Appendix 6

Iterative Estimation Methods

In this appendix we describe the various components involved in building an efficient and robust iterative estimation algorithm.

We start with two of the most common iterative parameter minimization methods, namely Newton iteration (and the closely related Gauss-Newton method) and Levenberg–Marquardt iteration. The general idea of Newton iteration is familiar to most students of numerical methods as a way of finding the zeros of a function of a single variable. Its generalization to several variables and application to finding least-squares solutions rather than exact solutions to sets of equations is relatively straightforward. The Levenberg–Marquardt method is a simple variation on Newton iteration designed to provide faster convergence and regularization in the case of over-parametrized problems. It may be seen as a hybrid between Newton iteration and a gradient descent method.

For the type of problem considered in this book, important reductions of computational complexity are obtained by dividing the set of parameters into two parts. The two parts generally consist of a set of parameters representing camera matrices or homographies, and a set of parameters representing points. This leads to a sparse structure to the problem that is described starting at section A6.3.

We discuss two further implementation issues – the choice of cost function, with respect to their robustness to outliers and convexity (section A6.8); and the parametrization of rotations, and homogeneous and constrained vectors (section A6.9). Finally, those readers who want to learn more about iterative techniques and bundle-adjustment are referred to [Triggs-00a] for more details.

A6.1 Newton iteration

Suppose we are given a hypothesized functional relation $\mathbf{X} = \mathbf{f}(\mathbf{P})$ where $\mathbf{X}$ is a *measurement vector* and $\mathbf{P}$ is a *parameter vector* in Euclidean spaces $\mathbb{R}^N$ and $\mathbb{R}^M$ respectively. A measured value of $\mathbf{X}$ approximating the true value $\overline{\mathbf{X}}$ is provided, and we wish to find the vector $\widehat{\mathbf{P}}$ that most nearly satisfies this functional relation. More precisely, we seek the vector $\widehat{\mathbf{P}}$ satisfying $\mathbf{X} = \mathbf{f}(\widehat{\mathbf{P}}) - \boldsymbol{\epsilon}$ for which $\|\boldsymbol{\epsilon}\|$ is minimized. Note that the linear least-squares problem considered in section A5.1 is exactly of this type, the function $\mathbf{f}$ being defined as a linear function $\mathbf{f}(\mathbf{P}) = \mathtt{A}\mathbf{P}$.

To solve the case where $\mathbf{f}$ is not a linear function, we may start with an initial estimated value $\mathbf{P}_0$, and proceed to refine the estimate under the assumption that the function $\mathbf{f}$ is locally linear. Let $\boldsymbol{\epsilon}_0$ be defined by $\boldsymbol{\epsilon}_0 = \mathbf{f}(\mathbf{P}_0) - \mathbf{X}$. We assume that the function $\mathbf{f}$ is approximated at $\mathbf{P}_0$ by $\mathbf{f}(\mathbf{P}_0 + \boldsymbol{\Delta}) = \mathbf{f}(\mathbf{P}_0) + \mathtt{J}\boldsymbol{\Delta}$, where $\mathtt{J}$ is the linear mapping represented by the Jacobian matrix $\mathtt{J} = \partial\mathbf{f}/\partial\mathbf{P}$. We seek a point $\mathbf{f}(\mathbf{P}_1)$, with $\mathbf{P}_1 = \mathbf{P}_0 + \boldsymbol{\Delta}$, which minimizes $\mathbf{f}(\mathbf{P}_1) - \mathbf{X} = \mathbf{f}(\mathbf{P}_0) + \mathtt{J}\boldsymbol{\Delta} - \mathbf{X} = \boldsymbol{\epsilon}_0 + \mathtt{J}\boldsymbol{\Delta}$. Thus, it is required to minimize $\|\boldsymbol{\epsilon}_0 + \mathtt{J}\boldsymbol{\Delta}\|$ over $\boldsymbol{\Delta}$, which is a linear minimization problem. The vector $\boldsymbol{\Delta}$ is obtained by solving the *normal* equations (see (A5.2))

$$\mathtt{J}^\mathsf{T}\mathtt{J}\boldsymbol{\Delta} = -\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}_0 \tag{A6.1}$$

or by using the pseudo-inverse $\boldsymbol{\Delta} = -\mathtt{J}^+\boldsymbol{\epsilon}_0$. Thus, the solution vector $\widehat{\mathbf{P}}$ is obtained by starting with an estimate $\mathbf{P}_0$ and computing successive approximations according to the formula

$$\mathbf{P}_{i+1} = \mathbf{P}_i + \boldsymbol{\Delta}_i$$

where $\boldsymbol{\Delta}_i$ is the solution to the linear least-squares problem

$$\mathtt{J}\boldsymbol{\Delta}_i = -\boldsymbol{\epsilon}_i.$$

Matrix $\mathtt{J}$ is the Jacobian $\partial\mathbf{f}/\partial\mathbf{P}$ evaluated at $\mathbf{P}_i$ and $\boldsymbol{\epsilon}_i = \mathbf{f}(\mathbf{P}_i) - \mathbf{X}$. One hopes that this algorithm will converge to the required least-squares solution $\widehat{\mathbf{P}}$. Unfortunately, it is possible that this iteration procedure converges to a local minimum value, or does not converge at all. The behaviour of the iteration algorithm depends very strongly on the initial estimate $\mathbf{P}_0$.

Weighted iteration. As an alternative to all the dependent variables being equally weighted it is possible to provide a weight matrix specifying the weights of the dependent variables $\mathbf{X}$. To be more precise, one may assume that the measurement $\mathbf{X}$ satisfies a Gaussian distribution with covariance matrix $\Sigma_\mathbf{X}$, and one wishes to minimize the Mahalanobis distance $\|\mathbf{f}(\widehat{\mathbf{P}}) - \mathbf{X}\|_\Sigma$. This covariance matrix may be diagonal, specifying that the individual coordinates of $\mathbf{X}$ are independent, or more generally it may be an arbitrary symmetric and positive definite matrix. In this case, the normal equations become $\mathtt{J}^\mathsf{T}\Sigma^{-1}\mathtt{J}\boldsymbol{\Delta}_i = -\mathtt{J}^\mathsf{T}\Sigma^{-1}\boldsymbol{\epsilon}_i$. The rest of the algorithm remains unchanged.

Newton's method and the Hessian. We pass now to consideration of finding minima of functions of many variables. For the present, we consider an arbitrary scalar-valued function $g(\mathbf{P})$ where $\mathbf{P}$ is a vector. The optimization problem is simply to minimize $g(\mathbf{P})$ over all values of $\mathbf{P}$. We make two assumptions: that $g(\mathbf{P})$ has a well-defined minimum value, and that we know a point $\mathbf{P}_0$ reasonably close to this minimum.

We may expand $g(\mathbf{P})$ about $\mathbf{P}_0$ in a Taylor series to get

$$g(\mathbf{P}_0 + \boldsymbol{\Delta}) = g + g_\mathbf{P}\boldsymbol{\Delta} + \boldsymbol{\Delta}^\mathsf{T} g_{\mathbf{PP}}\boldsymbol{\Delta}/2 + \dots$$

where subscript $\mathbf{P}$ denotes differentiation, and the right hand side is evaluated at $\mathbf{P}_0$.

We wish to minimize this quantity with respect to $\mathbf{\Delta}$. To this end, we differentiate with respect to $\mathbf{\Delta}$ and set the derivative to zero, arriving at the equation $g_{\mathbf{P}} + g_{\mathbf{PP}}\mathbf{\Delta} = 0$ or

$$g_{\mathbf{PP}}\mathbf{\Delta} = -g_{\mathbf{P}}. \tag{A6.2}$$

In this equation, $g_{\mathbf{PP}}$ is the matrix of second derivatives, the *Hessian* of g, the (i, j)-th entry of which is $\partial^2 g/\partial p_i \partial p_j$, and p_i and p_j are the i-th and j-th parameters. Vector $g_{\mathbf{P}}$ is the gradient of g. The method of *Newton iteration* consists in starting at an initial value of the parameters, $\mathbf{P}_0$, and iteratively computing parameter increments $\mathbf{\Delta}$ using (A6.2) until convergence occurs.

Now we turn to the sort of cost function that arises in the least-squares minimization problem considered above. Specifically, $g(\mathbf{P})$ is the squared norm of an error function

$$g(\mathbf{P}) = \frac{1}{2}\|\boldsymbol{\epsilon}(\mathbf{P})\|^2 = \boldsymbol{\epsilon}(\mathbf{P})^{\mathsf{T}}\boldsymbol{\epsilon}(\mathbf{P})/2$$

where $\boldsymbol{\epsilon}(\mathbf{P})$ is a vector-valued function of the parameter vector $\mathbf{P}$. In particular, $\boldsymbol{\epsilon}(\mathbf{P}) = \mathbf{f}(\mathbf{P}) - \mathbf{X}$. The factor $1/2$ is present for simplifying the succeeding computations.

The gradient vector $g_{\mathbf{P}}$ is easily computed to be $\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$. However, we may write $\boldsymbol{\epsilon}_{\mathbf{P}} = \mathbf{f}_{\mathbf{P}} = \mathtt{J}$ using the notation introduced previously. In short $g_{\mathbf{P}} = \mathtt{J}^{\mathsf{T}}\boldsymbol{\epsilon}$. Differentiating $g_{\mathbf{P}} = \boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$ a second time, we compute the following formula for the Hessian.[1]

$$g_{\mathbf{PP}} = \boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}_{\mathbf{P}} + \boldsymbol{\epsilon}_{\mathbf{PP}}^{\mathsf{T}}\boldsymbol{\epsilon}. \tag{A6.3}$$

Now, under the assumption that $\mathbf{f}(\mathbf{P})$ is linear, the second term on the right vanishes, leaving $g_{\mathbf{PP}} = \boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}_{\mathbf{P}} = \mathtt{J}^{\mathsf{T}}\mathtt{J}$. Now, substituting for the gradient and Hessian in (A6.2) yields $\mathtt{J}^{\mathsf{T}}\mathtt{J}\mathbf{\Delta} = -\mathtt{J}^{\mathsf{T}}\boldsymbol{\epsilon}$ which is nothing more than the normal equations (A6.1). Thus we have arrived at the same iterative procedure as previously solving the parameter estimation problem, under the assumption that $\mathtt{J}^{\mathsf{T}}\mathtt{J} = \boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}_{\mathbf{P}}$ is a reasonable approximation for the Hessian of the function $g(\mathbf{P})$. This procedure in which $\mathtt{J}^{\mathsf{T}}\mathtt{J}$ is used as an approximation for the Hessian is known as the *Gauss-Newton* method.

Gradient descent. The negative (or down-hill) gradient vector $-g_{\mathbf{P}} = -\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$ defines the direction of most rapid decrease of the cost function. A strategy for minimization of g is to move iteratively in the gradient direction. This is known as *gradient descent*. The length of the step may be computed by carrying out a line search for the function minimum in the negative gradient direction. In this case, the parameter increment $\mathbf{\Delta}$ is computed from the equation $\lambda\mathbf{\Delta} = -g_{\mathbf{P}}$, where λ controls the length of the step.

We may consider this as related to Newton iteration as expressed by the update equation (A6.2) in which the Hessian is approximated (somewhat arbitrarily) by the scalar matrix $\lambda\mathtt{I}$. Gradient descent by itself is not a very good minimization strategy, typically characterized by slow convergence due to zig-zagging. (See [Press-88] for a closer analysis.) It will be seen in the next section, however, that it can be useful in conjunction with Gauss-Newton iteration as a way of getting out of tight corners. The

[1] The last term in this formula needs some clarification. Since $\boldsymbol{\epsilon}$ is a vector, $\boldsymbol{\epsilon}_{\mathbf{PP}}$ is a 3-dimensional array (a tensor). The sum implied by the product $\boldsymbol{\epsilon}_{\mathbf{PP}}^{\mathsf{T}}\boldsymbol{\epsilon}$ is over the components of $\boldsymbol{\epsilon}$. It may be written more precisely as $\sum_i (\epsilon_i)_{\mathbf{PP}}\epsilon_i$ where ϵ_i is the i-th components of the vector $\boldsymbol{\epsilon}$, and $(\epsilon_i)_{\mathbf{PP}}$ is its Hessian.

Levenberg–Marquardt method is essentially a Gauss-Newton method that transitions smoothly to gradient descent when the Gauss-Newton updates fail.

To summarize, we have so far considered three methods of minimization of a cost function $g(\mathbf{P}) = \|\boldsymbol{\epsilon}(\mathbf{P})\|^2/2$:

(i) **Newton.** Update equation:

$$g_{\mathbf{PP}}\boldsymbol{\Delta} = -g_{\mathbf{P}}$$

where $g_{\mathbf{PP}} = \boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}_{\mathbf{P}} + \boldsymbol{\epsilon}_{\mathbf{PP}}^{\mathsf{T}}\boldsymbol{\epsilon}$ and $g_{\mathbf{P}} = \boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$. Newton iteration is based on the assumption of an approximately quadratic cost function near the minimum, and will show rapid convergence if this condition is met. The disadvantage of this approach is that the computation of the Hessian may be difficult. In addition, far from the minimum the assumption of quadratic behaviour is probably invalid, so a lot of extra work is done with little benefit.

(ii) **Gauss-Newton.** Update equation:

$$\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}_{\mathbf{P}}\boldsymbol{\Delta} = -\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}$$

This is equivalent to Newton iteration in which the Hessian is approximated by $\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon}_{\mathbf{P}}$. Generally this is a good approximation, particularly close to a minimum, or when $\boldsymbol{\epsilon}$ is nearly linear in $\mathbf{P}$.

(iii) **Gradient descent.** Update equation:

$$\lambda\boldsymbol{\Delta} = -\boldsymbol{\epsilon}_{\mathbf{P}}^{\mathsf{T}}\boldsymbol{\epsilon} = -g_{\mathbf{P}}.$$

The Hessian in Newton iteration is replaced by a multiple of the identity matrix. Each update is in the direction of most rapid local decrease of the function value. The value of λ may be chosen adaptively, or by a line search in the downward gradient direction. Generally, gradient descent by itself is not recommended, but in conjunction with Gauss-Newton it yields the commonly used Levenberg–Marquardt method.

A6.2 Levenberg–Marquardt iteration

The Levenberg–Marquardt (abbreviated LM) iteration method is a slight variation on the Gauss-Newton iteration method. The normal equations $\mathtt{J}^{\mathsf{T}}\mathtt{J}\boldsymbol{\Delta} = -\mathtt{J}^{\mathsf{T}}\boldsymbol{\epsilon}$ are replaced by the *augmented normal equations* $(\mathtt{J}^{\mathsf{T}}\mathtt{J} + \lambda\mathtt{I})\boldsymbol{\Delta} = -\mathtt{J}^{\mathsf{T}}\boldsymbol{\epsilon}$, for some value of λ that varies from iteration to iteration. Here $\mathtt{I}$ is the identity matrix. A typical initial value of λ is 10^{-3} times the average of the diagonal elements of $\mathtt{N} = \mathtt{J}^{\mathsf{T}}\mathtt{J}$.

If the value of $\boldsymbol{\Delta}$ obtained by solving the augmented normal equations leads to a reduction in the error, then the increment is accepted and λ is divided by a factor (typically 10) before the next iteration. On the other hand if the value $\boldsymbol{\Delta}$ leads to an increased error, then λ is multiplied by the same factor and the augmented normal equations are solved again, this process continuing until a value of $\boldsymbol{\Delta}$ is found that gives rise to a decreased error. This process of repeatedly solving the augmented normal equations for different values of λ until an acceptable $\boldsymbol{\Delta}$ is found constitutes one iteration of the LM algorithm. An implementation of the LM algorithm is given in [Press-88].

