LARGE SYNOPTIC SURVEY TELESCOPE

Large Synoptic Survey Telescope (LSST)
Data Management

jointcal: Simultaneous Astrometry & Photometry for thousands of Exposures with Large CCD Mosaics

John Parejko (University of Washington), Pierre Astier (LPNHE/IN2P3/CNRS Paris)

DMTN-036

Latest Revision: 2017-09-18

DRAFT

Abstract

The jointcal package simultaneously optimizes the astrometric and photometric calibrations of a set of astronomical images. In principle and often in practice, this approach produces distortion and thoroughput models which are more precise than when fitted independently. This is especially true when the images are deeper than the astrometric reference catalogs. In the "Astromatic" software suite, this simultaneous astrometry functionality is fulfilled by "SCAMP". The code we describe here has similar aims, but follows a slightly different route. Jointcal is built on top of the the LSST Data Management software stack.

Change Record

Version	Date	Description	Owner name
1	2017-09-18	Initial release. Moved from jointcal repo.	John Parejko

Contents

jointcal: Simultaneous Astrometry & Photometry for thousands of Exposures with Large CCD Mosaics

1 Introduction

With deep astronomical images, it is extremely common that the relative astrometry and photometry between images is considerably more precise than the accuracy of external catalogs, where "more precise" can be as large as two orders of magnitude. For applications where the quality of relative astrometry is important or vital, it is important to rely on some sort of simultaneous astrometry solution, if possible optimal in a statistical sense.

This package performs a least-squares fit to a set of images. Since it aims at statistical optimality, we maximize the likelihood of the measurements with respect to all unknown parameters required to describe the data. These parameters are mostly in two sets: the position on the sky of the objects in common between the images, and the mapping of each image to the sky. To these obvious parameters, one can add proper motions (where applicable), and parameters describing the differential effect of atmospheric refraction on the position of objects. It is clear that one cannot fit simultaneously the position on the sky and the mappings from CCD coordinates to the sky, without extra constraints: the "sky" coordinate system is then undefined, and one needs reference positions in order to fully define this frame. We use the GAIA catalog (Gaia Collaboration et al., 2016) as our reference catalog, and may supplement it with deeper data (e.g. PS1) where available.

SCAMP (Bertin, 2006) is the reference package for simultaneous astrometry in astronomy, at least for relative alignment of wide-field images prior to stacking. Regarding optimization, SCAMP follows a somewhat different route from ours: it does not optimize over the position of common objects but rather minimizes the distance between pairs of transformed measurements of the same object. This approach is not a maximum likelihood optimization, and is likely statistically sub-optimal. The main drawback of SCAMP in the context of LSST is the fact that it is a program and not a library, and hence not flexible regarding formats of images and catalogs. But since SCAMP has been used for almost a decade in production by various teams, the quality checking tools it provides should likely be reproduced in the context of our package. We provide residual ntuples and hope that the first serious users will contribute plotting tools.

Loading input catalogs and their metadata from disk is a large fraction of the total time. We can save time by re-using the input catalog to fit a similar style of multi-component model for the relative and absolute photometry.

The plan of this note is as follows: we first sketch the algorithm (Section 3). We provide our least-squares formulation in Section 4, and describe the how we evaluate the derivatives with respect to the parameters. We then describe how we associate the measurements of each object in different exposures in Section 5.

2 Past work

A summary of past work on this topic will go here eventually...

- Photographic Plates (Eichhorn, 1960)
- SDSS übercal (Padmanabhan et al., 2008)
- Pan-Starrs übercal (Magnier et al., 2013)
- DECam WcsFit (Bernstein et al., 2017)

3 Algorithm flow

The algorithm assumes that the initial single-frame WCS fits of the input images are accurate to $\sim 1''$. Currently, the code properly interprets the SIP WCS's (relying on `lsst::afw::wcs`), with or without distortions. The code might handle transparently the "PV" encoding of distortions (used in SCAMP and Swarp), but lacks the IO's required to use this format. Note that in both instances, the WCS boils down to a polynomial 2D transform from CCD space to a tangent plane, followed by a gnomonic de-projection to the celestial sphere. The difference between formats lies into the encoding of the polynomial, but they map exactly the same space of distortion functions.

The algorithm can be roughly split into these successive steps:

1. load the input catalogs and 'rough' WCS's and catalogs, and select the sources to use in

the fit (e.g. good centroids, unblended, high enough signal-to-noise) via the configured SourceSelectorTask.

2. Associate these catalogs, i.e. associate each detection of the same on-sky source, and associate those on-sky sources with a reference catalog (if provided).
3. Iteratively fit the model parameters and "true" on-sky values, clipping outliers at each iteration.
4. Output results.

4 Mathematical formalism

4.1 Definitions

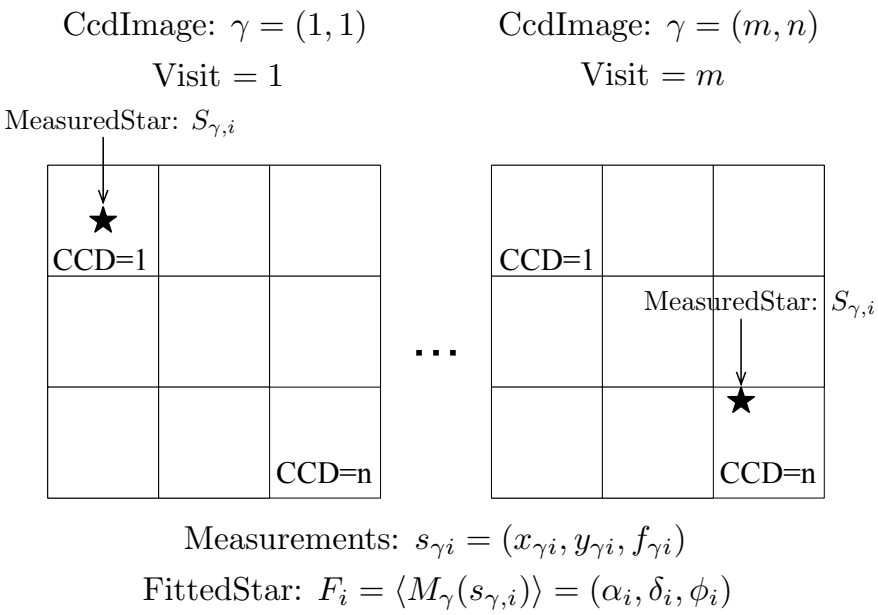

FIGURE 1: The relationship between MeasuredStars, CcdImages, and FittedStars.

We use the following notation in the mathematics that follows, with *CamelCased* words referring to the objects in the code. These terms are diagramed in Figure 1 on page 3.

- γ is the *CcdImage* representing one exposure (*visit* in LSST terminology) on one CCD. The

CcdImage contains metadata about that visit and CCD detector, as well as the catalog of sources that were selected for use in the fit;

- $S_{\gamma,i}$ is the position (pixels) and flux (instrument-flux counts) $(x_{\gamma i}, y_{\gamma i}, f_{\gamma i})$ of the *MeasuredStar* on *CcdImage* γ corresponding to the on-sky *FittedStar* i;
- F_i is the (sky) position and calibrated-flux $(\alpha_i, \beta_i, \phi_i)$ of the star i, with some corresponding number of measurements represented by *MeasuredStar*s;
- M_γ is the mapping for *CcdImage* γ from pixels/instrument-flux (x, y, f) to the tangent plane on the sky/calibrated-flux (α, β, ϕ). This mapping may consist of models constrained across visits (different between CCDs; e.g. an affine model for each CCD position, assuming CCDs do not move), constrained across CCDs and visits (e.g. a 2d radial polynomial of the optics) or constrained across CCDs (different between visits; e.g. a 2d polynomial of the sky);
- R_j refers to the (sky) position and (reference) calibrated-flux $(\alpha_j, \beta_j, \phi_j)$ of *RefStar* j.

