出し.
14 行目　平滑化の結果の取り出し.
17～18 行目　平滑化を行うインスタンスためのパラメータの設定.
19 行目　平滑化およびピーク探索を行う範囲の設定.
20 行目　設定したパアメータの整合性の検査.
21 行目　ピーク探索の実行.
23 行目　ピークデータの取り出し.

リスト 2.3　PeakSearch クラスの C++での使用例.

```
 1: #include  “ElementContainer.hh”
 2: #include  “peakData.hh”
 3: #include  “MovingAverage.hh”
 4: #include  “PeakSearch.hh”
 5:
 6: ElementContainer src;   // ソースデータ
 7: PeakData peakList;       // ピークデータの入れ物
 8:
 9: // ドライバーの生成. ソースデータおよび平滑化メソッドの登録
10: PeakSearch *peakSearch= new PeakSearch(&src, new MovingAverage());
11: peakSearch.execute();                          // ピークサーチの実行
12: peakList=peakSearch.getPeaks();               //結果の取り出し
13: peakList.Dump();                               // ピークデータの表示
14: param=peakSearch.getParam();                   //  計算パラメータの取り出し
15: param.dump();                                  // パラメータの表示
16:
17: peakSearch.setParam(MovingAverage::WINDOW_WIDTH, 111);  // 平均をとる窓の幅
18: peakSearch.setParam(MovingAvreage::WINDOW_UPPER, 55);   // 窓の上端
19: peakSearch.setDomain(0, src.PutSize(src.PutXKey())-1);   // 計算する範囲の指定
20: if ( peakSearch.checkParam() ) {      // 設定したパラメータの整合性の検査
21:     peakSearch.execute();             // ピーク探索の実行
22: }
23: peakList=peakSearch.getPeaks();  // ピークデータの取得
```

リスト 2.4　PeakSearch クラスの Python での使用例. 緑色の文字 Smoothing はテスト用モジュール名である. 実際のモジュール名に読み替えること.

```
 1: src=Manyo.ElementContainer()              # ソースデータ
 2:
 3: peakSearch=Manyo.PeakSearch(src, Manyo.MOVING_AVERAGE) # ドライバーの生成
 4: peakSearch.execute()                      # ピークサーチ
 5: peakList=peakSearch.getPeaks()            # 結果の取り出し.
 6: peakList.dump()                           # ピークデータの表示
 7: param=peakSearch.getParam()               # 計算パラメータの取り出し
 8: param.dump()                              # パラメータの表示
 9:
10: peakSearch.setParam(Manyo.MovingAverage.WINDOW_WIDTH, 111) # 窓の幅
11: peakSearch.setParam(Manyo.MovingAvreage.WINDOW_UPPER, 55)  # 窓の上端
12: peakSearch.setDomain(0, src.PutSize(src.PutXKey())-1)    # 計算する範囲の指定
13: if peakSearch.checkParam():       # 設定したパラメータの整合性の検査
14:     peakSearch.execute()                 # 平滑化の実行
15: peakList=peakSearch.getPeaks()
```

2.5. ピークフィットドライバー PeakFit

2.5.1. 概要

このクラスに固有のメソッドを表 2.3 に示す.

表 2.3 ピークフィットドライバクラス PeakFit に固有のメソッド

メソッド	説明
ElementContainerArray getFuncComponets()	関数成分ごとに各ビンでの値と推定誤差を返す.

2.5.2. 使用例

2.6. ピークフィットドライバーMultiDatapeakFit

2.6.1. 概要

2.6.2. 使用例

リスト 2.5 MultiDataLevmar クラスの C++での使用例.

```
#include  “Domain.hh”
#include  “MultiDataPeakFit.hh”

1: ElementContainerArray src;   // an element container array as source data
 2: vector< vector<Double> > funcMatrix;  // a set of fitting functions
 3: vector<Double> param;     // a list of paremeters
 4: vector<Double> lb;        // a list of the lower bounds for parameters
 5: vector<Double> ub;        // a list of the upper bounds for parameters
 6: ElementContainerArray result;   // an ElementContainerArray for results
 7: ElementContainerMatrix resultComponents;
 8: ParamSet fittedParamSet;  // data container for the fitted parameters
 9: ParamSet latestStat;      // data  container for the latest status
10
11: // ドライバーの生成
12: MultiDataPeakFit op=MultiDataPeakFit(&src, new MultiDataLevmar);
13: op.setDefaultParam();  //パラメータにデフォルト値を設定
14:
15: // Levenberg-Marquit 法用パラメータの設定
14: op.setParam(LevmarConsts::CONSTRAIN,           LevmarCons0ts::BOX);
15: op.setParam(LevmarConsts::USE_NUMERICAL_DIFF, true);
16: op.setParam(LevmarConsts::DIFF_METHOD,        LevmarConsts::FORWARD);
17: op.setParam(LevmarConsts::USE_DATA_WEIGHTS,   true);
18:
19: op.setParam(LevmarConsts::FUNCTIONS,          funcMatrix);
20: op.setParam(LevmarConsts::PARAMETER_VALUES,   param);
21: op.setParam(LevmarConsts::LOWER_BOUNDS,       lb);
22: op.setParam(LevmarConsts::UPPER_BOUNDS,       ub);
23:
24: if ( op.checkParam() ) {  // パラメータの整合性チェック
25:     op.fit();             // フィッティングの開始
26:     while (op.isFitting()) {
27:         sleep(1)          // wait 処理
28:         latestStat=op.getLatestConvergenceStat(); // 最新状態の取得
29:     }
30:     op.eval();    // フィッティング関数の値の計算
31:     result=op.getResults();  // フィッティング関数の値の取得
32:     resultCompoments=op.resultCompoments();
33:     fittedParamSet=op.FittedParam();  // フィッティング後のパラメータ値の取
得
```

リスト 2.6　MultiDatalevmar クラスの Python での使用例.

```
 1: import Manyo
 2:
 3: io=Manyo.NeXusFileIO()
 4: src = io.ReadElementContainerArray(options.inputFile) # ElementContainerArray
 5:
 6: exprList=[]                          #  フィッティング関数
 7: exprList.append( “g g g c” )
 8: …  同様に関数式を文字列で指定  …
 9: cppToPython=Manyo.CppToPython()
10: vecString = cppToPython.ListToStringVector(exprList)
11:
12: param=Manyo.VectorTool.MakeVectorDouble()  # フィッティングパラメータ
13: param.push_back(10000.0)
14: … 同様にパラメータの値を追加 …
15:
16: op = Adv.MultiDataPeakFit(src, Adv.MULTIDATA_LEVMAR) # ドライバーの生成
17: op.setDefaultParam()                  # パラメータにデフォルト値を設定
18:
19: # パラメータの設定
20: op.setParam(Adv.LevmarConsts.CONSTRAIN,           Adv.LevmarConsts.NO_CONSTRAIN)
21: op.setParam(Adv.LevmarConsts.USE_NUMERICAL_DIFF, Adv.LevmarConsts.DEFAULT_USE_NUMERICAL_DIFF)
22: op.setParam(Adv.LevmarConsts.DIFF_METHOD,        Adv.LevmarConsts.DEFAULT_DIFF_METHOD)
23: op.setParam(Adv.LevmarConsts.USE_DATA_WEIGHTS,   Adv.LevmarConsts.DEFAULT_USE_DATA_WEIGHTS)
24:
25: op.setParam(Adv.LevmarConsts.FUNCTIONS,         vecString) # フィッティング関数の設定
26: op.setParam(Adv.LevmarConsts.PARAMETER_VALUES, param)      # フィッティングパラメータの設定
27:
28: if op.checkParam():          # パラメータの整合性チェック
29:     op.fit()                 #  フィッティングの開始
30:     while op.isFitting():
31:         p=op.getLatestConvergenceStat()     # 最新の収束状態の取得
32:         waitTime=p.getInt4(Adv.LevmarConsts.ITERATION_TIME)/1000000.0
33:         tmpList=select.select([sys.stdin], [], [], waitTime)
34:         if not tmpList==([], [], []):
35:             c=raw_input()
36:             if c=="q":
37:                 op.stopFit() # 中断指令
38:                 break
39:     while op.isFitting(): # 実際に中断するまで待つ
40:         pass
41:     op.eval()     # フィッティング関数の値の計算
42:
43:     result=op.getResult()                          # フィッティング関数の取得
44:     resultComponents=op.getResultComponents()      #
45:     fittedParamSet=op.getFittedParam()             # フィッティング後のパラメータ値の取得
```

select を使った wait と中断処理

3. 計算アルゴリズムクラス

3.1. クラス BSpline

3.1.1. 概要

B-Splineによる関数フィッティングと与えられた座標点での関数値および2階までの微係数を求めることができる．実装のためにGNU Scientific Library[1]を使用している．

3.1.2. 詳細

3.1.2.1. B-Spline

B スプラインで平滑化を行うためにはまず、横軸(x 軸やt 軸など)を複数の小区間に分割する。各小区間の両端を**区切り点**（**break points**）と呼ぶ．図 3.1の$\{b_{\mathrm{i}}\,;0\le \mathrm{i}<N,N\ge 2\}$が区切り点である．

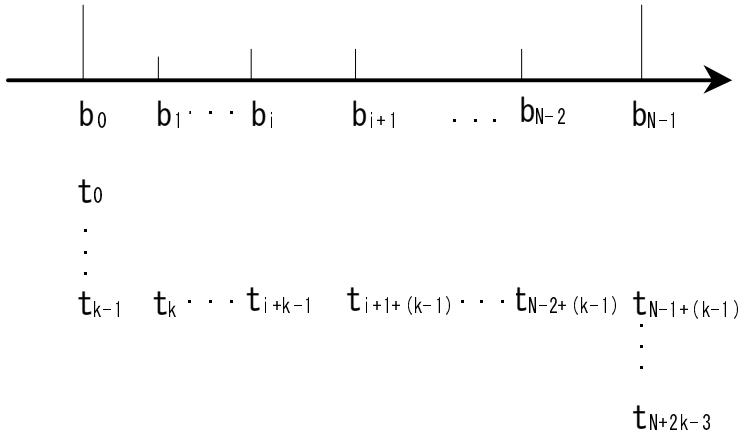


図 3.1 区切り点{bi；0≦i<N}と節点{ti；0≦i<N+2k-2}．k(≧1)は B-Spline 関数の次数である．

N個の区切り点から、**節点** (**knot**) $\{\mathrm{t_i}\,;0\le \mathrm{i}<N+2k-2,N\ge 2,k\ge 1\}$を次のように決める．

$$t_{\mathrm{i}}=\begin{cases} b_0 & 0\le \mathrm{i}<k \\ b_{\mathrm{i}-(\mathrm{k}-1)} & \mathrm{k}\le \mathrm{i}<N+k-2 \\ b_{\mathrm{N}-1} & \mathrm{N}+\mathrm{k}-2\le \mathrm{i}<N+2k-2 \end{cases}$$

ここで, kは次に説明するB-Spline関数の次数である．図 3.1に区切り点と等しい値の節点が各区切り点の下に描かれている．k≧2のとき始めと終わりのk個の節点の値はわざと重複させている．

節点を昇順に並べたベクトル$\{\mathrm{t_i}\,;0\le \mathrm{i}<N+2k-2,\ \mathrm{N}\ge 2,\ \mathrm{k}\ge 1\}$が与えられた時、$k$ 次のBスプライン基底関数は以下の漸化式で定義される．

$$B_{i,\ 1}(x) = \begin{cases} 1 & (t_i \le x < t_{i+1}) \\ 0 & \text{otherwize} \end{cases}$$

$$B_{i,\ k}(x) = \frac{x - t_i}{t_{i+k-1} - t_i} B_{i,\ k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1,\ k-1}(x) \quad (k \ge 2,\ 0 \le i < N + k - 2)$$

次数kは１から始まっていることに注意する．B-Spline基底関数$B_{i,k}(x)$の個数はN+k-2個である．

区切り点で定義された区間内にある誤差を含む測定点$\{(x_j, y_j \pm e_j)\,; 0 \le j < M\}$をB-Sspline基底関数の線型結合で近似または平滑化することを考える．

$$f(x) = \sum_{i=0}^{N+2k-3} C_i B_{i,\ k}(x)$$

とおき，統計量

$$\kappa^2 = \sum_{j=0}^{M-1} \left[\frac{y_j - f(x_j)}{e_j} \right]^2$$

が最小になるように係数$\{C_i\,; 0 \le i < N + k - 2,\ N \ge 2,\ k \ge 1\}$を決定する．この問題は線型最小二乗法で解くことができる．

係数を決定する連立一次方程式を解くためにしばしば正規方程式を解く方法が使われるが，本クラスでは特異値分解法で解く．ここで現れる正規方程式の係数行列は悪条件になりやすく，解の精度が悪化しやすいためである．

