Manual for the numerical functions package*

Bret R. Beck
Lawrence Livermore National Laboratory
UCRL-

August 1, 2023

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract #W-7405-ENG-48.

Contents

List of Tables

year	Male	Female
1871	1212	1231
1883	1215	1241
1885	—	621
1889	51	229
1895	11	31
1905	9	23
1915	9	21

Table 1: Male and female populations of a bird species on an island for the years census were taken. There was no census taken of the male population in 1885.

1 Introduction

The **ptwXY** C object (i.e., an instance of C typedef **ptwXYPoints**) and supporting C functions are designed to handle point-wise interpolative data representing a mathematical function[1] of one independent variable (i.e., y = f(x)). That is, a **ptwXY** object consist of an list of pairs (x_i, y_i) where $x_i < x_{i+1}$ and $y_i = f(x_i)$. Henceforth, the **ptwXY** C object and supporting C functions are called the **ptwXY** model. Routines supporting common operations on the points of f(x) are included in this package. As example, a function exists to add two **ptwXY** instances returning the sum as a **ptwXY** instances (i.e., $h(x) = f(x) + g(x)$ where $f(x)$, $g(x)$ and $h(x)$ are all **ptwXY** objects). The main intent for developing this library is for a fast XY math object for LLNL's FUDGE package which manipulates nuclear data (e.g., it can be used to add cross section from different reactions). However, this library may be useful for other packages.

As example of the usage of **ptwXY** objects consider the data in Table 1. This table list the male and female populations of a bird species on an island for several census years. In this example, x_i represents a census year and y_i represents the population for that year. Note that the male population is not given for the year 1885.

A portion of a C function to put the male and female populations into **ptwXY** objects and add them together to get the total population is shown in Table 2. In this example no error checking is shown. The output of this code, compressed into fewer lines, is show in Table 3.

The function **ptwXY_new2** allocates memory for a new **ptwXYPoints** object, initializes it and returns a pointer to the object. The second argument of this function in an interpolation flag. For all other arguments see Section 5.1.1. The function **ptwXY_setXYData** takes a pointer to a **ptwXYPoints** object as its second argument and copies the list of doubles given by the fourth

[1]By definition, a mathematical function is single valued which requires that $x_i < x_{i+1}$ if y_i is not equal to y_{i+1}. Since $x_i == x_{i+1}$ if $y_i == y_{i+1}$ adds nothing but complexity, x_i must always be less than x_{i+1}. The case $x_i > x_{i+1}$ for all i would also work as a mathematical function; but, supporting both the cases $x_i < x_{i+1}$ for all i and $x_i > x_{i+1}$ for all i would make the **ptwXYPoints** coding harder, is also not allowed.

```
#include <ptwXY.h>
#define nPairs 7

    double maleData[2 * (nPairs-1)] =   { 1871, 1212, 1883, 1215, 1889,
        51, 1895, 11, 1905,  9, 1915,  9 };
    double femaleData[2 * nPairs] =     { 1871, 1231, 1883, 1241, 1885,
        621, 1889, 229, 1895, 31, 1905, 23, 1915, 21 };
    ptwXYPoints *males, *females, *total;
    ptwXY_interpolation linlin = ptwXY_interpolationLinLin;
    nfu_status status;

    males = ptwXY_new2( NULL, linlin, nPairs, 4 );
    ptwXY_setXYData( NULL, males, nPairs - 1, maleData );

    females = ptwXY_new2( NULL, linlin, nPairs, 4 );
    ptwXY_setXYData( NULL, females, nPairs, femaleData );

    total = ptwXY_add_ptwXY( NULL, males, females, &status );

    printf( "\nMale population\n" );
    printf( "  Year | Count\n" );
    printf( " ------+------\n" );
    ptwXY_simplePrint( males, " %5.0f | %5.0f\n" );

    printf( "\nFemale population\n" );
    printf( "  Year | Count\n" );
    printf( " ------+------\n" );
    ptwXY_simplePrint( females, " %5.0f | %5.0f\n" );

    printf( "\nTotal population\n" );
    printf( "  Year | Count\n" );
    printf( " ------+------\n" );
    ptwXY_simplePrint( total, " %5.0f | %5.0f\n" );
```

Table 2: C code that shows how to create two ptwXY instances and add their data.

```
 Output from first  | Output from second  | Output from third
 ptwXY_simplePrint  | ptwXY_simplePrint   | ptwXY_simplePrint
--------------------+---------------------+-------------------
  Male population   |  Female population  |  Total population
    Year | Count    |    Year | Count     |    Year | Count
   ------+------    |   ------+------     |   ------+------
    1871 |  1212    |    1871 |  1231     |    1871 |  2443
    1883 |  1215    |    1883 |  1241     |    1883 |  2456
    1889 |    51    |    1885 |   621     |    1885 |  1448
    1895 |    11    |    1889 |   229     |    1889 |   280
    1905 |     9    |    1895 |    31     |    1895 |    42
    1915 |     9    |    1905 |    23     |    1905 |    32
                    |    1915 |    21     |    1915 |    30
```

Table 3: Output of the code of Table 2.

argument into the **ptwXYPoints** object's internal memory, deleting any data currently in the object. The third argument is the number of pairs of points in the fourth argument's data.

The function **ptwXY_add_ptwXY** takes a **ptwXYPoints** object as its second and third arguments, and returns a new **ptwXYPoints** object that is the sum. The summed object's x values are a union between the x value's of the operants. As can be seen from the example, this function interpolates to fill in missing data for either data set. That is, the male population was linear-linear interpolated to give 827 for the year 1885.

1.1 Important concepts

This section describes several important concepts and rules that the **ptwXY** model is build on.

1.1.1 Mutual domains

Most functions that have two or more **ptwXYPoints** instances as input (e.g., ptwXY_add_ptwXY, ptwXY_groupThreeFunctions) require that their domains be mutual. This section explains why mutual domains are needed and what a mutual domain is.

Consider the two point-wise linear-linear interpolable functions

```
f1 = (1,1), (9,3)
f2 = (1,2), (9,4).
```

where a point-wise function with n points is written as

$$(x_0, y_0),\ (x_1, y_1),\ (x_2, y_2),\ \ldots\ (x_{n-1}, y_{n-1})$$

and each point is the pair (x_i, y_i) with $x_i < x_{i+1}$. Each of these functions contains only two points. The first has domain $1 \leq x \leq 9$ with the y-value going from 1 to 3 and can be represented symbolically as $y = \mathrm{f1(x)} = (\mathrm{x} - 1)/4 + 1$ for the domain $1 \leq x \leq 9$. The second has the same domain with the y-value going from 2 to 4 and can be represented symbolically as $y = \mathrm{f2(x)} = (\mathrm{x} - 1)/4 + 2$. The rule that should be implemented for adding these two functions is clear and is $s(x) = (x - 1)/2 + 3 = \mathrm{f1(x)} + \mathrm{f2(x)}$ or in point-wise form

```
(1,3), (9,7) = f1 + f2
```

For the domain $1 \leq x \leq 9$, the point-wise linear-linear interpolable sum and the symbolic sum yield the same results. For example, both yield $s(3) = 4$. Figure 1 graphically shows f1, f2 and f1+f2.

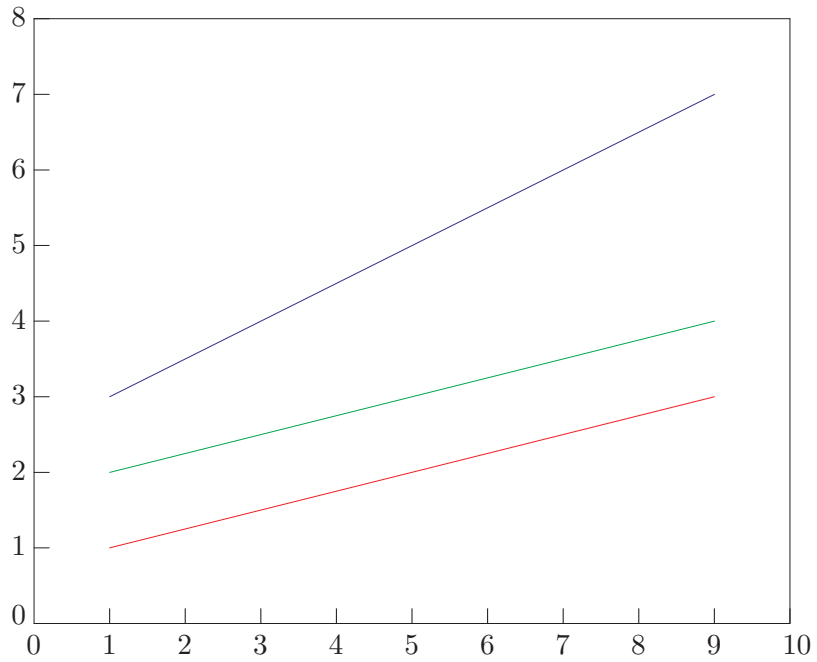

Figure 1: The red curve is f1, the green curve is f2 and the blue curve is f1 + f2.

Now consider the point-wise linear-linear interpolable function

```
f3 = (3,1), (7,3).
```

This function also contains only two points and has domain $3 \leq x \leq 7$ with a y-value going of 1 to 3. This function can be represented symbolically as $y = \text{f3(x)} = (\text{x} - 3)/2 + 1$. The rule that should be implemented for adding f1 and f3 is not obvious. For example, one could implement the rule which makes a union of the x-values in f1 and f3 (i.e., 1, 3, 7 and 9), interpolate each function onto these points using 0 where the function is not defined and then add the y-values. Let f1' and f3' be the functions f1 and f3 with the union points and the y-values filled in. In point-wise representation, f1' and f3' are

```
f1' = (1,1), (3,1.5), (7,2.5), (9,3)
f3' = (1,0), (3,1),   (7,3),   (9,0).
```

The sum resulting from this rule is then

```
(1,1), (3,2.5), (7,5.5), (9,3) = f1' + f3' = f1 + f3.
```

and is shown as the blue curve in Fgure 2. The blue curve is not vary appealing in part because for this addition rule the sum of f3 with f1 makes the assumption that $\text{f3(x)} = (\text{x} - 1)/2$ for $1 \leq x \leq 3$. But what is worse, this rule does not guarantee the associativity rule for addition. To see this, consider the three linear-linear point-wise functions

```
g1 = (1,0), (1,1),         (10,10)
g2 = (1,0),                (10,10)
g3 =                (2,1), (10,1).
```

Note that g1 and g2 represent the same function. The addition $(\text{g1} + \text{g2}) + \text{g3}$ is

```
(0,0), (1,2), (2,5), (10,21) = ( g1 + g2 ) + g3
```

while the addition $\text{g1} + (\text{g2} + \text{g3})$ is

```
(0,0), (1,2.5), (2,5), (10,21) = g1 + ( g2 + g3 ).
```

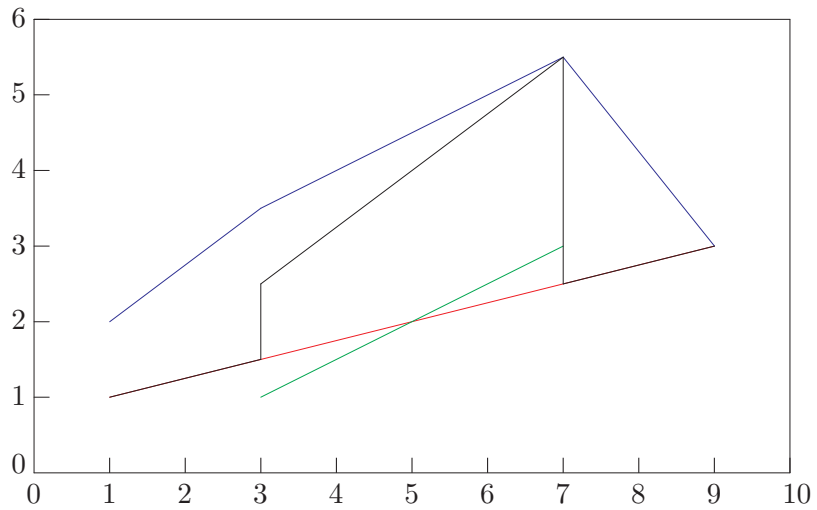

Figure 2: The red curve is f1, the green curve is f2 and the blue curve is f1 + f3 using the first rule and the black curve is f1 + f3 using the second rule.

Another rule one could implement would effectively add a point with $y = 0$ just below the first point of f3, one just above the last point of f3, one at $x = 1$ and one at $x = 9$ to yield an f3' as

```
f3' = (1,0), (2.99999,0), (3,1), (7,3), (7.00001,0), (9,0).
```

and the sum f1+f3 would then be

```
(1,1), (2.99999,1.4999975, (3,2.5), (7,5.5), (7.00001, 2.5000025), (9,3)
```

The sum resulting from this latter rule is shown as the black curve in Fgure 2. This rule looks much better and is. However, when designing the **ptwXYPoints** model, this rule was also rejected as it would require the **ptwXYPoints** library to know the appropriate distance below and above the end-points to add 0's.