In addition to the terms defined in the diagram above, we define the following terms for the measurement component,

- P_γ is a projector from sidereal coordinate to some tangent plane; P_γ is user-defined;
- $W_{\gamma,i}$ is the measurement weight of $M_\gamma(S_{\gamma,i})$, i.e. the inverse of the 2×2 covariance matrix (for astrometry), or the inverse of the transformed instrument-flux error;

and reference component,

- P is some (user-provided) sky to tangent plane projector;
- W_j is the weight matrix of the reference star, i.e. the inverse of the projected position error $P(R_j)$, or the inverse of the reference flux error.

4.2 Least-squares expression

The fit consists of minimizing, for photometry,

$$\chi^2 = \sum_{\gamma,i} [M_\gamma(S_{\gamma,i}) - F_i]^T W_{\gamma,i} [M_\gamma(S_{\gamma,i}) - F_i] \quad \text{(meas. terms)}$$
$$+ \sum_j [F_j - R_j]^T W_j [F_j - R_j] \quad \text{(ref. terms)} \quad (1)$$

and for astrometry, taking into account the projection from the sky to the tangent plane P:

$$\chi^2 = \sum_{\gamma,i} [M_\gamma(S_{\gamma,i}) - P_\gamma(F_i)]^T W_{\gamma,i} [M_\gamma(S_{\gamma,i}) - P_\gamma(F_i)] \quad \text{(meas. terms)}$$
$$+ \sum_j [P(F_j) - P(R_j)]^T W_j [P(F_j) - P(R_j)] \quad \text{(ref. terms)} \quad (2)$$

where the first line iterates on all *MeasuredStar* γ, i, from all *CcdImage* γ, and the second iterates on all *RefStar* j. In the first terms, the object at position F_i is the one that was measured at position $S_{\gamma,i}$ in image γ. The association between *MeasuredStar*s, *FittedStar*s, and *RefStar*s is described in Section 5.

The measurement terms compare the measurement positions/fluxes to objects positions/fluxes (the relative astrometry/photometry), the reference terms compare object positions/fluxes to reference positions/fluxes (the absolute astrometry/photometry). We need these two sets of terms because not all objects F_i in the first terms appear in the second terms: many objects in the images will not be in the reference catalogs, but those objects do help to constrain the mappings M_γ. In addition, with so many measurements of our sources, we may have better overall errors on the positions and fluxes than the reference catalog can provide.

The expressions above depend on two sets of parameters: the parameters defining the mappings M and the on-sky positions/calibrated fluxes F_i. For a practical problem, this amounts to a very large number of parameters, which becomes tractable when one notes that every term in the $\chi 2$ is only linked to a small number of parameters. We exploit this feature to rapidly compute the gradient and the Hessian of the χ^2 in order to find the minimum.

So far, we have not specified how we model the mappings M nor how we choose the various projectors that appear in expression (2). The code has been written to allow the user to provide their own versions of both the model for mappings and the projection scheme.

We however provide some implementations for both aspects that we discuss in the next two sections.

4.3 Minimization approach

The expressions for astrometry (eq. (2)) and photometry (eq. (1)) depend on two sets of parameters: the parameters η defining the mappings $M_\gamma(S_{\gamma,i}) \equiv M_\gamma(\eta_\gamma, S_{\gamma,i})$, and the "true" calibrated fluxes F_i. We write the measurement and reference residuals, with their respective weights $W_{\gamma i}$ and W_j, as separate functions of their parameters, for photometry,

$$D_{\gamma i} = M_\gamma(S_{\gamma,i}) - F_i \tag{3}$$
$$D_j = F_j - R_j \tag{4}$$

and for astrometry,

$$D_{\gamma i} = M_\gamma(S_{\gamma,i}) - P_\gamma(F_i) \tag{5}$$
$$D_j = [P(F_j) - P(R_j)] \tag{6}$$

This results in the generalized χ^2 expression (compare to eq. (2) and (1)),

$$\chi^2 = \sum_{\gamma,i} {D_{\gamma i}}^T W_{\gamma,i} D_{\gamma i} + \sum_j {D_j}^T W_j D_j \tag{7}$$

To minimize this χ^2, we want to find the point in parameter space where the gradient $\nabla\chi^2 = d\chi^2/d\theta = 0$, where θ denotes the vector of parameters (of size N_p– see below). Applying the

product rule and noting the symmetry of D and D^T, we have

$$\frac{1}{2}\frac{d\chi^2}{d\theta} = \sum_{\gamma,i} {D_{\gamma i}}^T W_{\gamma,i} \nabla D_{\gamma i} + \sum_j {D_j}^T W_j \nabla D_j \tag{8}$$

where the ∇D matrices have size $2 \times N_p$ (for astrometry) and $1 \times N_p$ (for photometry):

$$\nabla D_{\gamma i} = \frac{d D_{\gamma i}}{d\theta} \tag{9}$$

$$\nabla D_j = \frac{d D_j}{d\theta} \tag{10}$$

We call the vector that gathers all the parameters to be fit during the minimization $\theta = (\eta_\gamma, F_i)$. This vector can easily exceed $N_p > 10^5$ entries. As an example with LSST (having a 189-CCD camera), a mapping consisting of a 3rd order 2-D polynomial per visit (16 η_k parameters per visit) and 100 viable sources per CCD with 15 visits (roughly a year of LSST observations of one sky location in one filter), θ will have $N_p = 283,740$ entries ($283,500$ F_i parameters + 240 model parameters). However, the second derivative matrix, $d^2\chi^2/d\theta^2$, is very sparse, because there are no terms connecting F_i and F_j if $i \neq j$, and depending on how the mappings are parametrized, a set of η_γ parameters could be connected (in the second derivative matrix) to only a small set of F_j's. So, we can search for the minimum χ^2 using methods involving the second derivative matrix, taking advantage of its sparseness.

We are looking for the offset to the starting parameters that zeros the gradient, so we can Taylor expand about that starting parameter vector θ_0,

$$0 = \frac{d\chi^2}{d\theta}(\theta_0 + \delta\theta) = \frac{d\chi^2}{d\theta}(\theta_0) + \frac{d^2\chi^2}{d\theta^2}(\theta_0)\delta\theta + O(\delta\theta^2)$$

and solve for the offset $\delta\theta$ that zeroes it to first order,

$$\left[\frac{d^2\chi^2}{d\theta^2}(\theta_0)\right]\delta\theta = -\frac{d\chi^2}{d\theta}(\theta_0) \tag{11}$$

$$H\delta\theta = -\nabla\chi^2 \tag{12}$$

where we have identified the second derivative matrix with the Hessian matrix H. Working

from eq. (8), we can write the second derivative matrix (the Hessian) as:

$$\frac{1}{2}\frac{d^2\chi^2}{d\theta^2} = \sum_{\gamma,i} \nabla D_{\gamma i}{}^T W_{\gamma,i} \nabla D_{\gamma i} + \sum_j \nabla D_j{}^T W_j \nabla D_j \tag{13}$$

where we have neglected the second derivatives of the residual vectors D. This second derivative matrix is by construction symmetric, and hence the parameter offsets (defined in eq. (11)) can be evaluated using the (fast) Cholesky LDL^T factorization (see 4.10). If possible, mappings $M_\gamma(\eta_\gamma, S)$ linear with respect to their parameters η_γ, for example polynomials, are to be favored because the second derivatives will no longer depend on that parameter. If the problem were non-linear (more precisely, if the second derivative varies rapidly) we would have to implement a line search to minimize $\chi^2(\theta_0 + \lambda \times \delta\theta)$ over λ.

Returning to the weights, we note that the matrices $W_{\gamma,i}$ are positive-definite and thus they have square roots (e.g. the Cholesky square root) and can be written as: $W_{\gamma,i} = \alpha_{\gamma i}^T \alpha_{\gamma i}$. Defining $K_{\gamma i} = \alpha_{\gamma i} D_{\gamma i}$, the Hessian expression becomes:

$$\frac{1}{2}\frac{d^2\chi^2}{d\theta^2} = \sum_{\gamma,i} K_{\gamma i}{}^T K_{\gamma i} + \sum_j K_j{}^T K_j \tag{14}$$

The sums present in this expression can be performed using matrix algebra; we concatenate all the K matrices into a single large, sparse matrix (the Jacobian),

$$J \equiv \left[\{K_{\gamma i}, \forall\gamma, i\}, \{K_j, \forall j\}\right] \tag{15}$$

and thus we simply have

$$\frac{1}{2}\frac{d^2\chi^2}{d\theta^2} = J^T J \tag{16}$$

In the code, we take advantage of the fact that each term of the χ^2 only depends on a small number of parameters. The *Model::getMappingIndices* method allows us to rapidly collect the indices of these parameters, and we evaluate the D matrices at these indices only, as all other indices are zero.

