Manual for the numerical functions package*

Bret R. Beck
Lawrence Livermore National Laboratory
UCRL-

April 2, 2014

*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

1 Introduction

The **ptwXY** C object (i.e., an instance of C typedef **ptwXYPoints**) and supporting C routines are designed to handle point-wise interpolative data representing a function 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 routines 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 simple C routine to put the male and female populations into **ptwXY** objects and add them together to get the total population is:

```
#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_new( linlin, 5, 1e-3, nPairs, 4, &status, 0 );
    ptwXY_setXYData( males, nPairs - 1, maleData );

    females = ptwXY_new( linlin, 5, 1e-3, nPairs, 4, &status, 0 );
    ptwXY_setXYData( females, nPairs, femaleData );

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

    printf( "\nMale population\n" );
    printf( "  Year | Count\n" );
```

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: Males and females population of a bird species on an island for the years census were taken. There was no census taken of the male population in 1885.

```
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" );
```

The output of this code - compressed into fewer lines - is:

```
 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
```

In this example no error checking is shown. The routine **ptwXY_new** allocates memory for a new **ptwXYPoints** object, initializes it and returns a pointer to the object. The first argument of this routine in an interpolation flag. For all other arguments see Section 5.1.1 The routine **ptwXY_setXYData** takes a pointer to a **ptwXYPoints** object as its first argument and copies the list of doubles given by the third argument into the **ptwXYPoints** object's internal memory, deleting any data currently in the object. The second argument is the number of pairs of points in the third argument's data.

The routine **ptwXY_add_ptwXY** takes a **ptwXYPoints** object as its first and second 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 routine 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 accuracy

1.1.2 Mutual domains

Most routines 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 $y = f1(x) = (x-1)/4 + 1$ 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 $y = f2(x) = (x-1)/4 + 2$. The rule that should be implemented for adding these two functions is clear and is $s(x) = (x-1)/2 + 3 = f1(x) + f2(x)$ 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 $f1 + f2$.

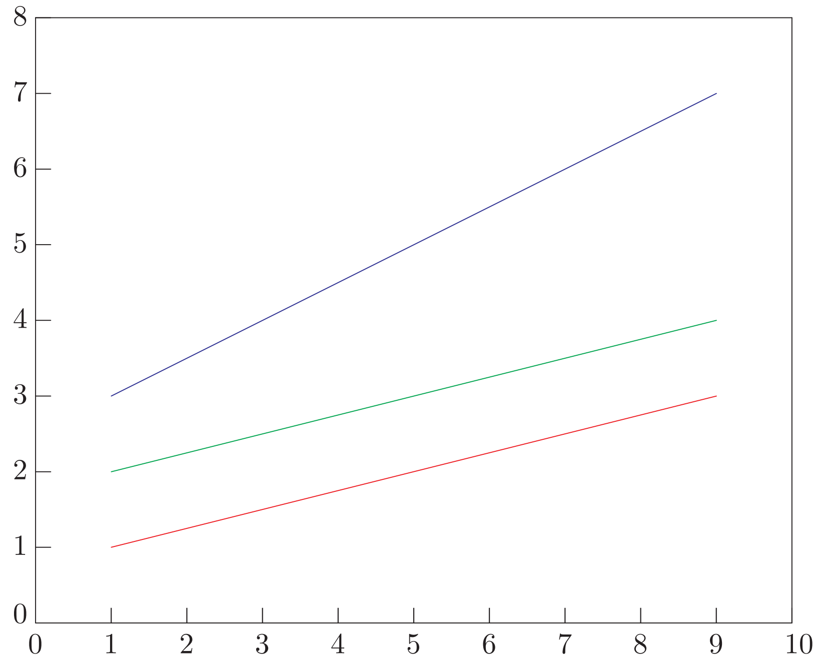

Figure 1: The red curve is $f1$, the green curve is $f2$ and the blue curve is $f1 + f2$.

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 $y = f3(x) = (x - 3)/2 + 1$. 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 $f3(x) = (x - 1)/2$ 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 $(g1 + g2) + g3$ is

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

while the addition $g1 + (g2 + g3)$ is

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

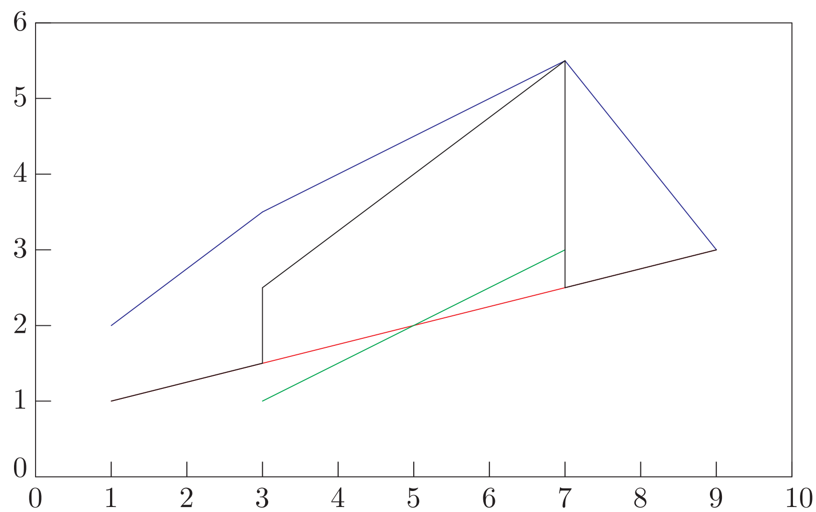

Figure 2: The red curve is $f1$, the green curve is $f2$ and the blue curve is $f1 + f3$ using the first rule and the black curve is $f1 + f3$ using the second rule.

