UNIVERSITY OF HAWAII AT MĀNOA
Institute for Astrononmy

Pan-STARRS Project Management System

Pan-STARRS PS-1 Image Processing Pipeline
Algorithm Design Description
IPP ADD

Grant Award No.	**: F29601-02-1-0268**
Prepared For	**: Pan-STARRS PMO**
Prepared By	**: Eugene Magnier, Paul Price, Joshua Hoblitt, Robert Lupton**
Document No.	**: PSDC-430-006**
Document Date	**: July 11, 2005**
Revision	**: 12**

DISTRIBUTION STATEMENT
Approved for Public Release – Distribution is Unlimited

2680 Woodlawn Drive, Honolulu, Hawaii 96822

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, then removed.
02	2004 Jun 22	modified stats specification
03	2004 Jul 13	
04	2004 Aug 16	
05	2004 Sep 01	
06	2004 Sep 07	Frozen for PSLib-2
07	2004 Nov 24	Frozen for Cycle 4
08	2005 Jan 21	Draft for Cycle 5
09	2005 Feb 14	Frozen for Cycle 5
10	2005 Apr 19	Frozen for Cycle 6
11	2005 Apr 27	Update for Cycle 6

1 Referenced Documents

1.1 Internal Documents

Reference	Title
PSDC-230-001	PS-1 Design Reference Mission
PSDC-430-004	Pan-STARRS PS-1 IPP C Code Conventions
PSDC-430-005	Pan-STARRS PS-1 IPP Software Requirements Specification
PSDC-430-006	Pan-STARRS PS-1 IPP Algorithm Design Document
PSDC-430-011	Pan-STARRS PS-1 IPP System/Subsystem Design Description

1.2 External Documents

Reference	Title
Posix Standard	Open Group Based Specifications Issue 6, IEEE Std 1003.1, 2003
Numerical Recipes (NR)	Press, Teukolsky, Vetterline, Flannery
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.htm`
GSL	
FFTW	(Fastest Fourier Transform in the West) `http://www.fftw.org`
FITS Projection Article	Greisen & Calabretta (1995, ADASS, 4, 233) `http://www.cv.nrao.edu/fits/documents/wcs/wcs.all.ps`
Hipparcos and Tycho Catalogues	`http://astro.estec.esa.nl/Hipparcos/CATALOGUE_VOL1/catalog_vol1.html`
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

2 PSLib Math Utilities

2.1 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]]`.

2.2 Smoothing: Boxcar and Gaussian

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

$$g_i = \sum_{j=i_{\min}}^{i_{\max}} c_{ij} f_j \tag{1}$$

where the values of c_{ij} determine the filter type. For boxcar smoothing, the values c_{ij} are constant and scaled 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_{ij} as fewer input data points may be used). For Gaussian smoothing, the crucial parameter is σ_x, the standard deviation. The values of $i_{\min}$ and $i_{\max}$ are functions of the standard deviation: $i_{\min}$ corresponds to the bin in which $x_i - N\sigma_x$ is found; similarly $i_{\max}$ is the bin corresponding to $x_i + N\sigma_x$. The value of N should be chosen to be large enough to sample the Gaussian, $N = 5$. The values of c_{ij} are then just the Gaussian curve:

$$c_{ij} = \frac{e^{\frac{-(x_j - x_i)^2}{2\sigma_x^2}}}{\sqrt{2\pi\sigma_x^2}} \tag{2}$$

2.3 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 statisic, 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.

2.3.1 Sample Statistics

We define the following statistical terms, assuming there is a set of data elements x_i with (standard) errors σ_i.

2.3.1.1 Mean The simple mean is defined as:

$$\bar{x} = \frac{1}{N} \sum_{i}^{N} x_i \tag{3}$$

2.3.1.2 Weighted Mean The weighted mean is defined as:

$$\bar{x} = \sum_{i} \frac{x_i}{\sigma_i^2} \; / \; \sum_{i} \frac{1}{\sigma_i^2} \tag{4}$$

In the event that all the errors are identical, this reduces to the standard definition of the mean.

2.3.1.3 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. Errors are ignored when calculating the sample median.

2.3.1.4 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. Errors are ignored when calculating the sample quartiles.

2.3.1.5 Standard Deviation The standard deviation of the sample is given by:

$$\sigma = \sqrt{\sum_{i=1}^{N} \frac{(x_i - \bar{x})^2}{N-1}} \tag{5}$$

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

$$\sigma = \sqrt{\frac{1}{N-1}\left[\sum_{i=1}^{N}(x_i - \bar{x})^2 - \frac{1}{N}\left(\sum_{i=1}^{N}(x_i - \bar{x})\right)^2\right]} \tag{6}$$

If the errors are known, then the sample standard deviation is:

$$\sigma = \left(\sum_i \frac{1}{\sigma_i^2}\right)^{-1/2} \tag{7}$$

2.3.2 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.

If the errors in the input values are known, then the clips are made on the basis of the errors in the input values instead of the standard deviation of the sample: values are excluded for which $|x_i - \bar{x}| > k\sigma_i$.

2.3.3 Robust Statistics

Robust statistics algorithms provide 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 bias the sample statistics. We define two levels of robust statistical estimators: robust statistics using the cumulative histogram, and statistics measured by fitting the differential histogram.

The robust statistics from the cumulative are calculated by constructing a cumulative histogram of the values and performing simple measurements of the histogram data distribution. The use of the cumulative histogram by this algorithm reduces the sensitivity to the exact bin size. The initial bin size is set to 1/1000 of the total data range. The data values are found using quadratic interpolation between the bin of interest and its two neighbors.

- Construct the histogram with the specified bin size.
- Construct the cumulative histogram from the specific histogram
- Find the bin which contains the 50% data point.
- Interpolate to the exact 50% position: this is the robust histogram median.
- Find the bins which contains the 15.8655% and 84.1345% data points.
- Interpolate to find these two positions exactly: these are the $\pm 1\sigma$ positions.
- Determine σ as 1/2 of the distance between these positions.
- If the measured σ is less than 2 times the bin size, exclude points which are more than 25 bins from the median, recalculate the bin size, and perform the algorithm again.
- Find the bins which contains the 25% and 75% data points.
- Interpolate to find these two positions exactly: these are the upper and lower quartile positions.

If the errors in the input values are known, then the same approach is used, except that the histograms become probability density functions (PDFs). In this case, the input values are spread out, so that they do not simply contribute a single unit to the histogram, but rather contribute a fraction of a value, equivalent to the weight. In the interests of speed, a boxcar PDF may be used to represent each input value (as opposed to a Gaussian), where the boxcar width is equal to $2\sqrt{2\ln 2}$ times the error and each input value contributes constant area. Then the robust median and standard deviation are estimated in the same manner as above.

2.3.4 Fitted Statistics

The fitted statistics algorithm starts with the histogram used for the robust statistics and determines the population statistics by fitting a Gaussian model to the histogram. The algorithm is as follows:

- Perform the Robust Histogram Statistics algorithm above
- Smooth the resulting histogram with a Gaussian with σ_s = 1 bin.
- Find the bin with the peak value in the range $\pm 2\sigma$ of the robust histogram median.
- Fit a Gaussian to the bins in the range $\pm 2\sigma$ of the robust histogram median.
- The robust mean mean$_r$ is derived directly from the fitted Gaussian mean.
- The robust standard deviation, σ_r, is determined by subtracting the smoothing scale in quadrature: $\sigma_r^2 = \sigma^2 - \sigma_s^2$

2.3.5 Histograms

When calculating histograms in the presence of known errors in the input values, the approach described above for the robust statistics is used (i.e., the histograms become probability density functions).

An example may help here. Say we have our histogram bounds being 0, 1, 2, 3, 4, 5; and our value is 2.5 ± 0.5. Then, the width of the contribution is $0.5 \times 2.35... \approx 1.175$. Half the width is 0.5875, so we will treat this value as a boxcar from $2.5 - 0.5875$ to $2.5 + 0.5875$.

Consequently, the bins 0 to 1 and 4 to 5 get no value, because none of the boxcar overlaps. The bin 1 to 2 gets 0.0875, because that's the fraction of the boxcar that overlaps with it; same thing with the bin 3 to 4. The bin 2 to 3 gets 0.825 because that's the fraction of the boxcar that overlaps with it. So the single value 2.5 ± 0.5 makes the following histogram:

Bin	Value
0–1	0
1–2	0.0875
2–3	0.8250
3–4	0.0875
4–5	0

Note that the total adds to one — the number of values added.

2.4 Polynomials

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

$$f(x) = \sum_{i=0}^{n} c_i T_i(x) \tag{8}$$

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:

$$T_0(x) = 1 \quad (9)$$
$$T_1(x) = x \quad (10)$$
$$T_2(x) = 2x^2 - 1 \quad (11)$$
$$T_3(x) = 4x^3 - 3x \quad (12)$$
$$T_4(x) = 8x^4 - 8x^2 + 1 \quad (13)$$
$$(14)$$

Chebyshev polynomials follow the recurrence relation:

$$T_{n+1} = 2xT_n - T_{n-1} \quad (15)$$

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

$$d_j = 2xd_{j+1} - d_{j+2} + c_j \quad (16)$$
$$f(x) = x * d_1 - d_2 + 1/2c_0 \quad (17)$$
$$(18)$$

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

2.4.1 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.

2.5 Fitting

2.5.1 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:

$$\chi^2 = \sum_{i=0}^{N-1} \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2 \quad (19)$$

2.5.2 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 (2.5.1), 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

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

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

$$\sum_j \alpha_j \sum_i \frac{x_i^k x_i^j}{\sigma_i^2} = \sum_i \frac{x_i^k y_i}{\sigma_i^2} \tag{21}$$

This matrix equation may be solved with LU Decomposition (section 2.8.1).

2.6 Non-linear Minimization

Non-linear minimization techniques use an iterative approach to find a minimization when an analytical inversion is impractical or not possible. These techniques use a starting guess for the parameters of interest, and make a sequence of new guess parameters based on the properties of the function at the previous position. If the new parameters yield a reduced function value, the new position is used as the starting position for the next iteration. Otherwise, the guess must be modified and another attempt is made. Convergence may be determined based on the absolute amount of change in the function value, or by comparison with the expectation for a linear system.

The two common techniques used to construct a guess parameter set are the 'steepest descent method' and the 'Gauss-Newton method'. In the first case, the guess is selected some distance along the local gradient. In the second case, a local Taylor expansion of the function is used to construct a linear model for the function, and the new guess is chosen to minimize that linear model. The methods discussed below make use of combinations of these two methods. Aside from the differences in their guess steps, the two methods differ in using first derivatives of the function supplied by the user, or by locally calculating the first derivatives.

Mathematically, we would like to choose the parameter set a_m to minimize a function of those parameters $F(a_m)$. We iterate by choosing a new parameter set $a'_m = a_m + \delta_m$ based on the behavior of the function at a_m.

The steepest descent method chooses a step direction of $\bar{\delta} = -\nabla F$, or defining g_m as a component of the gradient, $\delta_m = -g_m$. The Gauss-Newton method uses a Taylor expansion of the function to solve for the step: $(\nabla^2 F)\bar{\delta} = -\nabla F$. Defining an element of the Hessian matrix $H_{m,n}$ as a component of the second derivatives, we can write the Gauss-Newton step as $\delta_m = -H^{-1}_{m,n} g_n$.

2.6.1 Levenberg-Marquardt Method

In the Levenberg-Marquardt Method (LMM; see NR §15.5, Madsen et al), 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 varient based on the results. The LMM requires the second derivative of the function to be negligible, as in the case of minimizing χ^2.

Consider the chi-square function. Given some ordinates, x_i, we would like to find the parameters, a_m, of the function which minimize χ^2 for some measurements, y_i and associated errors, σ_i:

$$\chi^2(\bar{a}) \quad = \quad \sum_i \frac{1}{\sigma_i^2} \left(y_i - p(x_i; a_m)\right)^2 \tag{22}$$

We simplify this as:

$$p_i(a_m) = p(x_i; a_m) \quad (23)$$
$$f_i(a_m) = \frac{1}{\sigma_i}(y_i - p_i) \quad (24)$$
$$\chi^2(\bar{a}) = \sum_i f_i^2 \quad (25)$$

We write the minimization function $F(a_m) = \frac{1}{2}\chi^2$ to avoid the various extra factors of 2. We can now write out the needed derivatives in terms of f_i:

$$F(a_m) = \frac{1}{2}\sum_i f_i^2 \quad (26)$$
$$\nabla F(a_m) = \sum_i f_i \frac{\partial f_i}{\partial a_m} \quad (27)$$
$$\nabla^2 F(a_m) = \sum_i \frac{\partial f_i}{\partial a_m}\frac{\partial f_i}{\partial a_n} \quad (28)$$

where we have dropped the second-derivatives of the function in the representation of $\nabla^2 F(a_m)$. Since $\frac{\partial f_i}{\partial a_m} = -\frac{1}{\sigma_i}\frac{\partial p_i}{\partial a_m}$, we can write these in terms of the derivatives of p_i only:

$$\nabla F(a_m) = -\sum_i \frac{f_i}{\sigma_i}\frac{\partial p_i}{\partial a_m} \quad (29)$$
$$\nabla^2 F(a_m) = \sum_i \frac{1}{\sigma_i^2}\frac{\partial p_i}{\partial a_m}\frac{\partial p_i}{\partial a_n} \quad (30)$$

Writing these in matrix representation, and replacing f_i, we have:

$$-g_m = \sum_i \frac{(y_i - p_i)}{\sigma_i^2}\frac{\partial p_i}{\partial a_m} \quad (31)$$
$$H_{m,n} = \sum_i \frac{1}{\sigma_i^2}\frac{\partial p_i}{\partial a_m}\frac{\partial p_i}{\partial a_n} \quad (32)$$

In the Levenberg-Marquart Method, we define a new guess using a combination of the Steepest Descent and Gauss-Newton methods discussed above. We replace the Hessian matrix above with $A_{m,n}$ as a variant on $H_{m,n}$ as follows:

$$A_{m,n} = H_{m,n}\text{if}(j \neq k) \quad (33)$$
$$A_{m,n} = H_{m,n}(1+\lambda)\text{if}(j = k) \quad (34)$$

and solve the system of equations represented by:

$$A_{m,n}\delta_n = -g_m \quad (35)$$

The new parameter guess is then found from this value with $\alpha'_n = \alpha_n + \delta_n$. We use this parameter set to evaluate the function.