The computation of the Jacobian and the gradient is performed in the respective *PhotometryFit* and *AstrometryFit* classes. The methods *leastSquareDerivativesMeasurement* and *leastSquareDerivativesReference* compute the contributions to the Jacobian and gradient of the χ^2 from the

measurement terms and the references terms respectively. In these routines, the Jacobian is represented as a list of *Eigen::Triplets* (i, j, J_{ij}) describing its elements. This list is then transformed into a representation of sparse matrices suitable for algebra, and in particular suitable to evaluate the product $J^T J$. Once we have evaluated $H \equiv J^T J$, we can solve eq. (12) using a Cholesky factorization. For sparse linear algebra, the Cholmod and Eigen packages provide the required functionality. It turns out that for the mappings we have currently employed, the calculation of $J^T J$ and the factorization are the most CPU intensive parts of the calculations, and there is hence not much to be gained in speeding up the calculation of derivatives. For the factorization, we have tried both Eigen and Cholmod (via the Eigen interface) and their speeds differ by less than 10%.

4.4 Photometry example

As an illustrative example, we will work through a particular photometry mapping taking on-chip fluxes and positions $S_{\gamma i} = (f_{\gamma i}, x, y)$ to on-sky fluxes $\phi_{\gamma i}$, consisting of a constant zero-point per CCD (f_0: the CCD's filter response) and an $(n+m)$th order 2-D Chebyshev polynomial ($\sum a_{j,k} T_j(u) T_k(v)$: the optics+sky response, where $(u(x,y), v(x,y))$ are the focal plane coordinates of pixel (x, y) on a given CCD) per visit. Thus, the mapping will be

$$\begin{aligned} M_\gamma(\eta, S_{\gamma i}) &= M_{CCD}(f_0^{-1}, f_{\gamma i}) M_{visit}(a_{j,k}, x_{\gamma i}, y_{\gamma i}) \\ &= f_{\gamma i} [f_0]^{-1} \sum_{j=0}^{j=n} \sum_{k=0}^{k=m} a_{j,k} T_j(u_{\gamma i}) T_k(v_{\gamma i}) \\ &= \phi_{\gamma i} \end{aligned} \tag{17}$$

where we will fit f_0^{-1} instead of f_0 in order to simplify the derivatives (with respect to $\eta = \left(f_0^{-1}, a_{j,k} \forall j, k\right)$). Computing those derivatives gives us:

$$\nabla D_{\gamma i} = \left(\frac{\partial D_{\gamma i}}{\partial f_0^{-1}}, \frac{\partial D_{\gamma i}}{\partial a_{0,0}}, \ldots, \frac{\partial D_{\gamma i}}{\partial a_{n,m}}, \frac{\partial D_{\gamma i}}{\partial F_i}\right)$$

where, for the measurement terms we have (recall eq. (3)),

$$\begin{aligned}\frac{\partial D_{\gamma i}}{\partial f_0^{-1}} &= f_{\gamma i} M_{visit}(a_{j,k} x_{\gamma i}, y_{\gamma i}) \\ \frac{\partial D_{\gamma i}}{\partial a_{j,k}} &= f_{\gamma i} f_0^{-1} T_{jk}(u_{\gamma i}, v_{\gamma i}) \\ \frac{\partial D_{\gamma i}}{\partial F_i} &= -1\end{aligned} \tag{18}$$

and for the reference terms (recall eq. (4)),

$$\nabla D_j = \frac{\partial D_j}{\partial F_j} = 1 \tag{19}$$

This model is degenerate to multiplying by a scale factor: $M_{CCD} \to a M_{CCD}, M_{visit} \to a^{-1} M_{visit}$. This degeneracy is not removed by the reference catalog. To break this degeneracy, we hold fixed one CCD's f_0^{-1} (chosen to be the CCD closest to the center of the focal plane), and fit all other CCD's relative to that.

4.5 Magnitude-based photometry example

As an illustrative example, we will work through a particular photometry mapping taking on-chip fluxes and positions $S_{\gamma i} = (f_{\gamma i}, x, y)$ to on-sky magnitudes $m_{\gamma i}$, with the transform consisting of a constant zero-point per CCD (f_0: the CCD's filter response) and an $(n+m)$th order 2-D Chebyshev polynomial ($\sum a_{j,k} T_j(u) T_k(v)$: the optics+sky response, where $(u(x,y), v(x,y))$ are the focal plane coordinates of pixel (x,y) on a given CCD) per visit. Thus, taking m_{CCD}, m_{visit}, as the respective magnitude components, the mapping is

$$M_\gamma(\eta, S_{\gamma i}) = m_{\gamma i} + m_{CCD} + m_{visit} \tag{20}$$

$$= -2.5 \log_{10}(M_{CCD}(f_0^{-1}, f_{\gamma i})) - 2.5 \log_{10}(M_{visit}(a_{j,k}, x_{\gamma i}, y_{\gamma i})) \tag{21}$$

$$= m(f_{\gamma i}) + m(f_0^{-1}) + \sum a_{j,k} T_j(u) T_k(v)$$

$$= m_{\gamma i}$$

Making the visit component an additive polynomial of position makes the derivatives simpler, while adding complexity to the resulting *PhotoCalib* (it becomes an exponential term $10^{M_{visit}(x,y)/-2.5}$). Computing the derivatives with respect to $\eta = \left(f_0, a_{j,k} \forall j, k\right)$ gives us,

$$\nabla D_{\gamma i} = (\frac{\partial D_{\gamma i}}{\partial f_0^{-1}}, \frac{\partial D_{\gamma i}}{\partial a_{0,0}}, \dots, \frac{\partial D_{\gamma i}}{\partial a_{n,m}}, \frac{\partial D_{\gamma i}}{\partial F_i})$$

where, for the measurement terms we have (recall eq. (3)),

$$\begin{aligned} \frac{\partial D_{\gamma i}}{\partial f_0^{-1}} &= 1 \\ \frac{\partial D_{\gamma i}}{\partial a_{j,k}} &= T_{jk}(u_{\gamma i}, v_{\gamma i}) \\ \frac{\partial D_{\gamma i}}{\partial F_i} &= -1 \end{aligned} \tag{22}$$

and for the reference terms (recall eq. (4)),

$$\nabla D_j = \frac{\partial D_j}{\partial F_j} = 1 \tag{23}$$

This model is degenerate to multiplying by a scale factor: $M_{CCD} \to aM_{CCD}, M_{visit} \to -aM_{visit}$. This degeneracy is not removed by the reference catalog. To break this degeneracy, we hold fixed one CCD's f_0^{-1} (chosen to be the CCD closest to the center of the focal plane), and fit all other CCD's relative to that.

4.6 The astrometric distortion model

The routines in the *AstrometryFit* class do not really evaluate the derivatives of the mappings, but rather defer those to other classes. The main reason for this separation is that one could conceive different ways to model the mappings from pixel coordinates to the tangent plane, and the actual model should be abstract in the routines accumulating gradient and Jacobian. The class *AstrometryModel* is an abstract class aiming at connecting generically the fitting routines to actual models. We have so far coded two of these models:

- *SimplePolyModel* implements one polynomial mapping per input *CcdImage* (i.e. Calexp).

- *ConstrainedPolyModel* implements a model where the mapping for each CcdImage is a composition of a polynomial for each CCD and a polynomial for each exposure. For one of the exposures, the mapping should be fixed or the model is degenerate.

For example, if one fits 10 exposures from a 36-CCD camera, there will be 10×36 polynomials to fit with the first model, and $10 + 36$ with the second model. The *ConstrainedPolyModel* assumes that the focal plane of the instrument does not change across the data set. We could consider coding a model made from one *ConstrainedPolyModel* per set of images for which the instrument can be considered as geometrically stable. This is similar to how Scamp models the distortions.