リスト 3.1　C++での BSpline クラスの使用例.

```
 1: #include  “ElementContainer.hh”
 2: #include  “Domain.hh”
 3: #include  “ParamSet.hh”
 4: #include  “BSpline.hh”
 5:
 6: ElementContainer  src;
 7: Domain domain=*(new Domain(src, xmin, xmax));
 8: ParamSet paramSet;
 9:
10: BSpline spline = *(new BSpline());        // B-Spline
11:
12: paramSet=spline.setDefaultParam(src);    // B-Spline 用デフォルトパラメータの生成
13: paramSet.replace(BSpline::USE_UNIFORM_BREAK_POINTS, true); // パラメータ値の再設定
14: paramSet.replace(BSpline::NUMBER_OF_BREAK_PONTS, 16);      // パラメータ値の再設定
15: paramSet.replace(BSpline::ORDER, 3);                       // パラメータ値の再設定
16: domain.setBinBound(A, B);                                  // 計算領域の設定
17:
18: if (spline.checkParam(src, domain, paramSet)) {   // パラメータの整合性チェック
19:     spline.toInnderForm(src, domain, paramSet);   // パラメータを内部形式へ変換
20:     spline.fit();                                 // フィッティングパラメータを算出
21:     spline.eval();                                // B-Spline 関数の値を計算
22:     spline.toElementContainer(src, result);       // 結果の取得
23: }
```

リスト 3.2　Python での BSpline クラスの使用例.

```
 1: src    = Manyo.ElementContainer()
 2: domain = Manyo.Domain(src, xmin, xmax)
 3:
 4: spline = Manyo.BSpline()        // B-Spline
 5:
 6: paramSet=spline.setDefaultParam(src)    // B-Spline 用デフォルトパラメータの生成
 7: paramSet.replace(Manyo.BSpline.USE_UNIFORM_BREAK_POINTS, true) // パラメータ値の再設定
 8: paramSet.replace(Manyo.BSpline.NUMBER_OF_BREAK_PONTS, 16)      // パラメータ値の再設定
 9: paramSet.replace(Manyo.BSpline.ORDER, 3)                       // パラメータ値の再設定
10: domain.setBinBound(A, B)                                       // 計算領域の設定
11:
12: if (spline.checkParam(src, domain, paramSet)):   // パラメータの整合性チェック
13:     spline.toInnderForm(src, domain, paramSet)   // パラメータを内部形式へ変換
14:     spline.fit()                                 // フィッティングパラメータを算出
15:     spline.eval()                                // B-Spline 関数の値を計算
16:     spline.toElementContainer(src, result)       // 結果の取得
17: }
```

表 3.1 BSpline クラスで指定可能なパラメータのキーとパラメータの属性.

キー		属性			説明
記号	値	型	標準値	必須	
BSpline::AUTOMATIC_KNOTS	"automatic knots"	Bool	false	false	節点の自動設定
BSpline::USE_UNIFORM_BREAK_POINTS	"use uniform break points"	Bool	true	false	等間隔の区切り点を使用
BSpline::NUMBER_OF_BREAK_POINTS	"number of break points")	UInt4	10	false	区切り点の数
BSpline::ORDER	"order"	UInt4	3	false	スプライン多項式の次数．B-スプライン関数の次数 k=order+1
BSpline::BREAK_POINTS	"break points"	vector<Double>		false	ユーザが与える区切り点の座標リスト

3.2. クラス MovingAverage（移動平均）

3.2.1. 概要

与えられたデータに対する移動平均値および, 二階までの微係数に相当する差分商を計算できる.

3.2.2. 移動平均および平均化差分商の定義

3.2.2.1. 移動平均の定義

bin の総数が N であるがあるする．ヒストグラムのi $(0 \le \mathrm{i} < N)$番目の bin に対して移動平均の定義は一般に次のように定義する.

$$\overline{\mathrm{y_{i(W,U)}}} = \frac{1}{\mathrm{W}} \sum_{\mathrm{k=-L}}^{\mathrm{U}} \mathrm{y_{i+k}}$$

$$\mathrm{L = W - U - 1}, \quad \mathrm{W \ge 1}, \quad \mathrm{0 \le U \le W - 1},$$

和をとる範囲 –L≦k≦U (U, L≧0) をここでは**窓**と呼ぶことにする. W=U+L+1 は平均をとるデータ点の数である．これを**窓の幅**とよぶ.

また, L と U をそれぞれ**窓の下端**および**上端**と呼ぶことにする．当然のことながら, W>1 である.

移動平均による平滑化を行うということは，隣接する bin のノイズの和をとることでそれらが互いに打ち消しあうことを期待するためである．窓の幅 W をどの程度すればよいかは，データに（正確にはデータに含まれるノイズの性質に）依存する.

W>1 のとき，この定義のままではヒストグラムの端の領域で窓の幅を W にすることができなくなる．その場合には，bin の幅を適宜減らして平均をとることにする．すると移動平均の定義は以下のよう書ける.

$$\overline{\mathrm{y_{i(W,U)}}} = \begin{cases} \dfrac{1}{\mathrm{U+1+i}} \displaystyle\sum_{\mathrm{k=-i}}^{\mathrm{U}} \mathrm{y_{i+k}} & 0 \le \mathrm{i} < L \\ \dfrac{1}{\mathrm{W}} \displaystyle\sum_{\mathrm{k=-L}}^{\mathrm{U}} \mathrm{y_{i+k}} & \mathrm{L \le i} < N - U \\ \dfrac{1}{\mathrm{N-i+L}} \displaystyle\sum_{\mathrm{k=-L}}^{\mathrm{N-1-i}} \mathrm{y_{i+k}} & \mathrm{N - U} \le i < N \end{cases}$$

ここで N は bin の総数（ $\{\mathrm{y_i}\,;\, 0 \le \mathrm{i} < N\}$の数）である．ヒストグラムの端の領域で，窓の幅，窓の上端および下端が bin の番号に依存する．端に行くほど，窓の幅が W より減るため，ノイズの影響が大きくなる.

3.2.2.2. 平均化差分商の計算

$\{\overline{y_i(W,U)}\,;\,0 \le i < N\}$ の増減を調べるために差分商の移動平均を計算する．これは関数の増減を調べるために各点での微係数の値を計算することに相当する．
一階の差分商を次式で定義する．

$$\frac{\Delta\overline{y_i}}{\Delta x_i} = \begin{cases} \dfrac{\overline{y_{i+1}} - \overline{y_i}}{x_{i+1} - x_i} & i = 0 \\ \dfrac{\overline{y_{i+1}} - \overline{y_{i-1}}}{x_{i+1} - x_{i-1}} & 1 \le i \le N-2 \\ \dfrac{\overline{y_i} - \overline{y_{i-1}}}{x_i - x_{i-1}} & i = N-1 \end{cases}$$

ここで$\{x_i\,;\,0 \le i < N\}$は bin の区間の代表値（通常は各 bin の両端の中点）である．両端の bin で前進差分と後退差分をとり，その他の bin では中心差分をとっている．関数の数値微分をとる方法とよく似ている．この差分商の移動平均を先に定義した方法で計算したものを，**一階の平均化差分商**と呼ぶことにする．窓の幅 W および窓の上端 U は，ヒストグラムの値を平滑化したときの値をそのまま使用する．

$$\frac{\overline{\Delta\overline{y_i}}}{\Delta x_i} = \begin{cases} \dfrac{1}{U+1+i}\displaystyle\sum_{k=-i}^{U} \frac{\Delta\overline{y_{i+k}}}{\Delta x_{i+k}} & 0 \le i < L \\ \dfrac{1}{W}\displaystyle\sum_{k=-L}^{U} \frac{\Delta\overline{y_{i+k}}}{\Delta x_{i+k}} & L \le i < N-U \\ \dfrac{1}{N-i+L}\displaystyle\sum_{k=-i}^{N-1-i} \frac{\Delta\overline{y_{i+k}}}{\Delta x_{i+k}} & N-U \le i < N \end{cases}$$

同様に 2 **階の平均化差分商**を計算する．まず一階の平均化差分商の差分商を計算する．

$$\frac{\Delta^2\overline{y_i}}{\Delta^2 x_i} = \begin{cases} \dfrac{\dfrac{\overline{\Delta\overline{y_{i+1}}}}{\Delta x_{i+1}} - \dfrac{\overline{\Delta\overline{y_i}}}{\Delta x_i}}{x_{i+1} - x_i} & i = 0 \\ \dfrac{\dfrac{\overline{\Delta\overline{y_{i+1}}}}{\Delta x_{i+1}} - \dfrac{\overline{\Delta\overline{y_{i-1}}}}{\Delta x_{i-1}}}{x_{i+1} - x_{i-1}} & 1 \le i \le N-2 \\ \dfrac{\dfrac{\overline{\Delta\overline{y_i}}}{\Delta x_i} - \dfrac{\overline{\Delta\overline{y_{i-1}}}}{\Delta x_{i-1}}}{x_i - x_{i-1}} & i = N-1 \end{cases}$$

次にこの差分商の移動平均を計算する．

$$\frac{\overline{\Delta^2 y_i}}{\Delta^2 x_i} = \begin{cases} \dfrac{1}{U+1+i}\displaystyle\sum_{k=-i}^{U} \dfrac{\Delta^2 \overline{y_{i+k}}}{\Delta^2 x_{i+k}} & 0 \le i < L \\ \dfrac{1}{W}\displaystyle\sum_{k=-L}^{U} \dfrac{\Delta^2 \overline{y_{i+k}}}{\Delta^2 x_{i+k}} & L \le i < N-U \\ \dfrac{1}{N-i-L}\displaystyle\sum_{k=-i}^{N-1-i} \dfrac{\Delta^2 \overline{y_{i+k}}}{\Delta^2 x_{i+k}} & N-U \le i < N \end{cases}$$

上のような方法で平均化差分商を計算するとき，窓の幅が十分広ければ，ノイズを大幅に減らすことができる．したがって関数の増減を微係数の正負で判定するのと同様に，差分商の平均の正負でノイズのあるデータの増減を調べることができる．

リスト 3.3　C++での MovingAverage クラスの使用例.

```
 1: #include  “ElementContainer.hh”
 2: #include  “Domain.hh”
 3: #include  “ParamSet.hh”
 4: #include  “MovingAverage.hh”
 5:
 6: ElementContainer  src;
 7: Domain domain=*(new Domain(src, xmin, xmax));
 8: ParamSet paramSet;
 9:
10: BSpline ma = *(new MovingAverage());        // B-Spline
11:
12: paramSet=ma.setDefaultParam(src);   // B-Spline 用デフォルトパラメータの生成
13: paramSet.replace(MovingAverage::WINDOW_WIDTH, 111);      // パラメータ値の再設定
14: paramSet.replace(MovingAverage::WINDOW_UPPER, 55);       // パラメータ値の再設定
15:
16: domain.setBinBound(A, B);                                // 計算領域の設定
17:
18: if (ma.checkParam(src, domain, paramSet)) {   // パラメータの整合性チェック
19:     ma.toInnderForm(src, domain, paramSet);   // パラメータを内部形式へ変換
20:     ma.fit();                                 // フィッティングパラメータを算出
21:     ma.eval();                                // B-Spline 関数の値を計算
22:     ma.toElementContainer(src, result);       // 結果の取得
23: }
```

リスト 3.4　Python での MobingAverage の使用例.

```
 1: src    = Manyo.ElementContainer()
 2: domain = Manyo.Domain(src, xmin, xmax)
 3:
 4: ma = Manyo.MovingAverage()        // B-Spline
 5:
 6: paramSet=ma.setDefaultParam(src)   // B-Spline 用デフォルトパラメータの生成
 7: paramSet.replace(Manyo.MovingAverage.WINDOW_WIDTH, 111) // パラメータ値の再設定
 8: paramSet.replace(Manyo.MovingAverage.WINDOW_WIDTH, 55)  // パラメータ値の再設定
 9:
10: domain.setBinBound(A, B)                                  // 計算領域の設定
11:
12: if (ma.checkParam(src, domain, paramSet)):   // パラメータの整合性チェック
13:     ma.toInnderForm(src, domain, paramSet)   // パラメータを内部形式へ変換
14:     ma.fit()                                 // フィッティングパラメータを算出
15:     ma.eval()                                // B-Spline 関数の値を計算
16:     ma.toElementContainer(src, result)       // 結果の取得
17: }
```

表 3.2　MovingAverage で指定可能なパラメータのキーとパラメータの属性

キー		属性			説明
記号	値	型	標準値	必須	
MovingAverage::USE_STRICT_DEFINITION	"use strict definition"	Bool	false	false	false のとき，ヒストグラムの端で窓の幅が指定値の通りにとれないとき，幅を減らして平均をとる．true のとき，幅が指定通りにとれない bin の平均値の計算を行わない．
MovingAverage::WINDOW_WIDTH	"window width"	UInt4		false	窓の幅（平均をとる連続した bin の数）．1 より大きい奇数にする．
MovingAverage::WINDOW_UPPER	"window upper"	UInt4		false	窓の中で平均値を置く位置を窓の上端からの位置で指定する．この値を変えると，平均値が平行移動される．通常は MovingAverage::WINDOW_WIDTH の 1/2 に設定する．