Justification of LM. To understand the reasoning behind this method, consider what happens for different values of λ. When λ is very small, the method is essentially the same as Gauss-Newton iteration. If the error function $\|\boldsymbol{\epsilon}\|^2 = \|\mathbf{f}(\mathbf{P}) - \mathbf{X}\|^2$ is close to being quadratic in $\mathbf{P}$, then this method will converge fast to the minimum value. On the other hand when λ is large then the normal equation matrix is approximated by $\lambda\mathtt{I}$, and the normal equations become $\lambda\boldsymbol{\Delta} = -\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$. Recalling that $\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$ is simply the gradient vector of $\|\boldsymbol{\epsilon}\|^2$, we see that the direction of the parameter increment $\boldsymbol{\Delta}$ approaches that given by gradient descent. Thus, the LM algorithm moves seamlessly between Gauss-Newton iteration, which will cause rapid convergence in the neighbourhood of the solution, and a gradient descent approach, which will guarantee a decrease in the cost function when the going is difficult. Indeed, as λ becomes larger and larger, the length of the increment step $\boldsymbol{\Delta}$ decreases and eventually it will lead to a decrease of the cost function $\|\boldsymbol{\epsilon}\|^2$.

To demonstrate that the parameter increment $\boldsymbol{\Delta}$ obtained by solving the augmented normal equations for all values of λ is in the direction of decreasing cost, we will show that the inner product of $\boldsymbol{\Delta}$ and the negative gradient direction for the function $g(\mathbf{P}) = \|\boldsymbol{\epsilon}(\mathbf{P})\|^2$ is positive. This results from the following computation.

$$\begin{aligned} -g_\mathbf{P} \cdot \boldsymbol{\Delta} &= -g_\mathbf{P}^\mathsf{T}\boldsymbol{\Delta} \\ &= (\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon})^\mathsf{T}(\mathtt{J}^\mathsf{T}\mathtt{J} + \lambda\mathtt{I})^{-1}\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon} \end{aligned}$$

However, $\mathtt{J}^\mathsf{T}\mathtt{J}+\lambda\mathtt{I}$ is positive-definite for any value of λ, and so therefore is its inverse. By definition, this means that $(\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon})^\mathsf{T}(\mathtt{J}^\mathsf{T}\mathtt{J} + \lambda\mathtt{I})^{-1}\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$ is positive, unless $\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$ is zero. Thus, the increment $\boldsymbol{\Delta}$ is in a direction of locally decreasing cost, unless of course the gradient $\mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$ is zero.

A different augmentation. In some implementations of Levenberg-Marquardt, notably that given in [Press-88], a different method of augmenting the normal equations is used. The augmented normal equation matrix $\mathtt{N}'$ is defined in terms of the matrix $\mathtt{N} = \mathtt{J}^\mathsf{T}\mathtt{J}$ by $\mathtt{N}'_{ii} = (1+\lambda)\mathtt{N}_{ii}$ and $\mathtt{N}'_{ij} = \mathtt{N}_{ij}$ for $i \neq j$. Thus the diagonal of $\mathtt{N}$ is augmented by a multiplicative factor $(1+\lambda)$ instead of an additive factor. As before, a small value of λ, results essentially in a Gauss-Newton update. For large λ, the off-diagonal entries of the normal equation matrix become insignificant with respect to the diagonal ones.

The i-th diagonal entry of $\mathtt{N}'$ is simply $(1+\lambda)\mathbf{J}_i^\mathsf{T}\mathbf{J}_i$ where $\mathbf{J}_i = \partial\mathbf{f}/\partial p_i$, and p_i is the i-th parameter. The update equation is then $(1+\lambda)\mathbf{J}_i^\mathsf{T}\mathbf{J}_i\delta_i = \mathbf{J}_i^\mathsf{T}\boldsymbol{\epsilon}$ where δ_i is the increment in the i-th parameter. Apart from the factor $(1+\lambda)$, this is the normal equation that would result from minimizing the cost by varying only the i-th parameter δ_i. Thus, in the limit as λ becomes large, the increment to the parameters is the direction that would result from minimizing each one separately.

With this sort of augmentation, the parameter increments for large λ are not the same as for gradient descent. Nevertheless, the same analysis as before shows that the resulting increment is still in the down-hill direction for any value of λ.

One small problem may arise: if some parameter p_i has no effect on the value of the function $\mathbf{f}$, then $\mathbf{J}_i = \partial\mathbf{f}/\partial p_i$ is zero, and the i-th diagonal entry of $\mathtt{N}$ and hence $\mathtt{N}'$

is zero. The augmented normal equation matrix $\mathtt{N}'$ is then singular, which can cause trouble. In practice, this is a rare occurrence, but it can occur.

Implementation of LM. To carry out Levenberg–Marquardt minimization in its simplest form it is necessary only to provide a routine to compute the function being minimized, a goal vector $\hat{\mathbf{X}}$ of observed or desired values of the function, and an initial estimate $\mathbf{P}_0$. Computation of the Jacobian matrix J may be carried out either numerically, or by providing a custom routine.

Numerical differentiation may be carried out as follows. Each independent variable x_i is incremented in turn to $x_i + \delta$, the resulting function value is computed using the routine provided for computing f and the derivative is computed as a ratio. Good results have been found by setting the value δ to the maximum of $|10^{-4} \times x_i|$ and 10^{-6}. This choice seemingly gives a good approximation to the derivative. In practice, there seems to be little disadvantage in using numerical differentiation, though for simple functions f one may prefer to provide a routine to compute J, partly for aesthetic reasons, partly because of a possible slightly improved convergence and partly for speed.

A6.3 A sparse Levenberg–Marquardt algorithm

The LM algorithm as described above in section A6.2 is quite suitable for minimization with respect to a small number of parameters. Thus, in the case of 2D homography estimation (see chapter 4), for the simple cost functions (4.6–*p*94) and (4.7–*p*95) which are minimized with respect only to the entries of the homography matrix H the LM algorithm works well. However, for minimizing cost functions with respect to large numbers of parameters, the bare LM algorithm is not very suitable. This is because the central step of the LM algorithm, the solution of the normal equations (A5.2), has complexity N^3 in the number of parameters, and this step is repeated many times. However, in solving many estimation problems of the type considered in this book, the normal equation matrix has a certain sparse block structure that one may take advantage of to realize very great time savings.

An example of the situation in which this method is useful is the problem of estimating a 2D homography between two views assuming errors in both images, by minimizing the cost function (4.8–*p*95). This problem may be parametrized in terms of a set of parameters characterizing the 2D homography (perhaps the 9 entries of the homography matrix), plus parameters for each of the n points in the first view, a total of $2n + 9$ parameters.

Another instance where these methods are useful is the reconstruction problem in which one has image correspondences across two or more (let us say m) views and one wishes to estimate the camera parameters of all the cameras and also the 3D positions of all the points. One can assume arbitrary projective cameras or fully or partially calibrated cameras. Furthermore, to remove some of the incidental degrees of freedom one can fix one of the cameras. For instance in the projective reconstruction problem, the problem may be parametrized by the entries of all the camera matrices (a total of $12m$ or $11m$ parameters depending on how the cameras are parametrized), plus a set of $3n$ parameters for the coordinates of the 3D points.

The sparse LM algorithm is often perceived as complex and difficult to implement. To help overcome this, the algorithms are given cook-book style. Given a suitable library of standard matrix manipulation routines, a reader should be able to implement the algorithm without difficulty.

- **Notation:** *If $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ are vectors, then the vector obtained by placing them one after the other in a single column vector is denoted by $(\mathbf{a}_1^\mathsf{T}, \mathbf{a}_2^\mathsf{T}, \ldots, \mathbf{a}_n^\mathsf{T})^\mathsf{T}$. A similar notation is used for matrices.*

A6.3.1 Partitioning the parameters in the LM method

The sparse LM method will be described primarily in terms of the reconstruction problem, since this is the archetypal problem to which this method relates. The estimation problem will be treated in general terms first of all, since this will illuminate the general approach without excessive detail. At this level of abstraction, the benefits of this approach may not be apparent, but they will become clearer in section A6.3.3. We start with the simple observation that the set of parameters in this problem may be divided up into two sets: a set of parameters describing the cameras, and a set of parameters describing the points. More formally, the "parameter vector" $\mathbf{P} \in \mathbb{R}^M$ may be partitioned into parameter vectors $\mathbf{a}$ and $\mathbf{b}$ so that $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T}$. We are given a "measurement vector", denoted by $\mathbf{X}$ in a space $\mathbb{R}^N$. In the reconstruction problem, this consists of the vector of all image point coordinates. In addition let $\Sigma_\mathbf{X}$ be the covariance matrix for the measurement vector.[1] We consider a general function $f : \mathbb{R}^M \to \mathbb{R}^N$ taking the parameter vector $\mathbf{P}$ to the estimated measurement vector $\hat{\mathbf{X}} = f(\mathbf{P})$. Denoting by $\boldsymbol{\epsilon}$ the difference $\mathbf{X} - \hat{\mathbf{X}}$ between the measured and estimated quantities, one seeks the set of parameters that minimize the squared Mahalanobis distance $\|\boldsymbol{\epsilon}\|^2_{\Sigma_\mathbf{X}} = \boldsymbol{\epsilon}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\boldsymbol{\epsilon}$.

Corresponding to the division of parameters $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T}$, the Jacobian matrix $\mathtt{J} = [\partial\hat{\mathbf{X}}/\partial\mathbf{P}]$ has a block structure of the form $\mathtt{J} = [\mathtt{A}|\mathtt{B}]$, where Jacobian submatrices are defined by

$$\mathtt{A} = \left[\partial\hat{\mathbf{X}}/\partial\mathbf{a}\right]$$

and

$$\mathtt{B} = \left[\partial\hat{\mathbf{X}}/\partial\mathbf{b}\right].$$

The set of equations $\mathtt{J}\boldsymbol{\delta} = \boldsymbol{\epsilon}$ solved as the central step in the LM algorithm (see section A6.2) now has the form

$$\mathtt{J}\boldsymbol{\delta} = [\mathtt{A}|\mathtt{B}]\begin{pmatrix}\boldsymbol{\delta}_\mathbf{a}\\ \hline \boldsymbol{\delta}_\mathbf{b}\end{pmatrix} = \boldsymbol{\epsilon}. \tag{A6.4}$$

Then, the normal equations $\mathtt{J}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{J}\boldsymbol{\delta} = \mathtt{J}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\boldsymbol{\epsilon}$ to be solved at each step of the LM algorithm are of the form

$$\left[\begin{array}{c|c}\mathtt{A}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{A} & \mathtt{A}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{B}\\ \hline \mathtt{B}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{A} & \mathtt{B}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{B}\end{array}\right]\begin{pmatrix}\boldsymbol{\delta}_\mathbf{a}\\ \hline \boldsymbol{\delta}_\mathbf{b}\end{pmatrix} = \begin{pmatrix}\mathtt{A}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\boldsymbol{\epsilon}\\ \hline \mathtt{B}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\boldsymbol{\epsilon}\end{pmatrix}. \tag{A6.5}$$

[1] In the absence of other knowledge, one usually assumes that $\Sigma_\mathbf{X}$ is the identity matrix.

At this point in the LM algorithm the diagonal blocks of this matrix are augmented by multiplying their diagonal entries by a factor $1 + \lambda$, for the varying parameter λ. This augmentation alters the matrices $\mathtt{A}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{A}$ and $\mathtt{B}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{B}$. The resulting matrices will be denoted by $(\mathtt{A}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{A})^*$ and $(\mathtt{B}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{B})^*$. The reader may wish to look ahead to figure A6.1 and figure A6.2 which shows graphically the form of the Jacobian and normal equations in an estimation problem involving several cameras and points.

The set of equations (A6.5) may now be written in block form as

$$\begin{bmatrix} \mathtt{U}^* & \mathtt{W} \\ \mathtt{W}^\mathsf{T} & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \boldsymbol{\delta}_\mathbf{a} \\ \boldsymbol{\delta}_\mathbf{b} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\epsilon}_\mathtt{A} \\ \boldsymbol{\epsilon}_\mathtt{B} \end{pmatrix}. \tag{A6.6}$$

As a first step to solving these equations, both sides are now multiplied on the left by $\begin{bmatrix} \mathtt{I} & -\mathtt{W}\mathtt{V}^{*-1} \\ 0 & \mathtt{I} \end{bmatrix}$ resulting in

$$\begin{bmatrix} \mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^\mathsf{T} & 0 \\ \mathtt{W}^\mathsf{T} & \mathtt{V}^* \end{bmatrix} \begin{pmatrix} \boldsymbol{\delta}_\mathbf{a} \\ \boldsymbol{\delta}_\mathbf{b} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\epsilon}_\mathtt{A} - \mathtt{W}\mathtt{V}^{*-1}\boldsymbol{\epsilon}_\mathtt{B} \\ \boldsymbol{\epsilon}_\mathtt{B} \end{pmatrix}. \tag{A6.7}$$

This results in the elimination of the top right hand block. The top half of this set of equations is

$$(\mathtt{U}^* - \mathtt{W}\mathtt{V}^{*-1}\mathtt{W}^\mathsf{T})\boldsymbol{\delta}_\mathbf{a} = \boldsymbol{\epsilon}_\mathtt{A} - \mathtt{W}\mathtt{V}^{*-1}\boldsymbol{\epsilon}_\mathtt{B}. \tag{A6.8}$$

These equations may be solved to find $\delta_\mathbf{a}$. Subsequently, the value of $\delta_\mathbf{b}$ may be found by back-substitution, giving

$$\mathtt{V}^*\delta_\mathbf{b} = \boldsymbol{\epsilon}_\mathtt{B} - \mathtt{W}^\mathsf{T}\boldsymbol{\delta}_\mathbf{a}. \tag{A6.9}$$

As in section A6.2, if the newly computed value of the parameter vector $\mathbf{P} = ((\mathbf{a} + \delta_\mathbf{a})^\mathsf{T}, (\mathbf{b} + \delta_\mathbf{b})^\mathsf{T})^\mathsf{T}$ results in a diminished value of the error function, then one accepts the new parameter vector $\mathbf{P}$, diminishes the value of λ by a factor of 10, and proceeds to the next iteration. On the other hand if the error value is increased, then one rejects the new $\mathbf{P}$ and tries again with a new value of λ, increased by a factor of 10.

The complete partitioned Levenberg–Marquardt algorithm is given in algorithm A6.1.

Although in this method we solve first for $\boldsymbol{\delta}_\mathbf{a}$ and subsequently for $\boldsymbol{\delta}_\mathbf{b}$ based on the new value of $\mathbf{a}$, it should not be thought that the method amounts to no more than independent iterations over $\mathbf{a}$ and $\mathbf{b}$. If one were to solve for $\mathbf{a}$ holding $\mathbf{b}$ constant, then the normal equations used to solve for $\boldsymbol{\delta}_\mathbf{a}$ would take the simpler form $\mathtt{U}\boldsymbol{\delta}_\mathbf{a} = \boldsymbol{\epsilon}_\mathtt{A}$, compared with (A6.8). The method of alternating between solving for $\boldsymbol{\delta}_\mathbf{a}$ and $\boldsymbol{\delta}_\mathbf{b}$ is not recommended, however, because of potential slow convergence.

A6.3.2 Covariance

It was seen in result 5.12(*p*144) that the covariance matrix of the estimated parameters is given by

$$\Sigma_\mathbf{P} = (\mathtt{J}^\mathsf{T}\Sigma_\mathbf{X}^{-1}\mathtt{J})^+. \tag{A6.10}$$

Given A vector of measurements $\mathbf{X}$ with covariance matrix $\Sigma_{\mathbf{X}}$, an initial estimate of a set of parameters $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T}$ and a function $f : \mathbf{P} \mapsto \widehat{\mathbf{X}}$ taking the parameter vector to an estimate of the measurement vector.