In both of these models, we have used standard polynomials in 2 dimensions rather than an orthogonal set (e.g. Legendre, Laguerre, ...) because regular polynomials are easy to compose (i.e. one can easily compute the coefficients of P(Q(X))), and they map exactly the same space as the common orthogonal sets. We have taken care to normalize the input coordinates (mapping the range of fitted data over the $[-1, 1]$ interval), in order to alleviate the well-know numerical issues associated to fitting of polynomials.

4.7 Choice of projectors

In the least squares expression (2), the residuals of the measurement terms read:

$$D_{\gamma i} = M_\gamma(S_{\gamma i}) - P_\gamma(F_i)$$

If the coordinates F_i are sidereal coordinates, the projector P_γ determine the meaning of the mapping M_γ. If one is aiming at producing WCS's for the image, it seems wise to choose for P_γ the projection used for the envisioned WCS, so that the mapping M_γ just describes the transformation from pixel space to the projection plane. For a SIP WCS, one will then naturally choose a gnomonic projector, so that M_γ can eventually be split into the "CD" matrix and the SIP-specific higher order distortion terms (see Section A for a brief introduction to WCS concepts).

So, the choice of the projectors involved in the fit are naturally left to the user. This is done via a virtual class *ProjectionHandler*, an instance of which has to be provided to the *AstrometryFit* constructor. There are obviously ready-to-use *ProjectionHandler* implementations which should suit essentially any need. For the standard astrometric fit aiming at setting WCS's, we

provide the *OneTPPerShoot* derived class, which implements a common projection point for all chips of the same exposure. It is fairly easy to implement derived classes with other policies.

The choice of the projector appearing in the reference terms of eq. (2) is not left to the user because we could not find a good reason to provide this flexibility, and we have implemented a gnomonic projection. We use a projector there so that the comparison of positions is done using an Euclidean metric.

4.8 Proper motions and atmospheric refraction

The expression (2) above depends on two sets of parameters: the parameters defining the mappings and the positions F_k. This expression hides two details implemented in the code: accounting for proper motions and differential effects of atmospheric refraction.

Proper motions can be accounted for to predict the expected positions of objects and even be considered as fit parameters. At the moment we neither have code to detect that some (presumably stellar) object is moving, nor code to ingest proper motions from some external catalog. Each *FittedStar* has a flag that says whether it is affected by a proper motion and the proper motion parameters can all be fitted or not (see Section 4.11).

The code allows to account for differential chromatic effects of atmospheric refraction, i.e. the fact that objects positions in the image plane are shifted by atmospheric refraction in a way that depends on their color. The shift reads:

$$\delta S = k_b(c - c_0)\hat{n} \tag{24}$$

where k_b is a fit parameter (one per band b), c is the color of the object in hand, c_0 is the average color, and $\hat{n}$ is the direction of the displacement in the tangent plane (i.e. a normalized vector along the parallactic direction, computed once for all for each Calexp). We have not accounted for pressure variations because they are usually small, but it would not be difficult. The code accounts for color-driven differential effects within a given band, but ignores the differences across bands, would one attempt to fit images from different bands at the same time. Differences in recorded positions across bands will be accounted for in the fitted mappings. It is important to do so because we are fitting WCS's, and we want the fitted mappings to reflect at best the effects affecting measured positions. Since the color correction (24) is not accounted for when using WCS's to transform measured position, we have made

this correction zero on average. As for proper motions, fitting or not these refraction-induced differential position shifts is left to the user (see Section 4.11).

4.9 Astrometry example

The matrix $W_{\gamma,i}$ is obtained by transporting the measurement errors through the fitted mapping. This introduces an extra dependency of the χ^2 on the parameters, that we have decided not to track in the derivatives, because these errors mostly depend on the mapping scaling, which is very well determined from the beginning. However, small changes of scaling can lead to the χ^2 increasing between iterations. This is why we provide the *AstrometryModel::freezeErrorScales* which allows one to use the *current* state of the model to propagate errors, even if mappings are updated. $dD_{\gamma i}/d\theta$ has two non-zero blocks: the derivatives with respect to the parameters of the M_γ mapping (which are delivered by the Gtransfo-derived class that implements the fitted mapping, namely by the *Gtransfo::paramDerivatives* routine); and the derivative with respect to the F_k position which reads $dP_\gamma(F_k)/dF_k$ (delivered as well by the class that implements the projector, via the *Gtransfo::computeDerivative* routine).

Regarding reference terms, the matrix W_j should be derived from the reference catalog position uncertainty matrix V_0 (typically delivered for (α, δ) coordinates):

$$W_j = (P'^T V_0 P')^{-1}$$

where P' is the derivative of the projector. The inverse of W_j is in practice obtained using the routine *Gtransfo::transformPosAndErrors* which is attached to the projector. The derivative of the reference residual D_j with respect to the *FittedStar* position F_j (see eq. (6)), is just the 2×2 matrix of the derivative P' of the projector P, which we compute using *Gtransfo::computeDerivative*.

4.10 A note about our choice for linear solvers

The standard Cholesky decomposition of a matrix H consists in finding a factor L such that $H = LL^T$, with L triangular (possibly after a permutation of indices). Both Eigen and Cholmod offer a variant, $H = LDL^T$, where D is diagonal and L (still triangular) has 1's on its diagonal. We have settled for this variant, because it offers improved numerical stability and allows one, if needed, to add constraints (via Lagrange multipliers) to the problem. We have also improved the Eigen interface to Cholmod by exposing to the user the factorization update capability of Cholmod, which considerably speeds up the outlier removal. This is done in the

CholmodSimplicialLDLT2 class. Using Cholmod has a drawback: we need its run-time library. Cholmod is now packaged in SuiteSparse, much bigger than what we need. This is why we have packaged the smallest possible subset of SuiteSparse that fulfills our needs into `jointcal_cholmod`.

4.11 Indices of fits parameters and Fits of parameter subsets

Since we use vector algebra to represent the fit parameters, we need some sort of mechanism to associate indices in the vector parameter to some subset (e.g. the position of an FittedStar) of these parameters. Furthermore, the implementation we have chosen does not allow trivially to allocate the actual parameters at successive positions in memory. The *AstrometryFit::AssignIndices* takes care of assigning indices to all classes of parameters. For the mappings, the actual *AstrometryModel* implementation does this part of the job. All these indices are used to properly fill the Jacobian and gradient, and eventually to offset parameters in the *AstrometryFit::OffsetParams*.

Since the indexing of parameters is done dynamically, it is straightforward to only fit a subset of parameters. This is why the routine *AstrometryFit::AssignIndices* takes a string argument that specifies what is to be fitted.

5 Association of the input catalogs

In the LSST stack (Swinbank et al., LDM-151) framework, each reduced input image is call a "calexp" (Calibrated Exposure). Each calexp holds the data from one exposure of one CCD, and we associate the "reduction" products, typically a variance map, image mask planes, and the derived catalog and WCS obtained by matching the catalog to some external reference during single-frame processing. The data that `jointcal` needs from the calexp is stored in a *CcdImage* object. It stores the objects selected from the source catalog (using a configurable *SourceSelector*), the relevant exposure metadata and the initial WCS and *PhotoCalib*.