3.3. クラス Levmar

3.3.1. 概要

クラス Levmar は，Levenberg – Marquardth 法による非線形最小化アルゴリズム[3]を実装した levmar- ver. 2.5[2]のラッパークラスである．levmar ver.2.5 の全機能を利用するためには，Lapack またはその互換ライブラリが必要である．Levmar は，Levmar-2.5 をコンパイルする際に設定された Lapack を使用するか否かの設定を引き継ぐ．

3.3.1.1. 制約

levmar-2.5 は関数の非線形最小化を行う際に制約を課すことができる．箱型，線型等式，線型不等式およびこれらの任意の組み合わせた制約を課すことができる．組み合わせを考えると，課すことのできる制約は以下の 8 通りである．

- 無制約
- 箱型（Box Constrain 以下 BC）
- 線型等式（Linear Equations Constrain 以下 LEC）
- 線型不等式（Linear Inequalities Constrain 以下 LIC）
- 箱+線型等式（以下 BLEC）
- 箱+線型不等式(以下 BLIC)
- 線型等式+線型不等式（以下 LEIC）
- 箱+線型等式+線型不等式（以下 BLEIC）

箱型制約は，各パラメータ$\{p_{\mathrm{i}}\,;\,0 \le \mathrm{i} < m\}$の変化できる範囲を下限値と上限値で指定する．

$$l_{\mathrm{i}} \le p_{\mathrm{i}} \le \mathrm{u_i}\ \ (0 \le \mathrm{i} < m)$$

Levmar クラスには下限値を並べたベクトル $\boldsymbol{l}$ と上限値を並べたベクトル $\boldsymbol{u}$ を与える．あるパラメータの値を固定したい場合には，上限と下限の値をパラメータの値と一致させる．

$$l_{\mathrm{k}} = p_{\mathrm{k}} = \mathrm{u_k}$$

$\boldsymbol{l}$ を与えない場合は全てのパラメータについて下限の制約はないとみなされる．同様に $\boldsymbol{u}$ を与えない場合は全てのパラメータについて上限の制約はないとみなされる．

線型等式制約はパラメータの取りうる範囲を線型部分空間に限定するために与える．制約式はパラメータの線型結合がある値に等しいとする．

$$\sum_{\mathrm{k=0}}^{\mathrm{m-1}} A_{\mathrm{ik}} p_{\mathrm{k}} = b_{\mathrm{i}}\ \ (0 \le \mathrm{i} < \mathrm{N}_e)$$

ここで $\mathrm{N_e}$ は制約等式の個数である．線型結合係数でつくる行列を $\boldsymbol{A}$、定数項で作るベクトルを $\boldsymbol{b}$ とすると，Levmar クラスにはこれらをまとめた拡大係数行列$\boldsymbol{A'} = [\boldsymbol{A}, \boldsymbol{b}]$を与える．

線型不等式制約は，パラメータ $\boldsymbol{p}$ の範囲を単一の凸型の領域に限定するために課す．制約式はパラメータの線型結合がある定数値以上であるとみなされる．

$$\sum_{k=0}^{m-1} C_{ik} p_k \geq d_i \quad (0 \leq i < N_i)$$

ここで N_i は不等式の個数である．線型結合係数を成分とする行列 $\boldsymbol{C}$，定数項で作るベクトル $\boldsymbol{d}$ とすると，Levmar にはこれらをまとめた拡大係数行列 $\boldsymbol{C'} = [\boldsymbol{C}, \boldsymbol{d}]$ を与える．

ほとんどのユーザは無制約または箱型制約を課して Levmar クラスを使用するであろう．

3.3.1.2. フィッティング関数

フィッティングには関数の和を指定できる．使用できる関数の組み込みの関数は 5 章で説明する．

3.3.1.3. Jacobi 行列の評価

目的関数 $f(x_i ; \boldsymbol{p})$ のパラメータ $\boldsymbol{p}$ について最適化するために次のように定義される Jacobi 行列が必要である．

$$\boldsymbol{J}_{ij} = \frac{\partial f(x_i ; \boldsymbol{p})}{\partial p_j} \quad (0 \leq i < L \quad 0 \leq j < m)$$

Levmar クラスでは Jacobi 行列を計算する方針として，Levmar クラスに与えるパラメータによって，解析的な計算と数値微分から一方を選択することができる．数値微分の方法として前進差分または中心差分を選択できる．また数値微分の差分幅はパラメータで与えることができる．解析的に評価式を用いた方が正確ではあるが，評価式が複雑な場合には，数値微分の方が高速であることが多い．

3.3.1.4. 打ち切り判定

最適化は次の 7 条件のうちの一つが成立すると打ち切られる．

- 残差の L_2 ノルムが閾値 ε_r 以下あるとき

$$\|\boldsymbol{r}\|_2 = \sqrt{\sum_{i=0}^{N} \left[\frac{y_i - f(x_i; \boldsymbol{p})}{e_i}\right]^2} \leq \varepsilon_r$$

重み付き最小二乗法で関数のあてはめを行うとき，残差の L2 ノルムは，測定値 y とその誤差 e，フィッティング関数の値を使って上記のように定義されている．閾値 ε_r はユーザが設定できる．重み付きでない最小二乗法では，e=1 とする．

- フィッティング関数の勾配の L_∞ ノルムが閾値 ε_g 以下であるとき

$$\|\boldsymbol{g}\|_\infty = \left\|\boldsymbol{J}^{\mathrm{T}}\boldsymbol{r}\right\|_\infty = \max_{0\le i<m}\left\{\left|[\boldsymbol{J}^{\mathrm{T}}\boldsymbol{r}]_i\right|\right\} \le \varepsilon_g$$

勾配はフィッティング関数の Jacobi 行列の転置行列と残差ベクトルの積で定義される．L_∞ ノルムは，成分の絶対値の最大値で定義だれる．閾値 ε_g はユーザが指定できる．

- パラメータ $\boldsymbol{p}$ の変化の L_2 ノルムとパラメータ自身の L_2 ノルムが閾値 ε_p 以下であるとき

$$\frac{\|\Delta\boldsymbol{p}\|_2}{\|\boldsymbol{p}\|_2} \le \varepsilon_p$$

閾値 ε_p はユーザが設定できる．

- 繰り返し回数計算がユーザの設定した最大値に到達したとき

- singular matix が発生した，または，しそうなとき

この判断は次の条件でなされる．

$$\|\Delta\boldsymbol{p}\|_2 \ge (\|\boldsymbol{p}\|_2 + \varepsilon_g)/\varepsilon^2$$

$\varepsilon = 1.0\times10^{-12}$ である．この値は levmar ver2.5 の設定値でユーザは変更できない． ε_g はフィッティング関数の勾配の L_∞ ノルムの閾値である．

- もはや誤差の減少の余地がなくなったと判定されたとき

- フィッティング関数の計算中にエラー(NaN, ∞等)が発生したとき

・関数に与えるパラメータ（$\boldsymbol{x}, \boldsymbol{p}$）が関数の制約を満たさないとき

・フィッティング関数の値や Jacobi 行列の成分の値を計算した結果が，実数とみなされない結果 NaN（**N**ot **a N**umber），±∞となったとき

3.3.1.5. 中断対応

このクラス Levmar クラスは，ユーザによる中断を行うことができる．中断される，Levmar クラスにユーザが中断するまでに返された履歴以外の内部データは無効となる．

Levmar クラスは中断を受け付けるためにスレッド化されている．スレッド化するために pthread（POSIX thread）ライブラリ[4]を利用している．

3.3.2. Levmar で使用できるパラメータのリスト

Levmar クラスで入力として使用できるパラメータを表 3.3 と表 3.4 に示す．表 3.3 に Levmar

クラスの動作を決定するパラメータを列挙した．最適化の入力になるパラメータは表 3.4 で示す．

表 3.3　Levmar の動作を決定する入力パラメータのキーと属性.

キー		属性			説明
記号	値	型	標準値†	必須	
Levmar::CONSTRAIN	"constrain"	Constrain（列挙）	Levmar::BOX	false	制約の種類，　Levmar::NO_CONSTRAIN,（無制約）Levmar::BOX（箱型），Levmar::LEC（線型等式），Levmar;::LIC（線型不等式）, Levmar::BLEC（箱＋線型等式），Levmar::BLIC（箱＋線型不等式）, Levmar::LEIC（線型等式＋線型不等式）, Levmar::BLEIC（箱＋線型等式＋線型不等式）のうちの一つ. 赤字の制約を使用するためには Lapack が必要.
Levmar::USE_DATA_WEIGHTS	"use data weights"	Bool	true	false	重み付き最小二乗フィッティングを行う
Levmar::USE_NUMERICAL_DIFF	"use numerical diff"	Bool	true	false	数値微分で Jacobi 行列を計算するか否か
Levmar::DIFF_METHOD	"diff method"	DiffMethod（列挙）	Levmar::FORWARD	false	Levmar::USE_NUMERICAL_DIFF が ture のときに有効. 数値微分を行う方法. Levmar::FORWARD なら前進差分，Levmar::CENTRAL なら中心差分
Levmar::MAX_ITERATIONS	"max iterations"	UInt4	1000	fales	最大繰り返し回数
Levmar::OUTPUT_INTERVAL	"output interval"	UInt4	50	false	途中経過の出力間隔

表 3.4　Levmar の指定できる入力パラメータのキーとパラメータの属性

キー		属性			説明
記号	値	型	標準値†	必須	
Levmar::FUNCTIONS	"functions"	vector<FuncBase*>	-	true	フィッティングを行う関数，関数のクラスのポインターを入れた vector で渡す.
Levmar::INITIAL_PARAM_VALUES	“initial param values”	vector<Double>		ture	フィッティングを行うパラメータの初期値のリスト．FUNCTIONS で与えた関数が必要とする順でパラメータを与える.
Levmar::PARAMETER_VALUES	"parameter values"	vector<Double>	-	true	フィッティングを行うパラメータの値のリスト．FUNCTIONS で与えた関数が必要とする順でパラメータを与える．Levmar::INITIAL_PARAM_VALUES を使用する.
Levmar::REFERENCE_VALUES	"reference values"	vector<Double>		true	フィッティングの際に参照するデータの値
Levmar::LOWER_BOUNDS	"lower bounds"	vector<Double>		false	箱型制約で各パラメータの下限値を指定するリスト．指定がない場合は，下限に制限はない.
Levmar::UPPER_BOUNDS	"upper bounds"	vector<Double>		false	箱型制約で各パラメータの上限値を指定するリスト．指定がない場合は，上限に制限はない.
Levmar::BOX_WEIGHTS‡	"box weights"	vector<Double>	(__BC_WEIGHT__)†	false	LevmarCONSTRAIN=Levmar::BLEC のときのみ有効．箱型制約の重みを設定する．指定がない場合はデフォルト値を使用する. 使用するためには Lapack が必要.

Levmar:: EQUATIONS‡	"equations"	vector<vector<Double>>		false	LevmarCONSTRAIN=Levmar::LEC, Levmar::BLEC, Levmar::LEIC, Levmar::BLEIC のとき有効. 制約を線型等式の係数と定数項からなる拡大係数行列で指定するする. 使用するためには Lapack が必要.
Levmar::INEQUALITIES‡	"inequalities"	vector<vector<Double>>		false	LevmarCONSTRAIN=Levmar::LIC, Levmar::BLLC, Levmar::LEIC, Levmar::BLEIC のとき有効. 一連の線型不等式の係数と定数項からなる拡大係数行列. 使用するためには Lapack が必要.
Levmar::SCALING_FACTOR	"scaling factor"	Double	LM_INIT_MU†	false	スケール因子の初期値. 指定がなければデフォルト値を使用.
Levmar::RESIDU_ERR_THRESH	“residu err theash”	Double	LM_INIT_MU†	false	残差の L_2 ノルムによる打ち切りの下限値. 指定がなければデフォルト値を使用.
Levmar::GRADIENT_THRESH	“gradient thresh”	Double	LM_INIT_MU†	false	勾配ベクトルの L_∞ ノルムによる打ち切りの下限値. 指定がなければデフォルト値を使用.
Levmar::PARAM_DIFF_THRESH	“param diff. thresh”	Double	LM_INIT_MU†	false	パラメータ変化の L_2 ノルムによる打ち切りの下限値. 指定がなければデフォルト値を使用.
Levmar::TOLERANCE	"tolerance"	Double	LM_STOP_THRESH†	false	打ち切りの閾値. 指定がなければデフォルト値を使用. 廃止予定. 代りに Levmar::PARAM_EIFF_THRESH を使用.
Levmar::RELATIVE_TOLERANCE	"relative tolerance"	Double	LM_STOP_THRESH†	false	打ち切りの閾値. 指定がなければデフォルト値を使用. 廃止予定. 代りに Levmar::RESIDU_ERR_THRESH を使用.

Levmar::GRADIENT_TOLERANCE	"gradient tolerance")	Double	LM_STOP_THRESH†	false	打ち切りの閾値．指定がなければデフォルト値を使用．廃止予定．代りに Levmar::GRADIENT _THRESH を使用.
Levmar::DIFF_DELTA	"diff delta"	Double	LM_DIFF_DELTA†	false	Levmar::USE_NUMERICAL_DIFF が ture のとき有効．数値微分の差分幅．指定がなければデフォルト値を使用.