To assess the quality of the new parameter set, we compare the change in χ^2 with the change expected from the linear model (the Taylor expansion). If the linear model were correct, we would have expected a change (a reduction) in χ^2 of $\Delta = \frac{\lambda}{2}\sum \delta_m^2 + \frac{1}{2}\sum \delta_m g_m$. We use the 'gain ratio' $\rho = \frac{\chi^2_{\rm old} - \chi^2_{\rm new}}{\Delta}$ to judge the new step. If $\rho > 0$, 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_m are the best-fit parameters for the system.

The covariance matrix, $C_{i,j}$, which is the inverse of the matrix $H_{m,n}$ provides an estimate of the confidence limits of the parameters.

2.6.2 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 $f_Q P \geq f_P$, then there is no point in keeping the new vector, because there is no further minimization to be made in that direction.
   - If $2(f_P - 2f_Q + f_{QP})\left[(f_P - f_Q) - \Delta_{\max}\right]^2 \geq (f_P - f_{QP})^2\Delta_{\max}$, 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 (TBD)** .

2.7 Image Manipulations

2.7.1 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).

2.7.1.1 Nearest Pixel Interpolation (`PS_INTERPOLATE_FLAT`**)** In this interpolation, the value of the closest pixel is returned. This is equivalent to pixel duplication or replication.

2.7.1.2 Bilinear Interpolation (`PS_INTERPOLATE_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 fx, fy 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 expresion as:

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

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

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 $f_x - 0.5 - int(f_x - 0.5)$. 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).

2.7.1.3 Sinc Interpolation (`PS_INTERPOLATE_LANCZOS[234]`**)** Because it would be slow to specify the size of the kernel dynamically, we specify three hard-coded kernel sizes: 4, 6 and 8 pixels in each dimension (a kernel of size 2 pixels in each dimension is handled by the bilinear interpolation). These correspond to the options `PS_INTERPOLATE_LANCZOS2`, `PS_INTERPOLATE_LANCZOS3` and `PS_INTERPOLATE_LANCZOS4`, respectively.

Given a position on the input image, (x_0, y_0), a kernel is derived according to pixels local to the position:

$$h(x, y) = \mathrm{sinc}(\pi\delta x)\mathrm{sinc}(\pi\delta x/N)\mathrm{sinc}(\pi\delta \mathrm{y})\mathrm{sinc}(\pi\delta \mathrm{y}/\mathrm{N}) \quad (36)$$

where

$$\delta x = x - x_0 \quad (37)$$

$$\delta y = y - y_0 \quad (38)$$

$$\mathrm{sinc}(z) = \sin(z)/z \quad (39)$$

and N corresponds to the choice of kernel size. For $N = 2$, the kernel size is 4 pixels in each dimension (i.e., $-2 < \delta x \leq 2$). For $N = 3$, the kernel size is 6 pixels in each dimension (i.e., $-3 < \delta x \leq 3$). For $N = 4$, the kernel size is 8 pixels in each dimension (i.e., $-4 < \delta x \leq 4$).

The interpolated value at the given position, (x_0, y_0), is then simply the dot product of the kernel and the fluxes:

$$f(x_0, y_0) = \sum_R f(x, y)h(x, y) \tag{40}$$

where R is the region defined by the kernel size, and $f(x, y)$ is the flux at the pixel position.

For further information, see the SWarp manual.

2.7.2 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 posibility 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).

2.7.2.1 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.

2.7.2.2 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];
}
```

2.7.3 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,$-N_x \sin\theta$).

2.8 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`.

2.8.1 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:

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

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

$$U = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{34} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix} \tag{42}$$

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

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \tag{43}$$

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

$$\alpha_{ij} = \frac{1}{\beta_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj} \right) \tag{44}$$

2.8.2 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:

$$D = P_{i=1} N U_{ii} \tag{45}$$

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.

2.8.3 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 $Ly = B$:

$$y_1 = \frac{b_1}{\alpha_{11}} \tag{46}$$

$$y_i = \frac{1}{\alpha_{ii}} \left(b_i - \sum_{j=1}^{i-1} \alpha_{ij} y_i \right) \tag{47}$$

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

$$x_N = \frac{y_N}{\beta_{NN}} \tag{48}$$

$$x_i = \frac{1}{\beta_{ii}} \left(y_i - \sum_{j=i+1}^{N} \beta_{ij} y_i j \right) \tag{49}$$

2.8.4 Invert a matrix

Inversion of a matrix using the LU decomposition is performed by performing back-subsitution 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`.

2.8.5 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

colum from the input matrices. So, for example, given the input matrices α_{ij} and β_{ij}, the image and matrix multiplications correspond to:

$$\text{image}_{ij} = \alpha_{ij} \times \beta_{ij} \tag{50}$$

$$\text{matrix}_{jk} = \sum_{i=1}^{N} \alpha_{ij} \times \beta_{ki} \tag{51}$$

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: $+, -, \times$.

2.8.6 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

$$T_{ij} = M_{ji} \tag{52}$$

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

2.8.7 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.

2.9 (Fast) Fourier Transforms

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

2.9.1 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.

2.9.2 Function mapping

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

PSLib function	Major FFTW call
`psFFTForward()`	`fftw_plan_dft_r2c_2d()`
`psFFTReverse()`	`fftw_plan_dft_c2r_2d()`

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.

2.9.3 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 the elements of the power spectrum are (NR §13.4):

$$P_0 = |C_0|^2 / N^2 \quad (53)$$

$$P_j = \left(|C_j|^2 + |C_{N-j}|^2\right) / N^2 \quad (54)$$

$$P_{N/2} = \left|C_{N/2}\right|^2 / N^2 \quad (55)$$

$$(56)$$

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

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

3 PSLib Astronomy Utilities

3.1 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 nanoseconds in a time system defined by the `psTimeType` element `type`. All available time-systems are defined in terms of the reference epoch "1970-01-01T00:00:00Z" (Gregorian[1]), but with minor modifications, as needed, for features such as leap-seconds. The first represenatation, TAI (International Atomic Time), has seconds of uniform length (SI seconds) and no leap-seconds. The exact zero reference is "1970-01-01T00:00:10Z" UTC. The second representation is UTC, which has seconds of uniform length and leap-seconds as needed to adjust it to remain within $0.9s$ of the Earth's rotation. It has a zero-point of exactly "1970-01-01T00:00:00Z" UTC.

[1] Gregorian Calendar - http://en.wikipedia.org/wiki/Gregorian_calendar

3.1.1 Coordinated Universal Time (UTC)

Coordinated Univeral Time (UTC) is defined by the International Telecommunication Union (ITU)[2]. It is a system of time with SI length seconds but attempts to stay within $1s$ of UT1. This is done by the insertion of a "leap-second" whenever $|UTC - UT1| \geq 0.9s$. By definition[3], $UTC - TAI$ is an integer number of seconds. UTC went into effect on "1972-01-01T00:00:00Z" 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.

$$\mathrm{UTC} = \mathrm{TAI} - 10\mathrm{s} - \mathrm{leapseconds} \tag{57}$$

Leap-seconds are declared by the International Earth Rotation and Reference Systems Service (IERS)[4]. Leap-seconds 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 leap-second in a given UTC date a table of leap-seconds as annouced by the IERS must be consulted. This table will have to be updated each time a new leap-second occurs.

For ease of conversion, UTC should be represented as the number of seconds since the UNIX epoch of "1970-01-01T00:00:00Z", non-inclusive of leap-seconds. **what does this statement actually mean? what is the source of time (gettimeofday? let's be explicit here (TBD)**

Times will always be expressed in the 'UTC timezone'. Use of the local timezone is forbidden. – this makes no sense given that we define LST. In any case, this statement, or somethign equivalent, belongs in the SDRS not the ADD (TBD)

3.1.2 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)[5] 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 leap-seconds since "1972-01-01" up until the UTC date being converted.

$$\mathrm{TAI} = \mathrm{UTC} + 10\mathrm{s} + \mathrm{leapseconds} \tag{58}$$

For ease of conversion, TAI should be represented as the number of seconds since the UNIX epoch of "1970-01-01T00:00:00Z". **what does this statement actually mean? (TBD)**

3.1.3 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
```

[2] ITU website - http://www.itu.int/home/index.html
[3] UTC definition - http://www.cl.cam.ac.uk/ mgk25/volatile/ITU-R-TF.460-4.pdf
[4] IERS website - http://www.iers.org/
[5] BIPM website - http://www.bipm.fr/

```
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 USNO[6].

3.1.4 Universal Time (UT1)

UT1 is directly tied to the rotation of the Earth. Historically, time has been measured with respect to the rising and setting of the Sun. However, in the modern era of atomic clocks, the rotation of the Earth makes for a highly unstable time standard. Tidal effects, changes in the angular momentum of the atmosphere, seasonal changes in the polar ice caps, movement within the Earth's core, and other effects all cause measurable changes in the Earth's rotation on a daily basis. However, UT1 is still vitally important for determining the orientation of the Earth with respect to the sky. UT1 is calculated by applying the value UT1-UTC to the value of UTC. UT1 is continuously measured by the International Earth Rotation Service, and tabulated values of the offset of UT1 from UTC are published at regular intervals, along with predicted future values. The process of calculating the UT1-UTC offsets are discussed in the section on Earth Orientation (Section 3.5.5).

3.1.5 Gregorian dates to seconds

The Perl code below, based on an algorithm described in the book "Calendrical Calculations"[7] and modified to return seconds, converts from Gregorian-formatted dates to seconds since the UNIX epoch.

Given year, month, day as `$y, $m, $d`.

```
use integer;
```

[6] ftp://maia.usno.navy.mil/ser7/tai-utc.dat

[7] Calendrical Calculations - http://emr.cs.iit.edu/home/reingold/calendar-book/second-edition/

```
my $adj;

# make month in range 3..14 (treat Jan & Feb as months 13..14 of
# prev year)
if ( $m <= 2 )
{
    $y -= ( $adj = ( 14 - $m ) / 12 );
    $m += 12 * $adj;
}
elsif ( $m > 14 )
{
    $y += ( $adj = ( $m - 3 ) / 12 );
    $m -= 12 * $adj;
}

# make year positive (oh, for a use integer 'sane_div'!)
if ( $y < 0 )
{
    $d -= 146097 * ( $adj = ( 399 - $y ) / 400 );
    $y += 400 * $adj;
}

# 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 (whew)

$d += ( $m * 367 - 1094 ) / 12 + $y % 100 * 1461 / 4 +
      ( $y / 100 * 36524 + $y / 400 ) - 306;

# convert from count of days to seconds since the UNIX epoch
$unix = ( ( $d - 1 ) * 86400 ) - 62135596800;
$utc = $unix - leapseconds($unix);
```

Outputs seconds as `$utc`.

To go the other way:

Given the number of seconds since the UNIX epoch as `$utc`.

```
use integer;

my $unix = $utc + leapseconds( $utc )
$d = ( unix + 62135596800 ) / 86400

my $rd = $d;
```

```
    my $yadj = 0;
    my ( $c, $y, $m );

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

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

    if ( $_[0] )
    {
        my $dow;

        if ( $rd < -6 )
        {
            $dow = ( $rd + 6 ) % 7;
            $dow += $dow ? 8 : 1;
        }
        else
        {
            $dow = ( ( $rd + 6 ) % 7 ) + 1;
        }

        my $doy =
            $class->_end_of_last_month_day_of_year( $y, $m );

        $doy += $d;

        my $quarter;
        {
```

```
        no integer;
        $quarter = int( ( 1 / 3.1 ) * $m ) + 1;
    }

    my $qm = ( 3 * $quarter ) - 2;

    my $doq =
        ( $doy -
          $class->_end_of_last_month_day_of_year( $y, $qm )
        );
```

Outputs year, month, day as `$y, $m, $d`.

The above code was taken [and slightly altered] from `DateTime.pm`[8] *(C) 2003 Dave Rolsky. Please see the DateTime project website*[9] *for further details.*

3.1.6 Julian Date and Modified Julian Date

The follow definitions of Julian Date (JD) and Modified Julian Date (MJD) was taken from, "RESOLUTION B1: ON THE USE OF JULIAN DATES" of "The XXIIIrd International Astronomical Union General Assembly"[10].

3.1.6.1 Julian Date

```
1. Julian day number (JDN)

The Julian day number associated with the solar day is the number assigned to a
day in a continuous count of days beginning with the Julian day number 0
assigned to the day starting at Greenwich mean noon on 1 January 4713 BC,
Julian proleptic calendar -4712.

2. Julian Date (JD)

The Julian Date (JD) of any instant is the Julian day number for the preceding
noon plus the fraction of the day since that instant. A Julian Date begins at
12h 0m 0s and is composed of 86400 seconds. To determine time intervals in a
uniform time system it is necessary to express the JD in a uniform time scale.
For that purpose it is recommended that JD be specified as SI seconds in
Terrestrial Time (TT) where the length of day is 86,400 SI seconds.

