

UNIVERSITY OF HAWAII AT MĀNOA

Institute for Astronomy

Pan-STARRS Project Management System

Pan-STARRS Image Processing Pipeline Algorithm Design Description

Grant Award No.	: F29601-02-1-0268
Prepared For	: Pan-STARRS Team
Prepared By	: Eugene Magnier, Paul Price, Joshua Hoblitt, Robert Lupton
Document No.	: PSDC-430-006-06
Document Date	: September 9, 2004
Revision	: 06

DISTRIBUTION STATEMENT

Approved for Public Release – Distribution is Unlimited

©Institute for Astronomy, University of Hawaii
2680 Woodlawn Drive, Honolulu, Hawaii 96822
An Equal Opportunity/Affirmative Action Institution

Submitted By:

[Insert Signature Block of Authorized Developer Representative]

Date

Approved By:

[Insert Signature Block of Customer Developer Representative]

Date

Revision History

Revision Number	Release Date	Description
00	2004 Mar 11	Hacking
01	2004 May 21	Added section on 2D Chebyshev fitting.
02	2004 Jun 22	modified stats specification
03–05	???	???
06	2004 Sep 7	Frozen for PSLib-2
07		See change log in appendix

Referenced Documents

Internal Documents

Reference	Title
PSDC-430-xxx	PS-1 Design Reference Mission
PSDC-430-004	Pan-STARRS IPP C Code Conventions
PSDC-430-005	Pan-STARRS IPP SRS
PSDC-430-006	Pan-STARRS IPP ADD
PSDC-430-008	Pan-STARRS IPP Architecture SDR

External Documents

Reference	Title
Posix Standard	Open Group Based Specifications Issue 6, IEEE Std 1003.1, 2003
SLALIB Positional Astronomy Library	http://star-www.rl.ac.uk/star/docs/sun67.htx/sun67.html
Numerical Recipes (NR)	
Knuth, D.E.	Sorting and Searching; The Art of Computer Programming
Sedgewick, R.	Algorithms, Ch. 8
Sorting Summary	http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/algoen
GSL	
FFTW	http://www.fftw.org (Fastest Fourier Transform in the West)
FITS Projection Article	http://www.cv.nrao.edu/fits/documents/wcs/wcs.all.ps Greisen & Calabretta (1995, ADASS, 4, 233)
Hipparcos and Tycho Catalogues	http://astro.estec.esa.nl/Hipparcos/CATALOGUE_VOL1/catalog_vol1.h
Zombeck	“Handbook of Space Astronomy and Astrophysics”, second edition, http://ads.harvard.edu/books/hsaa/toc.html
Reingold & Dershowitz	“Calendrical Calculations: The Millenium Edition”, Cambridge University Press, 2002.

Contents

1	Pan-STARRS Library PSLib	1
1.1	Math Utilities	1
1.1.1	Sorting	1
1.1.2	Smoothing: Boxcar and Gaussian	1
1.1.3	Statistics	2
1.1.4	Matrix Operations	4
1.1.5	Fitting	7
1.1.6	Non-linear Minimization	8
1.1.7	Polynomials	9
1.1.8	(Fast) Fourier Transforms	10
1.2	Astronomy Utilities	11
1.2.1	Time	12
1.2.2	Astronomical Image Manipulations	16
1.2.3	Celestial Coordinate Conversions	19
1.2.4	Projections	21
1.2.5	Offset	24
1.2.6	Tangent Plane to Sky	25
1.2.7	The One-to-Many Problem with Mosaic Cameras	26
1.2.8	General Astronomy Functions	26
1.2.9	Positions of Major Solar System Objects	26
1.2.10	Missing and Todo	27
A	Change Log	27
A.1	Changes from version 06 to version 07	27

1 Pan-STARRS Library PSLib

1.1 Math Utilities

1.1.1 Sorting

A variety of sorting algorithms exist, with a wide range in speed for different set sizes. Three of the best sorting algorithms are “heapsort”, “quicksort”, and “mergesort” (see, e.g., knuth, sedgewick, press). These three sorting algorithms all run in a time of $O(n \log n)$ on the average, but have different worse-case run times. Implementations of all three sorting algorithms are described in references above and in various places online (e.g., <http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/algoen.htm>). The linux `qsort` function uses a heapsort if it can allocate a sufficiently large temporary buffer, and otherwise resorts to a quicksort in-place. The GSL function `gsl_heapsort` uses the heapsort algorithm. Both of these implemenations require a pointer to the comparison function, which may result in a slower code than if the comparison were done within the sort function.

For PSLib, the sorting functions shall be implemented via the system `qsort`. The function `psSort` shall return the sorted result of the input array without over-writing the input array. The function `psSortIndex` shall return an integer index to the sequence of the input array without overwriting the input array. Given the following line of code:

```
out = psSortIndex (NULL,&in);}
```

the elements of the array `out` are in the sequence `in.arr[out->arr[0]]` to `in.arr[out->arr[in.n - 1]]`.

1.1.2 Smoothing: Boxcar and Gaussian

Smoothing may occasionally be performed on data. We present the algorithms for two typical versions: boxcar and Gaussian smoothing. In both smoothing techniques, given a series of data values f_i , the smoothed values g_i are determined by calculating a linear combination based on the input data point and its nearest $2N$ neighbors in the form:

$$g_i = \sum_{n=-N}^N c_n f_i \quad (1)$$

where the values of c_n determine the filter type. For boxcar smoothing, the values c_n are constant, and must be equal to $1/(2N + 1)$ to maintain the zeroth moment of the data (care must be taken at the ends of the data range to reduce the value of c_n as fewer input data points may be used). For Gaussian smoothing, the crucial parameter is σ , the standard deviation. The value of N should be chosen to be large enough to sample the Gaussian, $N = 5\sigma$, and the values of c_n are then just the Gaussian curve:

$$c_n = \frac{e^{-\frac{n^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \quad (2)$$

1.1.3 Statistics

The general statistics function `psStats` performs a variety of statistical calculations on a collection of floating point numbers. These statistics include both sample values, in which the formal statistic is calculated for the complete sample, and robust values, in which the statistic is estimated on the basis of an assumption of an underlying population. While more reliable in general, the robust estimators may be somewhat slower than the sample statistic, so their use may vary depending on the context. In addition, certain sigma-clipped values are defined as an intermediate choice between the sample and robust estimators.

1.1.3.1 Sample Statistics

We define the following statistical terms, assuming there is a set of data elements x_i .

Mean

The mean is defined as:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \quad (3)$$

Median

The median is defined as the value for which 50% of the data values are larger and 50% are smaller. For a sample, the values are sorted and the median is given by the value of the middle element, if the number of values is odd, and by the average of the two middle values if the number of values is even. This median should be avoided for samples which are large (e.g., $N > 10^4$ elements) as the basic robust median is quicker and more accurate.

Upper and Lower Quartiles

The upper and lower quartiles ($U_{\frac{1}{4}}$ and $L_{\frac{1}{4}}$) are first and last quarter equivalents to the median. The upper quartile is the value for which 25% of the data are larger and 75% are smaller. The lower quartile is the value for which 75% of the data are larger and 25% are smaller. For a sample, the values are sorted and the quartiles are given by the values of elements $N/4$ and $3N/4$. For the purpose of a sample upper and lower quartile, it is sufficient to provide the closest integer entry to these values. The sample quartiles should be avoided for samples which are large (e.g., $N > 10^4$ elements) as the robust quartiles are quicker and more accurate.