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 $h2(x_{2,l}) = 0$. 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 $h1(x_{1,u}) = 0$. 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.

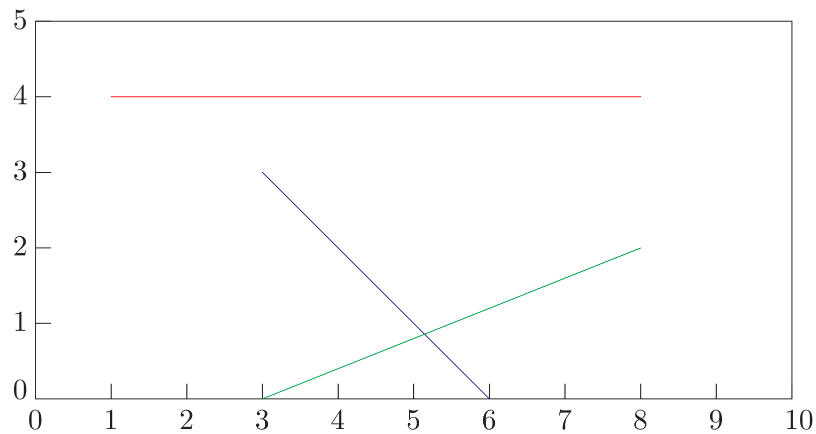

Figure 3: The red curve is $f4$, the green curve is $f5$ and the blue curve is $f6$. The domains of $f4$ and $f5$ are mutual as are the domains of $f5$ and $f6$. However, the domains of $f4$ and $f6$ are not mutual.

1.1.3 Infill

The addition of two linear functions yields another linear function. As example, the sum of $f1(x) = s1 \times x + y1$ and $f2(x) = s2 \times x + y2$ is $f1(x) + f2(x) = (s1 + s2) \times x + y1 + y2$. 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 (e.g., $f1(x) \times f2(x) = s1 \times s2 \times x^2 + (s1 \times y2 + s2 \times y1) \times x + y1 \times y2$). 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 $f_3(x) = x$ and $f_4(x) = 1 - x$ 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 it is, 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 initial segment of the union at most **biSectionMax** bisections are performed. Table 2 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 show the output from this code.

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

    double f3[4] = { 0., 0., 1., 1. }, f4[4] = { 0., 1., 1., 0. };
    double accuracy = 1e-3;
    ptwXYPoints *ptwXY3, *ptwXY4;
    nfu_status status;
    ptwXY_interpolation linlin = ptwXY_interpolationLinLin;

    ptwXY3 = ptwXY_create( linlin, 0, accuracy, 10, 10, 2, f3, &status, 0 );
    ptwXY4 = ptwXY_create( linlin, 0, accuracy, 10, 10, 2, f4, &status, 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;
    nfu_status status;

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

Table 2: This table show a snippet of the code used to generate the curves in Figure 4

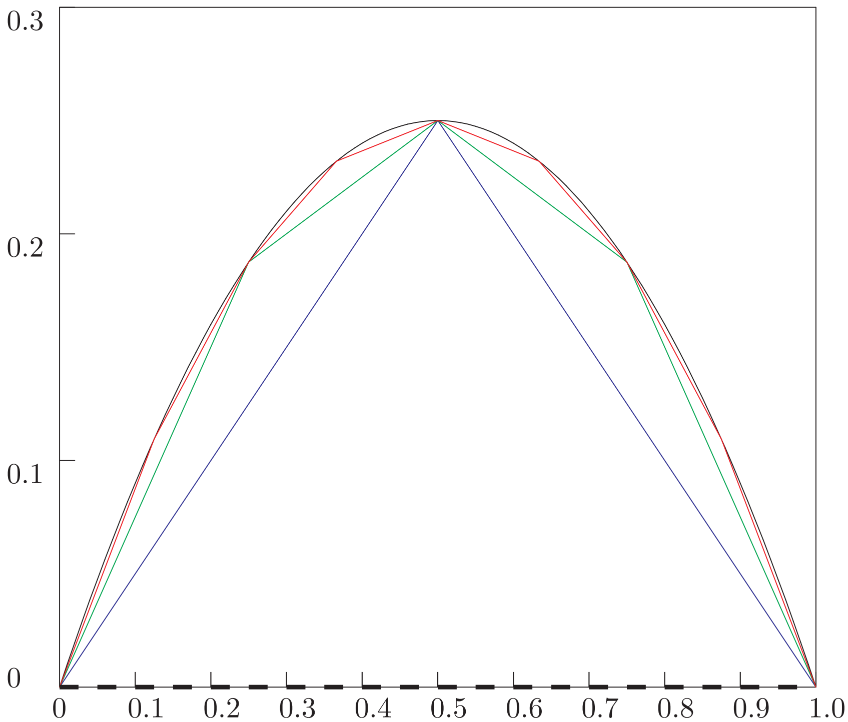

Figure 4: The solid black curve is the function $x(1-x)$. The blue, grean and red curves are products of $f3 \times f4$ from ptwXY_mul2_ptwXY for biSectionMax of 1, 2 and 3 respectively. The solid black curve and the red curve are nearly identical. The dashed thicker black line at the bottom of the plot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

1.1.4 Safe divide

2 Name convention

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

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

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 routines 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[2] 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

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

[2] 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.