† levmar ver. 2.5 で定義済みの値である．

‡ Lapack がなければ使用できない．

表 3.5　計算途中および終了後の出力.

キー		属性			説明
記号	値	型	標準値	必須	
Levmar::ITRATION_COUNT	“iteration count”	UInt4		true	出力時点の総繰り返し回数
Levmar::TERMINATION_STAT	“termination stat”	Levmar::LevmarStat（列挙）		true	出力時点での収束状況．以下の九つの値のうちの一つ． Levmar::SMALL_GRADIENT, Levmar::SMALL_DP, Levmar::MAX_ITERATIONS, Levmar::SINGULAR_MATRIX, Levmar::NO_FURTHER_ERROR_REDUCTION, Levmar::SMALL_RESIDUAL_ERROR, Levmar::INVALID_FUNC, Levmar::CONTINUE, Levmar::FORCE_QUIT_BY_USER Levmar::CONTINUE のときは，繰り返し計算が継続される．
Lemvar::TERMINATION_REASON	“termination reason”	string		true	Levmar::TERMINATION_STAT に対応する文字列
Levmar::R_FACTOR	“R-factor””	Double		true	フィッティングされた関数と測定値との R-因子． Levmar::USE_DATA_WEIGHT が ture の場合，計測誤差による重みがつく
Levmar::RESIDUAL_ERROR_NORM	“residual error norm”	Double		ture	残差ベクトル $\boldsymbol{e}$ の L_2 ノルム
Levmar::GRADIENT_NORM	“gradient norm”	Double		ture	勾配ベクトル $\boldsymbol{g}=\boldsymbol{J}^T\boldsymbol{e}$ の L_∞ ノルム．J はフィッティング関数のヤコビ行列 $J_{ij}=f(x_i, \boldsymbol{p})$，$\boldsymbol{e}$ は残差ベクトル
Levmar::PARAMETER_DIFF_NORM	“parameter diff norm”	Double		true	パラメータの差分の L_2 ノルム

Levmar::INITIAL_PARAM_VALUES	“initial param values”	vector<Double>			フィッティングパラメータの初期値．入力と同じ値.
Levmar::PARAMETER_VALUES	“parameter values”	vector<Double>	-		パラメータの値のリスト．FUNCTIONS で与えた関数が必要とする順でパラメータが並んでいる.
Levmar::PARAMETER_ERRORS	“parameter errors”	vector<Double>		true	フィッティングパラメータの誤差の推定値
Levmar::COVAIANCE_MATRIC	“covarivance matrix”	vector< vector<Double> >			共分散行列
Levmar::FUNCTION_EVALATIONS	“function evaluations”	Int4			関数値の評価回数
Levmar::JACOBIAN_EVALUATIONS	“Jacobian evaluations”	Int4			Jacobian の評価回数
Levmar::LINEAR_SYSTEMS_SOLVED	“linear system solved”	Int4			線型システムが解けた回数
Levmar::ITERATION_TIME	“iteration time”	Int4			最適化計算にかかった時間．μ 秒単位

$$\mathrm{R} = \frac{\sum_{\mathrm{i}=0}^{\mathrm{L}-1} \frac{\left||y_{\mathrm{i}}| - |f(\mathrm{x}_{\mathrm{i}}; \boldsymbol{p})|\right|}{e_{\mathrm{i}}}}{\sum_{\mathrm{i}=0}^{\mathrm{L}-1} \frac{|y_{\mathrm{i}}|}{e_{\mathrm{i}}}}$$

3.4. MultiDataLevmar

3.4.1. 概要

複数のデータ列$\left\{\left\{\left(x_{\mathrm{n,i}},\ y_{\mathrm{n,i}} \pm \mathrm{e}_{\mathrm{n,i}}\right);\ 0 \le \mathrm{i} < \mathrm{I_n}\right\},\ \ 0 \le \mathrm{n} < N\right\}$ に対する最小二乗法を考える．(x, y)はそれぞれ変数とデータ(測定値)である．e はデータ y の誤差を表す．i は各データ系列内でデータを区別する番号である．あるデータ列 n のデータ数は$\mathrm{I_n}$である．

データ系列 n に対するフィッティング関数を$f_{\mathrm{n}}(x;\ \boldsymbol{p})$とおく．関数のパラメータ$\boldsymbol{p} = \{\mathrm{p_m}; 0 \le \mathrm{m} < M\}$は全てのフィッティング関数で共通であるとする．
最小二乗法で最小化する目的関数は次式で与えられる．

$$\mathrm{Z}(\boldsymbol{p}) = \sum_{\mathrm{n=0}}^{\mathrm{N-1}} \mathrm{c_n^2} \sum_{\mathrm{i=0}}^{\mathrm{I_n-1}} \mathrm{w_{n,i}^2}\left[y_{\mathrm{n,i}} - f_{\mathrm{n}}\left(x_{\mathrm{n,i}}; \boldsymbol{p}\right)\right]^2$$

MultiDataLevmar は Levenberg-Marquit 法で$\mathrm{Z}(\boldsymbol{p})$を最小化するパラメータ$\boldsymbol{p}$の値を探索する．実際の探索は levmar ライブラリ [2] [3]を使用している．

3.4.1.1. 最上二乗法の重み

3.4.1.1.1. 重み付き最小二乗法

$\mathbf{w_n} = \left\{\mathrm{w_{n,i}}; 0 \le \mathrm{i} < \mathrm{I_n}\right\}$の各成分はそれぞれデータ列 n の各データ$\left(\mathrm{x_{n,i}},\ \mathrm{y_{n,i}}\right)$の重みであり，データの誤差を使って次のように定義する．

$$\mathrm{w_{n,i}^2} = \frac{1}{(\mathrm{I_n} - \mathrm{M})}\left(\frac{1}{\mathrm{e_{n,i}}}\right)^2$$

$\mathrm{I_n} - \mathrm{M}$はこのデータ系列の自由度である．このとき，

$$\sum_{\mathrm{i=0}}^{\mathrm{I_n-1}} \mathrm{w_{n,i}^2}\left[y_{\mathrm{n,i}} - f_{\mathrm{n}}\left(x_{\mathrm{n,i}}; \boldsymbol{p}\right)\right]^2 = \frac{1}{(\mathrm{I_n} - \mathrm{M})} \sum_{\mathrm{i=0}}^{\mathrm{I_n-1}} \left[\frac{y_{\mathrm{n,i}} - f_{\mathrm{n}}\left(x_{\mathrm{n,i}}; \boldsymbol{p}\right)}{\mathrm{e_{n,i}}}\right]^2$$

和記号の部分は，$\mathrm{y_{n,i}}$の誤差が Gauss 分布となる理想的場合であれば，自由度$\mathrm{I_n} - \mathrm{M}$の$\kappa^2$分布に従う．

$\boldsymbol{c} = \{\mathrm{c_n}; 0 \le \mathrm{n} < N\}$は

$$\mathrm{c_n^2} = \frac{1}{N}\ \ (0 \le \mathrm{n} < N)$$

とする．理想的な場合には，全ての関数がうまくフィットされたとき，目的関数 Z の値は 1 程度となることが期待される．

$$\mathrm{Z}(\boldsymbol{p}) = \sum_{\mathrm{n}=0}^{\mathrm{N}-1} \frac{1}{N} \left\{ \frac{1}{(\mathrm{I_n} - \mathrm{M})} \sum_{\mathrm{i}=0}^{\mathrm{I_n}-1} \left[\frac{y_{\mathrm{n,i}} - f_{\mathrm{n}}(x_{\mathrm{n,i}}; \boldsymbol{p})}{\mathrm{e_{n,i}}} \right]^2 \right\} \approx 1$$

3.4.1.1.2. 重みなし最小二乗法

$\mathbf{w_n}$を誤差によらずに一定にする場合には,

$$\mathrm{w}_{\mathrm{n,i}}^2 = \frac{1}{\sum_{\mathrm{i}=0}^{\mathrm{I_n}-1} \mathrm{e}_{\mathrm{n,i}}^2}$$

とする．これはデータ系列間で残差の二乗和のバランスをとるためである．

$$\sum_{\mathrm{i}=0}^{\mathrm{I_n}-1} \mathrm{w}_{\mathrm{n,i}}^2 \left[\mathrm{y_{n,i}} - f_{\mathrm{n}}(x_{\mathrm{n,i}}; \boldsymbol{p}) \right]^2 = \frac{1}{\sum_{\mathrm{i}=0}^{\mathrm{I_n}-1} \mathrm{e}_{\mathrm{n,i}}^2} \sum_{\mathrm{i}=0}^{\mathrm{I_n}-1} \left[\mathrm{y_{n,i}} - f_{\mathrm{n}}(x_{\mathrm{n,i}}; \boldsymbol{p}) \right]^2$$

$\boldsymbol{c} = \{\mathrm{c_n}; 0 \leq \mathrm{n} < N\}$は，重み付き最小二乗法と同様に，n に関わりなく

$$\mathrm{c}_{\mathrm{n}}^2 = \frac{1}{N} \ (0 \leq \mathrm{n} < N)$$

とする．このとき Z は以下のとおりである．

$$\mathrm{Z}(\boldsymbol{p}) = \sum_{\mathrm{n}=0}^{\mathrm{N}-1} \frac{1}{N} \left\{ \frac{1}{\sum_{\mathrm{i}=0}^{\mathrm{I_n}-1} \mathrm{e}_{\mathrm{n,i}}^2} \sum_{\mathrm{i}=0}^{\mathrm{I_n}-1} \left[\mathrm{y_{n,i}} - f_{\mathrm{n}}(x_{\mathrm{n,i}}; \boldsymbol{p}) \right]^2 \right\}$$

3.4.1.2. 制約

クラス Levmar と同様に，MultiDataLevmar はフィッティングパラメータに制約条件を課すことができる．かけられる制約は Levmar と全く同じである．詳細は 3.3.1.1 を参照する．

3.4.1.3. フィッティング関数

クラス Levmar は異なりデータ系列ごとにフィッティング関数を指定できる．ただし，指定されたフィッティング関数は一組のパラメータを共有する．

3.4.1.4. Jacobi 行列の評価

クラス Levmar と同様に，Jacobi 行列を解析的にも求めことも差分商で近似的に求める

こともできる．詳細は 3.3.1.3 を参照する．

3.4.1.5. 打ち切り判定

収束計算が打ち切られる条件は，以下の通りである．

●残差の二乗和

残差のノルムが指定された値ε_rより小さければ，繰り返し計算は打ち切られる．

$$\|\boldsymbol{r}\|_2 = \sqrt{Z(\boldsymbol{p})} \leq \varepsilon_r$$

ただし残差ベクトル $\boldsymbol{r}$は以下の通りである．

$$\boldsymbol{r} = \begin{pmatrix} \vdots \\ c_{n-1}w_{n-1,I_{n-1}-1}\left[y_{n-1,I_{n-1}-1} - f_{n-1}\left(x_{n-1,I_{n-1}-1};\boldsymbol{p}\right)\right] \\ c_n w_{n,0}\left[y_{n,0} - f_n\left(x_{n,0};\boldsymbol{p}\right)\right] \\ \vdots \\ c_n w_{n,i}\left[y_{n,i} - f_n\left(x_{n,i};\boldsymbol{p}\right)\right] \\ \vdots \\ c_n w_{n,I_n-1}\left[y_{n,I_n-1} - f_n\left(x_{n,I_n-1};\boldsymbol{p}\right)\right] \\ c_{n+1}w_{n+1,0}\left[y_{n+1,0} - f_{n+1}\left(x_{n+1,0};\boldsymbol{p}\right)\right] \\ \vdots \end{pmatrix}$$

係数$c_n w_{n,i}$は以下の通りである．

$$c_n w_{n,i} = \begin{cases} \dfrac{1}{\sqrt{N(I_n - M)}e_{n,i}} & \text{重み付き最小二乗法のとき} \\ \dfrac{1}{\sqrt{N\sum_{i=0}^{I_n-1} e_{n,i}^2}} & \text{重みなし最小二乗法のとき} \end{cases}$$

データ列 n には，先頭から$\sum_{p=0}^{n-1} I_p \leq k < \sum_{p=0}^{n} I_p$の成分（赤字の部分）が対応する．残差ベクトルのノルム 2 乗は$Z(\boldsymbol{p})$に一致する．

