Sparse Non-linear Least Squares Optimization for Geometric Vision

Manolis I.A. Lourakis

Institute of Computer Science, Foundation for Research and Technology - Hellas
N. Plastira 100, Vassilika Vouton, Heraklion, Crete, 700 13 Greece
http://www.ics.forth.gr/~lourakis/sparseLM/

Abstract. Several estimation problems in vision involve the minimization of cumulative geometric error using non-linear least-squares fitting. Typically, this error is characterized by the lack of interdependence among certain subgroups of the parameters to be estimated, which leads to minimization problems possessing a sparse structure. Taking advantage of this sparseness during minimization is known to achieve enormous computational savings. Nevertheless, since the underlying sparsity pattern is problem-dependent, its exploitation for a particular estimation problem requires non-trivial implementation effort, which often discourages its pursuance in practice. Based on recent developments in sparse linear solvers, this paper provides an overview of `sparseLM`, a general-purpose software package for sparse non-linear least squares that can exhibit arbitrary sparseness and presents results from its application to important sparse estimation problems in geometric vision.

1 Introduction

A plethora of estimation problems in multiple view geometry employ model fitting to infer mathematical objects from image data. Fitting is accomplished by minimizing the total geometric error pertaining to overdetermined sets of image measurements, which is an approach that has proven to constitute a major contributor to the success of contemporary algorithms in multiple view geometry [1]. The total geometric error is expressed by a sum-of-squares cost function (i.e., a L_2 norm), whose minimizer represents the statistically optimal estimate of the sought objects under Gaussian noise. Owing to their non-convexity, L_2 cost functions are minimized with iterative non-linear least squares techniques, of which the Levenberg-Marquardt (LM) algorithm has become the de facto standard. LM operates by repeatedly linearizing the function to be minimized in the neighborhood of the current minimizer estimate and computing an improvement to it through the solution of a linear system defined with the aid of the Jacobian and known as the *normal equations*. Considering that each computation of the solution to a dense linear system has complexity $O(N^3)$ in the number of unknown parameters, it is clear that general purpose LM implementations are computationally very demanding when employed to minimize functions involving a large number of parameters N.

K. Daniilidis, P. Maragos, N. Paragios (Eds.): ECCV 2010, Part II, LNCS 6312, pp. 43–56, 2010.

Fortunately, when dealing with large estimation problems arising in multiple view geometry, the corresponding geometric error exhibits lack of interdependence among certain subgroups of the parameters to be estimated. This observation translates to Jacobians for the least squares minimization that are sparse, that is, consist of mostly zero elements. In turn, sparse Jacobians yield normal equation systems with sparse block structure. Examples of such sparse problems include single view reconstruction [2], homography, fundamental matrix and trifocal tensor estimation with the "Gold Standard" algorithms [1] (pp.114, 285 & 397 resp.), mosaicking [3] and bundle adjustment [4,5]. It is well-known that by avoiding storing and operating on zero elements of the normal matrix during the course of LM, substantial memory and execution time benefits can be gained. For instance, Appendix 6 of [1] describes a scheme for effectively dealing with the commonly encountered "arrowhead" type of sparseness (see also Fig. (1)(a)). This scheme performs a partitioning of the set of parameters in two functionally distinct groups and solves the normal equations by employing the corresponding Schur complement of the normal matrix. Its adoption has facilitated the implementation of LM variants tailored to the problem of bundle adjustment that divide the normal matrix into camera and structure blocks and are capable of successfully dealing with large reconstruction problems [5]. Despite its usefulness, the aforementioned scheme is not suited to all sparse problems that might be encountered in multiple view geometry, while its implementation is problem-specific and rather complicated. Therefore, considerable effort is required for developing LM variants customized to a particular sparse problem, making the latter task to be perceived as a daunting endeavor by both vision researchers and practitioners.

The reason behind the lack of universal applicability of the partitioning scheme of [1] is that its assumption of only two functional groups of parameters is not valid for all estimation problems. In other words, there exist problems whose Jacobian (and, therefore, normal equations) sparsity pattern has a more complex structure (e.g. Fig. 1(b)). Nonetheless, if an effective mechanism of dealing with arbitrary sparseness is available, then all sparse geometric vision estimation problems can be cast as special cases of the general sparse non-linear least squares minimization problem. During the last few years, such mechanisms have emerged in the form of a number of algorithms and corresponding implementations for the direct solution of large sparse linear systems of equations [6]. Compared to iterative methods [7], sparse direct methods do not employ preconditioners, do not suffer from slow convergence, produce exact rather than approximate solutions and their technology is well developed. Thus, they are more general and robust, therefore better suited as general-purpose linear solvers.

This work builds upon existing direct sparse solvers and employs them for developing `sparseLM`, a package fulfilling the need for a quality software designed for general-purpose, arbitrarily sparse non-linear least squares fitting. `sparseLM` is implemented in C and its source code is publicly available under the GNU GPL. To the best of the author's knowledge, no other comparable software is currently freely available with an open source license. Brief introductions to the LM algorithm and sparse direct solvers are supplied in sections 2 and 3, respectively. Section 4 presents

the major design guidelines and implementation issues related to `sparseLM`. Experimental results from the application of `sparseLM` to practical vision problems are provided in section 5 and the paper concludes in section 6.