Standard Deviation

The standard deviation of the sample is given by:

$$\sigma = \sqrt{\sum_{i=1}^N \frac{(x_i - \bar{x})^2}{N - 1}} \quad (4)$$

To minimize the numerical rounding error, this should be calculated numerically as:

$$\sigma = \sqrt{\frac{1}{N - 1} \left[\sum_{i=1}^N (x_i - \bar{x})^2 - \frac{1}{N} \left(\sum_{i=1}^N (x_i - \bar{x}) \right)^2 \right]} \quad (5)$$

1.1.3.2 Clipped Statistics

The clipped statistics are used to determine the mean and standard deviation of a distribution in the presence of outliers. The clipped statistics are quicker than the complete robust statistical estimators and more reliable than the sample statistics. The clipped statistics algorithm produces the clipped mean and clipped standard deviation. The algorithm has 2 parameters: N , the number of iterations and k , the multiplying factor of the standard deviation used to exclude outliers. Typical values for both N and k are 3. The algorithm is as follows:

1. Compute the sample median. The number of data points must be limited to 10000; the input dataset must be randomly subsampled if more data points are used.
2. Compute the sample standard deviation.
3. Use the sample median as the first estimator of the mean, $\bar{x}$.
4. Use the sample standard deviation as the first estimator of the true standard deviation, σ .
5. Repeat the following N times:
 - (a) Exclude all values x_i for which $|x_i - \bar{x}| > k\sigma$.
 - (b) Compute the mean and standard deviation of the sub-sample.
 - (c) Use the new mean for $\bar{x}$.
 - (d) Use the new standard deviations for σ .
6. The last calculated value of $\bar{x}$ is the clipped mean.
7. The last calculated value of σ is the clipped standard deviation.

1.1.3.3 Robust Statistics

The robust version of the statistics provides estimators of basic statistical concepts which are reliable even for data samples with significant contamination. A typical case is the situation in which the data of interest represent a primary population of interest with a single-valued mean and standard deviation and a secondary population of data with a substantially different distribution. For example, an image of an uncrowded night-time field may consist of a sparse collection of stars and an overall background level. The majority of pixels have the background value, with some variance due to noise sources, but many are significantly higher (contributed by stars) or significantly lower (dead pixels). If we want to measure the mean of the background, a robust mean is necessary as the outliers will strongly bias the sample statistics.

The robust statistics are calculated by constructing a histogram of the values and performing the measurements on the histogram. The choice of a bin size requires some care. If the data are integer valued, the natural bin size is an integer. Otherwise, the bin should be a fraction of an estimate of the standard deviation. Use the 3σ clipped standard deviation as an estimator of the standard deviation. The bin size shall be set at $\sigma_e/10$. The remaining steps of the algorithm are as follows:

- Construct the histogram with the specified bin size.
- Smooth the histogram by a Gaussian with $\sigma_s = \sigma_e/4$.
- Find the bin with the peak value in the range $L_{\frac{1}{4}}$ to $U_{\frac{1}{4}}$; this is the robust mode, mode_r .
- Determine $dL = (U_{\frac{1}{4}} - L_{\frac{1}{4}})/4$.
- Fit a Gaussian to the bins in the range $\text{mode}_r - dL$ to $\text{mode}_r + dL$.
- The resulting fit parameters are the robust mean, mean_r and the robust standard deviation, σ_r .

To determine the robust median, construct the cumulative histogram from the histogram above. Select the bin which contains the 50th percentile value and its two neighbors. Fit a quadratic to these three points. The robust median value is the coordinate of the quadratic which returns the 50% value. For the upper and lower quartile points, the same process should be used, choosing the three bins in the vicinity of the upper and lower quartile points.

1.1.4 Matrix Operations

In this section, we define the linear algebra operations performed on matrices. We have defined APIs to implement the following matrix functions:

- Invert a matrix;
- Calculate a matrix determinant;
- Perform matrix addition, subtraction and multiplication;

- Transpose a matrix; and
- Convert a matrix to a vector.

Many of these operations are implemented using LU decomposition. We define LU decomposition in the following paragraph. Implementation of LU decomposition shall make use of the GSL function `gsl_linalg_LU_decomp`.

1.1.4.1 LU Decomposition

We wish to decompose the matrix A with elements a_{ij} into diagonal matrices that satisfy the relationship $A = LU$ where L is a lower-diagonal matrix of the form:

$$L = \begin{pmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix} \quad (6)$$

(where all diagonal values $\alpha_{ii} = 1$) and U is an upper-diagonal matrix of the form:

$$U = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix} \quad (7)$$

We can find the values of α_{ij} and β_{ij} by following this procedure. First, $\alpha_{ii} = 1$. For all values of $j = 1, 2, \dots, N$, follow the next steps: For each value of $i = 1, 2, \dots, j$, solve for β_{ij} using the relationship:

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik} \beta_{kj} \quad (8)$$

For the values of $i = j + 1, j + 2, \dots, N$, solve for the values of α_{ij} with the following:

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik} \beta_{kj} \right) \quad (9)$$

1.1.4.2 Calculate a matrix determinant

The determinant D of a matrix a_{ij} is calculated from the product of the diagonal elements of the LU decomposition:

$$D = P_{i=1} N U_{ii} \quad (10)$$

This shall be calculated using the GSL function `gsl_linalg_LU_det`. If the matrix is large or the magnitude of the elements is large, the determinant may overflow the data type. In these cases, the (natural) logarithm of the determinant may be calculated instead (as the sum of the logarithms of the diagonal elements). In this case, the GSL function `gsl_linalg_LU_lndet` shall be used.

1.1.4.3 Solving a Linear Equation

The LU decomposition of a matrix may be used to solve the matrix equation $\sum_{j=1}^N A_{i,j} \times x_j = B_j$

Given the LU decomposition of a matrix into α_{ij} and β_{ij} , the technique of back-substitution is used to solve the equation above. First solve the equation $Ly = B$:

$$y_1 = \frac{b_1}{\alpha_{11}} \quad (11)$$

$$y_i = \frac{1}{\alpha_{ii}} \left(b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_j \right) \quad (12)$$

The values of y may be used to solve for x_i using the relationship $Ux = b$:

$$x_N = \frac{y_N}{\beta_{NN}} \quad (13)$$

$$x_i = \frac{1}{\beta_{ii}} \left(y_i - \sum_{j=i+1}^N \beta_{ij} y_j \right) \quad (14)$$

1.1.4.4 Invert a matrix

Inversion of a matrix using the LU decomposition is performed by performing back-substitution to solve a series of linear equations of the form $\sum_{j=1}^N A_{i,j} \times x_j = B_j$ where the values of B_j represent the columns of the identity matrix. The solution vectors x_j represent the columns of the inverse matrix. This operation shall be implemented using the GSL function `gsl_linalg_LU_invert`.

1.1.4.5 Perform matrix addition, subtraction and multiplication

Matrix binary arithmetic operations differ from image binary arithmetic operations in a fundamental way: For image operations, the resulting pixel value is determined by performing the operation on the two corresponding input operand pixels. For matrix operations, the resulting element value results from the operation on the corresponding pair of row and column from the input matrices. So, for example, given the input matrices α_{ij} and β_{ij} , the image and matrix multiplications correspond to:

$$\text{image}_{ij} = \alpha_{ij} \times \beta_{ij} \quad (15)$$

$$\text{matrix}_{jk} = \sum_{i=1}^N \alpha_{ij} \times \beta_{ki} \quad (16)$$

The matrix and image operations for addition and subtraction are identical: the operation is performed on the corresponding elements in each matrix. Matrix division is not defined (to divide, the user should first invert the matrix, and then multiply). The matrix math function `psMatrixOp` shall implement the matrix version of the binary arithmetical operations for the following operators: $+$, $-$, $\times$.