The *Associations* class holds the list of input *CcdImage*'s and connects together the measurements of the same object. The input measurements are called *MeasuredStar* and the common detections are called *FittedStar*. The objects collected in an external catalog are called *RefStar*. Despite their names, these classes can represent galaxies as well as stars. The collections of sush objects are stored into *MeasuredStarList*, *RefStarList* and *FittedStarList*, which are con-

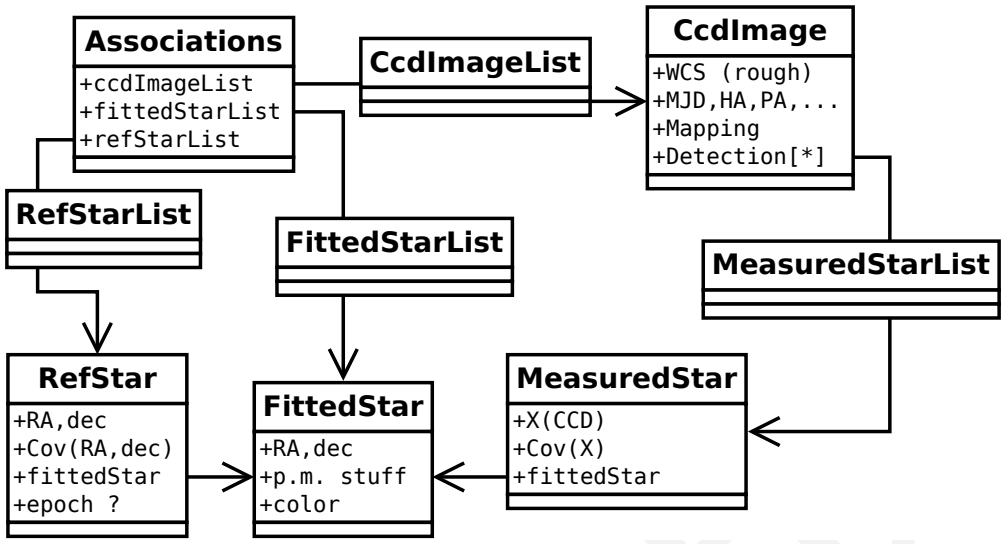

FIGURE 2: Chart of class relations which implement the associations between input catalogs. One *FittedStar* usually has several *MeasuredStar* pointing to it, and each *RefStar* points to exactly one *FittedStar*. Most *FittedStar*'s have no *RefStar*.

tainer derived from *std::list*. The relations between these classes, all implemented in C++, are displayed in Figure 2 on page 16.

6 Fitting the transformations between a set of images

Some applications require determination of transformations between images rather than mappings on the sky. For example a simultaneous fit of PSF photometry for the computation of the light curve a point-like transient requires mappings between images to transport the common position in pixel space from some reference image to any other in the series. The calexp series would typically involve the CCD from each exposure that covers the region of interest. The package described here can fit for the needed mappings:

- in order to remove all reference terms from the χ^2 of eq. (2), one avoids calling *Associations::CollectRefStars*.
- One chooses polynomial mappings for all Calexp except, reserving one to serve as a reference with a fixed identity mapping. The distortion model *SimplePolyModel* allows that.
- Chose identity projectors (the class *IdentityProjector* does precisely that).

So, fitting transformations between image sets can be done with the provided code.

A Representation of distortions in SIP WCS's

The purpose of the appendix is to provide the minimal introduction to WCS concepts required to understand the code (and the comments) when browsing through it. Readers familiar with WCSs can give up here.

WCS's are abstract concepts meant to map data on coordinate systems. In the astronomical imaging framework, this almost always means mapping the pixel space into sidereal coordinates, expressed in some conventional space[1]. One key aspect of the WCS "system" is that it proposes some implementation of the mappings in FITS headers, which comes with software libraries to decode and encode the mappings. The WCS conventions cover a very broad scope of applications, and wide-field imaging makes use of a very small subset of those.

For the mappings used in wide-field imaging, the transformation from pixel space to sky can be pictured in two steps:

1. mapping coordinates in pixel space onto a plane.
2. de-projecting this plane to the celestial sphere.

Let us clear up the projection/de-projection step first. There are plenty of choices possible here, and the differences only matter fir really large images. The projection used by default in the imaging community seems to be the gnomonic projection: the intermediate space is a plane tangent to the celestial sphere and the plane→sphere correspondence is obtained by drawing lines that go through the center of the sphere. In practice there is no need to know that, because any software dealing with WCS's can pick up the right FITS keywords and compute the required projection and de-projection. For this gnomonic projection, one finds `CTYPE1='RA---TAN'` and `CTYPE2='DEC--TAN'` in the FITS header. This projection is often used to generate re-sampled and/or co-added images and one should keep in mind that, for large images, the pixels are not exactly iso-area. One point of convention that might be useful to keep

[1] The WCS concepts are broad enough to accommodate mapping of planet images, but we will obviously not venture into that.

in mind; is that WCS conventions express angles in degrees. In the gnomonic projection, offsets in the tangent plane are expressed in degrees (defined through angles along great circles at the tangent point), so that the metric in the tangent plane is ortho-normal), and sidereal angles evaluated on the sky are also provided in degrees by the standard implementations. A notable exception is the LSST software stack where, by default, the angles are provided in radians.

We now come back to the first mapping step, i.e. converting coordinates measured in pixel units into some intermediate coordinate system. The universal WCS convention here is pretty minimal: it allows for an affine transform, which is in general not sufficient to map the optical distortions of the imaging system, even after a clever choice of the projection. Extensions of the WCS convention have been proposed here, but none is universally understood. The LSST software stack implements the SIP addition, which consists in applying a 2-d polynomial transform to the CCD space coordinates, prior to entering the standard WCS chain (affine transform, then de-projection). In practice, the SIP "twisting" is applied by the LSST software itself (in the class *afw::image::TanWcs*), and the "standard" part (affine and de-projection, or the reverse transform) are sub-contracted to the "libwcs" code.

One common complication of the WCS arena is that it was designed in the FITS framework convention, itself highly Fortran-biased for array indexing, so that the first corner pixel of an image is indexed (1,1). The LSST software, and most modern environments use C-like indexing, i.e. images stars at (0,0), as well as coordinates in images. The WCS LSST software hides this detail to users, by offsetting the pixel space coordinates provided and obtained from the wcs-handling library.

We now detail what is involved in the SIP convention: the SIP "twisting" itself is encoded through 4 polynomials of 2 variables, which encode the direct and reverse transformations. The standard affine transform is expressed through a 2×2 matrix (Cd) and a reference point X_{ref} (called `CRPIX` in the fits header):

$$Y_{TP} = Cd(X_{pix} - X_{ref})$$

X_{pix} is a point in the CCD space, and Y_{TP} is its transform in the tangent plane. Obviously, X_{pix} and X_{ref} should be expressed in the same frame so that the transform does not depend this frame choice. We write symbolically this transform as $Y_{TP} = L(X_{pix})$ The SIP distortions are defined by a polynomial transformation in pixel space, that we call P_A, for the forward

transformation. By convention, the transform from pixel space to tangent plane then reads:

$$Y_{TP} = L\left(X_{pix} - X_{ref} + P_A(X_{pix} - X_{ref})\right)$$

which again does not depend on the frame choice (0-based or 1-based), provided X_{pix} and X_{ref} are expressed in the same frame.

In `jointcal` ,the internal representation of SIP WCS's uses three straight 2d→2d transformations: the SIP correction, the affine transformation and the de-projection. Those are just composed to yield the actual transform, and the two first ones are generic polynomial transformations. We provide routines to translate the TanWcs objects into our representation (*ConvertTanWcs*) and back (*GtransfoToTanWcs*). In the latter case, we also derive the reverse distortion polynomials, which are built if needed in our representation of SIP WCSs.

B Notes on meas_mosaic (from HSC)

Naoki Yasuda wrote *meas_mosaic* for HSC processing, with a similar goal as `jointcal`.

For photometry, *meas_mosaic* fits a 7th order Chebyshev polynomial on the focal plane, plus a zeropoint offset per CCD. The polynomial coefficients are written to the header of **fcr-[visit]-[ccd].fits** files as **C_N_M** values, while the zeropoint and its error is written as **FLUXMAG0** and **FLUXMAG0ERR**. That calibration is applied to all of the fluxes in the catalog, which are written out to the same "*_*flux*" catalog fields (converting them to magnitudes in the process).

C Converting from sparse to dense via Variable Projection

C.1 Motivation

Profiling reveals that for large numbers of visits or objects, jointcal computational performance scales extremely poorly, and is completely dominated by the Cholesky factorization and rank-update steps that we delegate to `CHOLMOD`. While there may be some opportunity to increase performance by improving the outlier rejection logic and hence requesting fewer rank updates, the fact that most of the time is going into routines we do not control (and moreover are the *only* third-party implementation of this algorithm that meets our needs) suggests

that the only way to acceptable jointcal computational performance is to replace the sparse Cholesky factorization with something else. While the total time spent in full factorizations ($\sim 30\%$ at the largest scales we tested) is subdominant to the total time spend in rank updates, the number of outliers grows only linearly with the number of objects, and the time spent in full factorizations seems to be scaling poorly enough on its own to be problematic for LSST (and is already problematic for the HSC deep fields).