Objective Find the set of parameters $\mathbf{P}$ that minimizes $\boldsymbol{\epsilon}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\boldsymbol{\epsilon}$ where $\epsilon = \mathbf{X} - \widehat{\mathbf{X}}$.

Algorithm

(i) Initialize a constant $\lambda = 0.001$ (typical value).
(ii) Compute the derivative matrices $\mathtt{A} = [\partial\widehat{\mathbf{X}}/\partial\mathbf{a}]$ and $\mathtt{B} = [\partial\widehat{\mathbf{X}}/\partial\mathbf{b}]$ and the error vector $\boldsymbol{\epsilon}$.
(iii) Compute intermediate expressions

$$\mathtt{U} = \mathtt{A}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} \qquad \mathtt{V} = \mathtt{B}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \qquad \mathtt{W} = \mathtt{A}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{B}$$

$$\boldsymbol{\epsilon}_\mathtt{A} = \mathtt{A}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon}_\mathtt{B} = \mathtt{B}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\boldsymbol{\epsilon}$$

(iv) Augment $\mathtt{U}$ and $\mathtt{V}$ by multiplying their diagonal elements by $1 + \lambda$.
(v) Compute the inverse $\mathtt{V}^{*-1}$, and define $\mathtt{Y} = \mathtt{W}\mathtt{V}^{*-1}$. The inverse may overwrite the value of $\mathtt{V}^*$ which will not be needed again.
(vi) Find $\boldsymbol{\delta}_\mathbf{a}$ by solving $(\mathtt{U}^* - \mathtt{Y}\mathtt{W}^\mathsf{T})\boldsymbol{\delta}_\mathbf{a} = \boldsymbol{\epsilon}_\mathtt{A} - \mathtt{Y}\boldsymbol{\epsilon}_\mathtt{B}$.
(vii) Find $\boldsymbol{\delta}_\mathbf{b}$ by back-substitution: $\delta_\mathbf{b} = \mathtt{V}^{*-1}(\epsilon_\mathtt{B} - \mathtt{W}^\mathsf{T}\boldsymbol{\delta}_\mathbf{a})$.
(viii) Update the parameter vector by adding the incremental vector $(\boldsymbol{\delta}_\mathbf{a}^\mathsf{T}, \boldsymbol{\delta}_\mathbf{b}^\mathsf{T})^\mathsf{T}$ and compute the new error vector.
(ix) If the new error is less than the old error, then accept the new values of the parameters, diminish the value of λ by a factor of 10, and start again at step (ii), or else terminate.
(x) If the new error is greater than the old error, then revert to the old parameter values, increase the value of λ by a factor of 10, and try again from step (iv).

Algorithm A6.1. *A partitioned Levenberg–Marquardt algorithm.*

In the over-parametrized case, the covariance matrix $\Sigma_\mathbf{P}$ given by (A6.10) is singular, and in particular, no variation in the parameters is allowed in directions perpendicular to the constraint surface – the variance is zero in these directions.

In the case where the parameter set is partitioned as $\mathtt{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}^\mathsf{T})^\mathsf{T}$, matrix $(\mathtt{J}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{J})$ has the block form given in (A6.5) and (A6.6) (but without augmentation, represented by stars). Thus we may write

$$\mathtt{J}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{J} = \begin{bmatrix} \mathtt{A}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} & \mathtt{A}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \\ \mathtt{B}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{A} & \mathtt{B}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{B} \end{bmatrix} = \begin{bmatrix} \mathtt{U} & \mathtt{W} \\ \mathtt{W}^\mathsf{T} & \mathtt{V} \end{bmatrix}. \tag{A6.11}$$

The covariance matrix $\Sigma_\mathbf{P}$ is the pseudo-inverse of this matrix. Under the assumption that $\mathtt{V}$ is invertible, redefine $\mathtt{Y} = \mathtt{W}\mathtt{V}^{-1}$. Then the matrix may be diagonalized according to

$$\mathtt{J}^\mathsf{T}\Sigma_{\mathbf{X}}^{-1}\mathtt{J} = \begin{bmatrix} \mathtt{U} & \mathtt{W} \\ \mathtt{W}^\mathsf{T} & \mathtt{V} \end{bmatrix} = \begin{bmatrix} \mathtt{I} & \mathtt{Y} \\ 0 & \mathtt{I} \end{bmatrix} \begin{bmatrix} \mathtt{U} - \mathtt{W}\mathtt{V}^{-1}\mathtt{W}^\mathsf{T} & 0 \\ 0 & \mathtt{V} \end{bmatrix} \begin{bmatrix} \mathtt{I} & 0 \\ \mathtt{Y}^\mathsf{T} & \mathtt{I} \end{bmatrix} \tag{A6.12}$$

For matrices $\mathtt{G}$ and $\mathtt{H}$ with $\mathtt{G}$ invertible, we assume an identity

$$(\mathtt{G}\mathtt{H}\mathtt{G}^\mathsf{T})^+ = \mathtt{G}^{-\mathsf{T}}\mathtt{H}^+\mathtt{G}^{-1}$$

This identity is valid under conditions explored in the exercise at the end of this ap-

Objective Compute the covariance of the parameters $\mathbf{a}$ and $\mathbf{b}$ estimated using algorithm A6.1.
Algorithm

(i) Compute matrices $\mathtt{U}$, $\mathtt{V}$ and $\mathtt{W}$ as in algorithm A6.1, and also $\mathtt{Y} = \mathtt{W}\mathtt{V}^{-1}$.
(ii) $\Sigma_{\mathbf{a}} = (\mathtt{U} - \mathtt{W}\mathtt{V}^{-1}\mathtt{W}^{\mathsf{T}})^{+}$.
(iii) $\Sigma_{\mathbf{b}} = \mathtt{Y}^{\mathsf{T}}\Sigma_{\mathbf{a}}\mathtt{Y} + \mathtt{V}^{-1}$
(iv) The cross-covariance $\Sigma_{\mathbf{ab}} = -\Sigma_{\mathbf{a}}\mathtt{Y}$.

Algorithm A6.2. *Computation of the covariance matrix of the LM parameters.*

pendix. Applying this to (A6.12) and multiplying out provides a formula for the pseudo inverse

$$\Sigma_{\mathbf{P}} = (\mathtt{J}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{J})^{+} = \begin{bmatrix} \mathtt{X} & -\mathtt{X}\mathtt{Y} \\ -\mathtt{Y}^{\mathsf{T}}\mathtt{X} & \mathtt{Y}^{\mathsf{T}}\mathtt{X}\mathtt{Y} + \mathtt{V}^{-1} \end{bmatrix} \tag{A6.13}$$

where $\mathtt{X} = (\mathtt{U} - \mathtt{W}\mathtt{V}^{-1}\mathtt{W}^{\mathsf{T}})^{+}$.

The condition for this to be true is that $\text{span}(\mathtt{A}) \cap \text{span}(\mathtt{B}) = \emptyset$, where $\text{span}(\cdot)$ represents the span of the columns of the matrix. Here $\mathtt{A}$ and $\mathtt{B}$ are as in (A6.11). Proof of this fact and interpretation of the condition $\text{span}(\mathtt{A}) \cap \text{span}(\mathtt{B}) = \emptyset$ is outlined in exercise (i) on page 626.

The division of matrix $\mathtt{J}^{\mathsf{T}}\Sigma_{\mathbf{X}}^{-1}\mathtt{J}$ into blocks as in (A6.11) corresponds in (A6.5) to the division of $\mathbf{P}$ into parameters $\mathbf{a}$ and $\mathbf{b}$. Truncation of the covariance matrix for parameters $\mathbf{P}$ gives covariance matrices for parameters $\mathbf{a}$ and $\mathbf{b}$ separately. The result is summarized in algorithm A6.2.

A6.3.3 General sparse LM method

In the previous pages, a method has been described for carrying out LM iteration and computing covariances of the solution in the case where the parameter vector may be partitioned into two sub-vectors $\mathbf{a}$ and $\mathbf{b}$. It is not clear from the foregoing discussion that the methods described there actually give any computational advantage in the general case. However, such methods become important when the Jacobian matrix obeys a certain sparseness condition, as will be described now.

We suppose that the "measurement vector" $\mathbf{X} \in \mathbb{R}^N$ may broken up into pieces as $\mathbf{X} = (\mathbf{X}_1^{\mathsf{T}}, \mathbf{X}_2^{\mathsf{T}}, \ldots, \mathbf{X}_n^{\mathsf{T}})^{\mathsf{T}}$. Similarly, suppose that the "parameter vector" $\mathbf{P} \in \mathbb{R}^M$ may be divided up as $\mathbf{P} = (\mathbf{a}^{\mathsf{T}}, \mathbf{b}_1^{\mathsf{T}}, \mathbf{b}_2^{\mathsf{T}}, \ldots, \mathbf{b}_n^{\mathsf{T}})^{\mathsf{T}}$. The estimated values of $\mathbf{X}_i$ corresponding to a given assignment of parameters will be denoted by $\hat{\mathbf{X}}_i$. We make the *sparseness assumption* that each $\hat{\mathbf{X}}_i$ is dependent on $\mathbf{a}$ and $\mathbf{b}_i$ only, but not on the other parameters $\mathbf{b}_j$. In this case, $\partial\hat{\mathbf{X}}_i/\partial\mathbf{b}_j = \mathtt{0}$ for $i \neq j$. No assumption is made about $\partial\hat{\mathbf{X}}_i/\partial\mathbf{a}$. This situation arises in the reconstruction problem described at the start of this discussion, in which $\mathbf{b}_i$ is the vector of parameters of the i-th point, and $\hat{\mathbf{X}}_i$ is the vector of measurements of the image of this point in all the views. In this case, since the image of a point does not depend on any other point, one sees that $\partial\hat{\mathbf{X}}_i/\partial\mathbf{b}_j = 0$, as required, unless $i = j$.

Corresponding to this division, the Jacobian matrix $\mathtt{J} = [\partial\hat{\mathbf{X}}/\partial\mathbf{P}]$ has a sparse block

structure. We define Jacobian matrices by

$$\mathtt{A}_i = \left[\partial\widehat{\mathbf{X}}_i/\partial\mathbf{a}\right] \qquad \mathtt{B}_i = \left[\partial\widehat{\mathbf{X}}_i/\partial\mathbf{b}_i\right].$$

Given an error vector of the form $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}_1^\mathsf{T}, \ldots, \boldsymbol{\epsilon}_n^\mathsf{T})^\mathsf{T} = \mathbf{X} - \widehat{\mathbf{X}}$ the set of equations $\mathtt{J}\boldsymbol{\delta} = \boldsymbol{\epsilon}$ now has the form

$$\mathtt{J}\boldsymbol{\delta} = \left[\begin{array}{c|ccc} \mathtt{A}_1 & \mathtt{B}_1 & & \\ \mathtt{A}_2 & & \mathtt{B}_2 & \\ \vdots & & & \ddots \\ \mathtt{A}_n & & & \mathtt{B}_n \end{array}\right] \left(\begin{array}{c} \boldsymbol{\delta}_\mathbf{a} \\ \hline \boldsymbol{\delta}_{\mathbf{b}_1} \\ \vdots \\ \boldsymbol{\delta}_{\mathbf{b}_n} \end{array}\right) = \begin{pmatrix} \boldsymbol{\epsilon}_1 \\ \vdots \\ \boldsymbol{\epsilon}_n \end{pmatrix}. \tag{A6.14}$$

We suppose further that all of the measurements $\mathbf{X}_i$ are independent with covariance matrices $\Sigma_{\mathbf{X}_i}$. Thus the covariance matrix for the complete measurement vector $\Sigma_\mathbf{X}$ has the block-diagonal form $\Sigma_\mathbf{X} = \mathrm{diag}(\Sigma_{\mathbf{X}_1}, \ldots, \Sigma_{\mathbf{X}_n})$.

In the notation of algorithm A6.1, we have

$$\begin{aligned} \mathtt{A} &= [\mathtt{A}_1^\mathsf{T}, \mathtt{A}_2^\mathsf{T}, \ldots, \mathtt{A}_n^\mathsf{T}]^\mathsf{T} \\ \mathtt{B} &= \mathrm{diag}(\mathtt{B}_1, \mathtt{B}_2, \ldots, \mathtt{B}_n) \\ \Sigma_\mathbf{X} &= \mathrm{diag}(\Sigma_{\mathbf{X}_1}, \ldots, \Sigma_{\mathbf{X}_n}) \\ \boldsymbol{\delta}_\mathbf{b} &= (\boldsymbol{\delta}_{\mathbf{b}_1}^\mathsf{T}, \boldsymbol{\delta}_{\mathbf{b}_2}^\mathsf{T}, \ldots, \boldsymbol{\delta}_{\mathbf{b}_n}^\mathsf{T})^\mathsf{T} \\ \boldsymbol{\epsilon} &= (\boldsymbol{\epsilon}_1^\mathsf{T}, \boldsymbol{\epsilon}_2^\mathsf{T}, \ldots, \boldsymbol{\epsilon}_n^\mathsf{T})^\mathsf{T} \end{aligned}$$

Now, it is a straightforward task to substitute these formulae into algorithm A6.1. The result of this substitution is given in algorithm A6.3, representing the computation of one step of the LM algorithm. The important observation is that in this form each step of the algorithm requires computation time linear in n. Without the advantage resulting from the sparse structure, the algorithm (for instance a blind adherence to algorithm A6.1) would have complexity of order n^3.

A6.4 Application of sparse LM to 2D homography estimation

We apply the foregoing discussion to the estimation of a 2D homography $\mathtt{H}$ given a set of corresponding image points $\mathbf{x}_i \leftrightarrow \mathbf{x}_i'$ in two images. The points in each image are subject to noise, and one seeks to minimize the cost function (4.8–*p*95). We define a measurement vector $\mathbf{X}_i = (\mathbf{x}_i^\mathsf{T}, \mathbf{x}_i'^\mathsf{T})^\mathsf{T}$. In this case, the parameter vector $\mathbf{P}$ may be divided up as $\mathbf{P} = (\mathbf{h}^\mathsf{T}, \hat{\mathbf{x}}_1^\mathsf{T}, \hat{\mathbf{x}}_2^\mathsf{T}, \ldots, \hat{\mathbf{x}}_n^\mathsf{T})^\mathsf{T}$, where the values $\hat{\mathbf{x}}_i$ are the estimated values of the image points in the first image, and $\mathbf{h}$ is a vector of entries of the homography $\mathtt{H}$. Thus, one must simultaneously estimate the homography $\mathtt{H}$ and the parameters of each point in the first image. The function f maps $\mathbf{P}$ to $(\widehat{\mathbf{X}}_1^\mathsf{T}, \widehat{\mathbf{X}}_2^\mathsf{T}, \ldots, \widehat{\mathbf{X}}_n^\mathsf{T})^\mathsf{T}$, where each $\widehat{\mathbf{X}}_i = (\hat{\mathbf{x}}_i^\mathsf{T}, \mathtt{H}\hat{\mathbf{x}}_i^\mathsf{T})^\mathsf{T} = (\hat{\mathbf{x}}_i^\mathsf{T}, \hat{\mathbf{x}}_i'^\mathsf{T})^\mathsf{T}$. Then algorithm A6.3 applies directly.