1.1.4.6 Transpose a matrix

The transpose of a matrix is simply the reorganization of the matrix elements by 'flipping' the matrix along a diagonal. For non-square matrices with dimensions $N \times M$, the resulting matrix has dimensions $M \times N$. The element of the output matrix T_{ij} is given by

$$T_{ij} = M_{ji} \quad (17)$$

where M_{ij} is the matrix to be transposed.

1.1.4.7 Convert a matrix to a vector

Matrix-to-vector conversion is only defined for a matrix that has a size of one in at least one dimension. In that case, the elements should be extracted, and placed in a vector, with care that the `pSType` is defined correctly — a $1 \times N$ matrix is converted to a `PS_DIMEN_VECTOR`-type vector, while a $N \times 1$ matrix is converted to a `PS_DIMEN_TRANV`-type vector.

1.1.5 Fitting

1.1.5.1 Chi-squared

Given a set of N ordinates, x_i , measured coordinates, y_i , with errors, σ_i , and a model function, $f(x_i)$, then χ^2 ("chi-squared") is defined:

$$\chi^2 = \sum_{i=0}^{N-1} \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2 \quad (18)$$

1.1.5.2 General Polynomial Fitting

Given a set of data values y_i with errors σ_i , related to independent data values x_i , we would like to fit a polynomial model of N_{par} free parameters of the form $y = \sum_{j=0}^{N_{par}} \alpha_j x^j$. The model is determined by minimizing the value of χ^2 (1.1.5.1), and the minimization is determined by solving for the set of parameters α_j for which the partial derivatives of the χ^2 with respect to those parameters are zero. The partial derivatives are

$$\frac{\partial \chi^2}{\partial \alpha_k} = -2 \sum_i (y_i - \sum_j x_i^j \alpha_j) \frac{x_i^k}{\sigma_i^2} \quad (19)$$

Setting the derivatives to zero, this may be reduced to the following matrix equation:

$$\sum_j \alpha_j \sum_i \frac{x_i^k x_i^j}{\sigma_i^2} = \sum_i \frac{x_i^k y_i}{\sigma_i^2} \quad (20)$$

This matrix equation may be solved with LU Decomposition (section 1.1.4.1).

1.1.6 Non-linear Minimization

1.1.6.1 Levenberg-Marquardt Method

In the Levenberg-Marquardt Method (LMM; see NR §15.5), we make a guess at the input parameters, evaluate the function of interest, vary the parameters by a particular choice based on the gradient, evaluate the function again, and adjust the parameters and the parameter variant based on the results.

Given some constant/independent values, c_i , we would like to find the parameters a_k of the function $f(c_i; a_k)$ which minimize the function. We start with a set of parameter guesses, a_k . We calculate the gradient β_k and the Hessian matrix $\alpha_{j,k}$ at this parameter selection as follows:

$$\beta_k = \frac{\partial f(c_i)}{\partial a_k} \quad (21)$$

$$\alpha_{j,k} = \frac{\partial f(c_i)}{\partial a_k} \frac{\partial f(c_i)}{\partial a_j} \quad (22)$$

We now define the new parameter guess for a_k based on the gradient and Hessian by defining $A_{j,k}$ as a variant on $\alpha_{j,k}$ as follows:

$$A_{j,k} = \alpha_{j,k} \quad (j \neq k) \quad (23)$$

$$A_{j,k} = (1 + \lambda)\alpha_{j,k} \quad (j = k) \quad (24)$$

and solve the system of equations represented by:

$$A_{j,k} a'_k = \beta_j \quad (25)$$

where a'_k represents our new attempt at a parameter guess. We use this parameter set to evaluate the function. If the new value of the function is lower than the previous guess, we accept this new set of parameters and decrease λ by a factor of 10, otherwise we keep the old set, and increase the value of λ by a factor of 10. We repeat this process until the value of the function changes by much less than the tolerance. The resulting values of a_k are the best-fit parameters for the system.

The covariance matrix, $C_{i,j}$, which is the inverse of the matrix $\alpha_{j,k}$ allows simple calculation of the confidence limits of the parameters.

1.1.6.2 Powell's method

Powell's method is a type of "Direction Set" methods in multi-dimensions for finding a local minimum. Given a starting point (the "best guess" for the minimum) and a set of direction vectors, a direction set method advances

in the direction of the vectors, determines a new direction vector by some method, and proceeds in this manner until the advances along the vectors are smaller than some pre-defined tolerance. Such direction set methods, including Powell's Quadratically Convergent method are discussed in NR§10.5.

We will use for our algorithm the modified version of Powell's Quadratically Convergent Method, which is described below, adapted from NR.

1. Given a function in N dimensions to minimize, f , and a best guess for the minimum, point P in N dimensions, take an initial set of N vectors, v_i , to be the unit vectors.
2. Set point $Q = P$.
3. For each dimension in turn, move Q *only* in the direction v_i to minimize the function of interest.
4. Set vector $u = Q - P$.
5. Move Q *only* in the direction u
6. Replace the vector along which the largest minimization was made, $v_{i,\max}$, with u , except under either of the following circumstances:
 - If $f_Q \geq f_P$, then there is no point in keeping the new vector, because there is no further minimization to be made in that direction.
 - If $2(f_P - 2f_Q + f_{QP})[(f_P - f_Q) - \Delta_{\max}]^2 \geq (f_P - f_{QP})^2 \Delta_{\max}$, then either the decrease in the function was not due to any single direction, or we are close to the minimum.

where $f_P = f(P)$, $f_Q = f(Q)$, $f_{QP} = f(2Q - P)$, and $\Delta_{\max} \geq 0$ is the magnitude of the minimization made along $v_{i,\max}$.
7. Set P to Q .
8. Return to step 3 until the change in this last move is less than some specified tolerance, or a maximum number of iterations has been reached.

In regards to minimizing the function only in a particular direction, we shall adopt, as NR recommends, bracketing the minimum before applying Brent's method, **[which will be specified in detail later]**.

1.1.7 Polynomials

We will employ Chebyshev polynomials (NR §5.8) to approximate functions:

$$f(x) = \sum_{i=0}^n c_i T_i(x) \quad (26)$$

These have some desirable features:

- They are bounded on $-1 < x < 1$, with the maxima and minima over this range being 1 and -1 , respectively;
- Truncation of the higher-order terms leaves one with the most accurate lower-order polynomial representation of the desired function.

The first few Chebyshev polynomials are:

$$T_0(x) = 1 \quad (27)$$

$$T_1(x) = x \quad (28)$$

$$T_2(x) = 2x^2 - 1 \quad (29)$$

$$T_3(x) = 4x^3 - 3x \quad (30)$$

$$T_4(x) = 8x^4 - 8x^2 + 1 \quad (31)$$

$$(32)$$

Chebyshev polynomials follow the recurrence relation:

$$T_{n+1} = 2xT_n - T_{n-1} \quad (33)$$

Practically, Chebyshev polynomials should be evaluated using Clenshaw's recurrence formula (NR §5.5):

$$d_j = 2xd_{j+1} - d_{j+2} + c_j \quad (34)$$

$$f(x) = x * d_1 - d_2 + 1/2c_0 \quad (35)$$

$$(36)$$

It shall be the responsibility of the user to convert the domain into the range $-1 < x < 1$.

1.1.7.1 Multi-dimensional polynomials

Multi-dimensional polynomials shall be composed of multiplications of 1D Chebyshev polynomials, with the coefficients stored in tensors of the appropriate rank.