In some cases it may be necessary to specify Julian Date using a different time
scale. (See Seidelmann, 1992, for an explanation of the various time scales in
use). The time scale used should be indicated when required such as JD(UT1). It
should be noted that time intervals calculated from differences of Julian Dates
```

[8] DateTime.pm - http://search.cpan.org/ drolsky/DateTime/

[9] DateTime project - http://datetime.perl.org

[10] RESOLUTION B1: ON THE USE OF JULIAN DATES - http://www.iers.org/iers/earth/resolutions/UAI_b1.html

specified in non-uniform time scales, such as UTC, may need to be corrected for changes in time scales (e.g. leap seconds).

3.1.6.2 Modified Julian Date

"that for those cases where it is convenient to employ a day beginning at midnight, the Modified Julian Date (equivalent to the Julian Date minus 2 400 000.5) be used"

3.1.6.3 JD and MJD conversion Conversion between psTime values and MJD and JD are determined from:

Where psTime is a PS_TIME_TAI.

```
mjd = psTime.sec/86400.0 + psTime.nsec/86400000000000.0 + 40587.0;
 jd = psTime.sec/86400.0 + psTime.nsec/86400000000000.0 + 2440587.5;
```

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

$$\mathrm{JD} = \mathrm{MJD} + 2400000.5 \tag{59}$$

3.1.7 Terrestrial Time (TT)

Terrestrial Time (TT) is defined as a fixed offset from TAI.

$$\mathrm{TT} = \mathrm{TAI} + 32.184\mathrm{s} \tag{60}$$

3.1.8 TT as Julian Centuries since J2000.0

The algorithm for calulating GMST requires TT formated in Julian centruies since the J2000.0 epoch.

$$t_u = \frac{\mathrm{JD_{TT}} - 2451545.0}{36525} \tag{61}$$

3.1.9 UT1 as Julian Centuries since J2000.0

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

$$t = \frac{\mathrm{JD_{UT1}} - 2451545.0}{36525} \tag{62}$$

3.1.10 Local Mean Sidereal Time (LMST)

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

$$LMST = GMST00(t_u, t) + longitude \tag{63}$$

Gives $LMST$ in seconds.

3.1.11 Greenwich Mean Sidereal Time (GMST)

Greenwich Mean Sidereal Time (GMST) is caclulated from UT1 and TT. This algorithm requires that both time inputs are expressed as Julian centuries since J2000.0 [11].

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

$$\begin{aligned}
\mathrm{GMST00}(t_u, t) &= UT1 + 24110.5493771 && (64)\\
&\quad +8639877.3173760\, t_u + 307.4771600\, t && (65)\\
&\quad +0.0931118\, t^2 - 0.0000062\, t^3 && (66)\\
&\quad +0.0000013\, t^4 && (67)
\end{aligned}$$

Gives $GMST00$ in seconds.

3.2 2D transformations

In PSLib, we implement 2-dimensional transformations using `psPlaneTransform`, which contains a matrix of polynomial coefficients for each dimension. Since we are using these to model the real world, where, for example, a particular point on the detector maps to a particular point on the sky, we consider only transformations that are "one-to-one". This makes it possible to speak of inverse transformations, and of combining multiple transformations.

Given a transformation, $f(x, y)$, the inverse transformation, $g(x, y)$, is that for which $g(f(x, y)) = (x, y)$ for (x, y) over the range of interest (not necessarily the entire set of real numbers).

Given two transformations, $f(x, y)$ and $g(x, y)$, the combined transformation is the transformation, $h(x, y) = g(f(x, y))$ for (x, y) over the range of interest (not necessarily the entire set of real numbers).

Both of these operations are straightforward if the transformation is linear. If the function $(u, v) = f(x, y)$ is:

$$\begin{aligned}
u &= a + bx + cy && (68)\\
v &= d + ex + fy && (69)
\end{aligned}$$

then the inverse transformation $(x, y) = g(u, v)$ is:

$$\begin{aligned}
x &= (-fa + cd)/\Delta + fu/\Delta - cv/\Delta && (70)\\
y &= (ae - bd)/\Delta - eu/\Delta + bv/\Delta && (71)
\end{aligned}$$

[11] Expressions to implement the IAU 2000 definition of UT1 - http://www.edpsciences.org/articles//aa/abs/2003/30/aa3487/aa3487.html

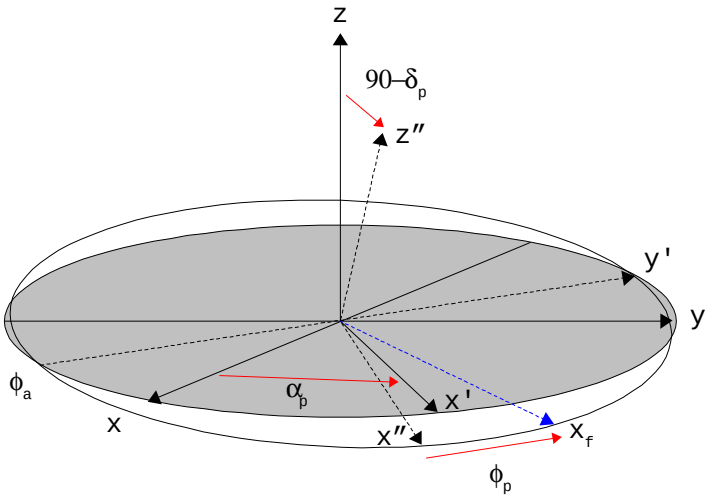

Figure 1: Definition of the rotation angles. Three rotations are performed in series: first, a rotation of α_p is made about the z axis; second, a rotation of δ_p (not $90 - \delta_p$ as shown) is made about the modified y axis, y'; finally, a rotation of ϕ_p is made about the modified z axis, z''. Note that it is the coordinate system that rotates, not the position of interest.

where $\Delta = bf - ce$ is the matrix determinant. Given two functions $f_i(x, y)$ for $i = 1, 2$:

$$u = a_i + b_i x + c_i y \quad (72)$$

$$v = d_i + e_i x + f_i y \quad (73)$$

then the combined transformation, $(u, v) = f_2(f_1(x, y))$ is:

$$u = (a_2 + b_2 a_1 + c_2 d_1) + (b_2 b_1 + c_2 e_1)x + (b_2 c_1 + c_2 f_1)y \quad (74)$$

$$v = (d_2 + e_2 a_1 + f_2 d_1) + (e_2 b_1 + f_2 e_1)x + (e_2 c_1 + f_2 f_1)y \quad (75)$$

When the transformations are not linear, the inverse and combined transformations can be estimated by sampling a grid over the region of interest, calculating the transformation (or double transformation) for each sample, and using this information to derive the best fit transformation that produces the inverse or combined transformation. The inverse transformation should be of the same order as that of the forward transformation, while the combined transformation should be of the higher order of the two component transformations.

3.3 Spherical Rotations

Spherical rotations may be implemented with a variety of mathematical methods. A single rotation may be specified by three angles which describe rotations about the principal axes. Figure 1 shows the rotation angles for an arbitrary 3-D rotation. A single rotation may be represented by defining this set of Euler angles. As an alternative, the quaternion is a convenient construct with which to represent a single rotation, and it has useful mathematical properties when applying those rotations.

3.3.1 Quaternions

A quaternion is an ordered set of four numbers, $q = (q_0, q_1, q_2, q_3)$, which is useful for specifying rotations. A quaternion is made up of a three-vector which specifies an axis about which to rotate, and a scalar which specifies the amount of

rotation. In the following, we call the final value, q_3, the scalar value; note that other sources (e.g., MathWorld) may choose to call the first value the scalar value.

The conjugate of a quaterion, $q = (q_0, q_1, q_2, q_3)$, is $\bar{q} = (-q_0, -q_1, -q_2, q_3)$. Note that the vector components are negated, but not the scalar component.

3.3.2 Quaternion for a position

Given an angular position on the sky, (α, δ), we can construct a quaternion by treating it as a unit vector in cartesian space:

$$p_0 = \cos\delta\cos\alpha \quad (76)$$
$$p_1 = \cos\delta\sin\alpha \quad (77)$$
$$p_2 = \sin\delta \quad (78)$$

and we set the scalar value to zero, $p_3 = 0$.

Given a quaternion, p, we can calculate the position using the inverse of the above equations:

$$\phi = \arctan(p_1, p_0) \quad (79)$$
$$\theta = \arcsin(p_2) \quad (80)$$

where ϕ is the longitude and θ is the latitude. Note that in this case, we neglect the scalar component of the quaternion — it should be zero — and that we implicitly assume that the quaternion is normalised to unity.

3.3.3 Quaternion for a rotation

A rotation of angle θ about the axis defined by the unit vector (v_x, v_y, v_z) is specified by a quaternion with components:

$$r_0 = v_x\sin(\theta/2) \quad (81)$$
$$r_1 = v_y\sin(\theta/2) \quad (82)$$
$$r_2 = v_z\sin(\theta/2) \quad (83)$$
$$r_3 = \cos(\theta/2) \quad (84)$$

Note that the sine and cosine are taken of the half-angle of the rotation. Note also that this implies that the quaternion components are normalized such that $|r| \equiv r_0^2 + r_1^2 + r_2^2 + r_3^2 = 1$.

3.3.4 Multiplication of quaternions

Given two quaternions a and b, there is a third quaternion, $p = ab$. The components of p are given by:

$$p_0 = b_3a_0 + b_2a_1 - b_1a_2 + b_0a_3 \quad (85)$$
$$p_1 = -b_2a_0 + b_3a_1 + b_0a_2 + b_1a_3 \quad (86)$$
$$p_2 = b_1a_0 - b_0a_1 + b_3a_2 + b_2a_3 \quad (87)$$
$$p_3 = -b_0a_0 - b_1a_1 - b_2a_2 + b_3a_3 \quad (88)$$

Note that quaternion multiplication is associative (whether you do the left pair or the right pair first doesn't matter):

$$(ab)c = a(bc) \tag{89}$$

but not commutative (you can't switch the order of the operands):

$$abc \neq acb \tag{90}$$

3.3.5 Rotating a Vector

Rotation of a position is performed by constructing the quaternion for the position, p, and the rotation, r, according to the above equations, and calculating the product:

$$q = rp\bar{r} \tag{91}$$

q is the quaternion of the result. Note the use of the conjugate of the rotation quaternion.

A general rotation may be specified by three individual rotations about a predefined set of axes. We choose to specify rotations around the z, y and z axes, in that order. The amount of rotation around each of these axes are known as Euler angles. Given the Euler angles of a rotation, the rotation may be performed by rotating in turn about the designated axes. Euler angles are specified below for the various rotations required. To use them, the following rotation quaternions are used:

- First, about the Z axis:

$$\begin{aligned} r_0 &= 0 && (92)\\ r_1 &= 0 && (93)\\ r_2 &= \sin(\alpha_p/2) && (94)\\ r_3 &= \cos(\alpha_p/2) && (95) \end{aligned}$$

- Second, about the Y axis:

$$\begin{aligned} s_0 &= 0 && (96)\\ s_1 &= \sin(\delta_p/2) && (97)\\ s_2 &= 0 && (98)\\ s_3 &= \cos(\delta_p/2) && (99) \end{aligned}$$

- Finally, about the Z axis again:

$$\begin{aligned} t_0 &= 0 && (100)\\ t_1 &= 0 && (101)\\ t_2 &= \sin(\phi_p/2) && (102)\\ t_3 &= \cos(\phi_p/2) && (103) \end{aligned}$$

These three quaternions may be multiplied together to yield the quaternion of the combined rotation: tsr (note the order — r is done first, so it is nearest the position quaternion, etc.).

3.3.6 Rotation Matrix

The rotation matrix representation of a rotation may be derived directly from the quaternion representation. The following formulae convert a quaternion to a rotation matrix:

$$
\begin{aligned}
rot_{x,x} &= q_0q_0 - q_1q_1 - q_2q_2 + q_3q_3 && (104)\\
rot_{y,y} &= -q_0q_0 + q_1q_1 - q_2q_2 + q_3q_3 && (105)\\
rot_{z,z} &= -q_0q_0 - q_1q_1 + q_2q_2 + q_3q_3 && (106)\\
rot_{x,y} &= 2(q_0q_1 + q_2q_3) && (107)\\
rot_{y,x} &= 2(q_0q_1 - q_2q_3) && (108)\\
rot_{x,z} &= 2(q_0q_2 - q_1q_3) && (109)\\
rot_{z,x} &= 2(q_0q_2 + q_1q_3) && (110)\\
rot_{y,z} &= 2(q_1q_2 + q_0q_3) && (111)\\
rot_{z,y} &= 2(q_1q_2 - q_0q_3) && (112)
\end{aligned}
$$

3.3.7 Conversion to Other Representations

You may convert a rotation matrix, m, to a quaternion, p, with the following code:

```
double diag_sum[3];
int maxi;
double recip;

diag_sum[0]=1+m[0][0]-m[1][1]-m[2][2];
diag_sum[1]=1-m[0][0]+m[1][1]-m[2][2];
diag_sum[2]=1-m[0][0]-m[1][1]+m[2][2];
diag_sum[3]=1+m[0][0]+m[1][1]+m[2][2];


maxi=0;
for(i=1;i<4;++i) {
    if(diag_sum[i]>diag_sum[maxi]) maxi=i;
}


p[maxi]=0.5*sqrt(diag_sum[maxi]);
recip=1./(4.*p[maxi]);

if(maxi==0) {
    p[1]=recip*(m[0][1]+m[1][0]);
    p[2]=recip*(m[2][0]+m[0][2]);
    p[3]=recip*(m[1][2]-m[2][1]);

} else if(maxi==1) {
```

```
    p[0]=recip*(m[0][1]+m[1][0]);
    p[2]=recip*(m[1][2]+m[2][1]);
    p[3]=recip*(m[2][0]-m[0][2]);