2 The Levenberg-Marquardt Algorithm

The LM algorithm is an iterative technique that locates a local minimum of a multivariate function that is expressed as the sum of squares of non-linear real-valued functions. For the sake of completeness, a short description of the LM algorithm is provided next. However, a detailed analysis of the LM algorithm is beyond the scope of this paper and the interested reader is referred elsewhere [8].

Let f be an assumed functional relation which maps a *parameter vector* $\mathbf{p} \in \mathcal{R}^m$ to an estimated *measurement vector* $\hat{\mathbf{x}} = f(\mathbf{p}),\ \hat{\mathbf{x}} \in \mathcal{R}^n$. An initial parameter estimate $\mathbf{p}_0$ and a measured vector $\mathbf{x}$ are provided and it is desired to find the vector $\mathbf{p}^+$ that best satisfies the functional relation f locally, i.e. minimizes the squared distance $\epsilon^T\epsilon$ with $\epsilon = \mathbf{x} - \hat{\mathbf{x}}$ for all $\mathbf{p}$ within a m-sphere having a small radius. The basis of the LM algorithm is a linear approximation to f in the neighborhood of $\mathbf{p}$. Denoting by $\mathbf{J}$ the Jacobian matrix $\frac{\partial f(\mathbf{p})}{\partial \mathbf{p}}$, a Taylor series expansion for a small $||\delta_\mathbf{p}||$ leads to the following approximation:

$$f(\mathbf{p} + \delta_\mathbf{p}) \approx f(\mathbf{p}) + \mathbf{J}\delta_\mathbf{p}. \tag{1}$$

Like all non-linear optimization methods, LM is iterative: Initiated at the starting point $\mathbf{p}_0$, it produces a series of vectors that converge towards a local minimizer $\mathbf{p}^+$ for f. Hence, at each iteration, it is required to find the step $\delta_\mathbf{p}$ that minimizes the quantity

$$||\mathbf{x} - f(\mathbf{p} + \delta_\mathbf{p})|| \approx ||\mathbf{x} - f(\mathbf{p}) \ - \ \mathbf{J}\delta_\mathbf{p}|| = ||\epsilon - \mathbf{J}\delta_\mathbf{p}||. \tag{2}$$

Thus, the sought $\delta_\mathbf{p}$ is obtained from a linear least-squares problem which is solved using the normal equations:

$$\mathbf{J}^T\mathbf{J}\delta_\mathbf{p} = \mathbf{J}^T\epsilon. \tag{3}$$

An alternative to minimizing (2) employs the QR decomposition, which is nevertheless up to a factor of two slower than the normal equations (cf. [4], p.315). Matrix $\mathbf{J}^T\mathbf{J}$ in Eq. (3) is the first order approximation to the Hessian of $\frac{1}{2}\epsilon^T\epsilon$ [8], whereas $\delta_\mathbf{p}$ is the *Gauss-Newton* step. The LM algorithm actually solves a slight variation of Eq. (3), known as the *augmented normal equations*

$$\mathbf{N}\delta_\mathbf{p} = \mathbf{J}^T\epsilon, \text{ with } \mathbf{N} \equiv \mathbf{J}^T\mathbf{J} + \mu\mathbf{I} \text{ and } \mu > 0, \tag{4}$$

where $\mathbf{I}$ is the identity matrix. The strategy of altering the diagonal elements of $\mathbf{J}^T\mathbf{J}$ is called *damping* and μ is a regularization parameter referred to as the *damping term*. If the updated parameter vector $\mathbf{p} + \delta_\mathbf{p}$ with $\delta_\mathbf{p}$ computed from Eq. (4) leads to a reduction in the error $\epsilon^T\epsilon$, the update is accepted and the process repeats with a decreased damping term. Otherwise, the damping term is increased,

the augmented normal equations are solved again and the process iterates until a value of $\delta_{\mathbf{p}}$ that decreases the error is found. The process of repeatedly solving Eq. (4) for different values of the damping term until an acceptable update to the parameter vector is found corresponds to one iteration of the LM algorithm.

The damping term is adaptively adjusted at each iteration of LM to assure a reduction in $\epsilon^T\epsilon$. By doing so, LM is capable of alternating between a slow descent approach when being far from the minimum and a fast convergence when being in the minimum's neighborhood: If the damping is set to a large value, matrix $\mathbf{N}$ in Eq. (4) is nearly diagonal and the LM update step $\delta_{\mathbf{p}}$ is near the steepest descent direction $\mathbf{J}^T\epsilon$. Moreover, the magnitude of $\delta_{\mathbf{p}}$ is reduced, ensuring that excessively large Gauss-Newton steps are not taken. A large damping term also handles situations where the Jacobian is rank deficient and $\mathbf{J}^T\mathbf{J}$ is therefore singular. The damping term can be chosen so that the symmetric matrix $\mathbf{N}$ in Eq. (4) is non-singular and, therefore, positive definite (SPD), ensuring that the $\delta_{\mathbf{p}}$ computed from it is a descent direction. In this way, LM can defensively navigate a region of the parameter space in which the model is highly non-linear. If, on the other hand, the damping is small, the LM step approximates the exact Gauss-Newton step, lending LM rapid convergence.

