Least-Squares Minimization Under Constraints
EPFL Technical Report # 150790

P. Fua, IC-CVLab, EPFL
A. Varol, IC-CVLab, EPFL
R. Urtasun, TTI Chicago
M. Salzmann, TTI Chicago

Unconstrained Least-Squares minimization is a well-studied problem. For example, the Levenberg-Marquardt is extremely effective and numerous implementations are readily available [Press *et al.*, 1992, Lourakis, 2009].

These algorithms are, however, not designed to perform least-squares minimization under hard constraints. This short report outlines two very simple approaches to doing this to solve problems such as the one depicted by Fig. 1. The first relies on standard Lagrange multipliers [Boyd and Vandenberghe, 2004]. The second is inspired by inverse kinematics techniques [Baerlocher and Boulic, 2004] and has been demonstrated for the purpose of monocular 3D surface modeling in [Salzmann and Fua, 2010]. In both cases, we outline matlab implementations.

This write-up is intended more as a working document than as a pure research report. Comments and suggestions are welcome.

Figure 1: Modeling the deformations of a main sail by minimizing an objective function under the constraint that the length of the mesh edges must be preserved. The black circles in the leftmost image are targets that were automatically detected and used to establish correspondences with a reference configuration. The algorithm can estimate the 3D deformations at a rate of approximately 10 Hz on a standard PC.

1 Using Lagrange Multipliers

We begin by discussing the enforcement of equality constraints. We then show how to extend the formalism to handle inequality constraints as well.

1.1 Equality Constraints

We seek to minimize

$$\frac{1}{2}||F(X)||^2 \text{ subject to } C(X) = 0 \ , \tag{1}$$

where X is an $n \times 1$ vector, $F(X)$ is a $r \times 1$ vector of image measurements, and $C(X)$ is an $m \times 1$ vector of constraints. We define the Lagrangian as

$$L(X, \Lambda) = \frac{1}{2}||F(X)||^2 + \Lambda^t C(X) \ , \tag{2}$$

where Λ is the $m \times 1$ vector of Lagrange multipliers. Linearizing F and C around X yields

$$\begin{aligned} L(X + dX, \Lambda) &= \frac{1}{2}||F(X) + JdX||^2 + \Lambda^t(C(X) + AdX) \ , \\ &= \frac{1}{2}(F(X) + JdX)^t(F(X) + JdX) + \Lambda^t(C(X) + AdX) \ , \end{aligned} \tag{3}$$

where J is the $r \times n$ Jacobian matrix of F and A is the $m \times n$ Jacobian matrix of C. Finding a solution for to this problem can be done by solving for the Karush-Kuhn-Tucker (KKT) conditions. The first KKT condition is *stationarity*: At the minimum of L with respect to dX, we must have

$$\begin{aligned} 0 &= \frac{\partial L}{\partial dX}(X + dX, \Lambda) \ , \\ &= J^t J dX + J^t F + A^t \Lambda \ , \end{aligned} \tag{4}$$

and therefore

$$dX = -(J^t J)^{-1}(J^t F + A^t \Lambda) \ . \tag{5}$$

The second KKT condition is *primal feasibility*, which translates into finding a solution $X + dX$ that satisfies the constraints. To this end, we must have

$$\begin{aligned} 0 &= C(X + dX) \ , \\ &= C(X) + AdX \ , \\ &= C(X) - A((J^t J)^{-1}(J^t F + A^t \Lambda)) \ , \\ &= C(X) - B\Lambda - Ab \ , \end{aligned} \tag{6}$$

where $B = A(J^t J)^{-1} A^t$ and $b = (J^t J)^{-1} J^t F$. We can now compute Λ by solving

$$B\Lambda = C(X) - Ab \tag{7}$$

and dX by injecting the resulting vector into Eq. 5, which can be rewritten as

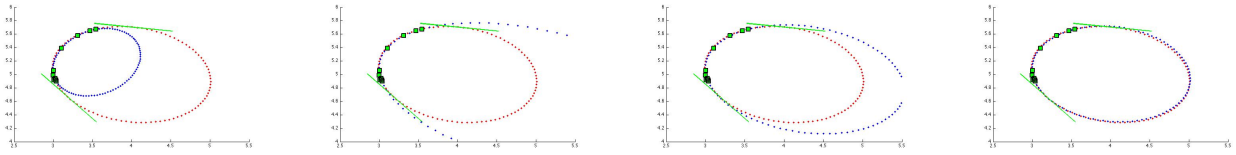
$$dX = -(b + (J^t J)^{-1} A^t \Lambda) \ . \tag{8}$$