} else if(maxi==2) {
    p[0]=recip*(m[2][0]+m[0][2]);
    p[1]=recip*(m[1][2]+m[2][1]);
    p[3]=recip*(m[0][1]-m[1][0]);

} else if(maxi==3) {
    p[0]=recip*(m[1][2]-m[2][1]);
    p[1]=recip*(m[2][0]-m[0][2]);
    p[2]=recip*(m[0][1]-m[1][0]);
}
```

3.4 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 latitutde 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. Transformations between arbitrary systems are specified by determining the appropriate values of α_p, δ_p, and ϕ_p, and then constructing the quarternion for this transformation.

The relevant trigonometric relationships are:

$$\sin\theta = \sin\delta\cos\delta_p - \cos\delta\sin\delta_p\sin(\alpha-\alpha_p) \quad (113)$$

$$\cos\theta\sin(\phi-\phi_p) = \cos\delta\cos\delta_p\sin(\alpha-\alpha_p) + \sin\delta\sin\delta_p \quad (114)$$

$$\cos\theta\cos(\phi-\phi_p) = \cos\delta\cos(\alpha-\alpha_p) \quad (115)$$

and for the inverse transformations, the equivalent relationships are:

$$\sin\delta = \sin\theta\cos\delta_p - \cos\theta\sin\delta_p\sin(\phi-\phi_p) \quad (116)$$

$$\cos\delta\sin(\alpha-\alpha_p) = \cos\theta\cos\delta_p\sin(\phi-\phi_p) + \sin\theta\sin\delta_p \quad (117)$$

$$\cos\delta\cos(\alpha-\alpha_p) = \cos\theta\cos(\phi-\phi_p) \quad (118)$$

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 `atan2(y,x)`, where $y = \cos\delta\sin(\alpha-\alpha_p)$ and $x = \cos\delta\cos(\alpha-\alpha_p)$; and similarly for $\phi - \phi_p$.

Note that the symmetry between the two sets of above equations means that inverse transformations can be made simply by switching α_p and ϕ_p, changing the sign of δ_p, and using the forward transformation.

3.4.1 Galactic to ICRS

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

$$\alpha_p = 180^\circ - 192.85948^\circ \quad (119)$$
$$\delta_p = 90^\circ - 62.87175^\circ \quad (120)$$
$$\phi_p = 90^\circ + 32.93192^\circ \quad (121)$$
$$(122)$$

3.4.2 Ecliptic to ICRS

The appropriate values, from Zombeck, are:

$$\alpha_p = 270^\circ \quad (123)$$
$$\delta_p = 23^\circ 27' 8''.26 - 46''.845\,T - 0''.0059\,T^2 + 0''.00181\,T^3 \quad (124)$$
$$\phi_p = 90^\circ \quad (125)$$

where T is the time in Julian centuries since 1900.

3.4.3 Precession

Approximate precession, good for modest sub-arcsecond precision, may be rapidly calculated using the following rotation angles:

$$\alpha_p = 180^\circ + (0^\circ.6406161\,T + 0^\circ.0000839\,T^2 + 0^\circ.0000050\,T^3) \quad (126)$$
$$\delta_p = 0^\circ.5567530\,T - 0^\circ.0001185\,T^2 - 0^\circ.0000116\,T^3 \quad (127)$$
$$\phi_p = 180^\circ + 0^\circ.6406161\,T + 0^\circ.0003041\,T^2 + 0^\circ.0000051\,T^3 \quad (128)$$

where T is $(\mathrm{MJD_{out}} - \mathrm{MJD_{in}})/36525$ is the difference between the two epochs, in Julian centuries. This precession form shall be used to implement `PS_PRECESS_ROUGH`.

3.4.4 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 $(177.423^\circ, 31.113^\circ)$.

3.5 Sky to Tangent Plane

This section describes the transformation between celestial coordinates (R.A., Dec.) and local terrestrial coordinates (Az, Alt). This transformation is broken down into a number of steps as described below.

3.5.1 Reference Implementations

There are two reference implementations for the code to account for the motion of the Earth in space. The first are the sample routines provided by the IERS to accompany chaper 5 of IERS Bulletin 32.[12] The second reference implementation is the SOFA software package managed by the IAU.[13] Only the 2003-04-29 version of SOFA should be considered. The IERS code requires a few of the rotation matrix utility routines from SOFA.

Both implementations are in FORTRAN 77. The SOFA code has a more complex implementation of precession-nutation for backward compatibility with the pre 2003-01-01 conventions. The IERS code includes some tricks to achieve greater precision in the fundamental arguments of nutation, which the SOFA code omits. Therefore, the main reference for psLib should be the IERS code. Note that the IERS code calculates the transform from terrestrial to celestial coordinates, while the SOFA code calculates its inverse. This code may be using as a comparison for testing purposes.

3.5.2 Coordinate Systems

Figure 2 shows the transformation steps and intermediate coordinate systems between celestial and local terrestrial coordinate systems. The intermediate coordinate systems are defined below.

3.5.2.1 ICRS The official IAU-sanctioned celestial coordinate system is the International Celestial Reference System (ICRS). It is defined in terms of a number of radio sources whose positions have been measured using VLBI. It can be tied to the optical through the Hipparcos catalog. The ICRS has its origin at the solar system barycenter.

3.5.2.2 GCRS The Geocentric Celestial Reference System (GCRS) corresponds to the ICRS, but has its origin at the center of the Earth. The differences between the two systems are due to the velocity of the Earth (aberration), the position of the Earth (parallax), and general relativistic bending of light rays. There is no net rotation between the ICRS and the GCRS.

3.5.2.3 ITRS The International Terrestrial Reference System (ITRS) is a coordinate system which is fixed with respect to the Earth's crust.

3.5.2.4 Intermediate Coordinate Systems - CIP, CEO, TEO The transform between the GCRS and ITRS is conventionally decomposed into three parts in order to isolate the relatively rapid rotation of the Earth from the movement of the Earth's rotational axis in the GCRS and ITRS. All three sub-transforms are rigid rotations.

This decomposition results in two intermediate coordinate systems. Both of these share the same pole, known as the Celestial Intermediate Pole (CIP). The CIP is defined by its motion in the GCRS to match the Tisserand mean axis of the

[12] http://maia.usno.navy.mil/conv2003.html
[13] http://www.iau-sofa.rl.ac.uk

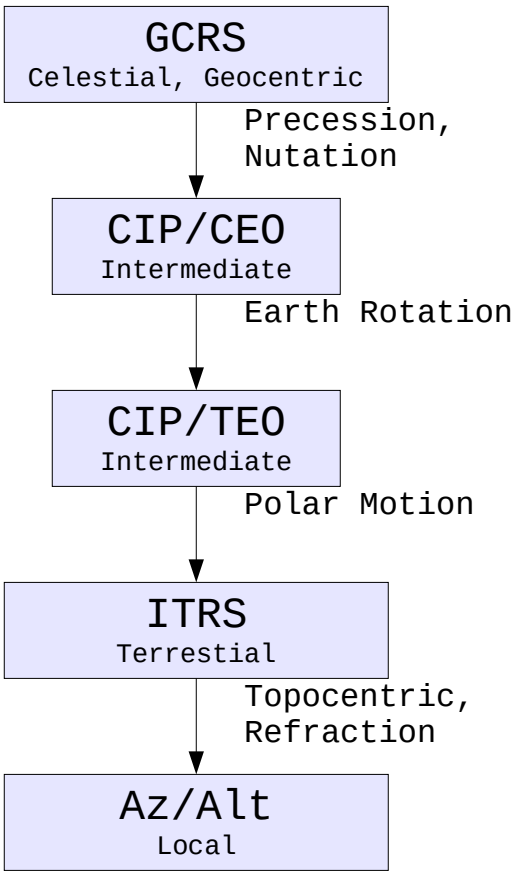

Figure 2: Coordinates systems and the transformations between them

Earth (Seidelmann 1982, Celesial Mechanics 27, 78-106), excluding motions with periods less than or equal to two days. The CIP approximates the angular momentum vector of the rotating Earth.

The X axes of the intermediate coordinate systems are known as the Celestial and Terrestrial Ephemeris Origins. (CEO and TEO). Both are defined to be non-rotating origins. A non-rotating origin is a point on the equator whose instantaneous motion is always orthogonal to the equator (Kaplan 2003 IAU XXV Joint Discussion 16[14]). Thus the CEO is defined by its position in the GCRS at some epoch and by the motion of the CIP in the GCRS since that date. Similarly the TEO is defined by its position in the ITRS at some epoch and the motion of the CIP in the ITRS since that date.

3.5.3 ICRS - GCRS

The transformation between barycentric (ICRS) and geocentric (GCRS) coordinates involves two components. These are the general relativistic deflection of light rays by the Sun's gravity, and aberration, due to the orbital motion of the Earth.

3.5.3.1 Gravitational Deflection The Sun's gravity bends the path of light rays which pass near it. To first order, a light ray is deflected by an angle of $4GM/c^2r_0$ radians, where G is the gravitational constant, M is the mass of the Sun, c is the speed of light, and r_0 is the point of closest approach to the light ray to the Sun. To the same order this is equal to the impact parameter - i.e. the point of closest approach if the light ray were not deflected. Note that $r_0/d = \tan(\theta)$, where d is the distance from the Earth to the Sun, and θ is the angular separation of the star from the center of the Sun.

There is a maximum deflection of 1.75 arc seconds if we set r_0 to the radius of the sun. Since the Sun bends light rays toward it, a star appears shifted away from the sun in the sky.

3.5.3.2 Aberration Aberration is the apparent change in direction of a ray of light in the reference frame of a moving observer. Traditionally the aberration calculation has been done with a linear expansion of the full relativistic expression, often neglecting all but the linear term in v/c, since the relativistic terms are on the order of a miliarcsecond. However, the full relativistic expression poses no challenge for modern computers, so psLib will use the following procedure to calculate aberration.

Suppose an observer has a velocity $\beta\hat{\beta}$, with respect to the Solar System barycenter, where β is in units of the speed of light, and $\hat{\beta}$ is a unit vector. Suppose also that the unit vector $\hat{r}$ points toward a star in the barycenter frame of reference (i.e. the "actual" position). and $\hat{r}'$ gives the direction of the star in the observer's frame, (i.e. the apparent position).

First, decompose $\hat{r}$ into components parallel and perpendicular to $\hat{\beta}$ by calculating $\mu = \hat{r} \cdot \hat{\beta}$ and $\vec{r}_\perp = \hat{r} - \mu\hat{\beta}$.

Next, use the following expression for relativistic beaming, modified slightly from equation 4.8b of Rybicki and Lightman:

$$\mu' = \mu + \beta\frac{\mu^2 - 1}{1 - \beta\mu} \tag{129}$$

where $\mu' = \hat{r}' \cdot \hat{\beta}$.

Now, the component of $\hat{r}'$ perpendicular to $\hat{\beta}$ (i.e. $\vec{r}'_\perp$) must point in the same direction as $\vec{r}_\perp$, but will have a different magnitude because $\hat{r}'$ is a unit vector. In other words, $\vec{r}'_\perp = a\vec{r}_\perp$, for some scalar a. So the next step is to calculate $a = \sqrt{(1 - \mu'^2)/\vec{r}_\perp}$.

Finally, reassemble the components of $\hat{r}' = \mu'\hat{\beta} + a\vec{r_\perp}$.

[14] http://aa.usno.navy.mil/kaplan/NROs%5BJD16proc%5D.pdf

3.5.4 GCRS - ITRS

The transformation between geocentric celestial coordinates and terrestrial coordinates is a solid body rotation due to the motion of the Earth is space. This is conventionally broken down into three components to isolate the relatively rapid rotation of the Earth from the motion of its rotational axis.

This section is largely a summary of Chapter 5 of IERS Technical Note 32[15] (hereafter IERS32), which is a description of the implementation of the Resoltions of the XXIVth General Assembly of the IAU, available from the same URL as above. These two documents describe a set of conventions which have been in effect since 2003-01-01. The conventions in effect before that date will not be implemented by psLib.

3.5.4.1 Precession/Nutation The transform between the GCRS and the CIP/CEO coordinate systems is described by the IAU 2000A precession-nutation model, which is accurate to the 0.2 mas level. For higher accuracy the user must apply corrections to the model, which are tabulated by the IERS.

3.5.4.1.1 IAU 200A Precession/Nutation Model : `psEOC_PrecessionModel` The IAU 2000A precession-nutation model may be calculated in the following way. First calculate the time t as the number of Julian centuries since 2000-01-01T12:00:00 TT.

Next calculate the fundamental arguments of nutation using equations (40) and (41) of IERS32, reproduced below:

$$\begin{aligned}
F_1 \equiv l &= Mean\ Anomaly\ of\ the\ Moon \\
&= 134.96340251^\circ + 1717915923.2178''t + 31.8792''t^2 + 0.051635''t^3 - 0.00024470''t^4, \\
F_2 \equiv l' &= Mean\ Anomaly\ of\ the\ Sun \\
&= 357.52910918^\circ + 129596581.0481''t - 0.5532''t^2 + 0.000136''t^3 - 0.00001149''t^4, \\
F_3 \equiv F &= L - \Omega \\
&= 93.27209062^\circ + 1739527262.8478''t - 12.7512''t^2 - 0.001037''t^3 + 0.00000417''t^4, \\
F_4 \equiv D &= Mean\ Elongation\ of\ the\ Moon\ from\ the\ Sun \\
&= 297.85019547^\circ + 1602961601.2090''t - 6.3706''t^2 + 0.006593''t^3 - 0.00003169''t^4, \\
F_5 \equiv \Omega &= Mean\ Longitude\ of\ the\ Ascending\ Node\ of\ the\ Moon \\
&= 125.04455501^\circ - 6962890.5431''t + 7.4722''t^2 + 0.007702''t^3 - 0.00005939''t^4 \\
F_6 \equiv l_{Me} &= 4.402608842 + 2608.7903141574 \times t, \\
F_7 \equiv l_{Ve} &= 3.176146697 + 1021.3285546211 \times t, \\
F_8 \equiv l_E &= 1.753470314 + 628.3075849991 \times t, \\
F_9 \equiv l_{Ma} &= 6.203480913 + 334.0612426700 \times t, \\
F_{10} \equiv l_{Ju} &= 0.599546497 + 52.9690962641 \times t, \\
F_{11} \equiv l_{Sa} &= 0.874016757 + 21.3299104960 \times t, \\
F_{12} \equiv l_{Ur} &= 5.481293872 + 7.4781598567 \times t, \\
F_{13} \equiv l_{Ne} &= 5.311886287 + 3.8133035638 \times t, \\
F_{14} \equiv p_a &= 0.024381750 \times t + 0.00000538691 \times t^2. \qquad (130)
\end{aligned}$$

Next calculate the quantities X, Y, and s, using expressions of the form:

$$\sum_j p_j t^j + \sum_j \sum_i [(a_{s,j})_i t^j \sin(\mathrm{ARG_{i,j}}) + (a_{c,j})_i t^j \cos(\mathrm{ARG_{i,j}})], \qquad (131)$$

[15] http://maia.usno.navy.mil/conv2003.html

where the $\mathrm{ARG_{i,j}}=\sum_{\mathrm{k}} \mathrm{w_{i,j,k}F_k}$ represent linear combinations of the fundamental arguments of nutation.

The constants p_j, $w_{i,j,k}$, $(a_{\mathrm{s},j})_i$, and $(a_{\mathrm{c},j})_i$ are given in the ASCII files: tab5.2a.txt[16] (for X), tab5.2b.txt[17] (for Y), and tab5.2c.txt[18] (for $s + XY/2$). Note that the expansion is given for $s + XY/2$, since this series converges more rapidly than the one for s alone.

Each file contains a human-readable header, which includes the polynomial coeficients, p_j under the heading "Polynomial part". The data part of the file lists the remaining constants, with rows cycling first through i, and then through j. There is a separate heading each time j increments. Each row contains the following columns:

- col 1 - A running index of rows in the table.
- col 2 - The sine coeficients, $(a_{\mathrm{s},j})_i$
- col 3 - The cosine coeficients, $(a_{\mathrm{c},j})_i$
- cols 4 - 17 The weighting factors for the fundamental arguments of nutation, $w_{i,j,k}$.

A FORTRAN reference implementation for the precession/nutation model is available from the IERS.[19] The psLib results should agree with the reference implementation to within the limits of numerical precision.

3.5.4.1.2 Corrections to the Model : `psEOC_PrecessionCorr` Corrections to X, and Y may be obtained from the IERS as part of Bulletin A, or B. It is recommended to use the values published daily by USNO in the table `finals2000A.daily`[20], which has the format described by `readme.finals2000A`[21]. The quantities of interest are labeled dX and dY. Note that UT1$-$UTC and the polar motion values are obtained from this same table.

By convention, nutation terms with periods of less than two days are accounted for by the corresponding polar motion. So it is sufficient to interpolate the corrections tabulated daily by the IERS, and take the result as instantaneous values.

3.5.4.1.3 Spherical Rotation from Polar Coordinates : `psSphereRot_CEOtoGCRS` In order to relate the values X, Y, and s to the rotation components, the rotation matrix below must be used. The definitions of X, Y, and s transform from the CIP/CEO system to the GCRS using IERS32 equation (10), reproduced below:

$$\begin{pmatrix} 1-aX^2 & -aXY & X \\ -aXY & 1-aY^2 & Y \\ -X & -Y & 1-a(X^2+Y^2) \end{pmatrix} \cdot R_3(s), \tag{132}$$

where R_3 denotes a rotation about the Z axis, $a = 1/(1 + \sqrt{1-(X^2+Y^2)})$, and X and Y are expressed in radians. A FORTRAN reference implementation for this calculation is given by the IERS.[22]

Note that above we gave the expression for the transform toward celestial coordinates (upward in Figure 2), in order to match the IERS reference code. The inverse transform may be found by inverting the resulting rotation.

[16] http://maia.usno.navy.mil/conv2000/chapter5/tab5.2a.txt
[17] http://maia.usno.navy.mil/conv2000/chapter5/tab5.2b.txt
[18] http://maia.usno.navy.mil/conv2000/chapter5/tab5.2c.txt
[19] http://maia.usno.navy.mil/conv2000/chapter5/XYS2000A.f
[20] http://maia.usno.navy.mil/ser7/finals2000A.daily
[21] http://maia.usno.navy.mil/ser7/readme.finals2000A
[22] http://maia.usno.navy.mil/conv2000/chapter5/BPN2000.f

3.5.4.2 Earth Rotation The transform from the CIP/CEO to CIP/TEO coordinate systems is a rotation about the CIP (i.e. the Z axis) by an angle known as the "Earth Rotation Angle". By definition the Earth Rotation Angle is given by equation (13) of IERS32, reproduced below:

$$\theta(T_u) = 2\pi(0.7790572732640 + 1.00273781191135448T_u), \quad (133)$$

where T_u is the Julian UT1 date minus 2451545.0 .

3.5.4.3 Polar Motion The motion of the CIP in the ITRS is known as "polar motion". Similarly to precession/nutation, the instantaneous position of the CIP in the ITRS is specified by the quantites x_p, and y_p, and a third quantity, s', which give the position of the TEO with respect to the ITRS. The values of x_p and y_p are published daily by the IERS,[23] with a format described by their `readme.finals2000A`[24]. The UT1$-$UTC, and the precession/nutation corrections (discussed elsewhere in this document) come from this same source.

3.5.4.3.1 Polar Motion from Bulletin : `psEOC_GetPolarMotion` The polar motion coordinates should be interpolated using a third order polynomial, as described in IERS Gazette #13 [25], which gives a FORTRAN reference implementation of the correct procedure.

The values published by the IERS are smoothed to remove noise and variations on the timescale of a day or less. There are two sources of short timescale variations - tidal effects on the order of 0.1 milliarcseconds, and short period nutation terms on the order of 15 microarcseconds. Both of these effects may be modeled and added to the interpolated values for higher accuracy.

The tidal effects should be included by using the Ray tidal model given in IERS Gazette #13. The definition of this correction is provided below (Section 3.5.6).

3.5.4.3.2 Polar Motion Nutation Correction : `psEOC_NutationCorr` By definition of the CIP, nutation terms with periods less than 2 days are not included in the IAU 2000A precession/nutation model. So these motions must be compensated for by their equivalent polar motions. These may be calculated using a form similar to that of the precession/nutation X, and Y. The constants to use are given in Table 5.1 of IERS32. Note that only the terms with periods less than 2 days should be used.

The quantity s' may be approximated with microarcsecond accuracy over this century by $s' = -4.7 \times 10^{-5} t$ in arcseconds. There is no need to apply short timescale corrections to s'.

3.5.4.3.3 Spherical Rotation from Polar Motion : `psSphereRot_ITRStoTEO` The transform from the ITRS to the CIP/TEO frame can be constructed by first rotating about the X axis by y_p, then rotating about the X axis by x_p, and finally rotating about the Z axis by s'. The IERS reference implementation for this is given in the subroutine POM2000 [26]. Note that we describe the transform toward celestial coordinates (upward in Figure 2), in order to match the reference implementation.

[23] http://maia.usno.navy.mil/ser7/finals2000A.daily
[24] http://maia.usno.navy.mil/ser7/readme.finals2000A
[25] http://maia.usno.navy.mil/iers-gaz13
[26] http://maia.usno.navy.mil/conv2000/chapter5/POM2000.f

3.5.5 Universal Time (UT1)

Since 2003-01-01, UT1 has been defined as directly proportional to the Earth Rotation Angle (see IERS Technical Note 32[27]). Previous to that date, a different definition was in effect (see IERS Technical Note 21[28]). We will always use the post-2003 definition.

UT1 is continuously measured by the International Earth Rotation Service, and tabulated values of the offset of UT1 from UTC are published at regular intervals, along with predicted future values. IERS Bulletin A gives "rapid response" values necessary for real-time and near real-time data analysis (such as Pan-STARRS Otis and IPP subsystems). Bulletin B gives the results of a final, definitive data reduction. An amalgam of Bulletin A and B values is published daily on the IERS website[29] along with a desciption of the format[30].

The UT1 offsets should be interpolated using the prescription of IERS Gazette #13[31]. This entails using a third order polynomial to interpolate the table values, and then applying a model for diurnal and semi-diurnal fluctuations due to tidal effects. An example implimentation[32] written in Fortran is available. The interpolated value of dT must then have the tidal correction from the Ray Tidal Model applied.

3.5.6 Ray Tidal Model : `psEOC_PolarTideCorr`

The Ray Model tidal corrections to X, Y, and dT are given by the the Fortran code listed below. The input information is the epoch of interest in MJD, while the output results are the corrections C_x, C_y, and C_dt, which are in turn added to the interpolated values determined above.