1.1.8 (Fast) Fourier Transforms

(Fast) Fourier Transforms (FFTs) shall be implemented using the *Fastest Fourier Transform in the West* (FFTW) library.

1.1.8.1 FFTW Plans

FFTW requires the user to create a “plan” for each transform size, the time to create which is a function of the desired speed of the transform — faster transforms for a given size (and machine) may be performed if more time is spent testing plans.

In the Pan-STARRS IPP, we will want to perform FFTs on images of common sizes (e.g. 512×512) regularly. This means that we would gain from determining an FFTW plan for each of these common sizes. FFTW provides a binary, `fftw-wisdom` which may be used to generate and save “wisdom”. The location of the `wisdom` file will be specified as a configuration variable for the IPP (defaulting to `/etc/fftw/wisdom`). The `wisdom` should be read in upon initialisation of the PSLib FFT functions and saved at the conclusion.

1.1.8.2 Function mapping

The forward and reverse transforms call the corresponding FFTW function to plan the transform:

PSLib function	Major FFTW call
<code>psFFTForward()</code>	<code>fftw_plan_dft_r2c_2d()</code>
<code>psFFTReverse()</code>	<code>fftw_plan_dft_c2r_2d()</code>

These plans should be formulated using the `FFTW_ESTIMATE` flag, which will allow FFTW to default to a minimal planning time if the wisdom has not been loaded. Transforms should be performed out of place to avoid the need to pad the input array to hold the output.

1.1.8.3 More Complicated Functions

`psFFTCrossCorrelate()` and `psFFTConvolve()` both involve multiplication of two Fourier transforms. In the former, the first Fourier transform is multiplied by the complex conjugate of the second Fourier transform to yield the Fourier transform of the cross-correlation (NR §13.2). In the latter, the two Fourier transforms are multiplied directly to yield the Fourier transform of the convolution (NR §13.1).

If the elements of the discrete Fourier transform are C_k , then the elements of the power spectrum are (NR §13.4):

$$P_0 = |C_0|^2 / N^2 \quad (37)$$

$$P_j = (|C_j|^2 + |C_{N-j}|^2) / N^2 \quad (38)$$

$$P_{N/2} = |C_{N/2}|^2 / N^2 \quad (39)$$

$$(40)$$

where $j = 1, 2, \dots, (N/2 - 1)$.

Note that we leave the issue of “windowing” the data up to the caller, and choose to normalise by $1/N^2$.

1.2 Astronomy Utilities

Most of the astronomy utilities will be implemented through wrapping the SLALIB Positional Astronomy Library.

1.2.1 Time

Correct time representation is critical in astronomical software. PSLib uses the `pSTime` structure to represent time values. This structure represents a time which consists of seconds and fractions of seconds in a time system defined by the `pSTimeType` element type. Two possible time systems are currently available: TAI and UTC. Both are defined in terms of the reference epoch 1970-01-01T00:00:00Z, but with minor modifications for leap seconds as needed. The first representation, TAI (International Atomic Time), has seconds of uniform length and no leap seconds. The exact zero reference is 1970/01/01,00:00:10 UTC. The second representation is UTC, which has seconds of uniform length and leap seconds as needed to adjust it to remain within 0.9 seconds of the Earth's rotation. It has a zero-point of exactly 1970/01/01,00:00:00 UTC.

1.2.1.1 Coordinated Universal Time (UTC)

Coordinated Universal Time (UTC) is a system of time with SI length seconds but attempts to stay within 1s of UT1. This is done by the insertion of leap second whenever $UTC - UT1 \geq 0.9s$. By definition $UTC - TAI$ is an integer number of seconds. UTC went into effect on "1972-01-01T00:00:00" and is defined as being $TAI - UTC = 10s$ on that date. For dates prior to 1972-01-01 a fixed offset of 10s relative to TAI will be assumed.

$$UTC = TAI - 10s - \text{leapseconds} \quad (41)$$

Leapseconds are declared by the International Earth Rotation and Reference Systems Service (IERS). Leapseconds only occur in the UTC time system and cannot be accurately predicted due to variations in the Earth's rotational period. To determine the number of leapsecond in a given UTC date a table of leapseconds as announced by the IERS must be consulted. This table will have to be updated each time a new leapsecond occurs.

For ease of conversion, UTC should be represented as the number of seconds since the UNIX epoch of "1970-01-01T00:00:00".

1.2.1.2 International Atomic Time (TAI)

International Atomic Time or Temps Atomique International (TAI) is a system of time defined by the Bureau International des Poids et Mesures (BIPM) with SI length seconds as measured at sea level. To convert from UTC to TAI add the base delta of 10s and all of the accumulated leapseconds since 1972-01-01 up until the UTC date being converted.

$$TAI = UTC + 10s + \text{leapseconds} \quad (42)$$

For ease of conversion, TAI should be represented as the number of seconds since the UNIX epoch of "1970-01-01T00:00:00".

1.2.1.3 Leap seconds

Leap seconds keep UTC within 0.9s of UT1. The offset between TAI and UTC must be looked up from tables. Jumps in the offset correspond to leap seconds.

1972	JUL	1	=JD	2441499.5	TAI-UTC=	11.0	S + (MJD - 41317.) X 0.0 S
1973	JAN	1	=JD	2441683.5	TAI-UTC=	12.0	S + (MJD - 41317.) X 0.0 S
1974	JAN	1	=JD	2442048.5	TAI-UTC=	13.0	S + (MJD - 41317.) X 0.0 S
1975	JAN	1	=JD	2442413.5	TAI-UTC=	14.0	S + (MJD - 41317.) X 0.0 S
1976	JAN	1	=JD	2442778.5	TAI-UTC=	15.0	S + (MJD - 41317.) X 0.0 S
1977	JAN	1	=JD	2443144.5	TAI-UTC=	16.0	S + (MJD - 41317.) X 0.0 S
1978	JAN	1	=JD	2443509.5	TAI-UTC=	17.0	S + (MJD - 41317.) X 0.0 S
1979	JAN	1	=JD	2443874.5	TAI-UTC=	18.0	S + (MJD - 41317.) X 0.0 S
1980	JAN	1	=JD	2444239.5	TAI-UTC=	19.0	S + (MJD - 41317.) X 0.0 S
1981	JUL	1	=JD	2444786.5	TAI-UTC=	20.0	S + (MJD - 41317.) X 0.0 S
1982	JUL	1	=JD	2445151.5	TAI-UTC=	21.0	S + (MJD - 41317.) X 0.0 S
1983	JUL	1	=JD	2445516.5	TAI-UTC=	22.0	S + (MJD - 41317.) X 0.0 S
1985	JUL	1	=JD	2446247.5	TAI-UTC=	23.0	S + (MJD - 41317.) X 0.0 S
1988	JAN	1	=JD	2447161.5	TAI-UTC=	24.0	S + (MJD - 41317.) X 0.0 S
1990	JAN	1	=JD	2447892.5	TAI-UTC=	25.0	S + (MJD - 41317.) X 0.0 S
1991	JAN	1	=JD	2448257.5	TAI-UTC=	26.0	S + (MJD - 41317.) X 0.0 S
1992	JUL	1	=JD	2448804.5	TAI-UTC=	27.0	S + (MJD - 41317.) X 0.0 S
1993	JUL	1	=JD	2449169.5	TAI-UTC=	28.0	S + (MJD - 41317.) X 0.0 S
1994	JUL	1	=JD	2449534.5	TAI-UTC=	29.0	S + (MJD - 41317.) X 0.0 S
1996	JAN	1	=JD	2450083.5	TAI-UTC=	30.0	S + (MJD - 41317.) X 0.0 S
1997	JUL	1	=JD	2450630.5	TAI-UTC=	31.0	S + (MJD - 41317.) X 0.0 S
1999	JAN	1	=JD	2451179.5	TAI-UTC=	32.0	S + (MJD - 41317.) X 0.0 S

For the present time, it should be assumed that this table resides on local disk in a known location (i.e., there is no need that it is downloaded from the internet by PSLib). Later, the location of this file will be made configurable.