The rule that the **ptwXYPoints** model implements is called "mutual domain". This rule states that the domains of the functions operated on must be the same with one exception. This exception will now be explained. Let h1 and h2 be two **ptwXYPoints** instances with the lower and upper domain limits of h1 being $x_{1,l}$ and $x_{1,u}$ and that of h2 being $x_{2,l}$ and $x_{2,u}$. If $x_{1,l} \neq x_{2,l}$ then the y-value for the greater lower-x-limit must be 0. For example, if $x_{1,l} > x_{2,l}$ then $\mathrm{h2}(\mathrm{x_{2,l}}) = 0$. If $x_{1,u} \neq x_{2,u}$ then the y-value for the lesser upper-x-limit must be 0. For example, if $x_{1,u} < x_{2,u}$ then $\mathrm{h1}(\mathrm{x_{1,u}}) = 0$. This rule works because the **ptwXYPoints** model assumes that if the y-value is 0 at the lower limit, then it is 0 for all x less than the lower limit. Likewise if the y-value is 0 at the upper limit, then it is 0 for all x greater than the upper limit.

If f4 and f5 have mutual domains, and f4 and f6 have mutual domains, the it is not guaranteed that f5 and f6 have mutual domains. As an example, let

```
f4 = (1,4), (8,4)
f5 = (3,0), (8,2)
f6 = (4,3), (6,0).
```

Then, f4 and f5 have mutual domains and so do f5 and f6. However, f4 and f6 do not have mutual domains. Because of this fact, the function **ptwXY_groupThreeFunctions** has to check the domains of **ptwXY1** to **ptwXY2**, **ptwXY1** to **ptwXY3** and **ptwXY2** to **ptwXY3**. Acutally, **ptwXY_groupThreeFunctions** first limits the domains of **ptwXY1**, **ptwXY2** and **ptwXY3** to that of **groupBoundaries** first.

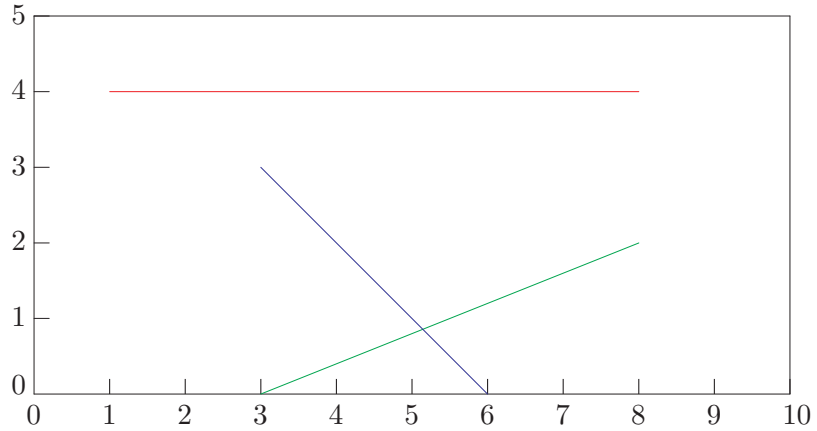

Figure 3: The red curve is f4, the green curve is f5 and the blue curve is f6. The domains of f4 and f5 are mutual as are the domains of f5 and f6. However, the domains of f4 and f6 are not mutual.

1.1.2 Infill

The addition of two linear functions yields another linear function. As example, the sum of f1(x) = s1 × x + y1 and f2(x) = s2 × x + y2 is

$$\text{f1(x)} + \text{f2(x)} = (\text{s1} + \text{s2}) \times \text{x} + \text{y1} + \text{y2} \quad . \tag{1}$$

Hence, when the function **ptwXY_add_ptwXY** adds two linear-linear pointwise functions, it only needs to make a union of the x-values of the two addends to maintain accuracy. However, the multiplication of f1(x) and f2(x) is not a linear function but a quadratic function. For example,

$$\text{f1(x)} \times \text{f2(x)} = \text{s1} \times \text{s2} \times \text{x}^2 + (\text{s1} \times \text{y2} + \text{s2} \times \text{y1}) \times \text{x} + \text{y1} \times \text{y2} \quad . \tag{2}$$

In an attempt to maintain accuracy, the function **ptwXY_mul2_ptwXY** may add additional points between the union points. For example, consider the following linear-linear point-wise functions f3 and f4,

```
f3 = (0,0), (1,1)
f4 = (0,1), (1,0)
```

which have the symbolic forms f3(x) = x and f4(x) = 1 − x over the domain $0 \leq x \leq 1$ and the symbolic product $x(1 - x)$. Making a union of the x-values and evaluating the product on the x-values yields

```
(0,0), (1,0) = f3 * f4
```

which is clearly inadequate. For this example, the only way to maintain the accuracy is to add points between $x = 0$ and $x = 1$. The adding of points in an attempt to maintain accuracy is called infilling and is done automatically by some **ptwXYPoints** functions including **ptwXY_mul2_ptwXY** but not by **ptwXY_mul_ptwXY**.

Infilling is done by bisecting (i.e., generating the point midway between) two consecutive points and asking if the accuracy of the operation (e.g., mutiplication) is satisfied. If the accuracy is satisfied, the midpoint is not added. However, if the accurcay is not satisfied, the midpoint is added then the segments on both side of the midpoint are tested.

In some cases, infilling can add a lot of points, more than one may like. Each **ptwXYPoints** instance has a member called **biSectionMax** to limit bisecting. The union function sets the **biSectionMax** of the returned **ptwXYPoints** instance, to the maximum of **biSectionMax** of its inputted **ptwXYPoints** instances. For each segment of the initial union at most **biSectionMax** bisections are performed. Table 4 contains a snippet of code which demonstrates the multiplication f3 and f4, without any error checking of course, for **biSectionMax** set to 0, 1, 2, and 3, and Figure 4 shows the output from this code.

```
int main( int argc, char **argc ) {

    double f3[4] = { 0., 0., 1., 1. }, f4[4] = { 0., 1., 1., 0. };
    ptwXYPoints *ptwXY3, *ptwXY4;
    ptwXY_interpolation linlin = ptwXY_interpolationLinLin;

    ptwXY3 = ptwXY_create2( NULL, linlin, 10, 10, 2, f3, 0 );
    ptwXY4 = ptwXY_create2( NULL, linlin, 10, 10, 2, f4, 0 );

    doProduct( ptwXY3, ptwXY4, 0 );
    doProduct( ptwXY3, ptwXY4, 1 );
    doProduct( ptwXY3, ptwXY4, 2 );
    doProduct( ptwXY3, ptwXY4, 3 );

}

void doProduct( ptwXYPoints *ptwXY3, ptwXYPoints *ptwXY4, double biSection ) {

    ptwXYPoints *product;

    ptwXY_setBiSectionMax( ptwXY3, biSection );
    product = ptwXY_mul2_ptwXY( NULL, ptwXY3, ptwXY4 );
}
```

Table 4: This table show a snippet of the code used to generate the curves in Figure 4.

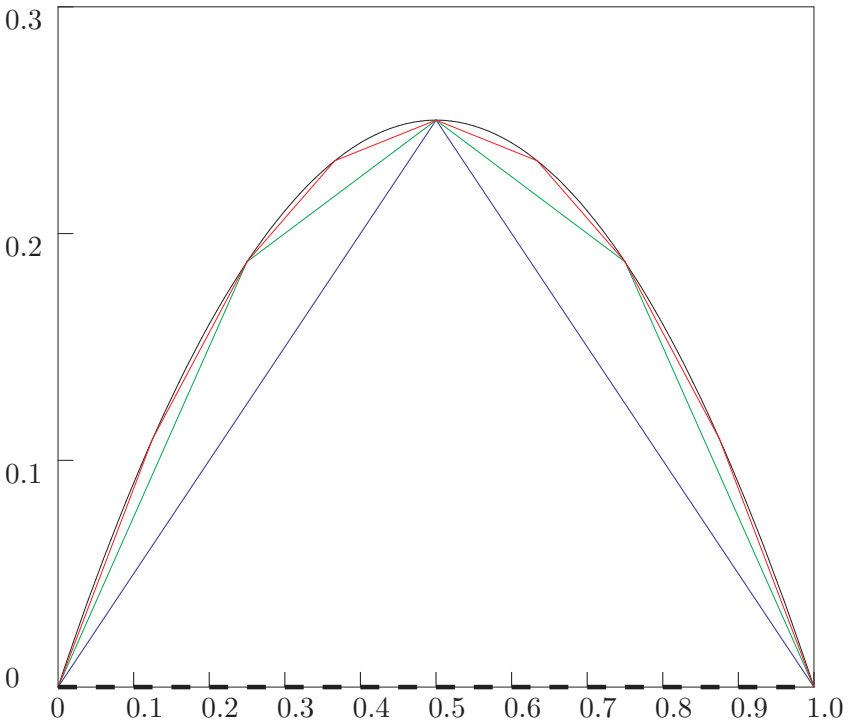

Figure 4: The solid black curve is the function $x(1-x)$. The blue, grean and red curves are products of f3 $\times$ f4 from ptwXY_mul2_ptwXY for biSectionMax of 1, 2 and 3 respectively. The solid black curve and the red curve are nearly identical. The dashed thicker black line at the bottom of the plot is the product for biSectionMax = 0.

If infilling is needed, the **biSectionMax** member of the returned **ptwXYPoints** instance is reduced by $\ln(l_f/l_u)/ln(2)$ where l_u is the lenght of the union and l_f is the final length after all bisections[2]. An **ptwXYPoints** instance's **biSectionMax** can be set using **ptwXY_setBiSectionMax** and got using **ptwXY_setBiSectionMax**. A **ptwXYPoints**'s **biSectionMax** is limited to the range 0 to **ptwXY_maxBiSectionMax**.

1.1.3 accuracy

FIXME

For the examples in Table 4, the infill is halted when **biSectionMax** is met. This is one of two ways that infill is halted. The other way is when an infill point agrees to the exact solution to a relative accuracy.

1.1.4 Safe divide

FIXME

1.1.5 Future change to biSectionMax and accuracy.

A future release of this library will change the way **biSectionMax** and **accuracy** are handled. Instead of being associated with each **ptwXYPoints** instance, they will be associated with the operators (e.g., multiplication and division) that require them. For example, the members **accuracy** and **biSectionMax** will be removed from the **ptwXYPoints** struct, and the declaration of **ptwXY_mul2_ptwXY** will change from

```
ptwXYPoints *ptwXY_mul2_ptwXY( statusMessageReporting *smr, ptwXYPoints *ptwXY1,
                               ptwXYPoints *ptwXY2 );
```

to something like

```
ptwXYPoints *ptwXY_mul_ptwXY( statusMessageReporting *smr, ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2, double accuracy,
                              int biSectionMax );
```

2 Name convention

This section defines some of the names used in this document.

point: A point is a pair of (x, y) values.

[2]This reduction is derived by setting $2^z = l_f/l_u$ and solving for z.

cache and array: In this document there is a distinction between a cache and an array of a cache. A cache is allocated memory used to store data. An array of a cache is a region of a cache containing valid data. As example, for the primary cache described in section 3, points are added to the cache as needed. The current points in the cache constitute the array of that cache.

3 Two-cached, Dynamic-Growth Data Array

Built into the **ptwXY** model is the ability to insert a point at any x. The supporting functions will automatically increase the size of an internal data cache if needed to accommodate a new x value. However, to make adding and deleting points potentially more efficient, the **ptwXY** model has two data caches, dubbed primary and secondary. In the primary cache, data are stored in a C array in ascending order which allows for quick accessing. However, inserting a new x value at any place other than the end of the array can be slow as it requires moving all x values that are greater than the new value up one element in the array. To over come this, a newly added x value is inserted into the secondary cache if: 1) the value cannot be inserted after the last element of the primary array[3] and 2) space is available in the secondary cache. The secondary cache is a static, linked list. Here, static means that the elements of the linked list are allocated during setup so there is no overhead associated with allocating or freeing elements of the linked list later. Typically, re-allocation of the memory of the primary cache is only required when a new x value cannot be inserted into either cache.

There are four parameters, two for each cache, that describe the current state of the caches. Each cache has a size which is the amount, in units of an element of that cache, of memory allocated for the cache and a length which is the amount, in units of an element of that cache, of the cache that is currently used (i.e., the size of the array of that cache).

The initial size of the two caches is set either through the function **ptwXY_new** or **ptwXY_setup** via their **primarySize** and **secondarySize** arguments. The size of the primary and secondary caches can be directly altered after they have been created via the functions **ptwXY_reallocatePoints** and **ptwXY_reallocateOverflowPoints** respectively. In general these last two functions should not be called by the users unless they know that the a cache is woefully too small.

The function **pwtXY_coalescePoints** can be called to transfer all secondary points into the primary cache.

4 ptwXYPoints's C structs, macros and enums

The following definitions are defined in the C header file "ptwXY.h".

[3] A value can only be inserted after the last element of the primary array if its x is greater than the current maximum x value and there is room in the primary cache.

4.1 ptwXYPoints

The **ptwXYPoints** type is defined as:

```
typedef
    struct ptwXYPoints_s {
        nfu_status status;
        ptwXY_interpolation interpolation;
        char const *interpolationString;
        int userFlag;
        double biSectionMax;
        double accuracy;
        double minFractional_dx;
        int64_t length;
        int64_t allocatedSize;
        int64_t overflowLength;
        int64_t overflowAllocatedSize;
        int64_t mallocFailedSize;
        ptwXYOverflowPoint overflowHeader;
        ptwXYPoint *points;
        ptwXYOverflowPoint *overflowPoints;
    } ptwXYPoints;
```

The **ptwXYPoint** type is defined as:

```
typedef
    struct ptwXYPoint_s {
        double x, y;
    } ptwXYPoint;
```

The type **ptwXYOverflowPoint** will not be described here as it is not used as an argument in any function and its members in **ptwXYPoints** should not be accessed by user codes.