The Jacobian matrices have a special form in this case, since one observes that

$$\mathtt{A}_i = \partial\widehat{\mathbf{X}}_i/\partial\mathbf{h} = \begin{bmatrix} 0 \\ \partial\hat{\mathbf{x}}_i'/\partial\mathbf{h} \end{bmatrix}$$

Objective Formulate LM algorithm in the case where the parameter vector is partitioned as $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}_1^\mathsf{T}, \ldots, \mathbf{b}_n^\mathsf{T})^\mathsf{T}$, and the measurement vector as $\mathbf{X} = (\mathbf{X}_1^\mathsf{T}, \ldots, \mathbf{X}_n^\mathsf{T})^\mathsf{T}$, such that $\partial\widehat{\mathbf{X}}_i/\partial\mathbf{b}_j = 0$ for $i \neq j$.

Algorithm Steps (ii) to (vii) of algorithm A6.1 become:

(i) Compute the derivative matrices $\mathtt{A}_i = [\partial\widehat{\mathbf{X}}_i/\partial\mathbf{a}]$ and $\mathtt{B}_i = [\partial\widehat{\mathbf{X}}_i/\partial\mathbf{b}_i]$ and the error vectors $\boldsymbol{\epsilon}_i = \mathbf{X}_i - \widehat{\mathbf{X}}_i$.

(ii) Compute the intermediate values

$$\begin{aligned}
\mathtt{U} &= \sum_i \mathtt{A}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\mathtt{A}_i \\
\mathtt{V} &= \mathrm{diag}(\mathtt{V}_1, \ldots, \mathtt{V}_n) \text{ where } \mathtt{V}_i = \mathtt{B}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\mathtt{B}_i \\
\mathtt{W} &= [\mathtt{W}_1, \mathtt{W}_2, \ldots, \mathtt{W}_n] \text{ where } \mathtt{W}_i = \mathtt{A}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\mathtt{B}_i \\
\boldsymbol{\epsilon}_\mathtt{A} &= \sum_i \mathtt{A}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\boldsymbol{\epsilon}_i \\
\boldsymbol{\epsilon}_\mathtt{B} &= (\boldsymbol{\epsilon}_{\mathtt{B}_1}^\mathsf{T}, \boldsymbol{\epsilon}_{\mathtt{B}_2}^\mathsf{T}, \ldots, \boldsymbol{\epsilon}_{\mathtt{B}_n}^\mathsf{T})^\mathsf{T} \text{ where } \boldsymbol{\epsilon}_{\mathtt{B}_i} = \mathtt{B}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\boldsymbol{\epsilon}_i \\
\mathtt{Y}_i &= \mathtt{W}_i\mathtt{V}_i^{*-1}.
\end{aligned}$$

(iii) Compute $\boldsymbol{\delta}_\mathbf{a}$ from the equation

$$(\mathtt{U}^* - \sum_i \mathtt{Y}_i\mathtt{W}_i^\mathsf{T})\delta_\mathbf{a} = \epsilon_\mathtt{A} - \sum_i \mathtt{Y}_i\boldsymbol{\epsilon}_{\mathtt{B}_i}.$$

(iv) Compute each $\boldsymbol{\delta}_{\mathbf{b}_i}$ in turn from the equation

$$\delta_{\mathbf{b}_i} = \mathtt{V}_i^{*-1}(\boldsymbol{\epsilon}_{\mathtt{B}_i} - \mathtt{W}_i^\mathsf{T}\boldsymbol{\delta}_\mathbf{a}).$$

Covariance

(i) Redefine $\mathtt{Y}_i = \mathtt{W}_i\mathtt{V}_i^{-1}$
(ii) $\Sigma_\mathbf{a} = (\mathtt{U} - \sum_i \mathtt{Y}_i\mathtt{W}_i^\mathsf{T})^+$
(iii) $\Sigma_{\mathbf{b}_i\mathbf{b}_j} = \mathtt{Y}_i^\mathsf{T}\Sigma_\mathbf{a}\mathtt{Y}_j + \delta_{ij}\mathtt{V}_i^{-1}$
(iv) $\Sigma_{\mathbf{a}\mathbf{b}_i} = -\Sigma_\mathbf{a}\mathtt{Y}_i$

Algorithm A6.3. *A sparse Levenberg–Marquardt algorithm.*

since $\hat{\mathbf{x}}_i$ is independent of $\mathbf{h}$. Also,

$$\mathtt{B}_i = \partial\widehat{\mathbf{X}}_i/\partial\hat{\mathbf{x}}_i = \begin{bmatrix} \mathtt{I} \\ \partial\hat{\mathbf{x}}_i'/\partial\hat{\mathbf{x}}_i \end{bmatrix}$$

because of the form of $\widehat{\mathbf{X}}_i = (\hat{\mathbf{x}}_i^\mathsf{T}, \hat{\mathbf{x}}_i'^\mathsf{T})^\mathsf{T}$.

A6.4.1 Computation of the covariance

As an example of covariance computation, we consider the same problem as in section 5.2.4(*p*145), in which a homography is estimated from point correspondences. As in that example, we consider a case in which the estimated homography is actually the identity mapping, $\mathtt{H} = \mathtt{I}$. For the purposes of this example, the number of points or their distribution is not important. It will be assumed, however, that the errors of all

point measurements are independent. We recall from (5.12–*p*146) that in the case of errors in the second image only, $\Sigma_{\mathbf{h}} = \left(\sum_i \mathtt{J}_i^\mathsf{T} \Sigma_i^{-1} \mathtt{J}_i\right)^+$, where $\mathtt{J}_i = [\partial \hat{\mathbf{x}}_i' / \partial \mathbf{h}]$.

We now proceed to compute the covariance of the camera parameter vector $\mathbf{h}$ in the case where the points in the first image are subject to noise as well. We assume further that $\Sigma_{\mathbf{x}_i}^{-1} = \Sigma_{\mathbf{x}_i'}^{-1}$, and denote them by $\mathtt{S}_i$. In this case, the inverse covariance matrix $\Sigma_{\mathbf{X}}^{-1}$ is block-diagonal, $\Sigma_{\mathbf{X}}^{-1} = \mathrm{diag}(\Sigma_{\mathbf{x}_i}^{-1}, \Sigma_{\mathbf{x}_i'}^{-1})$. Then, applying the steps of algorithm A6.3 to compute the covariance matrix for $\mathbf{h}$,

$$\begin{aligned}
\mathtt{A}_i &= [\mathtt{0}^\mathsf{T}, \mathtt{J}_i^\mathsf{T}]^\mathsf{T} \\
\mathtt{B}_i &= [\mathtt{I}^\mathsf{T}, \mathtt{I}^\mathsf{T}]^\mathsf{T} \text{ since } \mathtt{H} = \mathtt{I} \\
\mathtt{U} &= \sum_i \mathtt{A}_i^\mathsf{T} \mathrm{diag}(\mathtt{S}_i, \mathtt{S}_i) \mathtt{A}_i = \sum_i \mathtt{J}_i^\mathsf{T} \mathtt{S}_i \mathtt{J}_i \\
\mathtt{V}_i &= \mathtt{B}_i^\mathsf{T} \mathrm{diag}(\mathtt{S}_i, \mathtt{S}_i) \mathtt{B}_i = 2\mathtt{S}_i \\
\mathtt{W}_i &= \mathtt{A}_i^\mathsf{T} \mathrm{diag}(\mathtt{S}_i, \mathtt{S}_i) \mathtt{B}_i = \mathtt{J}_i^\mathsf{T} \mathtt{S}_i \\
\mathtt{U} - \sum_i \mathtt{W}_i \mathtt{V}_i^{-1} \mathtt{W}_i^\mathsf{T} &= \sum_i \mathtt{J}_i^\mathsf{T} (\mathtt{S}_i - \mathtt{S}_i/2) \mathtt{J}_i = \sum_i \mathtt{J}_i^\mathsf{T} \mathtt{S}_i \mathtt{J}_i / 2 \\
\Sigma_{\mathbf{h}} &= 2 \left(\sum_i \mathtt{J}_i^\mathsf{T} \mathtt{S}_i \mathtt{J}_i \right)^+ .
\end{aligned}$$

Hence, the covariance matrix for $\mathbf{h}$ is just twice the value of the covariance matrix for the case of error in one image. This is not generally true, and results from the fact that $\mathtt{H}$ is the identity mapping. As exercises, one may verify the following.

- *If* $\mathtt{H}$ *represents a scaling by a factor* s*, then* $\Sigma_{\mathbf{h}} = (s^2 + 1) \left(\sum_i \mathtt{J}_i^\mathsf{T} \mathtt{S}_i \mathtt{J}_i\right)^+$.
- *If* $\mathtt{H}$ *is an affine transformation and* $\mathtt{D}$ *is the upper* 2×2 *part of* $\mathtt{H}$ *(the non-translational part), and if* $\mathtt{S}_i = \mathtt{I}$ *for all* i *(isotropic and independent noise), then* $\Sigma_{\mathbf{h}} = \left(\sum_i \mathtt{J}_i^\mathsf{T} \left(\mathtt{I} - \mathtt{D}(\mathtt{I} + \mathtt{D}^\mathsf{T}\mathtt{D})^{-1}\mathtt{D}^\mathsf{T}\right) \mathtt{J}_i\right)^+$.

A6.5 Application of sparse LM to fundamental matrix estimation

In estimating the fundamental matrix and a set of 3D points, the algorithm is effectively that described in section A6.3.3, a sparse LM algorithm for the estimation of 2D homographies, but slightly modified to the present case. The analogy with the 2D homography estimation problem is as follows: in estimating 2D homographies, one has a mapping $\mathtt{H}$ that takes points $\mathbf{x}_i$ to corresponding points $\mathbf{x}_i'$; in the present problem, one has a mapping represented by a pair of camera matrices $\mathtt{P}$ and $\mathtt{P}'$ that map a 3D point to a pair of corresponding points $(\mathbf{x}_i, \mathbf{x}_i')$.

For convenience, the notation of section A6.3.3 will be used here. In particular, note that in section A6.3.3, and here the notation $\mathbf{X}$ is used to denote the total measurement vector (in this case $(\mathbf{x}_1, \mathbf{x}_1', \ldots, \mathbf{x}_n, \mathbf{x}_n')$) and not a 3D point. Also, be careful to distinguish between $\mathbf{P}$ the parameter vector and $\mathtt{P}$ the camera matrix.

The parameter vector $\mathbf{P}$ is partitioned as $\mathbf{P} = (\mathbf{a}^\mathsf{T}, \mathbf{b}_1^\mathsf{T}, \ldots, \mathbf{b}_n^\mathsf{T})^\mathsf{T}$, where

(i) $\mathbf{a} = \mathbf{p}'$ is made up of the entries of camera matrix $\mathtt{P}'$, and

(ii) $\mathbf{b}_i = (\mathtt{X}_i, \mathtt{Y}_i, \mathtt{T}_i)^\mathsf{T}$ is a 3-vector, parametrizing the i-th 3D point $(\mathtt{X}_i, \mathtt{Y}_i, 1, \mathtt{T}_i)$.

Thus, there are a total of $3n+12$ parameters, where n is the number of points. Parameter vector $\mathbf{a}$ provides a parametrization for the camera $\mathtt{P}'$ and the other camera $\mathtt{P}$ is taken as $[\mathtt{I} \mid \mathbf{0}]$. Note also that it is convenient and permissible to set the third coordinate of the 3D space point to 1 as here, since a point $(\mathtt{X}_i, \mathtt{Y}_i, 0, \mathtt{T}_i)^\mathsf{T}$ maps to $(\mathtt{X}_i, \mathtt{Y}_i, 0)^\mathsf{T}$ which is an infinite point, not anywhere close to the measured point $(x_i, y_i, 1)^\mathsf{T}$.

The measurement vector $\mathbf{X}$ is partitioned as $\mathbf{X} = (\mathbf{X}_1^\mathsf{T}, \mathbf{X}_2^\mathsf{T}, \ldots, \mathbf{X}_n^\mathsf{T})^\mathsf{T}$, where each $\mathbf{X}_i = (\mathbf{x}_i^\mathsf{T}, \mathbf{x}_i'^\mathsf{T})^\mathsf{T}$, the measured images of the i-th point.

Now, the Jacobian matrices $\mathtt{A}_i = \partial\hat{\mathbf{X}}_i/\partial\mathbf{a}$ and $\mathtt{B}_i = \partial\hat{\mathbf{X}}_i/\partial\mathbf{b}_i$ may be computed and algorithm A6.3 applied to estimate the parameters, and hence $\mathtt{P}'$, from which $\mathtt{F}$ may be computed.

Partial derivative matrices

Since $\hat{\mathbf{X}}_i = (\hat{\mathbf{x}}_i^\mathsf{T}, \hat{\mathbf{x}}_i'^\mathsf{T})^\mathsf{T}$ one may compute that $\mathtt{A}_i$ and $\mathtt{B}_i$ have a form similar to the Jacobian matrices in section A6.4:

$$\mathtt{A}_i = \begin{bmatrix} 0 \\ \partial\hat{\mathbf{x}}_i'/\partial\mathbf{a} \end{bmatrix} \qquad \mathtt{B}_i = \begin{bmatrix} \mathtt{I}_{2\times 2}|\mathbf{0} \\ \partial\hat{\mathbf{x}}_i'/\partial\mathbf{b}_i \end{bmatrix}.$$

The covariance matrix $\Sigma_{\mathbf{X}_i}$ breaks up into diagonal blocks $\mathrm{diag}(\mathtt{S}_i, \mathtt{S}_i')$, where $\mathtt{S}_i = \Sigma_{\mathbf{x}_i}^{-1}$ and $\mathtt{S}_i' = \Sigma_{\mathbf{x}_i'}^{-1}$. Now, calculating the intermediate expressions in step 2 of algorithm A6.3, we find

$$\mathtt{V}_i = \mathtt{B}_i^\mathsf{T}\mathrm{diag}(\mathtt{S}_i, \mathtt{S}_i')\mathtt{B}_i = [\mathtt{I}_{2\times 2} \mid \mathbf{0}]^\mathsf{T}\mathtt{S}_i[\mathtt{I}_{2\times 2} \mid \mathbf{0}] + (\partial\hat{\mathbf{x}}_i'/\partial\mathbf{b})^\mathsf{T}\mathtt{S}_i'(\partial\hat{\mathbf{x}}_i'/\partial\mathbf{b}) \quad \text{(A6.15)}$$

The abstract form of $\mathtt{A}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\mathtt{A}_i$ is the same as in the 2D homography case, and the other expressions $\mathtt{W}_i = \mathtt{A}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\mathtt{B}_i$, $\boldsymbol{\epsilon}_{\mathtt{B}_i} = \mathtt{B}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\boldsymbol{\epsilon}_i$, and $\boldsymbol{\epsilon}_{\mathtt{A}_i} = \mathtt{A}_i^\mathsf{T}\Sigma_{\mathbf{X}_i}^{-1}\boldsymbol{\epsilon}_i$ may easily be computed. The estimation procedure otherwise proceeds precisely as in algorithm A6.3.

Covariance of F

According to the discussion of section A6.3.3 and in particular algorithm A6.3, the covariance matrix of the camera parameters, namely the entries of $\mathtt{P}'$, is given by

$$\Sigma_{\mathtt{P}'} = (\mathtt{U} - \sum_i \mathtt{W}_i\mathtt{V}_i^{-1}\mathtt{W}_i^\mathsf{T})^+ \quad \text{(A6.16)}$$

with notation as in algorithm A6.3.

In computing this pseudo-inverse, it is useful to know the expected rank of $\Sigma_{\mathtt{P}'}$. In this case, this rank is 7, since the total number of degrees of freedom in the solution involving two cameras and n point matches is $3n + 7$. Looked at another way, $\mathtt{P}'$ is not determined uniquely, since if $\mathtt{P}' = [\mathtt{M} \mid \mathbf{m}]$, then any other matrix $[\mathtt{M} + \mathbf{t}\mathbf{m}^\mathsf{T} \mid \alpha\mathbf{m}]$ also determines the same fundamental matrix. Thus, in computing the pseudo-inverse appearing in the right hand side of (A6.16), one should set five of the singular values to zero.