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 routine **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 routines **ptwXY_reallocatePoints** and **ptwXY_reallocateOverflowPoints** respectively. In general these last two routines should not be called by the users unless they know that the a cache is woefully too small.

The routine **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".

4.1 ptwXYPoints

The **ptwXYPoints** type is defined as:

```
typedef
    struct ptwXYPoints_s {
        nfu_status status;
        ptwXY_sigma typeX, typeY;
        ptwXY_interpolation interpolation;
        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 lessThanEqualXPoint, greaterThanXPoint;
        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 routine and its members in **ptwXYPoints** should not be accessed user codes.

4.2 C macros

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** routines 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;
```

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. In the future uncertainty values may be added to both the x and y values. For examples, the y value could have an uncertainty dy, in which case a point would require 3 values (x, y, dy). The data type **ptwXY_sigma** specify the the number of values and their meaning for each axis and is defined as:

```
typedef enum ptwXY_sigma_e {
    ptwXY_sigma_none,       /* Datum contains only x or y values. */
```

```
    ptwXY_sigma_plusMinus,  /* Currently not supported. */
    ptwXY_sigma_Minus,      /* Currently not supported. */
    ptwXY_sigma_plus        /* Currently not supported. */
} ptwXY_sigma;
```

4.5 Miscellaneous types

The routine **ptwXY_getPointsAroundX** is used by other routines to determine where an x value fits into a **ptwXYPoints** object. The return value of this routine 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;
```

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 routines in the file "ptwXY_core.c".

5.1.1 ptwXY_new

This routine allocates memory for a new **ptwXYPoints** object and initializes it by calling **ptwXY-_setup**.

C declaration:

```
ptwXYPoints *ptwXY_new( ptwXY_interpolation interpolation,
                        double biSectionMax,
                        double accuracy,
                        int64_t primarySize,
                        int64_t secondarySize,
                        fnu_status *status ),
                        int userFlag );
```

`interpolation:`	The type of interpolation to use.
`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.
`status:`	On return, the status value.
`userFlag:`	An user defined integer value not used by any ptwXY function.

If this routine fails, NULL is returned.

5.1.2 ptwXY_setup

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

C declaration:

```
fnu_status ptwXY_setup( ptwXYPoints *ptwXY,
                        ptwXY_interpolation interpolation,
                        double biSectionMax,
                        double accuracy,
                        int64_t primarySize,
                        int64_t secondarySize );,
                        int userFlag );
```

`ptwXY:`	A pointer to a **ptwXYPoints** object to initialize.
`interpolation:`	The type of interpolation to use.

`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 routines **ptwXY_reallocatePoints** and **ptwXY_reallocateOverflowPoints** respectively.

5.1.3 ptwXY_create

This routines combines **ptwXY_new** and **ptwXY_setXYData**.

C declaration:

```
ptwXYPoints *ptwXY_create( ptwXY_interpolation interpolation,
                           double biSectionMax,
                           double accuracy,
                           int64_t primarySize,
                           int64_t secondarySize,
                           int64_t length,
                           double *xy ),
                           fnu_status *status,
                           int userFlag );
```

`interpolation:`	The type of interpolation to use.
`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.
`status:`	On return, the status value.
`userFlag:`	An user defined integer value not used by any ptwXY function.

If this routine fails, NULL is returned.

5.1.4 ptwXY_createFrom_Xs_Ys

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

C declaration:

```
ptwXYPoints *ptwXY_createFrom_Xs_Ys( ptwXY_interpolation interpolation,
                                     double biSectionMax,
                                     double accuracy,
                                     int64_t primarySize,
                                     int64_t secondarySize,
                                     int64_t length,
                                     double *Xs ),
                                     double *Ys ),
                                     fnu_status *status,
                                     int userFlag );
```

`interpolation:`	The type of interpolation to use.
`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.
`status:`	On return, the status value.
`userFlag:`	An user defined integer value not used by any ptwXY function.

If this routine fails, NULL is returned.

5.1.5 ptwXY_copy

This routine clears the points in **dest** and then copies the points from **src** into **dest**. The **src** object is not modified.

C declaration:

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

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

5.1.6 ptwXY_clone

This routine creates a new **ptwXYPoints** object and sets its points to the points in its first argument.

C declaration:

```
ptwXYPoints *ptwXY_clone( ptwXYPoints *ptwXY,
                          fnu_status *status );
```

ptwXY: A pointer to the **ptwXYPoints** object.
status: On return, the status value.

If an error occurs, NULL is returned.

5.1.7 ptwXY_slice

This routine 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( ptwXYPoints *ptwXY,
                          int64_t index1,
                          int64_t index2,
                          int64_t secondarySize,
                          fnu_status *status );
```

ptwXY:	A pointer to the **ptwXYPoints** object.
index1:	The lower index.
index2:	The upper index.
secondarySize:	Initial size of the secondary cahce.
status:	On return, the status value.

If an error occurs, NULL is returned.