3 Direct Sparse Linear Solvers

The solution of systems of sparse linear equations lies at the crux of numerous computational problems. Direct methods for solving the linear system $\mathbf{Ax} = \mathbf{b}$, where the coefficient matrix $\mathbf{A}$ is sparse, involve the explicit factorization of a suitable permutation of $\mathbf{A}$ into the product of lower and upper triangular matrices $\mathbf{L}$ and $\mathbf{U}$. If $\mathbf{A}$ is symmetric and, further, positive definite, $\mathbf{U} = \mathbf{L}^T$ (i.e., Cholesky factorization); in the indefinite case $\mathbf{U} = \mathbf{DL}^T$, where $\mathbf{D}$ is block diagonal. Forward elimination followed by backward substitution completes the solution procedure for the right-hand side $\mathbf{b}$. The main complication when developing direct solvers for sparse matrices stems from the requirement to efficiently handle *fill-in*, i.e. limit the number of elements which change from an initial zero in the permuted $\mathbf{A}$ to a non-zero value in the factors $\mathbf{L}$ and $\mathbf{U}$.

Several algorithms and corresponding software codes implementing direct methods have appeared in recent years. Despite their individual peculiarities, sparse direct solvers operate in distinct phases, outlined as follows [9,6]:

1. An ordering phase that permutes rows and columns to ensure either that the factors will suffer little fill-in or to yield a matrix with special structure (e.g. block triangular). The choice of an ordering algorithm is crucial to the efficiency of any direct solver. Since computing an optimal ordering is NP-complete, various heuristics are used in practice [10,11].
2. An analysis or symbolic factorization phase concerned with analyzing the matrix's structure to determine a pivot sequence (optional) and the non-zero structures of the factors. A good pivot sequence should significantly reduce the memory requirements as well as the floating point operations count. Occasionally, this phase is combined with the ordering one.

3. A numerical factorization phase that uses the pivot sequence to factorize the matrix.
4. A solve phase that performs forward elimination followed by back substitution using the computed factors.

The first two phases are independent of the matrix's numerical values and depend only on its non-zero structure. For SPD matrices, the pivot sequence may be chosen based solely on the sparsity pattern, therefore the analysis phase involves no computation on real numbers. When implemented serially, the factorization is typically the most time-consuming of the different phases whereas the solve phase is generally significantly faster. Performance can be accelerated with parallel processing, employing the MPI-based implementations for distributed memory architectures that are available for some of the solvers. Another potentially useful feature of some implementations is their ability to work out-of-core, i.e. to hold the coefficient matrix and/or its factor in disk files, thereby substantially reducing the amount of main memory required by the solver and enabling it to tackle larger problems.

4 Implementation Issues

This section discusses several choices made during the design of `sparseLM` with the twofold objective of maximizing its performance while shielding the user from the algorithmic details associated with direct solvers. Since the optimization aspects of `sparseLM` are more or less standard, the emphasis is on sparseness and means of better taking advantage of it.

4.1 Sparse Matrix Formats

We start with a short description of general storage formats for sparse matrices. These formats make no assumptions regarding the sparsity structure and store non-zero elements by allocating contiguous memory storage for them along with some additional index information for keeping track where they fit into the full matrix. The Compressed Row Storage (CRS) format stores non-zero elements in row-major order, whereas Compressed Column Storage (CCS) adopts column-major ordering. More details can be found in [12].

4.2 Jacobian Representation and Computation

From a user's perspective, the provision of derivatives is one of the most bewildering practical aspects of non-linear least squares solvers. In the case of `sparseLM`, the Jacobian has been further assigned the role of specifying the sparsity pattern of the problem at hand: Its element at position (i, j) is non-zero if and only if measurement i depends upon variable j. In other words, the Jacobian can be thought of as a parameter - observation connection graph prescribing which (parameter, observation) pairs have direct interaction. `sparseLM` accepts Jacobians in either CRS or CCS format, allowing user applications to choose

the representation that is most natural to them. Jacobians can be hand-coded by the user or, more conveniently, generated through the use of automatic differentiation tools that work by systematically applying the chain rule to a given code segment. Additionally, `sparseLM` offers the possibility of numerically approximating the Jacobian using forward finite differences on data provided by successive invocations of f (cf. Eq (1)). In that case, only the sparsity pattern of the Jacobian should be specified by the user, whereas its numerical values are approximated by `sparseLM`. To reduce the total number of invocations, the Jacobian is approximated using a scheme that computes several of its columns with a single evaluation of f, exploiting its sparse structure as explained in [8], ch. 7. For a m-dimensional parameter vector, this scheme requires much fewer evaluations than the $m + 1$ ones that would be required by the naive approach of computing a single column of the Jacobian per evaluation of f. However, considering that they lead to faster convergence, analytic Jacobians should be preferred over approximated ones whenever possible.