Because Cholesky is generally the fastest *direct* factorization method for symmetric matrices, the other obvious solution in the realm of sparse linear algebra would be to switch to an approximate factorization, such as preconditioned conjugate gradient (PCG). Utilizing a PCG solver effectively for nonlinear optimization would represent a major change to the jointcal algorithm, however, as one needs to accept the fact that initial steps can be poor early (due to the approximate nature of the solution), and that this is an acceptable tradeoff for the speed of the factorization, while more accuracy is needed for later steps. The need to identify an effective preconditioner adds another element of tuning and "black magic" to these algorithms, and while there is significant prior art using PCG solvers on this kind of problem, adopting that approach would be a larger change to jointcal than we'd ideally like to make.

The alternative proposed here is to switch away from sparse matrices entirely, by ordering the Hessian matrix H such that the vast majority of it is block diagonal. This allows it to be reduced via a series of small dense subproblems (one per *FittedStar*) to yield a moderate-size dense matrix (with dimensionality the number of non-star parameters) that must be factored each iteration. This is a variant of the Variable Projection algorithm of Golub & Pereyra (1973), but I will derive it directly from the jointcal astrometry objective function (2) to ensure the slight differences to the sparse solution are apparent. That is a major advantage of this approach – while implementing it efficiently in the code may require some restructuring, it should generate a very similar sequence of steps as the sparse Cholesky approach (with differences only due to floating-point round-off error and the use of line search).

I also believe this method is closely related to that used in the current version of the code described in Bernstein et al. (2017), though that may not have been the case in the version described in that paper.

C.2 Derivation and Formalism

We will start by rewriting the explicit sums in (2) as matrix products (we will focus on the astrometry problem only throughout this appendix):

$$\chi^2 = D^T W D \tag{25}$$

with

$$D \equiv \begin{bmatrix} D_1 \\ D_2 \\ \vdots \\ D_i \end{bmatrix} \tag{26}$$

$$W \equiv \begin{bmatrix} W_1 & 0 & 0 & 0 \\ 0 & W_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & W_i \end{bmatrix}, \tag{27}$$

where each block D_i or W_i corresponds to the residual terms or weights (respectively) for one *FittedStar* (including both *MeasuredStar* and *RefStar* terms, as appropriate). They thus have the same *content* as the D and W symbols introduced in Section 4, but different indexing notation and ordering of elements (the ordering is unspecified and unimportant in Section 4; here it is quite important). Note that the dimensionality of these blocks are not the same for all i – the size of the block for *FittedStar* i is set by the number of data points (including reference positions) for that star.

As in the earlier derivation, we recognize that W (and each of its blocks W_i) is positive definite, and hence has a square root. We then define

$$z \equiv W^{1/2} D \tag{28}$$

and

$$z_i \equiv W_i^{1/2} D_i \tag{29}$$

to further simplify the χ^2 to just

$$\chi^2 = z^T z. \tag{30}$$

Using this notation, we will now re-derive the solution for a minimization step (analogous to (16)). We start with the second-order Taylor expansion of χ^2, again using θ as the full vector of free parameters and neglecting the second derivatives of the residuals themselves:

$$\chi^2 \approx z^T z + 2 z^T \boldsymbol{J} \delta\theta + \delta\theta^T \boldsymbol{J}^T \boldsymbol{J} \delta\theta \tag{31}$$

with

$$\boldsymbol{J} \equiv \nabla z. \tag{32}$$

As with the other symbols, $\boldsymbol{J}$ has the same content as the $\boldsymbol{J}$ from Section 4, but with different indexing and the order of its rows more explicit. We wish to zero the derivative of this expansion with respect to $\delta\theta$:

$$\nabla \chi^2 \approx 2 \boldsymbol{J}^T z + 2 \boldsymbol{J}^T \boldsymbol{J} \delta\theta = 0 \tag{33}$$

which yields the equivalent of (16) in the new notation:

$$\boldsymbol{J}^T \boldsymbol{J} \delta\theta = -\boldsymbol{J}^T z. \tag{34}$$

To proceed, we now have to specify an ordering for the elements of the parameter vector θ (and hence the columns of $\boldsymbol{J}$):

- We start with the positional parameters (F_i in Section 4), and, in the future, proper motion and parallax, for each star i, in the same order we used for the data. For notational consistency (using lowercase Greek symbols for free parameters), we will call a segment of $\delta\theta$ for these parameters μ_i and their concatenation μ.
- We finish with all other parameters that are not per-star, using ν for that segment of $\delta\theta$.

These bullets can be expressed as in block-matrix form as

$$\delta\theta = \begin{bmatrix} \mu \\ \nu \end{bmatrix} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_i \\ \nu \end{bmatrix}, \tag{35}$$

and the block form of J is then

$$J = \begin{bmatrix} A & B \end{bmatrix} = \begin{bmatrix} A_1 & 0 & 0 & 0 & B_1 \\ 0 & A_2 & 0 & 0 & B_2 \\ 0 & 0 & \ddots & 0 & \vdots \\ 0 & 0 & 0 & A_i & B_i \end{bmatrix}, \tag{36}$$

where

$$A_i \equiv \nabla_{\mu_i} z_i \tag{37}$$

$$B_i \equiv \nabla_{\nu} z_i. \tag{38}$$

The block-diagonal structure of A is what makes this method practical; it comes about because the derivative of the residuals for one star with respect to the parameters of some other star is zero. Note that the total number of columns in A (typically $\sim 10^5$) is much larger than the number of columns in B ($\sim 10^3$), but the number of columns in a particular A_i (only 2, or perhaps 5 if stellar motion is being fit) is much smaller than the number of columns in a particular B_i (which is the same for all B_i, and the same as the full B: $\sim 10^3$).

Using the compressed block form, we can expand (34) as

$$\begin{bmatrix} A^T \\ B^T \end{bmatrix} \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} \mu \\ \nu \end{bmatrix} = - \begin{bmatrix} A^T z \\ B^T z \end{bmatrix} \tag{39}$$

$$A^T A\mu + A^T B\nu = -A^T z \tag{40}$$

$$B^T A\mu + B^T B\nu = -B^T z \tag{41}$$

while the expanded version of this system of matrix equations is

$$A_i^T A_i \mu_i + A_i^T B_i \nu = -A_i^T z_i \tag{42}$$

$$\sum_i B_i^T A_i \mu_i + \left[\sum_i B_i^T B_i\right] \nu = -\sum_i B_i^T z_i \tag{43}$$

This is enough to see the outlines of the algorithm:

- for each *FittedStar* i, solve (42) for μ_i *as a function of* ν;
- substitute the results into (43) and solve that for ν;
- use ν to fully evaluate each μ_i.

The per-*FittedStar* subproblems are tiny and dense, while the reduced final problem should usually be small enough that a dense factorization is probably more efficient than a sparse one (especially if using a dense solver makes a parallel algorithm available).

C.3 Simplification via QR Factorization

The algorithm derived in the last section is the bulk of this proposal, and it should work regardless of the details of the factorization used to solve the subproblems or the reduced problem. Using a QR factorization of each A_i in the subproblems (42) simplifies the linear algebra involved in substituting μ_i back into (43), however, so working through that approach seems worthwhile. It is quite likely that other factorizations would yield something similar.

We write the column-pivoted QR factorization of A_i as

$$A_i = \begin{bmatrix} U_i & V_i \end{bmatrix} \begin{bmatrix} R_i \\ 0 \end{bmatrix} P_i^T \tag{44}$$

$$= U_i R_i P_i^T, \tag{45}$$

where U_i and V_i are blocks in an orthogonal matrix:

$$\begin{bmatrix} U_i & V_i \end{bmatrix} \begin{bmatrix} U_i^T \\ V_i^T \end{bmatrix} = \begin{bmatrix} U_i^T \\ V_i^T \end{bmatrix} \begin{bmatrix} U_i & V_i \end{bmatrix} = I, \tag{46}$$

R_i is upper triangular, and P_i is a permutation matrix (R and P are *not* related to those defined in 4.1; we're just running out of symbols).