5.1.8 ptwXY_xSlice

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

C declaration:

```
ptwXYPoints *ptwXY_xSlice( ptwXYPoints *ptwXY,
                           double xMin,
                           double xMax,
                           int64_t secondarySize,
                           int fill,
                           fnu_status *status );
```

ptwXY:	A pointer to the **ptwXYPoints** object.
xMax:	The lower domain value.
xMax:	The upper domain value.
secondarySize:	Initial size of the secondary cahce.

`fill:`	Initial size of the secondary cahce.
`status:`	On return, the status value.

If an error occurs, NULL is returned.

5.1.9 ptwXY_xMinSlice

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

C declaration:

```
ptwXYPoints *ptwXY_xMinSlice( ptwXYPoints *ptwXY,
                              double xMin,
                              int64_t secondarySize,
                              int fill,
                              fnu_status *status );
```

`ptwXY:`	A pointer to the **ptwXYPoints** object.
`xMin:`	The lower domain value.
`secondarySize:`	Initial size of the secondary cahce.
`fill:`	Initial size of the secondary cahce.
`status:`	On return, the status value.

If an error occurs, NULL is returned.

5.1.10 ptwXY_xMaxSlice

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

C declaration:

```
ptwXYPoints *ptwXY_xMaxSlice( ptwXYPoints *ptwXY,
                              double xMax,
                              int64_t secondarySize,
                              int fill,
                              fnu_status *status );
```

`ptwXY:`	A pointer to the **ptwXYPoints** object.
`xMax:`	The upper domain value.
`secondarySize:`	Initial size of the secondary cahce.
`fill:`	Initial size of the secondary cahce.
`status:`	On return, the status value.

If an error occurs, NULL is returned.

5.1.11 ptwXY_getUserFlag

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

C declaration:

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

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

5.1.12 ptwXY_setUserFlag

This routine 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.

5.1.13 ptwXY_getAccuracy

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

C declaration:

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

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

5.1.14 ptwXY_setAccuracy

This routine 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.

5.1.15 ptwXY_getBiSectionMax

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

C declaration:

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

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

5.1.16 ptwXY_setBiSectionMax

This routine 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.

5.1.17 ptwXY_reallocatePoints

This routine changes the size of the primary cache.

C declaration:

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

`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**.

5.1.18 ptwXY_reallocateOverflowPoints

This routine changes the size of the secondary cache.

C declaration:

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

`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.

5.1.19 ptwXY_coalescePoints

This routine 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( ptwXYPoints *ptwXY,
                                  int64_t size,
                                  ptwXYPointsPoint *newPoint,
                                  int forceSmallerResize );
```

`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**.

5.1.20 ptwXY_simpleCoalescePoints

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

C declaration:

```
fnu_status ptwXY_simpleCoalescePoints( ptwXYPoints *ptwXY );
```

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

5.1.21 ptwXY_clear

This routine 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( ptwXYPoints *ptwXY );
```

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

5.1.22 ptwXY_release

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

C declaration:

```
fnu_status ptwXY_release( ptwXYPoints *ptwXY );
```

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

5.1.23 ptwXY_free

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

C declaration:

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

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

Any **ptwXYPoints** object allocated using **ptwXY_new** should be freed calling **ptwXY_free**. Once this routine is called, the **ptwXYPoints** object should never be used.

5.1.24 ptwXY_length

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

C declaration:

```
int64_t ptwXY_length( ptwXYPoints *ptwXY );
```

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

5.1.25 ptwXY_getNonOverflowLength

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

C declaration:

```
int64_t ptwXY_getNonOverflowLength( ptwXYPoints *ptwXY );
```

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

5.1.26 ptwXY_setXYData

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

C declaration:

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

`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.

5.1.27 ptwXY_setXYDataFromXsAndYs

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

C declaration:

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

`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.

5.1.28 ptwXY_deletePoints

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

C declaration:

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

`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 ($0 \leq \mathrm{i1} \leq \mathrm{i2} \leq n$) 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.

5.1.29 ptwXY_getPointAtIndex

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

C declaration:

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

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

5.1.30 ptwXY_getPointAtIndex_Unsafely

This routine returns the point at index. This routine 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 routine.

C declaration: — This routine 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.

5.1.31 ptwXY_getXYPairAtIndex

This routine 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:

```
ptwXYPoint *ptwXY_getPairAtIndex( ptwXYPoints *ptwXY,
                                  int64_t index,
                                  double *x,
                                  double *y );
```

`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.

5.1.32 ptwXY_getPointsAroundX

This routine sets the **lessThanEqualXPoint** and **greaterThanXPoint** members of the **ptwXY** object to the two points that bound a point x.

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

```
ptwXY_lessEqualGreaterX ptwXY_getPointsAroundX( ptwXYPoints *ptwXY,
                                                double x );
```

`ptwXY`: A pointer to the **ptwXYPoints** object.
`x`: The x value.

If the **ptwXY** object is empty then the return value is **ptwXY_lessEqualGreaterX_empty**. If x is less than xMin, then **ptwXY_lessEqualGreaterX_lessThan** is return. If x is greater than xMax, 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.

5.1.33 ptwXY_getValueAtX

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

C declaration:

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

`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**.

5.1.34 ptwXY_setValueAtX

This routine 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( ptwXYPoints *ptwXY,
                              double x,
                              double y );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`x:` The x value.
`y:` The y value.