```
C
      SUBROUTINE RAY (RJD,CORX,CORY,CORT)
C
C   THIS SUBROUTINE IMPLEMENTS THE RAY MODEL FOR
C   DIURNAL/SUBDIURNAL TIDES.  IT USES THE SIMON ET AL.
C   FUNDAMENTAL ARGUMENTS.  THE CORRECTIONS IN X AND Y ARE IN
C   UNITS OF SEC. OF ARC AND UT1-UTC IN SEC. OF TIME.  THESE
C   CORRECTIONS SHOULD BE ADDED TO "AVERAGE" EOP VALUES TO GET
C   ESTIMATES OF THE INSTANTANEOUS VALUES.
C
C     PARAMETERS ARE :
C     RJD   - EPOCH OF INTEREST GIVEN IN MJD
C     CORX  - TIDAL CORRECTION IN X (SEC. OF ARC)
C     CORY  - TIDAL CORRECTION IN Y (SEC. OF ARC)
C     CORT  - TIDAL CORRECTION IN UT1-UTC (SEC. OF TIME)
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DOUBLE PRECISION
     .   L,        LPRIME
```

[27] IERS Technical Note 32 - http://maia.usno.navy.mil/conv2003.html
[28] IERS Technical Note 21 - http://maia.usno.navy.mil/conventions.html
[29] IERS Bulletin A & B - http://maia.usno.navy.mil/ser7/finals2000A.daily
[30] IERS finals2000A.daily table format - http://maia.usno.navy.mil/ser7/readme.finals2000A
[31] IERS Gazette #13 - http://maia.usno.navy.mil/iers-gaz13
[32] interp.f - ftp://maia.usno.navy.mil/dist/interp.f

```
      HALFPI = 1.5707963267948966d0
      T = (RJD - 51544.5D0)/36525.0D0
      L = -0.00024470d0*T**4 + 0.051635d0*T**3 + 31.8792d0*T**2
     .  + 1717915923.2178d0*T + 485868.249036d0
      L = DMOD(L,1296000d0)
      LPRIME = -0.00001149d0*T**4 - 0.000136d0*T**3
     .  -  0.5532d0*T**2
     .  + 129596581.0481d0*T + 1287104.79305d0
      LPRIME = DMOD(LPRIME,1296000d0)
      CAPF = 0.00000417d0*T**4 - 0.001037d0*T**3 - 12.7512d0*T**2
     .  + 1739527262.8478d0*T + 335779.526232d0
      CAPF = DMOD(CAPF,1296000d0)
      CAPD = -0.00003169d0*T**4 + 0.006593d0*T**3 - 6.3706d0*T**2
     .  + 1602961601.2090d0*T + 1072260.70369d0
      CAPD = DMOD(CAPD,1296000d0)
      OMEGA = -0.00005939d0*T**4 + 0.007702d0*T**3
     .  + 7.4722d0*T**2
     .  - 6962890.2665d0*T + 450160.398036d0
      OMEGA = DMOD(OMEGA,1296000d0)
      THETA = (67310.54841d0 +
     .        (876600d0*3600d0 + 8640184.812866d0)*T +
     .         0.093104d0*T**2 -
     .         6.2d-6*T**3)*15.0d0 + 648000.0d0
      ARG7 = DMOD((-L - 2.0D0*CAPF - 2.0D0*OMEGA + THETA)
     .     * 3.14159265D0/648000.0D0,6.28318530718D0) - HALFPI
      ARG1 = DMOD((-2.0d0*CAPF - 2.0d0*OMEGA + THETA)
     .     * 3.14159265D0/648000.0D0,6.28318530718D0) - HALFPI
      ARG2 = DMOD((-2.0d0*CAPF + 2.0d0*CAPD - 2.0d0*OMEGA
     .       + THETA)
     .     * 3.14159265D0/648000.0D0,6.28318530718D0) - HALFPI
      ARG3 = DMOD(THETA *
     .        3.14159265D0/648000.0D0,6.28318530718D0)
     .     + HALFPI
      ARG4 = DMOD((-L - 2.0d0*CAPF - 2.0D0*OMEGA + 2.0d0*THETA)
     .     * 3.14159265D0/648000.0D0,6.28318530718D0)
      ARG5 = DMOD((-2.0D0*CAPF - 2.0D0*OMEGA + 2.0d0*THETA)
     .     * 3.14159265D0/648000.0D0,6.28318530718D0)
      ARG6 = DMOD((-2.0d0*CAPF + 2.0d0*CAPD - 2.0d0*OMEGA
     .     + 2.0d0*THETA)
     .     * 3.14159265D0/648000.0D0,6.28318530718D0)
      ARG8 = DMOD((2.0d0*THETA)
     .     * 3.14159265D0/648000.0D0,6.28318530718D0)
      CORX = - 0.026D0*DSIN(ARG7) + 0.006D0*DCOS(ARG7)
     .       - 0.133D0*DSIN(ARG1) + 0.049D0*DCOS(ARG1)
     .       - 0.050D0*DSIN(ARG2) + 0.025D0*DCOS(ARG2)
     .       - 0.152D0*DSIN(ARG3) + 0.078D0*DCOS(ARG3)
     .       - 0.057D0*DSIN(ARG4) - 0.013D0*DCOS(ARG4)
     .       - 0.330D0*DSIN(ARG5) - 0.028D0*DCOS(ARG5)
```