●勾配の∞ノルム

Z(p)の勾配の∞ノルムが指定された値ε_gよりも小さければ，繰り返し計算は打ち切られる．

$$\|\nabla Z(\boldsymbol{p})\|_\infty < \varepsilon_g$$

ここで，

$$\nabla Z(\boldsymbol{p}) = \begin{pmatrix} \vdots \\ \dfrac{\partial}{\partial p_m} Z(\boldsymbol{p}) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ -2\displaystyle\sum_{n=0}^{N-1}\sum_{i=0}^{I_n-1}\left\{c_n w_{n,i}\frac{\partial}{\partial p_m} f_n\left(x_{n,i};\boldsymbol{p}\right)\right\}\left\{c_n w_{n,i}\left[y_{n,i} - f_n\left(x_{n,i};\boldsymbol{p}\right)\right]\right\} \\ \vdots \end{pmatrix}$$

$$= -2\boldsymbol{J}^{\boldsymbol{T}}\boldsymbol{r}$$

ただし，$\boldsymbol{J}$は Jacobi 行列で以下の通りである．

$$
\boldsymbol{J} = \begin{pmatrix}
\vdots & & \vdots & & \vdots \\
c_{n-1} w_{n-1,I_{n-1}-1} \dfrac{\partial}{\partial p_0} f_{n-1}\left(x_{n-1,I_{n-1}-1}; \boldsymbol{p}\right) & \cdots & c_{n-1} w_{n-1,I_{n-1}-1} \dfrac{\partial}{\partial p_m} f_{n-1}\left(x_{n-1,I_{n-1}-1}; \boldsymbol{p}\right) & \cdots & c_{n-1} w_{n-1,I_{n-1}-1} \dfrac{\partial}{\partial p_M} f_{n-1}\left(x_{n-1,I_{n-1}-1}; \boldsymbol{p}\right) \\
c_n w_{n,0} \dfrac{\partial}{\partial p_0} f_n\left(x_{n,0}; \boldsymbol{p}\right) & \cdots & c_n w_{n,0} \dfrac{\partial}{\partial p_m} f_n\left(x_{n,0}; \boldsymbol{p}\right) & \cdots & c_n w_{n,0} \dfrac{\partial}{\partial p_M} f_n\left(x_{n,0}; \boldsymbol{p}\right) \\
\vdots & & \vdots & & \vdots \\
c_n w_{n,i} \dfrac{\partial}{\partial p_0} f_n\left(x_{n,i}; \boldsymbol{p}\right) & \cdots & c_n w_{n,i} \dfrac{\partial}{\partial p_m} f_n\left(x_{n,i}; \boldsymbol{p}\right) & \cdots & c_n w_{n,i} \dfrac{\partial}{\partial p_M} f_n\left(x_{n,i}; \boldsymbol{p}\right) \\
\vdots & & \vdots & & \vdots \\
c_n w_{n,I_n-1} \dfrac{\partial}{\partial p_0} f_n\left(x_{n,I_n-1}; \boldsymbol{p}\right) & \cdots & c_n w_{n,I_n-1} \dfrac{\partial}{\partial p_m} f_n\left(x_{n,I_n-1}; \boldsymbol{p}\right) & \cdots & c_n w_{n,I_n-1} \dfrac{\partial}{\partial p_M} f_n\left(x_{n,I_n-1}; \boldsymbol{p}\right) \\
c_{n+1} w_{n+1,0} \dfrac{\partial}{\partial p_0} f_{n+1}\left(x_{n+1,0}; \boldsymbol{p}\right) & \cdots & c_{n+1} w_{n+1,0} \dfrac{\partial}{\partial p_m} f_{n+1}\left(x_{n+1,0}; \boldsymbol{p}\right) & \cdots & c_{n+1} w_{n+1,0} \dfrac{\partial}{\partial p_M} f_{n+1}\left(x_{n+1,0}; \boldsymbol{p}\right) \\
\vdots & & \vdots & & \vdots
\end{pmatrix}
$$

$\boldsymbol{J}$は$\sum_{p=0}^{N-1} I_p \times M$型の行列である．データ列 n には，先頭行から$\sum_{p=0}^{n-1} I_p \leq k < \sum_{p=0}^{n} I_p$の行が対応する．係数$c_n w_{n,i}$は残差の場合と同じである．

●パラメータ変化

Levmar クラスと同じ判定基準を用いている．の 3.3.1.4 のパラメータ変化の項を参照する．

●繰り返し回数の上限

収束計算の繰り返し回数が設定された値に達すると，打ち切られる．

●Singular matrix の発生

Levmar クラスと同じ判定基準を用いている．3.3.1.4 の Singular mateix の発生の項を参照する．

●誤差減少の余地がなくなったとき

Levmar クラスと同じ判定基準を用いている．3.3.1.4 の差減少の余地がなくなったときの項を参照する．

●フィッティング関数のエラー

Levmar クラスと同様である．3.3.1.4 のフィッティング関数のエラーの項を参照する．

3.4.1.6. 中断対応

クラス Levmar と同様に中断ができる．

3.4.2. MultiDataLevmar のパラメータ

MultiDataLevmar クラスの使用法は Lemvar クラスとよく似ているが. 複数系列のデータを扱うために異なる点がある．クラス Levmar と異なる点だけを記述する．

3.4.2.1. 入力

Levmar クラスでは，参照データを ElemnetConainer に入れて渡していたが，MultiDatalevmar クラスでは，複数系列のデータを参照データとして渡すために ElementContainerArray を使用する．

Levmar クラスでは, データの範囲を指定するために Domain でデータの範囲を指定した．MultiDataLevmar では，複数のデータ系列の範囲を制御するために，複数の Domain クラ

スのインスタンスが必要である．vector<Domain>で MultiDataLevmar で渡す．ElementContainerArray の i 番目のデータは，vector<Domain> の i 番目の要素と対応する．

Levmar クラスで扱うフィッティング関数は一つだけであったが，MultiDataLevmar では ElementContainerArray 内のデータ系列数と同じ数のフィッティング関数を指定なければならない．vector< vector<FuncBase* > で複数のフィッティング関数を渡すことができる．このクラスのインスタンスを ParamSet のインスタンスに入れて MultiDataLevmar に渡す．ElementContainerArray の i 番目のデータは，vector< vector<FuncBase*> > の i 番目の要素と対応する．

3.4.2.2. 出力

MuliDataLevmar クラスは，各系列のデータに対応する関数の値を返す．その際，全てのフィッティング関数の値を一度に返すために ElementoContainerArray を使用する．

表 3.6　Levmar の指定できる入力パラメータのキーとパラメータの属性（Levmar と定義が変更された部分のみ掲載）.

キー		属性			説明
記号	値	型	標準値†	必須	
MultiDataLevmar::FUNCTIONS	"functions"	vector< vector<FuncBase*> >	-	true	フィッティングを行う関数，関数のクラスのポインターを入れた vector を要素とする vector で渡す.

表 3.7　計算途中および終了後の出力（Levmar と定義が変更された部分のみ掲載）.

キー		属性			説明
記号	値	型	標準値	必須	
MultiDataLevmar::R_FACTOR	“R-factor””	Double		true	フィッティングされた関数と測定値との R-因子. Levmar::USE_DATA_WEIGHT が ture の場合，計測誤差による重みがつく.また，Levmar クラスの返す値にに対して複数のデータ系列の値を含むように拡張されている.
MultiDataLevmar::RESIDUAL_ERROR_NORM	“residual error norm”	Double		ture	残差ベクトル $\boldsymbol{e}$ の L_2 ノルム．Levmar クラスの返す値にに対して複数のデータ系列の値を含むように拡張されている.
MultiDataLevmar::GRADIENT_NORM	“gradient norm”	Double		ture	勾配ベクトル $\boldsymbol{g}=\boldsymbol{J}^{\mathrm{T}}\boldsymbol{e}$ の L_∞ ノルム. J はフィッティング関数のヤコビ行列 $J_{ij}=\partial f(x_i, \boldsymbol{p})$，$\boldsymbol{e}$ は残差ベクトル．Levmar クラスの返す値にに対して複数のデータ系列の値を含むように拡張されている.

C++用コード

```
 1: #include  “ElementContainerArray.hh”
 2: #include  “Domain.hh”
 3: #include  “ParamSet.hh”
 4: #include <unistd.h>              // for sleep(3)
 5: #include <sys/select.h>          // for select(2)
 6:
 7: ElementContainerArray src;   // an element container array as source data
 8: vector<Domain> domains       // domain for each elementContaier
 9: ParamSet paramSet            // data container for initial parameter values
10:
11: vector< vector<Double> > funcMatrix;  // a set of fitting functions
12: vector<Double> param;     // a list of paremeters
13: ElementContainerArray result;
14:
15: for (UInt4 i=0; i<eca.PutSize(); ++i) {          // domain の初期化
17:     ElementContainer *ec=eca.PutPointer(i);
18:     string xkey=ec->PutXKey();
19:     domains.push_back(new Domain(*ec, 0, ec->PutSize(xkey)-1));
20: }
21:
22: MultiDataLevmar method=*(new MultiDataLevmar()); // MultiDataLevmar のインスタンスの生成
23:
24: ParamSet paramSet=method.setDefaultParam(src); // デフォルト値の設定
25: paramSet.replace(MultiDataLevmar.CONSTRAIN,           MultiDataLevmar.NO_CONSTRAIN); // 制約タイプ
26: paramSet.replace(MultiDataLevmar.USE_NUMERICAL_DIFF, true);                    // 数値微分の実行
27: paramSet.replace(MultiDataLevmar.DIFF_METHOD,        MultiDataMethod.FOWARD);  // 数値部分の方法
28: paramSet.replace(MultiDataLevmar.USE_DATA_WEIGHTS,   true);    // 重み付き最小二乗法
29: paramSet.replace(MultiDataLevmar.MAX_ITERATIONS,     1000);    // 最大繰り返し回数
30: paramSet.replace(MUltiDataLevmar.OUTPUT_INTERVAL,    50);      // 出力間隔
31: paramSet.Add(MultiDataLevmar.FUNCTIONS,          funcMatrix);  // フィッティング関数の設定
32: paramSet.Add(MultiDataLevmar.PARAMETER_VALUES,   param);       // フィッティングパラメータの設定
33:
34: fd_set rdfs; FD_ZERO(&rdfs); FD_SET(0 &rdfs);                  // 入力待ちデバイスの設定
35: time_val waitTime; waitTime.tv_sec=1; waitTime.tv_usec=0;      // 待ち合わせ時間の設定
36: string stopCharSet=string( “q” );                              // 中断文字
36: if (method.checkParam(src, domains, paramSet)) {   // パラメータの整合性のチェック
37:     method.toInnderForm(src, domain, paramSet);    // パラメータの内部形式への変換（初期化）
38:     method.fit();                                  // フィッティングの開始
39:
40:     while (method.isFitting()) {
41:         retval=select(1, rfds, NULL, NULL, waitTime);
42:         if (retval >= 0) {
43:             std::cin >> c;   // 1 文字読み取り
44:             if (stopCharSet.find(c) != string::npos) { // 読み取った文字が中断を意味する文字
45:                 method.stopFit();                      // フィッティングの強制停止指示
46:                 while (method.isFitting()) { sleep(1); } // 完全に停止するまでの待ち合わせ
47:                 break;
48:              }
49:         }
50:     }
51:
52:     method.eval();                                 // フィッティング関数の値の計算
53:     method.toElementContainerArray(src, result);   // 関数値を ElementContainerArray にコピー
54:     ParamSet fittedParam=method.getFittedParam();  // フィット後のパラメータの取得
55: }
```

Python 用コード (src/driver/test_src/testMultiDataPeakFit.py より抜粋)

```
 1: import Manyo
 2: import Adv
 3:
 1: src = io.ReadElementContainerArray(options.inputFile) #  ElementContainerArray
 2:
 3: domainList=[]
 4: for i in range(src.PutSize()):
 5:    ec=src.Put(i)
 6:    domain=Adv.Domain(ec, ec.Put(ec.PutXKey()).front(), ec.Put(ec.PutXKey()).back())
 8:    domainList.append(domain)
 9:
10: method=Adv.MultiDataLevmar(src, Adv.MULTIDATA_LEVMAR)       # メソッドの生成
11: paramSet=method.setDefaultParam()  # パラメータのデフォルト値を設定
12:
13: # Levmar 用パラメータを設定
14: paramSet.replace(Adv.MultiDataLevmar.CONSTRAIN,           Adv.MultiDataLevmar.NO_CONSTRAIN)
15: paramSet.replace(Adv.MultiDataLevmar.USE_NUMERICAL_DIFF, Adv.MultiDataLevmar.DEFAULT_USE_NUMERICAL_DIFF)
17: paramSet.replace(Adv.MultiDataLevmar.DIFF_METHOD,        Adv.MultiDataLevmar.DEFAULT_DIFF_METHOD)
18: paramSet.replace(Adv.MultiDataLevmar.USE_DATA_WEIGHTS,   Adv.MultiDataLevmar.DEFAULT_USE_DATA_WEIGHTS)
19: paramSet.replace(Adv.MultiDataLevmar.MAX_ITERATIONS,     Adv.MultiDataLevmar.DEFAULT_MAX_ITERATIONS)
20: paramSet.replace(Adv.MultiDataLevmar.OUTPUT_INTERVAL,    Adv.MultiDataLevmar.DEFAULT_OUTPUT_INTERVAL)
21:
22: functions=[]
23: cppToPython=Manyo.CppToPython()
24: for i in range(src.PutSize()):
25:     parser = Adv.FuncParser( “g g g” )
27:     functions.append(parser.parse)
28:
29: paramSet.add(Adv.MultiDataLevmar.FUNCTIONS, functions);
30:
31: fittingParams=Manyo.VrctorTool.MakeVectorDouble();  # フィッティングパラメータの初期値
32: fittingParams.push_back( 60000.0)
33: …
34: paramSet.add(Adv.MultiDataLevmar.PARAMETERS_VALUES, fittingParams);
35:
36: if method.checkParam(src, domainList, paramSet):
38:     method.toInnerForm(src. domainList, paramSet) # 内部変数の初期化
40:     method.fit()                                  # フィッティングの開始
41:
42:     while method.isFitting():  # wait 処理
43:         sleep(1)
44:
45:     method.eval()                             # フィッティング関数の値の計算
46:     result=Manyo.ElementContainerArray(src.PutHeader())
47:     method.toElementContainerArray(src, result)    # 結果(関数の値)を ElementContainerArray に
48:
49:     fittedParamSet = method.getFittedParam() # フィッティング後のパラメータの値の取得
```

4. データコンテナ

4.1. ParamSet クラス

ParamSet は，計算アルゴリズムで必要なデータをクラス間でやり取りするデータコンテナである．スカラー型の値だけでなく，ベクトルおよび行列まで収めることができる．使用できるパラメータの型を表 4.1 に示す．

表 4.1 パラメータとして使用できるデータ型.