The foregoing discussion shows how to compute the covariance matrix of the entries of $\mathtt{P}'$. We desire to compute the covariance matrix of the entries of $\mathtt{F}$. However, there is a simple formula for the entries of $\mathtt{F}$ in terms of the entries of $\mathtt{P}' = [\mathtt{M} \mid \mathbf{m}]$, namely

$\mathtt{F} = [\mathbf{m}]_\times \mathtt{M}$. If one desires to compute the covariance matrix of $\mathtt{F}$ normalized so that $\|\mathtt{F}\| = 1$, then one writes $\mathtt{F} = [\mathbf{m}]_\times \mathtt{M}/(\|[\mathbf{m}]_\times \mathtt{M}\|)$. Therefore, one may express the entries of $\mathtt{F}$ as a simple function in terms of the entries of $\mathtt{P}'$. Let $\mathtt{J}$ be the Jacobian matrix of this function. The covariance of $\mathtt{F}$ is then computed by propagating the covariance of $\mathtt{P}'$ using result 5.6(p139) to get

$$\Sigma_{\mathtt{F}} = \mathtt{J}\Sigma_{\mathtt{P}'}\mathtt{J}^\mathsf{T} = \mathtt{J}(\mathtt{U} - \sum_i \mathtt{W}_i \mathtt{V}_i^{-1} \mathtt{W}_i^\mathsf{T})^+ \mathtt{J}^\mathsf{T} \tag{A6.17}$$

where $\Sigma_{\mathtt{P}'}$ is given by (A6.16). This is the covariance of the fundamental matrix *estimated according to an ML algorithm* from the given point correspondences.

A6.6 Application of sparse LM to multiple image bundle adjustment

In the previous section, the application of the sparse Levenberg–Marquardt algorithm to the computation of the fundamental matrix was considered. This is essentially a reconstruction problem from two views. It should be clear how this may easily be extended to the computation of the trifocal tensor and the quadrifocal tensor. More generally, one may apply it to the simultaneous estimation of multiple camera and points to compute projective structure, or perhaps affine or metric structure given appropriate constraints. This technique is called *bundle adjustment*.

In the case of multiple cameras, one may also take advantage of the lack of interaction between parameters of the different cameras, as will be shown now. In the following discussion, for simplicity of notation it will be assumed that each point is visible in all the views. This is not at all necessary – points may in general be visible in some arbitrary subset of the available views.

We use the same notation as in section A6.3.3. The measurement data may be expressed as a vector $\mathbf{X}$, which may be divided up into parts $\mathbf{X}_i$, representing the measured image coordinates of some 3D point in all views. One may further subdivide $\mathbf{X}_i$ writing $\mathbf{X}_i = (\mathbf{x}_{i1}^\mathsf{T}, \mathbf{x}_{i2}^\mathsf{T}, \ldots, \mathbf{x}_{im}^\mathsf{T})^\mathsf{T}$ where $\mathbf{x}_{ij}$ is the image of the i-th point in the j-th image. The parameter vector $\mathbf{a}$ (camera parameters) may correspondingly be partitioned as $\mathbf{a} = (\mathbf{a}_1^\mathsf{T}, \mathbf{a}_2^\mathsf{T}, \ldots, \mathbf{a}_m^\mathsf{T})^\mathsf{T}$, where $\mathbf{a}_j$ are the parameters of the j-th camera. Since the image point $\mathbf{x}_{ij}$ does not depend on the parameters of any but the j-th camera, one observes that $\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{a}_k = 0$ unless $j = k$. In a similar way for derivatives with respect to the parameters $\mathbf{b}_k$ of the k-th 3D point, one has $\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{b}_k = 0$ unless $i = k$.

The form of the Jacobian matrix $\mathtt{J}$ for this problem and the resulting normal equations $\mathtt{J}^\mathsf{T}\mathtt{J}\boldsymbol{\delta} = \mathtt{J}^\mathsf{T}\boldsymbol{\epsilon}$ are shown schematically in figure A6.1 and figure A6.2. Referring to the Jacobian matrices defined in algorithm A6.3, one sees that $\mathtt{A}_i = [\partial\hat{\mathbf{X}}_i/\partial\mathbf{a}]$ is a block diagonal matrix $\mathtt{A}_i = \mathrm{diag}(\mathtt{A}_{i1}, \ldots, \mathtt{A}_{im})$, where $\mathtt{A}_{ij} = \partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{a}_j$. Similarly, matrix $\mathtt{B}_i = [\partial\hat{\mathbf{X}}_i/\partial\mathbf{b}_i]$ decomposes as $\mathtt{B}_i = [\mathtt{B}_{i1}^\mathsf{T}, \ldots, \mathtt{B}_{im}^\mathsf{T}]^\mathsf{T}$, where $\mathtt{B}_{ij} = \partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{b}_i$. It may normally be assumed also that $\Sigma_{\mathbf{X}_i}$ has a diagonal structure $\Sigma_{\mathbf{X}_i} = \mathrm{diag}(\Sigma_{\mathbf{x}_{i1}}, \ldots, \Sigma_{\mathbf{x}_{im}})$, meaning that the measurements of the projected points in separate images are independent (or more precisely, uncorrelated). With these assumptions, one is easily able to adapt algorithm A6.3, as shown in algorithm A6.4, as the reader is left to verify.

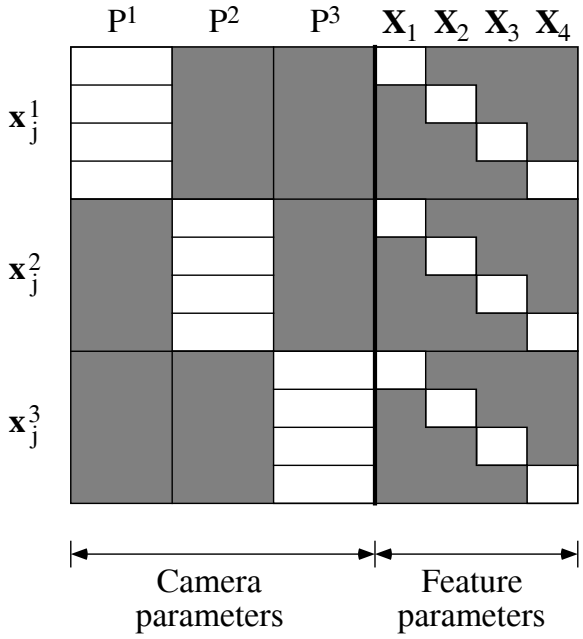

Fig. A6.1. *Form of the Jacobian matrix for a bundle-adjustment problem consisting of* 3 *cameras and* 4 *points.*

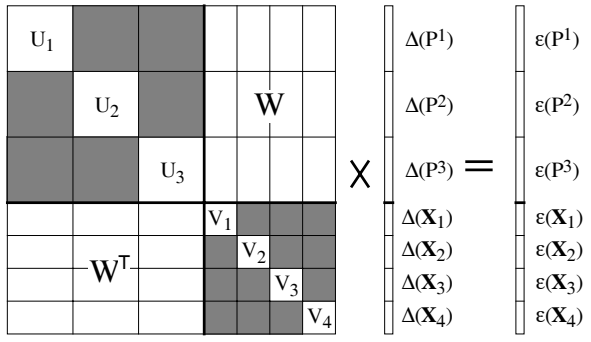

Fig. A6.2. *Form of the normal equations for the bundle-adjustment problem consisting of* 3 *cameras and* 4 *points.*

Missing data. Typically, in a bundle-adjustment problem some points are not visible in every image. Thus, some measurement $\mathbf{x}_{ij}$ may not exist, meaning that i-th point is not visible in the j-th image. Algorithm A6.4 is easily adapted to this situation, by ignoring terms subscripted by ij where the measurement $\mathbf{x}_{ij}$ does not exist. Such missing terms are simply omitted from the relevant summations. This includes all of $\mathtt{A}_{ij}$, $\mathtt{B}_{ij}$, $\Sigma^{-1}_{\mathbf{x}_{ij}}$, $\mathtt{W}_{ij}$ and $\mathtt{Y}_{ij}$. It may be seen that this is equivalent to setting $\mathtt{A}_{ij}$ and $\mathtt{B}_{ij}$ to zero, thus effectively giving zero weight to the missing measurements.

This is convenient when programming this algorithm, since the above quantities subscripted by ij may be associated only with existing measurements in a common data structure.

(i) Compute the derivative matrices $\mathtt{A}_{ij} = [\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{a}_j]$ and $\mathtt{B}_{ij} = [\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{b}_i]$ and the error vectors $\boldsymbol{\epsilon}_{ij} = \mathbf{x}_{ij} - \hat{\mathbf{x}}_{ij}$.
(ii) Compute the intermediate values

$$\mathtt{U}_j = \sum_i \mathtt{A}_{ij}^\mathsf{T}\Sigma_{\mathbf{x}_{ij}}^{-1}\mathtt{A}_{ij} \qquad \mathtt{V}_i = \sum_j \mathtt{B}_{ij}^\mathsf{T}\Sigma_{\mathbf{x}_{ij}}^{-1}\mathtt{B}_{ij} \qquad \mathtt{W}_{ij} = \mathtt{A}_{ij}^\mathsf{T}\Sigma_{\mathbf{x}_{ij}}^{-1}\mathtt{B}_{ij}$$

$$\boldsymbol{\epsilon}_{\mathbf{a}_j} = \sum_i \mathtt{A}_{ij}^\mathsf{T}\Sigma_{\mathbf{x}_{ij}}^{-1}\boldsymbol{\epsilon}_{ij} \qquad \boldsymbol{\epsilon}_{\mathbf{b}_i} = \sum_j \mathtt{B}_{ij}^\mathsf{T}\Sigma_{\mathbf{x}_{ij}}^{-1}\boldsymbol{\epsilon}_{ij} \qquad \mathtt{Y}_{ij} = \mathtt{W}_{ij}\mathtt{V}_i^{*-1}$$

where in each case $i = 1, \ldots, n$ and $j = 1, \ldots, m$.
(iii) Compute $\boldsymbol{\delta}_\mathbf{a} = (\boldsymbol{\delta}_{\mathbf{a}_1}^\mathsf{T}, \ldots, \boldsymbol{\delta}_{\mathbf{a}_m}^\mathsf{T})^\mathsf{T}$ from the equation

$$\mathtt{S}\delta_\mathbf{a} = (\mathbf{e}_1^\mathsf{T}, \ldots, \mathbf{e}_m^\mathsf{T})^\mathsf{T}$$

where $\mathtt{S}$ is an $m \times m$ block matrix with blocks $\mathtt{S}_{jk}$ defined by

$$\begin{aligned}\mathtt{S}_{jj} &= -\sum_i \mathtt{Y}_{ij}\mathtt{W}_{ij}^\mathsf{T} + \mathtt{U}_j^* \\ \mathtt{S}_{jk} &= -\sum_i \mathtt{Y}_{ij}\mathtt{W}_{ik}^\mathsf{T} \text{ if } j \neq k\end{aligned}$$

and

$$\mathbf{e}_j = \boldsymbol{\epsilon}_{\mathbf{a}_j} - \sum_i \mathtt{Y}_{ij}\boldsymbol{\epsilon}_{\mathbf{b}_i}$$

(iv) Compute each $\boldsymbol{\delta}_{\mathbf{b}_i}$ in turn from the equation

$$\delta_{\mathbf{b}_i} = \mathtt{V}_i^{*-1}(\boldsymbol{\epsilon}_{\mathbf{b}_i} - \sum_j \mathtt{W}_{ij}^\mathsf{T}\boldsymbol{\delta}_{\mathbf{a}_j})$$

Covariance

(i) Redefine $\mathtt{Y}_{ij} = \mathtt{W}_{ij}\mathtt{V}_i^{-1}$.
(ii) $\Sigma_\mathbf{a} = \mathtt{S}^+$, where $\mathtt{S}$ is defined as above, without the augmentation represented by the $*$.
(iii) $\Sigma_{\mathbf{b}_i\mathbf{b}_j} = \mathtt{Y}_i^\mathsf{T}\Sigma_\mathbf{a}\mathtt{Y}_j + \delta_{ij}\mathtt{V}_i^{-1}$.
(iv) $\Sigma_{\mathbf{a}\mathbf{b}_i} = -\Sigma_\mathbf{a}\mathtt{Y}_i$.

Algorithm A6.4. *General sparse Levenberg–Marquardt algorithm.*

A6.7 Sparse methods for equation solving

In a long sequence of images, it is rare for a point to be tracked through the whole sequence, and usually point tracks disappear and new ones start, causing the set of point tracks to have a banded structure, as seen in figure 19.7(*p*492)(c). This banded structure of the point track set leads to a banded structure for the set of equations that are solved to compute structure and motion – we refer here to the matrix $\mathtt{S}$ in algorithm A6.4. Thus, for bundle-adjustment problems with banded track structure, sparseness can appear at two levels, first at the level of independence of the individual point measurements, as exploited in algorithm A6.4, and secondly arising from the banded track structure as will be explained in this section.

Another similar context in which this will occur is solution for structure and motion

in section 18.5.1(*p*448) or simply motion, as in section 18.5.2(*p*450). In both these methods, large sparse sets of equations may arise. In order for large problems of this kind to be tractable, it is necessary to take advantage of this sparse structure in order to minimize the amount of storage and computation involved. In this section, we will consider sparse matrix techniques that are valuable in this context.

A6.7.1 Banded structure in bundle-adjustment

The time-consuming step in finding the parameter increments in the iterative step of bundle-adjustment, as given in algorithm A6.4 is the solution of the equations $\mathtt{S}\delta_a = \mathbf{e}$ in step (iii) of the algorithm. As shown there, the matrix $\mathtt{S}$ is a symmetric matrix with off-diagonal blocks of the form $\mathtt{S}_{jk} = -\sum_i \mathtt{W}_{ij}\mathtt{V}_i^{*-1}\mathtt{W}_{ik}^{\mathsf{T}}$. We see that the block $\mathtt{S}_{jk}$ is non-zero only when for some i, both $\mathtt{W}_{ij}$ and $\mathtt{W}_{ik}$ are non-zero. Since $\mathtt{W}_{ij} = [\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{a}_j]^{\mathsf{T}}\Sigma_{\mathbf{x}_{ij}}^{-1}[\partial\hat{\mathbf{x}}_{ij}/\partial\mathbf{b}_i]$ it follows that $\mathtt{W}_{ij}$ is non-zero only when the corresponding measurement $\hat{\mathbf{x}}_{ij}$ depends on parameters $\mathbf{a}_j$ and $\mathbf{b}_i$. To be more concrete, if $\mathbf{a}_j$ represents the parameters of the j-th camera, and $\mathbf{b}_i$ represents the parameters of the i-th point, then $\mathtt{W}_{ij}$ is non-zero only when the i-th point is visible in the j-th image, and $\mathbf{x}_{ij}$ is its measured image position.

The condition that for some i both $\mathtt{W}_{ij}$ and $\mathtt{W}_{ik}$ are non-zero means that there exists some point with index i that is visible in both the j-th and k-th images. To summarize,

- *The block $\mathtt{S}_{jk}$ is non-zero only if there exists a point that is visible in both the j-th and k-th images.*

Thus, if point tracks extend only over consecutive views, then the matrix $\mathtt{S}$ will be banded. In particular, if no point track extends over more than B views (B representing *bandwidth*), then the block $\mathtt{S}_{jk}$ is zero unless $|j-k| < B$.