```
.       - 0.145D0*DSIN(ARG6) + 0.064D0*DCOS(ARG6)
.       - 0.036D0*DSIN(ARG8) + 0.017D0*DCOS(ARG8)
 CORY = - 0.006D0*DSIN(ARG7) - 0.026D0*DCOS(ARG7)
.       - 0.049D0*DSIN(ARG1) - 0.133D0*DCOS(ARG1)
.       - 0.025D0*DSIN(ARG2) - 0.050D0*DCOS(ARG2)
.       - 0.078D0*DSIN(ARG3) - 0.152D0*DCOS(ARG3)
.       + 0.011D0*DSIN(ARG4) + 0.033D0*DCOS(ARG4)
.       + 0.037D0*DSIN(ARG5) + 0.196D0*DCOS(ARG5)
.       + 0.059D0*DSIN(ARG6) + 0.087D0*DCOS(ARG6)
.       + 0.018D0*DSIN(ARG8) + 0.022D0*DCOS(ARG8)
 CORT = + 0.0245D0*DSIN(ARG7) + 0.0503D0*DCOS(ARG7)
.       + 0.1210D0*DSIN(ARG1) + 0.1605D0*DCOS(ARG1)
.       + 0.0286D0*DSIN(ARG2) + 0.0516D0*DCOS(ARG2)
.       + 0.0864D0*DSIN(ARG3) + 0.1771D0*DCOS(ARG3)
.       - 0.0380D0*DSIN(ARG4) - 0.0154D0*DCOS(ARG4)
.       - 0.1617D0*DSIN(ARG5) - 0.0720D0*DCOS(ARG5)
.       - 0.0759D0*DSIN(ARG6) - 0.0004D0*DCOS(ARG6)
.       - 0.0196D0*DSIN(ARG8) - 0.0038D0*DCOS(ARG8)
 CORX = CORX * 1.0d-3
 CORY = CORY * 1.0d-3
 CORT = CORT * 0.1d-3
 RETURN
 END
```

The Polar motion X, and Y coordinates are also important for determining the orientation of the Earth with respect to the sky. This is also given in the IERS publications references above, and should be interpolated in the same way.

3.5.7 Longitude

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

$$1^\circ = 240s \tag{134}$$

3.5.8 ITRS - Alt/Az

3.5.8.1 Orientation of the Observer An observer's astronomical longitude and latitude give the orientation of the local vertical with respect to the ITRS. Note that these coordinates can be approximated by the geographic longitude and latitude of the observatory, but their exact values must be calibrated from observation of stars with known coordinates in the ICRS.

The transform from the ITRS to Az/Alt in the absence of atmospheric refraction is first a rotation about the Z axis by the observer's astronomical longitude, and then a rotation about the Y axis of 90 degrees minus the observer's astronomical latitude, followed by a rotation about the Z axis of 180 degrees so that North is zero azimuth.

3.5.8.2 Atmospheric Refraction *The following discussion is adapted from an article by Ken Chambers*

The hypsometric structure and index of refraction of the Earth's atmosphere produces:

- atmospheric refraction, resulting in an apparent positional displacement of astronomical objects towards smaller topocentric zenith distances,
- chromatic dispersion, along great circles intersecting the topocentric zenith with shorter wavelengths having smaller zenith distances; and
- extinction, from scattering and absorption of light by atmospheric gases (including water vapor) and aerosols.

Atmospheric refraction $R(\lambda) = z_{vac} - z_1$ is the difference between the topocentric zenith angle *in vacuo*, where the index of refraction is $n \equiv 1$, and the observed refracted zenith angle at the observatory z_1. (All subscripts "1" in this discussion indicate the local values of quantities at a given observatory site, and all units are SI.) There are various ways to express the equation for zenith angle refraction; common ones include the "general equation", e.g. *s1936,s1962*, the Auer & Standish transformation *as1979,as2000*, and the Saastamoinen approximation *saas73a,saas73b,saas73c*. All are derived from the "refractive invariant", the fact that along the refracted light path described by the locus of points $r(z, n)$, where r is the radial distance from the center of the Earth and z and n are the local values of the refracted zenith angle and the index of refraction of air respectively, the product $nr\sin z = \text{constant}$ remains invariant.

We have revisted the Saastamoinen approximation in search of an improved analytical expression accurate to ~ 1 mas up to zenith angles of 75 degrees. Although different approximations are made at different stages, both the general equation and the Saastamoinen approximation lead to an expression for the refraction $R(\lambda)$ in the form of a series in odd powers of $\tan z_1$:

$$R(\lambda) = R_1 \tan z_1 + R_3 \tan^3 z_1 + R_5 \tan^5 z_1 + R_7 \tan^7 z_1 + R_9 \tan^9 z_1 + \ldots \tag{135}$$

where we have chosen to label the coefficients with a subscript reflecting the exponent of the $\tan z_1$ terms. We find (c.f. Stone 1996) [33]

$$R(\lambda) = \gamma_1 (1 - h_1/r_1) \tan z_1 + [\gamma_1 (\gamma_1/2 - h_1/r_1) + \delta_1] \tan^3 z_1 \tag{136}$$

in radians, where we define for convenience the variable

$$\gamma_1 = n_1 - 1$$

where $n_1(\lambda)$ is the index of refraction of air at the observatory, which is dependent on the observatory's altitude and the meteorological conditions at the time of the observation (see Section 3 below). Each of the other variables Eq.(2) take detailed discussions which, for the sake of clarity, are divided into separate subsections.

3.5.8.2.1 Observatory height The height of the observatory from the geometric center of the Earth is

$$r_1 = r_e + r_h, \qquad r_h \approx r_n + r_o, \tag{137}$$

where r_e is the local radius of the reference ellipsoid, r_h is the local height above the reference ellipsoid, r_n is the local geoid height, normal to the reference ellipsoid, r_o is the orthometric height, which is the height above the geoid (or Mean Sea Level) in the direction of normal gravity (i.e. a local plumb line).

[33] The R_1 coefficient and the first two terms in the R_2 coefficient in equation 2 above give results similar to that of Stone's equation (4), if it is modified such that the κ factor multiplies only his β, rather than the whole coefficient as is done for both coefficients. We suspect this is a typographical error; his equation equation as written gives much greater residuals when compared to the Pulkovo Refraction Tables than he reports in his Table 2, whereas the agreement is comparable, if his equation is modified as described, and computed for the Pulkovo baseline model of $\lambda = 590$nm, $15\deg C$, 101325 Pa, zero water vapor, and mean sea level at latitude 45 degrees. An exact comparison is not possible because the residuals in his Table 2 are averages computed from a wide but unspecified range of meteorological conditions.

3.5.8.2.2 The magnitude of normal gravity at the observatory The local magnitude of normal gravity [34] g_1 is the acceleration due to the combination of the gravity of the ellipsoid, the local mass distribution, and the centripetal acceleration from the Earth's rotation, the vector sum being directed opposite to the local zenith by definition, being close but not identical to the normal to the ellipsoidal, but not intersecting the the geometric center of the Earth. i.e. the atmospheric topocentric zenith differs from the geocentric astronomical zenith, the impact of this difference on calculating the atmospheric refraction is minimal, see *seid1992*. The acceleration at the observatory is

$$g_1(r_h, \phi) = g(\phi) \left[1 - 2(1 + f + m_r - 2f \sin^2 \phi) \left(\frac{r_h}{a} \right) + 3 \left(\frac{r_h}{a} \right)^2 \right] \tag{138}$$

where

$$g(\phi) = g_e \left(\frac{1 + k_s \sin^2 \phi}{\sqrt{1 - \epsilon^2 \sin^2 \phi}} \right) \tag{139}$$

and ϕ is the latitude of the observatory and the other constants are given in Table 2.

Table 1: WGS-84 World Geodetic System Reference Ellipsoid for GPS

Semi-major axis a	6356752.3142 m
Flattening $f = (a - b)/a$	0.003352811
Eccentricity $\epsilon = \sqrt{a^2 + b^2}/a$	0.081819
Polar gravity g_p	9.8321849378 ms^{-2}
Equatorial gravity g_e	9.7803253359 ms^{-2}
Somigliana's Constant $k_s = ((b/a)(g_p/g_e) - 1)$	0.001931853
Angular velocity of the Earth ω	0.00007292115 rad/s
GM_{earth} including atmosphere	3986004.418$\times 10^8 m^3 s^{-2}$
Gravity ratio $m_r = \omega^2 a^2 b/(GM)$	0.003449787

3.5.8.2.3 The scale height above the observatory The scale height of the atmosphere above the observatory is

$$h_1 = \mathcal{Z}_1 \mathcal{R} T_1 / g_1 M_a [1 - x_w (1 - M_w/M_a)], \tag{140}$$

where T_1 is the local air temperature at the observatory in degrees Kelvin, $\mathcal{R} = 8.314472 \mathrm{J\ mol^{-1} K^{-1}}$ is the gas constant,

$$\begin{aligned} \mathcal{Z}_1 = 1 - (P_1/T_1) \left[a_0 + a_1 t_1 + a_2 t_1^2 + (b_0 + b_1) t_1 x_w + (c_0 + c_1 t_1) x_w^2 \right] \\ + (P_1/T_1)^2 (d + e x_w^2) \end{aligned} \tag{141}$$

is the compressibility of moist air at local air temperature $t_1 = T_1 - 273.15$ in degrees Celsius. The values of the other constants in $\mathcal{Z}_1$ are given in Appendix A. The quantity $M_a = 0.0289635 + 1.2001 \times 10^{-8}(x_{c1} - 400)$ kg/mole is the molar mass of dry air with a CO_2 concentration x_{c1} in μmol/mol, $M_w = 0.018015$ kg/mole is the molar mass of water vapor, and x_w is the molar fraction of water vapor in moist air, which depends on the local humidity.

To calculate both the scale height h_1 and the index of refraction of moist air at the observatory n_1 we need to calculate x_{w1}, the molar fraction of water vapor from local measurements of the relative humidity or, much better, the dew point. Simple formulas such as *davis1992* are only suitable above $0 \deg C$, whereas at Mauna Kea observatories the temperature is often

[34] Stone uses a common *[*e.g.][]allen1973,seid1992,allen2001 expression for the normal gravity as a function of lattitude and altitude, which apparently first appeared in *lamb1949*. However its derivation is unclear – cited as *"Some notes on the calculation of the geopotential", unpublished manuscript, lamb1949.*

below $0\deg C$, and occasionally the surrounding ground is covered in snow or ice, which also alters saturation vapor pressure. Thus, we adopt the best available equation for the saturation water vapor pressure p_{sv}, the IAPWS equations *huang1998*. Haung's equations are:

$$\begin{array}{ll} \Omega = T + K_9/(T - K_{10}) & A = \Omega^2 + K_1\Omega + K_2 \\ B = K_3\Omega^2 + K_4\Omega + K_5 & C = K_6\Omega^2 + K_7\Omega + K_8 \\ X = -B + \sqrt{B^2 - 4AC} & p_{sv}(t) = 10^6(2C/X)^4 \end{array} \tag{142}$$

For saturation vapor pressure over ice or snow, use

$$\begin{array}{ll} \Theta = T/273.16 & Y = A_1(1 - \Theta^{-1.5}) + A_2(1 - \Theta^{-1.25}) \\ p_{sv}(t) = 611.657e^Y & \end{array} \tag{143}$$

The constants are given in Table 1.

Table 2: Constants for Compressibility and Humidity Equations

$a_0 = 1.58123 \times 10^{-6}\text{KPa}^{-1}$	$K_1 = 1.16705214528E + 03$
$a_1 = -2.9331 \times 10^{-8}\text{Pa}^{-1}$	$K_2 = -7.24213167032E + 05$
$a_2 = 1.1043 \times 10^{-8}\text{K}^{-}1\text{Pa}^{-1}$	$K_3 = -1.70738469401e + 01$
$b_0 = 5.707 \times 10^{-6}\text{KPa}^{-1}$	$K_4 = 1.20208247025E + 04$
$b_1 = -2.051 \times 10^{-8}\text{Pa}^{-1}$	$K_5 = -3.23255503223E + 06$
$c_0 = 1.9898 \times 10^{-4}\text{KPa}^{-1}$	$K_6 = 1.49151086135E + 01$
$c_1 = -2.376 \times 10^{-6}\text{Pa}^{-1}$	$K_7 = -4.82326573616E + 03$
$d = 1.83 \times 10^{-11}\text{K}^2\text{Pa}^{-2}$	$K_8 = 4.05113405421E + 05$
$e = -0.765 \times 10^{-8}{}^2\text{KPa}^{-2}$	$K_9 = -2.38555575678E - 01$
	$K_{10} = 6.50175348448E + 02$
	$A_1 = -13.928169$
	$A_2 = 34.7078238$

Now, to calculate the mole fraction of water vapor x_w, we need the so called "enhancement factor"

$$f(p,t) = a' + b'p + c't^2 \tag{144}$$

where $a' = 1.00062$,$b' = 3.14 \times 10^{-8}$, and $c' = 5.60 \times 10^{-7}$, and where p and t are the air pressure in Pascals and air temperature.