5.1.35 ptwXY_setXYPairAtIndex

This routine sets the x and y values at index.

C declaration:

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

`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. $x \le x_{\text{index}-1}$ or $x \ge x_{\text{index}+1}$) then **nfu_badIndexForX** is return.

5.1.36 ptwXY_getSlopeAtX

This routine 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( ptwXYPoints *ptwXY,
                              double x,
                              const char side,
                              double *slope );
```

`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.

5.1.37 ptwXY_getXMinAndFrom — Not for general use

This routine returns the xMin value and indicates whether the minimum value resides in the primary or secondary cache.

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

```
double ptwXY_getXMinAndFrom( ptwXYPoints *ptwXY,
                             ptwXY_dataFrom *dataFrom );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`dataFrom:` The output of this argument indicates which cache the minimum value resides in.

The return value from this routine is xMin. 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.

5.1.38 ptwXY_getXMin

This routine returns the xMin value returned by **ptwXY_getXMinAndFrom**. The calling routine 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:

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

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

5.1.39 ptwXY_getXMaxAndFrom — Not for general use

This routine returns the xMax value and indicates whether the maximum value resides in the primary or secondary cache.

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

```
double ptwXY_getXMaxAndFrom( ptwXYPoints *ptwXY,
                             ptwXY_dataFrom *dataFrom );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`dataFrom:` The output of this argument indicates which cache the maximum value resides in.

The return value from this routine is xMax. If there are no data in the **ptwXYPoints** object, then **dataFrom** is set to **ptwXY_dataFrom_Unknown**. Otherwise, it is set to **ptwXY_data-**

From_Points or **ptwXY_dataFrom_Overflow** if the maximum value is in the primary or secondary cache respectively.

5.1.40 ptwXY_getXMax

This routine returns the xMax value returned by **ptwXY_getXMinAndFrom**. The calling routine 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:

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

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

5.1.41 ptwXY_getYMin

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

C declaration:

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

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

5.1.42 ptwXY_getYMax

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

C declaration:

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

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

5.1.43 ptwXY_initialOverflowPoint — Not for general use

This routine initializes a point in the secondary cache.

C declaration: — This routine 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.

5.2 Methods

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

5.2.1 ptwXY_clip

This routine clips the y-values of **ptwXY** to be within the range **yMin** and **yMax**.

C declaration:

```
fnu_status ptwXY_clip( ptwXYPoints *ptwXY,
                       double yMin,
                       double yMax );
```

`ptwXY:`	A pointer to the **ptwXYPoints** object.
`yMin:`	All y-values in **ptwXY** will be greater than or equal to this value.
`yMax:`	All y-values in **ptwXY** will be less than or equal to this value.

5.2.2 ptwXY_thicken

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

C declaration:

```
fnu_status ptwXY_thicken( ptwXYPoints *ptwXY,
                          int sectionSubdivideMax,
                          double dxMax,
                          double fxMax );
```

`ptwXY:`	A pointer to the **ptwXYPoints** object.
`sectionSubdivideMax:`	The maximum number of points to add between two initial consecutive points.
`dxMax:`	The desired maximum absolute x step between consecutive points.
`fxMax:`	The desired maximum relative x step between consecutive points.

This routine adds points so that $x_{j+1} - x_j \le$ **dxMax** and $x_{j+1}/x_j \le$ **fxMax** but will never add more then **sectionSubdivideMax** points bewteen any of the orginal points. If **sectionSubdivideMax** < 1 or **dxMax** < 0 or **fxMax** < 1, the error **nfu_badInput** is return.

5.2.3 ptwXY_thin

This routine thins (i.e., removes) points from **ptwXY** while maintaining interpolation **accuracy** with **ptwXY**.

C declaration:

```
ptwXPoints *ptwXY_thin( ptwXYPoints *ptwXY,
                        double accuracy,
                        nfu_status *status );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`accuracy:` The accuracy of the thinned **ptwXYPoints** object.
`status:` On return, the status value.

5.2.4 ptwXY_trim

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

C declaration:

```
fnu_status ptwXY_trim( ptwXYPoints *ptwXY );
```

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

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

5.2.5 ptwXY_union

This routine 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( ptwXYPoints *ptwXY1,
                          ptwXYPoints *ptwXY2,
                          fnu_status *status,
                          int unionOptions );
```

`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`status:` On return, the status value.
`unionOptions:` Specifies options (see below).

If an error occurs, NULL is returned. The default behavior of this routine 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.

5.3 Unitary operators

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

5.3.1 ptwXY_abs

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

C declaration:

```
fnu_status ptwXY_abs( ptwXYPoints *ptwXY );
```

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

5.3.2 ptwXY_neg

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

C declaration:

```
fnu_status ptwXY_neg( ptwXYPoints *ptwXY );
```

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

5.4 Binary operators

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

5.4.1 ptwXY_slopeOffset

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

C declaration:

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

`ptwXY:` A pointer to the **ptwXYPoints** object.
`slope:` The slope.
`offset:` The offset.

5.4.2 ptwXY_add_double

This routine applies the math operation ($y_i = y_i + \text{offset}$) to the y-values of **ptwXY**.

C declaration:

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

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

5.4.3 ptwXY_sub_doubleFrom

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

C declaration:

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

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

5.4.4 ptwXY_sub_fromDouble

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

C declaration:

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

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

5.4.5 ptwXY_mul_double

This routine applies the math operation (y_i = slope × y_i) to the y-values of **ptwXY**.

C declaration:

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

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

5.4.6 ptwXY_div_doubleFrom

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

C declaration:

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

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

C declaration:

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

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

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

5.4.8 ptwXY_mod

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

$$y_i = \mathrm{mod}(y_i, m) \quad . \tag{1}$$

C declaration:

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

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

$$y_i(x_i) = s_1 \times y_1(x_i) + s_2 \times y_2(x_i) + s_{12} \times y_1(x_i) \times y_2(x_i) \tag{2}$$

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 routine 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( ptwXYPoints *ptwXY1,
                                 ptwXYPoints *ptwXY2,
                                 double s1,
                                 double s2,
                                 double s12,
                                 fnu_status *status );
```

`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}.
`status:`	On return, the status value.

5.4.10 ptwXY_add_ptwXY

This routine 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( ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2,
                              fnu_status *status );
```

`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`status:` On return, the status value.

5.4.11 ptwXY_sub_ptwXY

This routine 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( ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2,
                              fnu_status *status );
```

`ptwXY1:` A pointer to a **ptwXYPoints** object which is the minuend.
`ptwXY2:` A pointer to a **ptwXYPoints** object which is the subtrahend.
`status:` On return, the status value.

5.4.12 ptwXY_mul_ptwXY

This routine 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 $s_{12} = 1.$).

C declaration:

```
ptwXYPoints *ptwXY_mul_ptwXY( ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2,
                              fnu_status *status );
```

`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`status:` On return, the status value.

5.4.13 ptwXY_mul2_ptwXY

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

C declaration:

```
ptwXYPoints *ptwXY_mul2_ptwXY( ptwXYPoints *ptwXY1,
                               ptwXYPoints *ptwXY2,
                               fnu_status *status );
```

`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`status:` On return, the status value.

5.4.14 ptwXY_div_ptwXY

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

C declaration:

```
ptwXYPoints *ptwXY_div_ptwXY( ptwXYPoints *ptwXY1,
                              ptwXYPoints *ptwXY2,
                              fnu_status *status,;
                              int safeDivide );
```

`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`status:` On return, the status value.
`safeDivide:` If true safe division is performed.

5.5 Functions

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

5.5.1 ptwXY_pow

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

C declaration:

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

`ptwXY:` A pointer to the **ptwXYPoints** object.
`p:` The exponent.

This routine infills to maintain the initial accuracy.

5.5.2 ptwXY_exp

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

C declaration:

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

`ptwXY:` A pointer to the **ptwXYPoints** object.
`a:` The exponent coefficient.

This routine infills to maintain the initial accuracy.

5.5.3 ptwXY_convolution

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

C declaration:

```
ptwXYPoints *ptwXY_convolution( ptwXYPoints *ptwXY1,
                                ptwXYPoints *ptwXY2,
                                nfu_status *status,
                                int mode );
```

`ptwXY1:` A pointer to a **ptwXYPoints** object.
`ptwXY2:` A pointer to a **ptwXYPoints** object.
`status:` On return, the status value.
`mode:` Flag to determine the initial x-values for calculating the convolutions.

User should set **mode** to 0.

5.6 Interpolation

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

5.6.1 ptwXY_interpolatePoint

This routine interpolates an x value between the points (x1,y1) and (x2,y2) to obtain its y value.

C declaration:

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

`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 routine returns a linear-linear interpolated representation of **ptwXY**.

C declaration:

```
ptwXYPoints *ptwXY_flatInterpolationToLinear( ptwXYPoints *ptwXY,
                                              double lowerEps,
                                              double upperEps,
                                              nfu_status *status );
```

`ptwXY:`	A pointer to a **ptwXYPoints** object.
`lowerEps:`	The amount to adjust every interior point down in x.
`upperEps:`	The amount to adjust every interior point up in x
`status:`	On return, the status value.

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

For every interior point (i.e., x_i, y_i for $0 < i < n - 1$ where n is the number of points), two points are 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_bad-Input** and NULL is returned. This condition is also returned if either **lowerEps** or **up-perEps** 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 3.

5.6.3 ptwXY_toOtherInterpolation

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

C declaration:

```
ptwXYPoints *ptwXY_toOtherInterpolation( ptwXYPoints *ptwXY,
                                         ptwXY_interpolation interpolation,
                                         double accuracy,
                                         nfu_status *status );
```

`ptwXY:`	A pointer to a **ptwXYPoints** object.
`interpolation:`	The interpolation to convert to.
`accuracy:`	The accuracy of the conversion.
`status:`	On return, the status value.

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

5.6.4 ptwXY_toUnitbase

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

C declaration:

```
ptwXYPoints *ptwXY_toUnitbase( ptwXYPoints *ptwXY,
                               nfu_status *status );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`status:` On return, the status value.

Unitbasing maps the domain to 0 to 1 by scaling each x-value as $x_i = (x_i - x_0)/(x_{n-1} - x_0)$ and scaling each y-value as $y_i = y_i \times (x_{n-1} - x_0)$. Unitbasing is most useful on pdf's.

5.6.5 ptwXY_fromUnitbase

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

C declaration:

```
ptwXYPoints *ptwXY_fromUnitbase( ptwXYPoints *ptwXY,
                                 double xMin,
                                 double xMax,
                                 nfu_status *status );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`xMin:` The lower domain for the returned **ptwXYPoints** instances.
`xMax:` The upper domain for the returned **ptwXYPoints** instances.
`status:` On return, the status value.