Rearranging (42) and substituting this yields

$$P_i R_i^T R_i P_i^T \mu_i = -P_i R_i^T U_i^T \left(z_i + B_i \nu\right) \tag{47}$$

Noting that μ_i only appears in (43) as $A_i \mu_i$, we can multiply (47) on the left by $U_i R_i^{-T} P_i^T$ to obtain

$$U_i R_i P_i^T \mu_i = -U_i U_i^T \left(z_i + B_i \nu\right) \tag{48}$$

$$A_i \mu_i = -U_i U_i^T \left(z_i + B_i \nu\right) \tag{49}$$

which we can then insert back into (43)

$$-\sum_i B_i^T U_i U_i^T \left(z_i + B_i \nu\right) + \left[\sum_i B_i^T B_i\right] \nu = -\sum_i B_i^T z_i \tag{50}$$

and rearrange to obtain

$$\left[\sum_i B_i^T \left(I - U_i U_i^T\right) B_i\right] \nu = -\sum_i B_i^T \left(I - U_i U_i^T\right) z_i \tag{51}$$

$$\left[\sum_i B_i^T V_i V_i^T B_i\right] \nu = -\sum_i B_i^T V_i V_i^T z_i \tag{52}$$

$$H \nu = -g. \tag{53}$$

We can use this to accumulate the reduced Hessian H and gradient g from their per-star contributions, factor H using the solver of our choice (dense LDL^T seems reasonable), and solve for ν at every iteration. For clarity below, it is useful to define symbols for the contibutions H_i and g_i from star i to H and g (respectively):

$$H_i \equiv B_i^T V_i V_i^T B_i = K_i^T K_i \tag{54}$$

$$g_i \equiv B_i^T V_i V_i^T z_i = K_i^T r_i \tag{55}$$

and

$$K_i \equiv V_i^T B_i \tag{56}$$

$$r_i \equiv V_i^T z \tag{57}$$

While this makes it seem as though we only need V_i from each QR factorization, note that we will need the full factorization (along with reduced solution ν for each iteration) to go back and fully solve for the step μ_i for each *FittedStar* as well. As discussed below in C.5.1, A_i should be approximately constant across iterations, so we should actually be able to compute the per-star QR factorizations once up front.

C.4 Implementation Concerns

C.4.1 Minimizing Refactoring

I believe jointcal already has routines that evaluate the model derivatives J in the form of a sequence of `Eigen::Triplet` objects. If so, these should contain all of the elements of A_i and B_i we need, and the easiest way to migrate to this algorithm would be to instead utilize those triplets to build those blocks as dense matrices instead. If the ordering of rows (measurements) is already consistent with that required by this approach (or can easily be made consistent by manipulating the index-calculation code), that could be done by forming the full J as a sparse matrix and using Eigen block-slicing to obtain A_i and B_i from it.

This may rule out or weaken some of the potential optimizations described in Section C.5, but as those mostly correspond to operations that are currently negligible in the compute budget, that's a reasonable starting point. If this minimal modification proves promising, but we're still not happy with the overall performance, we can then consider refactoring the model- and derivative-evaluation code.

C.4.2 Line Searches

The theoretical (but not numerical) equivalence between this approach and a direct sparse solution to the full problem breaks down if we use a line search to adjust the step size after computing the Gauss-Newton step $\delta\theta$, unless we apply the same scaling factor to both μ_i and

ν. That would entail a new factorization of H at every test point in the line search and break the simplified formula for H we obtained by using QR factorization. That makes it almost certainly a bad idea – it would be much better to simply always accept the full step for μ_i, and if necessary perform a line search on ν only. This makes sense algorithmically, too – we would essentially be assuming that our model positions for stars are always very close to the final solution, even if we are not as confident that we are close to the final solution for other parameters.

C.4.3 Outlier Rejection

Updating the reduced H and g to reflect the removal of some *FittedStar*s as outliers is straightforward – one can simply subtract in the terms in the sums that correspond to those stars. If we end up spending most of our time accumulating H rather than factoring it, an outlier rejection approach much like the current one may still be viable. If not, we will need to make other changes; I don't think a rank-update approach of the type we had in the sparse system is possible.

An obvious mitigation would be to work on reducing the number of outliers we need to reject (by better filtering the input catalogs or adding motion parameters); another would be deferring/bundling outliers to reduce the number of factorizations. In the latter category, we should certainly consider simply rejecting a group of outliers once (perhaps more agressively) every step, and then just factoring the reduced matrix and computing the next step, instead of iterating to reject multiple groups of outliers every step.

C.4.4 Using Eigen

`Eigen::ColPivHouseholderQR` computes exactly the QR factorization in (45). I suspect there is some way to use its `householderQ` method to form matrix products with V_i more efficiently than it would be to use the `matrixQ` method and perform a direct matrix multiplication, but I'm not certain, and it's probably not worth worrying about unless a profile shows those matrix products to be hotspots. I have not tried to work out exactly how best to solve for for μ_i given that factorization and ν; ideally we'd express that via a call to the `solve` method, but we may need to use other terms in the factorization to compute the appropriate right-hand-side vector for that.

C.5 Further Optimization

C.5.1 Utilizing Local Linearity

The residual vectors D_i have the form

$$D_i = \begin{bmatrix} M_1\left(\nu, S_{1,i}\right) - P_1\left(\mu_i\right) \\ M_2\left(\nu, S_{2,i}\right) - P_2\left(\mu_i\right) \\ \vdots \\ M_\gamma\left(\nu, S_{\gamma,i}\right) - P_\gamma\left(\mu_i\right) \\ P\left(\mu_i\right) - P\left(R_i\right) \end{bmatrix}, \tag{58}$$

using the same definitions for $M_{\gamma,i}$, $S_{\gamma,i}$, P_γ, P, and R_i as in 4.1, but using μ_i instead of F_i for the position parameters (as in the rest of this appendix), and making the dependence of $M_{\gamma,i}$ on ν explicit. The last row is present only if we have a reference-catalog match for star i.

The corresponding matrix of weights W_i is actually also derived from uncertainties originally measured either in pixel coordinates (and hence transformed by $M_{\gamma,i}$ as well) or, for reference positions, in celestial coordinates.[2] Using $C_{\gamma,i}$ as the 2×2 uncertainty covariance matrix for measurement $S_{\gamma,i}$ and C_i for the reference star uncertainty covariance matrix (also 2×2), then:

$$W_i^{-1} = \begin{bmatrix} \nabla_S M_{1,i}^T C_{1,i} \nabla_S M_{1,i} & 0 & 0 & 0 & 0 \\ 0 & \nabla_S M_{2,i}^T C_{2,i} \nabla_S M_{2,i} & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & \nabla_S M_{\gamma,i}^T C_{\gamma,i} \nabla_S M_{\gamma,i} & 0 \\ 0 & 0 & 0 & 0 & \nabla_R P^T C_{\gamma,i} \nabla_R P \end{bmatrix} \tag{59}$$

where $\nabla_S M_{\gamma,i}$ is the 2×2 derivative of M with respect to the input pixel position $S_{\gamma,i}$, not the model parameters ν, and $\nabla_R P$ is the derivative of the projection P with respect to the input celestial position (again, also 2×2). W_i thus has no dependence on μ_i, so the derivative matrix

[2] Reference catalogs may actually report uncertainties in a local per-object tangent plane, but the distinction is irrelevant for the primary purpose of this discussion.