A naive MATLAB implementation is listed in Fig. 3. It involves explicitly inverting the $n \times n$ $J^t J$ matrix, which is of computational complexity $0(n^3)$ and therefore slow for large values of n. To speed things up, it is possible to replace the explicit computation of the inverse by the solving of 3 $n \times n$ linear systems, each one being of complexity $0(n^2)$, as shown in Fig. 4.

Alternatively, if the matrices are sparse, it is more efficient to simultaneously solve equations 4 and 6. This results in the $(n+m) \times (n+m)$ linear system

$$\begin{bmatrix} J^t J & A^t \\ A & 0 \end{bmatrix} \begin{bmatrix} dX \\ \Lambda \end{bmatrix} = - \begin{bmatrix} J^t F \\ C(X) \end{bmatrix} \tag{9}$$

and the corresponding MATLAB implementation listed in Fig. 5. In theory, the complexity is $0((n+m)^2)$ but, in practice, much less if the matrices are sparse. Fig. 2 illustrates this computation for the purpose of fitting an ellipse to noisy data points under tangency constraints. In practice, this is a standard approach to constraining the least-square problems that arise when performing bundle-adjustment [Triggs *et al.*, 2000].



(a) (b) (c) (d)

Figure 2: Fitting an ellipse to noisy data points under tangency constraints. (a) Given a ground-truth ellipse shown in red, we fit in closed-form the ellipse shown in blue to noisy point samples overlaid as green squares. (b,c,d) Enforcing tangency to the green lines by recursively solving Eq. 9 to compute the dX increment, starting from the unconstrained solution represented by a 5-dimensional X vector.

```
function dX=ComputeStep(X,Cnst,Data)
% Jacobian and Vector of objective function
[J,F]=Data.Jacob(X);
% Jacobian and Vector of constraints
[A,C]=Cnst.Jacob(X);
% Compute auxiliary matrices
JtJ=inv(J'*J); B=A*JtJ*A'; b=JtJ*J'*F;
% Compute the Lagrange multipliers
Lambda=B\(C-A*b);
% Compute the iteration step
dX=-(b+JtJ*A'*Lambda);
```

Figure 3: Naive MATLAB code that implements the method of Section 1 by sequentially solving Eqs. 6 and then 4.

1.2 Inequality Constraints

The above formulation can easily be extended to also handle inequality constraints by using slack variables to reformulate them as equalities. To this end, we introduce a m-dimensional vector S of slack

```
function dX=ComputeStep(X,Cnst,Data)
% Jacobian and Vector of objective function
[J,F]=Data.Jacob(X);
% Jacobian and Vector of constraints
[A,C]=Cnst.Jacob(X);
% Compute auxiliary matrices
JtJ=J'*J; B=A*(JtJ\A'); b=(JtJ\J')*F;
% Compute the Lagrange Multipliers
Lambda=B\(C-A*b);
% Compute the iteration step
dX=-(b+(JtJ\(A'*Lambda)));
```

Figure 4: Improved MATLAB code that implements the method of Section 1 by sequentially solving Eqs. 6 and 4 without explicit matrix inversion.

```
function dX=ComputeStep(X,Cnst,Data)
% Jacobian and Vector of objective function
[J,F]=Data.Jacob(X);
% Jacobian and Vector of constraints
[A,C]=Cnst.Jacob(X);
% Get dimensions
m=size(A,1); n=size(A,2);
% Simultaneously Compute both the iteration step and Lagrange multipliers.
dXLambda=-[J'*J, A'; A , zeros(m,m)]\[J'*F;C]; dX=dXLambda(1:n);
```