データ型	説明
Bool	論理値，スカラー
UInt4	符号なし整数．スカラー
Int4	整数．スカラー
Double	倍精度実数．スカラー
vector<Double>	倍精度実数のベクトル
vector< vector<Double> >	倍精度実数の行列（二次元配列)
vector<FuncBase*>	ピークフィットを行うときフィッティングを行う関数（を指すポインタ）のリスト(配列).

リスト 4.1 C++での ParamSet の使用例．スカラーの場合

```
 1:#include “ParamSet.hh”          // ヘッダーファイル ParamSet.hh
 2:
 3: ParamSet param;                // ParamSet のインスタンス
 4: string key;                    // パラメータを識別するキー
 5: T value, another_value;        // T: Bool, UInt4, Int4, Double のいずれか
 6:
 7: param.add(key, value);          // パラメータの値の設定.
 8: param.replace(key, another_value) // 既存パラメータの値の再設定.
 9:
10: Bool   v1=param.getBool(key1)     // 論理値の読みだし.
11: Int4   v2=param.getInt4(key2)     // (符号付き)整数値の読み出し
12: UInt4  v3=param.getUInt4(key3)    // (符号なし)整数値の読み出し
13: Double v4=param.getDouble(key4)   // 実数値の読み出し
14:
15: param.erase(key)                 // 与えられたキーを持つパラメータの削除
16: Bool tmp=param.contain(key)       // 与えられたキーを持つパアメータの有無
```

スカラー型の値の設定例を

リスト 4.1 とリスト 4.2 に示す．パラメータの値を新規設定するためには，パラメータを区別するための文字列キーとパラメータの値の組を add メンバー関数に与える．既存のパラメータの値の書き換えは，文字列キーと別の値を replace 関数に与えればできる．値

は，get 型名 関数にキーを与えれば取り出せる．不要なデータは erase で削除できる．また，ParamSet 型のインスタンスにキーkey を持つパラメータが登録されているか否かを contain で調べることができる．

リスト 4.2　Python での ParamSet の使用例．スカラー型の場合

```
 1: param=Manyo.ParamSet()              # ParamSet のインスタンス
 2: key=” key”                          # パラメータを識別するキー
 3: value=…                             # v は整数または実数型
 4:
 5: param.add(key, value);              # パラメータの値の設定.
 6: param.replace(key, another_value)   # 既存パラメータの値の再設定.
 7:
 8: v1=param.getBool(key1)              # パラメータの値の読みだし
 9: v2=param.getInt4(key2)
10: v3=param.getDouble(key3)
11:
12: param.erase(key)                    # 与えられたキーを持つパラメータの削除
13: tmp=param.contain(key)              # 与えられたキーを持つパアメータの有無
```

vector<Double>のパラメータの使用法をリスト 4.3 に示す．基本的な使用法はスカラー型の場合と同じである．ただし，値の取り出しは getVector 関数を使う．また，vector<Double>の要素の値を変更することもできる．

リスト 4.3　C++での ParamSet の使用例．ベクトル型の場合

```
 1: #include “ParamSet.hh”
 2:
 2: ParamSet param;                     # ParamSet のインスタンス
 2: string key;                         # パラメータを識別するキー
 3: vector<Double> vect, another_vet;   # vector 型のインスタンス
 4: UInt4 i;                            # ベクトル成分を指す番号
 5: Double v;                           # スカラーの値
 6:
 7: param.add(key, vect);               # ベクトルの設定.
 8:
 9: param.replace(key, another_vect)    # 既存キーに対してベクトルを再設定.
10: param.replace(key, i, v)            # i 番目の成分の値を v に変更
11:
12: vector<Double> v=param.getVector(key);  # ベクトルの読み出し.
13: v=param.getDouble(key, i);              # i 番目の成分の値の読み出し
14: UInt4 param.size(key)               # 与えられたキーを持つベクトルの要素数
15:
16: param.erase(key)                    # 与えられたキーを持つベクトルの削除
17: Bool tmp=param.contain(key)         # 与えられたキーを持つベクトルの有無
```

リスト 4.4　Python での ParamSet の使用例．ベクトル型の場合

```
 1: param=Manyo.ParamSet();              # ParamSet のインスタンス
 2: key=”…”;                             # パラメータを識別するキー
 3: vect       =Manyo.DoubleVector()     # vector 型のインスタンス
 4:
 5: param.add(key, vect);                # ベクトルの設定.
 6:
 7: param.replace(key, another_vect)     # 既存キーに対してベクトルを再設定.
 8: param.replace(key, i, v)             # i 番目の成分の値を v に変更
 9:
10: v=param.getVector(key);       # ベクトルの読み出し.
11: v=param.getDouble(key, i);    # i 番目の成分の値の読み出し
12: n=param.size(key)             # 与えられたキーを持つベクトルの要素数
13:
14: param.erase(key)                    # 与えられたキーを持つベクトルの削除
15: Bool tmp=param.contain(key)         # 与えられたキーを持つベクトルの有無
```

vector< vector<Double> > 型のパラメータの使用例をリスト 4.5 に示す．基本的な使い方はスカラー型と同じである．ただし，値の取り出しは getMatrix 関数を使用する．行列の行数および列数はそれぞれ getRowSize と getColumnSize で得られる．

リスト 4.5　C++での PramSet の使用例．行列型の場合

```
 1: #include  “ParamSet.hh”
 2:
 3: ParamSet param;                      # ParamSet のインスタンス
 4: string key;                          # パラメータを識別するキー
 5: vector< vector<Double> > vect, another_vect; # 行列型のインスタンス
 6: UInt4 i, j;                          # ベクトル成分を指す番号
 7: Double v;                            # Double 型の値
 8:
 9: param.add(key, vect);                # 行列の設定.
10:
11: param.replace(key, another_vect)     # 既存キーに対して行列を再設定.
12: param.replace(key, i, j, v)          # (i, j)成分の値を v に変更
13:
14: vector< vector<Double> > m=param.getMatrix(key);  # 行列の読み出し.
15: v=param.getDouble(key, i, j);              # (i, j)成分の値の読み出し
16:
17: UInt4 r=param.getRowSize(key)        # 与えられたキーを持つ行列の行数
18: UInt4 c=param.getColumnSize(key)     # 与えられたキーを持つ行列の行数
19:
20: param.erase(key)                     # 与えられたキーを持つベクトルの削除
21: Bool tmp=param.contain(key)          # 与えられたキーを持つベクトルの有無
```

リスト 4.6　Python での ParamSet の使用例．行列型の場合

```
 1: param=Manyo.ParamSet();              # ParamSet のインスタンス
 2: string key;                          # パラメータを識別するキー
 6:
 7: param.add(key, vect);                # 行列の設定.

 8: param.replace(key, another_vect)     # 既存キーに対して行列を再設定.
 9: param.replace(key, i, j, v)          # (i, j)成分の値を v に変更

10: m=param.getMatrix(key);  # 行列の読み出し.
11: v=param.get(key, i, j);              # (i, j)成分の値の読み出し

12: r=param.getRowSize(key)              # 与えられたキーを持つ行列の行数
13: c=param.getColumnSize(key)           # 与えられたキーを持つ行列の行数

14: param.erase(key)                     # 与えられたキーを持つベクトルの削除
15: Bool tmp=param.contain(key)          # 与えられたキーを持つベクトルの有無
```

4.2.　Domain クラス

Domain クラスは ElementContainer が保持するヒストグラムの計算対象となる領域を保持する．領域を指定する方法は 3 つある．一つ目の方法は領域境界となる座標の下限値 A と上限値 B を指定することである．指定された座標値に対して最も近い bin 境界値[1]が検索されて，見つかった値が使用される．二つ目の方法は bin 境界値の番号 L と U を指定することである．三つ目の方法は，bin 番号 I と J を指定することである．

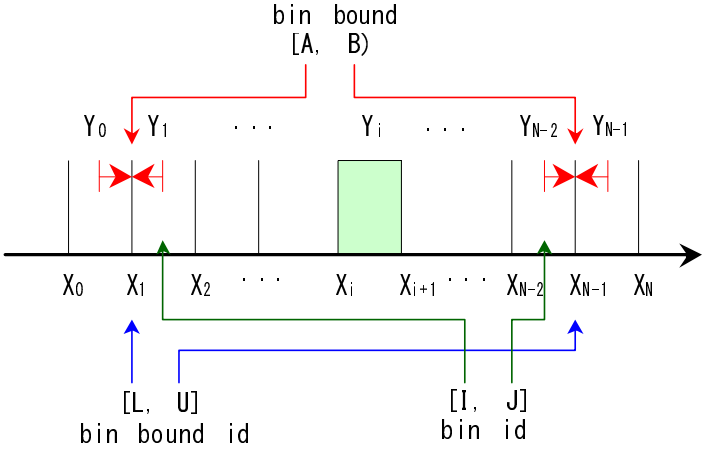


図 4.1　ElementContainer に対する Domain の指定法．計算対象となる領域はヒストグラムの bin 境界の座標値（A と B），bin 境界の番号（L と U）または bin 番号（I と J）で指定できる．

[1] ここでは閉開区間$I_i = [X_i, X_{i+1})$を bin，各機関の境界値を bin 境界値と呼んでいる．

Domain クラスの C++および Python での使用例をそれぞれリスト 4.7 とリスト 4.8 に示す．Domain クラスのインスタンスを生成するためには，ElementContainer のインスタンスが必須である（6 行目）．前述したように領域の指定は，座標値，bin 境界番号および bin 番号で設定できる（8-10 行目）．領域に含まれる bin だけを取り出すことができる（12 行目）．さらに bin の中心座標のリストを取得できる（13 行目）．領域境界の両端をだけを，座標値（16 行目），bin 境界値の番号（17 行目）または bin 番号（18 行目）で取り出すことができる．領域境界は片方だけを取り出すこともできる（20-25 行目）．

リスト 4.7　Domain クラスの使用例（C++版）.

```
 1: #include  “ElementContainer”
 2: #include  “Domain.hh”
 3:
 4: ElementContainer src;  // データソースとなる ElementContainer
 5:
 6: // Domain クラスのインスタンスの生成．データソースと対象領域を指定
 7: Domain domain = *(new Domain(src, 1.0, 10.0));  // 領域を bin 境界値で指定
 8: Domain domain = *(new Domain(src,   1,    9));  // 領域を bin 番号で指定
 9:
10: Domain domain = *(new Domain()               );  // ソースも領域も未指定
11:
12: domain.setSource(src);          // ソースとして src を指定
13:
14: domain.setBinBound(1.0, 10.0)  // 領域を座標値で指定
15: domain.setBinBoundID(1, 10);    // 領域を bin 境界の番号で指定
16: domain.setBinID(1, 9);          // 領域を bin 番号で指定
17:
18: vector<Double> bins=domain.getBin()       //領域内の各 bin 境界値を全て取得
19: vector<Double> center=domain.createXC()  //領域内の各 bin の中点を生成
20:
21: //領域の下限および上限を取得
22: vector<Double> bb   =domain.getBinBound()   //  bin 境界値で取得
23: vector<UInt4>  bbid =domain.getBinBoundID() //  bin 境界値の番号で取得
24: vector<UInt4>  binID=domain.getBinID()      //  bin 番号で取得
25:
26: Double upperBound  =domain.getUpperBound();   // 領域の上限の bin 境界値
27: Double lowerBound  =domain.getLowerBound();   // 領域の下限の bin 境界値
28: UInt4  upperBoundID=domain.getUpperBoundID(); // 領域の上限の bin 境界番号
29: UInt4  lowerBoundID=domain.getLowerBoundID(); // 領域の下限の bin 境界番号
30: UInt4  upperBinID  =domain.getUpperBinID();   // 領域の上限の bin 番号
31: UInt4  lowerBinID  =domain.getLowerBinID();   // 領域の下限の bin 番号
```

リスト 4.8　Domain クラスの使用例（Python 版）.

```
 1: src = Manyo.ElementContainer
 2:
 3: // Domain クラスのインスタンスの生成. データソースと対象領域を指定
 4: domain = Manyo.Domain(src, 1.0, 10.0)
 5: domain = Manyo.Domain(src,   1,    9)
 6  domain = Manyo.Domain()
 7:
 8: domain.setSource(src)     // ソースの指定
 9:
11: domain.setSource(src)           // ソースとして src を指定
12:
13: domain.setBinBound(1.0, 10.0)  // 領域を座標値で指定
14: domain.setBinBoundID(1, 10);   // 領域を bin 境界の番号で指定
15: domain.setBinID(1, 9);         // 領域を bin 番号で指定
16:
17: bins=domain.getBin()        //領域内の各 bin 境界を取得
18: center=domain.createXC()  //領域内の各 bin の中点を取得
19:
20: //領域の下限および上限を取得
21: bb   =domain.getBinBound()   //  bin 境界値で取得
22: bbid =domain.getBinBoundID() //  bin 境界値の番号で取得
23: binID=domain.getBinID()      //  bin 番号で取得
24:
25: upperBound  =domain.getUpperBound();   // 領域の上限の bin 境界値
26: lowerBound  =domain.getLowerBound();   // 領域の下限の bin 境界値
27: upperBoundID=domain.getUpperBoundID(); // 領域の上限の bin 境界番号
28: lowerBoundID=domain.getLowerBoundID(); // 領域の下限の bin 境界番号
29: upperBinID  =domain.getUpperBinID();   // 領域の上限の bin 番号
30: lowerBinID  =domain.getLowerBinID();   // 領域の下限の bin 番号
```

4.3.　PeakData クラス，Peak クラス

ピークサーチを行った結果を保持する．使用法を以下に示す.