A_i of $z_i = W_i^{1/2} D_i$ with respect to μ_i is

$$A_i = \nabla_\mu z_i = W_i^{1/2} \nabla_\mu D_i = W_i^{1/2} \begin{bmatrix} -\nabla_\mu P_1\left(\mu_i\right) \\ -\nabla_\mu P_2\left(\mu_i\right) \\ \vdots \\ -\nabla_\mu P_\gamma\left(\mu_i\right) \\ \nabla_\mu P\left(\mu_i\right) \end{bmatrix} \tag{60}$$

Both $M_{\gamma,i}$ and the projections P_i and P represent coordinate transformations that we expect to be locally linear: as long as we evaluate them in the neighborhood of a local point, their second derivative is approximately zero and hence their first derivative is approximately constant (in fact, if this was ever not true of an astronomical coordinate system, we would not be able to represent the action of the point-spread function on it as a convolution, so we would have much bigger problems than just fitting astrometric and photometric calibrations). Because the A_i matrices contain only the derivatives of these mappings, and not the mappings themselves, then, we should be able to consider them constant for the duration of the fit, and compute their QR decompositions only once, up front. This should continue to be true when μ_i and the columns of A_i are extended to include motion parameters.

The same is not quite true of B_i, which has the form

$$B_i = \nabla_\nu z_i = \nabla_\nu W_i^{1/2} D_i + W_i^{1/2} \nabla_\nu D_i \tag{61}$$

Propagating that derivative through the matrix square root would be unpleasant, to say the least, but again we can take advantage of local linearity to avoid this: for any plausible ν, we know that $M_{\gamma,i}$ should be locally linear, and hence $\nabla_\nu \nabla_S M_{\gamma,i} \approx 0$. We can then approximate B_i as

$$B_i \approx W_i^{1/2} \nabla_\nu D_i = W_i^{1/2} \begin{bmatrix} \nabla_\nu M_1\left(\nu, S_{1,i}\right) \\ \nabla_\nu M_2\left(\nu, S_{2,i}\right) \\ \vdots \\ \nabla_\nu M_\gamma\left(\nu, S_{\gamma,i}\right) \\ 0 \end{bmatrix} \tag{62}$$

I strongly suspect that jointcal already makes this approximation, but I also suspect that jointcal currently recomputes W_i and hence $W_i^{1/2}$ at every iteration even though it should be safe to consider it constant throughout the fit. Of course, even if it does, it's a negligible part of the

compute budget at present.

C.5.2 Sparsity

While the rows of each B_i correspond only to the measurements of *FittedStar* i, the columns correspond to all non-star parameters, even those associated with model terms (visits or CCDs) that do not have a measurement for that star, and hence some will be entirely zero. When the full set of input observations is is wide enough to cover many focal planes (or there are more global parameters than per-visit parameters), each B_i could thus have many zero-valued columns and its contribution H_i will be even more sparse. It may make sense to explicitly account for this in the accumulation of H and g and/or the final solutions for μ_i.

The first step – and probably the only one worth considering for now – would be to store each B_i as a sequence of matrix blocks (each corresponding to the parameters of one or more visits or CCDs relevant for star i) and their column offsets into the larger matrix, with each block having the same row extent (corresponding to the measurements and reference positions for i). The product $K_i \equiv V_i^T B_i$ could be stored in the same way (it has the same columns, but different rows), and computed as the standard dense matrix product of V_i^T with each block of B_i. For example, if some B_i has the block form:

$$B_i = \left[\begin{array}{cccccc} B_i^{(1)} & B_i^{(2)} & 0 & 0 & B_i^{(5)} & 0 \end{array} \right] \tag{63}$$

then K_i has the same form (with $K_i^{(j)} \equiv V_i^T B_i^{(j)}$):

$$K_i = \left[\begin{array}{cccccc} K_i^{(1)} & K_i^{(2)} & 0 & 0 & K_i^{(5)} & 0 \end{array} \right], \tag{64}$$

and we would store only $K_i^{(1)}$, $K_i^{(2)}$, and $K_i^{(5)}$ and their column offsets (note that the superscript indices here are not meaningful; these blocks may represent visits, CCDs, or groups thereof).

Accumulating H from this data structure would involve computing the inner product of all combinations of K_i blocks and adding the results to the blocks of H that correspond to their offsets; using the same example as above, the contribution H_i from that *FittedStar* would look

like:

$$H_i = \begin{bmatrix} K_i^{(1)T} K_i^{(1)} & K_i^{(1)T} K_i^{(2)} & 0 & 0 & K_i^{(1)T} K_i^{(5)} & 0 \\ K_i^{(2)T} K_i^{(1)} & K_i^{(2)T} K_i^{(2)} & 0 & 0 & K_i^{(2)T} K_i^{(5)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ K_i^{(5)T} K_i^{(1)} & K_i^{(5)T} K_i^{(2)} & 0 & 0 & K_i^{(5)T} K_i^{(5)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{65}$$

Accumulating g would similarly involve computing the matrix-vector product of each block with $r_i = V_i^T z_i$ and adding the results to the appropriate segments of g:

$$g_i = \begin{bmatrix} K_i^{(1)T} r_i \\ K_i^{(2)T} r_i \\ 0 \\ 0 \\ K_i^{(5)T} r_i \\ 0 \end{bmatrix}. \tag{66}$$

We do not expect r_i to have many zero elements, so no special storage will be useful there.

Note that while each H_i is sparse, the full H will not be, at least in our current mode of operation (per-tract, with tracts approximately the scale of a visit). If that changes, we could instead accumulate into a sparse matrix and again use a sparse solver (but still on a much smaller-scale problem, albeit one with a larger fill factor).

If we ever have many more global parameters than per-visit parameters (e.g. by fitting the optical distortion as a constant, and the per-visit atmospheric contribution is lower-order), it may be worth trying to the same approach again, by solving per-visit subproblems and accumulating those to form a fully-reduced Hessian containing only the global parameters. I have not attempted to derive this extension, and am by no means certain that it is even viable (it depends on whether the partially-reduced Hessian H derived above also has a block structure we can take advantage of). I would certainly try a sparse solver on H first.

C.5.3 Parallelization

One potential advantage of switching to dense solvers is that we're much more likely to be able to find a parallel implementation. The per-star subproblems are also completely independent, and could be run in parallel as well (if we can get OpenMP working at all with our build system and dependencies).

This is potentially extremely valuable: there are many fewer invocations of jointcal (i.e. one per tract) than most other pipeline steps, so we should always have plenty of cores to throw at it if it can make use of them.

I would not advocate trying to add multithreading at the same time we switch to this new approach to solving the linear algebra, but it would be worth keeping in mind when restructuring the code (e.g. by making the inputs and outputs of each per-*FittedStar* subproblem routine explicit as arguments or return values, rather than utilizing class data members that might be dangerous to share across threads).

It's also worth noting that while Eigen can be configured to use OpenMP, this is a compile-time option, and I don't think it's something we could turn on in Jointcal only without a lot of build-system cleverness and wrapping (we had the same problem with `meas_mosaic`). Eigen's parallel implementations also aren't nearly as optimized as their single-threaded versions, so if we do want to use a parallel solver for the reduced problem, I'd just look for a different one.

References

Bernstein, G.M., Armstrong, R., Plazas, A.A., et al., 2017, PASP, 129, 074503 (arXiv:1703.01679), doi:10.1088/1538-3873/aa6c55, ADS Link

Bertin, E., 2006, In: Gabriel, C., Arviset, C., Ponz, D., Enrique, S. (eds.) Astronomical Data Analysis Software and Systems XV, vol. 351 of Astronomical Society of the Pacific Conference Series, 112, ADS Link

Eichhorn, H., 1960, Astronomische Nachrichten, 285, 233, doi:10.1002/asna.19592850507, ADS Link

Gaia Collaboration, Brown, A.G.A., Vallenari, A., et al., 2016, A&A, 595, A2 (arXiv:1609.04172), doi:10.1051/0004-6361/201629512

Golub, G.H., Pereyra, V., 1973, SIAM Journal on Numerical Analysis, 10, 413, doi:10.1137/0710036, ADS Link

Magnier, E.A., Schlafly, E., Finkbeiner, D., et al., 2013, ApJS, 205, 20 (arXiv:1303.3634), doi:10.1088/0067-0049/205/2/20, ADS Link

Padmanabhan, N., Schlegel, D.J., Finkbeiner, D.P., et al., 2008, ApJ, 674, 1217 (arXiv:astro-ph/0703454), doi:10.1086/524677, ADS Link

[LDM-151], Swinbank, J.D., et al., 2017, *Data Management Science Pipelines Design*, LDM-151, URL https://ls.st/LDM-151