Consider tracking points over a long sequence, for instance along a path that may loop back and cross itself. In this case, it may be possible to recognize a point that had been seen previously in the sequence, and pick up its track again. The set of views in which a 3D point is seen will not be a consecutive set of views. This will destroy the banded nature of the matrix $\mathtt{S}$, by introducing non-zero blocks possibly far away from the central band. Nevertheless, if there is not too much filling-in of off-diagonal blocks, sparse solution techniques may still be utilized as we shall see later.

A6.7.2 Solution of symmetric linear equations

In solving a set of linear equations $\mathtt{A}\mathbf{x} = \mathtt{b}$ in which the matrix $\mathtt{A}$ is symmetric, it is best not to use a general purpose equation solver, such as Gaussian-elimination, but rather to take advantage of the symmetry of the matrix $\mathtt{A}$. One way of doing this is the use the $\mathtt{LDL}^{\mathsf{T}}$ decomposition of the matrix $\mathtt{A}$. This relies on the following observation:

Result A6.1. *Any positive-definite symmetric matrix $\mathtt{A}$ can be factored as $\mathtt{A} = \mathtt{LDL}^{\mathsf{T}}$, in which $\mathtt{L}$ is a lower-triangular matrix with unit diagonal entries, and $\mathtt{D}$ is diagonal.*

The reader is advised to consult [Golub-89] for details of efficient implementation and numeric properties of $\mathtt{LDL}^{\mathsf{T}}$ decomposition. The normal equations derived from structure and motion problems are always at least positive semi-definite, and with stabiliza-

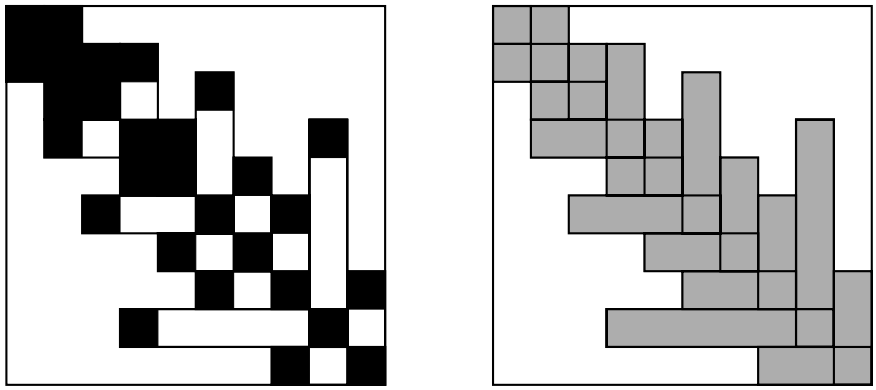

Fig. A6.3. *(a) A sparse matrix and (b) its corresponding "skyline" structure. Non-zero entries in the original matrix are shown in black. In the skyline format, all non-zero entries lie in the shaded area. For each row i, there is an integer m_i representing the first non-zero entry in that row. Note that non-zero off-diagonal elements cause "fill-in" of the skyline format for that row (or symmetric above-diagonal column). If such entries are relatively rare, the skyline format will remain sparse, and techniques for skyline matrices may be usefully applied. The* $\mathtt{LDL}^\mathsf{T}$ *decomposition of a matrix in skyline format has the same skyline structure. Thus, the* $\mathtt{LDL}^\mathsf{T}$ *decomposition of the original sparse matrix shown in (a), will have non-zero entries only within the shaded area of (b).*

tion through enhancement, they are positive-definite, and symmetric factorization is the recommended method of solution.

Given the $\mathtt{LDL}^\mathsf{T}$ factorization, the set of linear equations $\mathtt{A}\mathbf{x} = \mathtt{LDL}^\mathsf{T}\mathbf{x} = \mathbf{b}$ may be solved for $\mathbf{x}$ in three steps as (i) $\mathtt{L}\mathbf{x}' = \mathbf{b}$, (ii) $\mathbf{x}'' = \mathtt{D}^{-1}\mathbf{x}'$, (iii) $\mathtt{L}^\mathsf{T}\mathbf{x} = \mathbf{x}''$.

Solving the equations $\mathtt{L}\mathbf{x}' = \mathbf{b}$ is carried out by the process of "forward-substitution." In particular (bearing in mind that $\mathtt{L}$ has unit diagonal entries), the components of $\mathbf{x}$ may be computed in order as

$$\textit{forward-substitution:}\quad \mathbf{x}'_i = \mathbf{b}_i - \sum_{j=1}^{i-1} L_{ij}\mathbf{x}'_j.$$

Since $\mathtt{L}^\mathsf{T}$ is upper-triangular, the second set of equations is solved in a similar fashion, except that the values $\mathbf{x}_i$ are computed in inverse order in a process known as "back-substitution."

$$\textit{back-substitution:}\quad \mathbf{x}_i = \mathbf{x}''_i - \sum_{j=i+1}^{n} L_{ji}\mathbf{x}_j.$$

The number of operations involved in this computation is given in [Golub-89], and is equal to $n^3/3$, where n is the dimension of the matrix.

A6.7.3 Solution of sparse symmetric linear systems

We consider a special type of sparse structure for a symmetric matrix, known as "skyline" format. This is illustrated in figure A6.3. An $n \times n$ symmetric matrix $\mathtt{A}$ in skyline format is characterized by the existence of an array of integers m_i for $i = 1, \ldots, n$ such that $\mathtt{A}_{ij} = 0$ for $j < m_i$. A diagonally banded matrix is a special case of a matrix with skyline structure.

Although a diagonally banded or skyline matrix may be sparse, its inverse is not, and in fact will be completely filled out with non-zero elements. Thus, it is a very bad idea actually to compute the inverse of $\mathtt{A}$ to find the solution $\mathbf{x} = \mathtt{A}^{-1}\mathbf{b}$ to the set of equations. However, the importance of matrices in skyline (or banded) form is that skyline structure of the matrix *is preserved* by the $\mathtt{LDL}^\mathsf{T}$ decomposition, as expressed in the following result.

Result A6.2. *Let* $\mathtt{A}$ *be a symmetric matrix such that* $\mathtt{A}_{ij} = 0$ *for* $j < m_i$. *Let* $\mathtt{A} = \mathtt{LDL}^\mathsf{T}$. *Then* $\mathtt{L}_{ij} = 0$ *for* $j < m_i$.

In other words, the skyline structure of $\mathtt{L}$ is the same as that of $\mathtt{A}$.

Proof. Suppose that j is the smallest index such that $\mathtt{L}_{ij} = 0$. Then in multiplying out $\mathtt{A} = \mathtt{LDL}^\mathsf{T}$, it may be verified that only one product contributes to the (i, j)-th element of $\mathtt{A}_{ij}$. Specifically, $\mathtt{A}_{ij} = \mathtt{L}_{ij}\mathtt{D}_{jj}\mathtt{L}_{jj} \neq 0$. □

Thus, in computing the $\mathtt{LDL}^\mathsf{T}$ decomposition of $\mathtt{A}$ having a sparse skyline structure, we know in advance that many of the entries of $\mathtt{L}$ will be zero. The algorithm for computing the $\mathtt{LDL}^\mathsf{T}$ decomposition for such a matrix is much the same as the that for a full symmetric matrix, except that we do not need to consider the zero elements.

Forward and back substitution involving a matrix $\mathtt{L}$ with skyline structure easily take advantage of the sparse structure. In fact the forward substitution formula becomes

$$\mathbf{x}'_i = \mathbf{b}_i - \sum_{j=m_i}^{i-1} L_{ij}\mathbf{x}'_j.$$

Back-substitution is left for the reader to work out. More details of implementation are given in [Bathe-76].

A6.8 Robust cost functions

In estimation problems of the Newton or Levenberg-Marquardt type, an important decision to make is the precise form of the cost function. As we have seen, an assumption of Gaussian noise without outliers implies that the the Maximum Likelihood estimate is given by a least-squares cost function involving the predicted errors in the measurements, where the noise is introduced.

The same analysis may be carried out for other assumed probability models for the measurements. Thus, if all measurements are assumed to be independent, and $f(\delta)$ is the probability distribution of an error δ in the measurement, then the probability of a set of measurement with errors δ_i is given by $p(\delta_1, \ldots, \delta_n) = \prod_{i=1}^n f(\delta_i)$. Taking the negative logarithm gives $-\log(p(\delta_1, \ldots, \delta_n)) = -\sum_{i=1}^n \log(f(\delta_i))$ and the right-hand side of this expression is a suitable cost function for a set of measurements. It is usually appropriate to set the cost of an exact measurement to be zero, by subtracting $\log(f(0))$, though this is not strictly necessary if our purpose is cost minimization. Graphs of various specific cost functions to be discussed next are shown in figure A6.4.

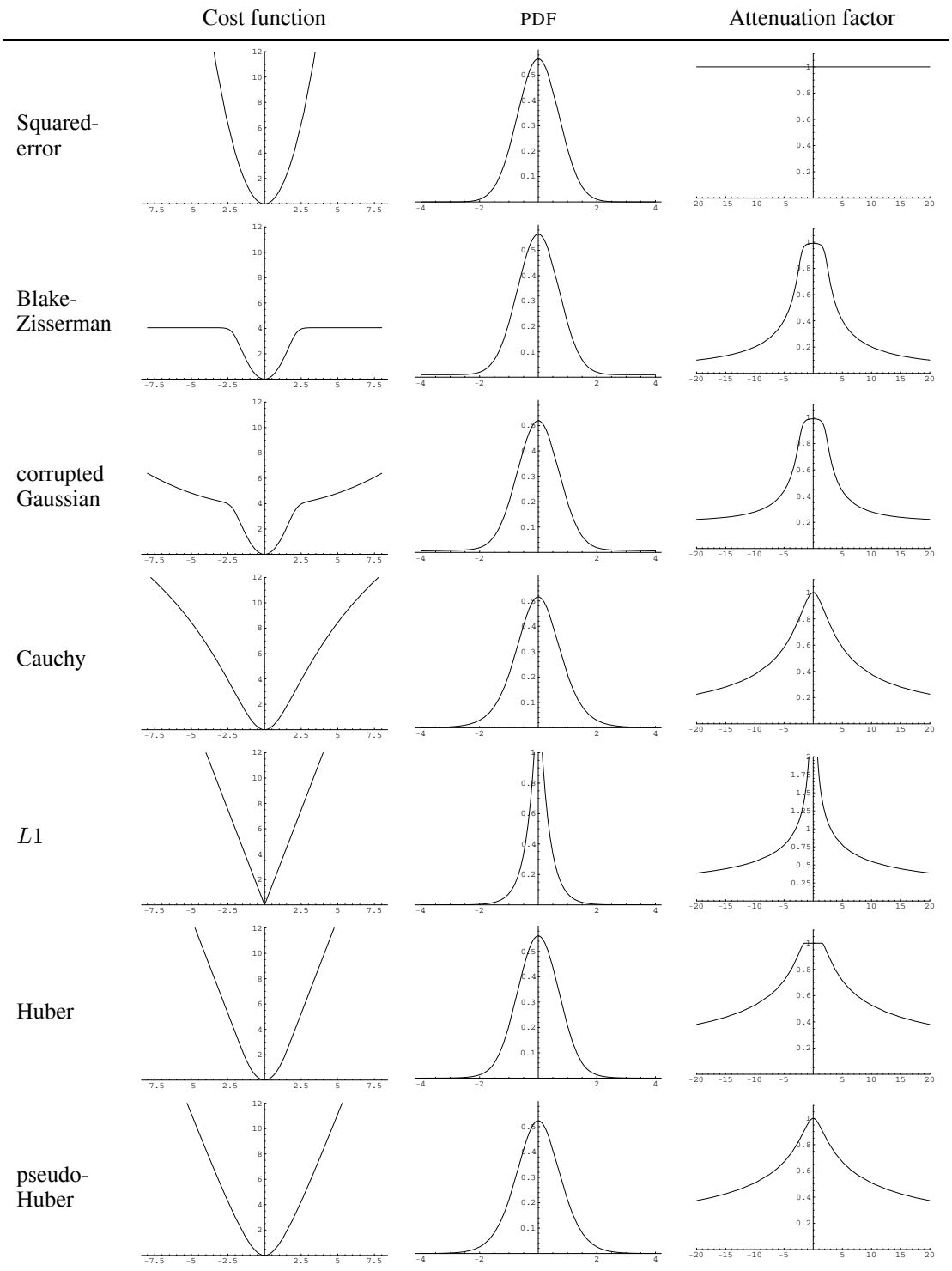

Fig. A6.4. *A comparison of different cost functions, $C(\delta)$, for robust estimation. Their corresponding* PDF*s, $\exp(-C(\delta))$, and attenuation factors ($w = C(\delta)^{1/2}/\delta$ see text) are also shown.*

Statistically based cost functions. Determination of a suitable cost function may be approached by estimating or guessing the distribution of errors for the particular measurement process involved, such as point extraction in an image. In the following list, for simplicity, we ignore the normalization constant for Gaussian distributions $(2\pi\sigma^2)^{-1/2}$, and assume that $2\sigma^2 = 1$.

(i) **Squared error.** Assuming the data is Gaussian distributed, the Probability Distribution Function (PDF) is $p(\delta) = \exp(-\delta^2)$ which leads to a cost function

$$C(\delta) = \delta^2.$$

(ii) **Blake-Zisserman.** The data is assumed to have a Gaussian distribution for inliers with a uniform distribution of outliers. The PDF is taken to be of the form $p(\delta) = \exp(-\delta^2) + \epsilon$. This is not actually a PDF, since it integrates to infinity. Nevertheless, it leads to a cost function of the form

$$C(\delta) = -\log(\exp(-\delta^2) + \epsilon).$$

For inliers (small δ), this approximates δ^2, whereas for outliers (large δ) the asymptotic cost is $-\log\epsilon$. Thus, the crossover point from inliers to outliers is given approximately by $\delta^2 = -\log\epsilon$. The actual cost function used by Blake and Zisserman in [Blake-87] was $\min(\delta^2, \alpha^2)$ and $\epsilon = \exp(-\alpha^2)$.

(iii) **Corrupted Gaussian.** The previous example has the theoretical disadvantage that it is not actually a PDF. An alternative is to model the outliers by a Gaussian with larger standard deviation, leading to a mixture model probability distribution of the form $p(\delta) = \alpha\ \exp(-\delta^2) + (1-\alpha)\exp(\delta^2/w^2)/w$ where w is the ratio of standard deviations of the outliers to the inliers, and α is the expected fraction of inliers. Then

$$C(\delta) = -\log(\alpha\exp(-\delta^2) + (1-\alpha)\exp(-\delta^2/w^2)/w).$$

Heuristic cost functions. Next we consider cost functions justified more by heuristics and required noise-immuneness properties than by adherence to a specific noise-distribution model. For this reason they will be introduced directly as a cost function, rather than a PDF.

(i) **Cauchy cost function.** The cost function is given by

$$C(\delta) = b^2\log(1 + \delta^2/b^2)$$

for some constant b. For small values of δ, this curve approximates δ^2, and the value of b determines for what range of δ this approximation is close. The cost function is derived from the Cauchy distribution $p(\delta) = 1/(\pi(1+\delta^2))$, which is a bell-curve similar to the Gaussian, but with heavier tails.

(ii) **The $L1$ cost function.** Instead of using the sum of squares, we use the sum of absolute errors. Thus,

$$C(\delta) = 2b|\delta|$$

where $2b$ is some positive constant (which normally could just be 1). This cost function is known as the *total variation*.

(iii) **Huber cost function.** This cost function is a hybrid between the $L1$ and least-squares cost function. Thus, we define

$$\begin{aligned} C(\delta) &= \delta^2 \text{ for } |\delta| < b \\ &= 2b|\delta| - b^2 \text{ otherwise} \end{aligned}$$

This cost function is continuous with continuous first derivative. The value of the threshold b is chosen approximately to equal the outlier threshold.