4.2 C macros

FIXME

This section lists some of the C macros defined in "ptwXY.h".

4.3 Interpolation

For an x value that is within the domain of a **ptwXYPoints** object but not one of its points, the **ptwXYPoints** functions interpolate, as instructed by the member **interpolation**, to obtain the y value. Interpolation types are defined using the type **ptwXY_interpolation** which is defined as:

```
typedef enum ptwXY_interpolation_e {
    ptwXY_interpolationLinLin,  /* x and y linear. */
    ptwXY_interpolationLinLog,  /* x linear and y logarithmic. */
    ptwXY_interpolationLogLin,  /* x logarithmic and y linear. */
    ptwXY_interpolationLogLog,  /* x and y logarithmic. */
    ptwXY_interpolationFlat,    /* see below */
    ptwXY_interpolationOther    /* see below */
} ptwXY_interpolation;
```

FIXME

The latter two interpolation types have many restrictions. For **ptwXY_interpolationFlat** the y for $x_i \leq x < x_{i+1}$ is y_i. This type is good for storing histogram type data. Many of the functions in the **ptwXY** library cannot handle the flat interpolation and return the error **nfu_invalidInterpolation** via their nfu_status argument. The interpolation type **ptwXY_interpolationOther** allows the use of ptwXY storage type for data that does not fit into one the other defined interpolation types. Most functions cannot handle the other interpolation and also return the error **nfu_invalidInterpolation**.

4.4 Data types

Currently, the **ptwXY** model only supports a point as an (x, y) pair.

4.5 Miscellaneous types

The function **ptwXY_getPointsAroundX** is used by other functions to determine where an x value fits into a **ptwXYPoints** object. The return value of this function is of type **ptwXY_lessEqualGreaterX** which is defined as:

```
typedef enum ptwXY_lessEqualGreaterX_e {
    ptwXY_lessEqualGreaterX_empty,          /* The object has no points. */
    ptwXY_lessEqualGreaterX_lessThan,       /* The x < xMin. */
    ptwXY_lessEqualGreaterX_equal,          /* x = x_i. */
    ptwXY_lessEqualGreaterX_between,        /* x_i < x < x_i+1. */
    ptwXY_lessEqualGreaterX_greater,        /* The x > xMax. */
    ptwXY_lessEqualGreaterX_Error           /* Something wrong with input. */
} ptwXY_lessEqualGreaterX;
```

Here, xMin and xMax are the minimum and maximum x values of the **ptwXYPoints** object, and x_i and x_{i+1} are the $(i-1)^{th}$ and i^{th} x values of the **ptwXYPoints** object respectively.

5 Routines

5.1 Core

This section decribes all the functions in the file "ptwXY_core.c".

5.1.1 ptwXY_new

This function allocates memory for a new **ptwXYPoints** object and initializes it by calling **ptwXY_initialize**.

C declaration:

```
ptwXYPoints *ptwXY_new( statusMessageReporting *smr,
                        ptwXY_interpolation interpolation,
                        char const *interpolationString,
                        double biSectionMax,
                        double accuracy,
                        int64_t primarySize,
                        int64_t secondarySize,
                        int userFlag );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`interpolation:`	The type of interpolation to use.
`interpolationString:`	The string representation of interpolation.
`biSectionMax:`	The maximum disection allowed.
`accuracy:`	The interpolation accuracy of the data.
`primarySize:`	Initial size of the primary cache.
`secondarySize:`	Initial size of the secondary cache.
`userFlag:`	A user defined integer value not used by any ptwXY function.

The value of **interpolation** and **interpolationString** must agree; otherwise, an error will be reported. For all but **ptwXY_interpolationOther** these must match the string returned by the function **ptwXY_interpolationToString**. For **ptwXY_interpolationOther**, the string can be anything that is not one of the standard interpolation string.

If this function fails, NULL is returned.

5.1.2 ptwXY_new2

This function calls **ptwXY_new** with the biSectionMax = 12, accuracy = 1e-3, userFlag = 0 and with interpolationString set to the proper string. This function should not be used for the interpolation value of **ptwXY_interpolationOther**.

C declaration:

```
ptwXYPoints *ptwXY_new( statusMessageReporting *smr,
                        ptwXY_interpolation interpolation,
                        int64_t primarySize,
                        int64_t secondarySize );
```

smr:	The **statusMessageReporting** instance to record errors.
interpolation:	The type of interpolation to use.
primarySize:	Initial size of the primary cache.
secondarySize:	Initial size of the secondary cache.

5.1.3 ptwXY_initialize

This function initializes a **ptwXYPoints** object and must be called for a **ptwXYPoints** object before that object can be used by any other function in this package.

C declaration:

```
fnu_status ptwXY_initialize( statusMessageReporting *smr,
                             ptwXYPoints *ptwXY,
                             ptwXY_interpolation interpolation,
                             char const *interpolationString,
                             double biSectionMax,
                             double accuracy,
                             int64_t primarySize,
                             int64_t secondarySize,
                             int userFlag );
```

smr:	The **statusMessageReporting** instance to record errors.
ptwXY:	A pointer to a **ptwXYPoints** object to initialize.
interpolation:	The type of interpolation to use.
interpolationString:	The string representation of interpolation.
biSectionMax:	The maximum disection allowed.
accuracy:	The interpolation accuracy of the data.
primarySize:	Initial size of the primary cache.
secondarySize:	Initial size of the secondary cache.
userFlag:	An user defined integer value not used by any ptwXY function.

The primary and secondary caches are allocated with functions **ptwXY_reallocatePoints** and **ptwXY_reallocateOverflowPoints** respectively.

5.1.4 ptwXY_create

This functions combines **ptwXY_new** and **ptwXY_setXYData**.

C declaration:

```
ptwXYPoints *ptwXY_create( statusMessageReporting *smr,
                          ptwXY_interpolation interpolation,
                          char const *interpolationString,
                          double biSectionMax,
                          double accuracy,
                          int64_t primarySize,
                          int64_t secondarySize,
                          int64_t length,
                          double *xy,
                          int userFlag );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`interpolation:`	The type of interpolation to use.
`interpolationString:`	The string representation of interpolation.
`biSectionMax:`	The maximum disection allowed.
`accuracy:`	The interpolation accuracy of the data.
`primarySize:`	Initial size of the primary cache.
`secondarySize:`	Initial size of the secondary cache.
`length:`	The number of points in xy.
`xy:`	The new points given as $x_0, y_0, x_1, y_1, \ \ldots, \ x_n, y_n$ where n = length - 1.
`userFlag:`	An user defined integer value not used by any ptwXY function.

If this function fails, NULL is returned.

5.1.5 ptwXY_create2

This function calls **ptwXY_create** with the biSectionMax = 12, accuracy = 1e-3, userFlag = 0 and with interpolationString set to the proper string. This function should not be used for the interpolation value of **ptwXY_interpolationOther**.

C declaration:

```
ptwXYPoints *ptwXY_new( statusMessageReporting *smr,
                        ptwXY_interpolation interpolation,
                        int64_t primarySize,
                        int64_t secondarySize,
                        int64_t length,
                        double *xy );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`interpolation:`	The type of interpolation to use.
`primarySize:`	Initial size of the primary cache.

`secondarySize:`	Initial size of the secondary cache.
`length:`	The number of points in xy.
`xy:`	The new points given as $x_0, y_0, x_1, y_1, \ldots, x_n, y_n$ where n = length - 1.

5.1.6 ptwXY_createFrom_Xs_Ys

This functions is like **ptwXY_create** except the x and y data are given in separate arrays.

C declaration:

```
ptwXYPoints *ptwXY_createFrom_Xs_Ys( statusMessageReporting *smr,
                                     ptwXY_interpolation interpolation,
                                     char const *interpolationString,
                                     double biSectionMax,
                                     double accuracy,
                                     int64_t primarySize,
                                     int64_t secondarySize,
                                     int64_t length,
                                     double *Xs,
                                     double *Ys,
                                     int userFlag );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`interpolation:`	The type of interpolation to use.
`interpolationString:`	The string representation of interpolation.
`biSectionMax:`	The maximum disection allowed.
`accuracy:`	The interpolation accuracy of the data.
`primarySize:`	Initial size of the primary cache.
`secondarySize:`	Initial size of the secondary cache.
`length:`	The number of points in xy.
`Xs:`	The new x points given as $x_0, x_1, \ldots, x_n$ where n = length - 1.
`Ys:`	The new y points given as $y_0, y_1, \ldots, y_n$ where n = length - 1.
`userFlag:`	An user defined integer value not used by any ptwXY function.

If this function fails, NULL is returned.

5.1.7 ptwXY_createFrom_Xs_Ys2

This function calls **ptwXY_createFrom_Xs_Ys** with the biSectionMax = 12, accuracy = 1e-3, userFlag = 0 and with interpolationString set to the proper string. This function should not be used for the interpolation value of **ptwXY_interpolationOther**.

C declaration:

```
ptwXYPoints *ptwXY_Xs_Ys2( statusMessageReporting *smr,
                           ptwXY_interpolation interpolation,
                           int64_t primarySize,
                           int64_t secondarySize,
                           int64_t length,
                           double *Xs,
                           double *Ys );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`interpolation:`	The type of interpolation to use.
`primarySize:`	Initial size of the primary cache.
`secondarySize:`	Initial size of the secondary cache.
`length:`	The number of points in xy.
`Xs:`	The new x points given as $x_0, x_1, \ldots, x_n$ where n = length - 1.
`Ys:`	The new y points given as $y_0, y_1, \ldots, y_n$ where n = length - 1.

5.1.8 ptwXY_copy

This function clears the points in **dest** and then copies the points from **src** into **dest**. It also copies all other data like the interplation flag. The **src** object is not modified.

C declaration:

```
fnu_status ptwXY_copy( statusMessageReporting *smr,
                       ptwXYPoints *dest,
                       ptwXYPoints *src );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`dest:`	A pointer to the destination **ptwXYPoints** object.
`src:`	A pointer to the source **ptwXYPoints** object.

5.1.9 ptwXY_copyPointsOnly

This function clears the points in **dest** and then copies the points from **src** into **dest**. Unlike **ptwXY_copy**, this function does not copy any of the other data (e.g., the interpolation flag). The **src** object is not modified.

C declaration:

```
fnu_status ptwXY_copyPointsOnly( statusMessageReporting *smr,
                                 ptwXYPoints *dest,
                                 ptwXYPoints *src );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`dest:`	A pointer to the destination **ptwXYPoints** object.

`src:` A pointer to the source **ptwXYPoints** object.

5.1.10 ptwXY_clone

This function creates a new **ptwXYPoints** object and sets its points to the points in its first argument. This function calls **ptwXY_simpleCoalescePoints**. Also see **ptwXY_clone2**.

C declaration:

```
ptwXYPoints *ptwXY_clone( statusMessageReporting *smr,
                          ptwXYPoints *ptwXY );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

If an error occurs, NULL is returned.

5.1.11 ptwXY_clone2

This function is like **ptwXY_clone** but does not call **ptwXY_simpleCoalescePoints**.

C declaration:

```
ptwXYPoints *ptwXY_clone2( statusMessageReporting *smr,
                           ptwXYPoints *ptwXY );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

If an error occurs, NULL is returned.

5.1.12 ptwXY_cloneToInterpolation

This function calls **ptwXY_clone** and set the interpolation of the returned **ptwXYPoints** instance to **interpolation** without adding any points.

C declaration:

```
ptwXYPoints *ptwXY_cloneToInterpolation( statusMessageReporting *smr,
                                         ptwXYPoints *ptwXY,
                                         ptwXY_interpolation interpolation );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`interpolation:` The interpolation of the return **ptwXYPoints** instance.

If an error occurs, NULL is returned.

5.1.13 ptwXY_slice

This function creates a new **ptwXYPoints** object and sets its points to the points from index **index1** inclusive to **index2** exclusive of **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_slice( statusMessageReporting *smr,
                          ptwXYPoints *ptwXY,
                          int64_t index1,
                          int64_t index2,
                          int64_t secondarySize
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`index1:`	The lower index.
`index2:`	The upper index.
`secondarySize:`	Initial size of the secondary cahce.

If an error occurs, NULL is returned.

5.1.14 ptwXY_domainSlice

This function creates a new **ptwXYPoints** object and sets its points to the points from the points between the domain **domainMin** and **domainMax** of **ptwXY**. If **fill** is true, points at **domainMin** and **domainMax** are added if not in the inputted **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_domainSlice( statusMessageReporting *smr,
                                ptwXYPoints *ptwXY,
                                double domainMin,
                                double doaminMax,
                                int64_t secondarySize,
                                int fill
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`domainMin:`	The lower domain value.
`domainMax:`	The upper domain value.
`secondarySize:`	Initial size of the secondary cahce.
`fill:`	If not 0, points at domainMin and domainMax are added if needed.

If an error occurs, NULL is returned.

5.1.15 ptwXY_domainMinSlice

This function creates a new **ptwXYPoints** object and sets its points to the points from the points between the domain **domainMin** to the end of **ptwXY**. If **fill** is true, point at **domainMin** is added if not in the inputted **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_domainMinSlice( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   double domainMin,
                                   int64_t secondarySize,
                                   int fill );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`domainMin:`	The lower domain value.
`secondarySize:`	Initial size of the secondary cahce.
`fill:`	If not 0, a point at domainMin is added if needed.