Figure 5: MATLAB code that implements the method of Section 1 by solving the single larger linear system of Eq. 9 without explicit matrix inversion.

variables, one for each inequality, and exploit the fact that

$$C(X) \leq 0 \Leftrightarrow C(X) + S \odot S = 0 \,, \tag{10}$$

where $\odot$ is the element-wise Hadamard product. This lets us rewrite the problem of Eq. 1 as one of minimizing

$$\frac{1}{2}||F(X)||^2 + \frac{\mu}{2}\|S\|^2 \text{ subject to } C(X) + S \odot S = 0 \;\;, \tag{11}$$

with respect to X and S simultaneously. Note that we added a regularizer on S that tends to favor solutions that are closer to satisfying the equality constraints $C(X) = 0$, which can prove useful in cases [Salzmann and Fua, 2011] such as the one depicted by Fig. 1. If the parameter μ is set to 0, the constraints behave like true inequalities without preference for any specific solution.

Following the same approach as before, we can linearize the different terms of our problem around the current solution (X, S), and write its Lagrangian as

$$L(dX, dS, \Lambda) = \frac{1}{2}||F(X) + JdX||^2 + \frac{\mu}{2}\|S + dS\|^2 + \Lambda^t \left(C(X) + AdX + S \odot S + 2diag(S)dS\right) \;, \tag{12}$$

where $diag(S)$ is an $m \times m$ diagonal matrix with S on its diagonal. We can then again solve for the KKT conditions to obtain the optimal solution for dX, dS, and Λ. Stationarity with respect to dX yields the same equation as in the equality constraints case, which we repeat here for convenience,

$$J^t JdX + J^t F + A^t \Lambda = 0 \,. \tag{13}$$

Similarly, stationarity of the Lagrangian with respect to dS yields

$$\mu dS + \mu S + R^t \Lambda = 0 \ , \tag{14}$$

where $R = 2 * diag(S)$. Finally, primal feasibility is expressed as the solution to the constraints, which yields

$$C(X) + AdX + P + RdS = 0 \ , \tag{15}$$

where $P = S \odot S$. The three equations above can be grouped in a single $(n + 2m) \times (n + 2m)$ linear system of the form

$$\begin{bmatrix} J^t J & 0 & A^t \\ 0 & \mu I & R^t \\ A & R & 0 \end{bmatrix} \begin{bmatrix} dX \\ dS \\ \Lambda \end{bmatrix} = - \begin{bmatrix} J^t F \\ \mu S \\ C(X) + P \end{bmatrix} \ , \tag{16}$$

whose solution can be obtained similarly as the one of Eq. 9. As discussed above, the parameter μ can be set to zero but, in practice, doing so results in poor convergence due to the linear system of Eq. 16 being ill-conditioned. It is therefore preferable to set μ to a small but non zero value if one wishes the system to enforce the $C(X) \leq 0$ constraints without unduly favoring solutions such that $C(X) \approx 0$.

If instead of seeking a solution that obeys a single inequality we seek the one that yields values between two bounds, we can separate the bounds into two sets of inequalities and use the same approach as before. This yields the transformation

$$B_0 \leq C(X) \leq B_1 \Leftrightarrow C(X) - S_0 \odot S_0 = B_0 \ , \ C(X) + S_1 \odot S_1 = B_1 \ . \tag{17}$$

It can then easily be verified that the solution to the corresponding linearized constrained minimization problem can be obtained by solving

$$\begin{bmatrix} J^t J & 0 & 0 & A^t & A^t \\ 0 & \mu I & 0 & -R_0^t & 0 \\ 0 & 0 & \mu I & 0 & R_1^t \\ A & -R_0 & 0 & 0 & 0 \\ A & 0 & R_1 & 0 & 0 \end{bmatrix} \begin{bmatrix} dX \\ dS_0 \\ dS_1 \\ \Lambda_0 \\ \Lambda_1 \end{bmatrix} = - \begin{bmatrix} J^t F \\ \mu S_0 \\ \mu S_1 \\ C_0(X) - P_0 \\ C_1(X) + P_1 \end{bmatrix} \ , \tag{18}$$

where $C_0(X) = C(X) - B_0$, $C_1(X) = C(X) - B_1$, $P_0 = S_0 \odot S_0$, $P_1 = S_1 \odot S_1$, $R_0 = 2diag(S_0)$, and $R_1 = 2diag(S_1)$. The parameter μ can now be used to set a prerefence for solutions whose value of $C(X)$ is close to the middle of the interval defined by the bounds. Defining other preferences can be achieved by introducing two such parameters, one to regularize S_0 and the other to regularize S_1.