(iv) **Pseudo-Huber cost function.** The cost function

$$C(\delta) = 2b^2(\sqrt{1 + (\delta/b)^2} - 1)$$

is very similar to the Huber cost function, but has continuous derivatives of all orders. Note that it approximates δ^2 for small δ and is linear with slope $2b$ for large δ.

A6.8.1 Properties of the different cost functions

Squared error. The most basic cost function is the squared error $C(\delta) = \delta^2$. Its main drawback is that it is not robust to outliers in the measurements, as we shall see. Because of the rapid growth of the quadratic curve, distant outliers exert an excessive influence, and can draw the cost minimum well away from the desired value.

Non-convex cost functions. The Blake-Zisserman, corrupted Gaussian and Cauchy cost functions seek to mitigate the deleterious effect of outliers by giving them diminished weight. As is seen in the plot of the first two of these, once the error exceeds a certain threshold, it is classified as an outlier, and the cost remains substantially constant. The Cauchy cost function also seeks to deemphasize the cost of outliers, but this is done more gradually. These three cost functions are non-convex, which has important effects as we will see.

Asymptotically linear cost functions. The $L1$ cost function measures the absolute value of the error. The main effect of this is to give outliers less weight compared with the squared error. The key to understanding the performance of this cost function is to observe that it acts to find the *median* of a set of data. Consider a set of real valued data $\{a_i\}$ and a cost function defined by $C(x) = \sum_i |x - a_i|$. The minimum of this function is at the median of the set $\{a_i\}$. To see this, note that the derivative of $|x - a_i|$ with respect to x is $+1$ when $x > a_i$ and -1 when $x < a_i$. Thus, the derivative is zero when there are as many values of a_i less than x as there are greater than x. Thus, the cost is minimized at the median of the values a_i. Note that the median is immune to changes in the values of data a_i that lie far from the median. The value of the cost function changes, but not the position of its minimum.

For higher dimensional data $\mathbf{a}_i \in \mathbb{R}^n$, the minimum of the cost function $C(\mathbf{x}) = \sum_i \|\mathbf{x} - \mathbf{a}_i\|$ has similar stability properties. Note that $\|\mathbf{x} - \mathbf{a}_i\|$ is a *convex* function

of x, and therefore so is a sum of such terms, $\sum_i \|\mathbf{x} - \mathbf{a}_i\|$. Consequently, this cost function has a single minimum (as do all convex functions).

The Huber cost function takes the form of a quadratic for small values of the error, δ, and becomes linear for values of δ beyond a given threshold. As such, it retains the outlier stability of the $L1$ cost function, while for inliers it reflects the property that the squared-error cost function gives the Maximum Likelihood estimate.

The Pseudo-Huber cost function is also near-quadratic for small δ, and linear for large δ. Thus, it may be used as a smooth approximation to the Huber cost function, and gives similar results. It is important to note that each of these three cost functions has the very desirable property of being convex.

A6.8.2 Performance of the different cost functions

To illustrate the properties of the different cost functions we will evaluate the cost $\sum_i C(x - a_i)$ for two synthetic example data sets $\{a_i\}$. Of the group of asymptotically linear cost functions, only the Huber cost function will be shown, since the other two ($L1$ and pseudo-Huber) give very similar results.

The data $\{a_i\}$ may be thought of as the outcome of an experiment to measure some quantity, with repeated measurements. The measurements are subject to random Gaussian noise, with outliers. The purpose of the estimation process is to estimate the value of the quantity by minimizing a cost function. The experiments and the results for the two data sets are described in the captions of figure A6.5 and figure A6.6.

Summary of findings. The squared-error cost function is generally very susceptible to outliers, and may be regarded as unusable as long as outliers are present. If outliers have been thoroughly eradicated, using for instance RANSAC, then it may be used.

The non-convex cost functions, though generally having a stable minimum, not much effected by outliers have the significant disadvantage of having local minima, which can make convergence to a global minimum chancy. The estimate is not strongly attracted to the minimum from outside of its immediate neighbourhood. Thus, they are not useful, unless (or until) the estimate is close to the final correct value.

The Huber cost function has the pleasant property of being convex, which makes convergence to a global minimum more reliable. The minimum is quite immune to the baleful influence of outliers since it represents a compromise between the Maximum Likelihood estimate of the inliers and the median of the outliers. The pseudo-Huber cost function is a good alternative to Huber, but use of $L1$ should be approached with care, because of its non-differentiablity at the origin.

These findings were illustrated on one-dimensional data, but they carry over to higher dimensional data also.

Parameter minimization. We have seen that the Huber and related cost functions are convex, and hence have a single minimum. We refer here to the cost $C(\delta)$ as a function of the error δ. In general in problems such as structure from motion, the error δ itself is a non-linear function of the parameters (such as camera and point positions). For this reason, the total cost expressed as a function of the motion and structure parameters

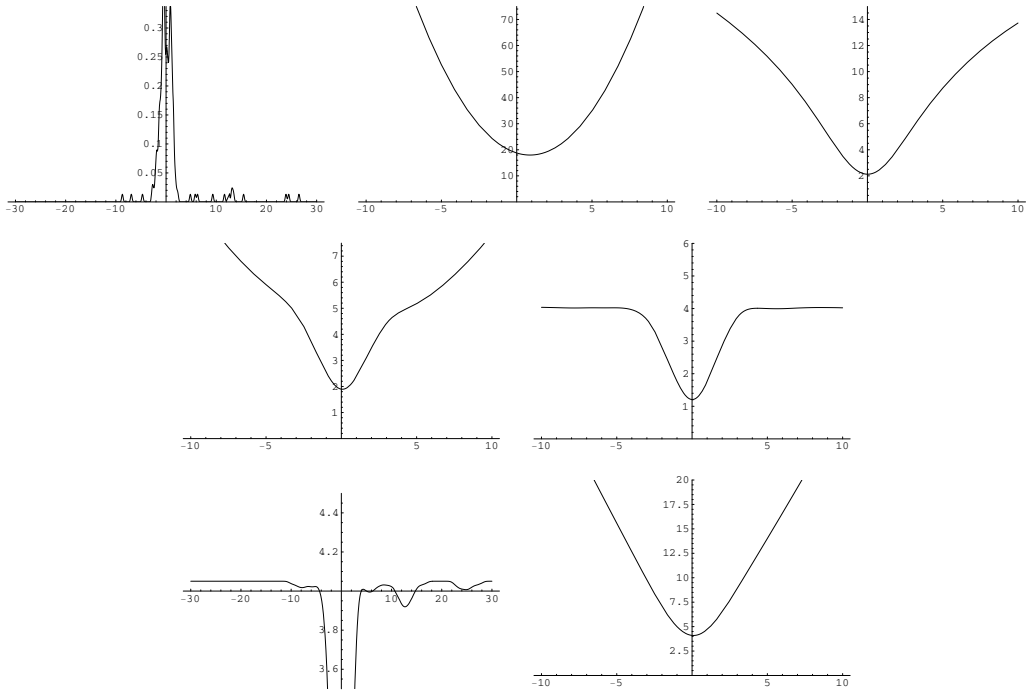

Fig. A6.5. *The data $\{a_i\}$ (illustrated in the top left graph) consists of a set of measurements centred around 0 with unit Gaussian noise, plus 10% of outliers biased towards the right of the true value. The graphs of $\sum_i C(|x - a_i|)$ correspond (left-to-right and top-to-bottom) to the cost functions Squared error, Cauchy, corrupted-Gaussian, Blake-Zisserman, a zoom of the Blake-Zisserman, and Huber cost functions. Note that the minimum of the squared-error cost function is pulled significantly to the right by the outliers, whereas the other cost-functions are relatively outlier-independent.*
The Blake-Zisserman cost function, which is based most nearly on the distribution of the data, has a very clear minimum. However, close examination (the zoomed plot) shows an undesirable characteristic, which is the presence of local minima near each of the outliers. An iterative method to find the cost minimum will fail if it is initiated outside the narrow basin of attraction surrounding the minimum.
By contrast, the Huber cost function is convex, which means the estimate will be drawn towards the single minimum from any initialization point.

can not be expected to be convex, and local minima are inevitable. Nevertheless, an important principle is:

- *choose a parameterization in which the error is as close as possible to being a linear function of the parameters, at least locally.*

Observing this principle will lead to simpler cost surfaces with fewer local minima, and generally quicker convergence.

Cost functions and least-squares. Usually cost functions of the type we have discussed are used in the context of a parameter minimization procedure, such as Levenberg-Marquardt or Newton iteration. Commonly, these procedures seek to minimize the norm of some vector $\boldsymbol{\Delta}$ depending on a set of parameters $\mathbf{p}$. Thus, they minimize $\|\boldsymbol{\Delta}(\mathbf{p})\|^2$ over all choices of the parameter vector $\mathbf{p}$. For instance in a structure and motion problem we may seek to minimize $\sum_i \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 = \sum_i \|\boldsymbol{\delta}_i\|^2 = \|\boldsymbol{\Delta}\|^2$ where the values $\mathbf{x}_i$ are measured image coordinates, the $\hat{\mathbf{x}}_i$ are the predicted values de-

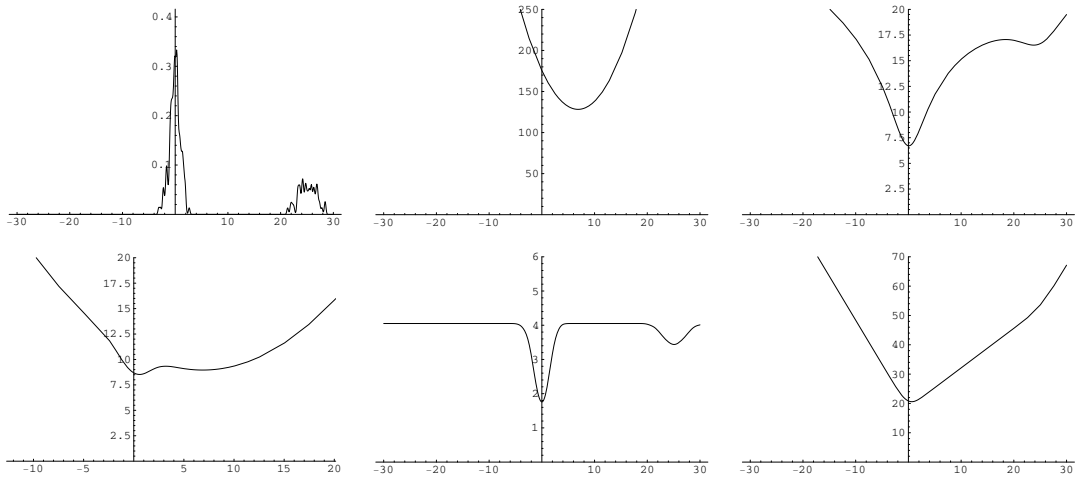

Fig. A6.6. *In this experiment, as in figure A6.5, the main part of the data (70%) is centred at the origin, with 30% of "outliers" concentrated in a block away from the origin (see top left graph). This type of measurement distribution is quite realistic in many imaging scenarios, for instance where point or edge measurement is confused by ghost edges. The cost functions in order from the top are: Square error, Cauchy, corrupted Gaussian, Blake-Zisserman and Huber.*
The Squared error cost function finds the mean of the measurement distribution, which is significantly pulled to the right by the block of outliers. The effect of the outlier block on the non-convex cost functions is also shown clearly here. Because of the non-convexity, the total cost function does not have a single minimum, but rather two minima around the separate blocks of measurements. Because of its convexity, the Huber cost function has a single minimum, which is located close to the median of the data, and is hardly influenced by the presence of the 30% of outliers.

rived from the current parameter values, and $\boldsymbol{\Delta}$ is the vector formed by concatenating the individual error vectors $\boldsymbol{\delta}_i$.

Since minimization of a squared vector norm $\|\boldsymbol{\Delta}\|^2$ is built into most implementations of Levenberg-Marquardt, we need to see how to apply the robust cost function in this case. The answer is to replace each vector $\boldsymbol{\delta}_i$ by a weighted vector $\boldsymbol{\delta}'_i = w_i\boldsymbol{\delta}_i$ such that

$$\|\boldsymbol{\delta}'_i\|^2 = w_i^2\|\boldsymbol{\delta}_i\|^2 = C(\|\boldsymbol{\delta}_i\|)$$

for then $\sum_i C(\|\boldsymbol{\delta}_i\|) = \sum_i \|\boldsymbol{\delta}_i\|^2$ as desired. From this equality, we find

$$w_i = C(\|\boldsymbol{\delta}_i\|)^{1/2}/\|\boldsymbol{\delta}_i\|. \tag{A6.18}$$

Thus, the minimization problem is to minimize $\|\boldsymbol{\Delta}'\|^2$ where $\boldsymbol{\Delta}'$ is the vector obtained by concatenating the vectors $\boldsymbol{\delta}'_i = w_i\boldsymbol{\delta}_i$, and each w_i is computed from (A6.18). Note that w_i is a function of $\|\boldsymbol{\delta}_i\|$, which normally seeks to attenuate the cost of the outliers. This attenuation function is shown in the final column of figure A6.4 for the different cost functions. For the Squared error cost function, the attenuation factor is 1, meaning no attenuation occurs. For the other cost functions, there is little attenuation within an inlier region, and points outside this region are attenuated to different degrees.

A6.9 Parametrization

In the sort of iterative estimate problems we are considering here, an important consideration is how the solution space is to be parametrized. Most of the geometric entities that we consider do not have a simple Euclidean parametrization. For example, in solving a geometric optimization problem, we often wish to iterate over all camera orientations, represented by rotations. Rotations are most conveniently represented for computational purposes by rotation matrices, but this is a major overparametrization. The set of 3D rotations is naturally 3-dimensional (in fact, it forms a 3-dimensional Lie group, $SO(3)$), and we may wish to parametrize it with only 3 parameters.

Homogeneous vectors or matrices are another common type of representation. It is tempting to use a parametrization of a homogeneous n-vector just by using the components of the vector itself. However, there are really only $n-1$ degrees of freedom in a homogeneous n-vector, and it is sometimes advantageous to parametrize it with only $n-1$ parameters.

Gauge freedom. There has been much attention paid to *gauge freedom* and *gauge independence* in recent papers (for instance [McLauchlan-00]). In this context, the word gauge means a coordinate system for a parameter set, and gauge-freedom essentially refers to a change in the representation of the parameter set that does not essentially change the underlying geometry, and hence has no effect on the cost function. The most important gauge freedoms commonly encountered are projective or other ambiguities, such as those arising in reconstruction problems. However, the scale ambiguity of homogeneous vectors can be counted as gauge freedom also. Gauge freedoms in the parametrization of an optimization problem cause the normal equations to be singular, and hence allow multiple solutions. This problem is avoided by the regularization (or enhancement) step in Levenberg-Marquardt, but there is evidence that excessive gauge freedoms, causing slop in the parametrization, can lead to slower convergence. In addition, when gauge freedoms are present the covariance matrix of the estimated parameters is troublesome, in that there will be infinite variance in unconstrained parameter directions. For instance, it makes no sense to talk of the covariance matrix of an estimated homogeneous vector, unless the scale of the vector is constrained. Therefore, in the following pages, we will give some methods for obtaining minimal parametrizations for certain common geometric parameter sets.