If an error occurs, NULL is returned.

5.1.16 ptwXY_domainMaxSlice

This function creates a new **ptwXYPoints** object and sets its points to the points from the points between the domain of the beginning of **ptwXY** to **domainMax**. If **fill** is true, point at **domainMax** is added if not in the inputted **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_domainMaxSlice( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   double domainMax,
                                   int64_t secondarySize,
                                   int fill );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`domainMax:`	The upper domain value.
`secondarySize:`	Initial size of the secondary cahce.
`fill:`	If not 0, a point at domainMax is added if needed.

If an error occurs, NULL is returned.

5.1.17 ptwXY_getInterpolation

This function returns the value of **ptwXY**'s interpolation member.

C declaration:

```
ptwXY_interpolation ptwXY_getInterpolation( ptwXYPoints *ptwXY );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.18 ptwXY_getInterpolationString

This function returns a pointer to **ptwXY**'s interpolationString member.

C declaration:

```
char const *ptwXY_getInterpolationString( ptwXYPoints *ptwXY );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.19 ptwXY_getStatus

This function returns the value of **ptwXY**'s status member.

C declaration:

```
nfu_status ptwXY_getStatus( ptwXYPoints *ptwXY );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.20 ptwXY_getUserFlag

This function returns the value of **ptwXY**'s userFlag member.

C declaration:

```
int ptwXY_getUserFlag( ptwXYPoints *ptwXY );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.21 ptwXY_setUserFlag

This function sets the value of the **ptwXY**'s userFlag member to userFlag.

C declaration:

```
void ptwXY_setUserFlag( ptwXYPoints *ptwXY,
                        int userFlag);
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`userFlag:` The value to set ptwXY's userFlag to.

5.1.22 ptwXY_getAccuracy

This function returns the value of **ptwXY**'s accuracy member.

C declaration:

```
double ptwXY_getAccuracy( ptwXYPoints *ptwXY );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.23 ptwXY_setAccuracy

This function sets the value of the **ptwXY**'s accuracy member to accuracy.

C declaration:

```
double ptwXY_setAccuracy( ptwXYPoints *ptwXY,
                          double accuracy );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`accuracy:` The value to set ptwXY's accuracy to.

Becuase the range of accuracy is limited, the actual value set may be different then the argument accuracy. The actual value set in ptwXY is returned.

5.1.24 ptwXY_getBiSectionMax

This function returns the value of **ptwXY**'s biSectionMax member.

C declaration:

```
double ptwXY_getBiSectionMax( ptwXYPoints *ptwXY );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.25 ptwXY_setBiSectionMax

This function sets the value of the **ptwXY**'s biSectionMax member to biSectionMax.

C declaration:

```
double ptwXY_setBiSectionMax( ptwXYPoints *ptwXY,
                              double biSectionMax );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`biSectionMax:` The value to set ptwXY's biSectionMax to.

Becuase the range of biSectionMax is limited, the actual value set may be different then the argument biSectionMax. The actual value set in ptwXY is returned.

5.1.26 ptwXY_reallocatePoints

This function changes the size of the primary cache.

C declaration:

```
fnu_status ptwXY_reallocatePoints( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   int64_t size,
                                   int forceSmallerResize );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`size:`	The desired size of the primary cache.
`forceSmallerResize:`	If true (i.e., non-zero) and size is smaller than the current size, the primary cache is resized. Otherwise, the primary cache is only reduced if the inputted size is significantly smaller than the current size.

The actual memory allocated is the maximum of `size`, the current length of the primary cache and **ptwXY_minimumSize**.

5.1.27 ptwXY_reallocateOverflowPoints

This function changes the size of the secondary cache.

C declaration:

```
fnu_status ptwXY_reallocateOverflowPoints( statusMessageReporting *smr,
                                           ptwXYPoints *ptwXY,
                                           int64_t size );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`size:`	The desired size of the secondary cache.

The actual memory allocated is the maximum of `size` and **ptwXY_minimumOverflowSize**. The function **ptwXY_coalescePoints** is called if the current length of the secondary cache is greater than the inputted size.

5.1.28 ptwXY_coalescePoints

This function adds the points from the secondary cache to the primary cache and then removes the points from the secondary cache. If the argument **newPoint** is not-NULL it is also added to the primary cache.

C declaration:

```
fnu_status ptwXY_coalescePoints( statusMessageReporting *smr,
                                 ptwXYPoints *ptwXY,
                                 int64_t size,
                                 ptwXYPointsPoint *newPoint,
                                 int forceSmallerResize );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`size:`	The desired size of the primary cache.
`newPoint:`	If not NULL, an additional point to add.
`forceSmallerResize:`	If true (i.e. non-zero) and size is smaller than the current size, the primary cache is resized. Otherwise, the primary cache is only reduced if the new size is significantly smaller than the current size.

The actual memory allocated is the maximum of `size`, the new length of the **ptwXY** object and **ptwXY_minimumSize**.

5.1.29 ptwXY_simpleCoalescePoints

This function is a simple wrapper for **ptwXY_coalescePoints** when only coalescing of the existing points is needed.

C declaration:

```
fnu_status ptwXY_simpleCoalescePoints( statusMessageReporting *smr,
                                       ptwXYPoints *ptwXY );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.

5.1.30 ptwXY_clear

This function removes all points from a **ptwXYPoints** object but does not free any allocated memory. Upon return, the length of the **ptwXYPoints** object is zero.

C declaration:

```
fnu_status ptwXY_clear( statusMessageReporting *smr,
                        ptwXYPoints *ptwXY );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.

5.1.31 ptwXY_release

This function frees all the internal memory allocated for a **ptwXYPoints** object.

C declaration:

```
fnu_status ptwXY_release( statusMessageReporting *smr,
                          ptwXYPoints *ptwXY );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.32 ptwXY_free

This function calls **ptwXY_release** and then calls free on **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_free( ptwXYPoints *ptwXY );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.

Any **ptwXYPoints** object allocated using **ptwXY_new** will be freed calling **ptwXY_free**. Once this function is called, the **ptwXYPoints** object should never be used. The return value is always NULL.

5.1.33 ptwXY_length

This function returns the length (i.e., number of points in the primary and secondary caches) for a **ptwXY** object.

C declaration:

```
int64_t ptwXY_length( statusMessageReporting *smr,
                      ptwXYPoints *ptwXY );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.34 ptwXY_getNonOverflowLength

This function returns the length of the primary caches (note, this is not its size).

C declaration:

```
int64_t ptwXY_getNonOverflowLength( statusMessageReporting *smr,
                                    ptwXYPoints *ptwXY );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

5.1.35 ptwXY_setXYData

This function replaces the current points in a **ptwXY** object with a new set of points.

C declaration:

```
fnu_status ptwXY_setXYData( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY,
                            int64_t length,
                            double *xy );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`length:` The number of points in xy.
`xy:` The new points given as $x_0, y_0, x_1, y_1, \ldots, x_n, y_n$ where n = length - 1.

5.1.36 ptwXY_setXYDataFromXsAndYs

This functions is like **ptwXY_setXYData** except the x and y data are given in separate arrays.

C declaration:

```
fnu_status ptwXY_setXYDataFromXsAndYs( statusMessageReporting *smr,
                                       ptwXYPoints *ptwXY,
                                       int64_t length,
                                       double *Xs,
                                       double *Ys );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`length:` The number of points in xy.
`Xs:` The new x points given as $x_0, x_1, \ldots, x_n$ where n = length - 1.
`Ys:` The new y points given as $y_0, y_1, \ldots, y_n$ where n = length - 1.

5.1.37 ptwXY_deletePoints

This function removes all the points from index **i1** inclusive to index **i2** exclusive. Indexing is 0 based.

C declaration:

```
fnu_status ptwXY_deletePoints( statusMessageReporting *smr,
                               ptwXYPoints *ptwXY,
                               int64_t i1,
                               int64_t i2 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`i1:` The lower index.
`i2:` The upper index.

As example, if an **ptwXY** object contains the points (1.2, 4), (1.3, 5), (1.6, 6), (1.9, 3) (2.0, 6), (2.1, 4) and (2.3, 1). Then calling **ptwXY_deletePoints** with i1 = 2 and i2 = 4 removes the points (1.6, 6) and (1.9, 3). The indices i1 and i2 must satisfy the relationship ($0 \leq$ i1 $\leq$ i2 $\leq n$) where n is the length of the **ptwXY** object; otherwise, no modification is done to the **ptwXY** object and the error **nfu_badIndex** is returned.

5.1.38 ptwXY_getLowerIndexBoundingX

This function sets the index such that the xs[index] <= x < xs[index] where xs is the list of x-values for **ptwXY**. If x is outside the domain of xs, index is set to -1. If x is the upper domain, it is length - 1 (i.e., for the last two points, the condition is xs[index] <= x <= xs[index]).

C declaration:

```
fnu_status ptwXY_getLowerIndexBoundingX( statusMessageReporting *smr,
                                         ptwXYPoints *ptwXY,
                                         double x,
                                         int64_t *index );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The domain value.
`index:` The set index.

5.1.39 ptwXY_getPointAtIndex

This function checks that the index argument is valid, and if it is, this function returns the result of **ptwXY_getPointAtIndex_Unsafely**. Otherwise, NULL is returned.

C declaration:

```
ptwXYPoint *ptwXY_getPointAtIndex( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   int64_t index );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`index:` The index of the point to return.

5.1.40 ptwXY_getPointAtIndex_Unsafely

This function returns the point at index. This function does not check if index is valid and thus is not intended for general use. Instead, see **ptwXY_getPointAtIndex** for a general use version of this function.

C declaration: — **This function is not intended for general use.** —

```
ptwXYPoint *ptwXY_getPointAtIndex_Unsafely( ptwXYPoints *ptwXY,
                                            int64_t index );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`index:` The index of the point to return.

5.1.41 ptwXY_getXYPairAtIndex

This function calls **ptwXY_getPointAtIndex** and if the index is valid it returns the point's x and y values via the arguments *x and *y. Otherwise, *x and *y are unaltered and an error signal is returned.

C declaration:

```
nfu_status ptwXY_getPairAtIndex( statusMessageReporting *smr,
                                 ptwXYPoints *ptwXY,
                                 int64_t index,
                                 double *x,
                                 double *y );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`index:` The index of the point to return.
`*x:` The point's x value is returned in this argument.
`*y:` The point's y value is returned in this argument.

5.1.42 ptwXY_getPointsAroundX

This function sets the **lessThanEqualXPoint** and **greaterThanXPoint** arguments to the two points that bound the point x.

C declaration: — **This function is not intended for general use.** —

```
ptwXY_lessEqualGreaterX ptwXY_getPointsAroundX( statusMessageReporting *smr,
                                                ptwXYPoints *ptwXY,
                                                double x ),
                                                ptwXYOverflowPoint
*lessThanEqualXPoint,
                                                ptwXYOverflowPoint
*greaterThanXPoint );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The x value.

`lessThanEqualXPoint:` The lower bounding point.
`greaterThanXPoint:` The upper bounding point.

If the **ptwXY** object is empty then the return value is **ptwXY_lessEqualGreaterX_empty**. If x is less than domainMin, then **ptwXY_lessEqualGreaterX_lessThan** is return. If x is greater than doaminMax, then **ptwXY_lessEqualGreaterX_greaterThan** is return. If x corresponds to a point in the **ptwXY** object then **ptwXY_lessEqualGreaterX_equal** is returned. Otherwise, **ptwXY_lessEqualGreaterX_between** is returned.

5.1.43 ptwXY_getPointsAroundX_closeIsEqual

This function is like **ptwXY_getPointsAroundX** except that when **eps** is greater than 0., it will set **closeIsEqual** to a non-zero value if a point is with relative **eps** of **x**. ***closePoint** is set to the which of **lessThanEqualXPoint** or **greaterThanXPoint** is closer. If **x** is below ***closePoint**, then closeIsEqual is set to -1. If it is above then closeIsEqual is set to 1. Otherwise, closeIsEqual is set to 0.

C declaration: — This function is not intended for general use. —

```
ptwXY_lessEqualGreaterX ptwXY_getPointsAroundX_closeIsEqual(
statusMessageReporting *smr,
                                                        ptwXYPoints
*ptwXY,
                                                        double x ),

ptwXYOverflowPoint *lessThanEqualXPoint,

ptwXYOverflowPoint *greaterThanXPoint ),
                                                        double eps,
                                                        int
*closeIsEqual,
                                                        ptwXYPoint
**closePoint );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The x value.
`lessThanEqualXPoint:` The lower bounding point.
`greaterThanXPoint:` The upper bounding point.
`eps:`
`closeIsEqual:`
`closePoint:`

5.1.44 ptwXY_getValueAtX

This function gets the y value at x, interpolating if necessary.

C declaration:

```
fnu_status ptwXY_getValueAtX( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY,
                              double x,
                              double *y );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The x value.
`y:` Upon return, contains the y value.

If the x value is outside the domain of the **ptwXY** object, y is set to zero and the returned value is **nfu_XOutsideDomain**.

5.1.45 ptwXY_setValueAtX

This function sets the point at x to y, if x does not corresponds to a point in the **ptwXY** object then a new point is added.

C declaration:

```
fnu_status ptwXY_setValueAtX( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY,
                              double x,
                              double y );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The x value.
`y:` The y value.