4.3 Approximate Hessian Computation

A key aspect of `sparseLM`'s implementation concerns the efficient computation of the first order approximation to the Hessian, i.e. of matrix $\mathbf{J}^T\mathbf{J}$ in Eq (4). $\mathbf{J}^T\mathbf{J}$ is stored internally in the CCS format since this is the one most frequently employed among the implementations of direct sparse solvers. Multiplication of sparse matrices is considerably more challenging than that of dense ones, since the sparsity pattern of the product should first be discovered and then the operations for calculating the product's non-zeros should be carried out in a manner efficient with respect to the matrices memory storage format. An important observation concerning the sparsity pattern of $\mathbf{J}^T\mathbf{J}$ is that it does not change among LM iterations. Therefore, `sparseLM` makes its computation more efficient by computing its non-zero structure only once ignoring numerical cancellation and then reusing it when evaluating an actual product. Another performance improvement stems from exploiting symmetry. Thus, `sparseLM` computes only the lower triangular part of $\mathbf{J}^T\mathbf{J}$ and then copies it to the upper half, effectively reducing the number of computations roughly in half. In fact, even the copying operation can be skipped for some of the solvers since those that are designed for symmetric systems access only the triangular part of the coefficient matrix. Depending on whether the Jacobian $\mathbf{J}$ is supplied in CRS or CCS format, the product $\mathbf{J}^T\mathbf{J}$ is formed by an efficient technique that traverses $\mathbf{J}$ in a row-wise or column-wise fashion, respectively, ensuring that the pattern of accesses to its elements matches their physical layout in memory.

4.4 Choice of Linear Solver

As in the case of dense linear systems, it is generally advantageous in terms of performance to employ a direct sparse solver whose prerequisites closely match the intrinsic properties of the problem at hand. In the context of sparse non-linear least squares, the augmented normal equations matrix of Eq. (4) is SPD, thus

the direct solvers of choice are those designed to perform sparse Cholesky factorization. Still, more general solvers targeted to indefinite or even non-symmetric systems are clearly also usable. Advanced features such as provision for parallel or out-of-core processing should also be taken into consideration. Comparative evaluations of direct solvers in the literature indicate that no single one is universally the best [9]. For this reason, `sparseLM` includes interfaces to a wide variety of codes, the list of which currently consists of `LDL` [13], HSL's `MA57/MA47/MA27` [14], `PARDISO` [15], `SuperLU` [16], `TAUCS` [17], `UMFPACK` [18], `CSparse` [6], `CHOLMOD` [19] `SPOOLES` [20], and `MUMPS` [21]. Moreover, `sparseLM` has been designed so that expanding this list with more solvers in the future is straightforward. `CHOLMOD` [19], a set of routines for factorizing sparse SPD matrices, is used as `sparseLM`'s default solver. Regarding ordering, `CHOLMOD` automatically chooses between approximate minimum degree (AMD) [10] and graph-based nested dissection (METIS) [11]. Its overall performance was found to be quite competitive by the recent survey of Gould et al. [9].

Independently of the choice of a direct solver, its application in the context of the LM algorithm can be made more efficient by the following observation: During the course of the LM algorithm, several linear systems with identical sparsity patterns are repeatedly solved. Thus, as explained in section 3, the corresponding symbolic factorization is computed only once and then reused for numerically solving all subsequent linear systems.

5 Experimental Results

This section provides an experimental evaluation of `sparseLM`, applying it to three important problems in multiple view geometry and comparing its performance against alternative established approaches. The problems in question are bundle adjustment, trifocal tensor and homography estimation.

5.1 Euclidean Bundle Adjustment

In this section two sets of experiments are conducted, aiming at comparing the performance of `sba` [5] against that of `sparseLM` applied to Euclidean sparse Bundle Adjustment (BA). `sba` is our freely available package for BA that implements the partitioning scheme of [1] to solve the sparse augmented equations. It is heavily optimized and provides increased flexibility by allowing user-defined parameterizations for cameras and points as well as projection functions, thus being able to support a wide range of manifestations of the multiple view reconstruction problem. Being custom-written to match the sparsity structure of the BA problem, `sba` is expected to generally excel in performance. Nevertheless, it is instructive to examine when this conjecture holds and how close are the performances of the two approaches.

The first set of experiments relies on the eight test sequences also employed in [5]. Each experiment involves a set of 3D points whose image projections have been identified in a number of real images acquired by an intrinsically calibrated

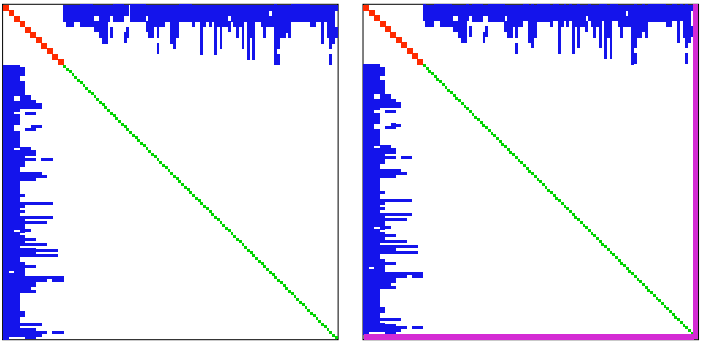
Table 1. Statistics for Euclidean BA using the `sparseLM` and `sba` packages: Total number of images, total number of variables, total number of objective function/Jacobian evaluations, total number of iterations and linear systems solved, elapsed execution time in seconds. Identical values for the user-defined minimization parameters have been used throughout all experiments.