This data is available from `ftp://maia.usno.navy.mil/ser7/tai-utc.dat`

1.2.1.4 Gregorian dates to seconds

The below algorithm converts from Gregorian-formatted dates to seconds since the UNIX epoch.

```

Given year, month, day.

### Make month in range 3..14 (treat Jan & Feb as months 13..14 of prev year):
if ( month <= 2 )
{
    year -= ( temp = ( 14 - month ) / 12 )
    month += 12 * temp
}
else if ( month > 14 )
{
    year += ( temp = ( month - 3 ) / 12 )
    month -= 12 * temp
}

### make year positive
if ( year < 0 )
{
    day -= 146097 * ( temp = ( 399 - year ) / 400 )
    year += 400 * temp
}

```

```

### add: day of month, days of previous 0-11 month period that began
### w/March, days of previous 0-399 year period that began w/March
### of a 400-multiple year), days of any 400-year periods before
### that, and 306 days to adjust from Mar 1, year 0-relative to Jan
### 1, year 1-relative
day += ( month * 367 - 1094 ) / 12 + year % 100 * 1461 / 4 +
      ( year / 100 * 36524 + year / 400 ) - 306

unix = ( ( day - 1 ) * 86400 ) - 62135596800
utc = unix - leapseconds(unix)

```

To go the other way:

```

unix = utc + leapseconds(utc)
day = ( unix + 62135596800 ) / 86400
temp = 0

### add 306 days to make relative to Mar 1, 0; also adjust day to be
### within a range (1..2**28-1) where our calculations will work
### with 32bit ints
if ( day > 2**28 - 307 )
{
    ### avoid overflow if day close to maxint
    temp = ( day - 146097 + 306 ) / 146097 + 1
    day -= temp * 146097 - 306
}
else if ( ( day += 306 ) <= 0 )
{
    temp = -( -day / 146097 + 1 ) ### avoid ambiguity in C division of negatives
    day -= temp * 146097
}

cent = ( day * 4 - 1 ) / 146097    ### calc number of centuries day is after 29 Feb of yr 0
day -= cent * 146097 / 4          ### (4 centuries = 146097 days)
year = ( day * 4 - 1 ) / 1461      ### calc number of years into the century,
day -= year * 1461 / 4            ### again March-based (4 yrs = 146[01] days)
month = ( day * 12 + 1093 ) / 367  ### get the month (3..14 represent March through
day -= ( month * 367 - 1094 ) / 12 ### February of following year)
year += cent * 100 + temp * 400    ### get the real year, which is off by
if ( month > 12 )                  ### one if month is January or February
{
    year++
month -= 12
}

Output year, month, day.

```

(Above taken from `DateTime.pm` (C) 2003 Dave Rolsky, available from `datetime.perl.org`.)

1.2.1.5 Universal Time (UT1)

Universal Time is a measure of the rotation angle of the Earth. When corrected for polar motion it is referred to as UT1. This is distinct from UT0 which does not involve corrections for polar motion. UT1 may be calculated from UTC through a table lookup of the appropriate value of UTC - UT1.

1.2.1.6 Julian Day and Modified Julian Day

Julian Day (JD) and Modified Julian Day (MJD) are both continuous time representations, with one julian day interval having a length of 86400 TAI seconds. MJD is equal to JD - 2400000.5 and has a zero point equal to that of TAI, while JD has a zero point 0.5 off of TAI. Conversion between psTime values and MJD and JD are determined from:

```
mjd = psTime.sec/86400.0 + psTime.usec/86400000000.0 + 40587.0;
jd = psTime.sec/86400.0 + psTime.usec/86400000000.0 + 2440587.5;
```

2451545.0 JD = 51544.5 MJD is equivalent to "2000-01-01T00:00:00".

$$JD = MJD + 2400000.5 \quad (43)$$

1.2.1.7 Terrestrial Dynamical Time (TDT)

Terrestrial Dynamical Time (TDT) is defined as a fixed offset from TAI. Its only purpose as far as we are concerned is for its utility in obtaining the GMST.

$$TDT = TAI + 32.184s \quad (44)$$

1.2.1.8 TDT as Julian Centuries since J2000.0

The algorithm for calculating GMST requires TDT formatted in Julian centuries since the J2000.0 epoch.

$$t_u = \frac{JD_{TDT} - 2451545.0}{36525} \quad (45)$$

1.2.1.9 UT1 as Julian Centuries since J2000.0

The algorithm for calculating GMST requires UT1 be formatted in Julian centuries since the J2000.0 epoch.

$$t = \frac{JD_{UT1} - 2451545.0}{36525} \quad (46)$$

1.2.1.10 Greenwich Mean Sidereal Time (GMST)

Greenwich Mean Sidereal Time (GMST) is calculated from UT1 and TDT. This algorithm requires that both time inputs are expressed as Julian centuries since J2000.0.

Here t_u is UT1 expressed in Julian centuries since J2000.0, and t is TDT expressed in Julian centuries since J2000.0.

$$\text{GMST00}(t_u, t) = \text{UT1} + 24110.5493771 \quad (47)$$

$$+ 8639877.3173760 t_u + 307.4771600 t \quad (48)$$

$$+ 0.0931118 t^2 - 0.0000062 t^3 \quad (49)$$

$$+ 0.0000013 t^4 \quad (50)$$

Gives GMST00 in seconds.

1.2.1.11 Longitude

Longitudes are often expressed in the form of decimal degrees while the algorithm for calculating GMST outputs seconds.

$$1^\circ = 240s \quad (51)$$

1.2.1.12 Local Mean Sidereal Time (LMST)

Local Mean Sidereal Time (LMST) is Greenwich Mean Sidereal Time (GMST) plus the observer's location in East longitude. Calculating LMST requires the input of Universal Time (UT1), Terrestrial Dynamical Time (TDT) and a longitude (measured East of Greenwich).

$$\text{LMST} = \text{GMST00}(t_u, t) + \text{longitude} \quad (52)$$

Gives LMST in seconds.

1.2.1.13 Polar Coordinates

The polar coordinates, x_p and y_p , required for SLA_AOPPA (and hence the `psGrommits`), may be calculated through table lookups.

1.2.2 Astronomical Image Manipulations

1.2.2.1 Interpolation

Interpolation is needed in various image manipulation operations, including rotation and resampling. We have specified a function to perform the interpolation using one of several possible interpolation methods, defined

below. It is important in the discussions that follow to remember that a pixel with column,row if i, j has coordinate at the center of $i + 0.5, j + 0.5$ and corners with coordinates from i, j to $i + 1, j + 1$. Thus, the interpolation of a coordinate $x, y = 5.0, 4.0$ is a value midway between the four pixels with column,row of (5,4), (5,5), (6,4), (6,5).

Nearest Pixel Interpolation (PS_RESAMPLE_FLAT)

In this interpolation, the value of the closest pixel is returned. This is equivalent to pixel duplication or replication.