If you have the more precise measurement of the dew point t_d (or frost point)then

$$x_w = f(p,t) \times p_{sv}(t_d)/p. \tag{145}$$

On the other hand if you only have the relative humidity RH, a less accurate expression is

$$x_w = (RH/100) \times f(p,t) \times p_{sv}(t_d)/p. \tag{146}$$

3.5.8.2.4 The index of refraction of moist air at the observatory The Ciddor equation for the index of refraction of moist air *ciddor1996* has been adopted by the International Association of Geodesy (IAG) as the standard as it is believed to provide the most accurate results under the largest range of wavelength, temperature, and humidity conditions (300 to 1690 nm; -40 to 100 C; 0-80% RH). Note for astronomy, the air temperature at the tropopause in the 1976 standard

atmosphere is 216.6 K, below the stated range ofvalidity. Astronomical observers often observe up to 90% RH, where water droplets can form and change the effective index of refraction. Note that developments in the equation for the index of refraction of moist air have been poorly tracked in the astronomical literature and it is critical to examine in every application what equation is actually being used. [35]

The index of refraction of air at standard temperature and pressure is *ciddor1996*

$$\gamma_{as} = 10^{-8}\left(\left[\frac{k_1}{k_0 - \sigma^2}\right] + \left[\frac{k_3}{k_2 - \sigma^2}\right]\right), \tag{147}$$

where $\sigma = 1/\lambda$ is the wavenumber of wavelength of light in microns. Adjusting for the (annually varying and secularly increasing) value of atmospheric CO_2 concentration x_{CO2} in units of μmole/mole, the expression for dry air becomes

$$\gamma_{axs} = \gamma_{as}\left[1 + 5.34 \times 10^{-7}(x_{CO2} - 450\ \mu\mathrm{mole/mole})\right]. \tag{148}$$

For water vapor under standard conditions, Ciddor finds

$$\gamma_{ws} = 1.022 \times 10^{-8}\left[\omega_0 + \omega_1\sigma^2 + \omega_2\sigma^4 + \omega_3\sigma^6\right]. \tag{149}$$

Following *owens1967*, the indicies can be combined in proportion to their densities, thus the index of refraction of moist air at the observatory $n_1 = \gamma_1 + 1$ is given by

$$\gamma_1 = (\rho_a/\rho_{axs})\gamma_{axs} + (\rho_w/\rho_{ws})\gamma_{ws}, \tag{150}$$

where

$$\begin{aligned} \rho_a &= (1 - x_w)P_1M_a/(\mathcal{Z}_m\mathcal{R}T_1) \\ \rho_w &= x_wP_1M_w/(\mathcal{Z}_m\mathcal{R}T_1) \\ \rho_{axs} &= P_{STP}M_a/(\mathcal{Z}_a\mathcal{R}T_{STP}) \end{aligned} \tag{151}$$

and ρ_{ws} is given in Table 3.

Table 3: Constants in the Ciddor Eq. for index of refraction of moist air

$k_0 = 23.0185\,\mu\mathrm{m}^{-2}$	$\omega_0 = 295.235\,\mu\mathrm{m}^{-2}$	$P_{STP} = 101325$ Pa
$k_1 = 5792105\,\mu\mathrm{m}^{-2}$	$\omega_1 = 2.6422\,\mu\mathrm{m}^{-2}$	$T_{STP} = 288.15$ K
$k_2 = 57.362\,\mu\mathrm{m}^{-2}$	$\omega_2 = -0.03238\,\mu\mathrm{m}^{-4}$	$\mathcal{Z}_a = 0.9995922115$
$k_3 = 167917\,\mu\mathrm{m}^{-2}$	$\omega_3 = 0.004028\,\mu\mathrm{m}^{-6}$	$\rho_{ws} = 0.00985938\,\mathrm{kg\ m}^3$

3.5.8.2.5 The tropopause term in the equation of refraction The final term in the Refraction Equation (2) is δ_1, which comes from our re-derivation of the Saastamoinen approximation:

$$\delta_1 = 5\left(\frac{h_1}{r_1T_1}\right)^2\left[\frac{\gamma_{ft}T_{ft}^2 - \gamma_tT_t^2}{1 - (h_1\beta/T_1)} + \gamma_tT_t^2\right] \tag{152}$$

[35] Edlen's (1953) original fit to the available data covered the wavelength range 2752 to 6440 /AA with reasonable residuals. His 1953 constants still survive in the astronomical literature in the equations of *allen1973*; *stone1996*; *allen2001*; Roe (2002 - who extrapolates the 1953 equation to K band to correct for dichoric adaptive optics) and in the computer codes SLALIB and ZEEMAX. None discuss the range of validity. However, Elden himself revised them *elden1966*, and these were further discussed and updated by *pr1972, bd1993, bd1994, ciddor1996, bp1998. rueg1998* and *sz2004* make convincing arguments for the Ciddor equation, and the latter's approach is followed here.

where β is the lapse rate of the troposphere, and the index of refraction of the free troposphere is given by

$$\gamma_{ft} = \gamma_t (T_{ft}/T_t)^{-(T_1/h_1\beta)-1} \tag{153}$$

and the index of refraction of the tropopause is given by

$$\gamma_t = \gamma_1 \exp\left[\frac{T_1(r_t - r_1)}{T_t h_1}\right] \tag{154}$$

where β is the lapse rate in K/m, the temperature of the tropopause T_t, and height of the tropopause r_t are all determined from contemporaneous meterological data (radiosonde or modern forecast models). Then the temperature of the free troposphere is given by

$$T_{ft} = T_t - \beta(r_t - r_1). \tag{155}$$

3.5.8.2.6 Calculating the atmospheric refraction from both the observed and true zenith angle The monochromatic refraction can now be calculated for any given wavelength λ_{air} (formally only within the range of validity - 300 to 1670 nm) given the altitude of the observatory h_1; contemporaneous meteorological measurements at the observatory of air temperaure T_1 (K); atmospheric pressure P_1 (Pa); percent relative humidity RH, or preferably dew point temperature t_d (degC); as well as a small set of additional meteorological data: the lapse rate of the troposphere β (K/m); the radius of the tropopause $r_t = r_e + h_t$(m), where h_t is the height above MSL of the tropopause; the temperature of the tropopause T_t(K); and the atmospheric concentration of CO_2 x_{CO2} (μmole/mole). The characterization of the troposphere and tropopause can be determined either by interpolation between radiosonde measurements or from hourly updated meteorological models available at many observatory sites, or, worst case, simply adopting the 1976 Standard Atmosphere values:[36] $\beta = -0.0065$ K/m, $h_t = 11000$m, $T_t = 216.6$ K and assuming $x_{C02} = 375\mu$mole/mole.[37]

If the number of electrons being created from the illumination from the source in the interval of wavelength $d\lambda$ is $N_\lambda d\lambda$, then the mean refraction is

$$\bar{R} = \frac{\int R(\lambda) N_\lambda d\lambda}{\int N_\lambda d\lambda} \tag{156}$$

3.5.9 Air Mass and Extinction

By Laplace's theorem, the monochromatic airmass (mass per unit area along the refracted path) is

$$M(z_1) = (P_1/g_1) R(\lambda)/\sin z_1 \tag{157}$$

in kg/m^2. Thus

$$M(z_1) = (P_1/g_1)\left(\gamma_1(1 - h_1/r_1)\sec z_1 + [\gamma_1(\gamma_1/2 - h_1/r_1) + \delta_1]\sec^3 z_1\right). \tag{158}$$

This is generally normalized to P_1/g_1, i.e. in spite of the daily changes in barometric pressure, and thus daily changes in the true mass of air over the observatory, the resulting change in extinction is generally treated as a drift in photometric zeropoint. For a survey program like Pan-STARRS, one could instead normalize to a standard barometric pressure (i.e. the altitute pressure), and thus during a low in atmospheric pressure, the airmass would be less than 1 at zenith, and during periods of high pressure the airmass would be greater than 1 at zenith. From the variation with zeropoint and temperature and barometric pressure, this would remove most of the observed variation in zeropoint in the CFHT legacy program. (Magnier, private comunication).

[36] *seid1992* contains a typographical error quoting this value in units of K/km

[37] Closed rooms have higher CO_2 concentration, thus the STP laboratory measurements have concentrations near 450 μmole/mole (the prefered unit to parts per million per volume). The secular increase of atmospheric CO_2 in the industrial age is well documented, with an annual cycle superimposed due to terrestrial biomass and ocean exchange.

The mean airmass is then

$$\bar{M}(z_1) = \left(\frac{P_1}{g_1}\right) \int \left(\gamma_1(1 - h_1/r_1)\mathrm{sec}z_1 + [\gamma_1(\gamma_1/2 - h_1/r_1) + \delta_1]\mathrm{sec}^3 z_1\right) N_\lambda d\lambda. \quad (159)$$

and depends weakly on the filter bandpass. Use of this more accurate expression for airmass should lead to improved extinction corrections at high airmass.

3.6 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):

$$x_p = \rho_x x \quad (160)$$
$$y_p = \rho_y y \quad (161)$$
$$(162)$$

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

3.6.1 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:

$$\sin\theta = \sin\delta \sin\delta_p + \cos\delta \cos\delta_p \cos(\alpha - \alpha_p) \quad (163)$$
$$\cos\theta \cos\phi = \sin\delta \cos\delta_p - \cos\delta \sin\delta_p \cos(\alpha - \alpha_p) \quad (164)$$
$$\cos\theta \sin\phi = -\cos\delta \sin(\alpha - \alpha_p) \quad (165)$$

For the inverse transformations, the equivalent relationships are:

$$\sin\delta = \sin\theta \sin\delta_p + \cos\theta \cos\delta_p \cos\phi \quad (166)$$
$$\cos\delta \cos(\alpha - \alpha_p) = \sin\theta \cos\delta_p - \cos\theta \sin\delta_p \cos\phi \quad (167)$$
$$\cos\delta \sin(\alpha - \alpha_p) = -\cos\theta \sin\phi \quad (168)$$

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

$$x = R_\theta \sin\phi \quad (169)$$
$$y = -R_\theta \cos\phi \quad (170)$$

and the inverse:

$$R_\theta = \sqrt{x^2 + y^2} \quad (171)$$
$$\phi = \mathrm{atan}(-y, x) \quad (172)$$

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

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

$$\alpha - \alpha_p = \arctan(\sin\alpha, \cos\alpha) \quad (173)$$
$$\delta = \arcsin(\sin\delta) \quad (174)$$
$$(175)$$

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

3.6.1.1 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:

$$x = \frac{\cos\theta \sin\phi}{\sin\theta} \quad (176)$$
$$y = \frac{-\cos\theta \cos\phi}{\sin\theta} \quad (177)$$
$$\sin\theta = \zeta/\sqrt{1+\zeta^2} \quad (178)$$
$$\cos\theta = 1/\sqrt{1+\zeta^2} \quad (179)$$
$$(180)$$

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

3.6.1.2 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:

$$x = \cos\theta \sin\phi \quad (181)$$
$$y = -\cos\theta \cos\phi \quad (182)$$
$$\sin\theta = \sqrt{1 - R_\theta^2} \quad (183)$$
$$\cos\theta = R_\theta \quad (184)$$
$$(185)$$

3.6.2 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.

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

$$x = \phi \tag{186}$$
$$y = \theta \tag{187}$$

3.6.2.2 Mercator The Mercator projection ("MER") is a cylindrical projection.

$$x = \phi \tag{188}$$
$$y = \ln\left(\tan(\pi/4 + \theta/2)\right) \tag{189}$$
$$\text{and} \qquad \theta = 2\arctan\left(e^y\right) - \pi/2 \tag{190}$$

3.6.2.3 Hammer-Aitoff The Hammer-Aitoff projection("AIT") is a pseudocylindrical projection, and is defined:

$$x = 2\zeta\cos\theta\sin\frac{\phi}{2} \tag{191}$$
$$y = \zeta\sin\theta \tag{192}$$
$$\text{where} \qquad \zeta^{-1} \equiv \sqrt{\frac{1}{2}\left(1 + \cos\theta\cos\frac{\phi}{2}\right)} \tag{193}$$

And in reverse:

$$\phi = 2\text{arctan}(2z^2 - 1, xz) \tag{194}$$
$$\theta = \text{arcsin}(yz) \tag{195}$$
$$\text{where} \qquad z \equiv \sqrt{1 - (x/2)^2 - y^2} \tag{196}$$

3.6.2.4 Parabolic The Parabolic projection ("PAR") is a pseudocylindrical projection, and is defined:

$$x = \phi\left(2\cos\frac{2\theta}{3} - 1\right) \quad (197)$$

$$y = \pi\sin\frac{\theta}{3} \quad (198)$$

$$(199)$$

And in reverse:

$$\theta = 3\sin^{-1}\rho \quad (200)$$

$$\phi = \frac{x}{1-4\rho^2} \quad (201)$$

$$\text{where} \quad \rho \equiv y/\pi \quad (202)$$

$$(203)$$

3.7 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 the 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);
}
```