			func/jac evals		**iter./sys. solved**		**exec. time**	
Sequence	**# imgs**	**# vars**	`sparseLM`	`sba`	`sparseLM`	`sba`	`sparseLM`	`sba`
"movi"	59	5688	20/18	20/20	18/20	20/20	4.26	3.69
"sagalassos"	26	5283	41/33	40/30	33/41	30/40	6.55	3.95
"arenberg"	22	4137	22/15	25/17	15/22	17/25	3.89	2.68
"basement"	11	981	33/22	32/23	22/33	23/32	0.57	0.28
"house"	10	1605	24/17	27/20	17/24	20/27	0.73	0.38
"maquette"	54	15945	29/21	30/23	21/29	23/30	13.20	7.98
"desk"	46	10542	28/20	32/22	20/28	22/32	8.51	6.06
"calgrid"	27	2328	25/19	21/20	19/25	20/21	15.61	8.58

moving camera. Estimates of the Euclidean 3D structure and camera motions have been computed using a sequential structure and motion estimation technique. Those estimates serve as starting points for bootstrapping refinements that are based on BA using `sba` and `sparseLM`. Camera motions corresponding to all but the first frame are defined relative to the initial camera location. The former is taken to coincide with the employed world coordinate frame. Camera rotations are parameterized by quaternions while translations and 3D points by 3D vectors.

Table 1 illustrates several statistics gathered from the application of `sba` and `sparseLM`-based Euclidean BA to the eight test sequences. Each row corresponds to a single sequence and columns are as follows: The first column corresponds to the total number of images that were employed in BA. The second column is the total number of motion and structure variables pertaining to the minimization. The third column shows the total number of objective function/Jacobian evaluations during BA for both approaches. The number of iterations needed for convergence and the total number of linear systems that were solved are shown in the fourth column. The last column shows the time in seconds elapsed during execution of BA. All experiments were conducted on an Intel P4@1.8 GHz running Linux and unoptimized BLAS. Both approaches converged to the same solutions for each sequence, therefore the corresponding final reprojection errors are not reported. As it is evident from the last column, BA with the aid of `sparseLM` is roughly at most two times slower than that employing `sba`. This is a remarkable result showing that the increased generality of `sparseLM` does not come at the price of performance.

At this point, it is enlightening to point out a few limitations of `sba` that are removed by the `sparseLM`-based approach to BA. `sba` assumes no coupling among the parameters for different cameras or different points. While this assumption is valid in many cases, there exist some situations where it imposes insurmountable restrictions. One such situation is illustrated in Fig. (1)(b) and concerns a sequence acquired with a camera having constant intrinsics that are to be refined



(a) (b)

Fig. 1. Visualization of the approximate Hessian's structure for two BA problems involving the "basement" sequence. (a) is 366×366 and arises in BA for camera motion and structure parameters (arranged in that order), (b) is 371×371 and corresponds to BA for camera motion, structure and constant across frames intrinsic parameters. Colored dots correspond to non-zero elements with red arising from motion-motion parameter pairs, green from point-point pairs, blue from motion-point pairs and magenta from motion-intrinsic, point-intrinsic and intrinsic-intrinsic pairs. `sba` cannot handle (b) due to the horizontal and vertical non-zero bands (in magenta) induced at its bottommost and rightmost parts by the sharing of intrinsic parameters. To improve the readability of graphs, only the first 100 points have been included in the BA.

via BA. In this case, the intrinsic calibration parameters must be shared by all images, violating `sba`'s assumption of independent camera parameters. Other examples involve the cases of employing inter-feature measurements such as distances or angles between points, coplanarity constraints on subgroups of points, articulated motion, etc. Another limitation stems from `sba`'s current implementation, which when forming the reduced bundle system assumes a dense structure for the Schur complement of the points submatrix in the approximate Hessian (i.e. the block matrix $\mathbf{S}$ in p.2:13 of [5]). This matrix, whose ij block is zero if images i and j have no points in common, is factored with a dense Cholesky decomposition to update the camera parameters. While it is reasonable to expect that for small problems such as the ones employed here most features are seen in all images and, therefore, matrix $\mathbf{S}$ is dense[1], for larger, more loosely connected image sets where each image only sees a small fraction of the features, $\mathbf{S}$ can become quite sparse. BA using `sparseLM` does not suffer from any of the aforementioned shortcomings since it treats the Jacobian (and therefore the Hessian) as a matrix with arbitrary sparseness, not needing to compute and factor $\mathbf{S}$.

To study the effect on performance of the density of matrix $\mathbf{S}$, a second set of experiments was designed. First, a fairly large, densely connected initial reconstruction consisting of 404 images, 77864 3D points and involving 236016 variables was obtained. The longest trajectory of image projections included in this

[1] The densities of the eight test sequences are at least 84% and in most cases 100% [5].