2 Minimizing in the Null Space of the Constraints

Given the current estimate for X, we can iteratively find the increment dX such that $X + dX$ both satisfies the linearized constraints and minimizes $||F(X + dX)||^2$. In other words, at each iteration, we seek dX such that

$$\begin{aligned} C(X + dX) = 0 \ &\Rightarrow \ AdX = -C(X) \ , \\ &\Rightarrow \ dX = -A^\dagger C(X) + (I - A^\dagger A) dZ \ , \end{aligned} \tag{19}$$

where A is the $m \times n$ Jacobian matrix of C, $A^\dagger$ its $n \times m$ pseudo-inverse, and dZ an arbitrary $n \times 1$ vector. When there are fewer constraints than variables, which means that $m < n$ and is the normal

situation, $A^\dagger$ can be computed as $\lim_{\delta \to 0} A^t (AA^t + \delta I)^{-1}$, which involves solving an $m \times m$ linear system and can be done even if AA^t itself is non invertible. $P = I - A^\dagger A$ is known as the projection operator, or projector, onto the kernel of A.

Let $dX_0 = -A^\dagger C(X)$ be the minimum norm solution of Eq. 19. We choose dZ by minimizing

$$||F(X + dX_0 + PdZ)||^2 \approx ||F(X) + J(dX_0 + PdZ)||^2 \ , \tag{20}$$

or, equivalently, solving in the least square sense

$$JPdZ = -(F(X) + JdX_0) \ . \tag{21}$$

Since J is an $r \times n$ matrix, this implies solving an $n \times n$ system with complexity $0(n^2)$, which is the dominant cost assuming that $n > m$. This is implemented by the MATLAB of Fig. 6 in which we use $\delta = 1e - 10$ to compute the pseudo-inverses. The convergence properties of the algorithm can be improved by scaling dZ to guarantee that $||F(X + dX_0 + PdZ)||$ decreases as is done in the Levenberg-Marquardt algorithm, which yields the slightly more involved code of Fig. 7.

A similar subspace minimization scheme was proposed in [Shen *et al.*, 2009]. Instead of using the projector P to formulate the minimization problem of Eq. 20 in terms of the dZ vector whose dimensionality is the same as that of X, it involves performing a Singular Value Decomposition of A to find the $n - m$ Singular Vectors associated to zero values–assuming that the constraints are all independent–and writing the dX increment as dX_0 plus a weighted sum of these singular vectors. The unknown become the weights and the minimization can be achieved by solving a linear smaller system, at the cost however of performing an SVD that the scheme of Eqs. 19 and 21 does not require.

This projector-based scheme has been extensively tested for 3D sail reconstruction purposes, as depicted by Fig. 1. In those input images, the black circles on the sails are automatically detected and put into correspondence with the same circles on a reference image in which the shape of the sail is known *a priori* and represented by a triangulated mesh. Recovering the unknown 3D coordinates of the deformed mesh vertices can then be achieved by solving a linear system of the form

$$MX = 0, \tag{22}$$

where X is the vector of all x, y, and z coordinates of the mesh describing the sail expressed in the camera coordinate system and M is $r \times n$ matrix where r is twice the number of correspondences and n the dimension of X. It can be shown that the rank of M is at most $\frac{2n}{3}$ and that constraints are therefore required to guarantee a unique non-degenerate solutionh [Salzmann and Fua, 2010]. Furthermore, because the correspondences are noisy, we minimize

$$||MX||^2 \text{ subject to } C(X) = 0 \ \ , \tag{23}$$

where $C(X)$ is the vector of changes in the length of triangulation edges. In other words, we force the mesh edges to retain their reference length. In this case, $J = M$ and Eq. 21 simplifies to

$$MPdZ = -M(X + dX_0) \ . \tag{24}$$

Because the deformations are relatively small, the optimization typically converges to a local minimum in about 10 iterations, which allows for real-time performance.