リスト 4.9　C++での PeakData クラスの使用例. ピークサーチの結果を取り出している.

```
 1: #include  “ElementContainer.hh”
 2: #include  “MovingAverage.hh”
 3: #include  “PeakData.hh”
 4: #include  “PeakSearch.hh”
 5:
 6: ElementContainer src
 7:
 8: // ピークサーチドライバーの生成, ソース:src,  平滑化法: MovingAverage
 9: PeakSearch *peakSearch=new PeakSearch(&src, new MovingAverage());
10: // 平滑化用パラメータの設定
11: peakSearch.setParam(MovingAverage::WINDOW_WIDTH, 111);
12: peakSearch.setParam(MovingAverage::WINDOW_UPPER,  55);
13:
14: if ( peakSearch.checkParam() ) {  // パラメータの整合性チェック
15:     peakSearch.execute();          // ピークサーチ
16: }
17:
18: PeakData peakData = peakSearch.getPeaks(); // ピークデータの取り出し
19: peakData.Dump();                           // ピークデータを標準出力へ表示
20:
21: UInt4 i;
22: UInt4 n=peakData.size();       // 検出されたピークまたは肩の総数
23: Peak p=peakData.getPeak(i);    // i 番目のピーク/肩の取り出し
24: Double h=p.height();           // ピーク/肩の高さ
25: Double x=p.position();         // ピーク/肩の位置
26: Double w=p.width();            // ピーク/肩の高さ h/2 での幅
27:
28: vector<Double> v=peakData.toVector(); // 全ピークのシリアライズ
29:                                       //     各ピークの高さ, 位置, 幅の順
```

5. 組み込み関数

5.1. 組み込み関数の共通形式

頻繁に使用する関数については組み込み関数が用意されている．

現時点では全ての組み込み関数のコンストラクタは引数を持たない．したがって組み込み関数を直接使用する場合，単に組み込み関数のクラス名に‘()’を付けるだけである．C++では以下の通りである．

```
FuncBase *func = new クラス名();
クラス名 *func = new クラス名();
```

ここで FuncBase は組み込み関数クラスの親となる抽象クラスである．

Python では，以下のようにすれば直ちに該当する関数が使用できる．

```
func=Manyo.Gaussian()
```

全ての組み込み関数が持つメソッドを表 5.1 に示す．

表 5.1 関数の共通仕様

メンバー関数	説明
string getName()	関数名を返す．
string getSymbol()	関数のシンボル文字列（略記号）を返す．
UInt4 getNumberOfParams()	パラメータ $\boldsymbol{p}$ の要素数を返す．
Double eval(Double x, vector<Double> &p)	座標点 x での関数値 $f(x;\boldsymbol{p})$ を計算
Double der1st(Double x, vector<Double> &p)	座標点 x での 1 階導関数の値 $\frac{\mathrm{d}}{\mathrm{d}x}f(x;\boldsymbol{p})$
Double der2nd(Double x, vector<Double> &p)	座標点 x での 2 階導関数の値 $\frac{\mathrm{d}^2}{\mathrm{d}x^2}f(x;\boldsymbol{p})$
vector<Dobule> *gradient(Double x, vector<Double> &p)	座標点 x でのパラメータ $\boldsymbol{p}$ による勾配 $\nabla_{\mathrm{p}}f(x;\boldsymbol{p})$． i 番目の成分は $\frac{\partial}{\partial p_i}f(x;\boldsymbol{p})$
vector<Double> eval(vector<Double> &x, vector<Double> &p)	各座標点 x_i の関数値 $f(x_i;\boldsymbol{p})$ を一度に返す．

vector<Double> der1st(vector<Double> &x, vector<Double> &p)	各座標点x_iの 1 階導関数の値$\frac{\mathrm{d}}{\mathrm{d}x}f(x_i;\boldsymbol{p})$を一度に返す.
vector<Double> der2nd(vector<Double> &x, vector<Double> &p)	各座標点x_iの 2 階導関数の値$\frac{\mathrm{d}^2}{\mathrm{d}x^2}f(x_i;\boldsymbol{p})$ 一度に返す.
vector< vector<Dobule> >*gradient(vector<Double> &x, vector<Double> &p)	各座標点x_iのパラメータ $\boldsymbol{p}$ による勾配$\nabla_{\mathrm{p}}f(x_i;\boldsymbol{p})$を一度に返す．その(i, j)成分 gradient(x, p)->at(i)->at(j)は$\frac{\partial}{\partial p_j}f(x_i;\boldsymbol{p})$である.

5.2. 組み込み関数

以下に使用できる関数について記述する．各関数については，関数の標識と関数のパラメータ表が書かれている.

関数のパラメータは，実際は Double 型の配列（を指すポインター）または vector<Double>型として渡される．パラメータの並び順は，Double 型の配列でも vector<Double>型でも共通である．先頭から順にどのような役割を担っているかは，パラメータ表で示されている.

5.2.1. 定数関数（クラス Constant, 関数名 constant, 記号 *c*）

$$c(x;\mathrm{c}) = c(x;\boldsymbol{p}) = c$$

表 5.2 定数関数のパパラメータ

パラメータ	意味	属性，制約
c=*p*[0]	定数値	

定数を表す関数である．xには依存しない.
次の関係式が成り立つ.

$$\frac{\partial \mathrm{c}(x;c)}{\partial x} = \frac{\partial^2 c(x;c)}{\partial x^2} = 0$$

$$\frac{\partial \mathrm{c}(x;c)}{\partial \mathrm{c}} = 1$$

5.2.2. ガウス関数（クラス Gaussian，関数名 gaussian，記号 *g*）

$$g(x;\ h,c,w) = g(x;\ \boldsymbol{p}) = h\exp\left[-(\log 2)\left(\frac{x-c}{w}\right)^2\right]$$

表 5.3　ガウス関数のパラメータ

パラメータ	意味	属性，制約
$h = p[0]$	高さ	
$c = p[1]$	ピーク位置	
$w = p[2]$	半値半幅(HWHM)	$w \neq 0$ 標準偏差 $\sigma = \frac{w}{\sqrt{2\log 2}}$

関数の広がりを表すパラメータとして標準偏差ではなく**半値半幅**を採用している．
以下の等式が成り立つ．

$$g(x;\ h,c,w) = \ g(x-c;\ h,0,w) = g(0;\ h,x-c,w) = hg(x;\ 1,c,w)$$

$$g(c;\ h,c,w) = g(0;\ h,0,w) = h$$

$$g(c \pm \mathrm{w};\ h,c,w) = g(\pm \mathrm{w};\ h,0,w) = g(0;\ h,\pm w,w) = \frac{1}{2}g(0;\ h,0,w) = \frac{h}{2}g(0;\ 1,0,w)$$

$$\frac{\partial g(x;\ h,c,w)}{\partial h} = \ g(x;\ 1,c,w)$$

$$\frac{\partial g(x;\ h,c,w)}{\partial c} = \ -\frac{\partial g(x;\ h,c,w)}{\partial x}$$

$$\frac{\partial g(x;\ h,c,w)}{\partial w} = \ -\left(\frac{x-c}{w}\right)\frac{\partial g(x;\ h,c,w)}{\partial x}$$

5.2.3. 誤差関数（クラス ErrorFunc，関数名 error function，記号 erf）

$$\mathrm{erf}(x;h,c,w) = \mathrm{erf}(x;\ \boldsymbol{p}) = h\frac{2}{\sqrt{\pi}}\frac{\sqrt{\log 2}}{w}\int_c^x \mathrm{d}t\ \exp\left[-(\log 2)\left(\frac{t-c}{w}\right)^2\right]$$

$$= h\frac{2}{\sqrt{\pi}}\int_0^{\sqrt{\log 2}\frac{x-c}{w}} \mathrm{d}u\ \exp[-u^2] = h\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w}\right)$$

表 5.4　誤差関数のパラメータ

パラメータ	意味	属性，制約
$h = p[0]$	高さ	
$c = p[1]$	関数値が 0 となる位置	
$w = p[2]$	被積分関数であるガウス	$w \neq 0$

	関数の半値半幅	標準偏差 $\sigma = \frac{w}{\sqrt{2\log 2}}$

以下の関係式が成り立つ.

$$\mathrm{erf}(x;h,c,w) = h\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w};1,0,1\right) = h\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w}\right)$$

$$\frac{\partial}{\partial x}\mathrm{erf}(x;h,c,w) = h\frac{\partial}{\partial x}\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w}\right)$$

$$= \frac{\partial\left[\sqrt{\log 2}\frac{x-c}{w}\right]}{\partial x}\frac{\partial}{\partial\left[\sqrt{\log 2}\frac{x-c}{w}\right]}h\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w}\right)$$

$$= \frac{2}{\sqrt{\pi}}\frac{\sqrt{\log 2}}{w}\ h\exp\left[-(\log 2)\left(\frac{x-c}{w}\right)^2\right] = \frac{2}{\sqrt{\pi}}\frac{\sqrt{\log 2}}{w}g(x;\ h,c,w)$$

$$\frac{\partial^2}{\partial x^2}\mathrm{erf}(x;h,c,w) = \frac{2}{\sqrt{\pi}}\frac{\sqrt{\log 2}}{w}\ \frac{\partial}{\partial x}g(x;\ h,c,w) = \frac{2}{\sqrt{\pi}}\frac{\sqrt{\log 2}}{w}$$

$$\frac{\partial}{\partial h}\mathrm{erf}(x;h,c,\sigma) = \frac{\partial}{\partial h}h\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w};1,0,1\right) = \mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w}\right)$$

$$\frac{\partial}{\partial c}\mathrm{erf}(x;h,c,w) = h\frac{\partial\left[\sqrt{\log 2}\frac{x-c}{w}\right]}{\partial c}\frac{\partial}{\partial\left[\sqrt{\log 2}\frac{x-c}{w}\right]}\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w}\right)$$

$$= -\frac{2\sqrt{\log 2}}{\sqrt{\pi}}\frac{h}{w}\ \exp\left[-(\log 2)\left(\frac{x-c}{w}\right)^2\right] = -\frac{\partial}{\partial x}\mathrm{erf}(x;h,c,w)$$

$$\frac{\partial}{\partial w}\mathrm{erf}(x;h,c,w) = h\frac{\partial\left[\sqrt{\log 2}\frac{x-c}{w}\right]}{\partial w}\frac{\partial}{\partial\left[\sqrt{\log 2}\frac{x-c}{w}\right]}\mathrm{erf}\left(\sqrt{\log 2}\frac{x-c}{w}\right)$$

$$= -\frac{2}{\sqrt{\pi}}\frac{1}{w}\left(\sqrt{\log 2}\frac{x-c}{w}\right)g(x;\ h,c,w)$$

5.2.4. ローレンツ関数（クラス Lorentzian, 関数名 lorentzian, 記号 *l*）

$$l(x;h,c,w) = l(x;\ \boldsymbol{p}) = \frac{h}{\pi w\left[\left(\frac{x-c}{w}\right)^2+1\right]}$$

表 5.5　ローレンツ関数のパラメータ

パラメータ	意味	属性，制約
$h = p[0]$	高さ	
$c = p[1]$	ピーク位置	
$w = p[2]$	半値半幅(HWHM)	$w \neq 0$

数学（統計学）では Chochy 分布関数と呼ばれることが多い.
以下の等式が成り立つ.