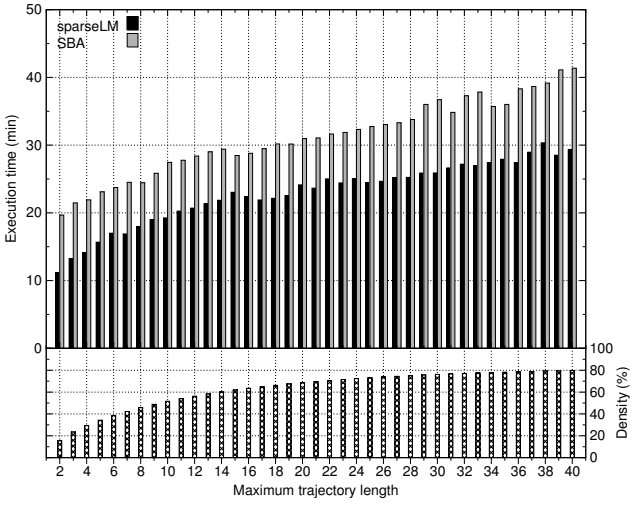
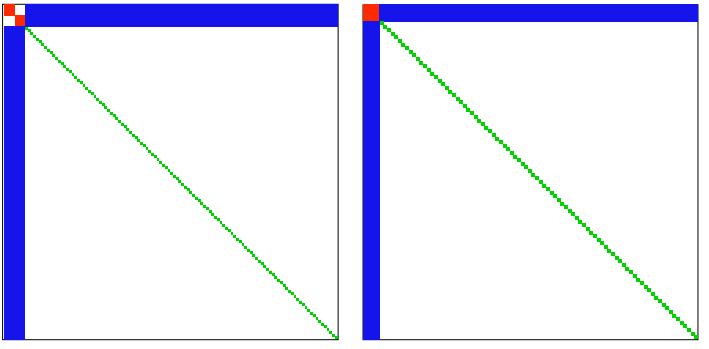


Fig. 2. Performance comparison of Euclidean BA for large reconstructions using `sparseLM` and `sba`: execution time (top) and **S** matrix density (bottom) vs. the maximum trajectory length l

reconstruction has a length of 40. Then, for a length limit l assuming values in $\{2, \ldots 40\}$, several other reconstructions were generated from the initial one by truncating projection trajectories that were more than l images long, taking care to avoid disconnecting the camera network. In this manner, the generated reconstructions differ only in the densities of their point submatrices, thus providing a basis for comparing the performance of sparseLM-based BA against that of `sba` for varying densities of the matrix **S**. The top part of Fig. 2 summarizes the execution times of the two alternatives to BA applied to the 39 generated reconstructions, whereas the bottom part shows their corresponding **S** matrix densities. Clearly, the performance difference between `sparseLM` and `sba` is reversed in favor of the former and is more pronounced for less connected image sets. As has been also observed in [5], this difference stems from the fact that the computation of the dense Cholesky decomposition of **S** has time complexity $O(N^3)$ and thus becomes appreciable for large N ($N = 2424$ in this particular case). Furthermore, the time spent by `sparseLM` for carrying out the the symbolic factorization once in the beginning pays off by enabling it to numerically compute the sparse Cholesky in less time at each subsequent iteration. The downside of using `sparseLM` is that it requires about two to three times more memory than `sba`. This is because direct sparse solvers require additional memory to store the symbolic factorization, whose size depends on the matrix's sparsity structure.

5.2 Trifocal Tensor Estimation

The trifocal tensor $\mathcal{T}$ encapsulates all geometric relations among three images that are independent of scene structure. According to the "Gold Standard"



(a) (b)

Fig. 3. (a) Hessian structure for trifocal tensor estimation involving the 114 3D points visible in the first 3 frames of the "basement" sequence. Color coding is as in Fig. 1. (b) Hessian structure for homography refinement involving 88 image point pairs. Red dots correspond to homography-homography variable pairs, green to point-point pairs and blue to homography-point pairs.

algorithm for obtaining the Maximum Likelihood Estimate of $\mathcal{T}$ [1], p.397 from triplets of corresponding points $\mathbf{x}_i$, $\mathbf{x}_i^{'}$ and $\mathbf{x}_i^{''}$, the procedure proceeds as follows. First, a geometrically valid estimate of $\mathcal{T}$ is computed with a linear algorithm that minimizes the algebraic error and a canonical triplet of camera matrices is recovered from this estimate. Subsidiary variables corresponding to 3D points $\mathbf{X}_i$ are then introduced and initialized via triangulation. $\mathcal{T}$ is parametrized by the elements of the camera matrices $\mathbf{P}^{'}$ and $\mathbf{P}^{''}$. Subsequently, the cost function

$$\sum_i d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 \; + \; d(\mathbf{x}_i^{'}, \hat{\mathbf{x}}_i^{'})^2 \; + \; d(\mathbf{x}_i^{''}, \hat{\mathbf{x}}_i^{''})^2 \tag{5}$$

is minimized over the 3D points $\mathbf{X}_i$ and the elements of the two camera matrices $\mathbf{P}^{'}$, $\mathbf{P}^{''}$ with $\hat{\mathbf{x}}_i = [\mathbf{I} \mid \mathbf{0}]\mathbf{X}_i$, $\hat{\mathbf{x}}_i^{'} = \mathbf{P}^{'}\mathbf{X}_i$ and $\hat{\mathbf{x}}_i^{''} = \mathbf{P}^{''}\mathbf{X}_i$. For n 3D points, the minimization involves $3n + 24$ variables and amounts to a sparse problem (cf. Fig. 3(a)) solvable by `sparseLM`. Finally, the three correlation slices of $\mathcal{T}$ are set to $\mathbf{T}_i = \mathbf{a}_i\mathbf{b}_4^T - \mathbf{a}_4\mathbf{b}_i^T$, $i = 1 \ldots 3$, where $\mathbf{a}_i$, $\mathbf{b}_i$ are respectively the i-th columns of the refined camera matrices $\mathbf{P}^{'} = [\mathbf{A} \,|\, \mathbf{a}_4]$, $\mathbf{P}^{''} = [\mathbf{B} \,|\, \mathbf{b}_4]$. The tensor estimated in this manner satisfies by construction the trilinear constraints for a triplet of refined corresponding points.