Each x-value is scaled as $x_i = (\text{xMax} - \text{xMax}) \times x_i + \text{xMax}$ and each y-value is scaled as $y_i = y_i/(\text{xMax} - \text{xMax})$.

5.6.6 ptwXY_unitbaseInterpolate

This routine 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:

```
nfu_status ptwXY_unitbaseInterpolate( double w,
                                      double w1,
                                      ptwXYPoints *ptwXY1,
                                      double w2,
                                      ptwXYPoints *ptwXY2,
                                      nfu_status *status );
```

`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.
`status:`	On return, the status value.

5.7 Integration

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

5.7.1 ptwXY_f_integrate

This routine returns the integral bewteen two points.

C declaration:

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

`ptwXY:`	A pointer to a **ptwXYPoints** object.
`interpolation:`	The interpolation bewteen 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 routine returns the integral of **ptwXY** from **xMin** to **xMax**.

C declaration:

```
ptwXPoints *ptwXY_integrate( ptwXYPoints *ptwXY,
                             double xl,
                             double xu,
                             nfu_status *status );
```

`ptwXY:`	A pointer to the **ptwXYPoints** object.
`xl:`	The lower limit of integration.
`xu:`	The upperlimit of integration.
`status:`	On return, the status value.

The return value is $\int_{\mathrm{xl}}^{\mathrm{xu}} f(x)dx$.

5.7.3 ptwXY_integrateDomain

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

C declaration:

```
ptwXPoints *ptwXY_integrateDomain( ptwXYPoints *ptwXY,
                                    nfu_status *status );
```

`ptwXY`: A pointer to the **ptwXYPoints** object.
`status`: On return, the status value.

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

5.7.4 ptwXY_normalize

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

C declaration:

```
ptwXPoints *ptwXY_normalize( ptwXYPoints *ptwXY );
```

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

5.7.5 ptwXY_integrateDomainWithWeight_x

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

C declaration:

```
ptwXPoints *ptwXY_integrateDomainWithWeight_x( ptwXYPoints *ptwXY,
                                                nfu_status *status );
```

`ptwXY`: A pointer to the **ptwXYPoints** object.
`status`: On return, the status value.

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

5.7.6 ptwXY_integrateWithWeight_x

This routine returns the integral of **ptwXY** weighted by x from xMin to xMax.

C declaration:

```
ptwXPoints *ptwXY_integrateWithWeight_x( ptwXYPoints *ptwXY,
                                          double xMin,
                                          double xMax,
                                          nfu_status *status );
```

ptwXY: A pointer to the **ptwXYPoints** object.
xMin: The lower limit of the integration.
xMax: The upper limit of the integration.
status: On return, the status value.

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

5.7.7 ptwXY_integrateDomainWithWeight_sqrt_x

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

C declaration:

```
ptwXPoints *ptwXY_integrateDomainWithWeight_sqrt_x( ptwXYPoints *ptwXY,
                                                    nfu_status *status );
```

ptwXY: A pointer to the **ptwXYPoints** object.
status: On return, the status value.

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

5.7.8 ptwXY_integrateWithWeight_sqrt_x

This routine returns the integral of **ptwXY** weighted by x from xMin to xMax.

C declaration:

```
ptwXPoints *ptwXY_integrateWithWeight_sqrt_x( ptwXYPoints *ptwXY,
                                              double xMin,
                                              double xMax,
                                              nfu_status *status );
```

ptwXY: A pointer to the **ptwXYPoints** object.
xMin: The lower limit of the integration.
xMax: The upper limit of the integration.
status: On return, the status value.

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

5.7.9 ptwXY_groupOneFunction

This routine 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( ptwXYPoints *ptwXY,
                                    ptwXPoints *groupBoundaries,
                                    ptwXY_group_normType normType,
                                    ptwXPoints *norm,
                                    nfu_status *status );
```

`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**.
`status:`	On return, the status value.

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

$$I_i = \frac{1}{n_i} \int_{x_i}^{x_{i+1}} f(x) dx \tag{3}$$

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 routine 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( ptwXYPoints *ptwXY1,
                                     ptwXYPoints *ptwXY2,
                                     ptwXPoints *groupBoundaries,
                                     ptwXY_group_normType normType,
                                     ptwXPoints *norm,
                                     nfu_status *status );
```

`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**.
`status:`	On return, the status value.

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

$$I_i = \frac{1}{n_i} \int_{x_i}^{x_{i+1}} f(x)\ g(x) dx \tag{4}$$

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 routine 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( ptwXYPoints *ptwXY1,
                                       ptwXYPoints *ptwXY2,
                                       ptwXYPoints *ptwXY3,
                                       ptwXPoints *groupBoundaries,
                                       ptwXY_group_normType normType,
                                       ptwXPoints *norm,
                                       nfu_status *status );
```

`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**.
`status:`	On return, the status value.

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

$$I_i = \int_{x_i}^{x_{i+1}} \frac{f(x)g(x)h(x)}{n_i} dx \tag{5}$$

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 routines in the file "ptwXY_convenient.c".

5.8.1 ptwXY_getXArray

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

C declaration:

```
ptwXPoints *ptwXY_getXArray( ptwXYPoints *ptwXY,
                             fnu_status *status );
```

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

Returns NULL if an error occured.

5.8.2 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( ptwXYPoints *ptwXY,
                            double lowerEps,
                            double upperEps,
                            int positiveXOnly );
```

`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.