5.1.46 ptwXY_setValueAtX_overrideIfClose

FIXME
This function

C declaration:

```
fnu_status ptwXY_setValueAtX_overrideIfClose( statusMessageReporting *smr,
                                              ptwXYPoints *ptwXY,
                                              double x,
                                              double y,
                                              double eps,
                                              int override );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The x value.
`y:` The y value.
`eps:`
`override:`

5.1.47 ptwXY_mergeFromXsAndYs

This function calls **ptwXY_mergeFrom** to add the points give by **xs** and **ys** into **ptwXY**.

C declaration:

```
fnu_status ptwXY_mergeFromXsAndYs( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   int64_t length
                                   double *xs,
                                   double *ys );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`length:` The number of values in xs.
`xs:` The x-values to merge.
`ys:` The y-values to merge.

5.1.48 ptwXY_mergeFromXYs

This function calls **ptwXY_mergeFrom** to add the points give by **xs** and **ys** into **ptwXY**.

C declaration:

```
fnu_status ptwXY_mergeFromXYs( statusMessageReporting *smr,
                               ptwXYPoints *ptwXY,
                               int64_t length
                               double *xys );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

`length:` The number of values in xs.
`xs:` The (x,y) pairs to merge.

5.1.49 ptwXY_mergeFrom

This function merges the points gives by **xs** and **ys** into **ptwXY**.

C declaration:

```
fnu_status ptwXY_mergeFrom( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY,
                            int incY
                            int64_t length
                            double *xs,
                            double *ys );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`incY:` Not used.
`length:` The number of values in xs.
`xs:` The x-values to merge.
`ys:` The y-values to merge.

5.1.50 ptwXY_appendXY

This function appends the point **(x,y)** to the end of **ptwXY**. Note, **x** must be greater than domainMax.

C declaration:

```
fnu_status ptwXY_appendXY( statusMessageReporting *smr,
                           ptwXYPoints *ptwXY,
                           double x,
                           double y );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The x-value to append.
`y:` The y-value to append.

5.1.51 ptwXY_setXYPairAtIndex

This function sets the x and y values at index.

C declaration:

```
fnu_status ptwXY_setXYPairAtIndex( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   int64_t index
                                   double x,
                                   double y );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`index:`	The index of the point to set.
`x:`	The x value.
`y:`	The y value.

If index is invalid, **nfu_badIndex** is returned. If the x value is not valid for index (i.e. $x \leq x_{\text{index}-1}$ or $x \geq x_{\text{index}+1}$) then **nfu_badIndexForX** is return.

5.1.52 ptwXY_getSlopeAtX

This function calculates the slope at the point x assuming linear-linear interpolation. That is, for $x_i < x < x_{i+1}$, the slope is $(y_{i+1} - y_i)/(x_{i+1} - x_i)$. If $x = x_j$ is the point in **ptwXY** at index j then for side = '+', $i = j$ is used in the above slope equation. Else, if side = '-', $i = j - 1$ is used in the above slope equation.

C declaration:

```
fnu_status ptwXY_getSlopeAtX( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY,
                              double x,
                              const char side,
                              double *slope );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`index:`	The index of the point to set.
`x:`	The x value.
`y:`	The y value.

If side is neither '-' or '+', the error **nfu_badInput** is returned.

5.1.53 ptwXY_domainMinAndFrom — Not for general use

This function returns the domainMin value and indicates whether the minimum value resides in the primary or secondary cache.

C declaration: — **This function is not intended for general use.** —

```
nfu_status ptwXY_domainMinAndFrom( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   ptwXY_dataFrom *dataFrom,
                                   double *domainMin );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`dataFrom:` The output of this argument indicates which cache the minimum value resides in.
`domainMin:` The value of the domain min.

The return value from this function is domainMin. If there are no data in the **ptwXYPoints** object, then **dataFrom** is set to **ptwXY_dataFrom_Unknown**. Otherwise, it is set to **ptwXY_dataFrom_Points** or **ptwXY_dataFrom_Overflow** if the minimum value is in the primary or secondary cache respectively.

5.1.54 ptwXY_domainMin

This function returns the domainMin value returned by **ptwXY_domainMinAndFrom**. The calling function should check that the **ptwXYPoints** object contains at least one point (i.e., that the length is greater than 0). If the length is 0, the return value is undefined.

C declaration:

```
nfu_status ptwXY_domainMin( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY,
                            double *domainMin );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`domainMin:` The value of the domain min.

5.1.55 ptwXY_domainMaxAndFrom — Not for general use

This function returns the domainMax value and indicates whether the maximum value resides in the primary or secondary cache.

C declaration: — **This function is not intended for general use.** —

```
nfu_status ptwXY_doaminMaxAndFrom( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   ptwXY_dataFrom *dataFrom,
                                   double *domainMax );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

`dataFrom:` The output of this argument indicates which cache the maximum value resides in.

`domainMax:` The value of the domain max.

The return value from this function is domainMax. If there are no data in the **ptwXYPoints** object, then **dataFrom** is set to **ptwXY_dataFrom_Unknown**. Otherwise, it is set to **ptwXY_dataFrom_Points** or **ptwXY_dataFrom_Overflow** if the maximum value is in the primary or secondary cache respectively.

5.1.56 ptwXY_domainMax

This function returns the doaminMax value returned by **ptwXY_domainMinAndFrom**. The calling function should check that the **ptwXYPoints** object contains at least one point (i.e., that the length is greater than 0). If the length is 0, the return value is undefined.

C declaration:

```
nfu_status ptwXY_domainMax( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY,
                            double *domainMax );
```

`smr:` The **statusMessageReporting** instance to record errors.

`ptwXY:` A pointer to the **ptwXYPoints** object.

`domainMax:` The value of the domain max.

5.1.57 ptwXY_range

This function returns the minimum and maximum y values in **ptwXY**.

C declaration:

```
nfu_status ptwXY_getYMin( statusMessageReporting *smr,
                          ptwXYPoints *ptwXY,
                          double *rangeMin,
                          double *rangeMax );
```

`smr:` The **statusMessageReporting** instance to record errors.

`ptwXY:` A pointer to the **ptwXYPoints** object.

`rangeMin:` The value of the range min.

`rangeMax:` The value of the range max.

5.1.58 ptwXY_rangeMin

This function returns the minimum y value in **ptwXY**.

C declaration:

```
nfu_status ptwXY_getYMin( statusMessageReporting *smr,
                          ptwXYPoints *ptwXY,
                          double *rangeMin );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`rangeMin:` The value of the range min.

5.1.59 ptwXY_rangeMax

This function returns the maximum y value in **ptwXY**.

C declaration:

```
nfu_status ptwXY_getYMax( statusMessageReporting *smr,
                          ptwXYPoints *ptwXY,
                          double *rangeMax );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`rangeMax:` The value of the range max.

5.1.60 ptwXY_initialOverflowPoint — Not for general use

This function initializes a point in the secondary cache.

C declaration: — This function is not intended for general use. —

```
void ptwXY_initialOverflowPoint(
                    ptwXYOverflowPoint *overflowPoint,
                    ptwXYOverflowPoint *prior,
                    ptwXYOverflowPoint *next );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`prior:` The prior point in the linked list.
`next:` The next point in the linked list.

5.1.61 ptwXY_interpolationToString

This function returns the string representation of **interpolation**.

C declaration:

```
char const *ptwXY_interpolationToString( ptwXY_interpolation interpolation );
```

`interpolation:` The interpolation values.

5.1.62 ptwXY_stringToInterpolation

This function returns the **ptwXY_interpolation** value for **interpolationString**.

C declaration:

```
ptwXY_interpolation *ptwXY_stringToInterpolation(
                                    char const *interpolationString );
```

`interpolationString:` The interpolation string.

5.2 Methods

This section decribes all the functions in the file "ptwXY_method.c".

5.2.1 ptwXY_clip

This function clips the y-values of **ptwXY** to be within the range **rangeMin** and **rangeMax**. Points will be added to insure that the curve within the **rangeMin** and **rangeMax** is not altered.

C declaration:

```
fnu_status ptwXY_clip( statusMessageReporting *smr,
                       ptwXYPoints *ptwXY,
                       double rangeMin,
                       double rangeMax );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`rangeMin:`	All y-values in **ptwXY** will be greater than or equal to this value.
`rangeMax:`	All y-values in **ptwXY** will be less than or equal to this value.

5.2.2 ptwXY_thicken

This function thicken the points in **ptwXY** by adding points as determined by the input parameters.

C declaration:

```
fnu_status ptwXY_thicken( statusMessageReporting *smr,
                          ptwXYPoints *ptwXY,
                          int sectionSubdivideMax,
                          double dDomainMax,
                          double fDomainMax );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`sectionSubdivideMax:`	The maximum number of points to add between two initial consecutive points.
`dDomainMax:`	The desired maximum absolute x step between consecutive points.
`fDomainMax:`	The desired maximum relative x step between consecutive points.

This function adds points so that $x_{j+1} - x_j \leq$ **dDoaminMax** and $x_{j+1}/x_j \leq$ **fDomainMax** but will never add more then **sectionSubdivideMax** points between any of the orginal points. If **sectionSubdivideMax** < 1 or **dDomainMax** < 0 or **fDomainMax** < 1, the error **nfu_badInput** is return.

5.2.3 ptwXY_thin

This function returns a clone of **ptwXY** with its points thinned (i.e., removed) while maintaining interpolation **accuracy** with **ptwXY**.

C declaration:

```
ptwXPoints *ptwXY_thin( statusMessageReporting *smr,
                        ptwXYPoints *ptwXY,
                        double accuracy );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`accuracy:`	The accuracy of the thinned **ptwXYPoints** object.

5.2.4 ptwXY_thinDomain

This function returns a clone of **ptwXYPoints** whose x-values are those of **ptwXY** but thinned so that

$$x[i+1] - x[i] \geq 0.5 \times \text{epsilon} \times (\text{x[i} + 1] + \text{x[i]}) \quad . \tag{3}$$

C declaration:

```
ptwXPoints *ptwXY_thinDomain( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY,
                              double epsilon );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`epsilon:`	The epsilon to thin the x-values to.

5.2.5 ptwXY_trim

This function removes all extra 0.'s at the beginning and end of **ptwXY**.

C declaration:

```
fnu_status ptwXY_trim( statusMessageReporting *smr,
                       ptwXYPoints *ptwXY );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.

If **ptwXYPoints** starts (ends) with more than two 0.'s then all intermediary are removed.

5.2.6 ptwXY_union

This function creates a new **ptwXY** instance whose x-values are the union of **ptwXY1**'s and **ptwXY2**'s x-values. The domains of **ptwXY1** and **ptwXY2** do not have to be mutual.

C declaration:

```
ptwXYPoints *ptwXY_union( statusMessageReporting *smr,
                         ptwXYPoints *ptwXY1,
                         ptwXYPoints *ptwXY2,
                         int unionOptions );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY1:`	A pointer to a **ptwXYPoints** object.
`ptwXY2:`	A pointer to a **ptwXYPoints** object.
`unionOptions:`	Specifies options (see below).

If an error occurs, NULL is returned. The default behavior of this function can be altered by setting bits in the argument **unionOptions** . Currently, there are two bits, set via the C marcos **ptwXY_union_fill** and **ptwXY_union_trim**, that alter **ptwXY_union**'s behavior. The macro **ptwXY_union_fill** causes all y-values of the new **ptwXYPoints** object to be filled via the y-values of **ptwXY1**; otherwise, the y-values are all zero. Normally, the new **ptwXYPoints** object's x domain spans all x-values in both **ptwXY1** and **ptwXY2**. The macro **ptwXY_union_trim** limits the x domain to the common x domain of **ptwXY1** and **ptwXY2**.

The returned **ptwXYPoints** object will always contain no points in the **overflowPoints** region.

5.2.7 ptwXY_scaleOffsetXAndY

This function scales and offset the x-values and y-values.

C declaration:

```
nfu_status ptwXY_scaleOffsetXAndY( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY1,
                                   double domainScale,
                                   double domainOffset,
                                   double rangeScale,
                                   double rangeOffset );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`domainScale:`	The scale for the x-values.
`domainOffset:`	The offset for the x-values.
`rangeScale:`	The scale for the y-values.
`rangeOffset:`	The offset for the y-values.

5.3 Unitary operators

This section decribes all the functions in the file "ptwXY_unitaryOperators.c".

5.3.1 ptwXY_abs

This function applies the math absolute operation to every y-value in **ptwXY**.

C declaration:

```
fnu_status ptwXY_abs( statusMessageReporting *smr,
                      ptwXYPoints *ptwXY );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.

5.3.2 ptwXY_neg

This function applies the math negate operation to every y-value in **ptwXY**.

C declaration:

```
fnu_status ptwXY_neg( statusMessageReporting *smr,
                      ptwXYPoints *ptwXY );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.

5.4 Binary operators

This section decribes all the functions in the file "ptwXY_binaryOperators.c".

5.4.1 ptwXY_slopeOffset

This function applies the math operation (y_i = slope $\times$ y_i + offset) to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_slopeOffset( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY,
                              double slope,
                              double offset );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`slope:` The slope.
`offset:` The offset.

5.4.2 ptwXY_add_double

This function applies the math operation ($y_i = y_i$ + offset) to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_add_double( statusMessageReporting *smr,
                             ptwXYPoints *ptwXY,
                             double offset );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`offset:` The offset.