The reprojection error of (5) is quite complex and minimizing it involves a large number of parameters. An approximate solution to overcome this is to substitute (5) with the so-called Sampson error [1], p.98, which is the distance to the first order approximation of the algebraic variety defined by the trilinear constraints. Minimization of the sum of Sampson errors for all points relates to only 24 variables, appendix B of [22] provides more details. While it might seem reasonable to expect that the fewer variables of the Sampson approximation will result in faster performance, it is demonstrated next that an application of `sparseLM` performs faster and is more accurate.

Table 2. Statistics for tensor estimation using `sparseLM` (*sLM*), the Sampson approximation (*SA*) and dense LM (*dLM*) approaches. The columns are as follows: Total number of variables for *sLM* & *dLM*, average initial transfer error in pixels, average final transfer error in pixels, total number of objective function/Jacobian evaluations, total number of iterations and linear systems solved, elapsed execution time in seconds. Again, identical values for the user-defined minimization parameters have been used throughout all experiments.

	# vars	**initial**	**final error**		**func/jac evals**			**iter./sys. solved**			**exec. time**		
Sequence	sLM & dLM	**error**	sLM & dLM	SA	sLM	SA	dLM	sLM	SA	dLM	sLM	SA	dLM
"movi"	729	0.330	0.286	0.320	32/30	59/2	38/30	30/32	10/11	30/38	0.42	1.92	43.38
"sagalassos"	681	0.745	0.335	0.737	38/31	80/2	39/34	31/38	31/32	34/39	0.41	2.34	43.52
"arenberg"	960	0.428	0.357	0.428	35/29	41/1	35/31	29/35	16/17	31/35	0.60	1.73	99.83
"basement"	366	0.472	0.397	0.459	35/28	159/4	32/29	28/35	62/63	29/32	0.18	2.53	5.01
"house"	636	0.393	0.367	0.389	60/49	39/1	65/52	49/60	14/15	52/65	0.58	1.07	43.03
"maquette"	1041	0.771	0.429	0.739	37/33	83/2	37/31	33/37	34/35	31/37	0.75	3.81	133.61
"desk"	594	0.545	0.511	0.545	37/30	32/1	34/31	30/37	7/8	31/34	0.32	0.83	23.99
"calgrid"	2097	0.420	0.155	0.320	39/35	62/2	41/34	35/39	13/14	34/41	1.83	5.75	1151.75

The cost function (5) was minimized with `sparseLM` and `levmar` [23], which includes a dense version of the LM algorithm implemented by `sparseLM`. These two approaches are labeled *sLM* and *dLM*, respectively. Furthermore, the total error of the Sampson approximation was minimized with `levmar` using a secant variant of the dense LM; this approach is labeled *SA*. *dLM* serves as a reference for the time savings achieved by *SA* and *sLM*. The three alternative approaches were applied to the estimation of the trifocal tensor corresponding to the first three frames of each sequence used in section 5.1 and the related statistics are presented in Table 2.

The performance of the three approaches is evaluated for accuracy and efficiency, using the average tensorial transfer error for all points in all three frames and the total execution time, respectively. *sLM* and *dLM* employ the same objective function and, therefore, perform identically with respect to accuracy. However, *dLM* is at least two orders of magnitude slower. On the other hand, *SA* is less accurate than *sLM* and, being between 2 to 14 times slower, is also considerably less efficient. The reasons for the worse performance of *SA* can be partly attributed to the fact that the computation of the Sampson error for each point triplet calls for a costly SVD operation to estimate the pseudoinverse of a 9×9 rank 3 matrix [22]. Furthermore, the Jacobian of the Sampson error is too complicated to express analytically, which necessitates its approximation using finite differences that raise the total number of performed SVDs even further. As a matter of fact, the motivation for using a secant variant of dense LM with Broyden's rank one update for minimizing the Sampson error was to ease down the overhead of finite differentiation. The overall superior performance of *sLM* combined with the restrictive assumption made by the Sampson approximation according to which the variety of trilinear constraints has to be well approximated by a first order expansion in the vicinity of the current estimate, clearly suggests *sLM* as the preferred alternative for tensor estimation.

5.3 Homography Estimation

A homography is a general plane to plane projective transformation that is represented by a non-singular homogeneous 3×3 matrix $\mathbf{H}$. Assuming that a set of corresponding coplanar image point pairs $\mathbf{x}_i,\ \mathbf{x}_i^{'}$ is available, the "Gold Standard" algorithm for estimating $\mathbf{H}$ is as follows (cf. [1], p.114): First, an initial estimate is computed using a linear normalized DLT algorithm embedded in a robust regression framework to safeguard against outliers. Then, considering only the inliers, the initial estimate is used as a starting point for minimizing the following geometric cost over $\mathbf{H}$ and the subsidiary points $\hat{\mathbf{x}}_i$:

$$\sum_i d(\mathbf{x}_i, \hat{\mathbf{x}}_i)^2 \ + \ d(\mathbf{x}_i^{'}, \mathbf{H}\hat{\mathbf{x}}_i)^2. \tag{6}$$

The minimization corresponds to a sparse problem which involves $9 + 2n$ variables, n being the number of inlying point pairs (cf. Fig. 3(b)). In a manner similar to the estimation of the trifocal tensor, the geometric error of (6) can be approximated with the Sampson error involving 9 variables.