Bilinear Interpolation (PS_RESAMPLE_BILINEAR)

In this interpolation, the value at the coordinate is calculated using linear interpolation in two dimensions from the four nearest neighbor pixels. The bilinear interpolation value at a coordinate x, y depends on the four nearest neighbor pixels and the fractional distance f_x, f_y of the given coordinates from the centers of those four pixels. Consider four neighboring pixels at column,row of $i, j, i + 1, j, i, j + 1$, and $i + 1, j + 1$ with pixel values $V_{0,0}, V_{1,0}, V_{0,1}, V_{1,1}$. The value at x, y is given by:

$$V = (V_{0,0}(1 - f_x) + V_{1,0}f_x)(1 - f_y) + (V_{0,1}(1 - f_x) + V_{1,1}f_x)f_y$$

This expression is more efficiently evaluated by factoring and calculating the expression as:

$$r_x = V_{0,0} + (V_{1,0} - V_{0,0})f_x$$

$$V = r_x + (V_{0,1} - V_{0,0})f_y + r_x f_y$$

Note that the values of f_x and f_y require some care. Given a coordinate x, y , the value of f_x is calculated as $f_x - 0.5 - \text{int}(f_x - 0.5)$. For example, when interpolating the value at (5.8, 5.2), the relevant neighbor pixels are (5,4), (6,4), (5,5), (6,5) and the fractional coordinate values $f_x, f_y = 0.3, 0.7$. The resulting coordinate would be contained within the pixel at column,row (5,5).

Sinc Interpolation (PS_RESAMPLE_SINC)

Lagrange Interpolation (PS_RESAMPLE_LAGRANGE)

1.2.2.2 Image Cuts and Slices

Several functions specify operations which manipulate a collection of pixels to return a statistic on the pixel collection. In the simplest case, these are trivial to define: if the boundaries of the region of interest are specified along integral pixel coordinates, then the pixels used to measure the statistic are always an exact integer. This is the case for the function `psImageSlice` which requires a starting coordinate which is an integer and a width in both dimensions which is an integer. For the case of the functions `psImageCut` and

`psImageRadialCut`, the situation is a bit more subtle. In both of these cases, the region is unlikely to contain only whole pixels and some choices must be made.

One possibility which we reject is to identify the fractional pixels which are overlapped by the region of interest and add that fraction of the pixel's flux when calculating the statistic of interest. This is computationally intensive, and not necessarily well defined for all statistics.

In PSLib, we instead identify the pixels overlapped by the region, use the complete set of pixel values, treating all pixels equally, and renormalize as needed. To perform this, the region of interest is laid on top of the image pixels. Any pixels which overlap the region are identified as part of the input sample. The statistic (ie, sample mean, robust mode, etc), is then calculated on this collection of pixels. If the output statistic is an average value, the measured value is reported. If the output statistic is a sum value (sum of counts, sum of pixels), then the value is renormalized by the ratio of pixels used in the calculation to the pixel area of the region of interest. For example, if the sum within a radial aperture is requested, the circle of the specified radius and center is placed on the pixel grid. Any pixels which touch the circle are then placed in a list to be analysed. The statistic of interest is the measured for this collection of pixels. In the case of a circular aperture which is centered at the coordinate (2,2) and has a radius of 2, the number of pixels which are touched by the circle is 16, while the total pixel area of the circle is 12.57 square pixels. In this case, the pixel sum is renormalized by the ratio (12.57/16.00).

Radial Cuts

Consider an image with pixels x_i, y_i and a reference coordinate x_c, y_c . We want to construct a radial cut by measuring statistics for pixels in a sequence of radial annulii $r_s < r < r_e$. For each annulus, we need to select the pixels which fall within this annulus. The coordinates of the center of pixel i, j are $i + 0.5, j + 0.5$. A given pixel has a distance from the reference coordinate of $dX = x_c - i - 0.5, dY = y_c - j - 0.5$. The pixels to be used for a given radial annulus are all of those pixels for which $r_s < \sqrt{dX^2 + dY^2} < r_e$. This is more efficiently calculated by comparing the square of the radii and distances. All pixels which satisfy the above condition are included in a specific annular radius. All average quantities are calculated directly from the pixel ensemble statistics.

Arbitrary Linear Cuts

Select the pixels which lie along a line following steps of 1 pixel length:

```

dX = xe - xs;
dY = ye - ys;
L = hypot (dX, dY);
dX = dX / L;
dY = dY / L;

REALLOCATE (xvec[0].elements, float, MAX (L, 1));
REALLOCATE (yvec[0].elements, float, MAX (L, 1));
xvec[0].Nelements = L;
yvec[0].Nelements = L;

V = (float *)buf[0].matrix.buffer;
```

```

for (i = 0; i < L; i++) {
  xi = xs + i*dX - 0.5;
  yi = ys + i*dY - 0.5;
  xvec[0].elements[i] = i;
  yvec[0].elements[i] = V[xi + Nx*yi];
}

```

1.2.2.3 Image Rotation

Image rotation can be performed in two possible ways under different circumstances, identified in the following discussion.

In the simplest case, the rotation angle is an integer multiple of 90 degrees ($\pi/2$ rad). In these cases, the input and output pixels have a one-to-one mapping. If the input image has dimensions of N_x, N_y , then the output image will have dimensions of either N_x, N_y (for even multiples of 90 degrees) or N_y, N_x (for odd multiples).

If the angle of the rotation is not a multiple of 90, then the output pixels necessarily result from the interpolation of several input pixels. In this case, for an input image of dimensions N_x, N_y and rotation angle θ , the output image has dimensions $Lx = |N_x \cos \theta| + |N_y \sin \theta|$ and $Ly = |N_x \sin \theta| + |N_y \cos \theta|$, each dimension rounded up to the nearest integer as needed. Every pixel in the output image is in general derived from an interpolation over 4 neighboring pixels. The coordinate of a pixel in the output image (i, j) corresponds to a fractional pixel coordinate (x, y) in the input image according to:

$$x = (i - i_o) * \cos \theta + (j - j_o) * \sin \theta$$

$$y = (i_o - i) * \sin \theta + (j - j_o) * \cos \theta$$

where the offset coordinate (i_o, j_o) depends on the sign of the sine of the angle θ . If the sign of that sine is positive, the offset coordinate is $(N_y \sin \theta, 0)$, otherwise it is $(0, -N_x \sin \theta)$.

1.2.3 Celestial Coordinate Conversions

Changes between spherical coordinate systems (ie, Ecliptic, Galactic, and ICRS, or Celestial, coordinates) is equivalent to a rotation in 3D. Given two coordinate system, α, δ and ϕ, θ which differ by only a rotation, the transformation between these two systems are defined by the following three parameters:

- α_p : the longitude of the target system pole in the source system
- δ_p : the latitude of the target system pole in the source system.
- ϕ_p : the longitude of the ascending node in the target system

Note that θ_p , the latitude of the source system pole in the target system, is equal to δ_p by symmetry.

The relevant trigonometric relationships are:

$$\sin \theta = \sin \delta \cos \delta_p - \cos \delta \sin \delta_p \sin(\alpha - \alpha_p) \quad (53)$$

$$\cos \theta \sin(\phi - \phi_p) = \cos \delta \cos \delta_p \sin(\alpha - \alpha_p) + \sin \delta \sin \delta_p \quad (54)$$

$$\cos \theta \cos(\phi - \phi_p) = \cos \delta \cos(\alpha - \alpha_p) \quad (55)$$

and for the inverse transformations, the equivalent relationships are:

$$\sin \delta = \sin \theta \cos \delta_p - \cos \theta \sin \delta_p \sin(\phi - \phi_p) \quad (56)$$

$$\cos \delta \sin(\alpha - \alpha_p) = \cos \theta \cos \delta_p \sin(\phi - \phi_p) + \sin \theta \sin \delta_p \quad (57)$$

$$\cos \delta \cos(\alpha - \alpha_p) = \cos \theta \cos(\phi - \phi_p) \quad (58)$$

Since θ and δ have domains of $-\pi/2, \pi/2$, the value of these angles are found by applying the arcsin to the sine of these angles ($\theta = \arcsin \sin \theta$) which is always single-valued and defined. The value of $\alpha - \alpha_p$ may be found from $\text{atan2}(y, x)$, where $y = \cos \delta \sin(\alpha - \alpha_p)$ and $x = \cos \delta \cos(\alpha - \alpha_p)$; and similarly for $\phi - \phi_p$.