5.4.3 ptwXY_sub_doubleFrom

This function applies the math operation ($y_i = y_i$ - offset) to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_sub_double( statusMessageReporting *smr,
                             ptwXYPoints *ptwXY,
                             double offset );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`offset:` The offset.

5.4.4 ptwXY_sub_fromDouble

This function applies the math operation (y_i = offset - y_i) to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_sub_fromDouble( statusMessageReporting *smr,
                                 ptwXYPoints *ptwXY,
                                 double offset );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`offset:` The offset.

5.4.5 ptwXY_mul_double

This function applies the math operation (y_i = slope $\times$ y_i) to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_mul_double( statusMessageReporting *smr,
                             ptwXYPoints *ptwXY,
                             double slope );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`slope:` The slope.

5.4.6 ptwXY_div_doubleFrom

This function applies the math operation (y_i = y_i / divisor) to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_div_doubleFrom( statusMessageReporting *smr,
                                 ptwXYPoints *ptwXY,
                                 double divisor );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`divisor:` The divisor.

If **divisor** is zero, the error **nfu_divByZero** is returned.

5.4.7 ptwXY_div_fromDouble

This function applies the math operation (y_i = dividend / y_i) to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_div_fromDouble( statusMessageReporting *smr,
                                 ptwXYPoints *ptwXY,
                                 double dividend );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`dividend:` The dividend.

This function does not handle safe division (see Section 5.4.14). One way to do safe division is to use the function **ptwXY_valueTo_ptwXY** to convert the **dividend** value to a **ptwXYPoints** object and then use **ptwXY_div_ptwXY**.

5.4.8 ptwXY_mod

This function gives the remainer of y_i divide by m. That is, it set **ptwXY**'s y-values to

$$y_i = \mathrm{mod}(y_i, m) \quad . \tag{4}$$

C declaration:

```
fnu_status ptwXY_mod( statusMessageReporting *smr,
                      ptwXYPoints *ptwXY,
                      double m,
                      int pythonMod );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`m:` The modulus.
`pythonMod:` Controls whether the Python or C form of mod is implemented.

Python's and C's mod functions act differently for negative values. If **pythonMod** then the Python form is executed; otherwise, the C form is executed.

5.4.9 ptwXY_binary_ptwXY

This function creates a new **ptwXYPoints** object from the union of **ptwXY1** and **ptwXY2** and then applies the math operation

$$y_i(x_i) = s_1 \times y_1(x_i) + s_2 \times y_2(x_i) + s_{12} \times y_1(x_i) \times y_2(x_i) \tag{5}$$

to the new object. Here (x_i, y_i) is a point in the new object, $y_1(x_i)$ is **ptwXY1**'s y-value at x_i and $y_2(x_i)$ is **ptwXY2**'s y-value at x_i. This function is used internally to add, subtract and multiply

two **ptwXYPoints** objects. For example, addition is performed by setting s_1 and s_2 to 1. and s_{12} to 0.

C declaration:

```
ptwXYPoints *ptwXY_binary_ptwXY( statusMessageReporting *smr,
                                 ptwXYPoints *ptwXY1,
                                 ptwXYPoints *ptwXY2,
                                 double s1,
                                 double s2,
                                 double s12 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`s1:` The value s_1.
`s2:` The value s_2.
`s12:` The value s_{12}.

5.4.10 ptwXY_add_ptwXY

This function adds two **ptwXYPoints** objects and returns the result as a new **ptwXYPoints** object (i.e., it calls ptwXY_binary_ptwXY with $s_1 = s_2 = 1.$ and $s_{12} = 0.$).

C declaration:

```
ptwXYPoints *ptwXY_add_ptwXY( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.

5.4.11 ptwXY_sub_ptwXY

This function subtracts one **ptwXYPoints** objects from another, and returns the result as a new **ptwXY** object (i.e., it calls ptwXY_binary_ptwXY with $s_1 = 1.$, $s_2 = -1.$ and $s_{12} = 0.$).

C declaration:

```
ptwXYPoints *ptwXY_sub_ptwXY( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object which is the minuend.

`ptwXY2:` A pointer to a **ptwXYPoints** object which is the subtrahend.

5.4.12 ptwXY_mul_ptwXY

This function multiplies two **ptwXYPoints** objects and returns the result as a new **ptwXY** object (i.e., it calls ptwXY_binary_ptwXY with $s_1 = s_2 = 0.$ and $s_{12} = 1.$). This function does not infill.

C declaration:

```
ptwXYPoints *ptwXY_mul_ptwXY( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.

5.4.13 ptwXY_mul2_ptwXY

This function multiplies two **ptwXYPoints** objects and returns the result as a new **ptwXY** object. Unlike **ptwXY_mul_ptwXY**, this function will infill to obtain the desired accuracy.

C declaration:

```
ptwXYPoints *ptwXY_mul2_ptwXY( statusMessageReporting *smr,
                               ptwXYPoints *ptwXY1,
                               ptwXYPoints *ptwXY2 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.

5.4.14 ptwXY_div_ptwXY

This function divides two **ptwXYPoints** objects and returns the result as a new **ptwXY** object.

C declaration:

```
ptwXYPoints *ptwXY_div_ptwXY( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2,
                              int safeDivide );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.

`ptwXY2:`	A pointer to a **ptwXYPoints** object.
`safeDivide:`	If true safe division is performed.

5.5 Functions

This section decribes all the functions in the file "ptwXY_functions.c".

5.5.1 ptwXY_pow

This function applies the math operation $y_i = y_i^p$ to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_pow( statusMessageReporting *smr,
                      ptwXYPoints *ptwXY,
                      double p );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`p:` The exponent.

This function infills to maintain the initial accuracy.

5.5.2 ptwXY_exp

This function applies the math operation $y_i = \exp(a\, y_i)$ to the y-values of **ptwXY**.

C declaration:

```
fnu_status ptwXY_exp( statusMessageReporting *smr,
                      ptwXYPoints *ptwXY,
                      double a );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`a:` The exponent coefficient.

This function infills to maintain the initial accuracy.

5.5.3 ptwXY_convolution

This function returns the convolution of **ptwXY1** and **ptwXY2**.

C declaration:

```
ptwXYPoints *ptwXY_convolution( statusMessageReporting *smr,
                                ptwXYPoints *ptwXY1,
                                ptwXYPoints *ptwXY2,
                                int mode );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.

`ptwXY2:` A pointer to a **ptwXYPoints** object.
`mode:` Flag to determine the initial x-values for calculating the convolutions.

User should set **mode** to 0.

5.5.4 ptwXY_inverse

This function returns a new instance of **ptwXYPoints** that is the inverse of **ptwXY1**. That is, the returned points are (y_i, x_i) where (x_i, y_i) are the points of **ptwXY1**. All y-values of **ptwXY1** must be descending or increasing. If the y-values of **ptwXY1** are descending, the returned points are reversed to insure that in the returned instance $X_i < X_{i+1}$ where X_i is a x-value of new returned instance.

C declaration:

```
ptwXYPoints *ptwXY_inverse( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY1 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`mode:` Flag to determine the initial x-values for calculating the convolutions.

If an error occurs, NULL is returned.

5.6 Interpolation

This section decribes all the functions in the file "ptwXY_interpolation.c".

5.6.1 ptwXY_interpolatePoint

This function interpolates an x value between the points (x1,y1) and (x2,y2) to obtain its y value for the requested **interpolation**.

C declaration:

```
fnu_status ptwXY_interpolatePoint( statusMessageReporting *smr,
                                   ptwXY_interpolation interpolation,
                                   double x,
                                   double *y,
                                   double x1,
                                   double y1,
                                   double x2,
                                   double y2 );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`interpolation:`	Type of interpolation to perform (see Section 4.3).
`x:`	The x value at which the y value is desired.
`x1:`	The x value of the first point.
`y1:`	The y value of the first point.
`x2:`	The x value of the second point.
`y2:`	The y value of the second point.

If the interpolation flag is invalid or (x1 > x2) then **nfu_invalidInterpolation** is returned. If logarithm interpolation is requested for an axis, and one of the input values for that axis is less than or equal to 0., then **nfu_invalidInterpolation** is also returned. If interpolation is **ptwXY_interpolationOther** then **nfu_otherInterpolation** is returned.

5.6.2 ptwXY_flatInterpolationToLinear

This function returns a linear-linear interpolated representation of **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_flatInterpolationToLinear( statusMessageReporting *smr,
                                              ptwXYPoints *ptwXY,
                                              double lowerEps,
                                              double upperEps );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to a **ptwXYPoints** object.

	x_m	x_p
$x_i < 0$	$x_i(1+\epsilon_l)$	$x_p = x_i(1-\epsilon_p)$
$x_i == 0$	$-\epsilon_l$	ϵ_p
$x_i > 0$	$x_i(1-\epsilon_l)$	$x_p = x_i(1+\epsilon_p)$

Table 5: The value of x_m and x_p used to adjust interior points in **ptwXY_fla-InterpolationToLinear**. Here, ϵ_l = **lowerEps** and ϵ_p = **upperEps**.

`lowerEps:` The amount to adjust every interior point down in x.
`upperEps:` The amount to adjust every interior point up in x

For every interior point (i.e., (x_i, y_i) for $0 < i < n-1$ where n is the number of points), two points may be added. The positions of these points depend on **lowerEps** and **upperEps** as follows:

lowerEps == 0 and upperEps == 0: This condition is not allowed. status is set to **nfu_badInput** and NULL is returned. This condition is also returned if either **lowerEps** or **upperEps** is negative.

lowerEps > 0 and upperEps == 0: At each interior point (x_i, y_i) the two points (x_m, y_{i-1}) and (x_i, y_i) are set.

lowerEps == 0 and upperEps > 0: At each interior point (x_i, y_i) the two points (x_i, y_{i-1}) and (x_p, y_i) are set.

lowerEps > 0 and upperEps > 0: At each interior point (x_i, y_i), this point is removed and the two points (x_m, y_{i-1}) and (x_p, y_i) are set.

where x_m and x_p are given in Table 5.

5.6.3 ptwXY_toOtherInterpolation

This function returns **ptwXY** converted to interpolation **interpolation**.

C declaration:

```
ptwXYPoints *ptwXY_toOtherInterpolation( statusMessageReporting *smr,
                                         ptwXYPoints *ptwXY,
                                         ptwXY_interpolation interpolation,
                                         double accuracy );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to a **ptwXYPoints** object.
`interpolation:` The interpolation to convert to.
`accuracy:` The accuracy of the conversion.

Currently, **interpolation** can only be **ptwXY_interpolationLinLin**.

5.6.4 ptwXY_toUnitbase

This function returns a unit-based version of **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_toUnitbase( statusMessageReporting *smr,
                               ptwXYPoints *ptwXY,
                               int scaleRange );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`scaleRange:`	The y-values are not scaled if this is 0.

Unitbasing maps the domain to 0 to 1 by scaling each x-value as

$$x_i = (x_i - x_0)/(x_{n-1} - x_0) \tag{6}$$

and if **scaleRange** is not 0, scaling each y-value as

$$y_i = y_i \times (x_{n-1} - x_0) \quad . \tag{7}$$

Unitbasing is most useful for pdf's.

5.6.5 ptwXY_fromUnitbase

This function undoes the unit base mapping done by **ptwXY_toUnitbase**.

C declaration:

```
ptwXYPoints *ptwXY_fromUnitbase( statusMessageReporting *smr,
                                 ptwXYPoints *ptwXY,
                                 double domainMin,
                                 double domainMax,
                                 int scaleRange );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`domainMin:`	The lower domain for the returned **ptwXYPoints** instances.
`domainMax:`	The upper domain for the returned **ptwXYPoints** instances.
`scaleRange:`	The y-values are not scaled if this is 0.

Each x-value is scaled as

$$x_i = (\text{domainMax} - \text{domainMin}) \times x_i + \text{domainMin} \tag{8}$$

and if **scaleRange** is not 0, each y-value is scaled as

$$y_i = y_i/(\text{domainMax} - \text{domainMin}) \tag{9}$$

5.6.6 ptwXY_unitbaseInterpolate

This function returns a **ptwXYPoints** instance that is the unit-base interpolation of **ptwXY1** at w_1 and **ptwXY2** at w_2 at the w-value w.

C declaration:

```
ptwXYPoints *ptwXY_unitbaseInterpolate( statusMessageReporting *smr,
                                        double w,
                                        double w1,
                                        ptwXYPoints *ptwXY1,
                                        double w2,
                                        ptwXYPoints *ptwXY2,
                                        scaleRange );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`w:`	The w-value to interpole to.
`w1:`	The lower w-value
`ptwXY1:`	A pointer to a **ptwXYPoints** object at w1.
`w2:`	The upper w-value
`ptwXY2:`	A pointer to a **ptwXYPoints** object at w2.
`scaleRange:`	The y-values are not scaled if this is 0.

5.7 Integration

This section decribes all the functions in the file "ptwXY_integration.c".

5.7.1 ptwXY_f_integrate

This function returns the integral between two points using **interpolation**.

C declaration:

```
nfu_status ptwXY_f_integrate( statusMessageReporting *smr,
                              ptwXYPoints *ptwXY,
                              ptwXY_interpolation interpolation,
                              double x1,
                              double y1,
                              double x2,
                              double y2,
                              double *value );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to a **ptwXYPoints** object.
`interpolation:`	The interpolation between the two points.
`x2:`	The x-value of the lower point.
`y2:`	The y-value of the lower point
`x2:`	The x-value of the upper point.
`y2:`	The y-value of the upper point
`value:`	On return, the value of the integral.

5.7.2 ptwXY_integrate

This function returns the integral of **ptwXY** from **domainMin** to **domainMax**.

C declaration:

```
nfu_status ptwXY_integrate( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY,
                            double domainMin,
                            double domainMax,
                            double *integral);
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`domainMin:`	The lower limit of integration.
`domainMax:`	The upperlimit of integration.
`integral:`	On return, the value of the integral.