Furthermore, even better results can be obtained by replacing the length equality constraints by inequality constraints that state that the Euclidean distance between vertices is bounded by the known

geodesic constraints in reference configuration. This only involves a trivial modification of the algorithm above: At each iteration, only currently active constraints are taken into account in the computation of $C(X)$ and its Jacobian A [Salzmann and Fua, 2010]. The results are very similar to those obtained using the slightly more involved approach to handling inequalities by introducing slack variables discussed in Section 1.

References

[Baerlocher and Boulic, 2004] P. Baerlocher and R. Boulic. An Inverse Kinematics Architecture for Enforcing an Arbitrary Number of Strict Priority Levels. *The Visual Computer*, 2004.

[Boyd and Vandenberghe, 2004] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Lourakis, 2009] M. Lourakis. Levmar : Levenberg-Marquardt Nonlinear Least Squares Algorithms in C/c++, 2009. http://www.ics.forth.gr/ lourakis/levmar/.

[Press *et al.*, 1992] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes, the Art of Scientific Computing*. Cambridge U. Press, 1992.

[Salzmann and Fua, 2010] M. Salzmann and P. Fua. *Deformable Surface 3D Reconstruction from Monocular Images*. Morgan-Claypool Publishers, 2010.

[Salzmann and Fua, 2011] M. Salzmann and P. Fua. Linear Local Models for Monocular Reconstruction of Deformable Surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):931–944, May 2011.

[Shen *et al.*, 2009] S. Shen, W. Shi, and Y. Liu. Monocular 3D Tracking of Inextensible Deformable Surfaces Under L2-Norm. In *Asian Conference on Computer Vision*, 2009.

[Triggs *et al.*, 2000] B. Triggs, P. Mclauchlan, R. Hartley, and A. Fitzgibbon. Bundle Adjustment – a Modern Synthesis. In *Vision Algorithms: Theory and Practice*, pages 298–372, 2000.


```
function dX=ComputeStep(X,Cnst,Data)
% Jacobian and Vector of objective function
[J,F]=Data.Jacob(X);
% Jacobian and Vector of constraints
[A,C]=Cnst.Jacob(X);
% Compute pseudo inverse and projector
AInv=PseudoInverse(A,1e-10); Proj=eye(size(A,2))-AInv*A;
% Compute projection increment to satisfy constraints only
dX=-AInv*C;
% Compute objective function increment
JP=J*Proj; dZ=LsqSolve(JP,F+J*dX,1e-10); dX=dX-Proj*dZ;
```

Figure 6: Simple MATLAB code that implements the method of Section 2.

```
function dX=ComputeStep(X,Cnst,Data)
% Jacobian and Vector of objective function
[J,F]=Data.Jacob(X);
% Jacobian and Vector of constraints
[A,C]=Cnst.Jacob(X);
% Compute pseudo inverse and projector
AInv=PseudoInverse(A,1e-10); Proj=eye(size(A,2))-AInv*A;
% Compute projection increment to satisfy constraints only
dX=-AInv*C;
% Compute objective function increment
JP=J*Proj;
dZ=LsqSolve(JP,F+J*dX,1e-10);
% Scale dZ to ensure that the objective function decreases.
Scale=1.0;
BestN=inf;
BestX=[];
for i=1:10
 ScaledX=dX-Scale*Proj*dZ;
 ScaledF=Data.Vector(X+ScaledX);
 ScaledN=norm(ScaledF);
 if(ScaledN<norm(F))
  dX=ScaledX;
  return;
 else if(ScaledN<BestN)
  BestN=ScaledN;
  BestX=ScaledX;
 end
 Scale=Scale/2.0;
 end
end
dX=BestX;
```

Figure 7: Slightly refined MATLAB code that implements the method of Section 2.