1.2.3.1 Galactic to ICRS

The appropriate values, from the Hipparcos and Tycho Catalogues are:

$$\alpha_p = 282.85948^\circ \quad (59)$$

$$\delta_p = 62.87175^\circ \quad (60)$$

$$\phi_p = 32.93192^\circ \quad (61)$$

$$(62)$$

1.2.3.2 Ecliptic to ICRS

The appropriate values, from Zombeck, are:

$$\alpha_p = 0^\circ \quad (63)$$

$$\delta_p = 23^\circ 27' 8''.26 - 46''.845 T - 0''.0059 T^2 + 0''.00181 T^3 \quad (64)$$

$$\phi_p = 0^\circ \quad (65)$$

where T is the time in Julian centuries since 1900.

1.2.3.3 Precession

The appropriate values, from Elixir, are:

$$\alpha_p = 90^\circ - (0^\circ.6406161 T + 0^\circ.0000839 T^2 + 0^\circ.0000050 T^3) \quad (66)$$

$$\delta_p = 0^\circ.5567530 T - 0^\circ.0001185 T^2 - 0^\circ.0000116 T^3 \quad (67)$$

$$\phi_p = 90^\circ + 0^\circ.6406161 T + 0^\circ.0003041 T^2 + 0^\circ.0000051 T^3 \quad (68)$$

where T is $(\text{MJD}_{\text{out}} - \text{MJD}_{\text{in}})/36525$ is the difference between the two epochs, in Julian centuries.

1.2.3.4 Suggested test cases

$(\alpha, \delta) = (0^\circ, 0^\circ)$ transforms to Galactic coordinates $(l, b) = (96.337272^\circ, -60.188553^\circ)$, and Ecliptic coordinates $(\lambda, \beta) = (0^\circ, 0^\circ)$.

$(\alpha, \delta) = (0^\circ, 90^\circ)$ transforms to Galactic coordinates $(l, b) = (122.93192^\circ, 27.12825^\circ)$, and Ecliptic coordinates at J2000.0 (i.e., $T = 1$), $(\lambda, \beta) = (90^\circ, 66.560719^\circ)$.

$(\alpha, \delta) = (180^\circ, 30^\circ)$ transforms to Galactic coordinates $(l, b) = (195.639488^\circ, 78.353806^\circ)$, and Ecliptic coordinates at J2100.0 (i.e., $T = 2$), $(\lambda, \beta) = (167.072470^\circ, 27.308813^\circ)$.

For each of the above input coordinates, precessing from J2100 to J1900 gives the following output coordinates: $(357.437^\circ, -1.113^\circ)$, $(358.719^\circ, 88.886^\circ)$ and $(177.423^\circ, 31.113^\circ)$.

1.2.4 Projections

We implement three types of projections: *zenithal*, *cylindrical* and *pseudocylindrical*, each requiring slightly different handling. Our representations are based on the treatment of projections presented by Greisen & Calabretta (1995, ADASS, 4, 233). In all of these projections, we are converting from a spherical coordinate α, δ to a linear (2-D) coordinate x_p, y_p . The projection is defined by the projection type, the projection center (α_p, δ_p) and the the plate scales in the x_p and y_p directions (ρ_x, ρ_y) .

In the structure, `psProjection`, the projection type is defined by the element `type`, the projection center α_p, δ_p is defined by the elements `R`, `D`, and the plate scales, ρ_x, ρ_y , are defined by the elements `XS`, `YS`. The plate scales are applied independently to the x and y coordinates to convert them to the corresponding linear units (ie, pixels):

$$x_p = \rho_x x \tag{69}$$

$$y_p = \rho_y y \tag{70}$$

$$\tag{71}$$

In the discussions below, we ignore this last step (or first step, depending on the direction of the conversion).

1.2.4.1 Zenithal Projections

The *zenithal* projections are defined relative to a set of spherical coordinates with pole at the center of the projection (α_p, δ_p) , and which thus represents a coordinate system rotated relative to the coordinate system of α, δ . In this spherical coordinate system, the coordinate of longitude is labeled ϕ , and has domain of $-\pi < \phi \leq \pi$, while the latitude, measured from the pole, is labeled θ and has domain $0 \leq \theta \leq \pi$. The coordinate frame of ϕ, θ is defined so that ϕ_p , the longitude of the target system pole, is 0.0.

For an arbitrary projection center, it is necessary to convert the spherical coordinates to be projected (α, δ) to the projection spherical coordinate system coordinates (ϕ, θ) . In practice, we construct the following useful

trigonometric relationships between ϕ and θ which may be employed in the equations of x, y below:

$$\sin \theta = \sin \delta \sin \delta_p + \cos \delta \cos \delta_p \cos(\alpha - \alpha_p) \quad (72)$$

$$\cos \theta \cos \phi = \sin \delta \cos \delta_p - \cos \delta \sin \delta_p \cos(\alpha - \alpha_p) \quad (73)$$

$$\cos \theta \sin \phi = -\cos \delta \sin(\alpha - \alpha_p) \quad (74)$$

For the inverse transformations, the equivalent relationships are:

$$\sin \delta = \sin \theta \sin \delta_p + \cos \theta \cos \delta_p \cos \phi \quad (75)$$

$$\cos \delta \cos(\alpha - \alpha_p) = \sin \theta \cos \delta_p - \cos \theta \sin \delta_p \cos \phi \quad (76)$$

$$\cos \delta \sin(\alpha - \alpha_p) = -\cos \theta \sin \phi \quad (77)$$

For zenithal projections, the linear coordinates are related to ϕ, θ by:

$$x = R_\theta \sin \phi \quad (78)$$

$$y = -R_\theta \cos \phi \quad (79)$$

and the inverse:

$$R_\theta = \sqrt{x^2 + y^2} \quad (80)$$

$$\phi = \arg(-y, x) \quad (81)$$

The coordinates x, y above are defined to be in angular units (ie, radians).

From these relationships, we can calculate α, δ as:

$$\alpha - \alpha_p = \arctan(\sin \alpha, \cos \alpha) \quad (82)$$

$$\delta = \arcsin(\sin \delta) \quad (83)$$

$$(84)$$

Note that if $(x, y) = (0, 0)$, then $\alpha = \alpha_p, \delta = \delta_p$.

Gnomonic

The Gnomonic projection (“TAN”) is a zenithal projection with $R_\theta = \cot \theta$. The resulting relationships for (x, y) and for $\sin \theta, \cos \theta$ are:

$$x = \frac{\cos \theta \sin \phi}{\sin \theta} \quad (85)$$

$$y = \frac{-\cos \theta \cos \phi}{\sin \theta} \quad (86)$$

$$\sin \theta = \zeta / \sqrt{1 + \zeta^2} \quad (87)$$

$$\cos \theta = 1 / \sqrt{1 + \zeta^2} \quad (88)$$

$$(89)$$

where $\zeta = 1/R_\theta$.