3.8 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.

3.9 General Astronomy Functions

we will provide a new airmass function, and a new mean-to-apparent conversion (TBD)

3.10 Positions of Major Solar System Objects

ephemerides code? (TBD)

4 Pan-STARRS Modules

4.1 Object Models

In the discussions below, the following relationships between the given C variables and the `pmSource` entries should be used:

- `Xo` is `pmMomemt.x`
- `Yo` is `pmMomemt.y`
- `SigmaX` is `pmMomemt.Sx`
- `SigmaY` is `pmMomemt.Sy`
- `Zo` is `pmPeak.counts - pmMoment.Sky`
- `So` is `pmMomemt.Sky`

4.1.1 Real 2D Gaussian

This function is a two-dimensional Gaussian with an elliptical cross-section and a constant local background:

$$f(x,y) = Z_o e^{-z} + S_o$$

where

$$z = \frac{(x-x_o)^2}{2\sigma_x^2} + \frac{(y-y_o)^2}{2\sigma_y^2} + (x-x_o)(y-y_o)\sigma_{xy}$$

Below is the relationship between the `psModel` parameters and the function parameters, sample C-code implementing the function efficiently, and the value of the derivatives.

```
param[0] = So;
param[1] = Zo;
param[2] = Xo;
param[3] = Yo;
param[4] = sqrt(2) / SigmaX;
param[5] = sqrt(2) / SigmaY;
param[6] = Sxy;

X = x[0] - param[2];
Y = x[1] - param[3];

px = param[4]*X;
py = param[5]*Y;

z = 0.5*SQ(px) + 0.5*SQ(py) + param[6]*X*Y;
r = exp(-z);
```

```
f = param[1]*r + param[0];
/* f is the function value */

q = param[1]*r;
deriv[0] = +1;
deriv[1] = +r;
deriv[2] = q*(2*px*param[4] + param[6]*Y);
deriv[3] = q*(2*py*param[5] + param[6]*X);
deriv[4] = -2*q*px*X;
deriv[5] = -2*q*py*Y;
deriv[6] = -q*X*Y;
```

The intial guess for the Gaussian parameters may be taken from the moments, peak value, and local sky.

4.1.2 Pseudo-Gaussian

This function is a polynomial approximation of a 2D Gaussian. The function is very similar to the real Gaussian:

$$f(x,y) = Z_o(1 + z + z^2/2 + z^3/6)^{-1} + S_o$$

where

$$z = \frac{(x - x_o)^2}{2\sigma_x^2} + \frac{(y - y_o)^2}{2\sigma_y^2} + (x - x_o)(y - y_o)\sigma_{xy}$$

Below is the relationship between the `psModel` parameters and the function parameters, sample C-code implementing the function efficiently, and the value of the derivatives:

```
param[0] = So;
param[1] = Zo;
param[2] = Xo;
param[3] = Yo;
param[4] = sqrt(2) / SigmaX;
param[5] = sqrt(2) / SigmaY;
param[6] = Sxy;

X = x[0] - param[2];
Y = x[1] - param[3];

px = param[4]*X;
py = param[5]*Y;

z = 0.5*SQ(px) + 0.5*SQ(py) + param[6]*X*Y;
t = 1 + z + 0.5*z*z;
r = 1.0 / (t*(1 + z/3)); /* ~ exp (-Z) */
f = param[1]*r + param[0];
/* f is the function value */
```

```
/* note difference from a pure gaussian: q = param[1]*r */
q = param[1]*r*r*t;
deriv[0] = +1;
deriv[1] = +r;
deriv[2] = q*(2*px*param[4] + param[6]*Y);
deriv[3] = q*(2*py*param[5] + param[6]*X);
deriv[4] = -2*q*px*X;
deriv[5] = -2*q*py*Y;
deriv[6] = -q*X*Y;
```

The intial guess for the Gaussian parameters may be taken from the moments, peak value, and local sky.

4.1.3 Waussian

The Waussian is a modified polynomial approximation of a 2D Gaussian, with non-linear polynomial terms having variable coefficients, rather than the Taylor series values of 1/2 and 1/6. The function is very similar to the pseudo-Gaussian:

$$f(x,y) = Z_o(1 + z + B_2(z^2/2 + B_3 z^3/6))^{-1} + S_o$$

where

$$z = \frac{(x-x_o)^2}{2\sigma_x^2} + \frac{(y-y_o)^2}{2\sigma_y^2} + (x-x_o)(y-y_o)\sigma_{xy}$$

Below is the relationship between the `psModel` parameters and the function parameters, sample C-code implementing the function efficiently, and the value of the derivatives. Note the fudge factors of 100 in the derivatives of B_2 and B_3: these are included to slow the variation of these parameters, which are otherwise very sensitive to small errors.

```
param[0] = So;
param[1] = Zo;
param[2] = Xo;
param[3] = Yo;
param[4] = Sx;
param[5] = Sy;
param[6] = Sxy;
param[7] = B2;
param[8] = B3;

X = x - param[2];
Y = y - param[2];

px = param[4]*X;
py = param[5]*Y;

z = 0.5*SQ(px) + 0.5*SQ(py) + param[6]*X*Y;
t = 0.5*z*z*(1 + param[8]*z/3);
r = 1.0 / (1 + z + param[7]*t); /* ~ exp (-Z) */
f = param[1]*r + param[0];
```

```
/* note difference from gaussian: q = param[1]*r */
q = param[1]*r*r*(1 + param[7]*z*(1 + param[8]*z/2));
deriv[0] = +1;
deriv[1] = +r;
deriv[2] = q*(2*px*param[4] + param[6]*Y);
deriv[3] = q*(2*py*param[5] + param[6]*X);
deriv[4] = -2*q*px*X;
deriv[5] = -2*q*py*Y;
deriv[6] = -q*X*Y;
deriv[7] = -100*param[1]*r*r*t;
deriv[8] = -100*param[1]*r*r*param[7]*(z*z*z)/6;
/* the values of 100 dampen the swing of param[7,8] */
```

4.1.4 Twisted Gaussian

This function describes an object with power-law wings and a flattened core, where the core has a different contour from the wings.

$$f(x,y) = Z_{\rm pk}(1 + z_1 + z_2^M)^{-1} + Sky$$

where

$$z_1 = \frac{x^2}{2\sigma_{x,in}^2} + \frac{y^2}{2\sigma_{y,in}^2} + xy\sigma_{xy,in} z_2 = \frac{x^2}{2\sigma_{x,out}^2} + \frac{y^2}{2\sigma_{y,out}^2} + xy\sigma_{xy,out}$$

```
param[0]  = So;
param[1]  = Zo;
param[2]  = Xo;
param[3]  = Yo;
param[4]  = SxInner;
param[5]  = SyInner;
param[6]  = SxyInner;
param[7]  = SxOuter;
param[8]  = SyOuter;
param[9]  = SxyOuter;
param[10] = N;

X = x - param[2];
Y = y - param[3];

px1 = param[4]*X;
py1 = param[5]*Y;
px2 = param[7]*X;
py2 = param[8]*Y;

z1 = 0.5*SQ(px1) + 0.5*SQ(py1) + param[4]*X*Y;
z2 = 0.5*SQ(px2) + 0.5*SQ(py2) + param[9]*X*Y;
```

```
  r  = 1.0 / (1 + z1 + pow(z2,param[10]));
  f  = param[5]*r + param[6];

  q1 = param[5]*SQ(r);
  q2 = param[5]*SQ(r)*param[10]*pow(z2,(param[10]-1));

  deriv[0] = +1;
  deriv[1] = +r;
  deriv[2] = q1*(2*px1*param[4] + param[6]*Y) + q2*(2*px2*param[7] + param[9]*Y);
  deriv[3] = q1*(2*py1*param[5] + param[6]*X) + q2*(2*py2*param[8] + param[9]*X);

  /* these fudge factors impede the growth of param[4] beyond param[7] */
  f1 = fabs(param[7]) / fabs(param[4]);
  f2 = (f1 < FSCALE) ? 1 : FFACTOR*(f1 - FSCALE) + 1;
  deriv[4] = -2*q1*px1*X*f2;

  /* these fudge factors impede the growth of param[5] beyond param[8] */
  f1 = fabs(param[8]) / fabs(param[5]);
  f2 = (f1 < FSCALE) ? 1 : FFACTOR*(f1 - FSCALE) + 1;
  deriv[5] = -2*q1*py1*Y*f2;

  deriv[6] = -q1*X*Y;

  deriv[7] = -2*q2*px2*X;
  deriv[8] = -2*q2*py2*Y;
  deriv[9] = -q2*X*Y;
  deriv[10] = -q1*ln(z2);
```

The intial guess for the Gaussian parameters may be taken from the moments, peak value, and local sky.

future galaxy models to be implemented (TBD)

```
float Sersic()
  param[0] = So;
  param[1] = Zo;
  param[2] = Xo;
  param[3] = Yo;
  param[4] = Sx;
  param[5] = Sy;
  param[6] = Sxy;
  param[7] = Nexp;

float SersicBulge()
  param[0]  So;
  param[1]  Zo;
  param[2]  Xo;
  param[3]  Yo;
```

```
param[4]  SxInner;
param[5]  SyInner;
param[6]  SxyInner;
param[7]  Zd;
param[8]  SxOuter;
param[9]  SyOuter;
param[10] = SxyOuter;
param[11] = Nexp;
```

4.2 WCS Translation

The FITS World Coordinate System (WCS) standard is specified in two papers: Greisen & Calabretta, 2002, A&A, 375, 1061 (Paper I); and Calabretta & Greisen, 2002, A&A, 375, 1077 (Paper II). Two further papers (Papers III and IV) are available as drafts, and have not yet been accepted. All these papers, in their most up-to-date form, are available from Mark Calabretta's page.

Papers I and II together lay out a system for converting pixel coordinates to celestial coordinates (RA, Dec). However, these assume that linear transformations, followed by projection or deprojection are sufficient, whereas we do not expect that this is adequate to describe the Pan-STARRS focal plane. Paper III has to do with spectral coordinates, and does not concern us. Paper IV has a proposed mechanism for dealing with distortions which appears to be adequate to our needs. While the formalism has not been officially approved, and the syntax may change, the current version of the paper provides a means for translating the multilayered Pan-STARRS system to FITS. Consequently, we will use the current version (the version we consider here is dated 22 April 2004) and update any syntax changes later as required.

Paper IV proposes two separate distortion corrections — before and after a linear transformation. Given pixel coordinates, a distortion correction is made yielding "corrected pixel coordinates". This first distortion allows correction of the individual detector (for example, if the detector is not flat, as may be the case for MegaCam). A linear transformation is then performed to "intermediate pixel coordinates", accounting for translation, rotation and scale. The second distortion correction to "corrected intermediate pixel coordinates" is performed, which corrects for optical distortion. The resultant is then scaled and deprojected onto the sky, yielding the celestial coordinates.

4.2.1 Implementation

The first distortion will correct for tilts or bends of the detectors so that pixels are in the same plane as the focal plane (`psCell.toChip`). The linear transformation will correct for the position on the focal plane (`psChip.toFPA`). The second distortion will correct for distortions in the optics (`psFPA.toTangentPlane`). The projection will be a simple gnomonic ("TAN") projection. Thus, the Paper IV formalism satisfies our needs.

Paper IV also goes far beyond our needs, by providing several types of distortion functions. We are interested (at least, for now) solely in simple polynomial distortion functions, as this is what is currently implemented for Pan-STARRS (i.e., we are not interested in cubic spline, B-spline or lookup tables; nor are we interested in the use of auxiliary variables, though this may change in the future). In particular, the proposed WCS system cannot handle the color and magnitude dependence currently built into psLib's `psDistortion`, so we will need to specify a mean color and magnitude at which to evaluate the spatial polynomials.

In the event that the multiple layers (`psCell.toChip` → `psChip.toFPA` → `psFPA.toTangentPlane`) are not available, the short-cut transformation (`psCell.toFPA`) can be used as the first WCS distortion. If the only available information is the "quick and dirty" transformation (`psCell.toSky`), this may be implemented using the linear

transformation without any of the WCS distortions, followed by a linear 'projection' ("LIN").

Reading the WCS into a psFPA can be done in the reverse order of writing the WCS.

5 Missing and Todo

define sunrise, sunset, sun position (TBD)

define moonrise, moonset, moon position, moon phase (TBD)

define planet functions (TBD)

clean up FITS I/O issues (TBD)

define Brent's method & minimization bracketing (TBD)

A Change Log

A.1 Changes from version 06 (7 September 2004) to version 07 (24 November 2004)

- 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.
- Added discussion about using errors in calculating statistics.
- Fixed up LMM to be specific to χ^2 fitting.
- Generalised entry on Gaussian smoothing.
- modifications to the document name and PSDC number (and refs).

A.2 Changes from version 07 (24 November 2004) to version 08 (8 January 2005)

- Added short section on histograms in the presence of errors.
- Added short note on inverse spherical transformations.
- Added section on astronomical object models

A.3 Changes from version 08 (8 January 2005) to version 09 (14 February 2005)

- Added section on inverse and combined transformations.
- Added `PS_RESAMPLE_LANCZOS[234]`, dropped `PS_RESAMPLE_LAGRANGE`.
- Added section on FITS WCS.

A.4 Changes from version 09 (14 February 2005) to version 10 (19 April 2005)

- Changes to the Time section:
  - Renamed TDT to TT
  - Misc. cleanups and clarifications
  - Added references
  - Updated definitions of UTC, UT1, JD, and MJD
  - Verbatim Perl code for YMD→sec and sec→YMD conversion
  - Verbatim Fortran code for UT1 interpolation
- section reorganization:
  - renamed Astronomical Image Manipulations to Image Manipulations
  - moved Image Manipulations to own subsection before Astronomy Utilities
  - promoted all PSLib subsections to sections
- defined earth orientation algorithms
- provided quaternions for rotations
- moved some sections to reflect order in SDRS (matrix, fftw)

A.5 Changes from version 10 (19 April 2005) to version 11 (27 April 2005)

- fixed some typos in the definition of the rotation from CEO to GCRS (Eqn 132).
- added references to the SDRS APIs for the Earth Orientation section

A.6 Changes from version 11 (27 April 2005) to version 12 (11 July 2005)

- Removed all references to slalib.
- Updated section on rotations.
- Divided the robust statistics into robust and fitted statistics, modified the algorithm somewhat.
- Clarifies the LM minimization coding, added gain-factor test description