The performance of `sparseLM` minimizing (6) was compared against those of a dense LM algorithm utilized to minimize (6) and the Sampson approximation. Five experiments were carried out using around 900 SIFT keypoints extracted and matched between successive pairs from the six images of the "graffiti" sequence. Although lack of space prevents the inclusion of detailed statistics, it is noted that the performance of `sparseLM` was between 455 to 886 times better than that of the dense LM algorithm minimizing (6) and between only 1.1 to 1.6 times worse than that of the dense LM applied to the Sampson approximation. As expected, minimizing (6) was slightly more accurate than employing the corresponding Sampson approximation.

6 Conclusions

A general-purpose, computationally efficient implementation of sparse non-linear least squares optimization is beneficial to a wide range of vision tasks. This paper has presented an overview of `sparseLM`, a such open source implementation and has demonstrated its versatility and effectiveness in different practical situations. Considering that its applicability extends beyond geometric vision, `sparseLM` can potentially prove invaluable to a variety of research fields and disciplines.

References

1. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge (2004)
2. Sturm, P., Maybank, S.: A Method for Interactive 3D Reconstruction of Piecewise Planar Objects from Single Images. In: Proc. of BMVC 1999, pp. 265–274 (1999)
3. Sinha, S., Pollefeys, M.: Pan-Tilt-Zoom Camera Calibration and High-Resolution Mosaic Generation. Computer Vision and Image Understanding Journal 103, 170–183 (2006)

4. Triggs, B., McLauchlan, P., Hartley, R., Fitzgibbon, A.: Bundle Adjustment – A Modern Synthesis. In: Proc. of the Int'l Workshop on Vision Algorithms: Theory and Practice, pp. 298–372 (1999)
5. Lourakis, M.A., Argyros, A.: SBA: A Software Package for Generic Sparse Bundle Adjustment. ACM Trans. Math. Software 36, 1–30 (2009)
6. Davis, T.: Direct Methods for Sparse Linear Systems. In: Fundamentals of Algorithms. SIAM, Philadelphia (2006)
7. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM, Philadelphia (2003)
8. Nocedal, J., Wright, S.: Numerical Optimization. Springer, New York (1999)
9. Gould, N., Scott, J., Hu, Y.: A Numerical Evaluation of Sparse Direct Solvers for the Solution of Large Sparse Symmetric Linear Systems of Equations. ACM Trans. Math. Softw. 33, 10 (2007)
10. Amestoy, P., Davis, T., Duff, I.: Algorithm 837: AMD, an Approximate Minimum Degree Ordering Algorithm. ACM Trans. Math. Softw. 30, 381–388 (2004)
11. Karypis, G., Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM J. Sci. Comput. 20, 359–392 (1998)
12. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., der Vorst, H.V.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd edn. SIAM, Philadelphia (1994)
13. Davis, T.: Algorithm 849: A Concise Sparse Cholesky Factorization Package. ACM Trans. Math. Softw. 31, 587–591 (2005)
14. Duff, I.: MA57—A Code for the Solution of Sparse Symmetric Definite and Indefinite Systems. ACM Trans. Math. Softw. 30, 118–144 (2004)
15. Schenk, O., Gartner, K.: Solving Unsymmetric Sparse Systems of Linear Equations with PARDISO. J. of Future Generation Computer Systems 20, 475–487 (2004)
16. Demmel, J., Eisenstat, S., Gilbert, J., Li, X., Liu, J.: A Supernodal Approach to Sparse Partial Pivoting. SIAM J. Matrix Analysis and Applications 20, 720–755 (1999)
17. Rotkin, V., Toledo, S.: The Design and Implementation of a New Out-of-Core Sparse Cholesky Factorization Method. ACM Trans. Math. Softw. 30, 19–46 (2004)
18. Davis, T.: Algorithm 832: UMFPACK, An Unsymmetric-Pattern Multifrontal Method. ACM Trans. Math. Softw. 30, 196–199 (2004)
19. Chen, Y., Davis, T., Hager, W., Rajamanickam, S.: Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. ACM Trans. Math. Softw. 35, 1–14 (2008)
20. Ashcraft, C., Grimes, R.: SPOOLES: An Object-Oriented Sparse Matrix Library. In: Proc. of SIAM Conf. on Parallel Processing for Scientific Computing (1999)
21. Amestoy, P., Duff, I., Koster, J., L'Excellent, J.Y.: A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. SIAM Journal of Matrix Analysis and Applications 23, 15–41 (2001)
22. Torr, P., Zisserman, A.: Robust Parameterization and Computation of the Trifocal Tensor. Image and Vision Computing 15, 591–605 (1997)
23. Lourakis, M.: levmar: Levenberg-Marquardt Nonlinear Least Squares Algorithms in C/C++ (2004), `http://www.ics.forth.gr/~lourakis/levmar/`