5.8.3 ptwXY_mergeClosePoints

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

C declaration:

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

`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** $\times (x_i + x_{i+1})/2$.

5.8.4 ptwXY_intersectionWith_ptwX

This routine 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( ptwXYPoints *ptwXY,
                             ptwXPoints *ptwX,
                             nfu_status *status );
```

`ptwXY:` A pointer to the **ptwXYPoints** object.
`ptwX:` A pointer to the **ptwXPoints** object.
`status:` On return, the status value.

5.8.5 ptwXY_areDomainsMutual

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

C declaration:

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

`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.

5.8.6 ptwXY_mutualifyDomains

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

C declaration:

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

`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**.

5.8.7 ptwXY_copyToC_XY

This routine 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( ptwXYPoints *ptwXY,
                             int64_t index1,
                             int64_t index2,
                             int64_t allocatedSize,
                             int64_t numberOfPoints,
                             double *xys );
```

`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 $\times$ sizeof(double) $\times$ **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;

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

5.8.8 ptwXY_valueTo_ptwXY

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

C declaration:

```
ptwXYPoints *ptwXY_valueTo_ptwXY( ptwXY_interpolation interpolation,
                                  double x1,
                                  double x2,
                                  double y,
                                  fnu_status *status );
```

`interpolation:`	Type of interpolation to perform (see Section 4.3).
`x1:`	x value for the lower point.
`x2:`	x value for the upper point.
`y:`	y value for both points.
`status:`	On return, the status value.

If an error occurs, NULL is returned.

5.8.9 ptwXY_createGaussianCenteredSigma1

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

C declaration:

```
ptwXYPoints *ptwXY_createGaussianCenteredSigma1( double accuracy,
                                                  nfu_status *status );
```

`accuracy:` The returned points are accurate to accuracy.
`status:` On return, the status value.

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

5.8.10 ptwXY_createGaussian

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

C declaration:

```
ptwXYPoints *ptwXY_createGaussian( double accuracy,
                                   double xCenter,
                                   double sigma,
                                   double amplitude,
                                   double xMin,
                                   double xMax,
                                   double dullEps,
                                   nfu_status *status );
```

`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.
`xMin:`	The lower domain of the returned Gaussian.
`xMax:`	The upper lower domain of the returned Gaussian.
`dullEps:`	Currently not implemented.
`status:`	On return, the status value.

In the equation a = **amplitude**, c = **xCenter** and s = **sigma**. This routine calls **ptwXY_create-GaussianCenteredSigma1** and then scales the x and y values.

5.9 Miscellaneous

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

5.9.1 ptwXY_update_biSectionMax — Not for general use

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

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

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

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

5.9.2 ptwXY_applyFunction

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

C declaration:

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

`ptwXY:` A pointer to the **ptwXYPoints** object.
`func:` A function called to calculate func(x_i, y_i).
`argList:` A pointer passed to **func**.

This routine infills to maintain the initial accuracy. The function **func** called as if defined as

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

5.9.3 ptwXY_showInteralStructure — Not for general use

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

C declaration: — This routine 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.

5.9.4 ptwXY_simpleWrite

This routine 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**.

5.9.5 ptwXY_simplePrint

This routine 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

$$y = \frac{y_2(x - x_1) + y_1(x_2 - x)}{(x_2 - x_1)} \tag{6}$$

Lin-log interpolation

$$y = y_1 \left(\frac{y_2}{y_1}\right)^{\frac{x - x_1}{x_2 - x_1}} \tag{7}$$

Log-lin interpolation

$$y = \frac{y_1 \log(x_2/x) + y_2 \log(x/x_1)}{\log(x_2/x_1)} \tag{8}$$

Log-log interpolation

$$y = y_1 \left(\frac{x}{x_1}\right)^{\frac{\log(y_2/y_1)}{\log(x_2/x_1)}} \tag{9}$$

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:

$$\frac{d((y' - y)/y)}{dx} = \frac{d(y'/y - 1)}{dx} = \frac{d(y'/y)}{dx} = \frac{1}{y^2}\left(y\frac{dy'}{dx} - y'\frac{dy}{dx}\right) = 0 \tag{10}$$

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_inter-pol-ation-LinLin** (hence called lin–lin).

From Eq. 10 the maximum of the relative error occurs where,

$$\frac{1}{y}\left(\frac{dy'}{dx} - \frac{y'}{y}\frac{dy}{dx}\right) = \left(\frac{1}{y}\right)\left\{\frac{y_2 - y_1}{x_2 - x_1} - \left(\frac{y'}{x}\right)\frac{\log(y_2/y_1)}{\log(x_2/x_1)}\right\} = 0 \tag{11}$$

The solution is

$$\frac{x}{x_1} = \frac{a(x_2/x_1 - y_2/y_1)}{(1 - a)(y_2/y_1 - 1)} \tag{12}$$

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