Orthographic

The Orthographic projection (“SIN”) is a zenithal projection with $R_\theta = \cos \theta$. The resulting relationships for (x, y) and for $\sin \theta, \cos \theta$ are:

$$x = \cos \theta \sin \phi \quad (90)$$

$$y = -\cos \theta \cos \phi \quad (91)$$

$$\sin \theta = \sqrt{1 - R_\theta^2} \quad (92)$$

$$\cos \theta = R_\theta \quad (93)$$

$$(94)$$

1.2.4.2 Cylindrical and Pseudocylindrical Projections

The *cylindrical* and *pseudocylindrical* projections are defined relative to a set of cylindrical coordinates whose pole is coincident with the pole of the spherical coordinates. These projections are particularly used for full-sky representations, and are only defined for projection centers with $\delta_p = 0$. In this spherical coordinate system, the coordinate of longitude is labeled ϕ , and has domain of $-\pi < \phi \leq \pi$, while the latitude, measured from the pole, is labeled θ and has domain $0 \leq \theta \leq \pi$. The projection center longitude, α_p corresponds to $\phi = 0$, thus the value of ϕ is determined as $\alpha - \alpha_p$ for all such projections.

Cartesian

The Cartesian projection (“CAR”) is a very simple cylindrical projection with the following relationships between x, y and ϕ, θ :

$$x = \phi \quad (95)$$

$$y = \theta \quad (96)$$

Mercator

The Mercator projection (“MER”) is a cylindrical projection.

$$x = \phi \quad (97)$$

$$y = \ln(\tan(\pi/4 + \theta/2)) \quad (98)$$

$$\text{and } \theta = 2 \arctan(e^y) - \pi/2 \quad (99)$$

Hammer-Aitoff

The Hammer-Aitoff projection(“AIT”) is a pseudocylindrical projection, and is defined:

$$x = 2\zeta \cos \theta \sin \frac{\phi}{2} \quad (100)$$

$$y = \zeta \sin \theta \quad (101)$$

$$\text{where} \quad \zeta^{-1} \equiv \sqrt{\frac{1}{2} \left(1 + \cos \theta \cos \frac{\phi}{2} \right)} \quad (102)$$

And in reverse:

$$\phi = 2\arctan(2z^2 - 1, xz) \quad (103)$$

$$\theta = \arcsin(yz) \quad (104)$$

$$\text{where} \quad z \equiv \sqrt{1 - (x/2)^2 - y^2} \quad (105)$$

Parabolic

The Parabolic projection (“PAR”) is a pseudocylindrical projection, and is defined:

$$x = \phi \left(2 \cos \frac{2\theta}{3} - 1 \right) \quad (106)$$

$$y = \pi \sin \frac{\theta}{3} \quad (107)$$

$$(108)$$

And in reverse:

$$\theta = 3 \sin^{-1} \rho \quad (109)$$

$$\phi = \frac{x}{1 - 4\rho^2} \quad (110)$$

$$\text{where} \quad \rho \equiv y/\pi \quad (111)$$

$$(112)$$

1.2.5 Offset

Coordinate offsets can be either spherical offsets or linear offsets.

A spherical offset is performed by adding the components of the offset, after unit conversion, to the given position. The resulting coordinates must be wrapped to within the allowed range ($-\pi$ to π , 0 to 2π).

A linear offset is defined to be a linear offset in a tangent projection centered on the starting coordinate with y axis aligned with the local direction or increasing Declination. This projection is undefined only for the coordinates exactly at the north and south poles, in which case the orientation is defined to have the y axis parallel to the line of RA = 0.0. The scale of the projection is 1.0 (ie, 1 'pixel' is 1 radian) and the given offsets must be scaled based on the given offset units.

Pseudo-code to implement the above for an offset:

```
psSphere *psSphereSetOffset (psSphere pos, psSphere offset) {
    psPlane lin;
    psSphere new;
    psProjection proj;

    proj.R = pos->r;
    proj.D = pos->d;
    proj.X = 0;
    proj.Y = 0;
    proj.type = PS_PROJ_TAN;

    lin.x = offset.r;
    lin.y = offset.d;

    new = psDeproject (&lin, &proj);
    return (new);
}
```

1.2.6 Tangent Plane to Sky

Mappings between the tangent plane and the sky will be implemented using SLALIB.

To speed up the transformations, SLALIB allows the storage of “apparent-to-observed parameters”, which will be contained in a `psGrommit`. The `psGrommit` consists of the following:

1. geodetic latitude (radians)
2. sine and cosine of geodetic latitude
3. magnitude of diurnal aberration vector
4. height (metres)
5. ambient temperature (degrees K)
6. pressure (mB)
7. relative humidity (0–1)
8. wavelength (μm)
9. lapse rate (degrees K per metre)

10. refraction constants A and B (radians)
11. longitude + eqn of equinoxes + “sidereal Δ UT” (radians)
12. local apparent sidereal time (radians)

These may be calculated using `sla_AOPPA`. Note that a `psGrommit` is only appropriate for a single exposure.

Once the `psGrommit` has been calculated, the functions `sla_OAPQK` and `sla_AOPQK` convert from the tangent plane to the sky, and the sky to the tangent plane, respectively. (Note that “observed” in SLALIB refers to the tangent plane, and “apparent” refers to the apparent position on the sky.)

1.2.7 The One-to-Many Problem with Mosaic Cameras

The Pan-STARRS focal plane consists of several chips, so we will often want to identify which chip a source lies on, when we know the coordinates in the focal plane. This is an example of the one-to-many problem (one coordinate, many chips that it may lie on).

If this needs to be repeated for only one (or a small number of) focal plane coordinates, then the fastest method is to simply convert the focal plane coordinates to chip coordinates for each of the chips, and determine for which of the chips the chip coordinates are valid (i.e. on the chip).

On the other hand, if this needs to be repeated for many source focal plane coordinates, then it is most efficient to convert the centers of each of the chips to coordinates on the focal plane and to use the distance of the source to each of the centers to optimise which chips are tested first. This saves testing many chips for every source.

1.2.8 General Astronomy Functions

The airmass is calculated using the SLALIB function `sla_AIRMAS`.

The parallactic angle is calculated using the SLALIB function `sla_PA`.

To calculate the parallax factors, get the mean-to-apparent parameters (`sla_MAPPA`) for a mean epoch of 2000.0, and, given the mean position of interest, calculate the apparent position (`sla_MAPQK`) for a parallax of 1.0 arcsec (α_1, δ_1), and a parallax of 0.0 arcsec (α_0, δ_0). Then the parallax factors in radians are:

$$P_x = 3,600(180^\circ/\pi)(\alpha_1 - \alpha_0)\cos(\delta_0) \quad (113)$$

$$P_y = 3,600(180^\circ/\pi)(\delta_1 - \delta_0) \quad (114)$$

1.2.9 Positions of Major Solar System Objects

The SLALIB function `SLA_RDPLAN` returns the apparent position of a specified planet, or the Moon.

To calculate the position of the Sun, use `sla_EVP` to get the position of the earth relative to the Sun, and convert from the cartesian coordinates to spherical using `sla_DCC2S`, and calculate the position on the opposite side of the sphere ($\alpha \rightarrow \alpha + 12\text{hrs}$ and $\delta \rightarrow -\delta$).

1.2.10 Missing and Todo

[define SINC, LAGRANGE interpolation]

[define sunrise, sunset, sun position]

[define moonrise, moonset, moon position, moon phase]

[define planet functions]

[clean up FITS I/O issues]

[define Brent's method & minimization bracketing]

A Change Log

A.1 Changes from version 06 to version 07

- Reworked discussion about lookup tables for `pSTime`.
- Moved discussion of lookup tables for ΔUT , x_p and y_p into the SDRS, along with more thorough specification.