$$l(x;\ h, c, w) = \ l(x - c;\ h, 0, w) = l(0;\ h, x - c, w)$$

$$l(c;\ h, c, w) = l(0;\ h, 0, w) = \frac{\mathrm{h}}{\pi \mathrm{w}}$$

$$l(c \pm \mathrm{w};\ h, c, w) = l(\pm \mathrm{w};\ h, 0, w) = l(0;\ h, \pm w, w) = \frac{1}{2} l(0;\ h, 0, w)$$

$$\frac{\partial l(x;\ h, c, w)}{\partial h} = \ l(x;\ 1, c, w)$$

$$\frac{\partial l(x;\ h, c, w)}{\partial c} = \ -\frac{\partial l(x;\ h, c, w)}{\partial x}$$

$$\frac{\partial l(x;\ h, c, w)}{\partial w} = \ -\left(\frac{x - c}{w}\right) \frac{\partial l(x;\ h, c, w)}{\partial x}$$

5.2.5.　拡張ローレンツ関数（クラス AugmentedLorentzian, 関数名 augmented lorentzian, 記号 *al*）

$$al(x; h, c, \beta, \alpha) = \ al(x;\ \boldsymbol{p}) = h \frac{\Gamma(\alpha)}{\Gamma\left(\alpha - \frac{1}{2}\right) \Gamma\left(\frac{1}{2}\right)} \frac{1}{\beta \left[\left(\frac{x - c}{\beta}\right)^2 + 1\right]^{\alpha}}$$

表 5.6　拡張ローレンツ関数のパラメータ

パラメータ	意味	属性，制約
$h = p[0]$	高さ	
$c = p[1]$	ピーク位置	
$\beta = p[2]$	幅	$\beta \neq 0 \quad HWHM\ w = \beta\sqrt{2^{\frac{1}{\alpha}} - 1}$
$\alpha = p[3]$	冪	$\alpha \geq 1$

α=1 のときローレンツ関数に一致する.
以下の等式が成り立つ.

$$al(x;\ h, c, \beta, \alpha) = \ al(x - c;\ h, 0, \beta, \alpha) = al(0;\ h, x - c, \beta, \alpha)$$

$$al(c;\ h,c,\beta,\alpha) = al(0;\ h,0,\beta,\alpha) = h\frac{\Gamma(\alpha)}{\Gamma\left(\alpha-\frac{1}{2}\right)\Gamma\left(\frac{1}{2}\right)\beta}$$

ここで$\Gamma(\alpha)$は gamma 関数である.

$$al(c \pm \mathrm{w};\ h,c,\beta,\alpha) = al(\pm\mathrm{w};\ h,0,\beta,\alpha) = al(0;\ h,\pm w,\beta,\alpha) = \frac{1}{2}al(0;\ h,0,\beta,\alpha)$$

$$\frac{\partial al(x;\ h,c,w)}{\partial h} = \ al(x;\ 1,c,w)$$

$$\frac{\partial al(x;\ h,c,w)}{\partial c} = \ -\frac{\partial al(x;\ h,c,w)}{\partial x}$$

$$\frac{\partial al(x;\ h,c,w)}{\partial w} = \ -\left(\frac{x-c}{w}\right)\frac{\partial al(x;\ h,c,w)}{\partial x}$$

5.2.6. 擬フォークト関数 I（クラス PseudoVoigt1，関数名 pseudo voigt 1，記号 pv1）

$$pv1(x;\ h,c,\mathrm{w},\mu) = pv1(x;\ \boldsymbol{p}) = \mu\mathrm{g}(x;\ h,c,\mathrm{w}) + \ (1-\ \mu)\mathrm{l}(x;h,c,\mathrm{w})$$

$$= h\left[\mu\exp\left\{-(\log 2)\left(\frac{x-c}{w}\right)^2\right\} + \frac{(1-\mu)}{\pi\mathrm{w}\left\{\left(\frac{x-c}{\mathrm{w}}\right)^2+1\right\}}\right]$$

表 5.7　擬フォークト関数 I のパラメータ

パラメータ	意味	属性，制約
$h = p[0]$	高さ	
$c = p[1]$	ピーク位置	
$w = p[2]$	半値半幅(HWHM)	$w \neq 0$
$\mu = p[3]$	線型混合率	$0 \leq \mu \leq 1$

ガウス関数とローレンツ関数の半値半幅は共通である.

以下の等式が成り立つ.

$$pv1(x;\ h,c,\mathrm{w},\mu) = pv1(x-c;\ h,0,\mathrm{w},\mu) = pv1(0;\ h,x-c,\mathrm{w},\mu)$$

$$pv1(c;\ h,c,\mathrm{w},\mu) = pv1(0;\ h,0,\mathrm{w},\mu) = \mathrm{h}\left(\mu - \frac{(1-\mu)}{\pi\mathrm{w}}\right)$$

$$pV1(c \pm w;\ h,c,\mathrm{w},\mu) = \mathrm{pV1}(\pm w;\ h,0,\mathrm{w},\mu) = \mathrm{pV1}(0;\ h,\pm w,\mathrm{w},\mu) = \frac{\mathrm{h}}{2}\left(\mu - \frac{(1-\mu)}{\pi\mathrm{w}}\right)$$

$$\frac{\partial pv1(x;\ h,c,\mathrm{w},\mu)}{\partial h} = pv1(x;\ 1,c,\mathrm{w},\mu)$$

$$\frac{\partial pv1(x;\ h,c,\mathrm{w},\mu)}{\partial c} = -\frac{\partial pv1(x;\ h,c,\mathrm{w},\mu)}{\partial x}$$

5.2.7. 擬フォークト関数(II) （クラス PseudoVoigt2，関数名 pseudo voigt 2，記号 *pv2*）

$$pv2(x;\ h,c,\sigma,\gamma) = pv2(x;\ \boldsymbol{p}) = \mu g(x;\ h,c,\sigma) + \ (1-\ \mu)l(x;h,c,\gamma)$$

$$= h\left[\mu\exp\left\{-(\log 2)\left(\frac{x-c}{\sigma}\right)^2\right\} + \frac{(1-\mu)}{\pi\gamma\left\{\left(\frac{x-c}{\gamma}\right)^2+1\right\}}\right]$$

表 5.8　擬フォークト関数 II のパラメータ

パラメータ	意味	属性，制約
$h = p[0]$	高さ	
$c = p[1]$	ピーク位置	
$\sigma = p[2]$	ガウス関数の半値半幅(HWHM)	$\sigma > 0$
$\gamma = p[3]$	ローレンツ関数の半値半幅(HWHM)	$\gamma > 0$
$\mu = p[4]$	線型混合率	$0 \le \mu \le 1$

ガウス関数とローレンツ関数の**半値半幅**を別々に設定できる．

$$pv2(x;\ h,c,\sigma,\gamma,\mu) = pv2(x-c;\ h,0,\sigma,\gamma,\mu) = pv2(0;\ h,x-c,\sigma,\gamma,\mu)$$

$$pv2(c;\ h,c,\sigma,\gamma,\mu) = \mathrm{pv2}(0;\ h,0,\sigma,\gamma,\mu) = h\left(\mu - \frac{(1-\mu)}{\pi\gamma}\right)$$

$$\frac{\partial pv2(x;\ h,c,\sigma,\gamma,\mu)}{\partial h} = pv2(x;\ 1,c,\sigma,\gamma,\mu)$$

$$\frac{\partial pv2(x;\ h,c,\sigma,\gamma,\mu)}{\partial c} = -\frac{\partial pv2(x;\ h,c,\sigma,\gamma,\mu)}{\partial x}$$

5.2.8. Stretched 指数関数のフーリエ変換

未実装

$$\mathrm{F}(\omega;h,\tau,\beta) = h\int_0^\infty \mathrm{d}t\ \exp\left[-\left(\frac{t}{\tau}\right)^\beta\right]\exp(\mathrm{i}\omega t)$$

表 5.9　stretched 指数関数のパラメータ

パラメータ	意味	属性，制約
$h = p[0]$	高さ	
$\tau = p[1]$	時定数	$\tau > 0$
$\beta = p[2]$	冪	$\beta > 0.1$

$$\mathrm{F}(\omega;h,\tau,\beta)=h\mathrm{F}(\omega;1,\tau,\beta)=\tau\mathrm{F}(\omega\tau;h,1,\beta)=h\tau\mathrm{F}(\omega\tau;1,1,\beta)$$

$$\mathrm{F}(-\omega;h,\tau,\beta)=\mathrm{F}^{*}(\omega;h,\tau,\beta)$$

$$\mathrm{F}(0;h,\tau,\beta)=h\tau\mathrm{F}(0;1,1,\beta)=h\tau\int_0^\infty \mathrm{d}t\,\exp\left[-t^{\beta}\right]=\frac{h\tau}{\beta}\int_0^\infty \mathrm{d}t\,t^{\frac{1}{\beta}-1}\exp[-t]=\frac{h\tau}{\beta}\Gamma\left(\frac{1}{\beta}\right)$$

$$=h\tau\Gamma\left(\frac{1}{\beta}+1\right)\to h\tau\Gamma(1)=h\tau\quad(\beta\to\infty)$$

$$\mathrm{F}(\omega;h,\tau,1)=h\tau\mathrm{F}(\omega\tau;1,1,1)=\ h\tau\int_0^\infty \mathrm{d}t\,\exp[-t]\exp(\mathrm{i}\omega\tau t)=h\tau\frac{1+\mathrm{i}\omega\tau}{1+(\omega\tau)^2}$$

$$\mathrm{F}(\omega;h,\tau,2)=h\tau\mathrm{F}(\omega\tau;1,1,2)=\ h\tau\int_0^\infty \mathrm{d}t\,\exp[-t^2]\exp(\mathrm{i}\omega\tau t)=h\tau\frac{\sqrt{\pi}}{2}\exp\left[-\frac{(\omega\tau)^2}{4}\right]$$

$$\frac{\partial\mathrm{F}(\omega;h,\tau,\beta)}{\partial\omega}=h\tau\frac{\partial\mathrm{F}(\omega\tau;1,1,\beta)}{\partial\omega}=h\tau\frac{\partial(\omega\tau)}{\partial\omega}\frac{\partial\mathrm{F}(\omega\tau;1,1,\beta)}{\partial(\omega\tau)}=h\tau^2\frac{\partial\mathrm{F}(\omega\tau;1,1,\beta)}{\partial(\omega\tau)}$$

$$\frac{\partial^{\mathrm{n}}\mathrm{F}(\omega;h,\tau,\beta)}{\partial\omega^n}=h\tau^{n+1}\frac{\partial^{\mathrm{n}}\mathrm{F}(\omega\tau;1,1,\beta)}{\partial(\omega\tau)^n}$$

$$\frac{\partial\mathrm{F}(\omega;h,\tau,\beta)}{\partial h}=\tau\mathrm{F}(\omega\tau;1,1,\beta)$$

$$\frac{\partial\mathrm{F}(\omega;h,\tau,\beta)}{\partial\tau}=h\left[\mathrm{F}(\omega\tau;1,1,\beta)+\tau\frac{\partial\mathrm{F}(\omega\tau;1,1,\beta)}{\partial\tau}\right]$$

$$=h\left[\mathrm{F}(\omega\tau;1,1,\beta)+\tau\frac{\partial(\omega\tau)}{\partial\tau}\frac{\partial\mathrm{F}(\omega\tau;1,1,\beta)}{\partial(\omega\tau)}\right]$$

$$=h\left[\mathrm{F}(\omega\tau;1,1,\beta)+\omega\tau\frac{\partial\mathrm{F}(\omega\tau;1,1,\beta)}{\partial(\omega\tau)}\right]$$

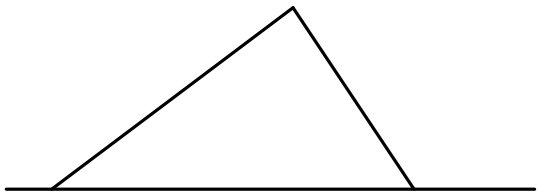
5.2.9. 減衰調和振子Ⅰ（クラス DampedHarmonicOscillator1，関数名 damped harmonic oscillator 1，記号 dho1）

$$dho1(\omega;h,\omega_0,\Gamma)=dho1(\omega;\boldsymbol{p})=\frac{h\sqrt{\omega_0^2-\Gamma^2}\Gamma\omega}{(\omega_0^2-\omega^2)^2+\ 4\Gamma^2\omega^2}$$

表 5.10　減衰調和振子のパラメータ

パラメータ	意味	属性，制約
$h=p[0]$	高さ	
$\omega_0=p[1]$	単振動の振動数	$\Gamma\leq\omega_0$
$\Gamma=p[0]$	摩擦係数	

5.2.10. 減衰調和振子Ⅱ（クラス DampedHarmonicOscillator2，関数名 damped harmonic



ERROR: undefined
OFFENDING COMMAND: -2147483648.-2147483648

STACK:

-mark-