The return value is $\int_{\text{domainMin}}^{\text{domainMax}} f(x)dx$.

5.7.3 ptwXY_integrateDomain

This function returns the integral of **ptwXY** over its domain.

C declaration:

```
nfu_status ptwXY_integrateDomain( statusMessageReporting *smr,
                                  ptwXYPoints *ptwXY,
                                  double *value );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`integral:` On return, the value of the integral.

The return value is $\int f(x)dx$ over the domain of **ptwXY**.

5.7.4 ptwXY_normalize

This function multiplies each y-value of **ptwXY** by a constant so that its integral is then normalized to 1.

C declaration:

```
nfu_status ptwXY_normalize( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

5.7.5 ptwXY_integrateDomainWithWeight_x

This function returns the integral of **ptwXY** weighted by x over its domain.

C declaration:

```
nfu_status ptwXY_integrateDomainWithWeight_x( statusMessageReporting *smr,
                                              ptwXYPoints *ptwXY,
                                              double *integral );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`integral:` On return, the value of the integral.

The return value is $\int xf(x)dx$ over the domain of **ptwXY**.

5.7.6 ptwXY_integrateWithWeight_x

This function returns the integral of **ptwXY** weighted by x from domainMin to domainMax.

C declaration:

```
nfu_status ptwXY_integrateWithWeight_x( statusMessageReporting *smr,
                                        ptwXYPoints *ptwXY,
                                        double domainMin,
                                        double domainMax,
                                        double *integral );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`domainMin:`	The lower limit of the integration.
`domainMax:`	The upper limit of the integration.
`integral:`	On return, the value of the integral.

The return value is $\int_{\text{domainMin}}^{\text{domainMax}} xf(x)dx$ over the domain of **ptwXY**.

5.7.7 ptwXY_integrateDomainWithWeight_sqrt_x

This function returns the integral of **ptwXY** weighted by $\sqrt{x}$ over its domain.

C declaration:

```
nfu_status ptwXY_integrateDomainWithWeight_sqrt_x( statusMessageReporting
*smr,
                                                   ptwXYPoints *ptwXY,
                                                   double *integral );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`integral:`	On return, the value of the integral.

The return value is $\int \sqrt{x} f(x)dx$ over the domain of **ptwXY**.

5.7.8 ptwXY_integrateWithWeight_sqrt_x

This function returns the integral of **ptwXY** weighted by x from domainMin to domainMax.

C declaration:

```
nfu_status ptwXY_integrateWithWeight_sqrt_x( statusMessageReporting *smr,
                                             ptwXYPoints *ptwXY,
                                             double domainMin,
                                             double domainMax,
                                             double *integral );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`domainMin:`	The lower limit of the integration.
`domainMax:`	The upper limit of the integration.
`integral:`	On return, the value of the integral.

The return value is $\int_{\text{domainMin}}^{\text{domainMax}} \sqrt{x} f(x) dx$ over the domain of **ptwXY**.

5.7.9 ptwXY_groupOneFunction

This function integrates **ptwXY** between each pair of consecutive points in **groupBoundaries** and returns each integral's value as an element of the returned **ptwXPoints**.

C declaration:

```
ptwXPoints *ptwXY_groupOneFunction( statusMessageReporting *smr,
                                    ptwXYPoints *ptwXY,
                                    ptwXPoints *groupBoundaries,
                                    ptwXY_group_normType normType,
                                    ptwXPoints *norm );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`groupBoundaries:`	A list of x-values.
`normType:`	The type of normalization to apply to integration.
`norm:`	A list of normalizations to be applied when normType is **ptwXY_group_normType_norm**.

Let **groupBoundaries** contain n x-values with $x_i < x_{i+1}$. The returned **ptwXPoints** will contain $n-1$ values I_i such that

$$I_i = \frac{1}{n_i} \int_{x_i}^{x_{i+1}} f(x) dx \tag{10}$$

where n_i is determined by **normType** as,

ptwXY_group_normType_none: $n_i = 1$.

ptwXY_group_normType_dx: $n_i = x_{i+1} - x_i$.

ptwXY_group_normType_norm: $n_i =$ the $(i-1)^{th}$ element of norm.

5.7.10 ptwXY_groupTwoFunctions

This function integrates the product of **ptwXY1** and **ptwXY2** between each pair of consecutive points in **groupBoundaries** and returns each integral's value as an element of the returned **ptwXPoints**.

C declaration:

```
ptwXPoints *ptwXY_groupTwoFunctions( statusMessageReporting *smr,
                                     ptwXYPoints *ptwXY1,
                                     ptwXYPoints *ptwXY2,
                                     ptwXPoints *groupBoundaries,
                                     ptwXY_group_normType normType,
                                     ptwXPoints *norm );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY1:`	A pointer to the **ptwXYPoints** object.
`ptwXY2:`	A pointer to the **ptwXYPoints** object.
`groupBoundaries:`	A list of x-values.
`normType:`	The type of normalization to apply to integration.
`norm:`	A list of normalizations to be applied when normType is **ptwXY_group_normType_norm**.

Let **groupBoundaries** contain n x-values with $x_i < x_{i+1}$. The returned **ptwXPoints** will contain $n-1$ values I_i such that

$$I_i = \frac{1}{n_i} \int_{x_i}^{x_{i+1}} f(x)\, g(x) dx \tag{11}$$

where n_i is determined by **normType** as,

ptwXY_group_normType_none: $n_i = 1$.

ptwXY_group_normType_dx: $n_i = x_{i+1} - x_i$.

ptwXY_group_normType_norm: $n_i =$ the $(i-1)^{th}$ element of norm.

5.7.11 ptwXY_groupThreeFunctions

This function integrates the product **ptwXY1**, **ptwXY2** and **ptwXY3** between each pair of consecutive points in **groupBoundaries** and returns each integral's value as an element of the returned **ptwXPoints**.

C declaration:

```
ptwXPoints *ptwXY_groupThreeFunctions( statusMessageReporting *smr,
                                       ptwXYPoints *ptwXY1,
                                       ptwXYPoints *ptwXY2,
                                       ptwXYPoints *ptwXY3,
                                       ptwXPoints *groupBoundaries,
                                       ptwXY_group_normType normType,
                                       ptwXPoints *norm );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY1:`	A pointer to the **ptwXYPoints** object.
`ptwXY2:`	A pointer to the **ptwXYPoints** object.
`ptwXY3:`	A pointer to the **ptwXYPoints** object.
`groupBoundaries:`	A list of x-values.
`normType:`	The type of normalization to apply to integration.
`norm:`	A list of normalizations to be applied when normType is **ptwXY_group_normType_norm**.

Let **groupBoundaries** contain n x-values with $x_i < x_{i+1}$. The returned **ptwXPoints** will contain $n-1$ values I_i such that

$$I_i = \frac{1}{n_i} \int_{x_i}^{x_{i+1}} f(x)g(x)h(x)dx \tag{12}$$

where n_i is determined by **normType** as,

ptwXY_group_normType_none: $n_i = 1$.

ptwXY_group_normType_dx: $n_i = x_{i+1} - x_i$.

ptwXY_group_normType_norm: $n_i =$ the $(i-1)^{th}$ element of norm.

5.8 Convenient

This section decribes all the functions in the file "ptwXY_convenient.c".

5.8.1 ptwXY_getXArray

This function returns, as an **ptwXPoints**, the list of x values in **ptwXY** instance. The returned object is allocated by **ptwXY_getXArray** and must be freed by the user.

C declaration:

```
ptwXPoints *ptwXY_getXArray( statusMessageReporting *smr,
                             ptwXYPoints *ptwXY );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.

Returns NULL if an error occured.

5.8.2 ptwXY_ysMappedToXs

This function returns the y-values of **ptwXY** at the points defined by the values of **Xs**.

C declaration:

```
ptwXPoints *ptwXY_ysMappedToXs( statusMessageReporting *smr,
                                ptwXYPoints *ptwXY,
                                ptwXPoints *Xs,
                                int64_t *offset );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`Xs:` The list of x-values to evaluate **ptwXY** at.
`offset:` The index in **ptwXY** of the first point greater than or equal to the first point of **Xs**.

Returns NULL if an error occured.

5.8.3 ptwXY_dullEdges

This function insures that the y-values at the end-points of **ptwXY** are 0. This can be usefull for making sure two **ptwXYPoints** instances have mutual domains.

C declaration:

```
nfu_status ptwXY_dullEdges( statusMessageReporting *smr,
                            ptwXYPoints *ptwXY,
                            double lowerEps,
                            double upperEps,
                            int positiveXOnly );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`lowerEps:` The amount to adjust the first points.
`upperEps:` The amount to adjust the last points.
`positiveXOnly:` The next point in the linked list.

The description here will mainly focuses on the dulling of the low point of ptwXY, the upper point's dulling is similar. Let ϵ_l = lowerEps, x_0 and y_0 be the first point of ptwXY and x_1 and y_1 be the second point of ptwXY. Also, if $x_0 \neq 0$ then let $\Delta x = |\epsilon_l| x_0$ otherwise let $\Delta x = |\epsilon_l|$. Then, the points around x_0 are modified only if lowerEps $\neq 0$ and $y_0 \neq 0$. The dulling of the lower edge can have one of the four outcomes listed here,

	$x_0, 0$	x_p, y_p	x_1, y_1	outcome 1
	$x_0, 0$		x_1, y_1	outcome 2
$x_m, 0$	x_0, y'_0	x_p, y_p	x_1, y_1	outcome 3
$x_m, 0$	x_0, y'_0		x_1, y_1	outcome 4

In all outcomes, the lower point now has $y = 0$. The point is added at $x_p = x_0 + \Delta x$ with $y = f(x_p)$ only if $x_0 + 2\Delta x < x_2$. If the point at $x_m = x_0 - \Delta x$ is not added, then y_0 is set to 0 as shown in outcomes 1 and 2. The point x_m is not added if $\epsilon_l > 0$, or positiveXOnly is true and $x_m < 0$ and $x_0 \geq 0$.

The dulling of the upper edge can have one of the four outcomes listed here,

x_{k-1}, y_{k-1}	x_m, y_m	$x_k, 0$		outcome 1
x_{k-1}, y_{k-1}		$x_k, 0$		outcome 2
x_{k-1}, y_{k-1}	x_m, y_m	x_k, y_k	$x_p, 0$	outcome 3
x_{k-1}, y_{k-1}		x_k, y_k	$x_p, 0$	outcome 4

where k is the index of the last point.

5.8.4 ptwXY_mergeClosePoints

Removes and/or moves points so that no two consecutive points are too close to others.

C declaration:

```
fnu_status ptwXY_mergeClosePoints( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY,
                                   double epsilon);
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`epsilon:` The minimum relative spacing desired.

Points are removed and/or moved so the $x_{i+1} - x_i \leq$ **epsilon** $\times (x_i + x_{i+1})/2$.

5.8.5 ptwXY_intersectionWith_ptwX

This function returns an **ptwXYPoints** instance whose x-values are the intersection of **ptwXY**'s and **ptwX**'s x-values. The domains of **ptwXY** and **ptwX** do not have to be mutual.

C declaration:

```
ptwXY_intersectionWith_ptwX( statusMessageReporting *smr,
                             ptwXYPoints *ptwXY,
                             ptwXPoints *ptwX );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY:` A pointer to the **ptwXYPoints** object.
`ptwX:` A pointer to the **ptwXPoints** object.

5.8.6 ptwXY_areDomainsMutual

This function returns **nfu_Okay** if **ptwXY1** and **ptwXY2** are mutual.

C declaration:

```
fnu_status ptwXY_areDomainsMutual( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY1,
                                   ptwXYPoints *ptwXY2 );
```

`smr:` The **statusMessageReporting** instance to record errors.
`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.

If one or both of **ptwXY1** and **ptwXY2** are empty, **nfu_empty** is returned. If one or both of **ptwXY1** and **ptwXY2** has only one point, **nfu_tooFewPoints** is returned. If the domains are not mutual, **nfu_domainsNotMutual** is returned.