What makes a good parametrization? The foremost requirement of a good parametrization is that it be singularity-free, at least in the areas that are visited during the course of an iterative optimization. This means that the parametrization should be locally continuous, differentiable and one-to-one – in short a diffeomorphism. A simple example of what is meant by this is given by the latitude-longitude parametrization of a sphere, that is, spherical-polar coordinates. This has a singularity at the poles, where the mapping from the lat-long coordinates to a neighbourhood of the pole is not one-to-one. The difficulty is that the point with coordinates (latitude = $89°$, longitude = $0°$) is very close to the point (latitude = $89°$, longitude = $90°$) – they are both points very close to the pole. However, they are a long way apart in parameter space. To see

the effect of this, suppose that an optimization is taking place on the sphere, tracking down a minimum of a cost function, which exists at the point $(89^\circ, 90^\circ)$. If the current estimate is proceeding in the general direction of this minimum, up the line of zero longitude, when it gets near the pole it will find that although close to the minimum, it can not get there without a long detour in lat-long parameter space. The difficulty is that arbitrarily close points on the sphere, in the neighbourhood of the singularity (the pole) can have large differences in parameter values. The same sort of thing happens with representations of rotations using Euler angles.

Now, we move on to consider some specific parametrizations.

A6.9.1 Parametrization of 3D rotations

Using the angle-axis formula of (A4.9–*p*584) we may parametrize a 3×3 rotation by a 3-vector $\mathbf{t}$. This represents a rotation through an angle $\|\mathbf{t}\|$ about the axis determined by the vector $\mathbf{t}$. We denote the corresponding rotation matrix by $\mathtt{R}(\mathbf{t})$.

Regarding this representation, we may make certain simple observations:

(i) The identity map (no rotation) is represented by the zero vector $\mathbf{t} = \mathbf{0}$.

(ii) If some rotation $\mathtt{R}$ is represented by a vector $\mathbf{t}$, then the inverse rotation is represented by $-\mathbf{t}$. In symbols: $\mathtt{R}(\mathbf{t})^{-1} = \mathtt{R}(-\mathbf{t})$.

(iii) If $\mathbf{t}$ is small, then the rotation matrix is approximated by $\mathtt{I} + [\mathbf{t}]_\times$.

(iv) For small rotations represented by $\mathbf{t}_1$ and $\mathbf{t}_2$, the composite rotation is represented to first-order by $\mathbf{t}_1 + \mathbf{t}_2$. In other words $\mathtt{R}(\mathbf{t}_1)\mathtt{R}(\mathbf{t}_2) \approx \mathtt{R}(\mathbf{t}_1 + \mathbf{t}_2)$. Thus the mapping $\mathbf{t} \mapsto \mathtt{R}(\mathbf{t})$ is to first order a group isomorphism for small $\mathbf{t}$. In fact, for small $\mathbf{t}$, the map is an isometry (distance preserving map) in terms of the *distance* between two rotations $\mathtt{R}_1$ and $\mathtt{R}_2$ defined to equal to the angle of the rotation $\mathtt{R}_1\mathtt{R}_2^{-1}$.

(v) Any rotation may be represented as $\mathtt{R}(\mathbf{t})$ for some $\mathbf{t}$ such that $\|\mathbf{t}\| \leq \pi$. That is, any rotation is a rotation through an angle of at most π radians about some axis. The mapping $\mathbf{t} \mapsto \mathtt{R}(\mathbf{t})$ is one-to-one for $\|\mathbf{t}\| < \pi$ and two-to-one for $\|\mathbf{t}\| < 2\pi$. If $\|\mathbf{t}\| = 2\pi$, then $\mathtt{R}(\mathbf{t})$ is the identity map, regardless of $\mathbf{t}$. Thus, the parametrization has a singularity at $\|\mathbf{t}\| = 2\pi$.

(vi) **Normalization:** In parametrizing rotations by a vector $\mathbf{t}$ it is best to maintain the condition that $\|\mathbf{t}\| \leq \pi$, in order to keep away from the singularity when $\|\mathbf{t}\| = 2\pi$. If $\|\mathbf{t}\| > \pi$, then it may be replaced by the vector $(\|\mathbf{t}\| - 2\pi)\mathbf{t}/\|\mathbf{t}\| = \mathbf{t}(1 - 2\pi/\|\mathbf{t}\|)$, which represents the same rotation.

A6.9.2 Parametrization of homogeneous vectors

The quaternion representation of a rotation (section A4.3.3(*p*585)) is a redundant representation in that it contains 4 parameters where 3 will suffice. The angle-axis representation on the other hand is a minimum parametrization. Many entities in projective geometry are represented by homogeneous quantities, either vectors or matrices, for instance points in projective space or the fundamental matrix to name a few. For computational purposes, it is possible to represent such quantities as vectors with a min-

imum number of parameters in a similar way to which the angle-axis representation gives an alternative to the quaternion representation of a rotation.

Let $\mathbf{v}$ be a vector of any dimension, and represent by $\bar{\mathbf{v}}$ the unit vector $(\text{sinc}(\|\mathbf{v}\|/2)\mathbf{v}^\mathsf{T},\ \cos(\|\mathbf{v}\|/2))^\mathsf{T}$. This mapping $\mathbf{v} \mapsto \bar{\mathbf{v}}$ maps the disk of radius π (that is, the set of vectors of length at most π) smoothly and one-to-one onto the set of unit vectors $\bar{\mathbf{v}}$ with non-negative final coordinate. Thus, it provides a mapping onto the set of homogeneous vectors. The only difficult point with this mapping is that it takes any vector of length 2π to the same vector $(\mathbf{0}, -1)^\mathsf{T}$. However, this singular point may be avoided by renormalizing any vector $\mathbf{v}$ of length $\|\mathbf{v}\| > \pi$, replacing it with $(\|\mathbf{v}\| - 2\pi)\mathbf{v}/\|\mathbf{v}\|$ which represents the same homogeneous vector $\bar{\mathbf{v}}$.

A6.9.3 Parametrization of the n-sphere

Commonly it occurs in geometric optimization problems that some vector of parameters is required to lie on a unit sphere. As an example, consider a complete Euclidean bundle-adjustment with two views. The two cameras may be taken as $\mathtt{P} = [\mathtt{I} \mid \mathbf{0}]$ and $\mathtt{P}' = [\mathtt{R}|\mathbf{t}]$, where $\mathtt{R}$ is a rotation and $\mathbf{t}$ is a translation. In addition, 3D points $\mathbf{X}_i$ are defined, which are to map via the two camera matrices to image points. This defines an optimization problem in which the parameters are $\mathtt{R}$, $\mathbf{t}$ and the points $\mathbf{X}_i$, and the cost to be minimized is a geometric residual of the projections with respect to the image measurements. In this problem, there is an overall scale ambiguity, and this is conveniently resolved by requiring that $\|\mathbf{t}\| = 1$.

A minimum parametrization for the rotation matrix $\mathtt{R}$ is given by the rotation parametrization of section A6.9.1. Similarly, the points $\mathbf{X}_i$ are conveniently parametrized as homogeneous 4-vectors, using the parametrization of section A6.9.2. We consider in this section how one may parametrize the unit vector $\mathbf{t}$. Note that we can not simply parametrize $\mathbf{t}$ as a homogeneous vector, since change of sign of $\mathbf{t}$ changes the projection $\mathtt{P}'\mathbf{X} = [\mathtt{R}|\mathbf{t}]\mathbf{X}$.

The same problem arises in multiple-view Euclidean bundle-adjustment. In this case we have many camera matrices $\mathtt{P}^i = [\mathtt{R}^i|\mathbf{t}^i]$ and we may fix $\mathtt{P}^0 = [\mathtt{I} \mid \mathbf{0}]$. The set of translations $\mathbf{t}^i$ for all $i > 0$ are subject to scale ambiguity. We can minimally parametrize the translations by requiring that $\|\mathbf{T}\| = 1$, where $\mathbf{T}$ is the vector formed by concatenating all the $\mathbf{t}^i$ for $i > 0$.

There may be several ways of parametrizing a unit vector. We consider one particular parametrization here based on a local parametrization of the tangent plane to the unit sphere. We consider a sphere of dimension n, which consists of the set of $(n+1)$-vectors of unit length. Let $\mathbf{x}$ be a such a vector. Let $\mathtt{H}_{\mathbf{v}(\mathbf{x})}$ be a Householder matrix (see section A4.1.2(*p*580)) such that $\mathtt{H}_{\mathbf{v}(\mathbf{x})}\mathbf{x} = (0, \ldots, 0, 1)^\mathsf{T}$. Thus, we have transformed the vector $\mathbf{x}$ to lie along the coordinate axis. Now, we consider a parametrization of the unit sphere in the vicinity of $(0, \ldots, 0, 1)^\mathsf{T}$. Such a parametrization is a map $\mathbb{R}^n \to S^n$ that is well behaved in the vicinity of the origin. There are many choices, of which two possibilities are

(i) $f(\mathbf{y}) = \hat{\mathbf{y}}/\|\hat{\mathbf{y}}\|$ where $\hat{\mathbf{y}} = (\mathbf{y}^\mathsf{T}, 1)^\mathsf{T}$
(ii) $f(\mathbf{y}) = (\text{sinc}(\|\mathbf{y}\|/2)\mathbf{y}^\mathsf{T}, \cos(\|\mathbf{y}\|)/2)^\mathsf{T}$.

Both these functions map the origin $(0,\ldots,0)^\mathsf{T}$ to $(0,\ldots,0,1)^\mathsf{T}$, and their Jacobian is $\partial f/\partial \mathbf{y} = [\mathtt{I}|\mathbf{0}]^\mathsf{T}$. Note that although we are interested in these functions just as local parametrizations, the first provides a parametrization for half the sphere, whereas for $\|\mathbf{y}\| \leq \pi$ the second map parametrizes the whole of the sphere with no singularity except at $\|\mathbf{y}\| = 2\pi$.

The composite map $\mathbf{y} \mapsto \mathtt{H}_{\mathbf{v}(\mathbf{x})} f(\mathbf{y})$ provides a local parametrization for a neighbourhood of the point $\mathbf{x}$ on the sphere (note here that we should write $\mathtt{H}^{-1}_{\mathbf{v}(\mathbf{x})}$, but $\mathtt{H}_{\mathbf{v}(\mathbf{x})} = \mathtt{H}^{-1}_{\mathbf{v}(\mathbf{x})}$). The Jacobian of this map is simply $\mathtt{H}_{\mathbf{v}(\mathbf{x})}[\mathtt{I}|\mathbf{0}]^\mathsf{T}$, which consists of the first n columns of the Householder matrix, and hence is easy to compute.

In minimization problems, we usually need to compute a Jacobian matrix $\partial C/\partial y$, of a vector valued cost function C with respect to a set of parameters $\mathbf{y}$. In the case where the parameters are constrainted to lie on a sphere S^n in $\mathbb{R}^{n+1}$ the cost function is nonetheless usually defined for all values of the parameter $\mathbf{x}$ in $\mathbb{R}^{n+1}$. As an example, in the Euclidean bundle-adjustment problem considered at the start of this section, the cost function (for instance residual reprojection error) can be defined for all pairs of cameras $\mathtt{P} = [\mathtt{I} \mid \mathbf{0}]$ and $\mathtt{P}' = [\mathtt{R}|\mathbf{t}]$ with $\mathbf{t}$ taking any value. Nevertheless, we may wish to minimize the cost function constraining $\mathbf{t}$ to lie on a sphere.

Thus, consider the case where a cost function $C(\mathbf{x})$ is defined for $\mathbf{x} \in \mathbb{R}^{n+1}$, but we parametrize $\mathbf{x}$ to lie on a sphere by setting $\mathbf{x} = \mathtt{H}_{\mathbf{v}(\mathbf{x})} f(\mathbf{y})$ with $\mathbf{y} \in \mathbb{R}^n$. In this case, we see that

$$\mathtt{J} = \frac{\partial C}{\partial \mathbf{y}} = \frac{\partial C}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{y}} = \frac{\partial C}{\partial \mathbf{x}}\mathtt{H}_{\mathbf{v}(\mathbf{x})}[\mathtt{I}|\mathbf{0}]^\mathsf{T}.$$

In summary, by using local parametrizations, a parameter vector may be constrained to lie on an n-sphere with a modest added computational cost compared with the over-parametrization of allowing the vector to vary over the whole of $\mathbb{R}^{n+1}$. The key points of the method are as follows.

(i) Store the parameter vector $\mathbf{x} \in \mathbb{R}^{n+1}$, satisfying $\|\mathbf{x}\| = 1$.

(ii) In forming the linear update equations, compute the Jacobian matrix $\partial C/\partial \mathbf{x}$, and multiply it by $\mathtt{H}_{\mathbf{v}(\mathbf{x})}[\mathtt{I}|\mathbf{0}]^\mathsf{T}$ to obtain the Jacobian with respect to a minimal parameter set $\mathbf{y}$. Multiplication of $\partial C/\partial \mathbf{x}$ by $\mathtt{H}_{\mathbf{v}(\mathbf{x})}[\mathtt{I}|\mathbf{0}]^\mathsf{T}$ is efficiently carried out by the method of (A4.4–*p*580).

(iii) The iteration step provides an increment parameter vector $\boldsymbol{\delta}_\mathbf{y}$. Compute the new value of $\mathbf{x} = \mathtt{H}_{\mathbf{v}(\mathbf{x})} f(\boldsymbol{\delta}_\mathbf{y})$.

Essentially the same method of using local-parametrizations may be used more generally in other situations where a minimal parametrization is required. For instance, in section 11.4.2(*p*285) it was seen how the fundamental matrix may be parametrized locally with a minimum number of parameters, but no minimal parametrization can cover the whole set of fundamental matrices.

A6.10 Notes and exercises

(i) We prove in various steps the form of the pseudo-inverse of a block matrix given in (A6.13–*p*606).

(a) Recall that $\mathtt{H}^{+} = \mathtt{G}(\mathtt{G}^{\mathsf{T}}\mathtt{H}\mathtt{G})^{-1}\mathtt{G}^{\mathsf{T}}$ if and only if $N_L(\mathtt{G}) = N_L(\mathtt{H})$ (see section A5.2(*p*590)).

(b) Let $\mathtt{G}$ be invertible. Then $(\mathtt{G}\mathtt{H}\mathtt{G}^{\mathsf{T}})^{+} = \mathtt{G}^{-\mathsf{T}}\mathtt{H}^{+}\mathtt{G}^{-1}$ if and only if $N_L(\mathtt{H})\mathtt{G}^{\mathsf{T}} = N_L(\mathtt{H})\mathtt{G}^{-1}$.

(c) Applying this condition to (A6.12–*p*605) the necessary and sufficient condition for (A6.13–*p*606) to be valid is that $N_L(\mathtt{U} - \mathtt{W}\mathtt{V}^{-1}\mathtt{W}^{\mathsf{T}}) \subseteq N_L(\mathtt{Y}) = N_L(\mathtt{W})$.

(d) With $\mathtt{U}$, $\mathtt{V}$ and $\mathtt{W}$ defined in terms of $\mathtt{A}$ and $\mathtt{B}$ as in (A6.11–*p*605) this is equivalent to the condition $\mathrm{span}(\mathtt{A}) \cap \mathrm{span}(\mathtt{B}) = \emptyset$.

(ii) Investigate conditions under which the condition $\mathrm{span}(\mathtt{A}) \cap \mathrm{span}(\mathtt{B}) = \emptyset$ is true. It may be interpreted as meaning that the effects of varying the parameters $\mathbf{a}$ (for instance camera parameters) and the effects of varying $\mathbf{b}$ (point parameters) can not be complementary. Clearly this is not the case with (for instance) unconstrained projective reconstruction where both cameras and points may vary without affecting the measurements. In such a case, the variance of parameters $\mathbf{a}$ and $\mathbf{b}$ is infinite in directions $\delta_{\mathbf{a}}$ and $\delta_{\mathbf{b}}$ such that $\mathtt{A}\delta_{\mathbf{a}} = \mathtt{B}\delta_{\mathbf{b}}$.