5.8.7 ptwXY_tweakDomainsToMutualify

If a small tweak of one or more end points will make the domains of **ptwXY1** and **ptwXY2** mutual, then the tweak is made. A small tweak is defined as

$$|x2 - x1| \leq \text{Epsilon}(\ |x1| + |x2|\ ) \tag{13}$$

where $x1$ is one endpoint of **ptwXY1**, $x2$ is the corresponding endpoint of **ptwXY2** and

$$\text{Epsilon} = \text{fabs}(\ \text{epsilon}\ ) + \text{fabs}(\ \text{epsilonFactor} * \text{DBL\_EPSILON}\ ) \tag{14}$$

C declaration:

```
fnu_status ptwXY_tweakDomainsToMutualify( statusMessageReporting *smr,
                                          ptwXYPoints *ptwXY1,
                                          ptwXYPoints *ptwXY2,
                                          int epsilonFactor,
                                          double epsilon );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY1:`	A pointer to a **ptwXYPoints** object.
`ptwXY2:`	A pointer to a **ptwXYPoints** object.
`epsilonFactor:`	Relative comparision value.
`epsilon:`	Relative comparision value in units of DBL_EPSILON.

5.8.8 ptwXY_mutualifyDomains

If possible and needed, this function mutualifies the domains of **ptwXY1** and **ptwXY2** by calling **ptwXY_dullEdges** on one or both of **ptwXY1** and **ptwXY2** as needed.

C declaration:

```
fnu_status ptwXY_mutualifyDomains( statusMessageReporting *smr,
                                   ptwXYPoints *ptwXY1,
                                   double lowerEps1,
                                   double upperEps1,
                                   int positiveXOnly1,
                                   ptwXYPoints *ptwXY2,
                                   double lowerEps2,
                                   double upperEps2,
                                   int positiveXOnly2 );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY1:`	A pointer to a **ptwXYPoints** object.
`lowerEps1:`	If needed the value of **lowerEps** passed to **ptwXY_dullEdges** when dulling **ptwXY1**.
`upperEps1:`	If needed the value of **upperEps** passed to **ptwXY_dullEdges** when dulling **ptwXY1**.
`positiveXOnly1:`	The value of **positiveXOnly** passed to **ptwXY_dullEdges** when dulling **ptwXY1**.
`ptwXY2:`	A pointer to a **ptwXYPoints** object.
`lowerEps2:`	If needed the value of **lowerEps** passed to **ptwXY_dullEdges** when dulling **ptwXY2**.
`upperEps2:`	If needed the value of **upperEps** passed to **ptwXY_dullEdges** when dulling **ptwXY2**.

`positiveXOnly2:` The value of **positiveXOnly** passed to **ptwXY_dullEdges** when dulling **ptwXY2**.

5.8.9 ptwXY_copyToC_XY

This function copies the points from index **index1** inclusive to **index2** exclusive of **ptwXY** into the address pointed to by **xys**.

C declaration:

```
fnu_status ptwXY_copyToC_XY( statusMessageReporting *smr,
                             ptwXYPoints *ptwXY,
                             int64_t index1,
                             int64_t index2,
                             int64_t allocatedSize,
                             int64_t numberOfPoints,
                             double *xys );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`index1:`	The lower index.
`index2:`	The upper index.
`allocatedSize:`	The size of the space allocated for xys in pairs of C-double.
`numberOfPoints:`	The number of (x,y) points filled into *xys.
`xys:`	A pointer to the space to write the data.

The size of **xys** must be at least 2 × sizeof(double) × **allocatedSize** bytes. The values of **index1** and **index2** are ajusted as follows. If **index1** is less than 0, it is set to 0. Then if **index2** is less than **index1**, it is set to **index1**. Finally, if **index2** is greater than the length of **ptwXY**, it is set to the length of **ptwXY**. If **allocatedSize** is less than the number of points to be copied (i.e., **index2** - **index1** after **index1** and **index2** are adjusted) then **nfu_insufficientMemory** is returned;

5.8.10 ptwXY_valuesToC_XsAndYs

This function copies the points of **ptwXY** to **xs** and **ys**. Memory for **xs** and **ys** are allocated and the callers is responsible for free-ing their memory.

C declaration:

```
fnu_status ptwXY_valuesToC_XsAndYs( statusMessageReporting *smr,
                                    ptwXYPoints *ptwXY,
                                    double **xs,
                                    double **ys );
```

`smr:` The **statusMessageReporting** instance to record errors.

`ptwXY:` A pointer to the **ptwXYPoints** object.
`xs:` The x-values of **ptwXY**
`ys:` The y-values of **ptwXY**

5.8.11 ptwXY_valueTo_ptwXY

This function creates a **ptwXYPoints** instance with the two points (x1,y), (x2,y) where x1 < x2.

C declaration:

```
ptwXYPoints *ptwXY_valueTo_ptwXY( statusMessageReporting *smr,
                                  double x1,
                                  double x2,
                                  double y );
```

`smr:` The **statusMessageReporting** instance to record errors.
`x1:` x value for the lower point.
`x2:` x value for the upper point.
`y:` y value for both points.

If an error occurs, NULL is returned.

5.8.12 ptwXY_createGaussianCenteredSigma1

This function returns a **ptwXYPoints** instance of the simple Gaussian $y(x) = \exp(-x^2/2)$.

C declaration:

```
ptwXYPoints *ptwXY_createGaussianCenteredSigma1( statusMessageReporting *smr,
                                                 double accuracy );
```

`smr:` The **statusMessageReporting** instance to record errors.
`accuracy:` The returned points are accurate to accuracy.

The domain ranges from $-\sqrt{2\log(\text{yMin})}$ to $\sqrt{2\log(\text{yMin})}$ where yMin = 10^{-10}.

5.8.13 ptwXY_createGaussian

This function returns a **ptwXYPoints** instance of the Gaussian $y(x) = a\exp(-(x-c)^2/(2s))$. In the equation a = **amplitude**, c = **xCenter** and s = **sigma**. This function calls **ptwXY_createGaussianCenteredSigma1** and then scales the x and y values.

C declaration:

```
ptwXYPoints *ptwXY_createGaussian( statusMessageReporting *smr,
                                   double accuracy,
                                   double xCenter,
                                   double sigma,
                                   double amplitude,
                                   double domainMin,
                                   double domainMax,
                                   double dullEps );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`accuracy:`	The returned points are accurate to accuracy.
`xCenter:`	The center of the Gaussian.
`sigma:`	The width of the Gaussian.
`amplitude:`	The amplitude of the Gaussian.
`domainMin:`	The lower domain of the returned Gaussian.
`domainMax:`	The upper lower domain of the returned Gaussian.
`dullEps:`	Currently not implemented.

5.9 Miscellaneous

This section decribes all the functions in the file "ptwXY_misc.c".

5.9.1 ptwXY_limitAccuracy

This function returns accuracy limited between **ptwXY_minAccuracy** and 1. Ergo, it is similar to the pseudo code

$$\text{accuracy} = \min(\ 1,\ \max(\ \text{accuracy},\ \text{ptwXY\_minAccuracy}\ )\ ) \quad . \tag{15}$$

C declaration:

```
void ptwXY_limitAccuracy(  double accuracy );
```

`accuracy:` The desired accuracy.

5.9.2 ptwXY_update_biSectionMax — Not for general use

This function is used by **ptwXY** functions to update the member **biSectionMax** base on the prior length, given by **oldLength**, and the current length of **ptwXY**.

C declaration: — This function is not intended for general use. —

```
void ptwXY_update_biSectionMax( ptwXYPoints *ptwXY,
                                double oldLength );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`oldLength:` The prior length of **ptwXY**.

5.9.3 ptwXY_createFromFunction

This function creates a **ptwXYPoints** whose domain ranges from xs[0] to xs[n-1] and whose y-values are obtained from **func**. All values of **xs** are added and infill between them is done until either **accuracy** or **biSectionMax** is satisfied.

C declaration:

```
ptwXYPoints *ptwXY_createFromFunction( statusMessageReporting *smr,
                                       int N,
                                       double *xs,
                                       ptwXY_createFromFunction_callback func,
                                       void *argList,
                                       double accuracy,
                                       int checkForRoots,
                                       int biSectionMax );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`N:`	Number of values in **xs**
`xs:`	Minimum list of x-values to add.
`func:`	A function called to calculate $y(x)$.
`argList:`	A pointer passed to **func**.
`accuracy:`	The desired accuracy.
`checkForRoots:`	If true, points where $y = 0$ are searched and added.
`biSectionMax:`	Maximum number of bisections between consecutive points of **xs**.

The function **func** is called as:

```
nfu_status func( statusMessageReporting *smr, ptwXYPoint *point, void *argList );
```

Often, only 2 values in **xs** are needed. However, in some cases more values help with the bisection algorithm. For example, if the function is $y = \sin(x)$ for $0 \leq x \leq 2\pi$ and **xs** only contains the points 0 and 2π, then bisection will not add the point at π as it, like the bounding points, is 0.0. In this case, **xs** should contain the values 0.0, π and 2π.

5.9.4 ptwXY_applyFunction

This function is used by other functions to map y_i to func(x_i, y_i) with infilling as needed. For example, this function is used by **ptwXY_pow**.

C declaration:

```
nf_status ptwXY_applyFunction( statusMessageReporting *smr,
                               ptwXYPoints *ptwXY,
                               ptwXY_applyFunction_callback func,
                               void *argList );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`ptwXY:`	A pointer to the **ptwXYPoints** object.
`func:`	A function called to calculate $y(x)$.
`argList:`	A pointer passed to **func**.

This function infills to maintain the initial accuracy. The function **func** is called as:

```
nfu_status func( statusMessageReporting *smr, ptwXYPoint *point, void *argList );
```

5.9.5 ptwXY_fromString

This function creates a **ptwXYPoints** from the string of double values in **str**. There must be an even number of string doubles in **str**.

C declaration:

```
ptwXYPoints *ptwXY_fromString( statusMessageReporting *smr,
                               char const *str,
                               char sep,
                               ptwXY_interpolation interpolation,
                               char const *interpolationString,
                               double biSectionMax,
                               double accuracy,
                               char **endCharacter,
                               int useSystem_strtod );
```

`smr:`	The **statusMessageReporting** instance to record errors.
`str:`	The list of double values as a string.
`sep:`	The separator character between each double.
`interpolation:`	The interpolation of the data.
`interpolationString:`	The string representation of the string.
`biSectionMax:`	The biSectionMax of the returned **ptwXYPoints**.
`accuracy:`	The accuracy of the returned **ptwXYPoints**.
`endCharacter:`	The pointer to the character after the last character converted.
`useSystem_strtod:`	See the function **nfu_stringToListOfDoubles**.

5.9.6 ptwXY_showInteralStructure — Not for general use

This function writes out details of the data in a **ptwXYPoints** object, including much of the internal data normally not useful to a user. This function is intended for debugging.

C declaration: — This function is not intended for general use. —

```
void ptwXY_showInteralStructure( ptwXYPoints *ptwXY,
                                 FILE *f,
                                 int printPointersAsNull );
```

`ptwXY:`	A pointer to the **ptwXYPoints** object.
`f:`	The stream where the structure is written.
`printPointersAsNull:`	If true, all pointers are printed as if their value is NULL.

5.9.7 ptwXY_simpleWrite

This function writes out the (x,y) points of the **ptwXYPoints** object to a specified stream.

C declaration:

```
void ptwXY_simpleWrite( ptwXYPoints *ptwXY,
                        FILE *f,
                        char *format );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`f:` The stream where the points are written.
`format:` The format specifier to use for writing an (x,y) point.

The **format** must contain two C double specifier (e.g., `"%12.4f %17.7e\n"`), one each for the x- and y-values of a point. No line feed characters (e.g., `"\n"`) are printed, except those in **format**.

5.9.8 ptwXY_simplePrint

This function calls **ptwXY_simpleWrite** with stdout as the output stream.

C declaration:

```
void ptwXY_simplePrint( ptwXYPoints *ptwXY,
                        char *format );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`format:` The format specifier to use for writing an (x,y) point.

6 The detail of the calculations

The following sub-sections describe the details on some of the calculations. Consider two consecutive points (x_1, y_1) and (x_2, y_2) where $x_1 <= x <= x_2$ and $x_1 < x_2$, then interpolation is defined as

Lin-lin interpolation

$$y = \frac{y_2(x - x_1) + y_1(x_2 - x)}{(x_2 - x_1)} \tag{16}$$

Lin-log interpolation

$$y = y_1 \left(\frac{y_2}{y_1}\right)^{\frac{x-x_1}{x_2-x_1}} \tag{17}$$

Log-lin interpolation

$$y = \frac{y_1 \log(x_2/x) + y_2 \log(x/x_1)}{\log(x_2/x_1)} \tag{18}$$

Log-log interpolation

$$y = y_1 \left(\frac{x}{x_1}\right)^{\frac{\log(y_2/y_1)}{\log(x_2/x_1)}} \tag{19}$$

In some calculation we will need the x location for the maximum of the relative error, $(y' - y)/y$, between the approximate value, y', and the "exact" value, y. This x location occurs where the derivative of the relative error is zero:

$$\frac{d((y' - y)/y)}{dx} = \frac{d(y'/y - 1)}{dx} = \frac{d(y'/y)}{dx} = \frac{1}{y^2}\left(y\frac{dy'}{dx} - y'\frac{dy}{dx}\right) = 0 \tag{20}$$

6.1 Converting log-log to lin-lin

This section describes how fudge2dmath converts a **fudge2dmathXY** object with interpolation of **f2dmC_interpolationLogLog** (hence called log-log) to one with interpolation of **f2dmC_interpol-ation-LinLin** (hence called lin–lin).

From Eq. 20 the maximum of the relative error occurs where,

$$\frac{1}{y}\left(\frac{dy'}{dx} - \frac{y'}{y}\frac{dy}{dx}\right) = \left(\frac{1}{y}\right)\left\{\frac{y_2 - y_1}{x_2 - x_1} - \left(\frac{y'}{x}\right)\frac{\log(y_2/y_1)}{\log(x_2/x_1)}\right\} = 0 \tag{21}$$

The solution is

$$\frac{x}{x_1} = \frac{a(x_2/x_1 - y_2/y_1)}{(1 - a)(y_2/y_1 - 1)} \tag{22}$$

where $a = \log(y_2/y_1)/\log(x_2/x_1)$.