RISE: An Incremental Trust-Region Method for Robust Online Sparse Least-Squares Estimation

The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.


As Published: http://dx.doi.org/10.1109/tro.2014.2321852

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Persistent URL: http://hdl.handle.net/1721.1/97576

Version: Author’s final manuscript: final author’s manuscript post peer review, without publisher’s formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike
RISE: An Incremental Trust-Region Method for Robust Online Sparse Least-Squares Estimation

David M. Rosen, Student Member, IEEE, Michael Kaess, Member, IEEE,
and John J. Leonard, Fellow, IEEE

Abstract—Many point estimation problems in robotics, computer vision and machine learning can be formulated as instances of the general problem of minimizing a sparse nonlinear sum-of-squares objective function. For inference problems of this type, each input datum gives rise to a summand in the objective function, and therefore performing online inference corresponds to solving a sequence of sparse nonlinear least-squares minimization problems in which additional summands are added to the objective function over time. In this paper we present Robust Incremental least-Squares Estimation (RISE), an incrementalized version of the Powell’s Dog-Leg numerical optimization method suitable for use in online sequential sparse least-squares minimization. As a trust-region method, RISE is naturally robust to objective function nonlinearity and numerical ill-conditioning, and is provably globally convergent for a broad class of inferential cost functions (twice-continuously differentiable functions with bounded sublevel sets). Consequently, RISE maintains the speed of current state-of-the-art online sparse least-squares methods while providing superior reliability.

Index Terms—Sparse least-squares minimization, online estimation, SLAM, computer vision, machine learning

I. INTRODUCTION

M ANY point estimation problems in robotics, computer vision and machine learning can be formulated as instances of the general problem of minimizing a sparse nonlinear sum-of-squares objective function; for example, the archetypal problems of full (smoothing) simultaneous localization and mapping (SLAM) [1] (in robotics), bundle adjustment (BA) [2], [3] (in computer vision), and sparse (kernel) regularized least-squares classification and regression [4], [5] (in machine learning) all belong to this class. For inference problems of this type, each input datum gives rise to a summand in the objective function, and therefore performing online inference (in which the data is collected sequentially and the estimate updated after the incorporation of each new datum) corresponds to solving a sequence of sparse least-squares minimization problems in which additional summands are added to the objective function over time.

In practice, these online inference problems are often solved by computing each estimate in the sequence as the solution of an independent minimization problem using standard sparse least-squares techniques (most commonly Levenberg-Marquardt [6]–[8]). While this approach is general and produces good results, it is computationally expensive, and does not exploit the sequential structure of the underlying inference problem; this limits its utility in real-time online applications, where speed is crucial.

More sophisticated solutions achieve faster computation by directly exploiting the sequentiality of the online inference problem. The canonical example is online gradient descent, which is attractive for its robustness, simplicity, and low memory and per-iteration computational costs, but its first-order rate can lead to painfully slow convergence [8]. Alternatively, Kaess et al. have developed incremental smoothing and mapping (iSAM), [9], [10], which exploits recent algorithmic advances in sparse numerical linear algebra to implement an efficient incrementalized version of the Gauss-Newton method [8] for use in online sparse least-squares minimization. This incremental approach, together with the Gauss-Newton method’s superlinear convergence rate, enables iSAM to achieve computational speeds unmatched by iterated batch techniques. However, the Gauss-Newton method can exhibit poor (even globally divergent) behavior when applied to objective functions with significant nonlinearity [11], which restricts the class of problems to which iSAM can be reliably applied. To date, the development of a fully incremental online sparse least-squares solver that combines the robustness of gradient descent with the superlinear convergence rate of Newton-type methods has remained an outstanding problem.
To that end, in this paper we present Robust Incremental Least-Squares Estimation (RISE), an incrementalized version of the Powell’s Dog-Leg numerical optimization algorithm [8], [12] suitable for use in online sequential sparse least-squares minimization. As a trust-region method (Fig. 1), Powell’s Dog-Leg is naturally robust to objective function nonlinearity and numerical ill-conditioning, and enjoys excellent global convergence properties [13]–[15]; furthermore, it is known to perform significantly faster than Levenberg-Marquardt in batch sparse least-squares minimization while obtaining solutions of comparable quality [16]. By exploiting iSAM’s pre-existing functionality to incrementalize the computation of the dog-leg step, RISE maintains the speed of current state-of-the-art online sparse least-squares solvers while providing superior robustness to objective function nonlinearity and numerical ill-conditioning.

The rest of this paper is organized as follows. In the next section we formulate the sequential sparse least-squares minimization problem and discuss its connections to online inference. In Section III we review the class of Newton-type optimization methods, focusing in particular on the Gauss-Newton method and its incrementalization to produce iSAM. Section IV introduces the general class of trust-region methods, paying particular attention to Powell’s Dog-Leg. Here we derive the indefinite Gauss-Newton-Powell’s Dog-Leg (IGN-PDL) algorithm (an extension of Powell’s Dog-Leg with Gauss-Newton steps to the case of indefinite Jacobians), analyze its robustness with respect to objective function nonlinearity and numerical ill-conditioning, and establish sufficient conditions for its global convergence (Theorem 3). We then derive the RISE and RISE2 algorithms in Section V by incrementalizing IGN-PDL with the aid of iSAM. We contextualize RISE with a discussion of related work in Section VI, and evaluate its performance in Section VII on standard 6DOF pose-graph SLAM benchmarks and on a real-world visual mapping task using a calibrated monocular camera. Finally, Section VIII concludes with a summary of this paper’s contributions and a discussion of future research directions.

II. PROBLEM FORMULATION

We are interested in the general problem of obtaining a point estimate \( x^* \in \mathbb{R}^n \) of some quantity of interest \( X \) as the solution of a sparse nonlinear least-squares problem

\[
\min_{x \in \mathbb{R}^n} S(x)
\]

\[
S(x) = \sum_{i=1}^{m} r_i(x)^2 = \|r(x)\|^2
\]

for \( r: \mathbb{R}^n \to \mathbb{R}^m \) with \( m \geq n \). Problems of this form frequently arise in probabilistic inference in the context of maximum likelihood (ML) or maximum a posteriori (MAP) parameter estimation; indeed, performing ML or MAP estimation over any probability distribution \( p: \mathbb{R}^n \to \mathbb{R}^+ \) whose factor graph representation \( G = (F, \mathcal{X}, \mathcal{E}) \) [17] is sparse and whose factors are positive and bounded is equivalent to solving a sparse least-squares problem of the form (1) in which each summand \( r_i \) corresponds to a factor \( p_i \) of \( p \) [18]. Given the ubiquity of these models, robust and computationally efficient methods for solving (1) are thus of significant practical import.

In the case of online inference, the input data is collected sequentially, and we wish to obtain a revised estimate for \( X \) after the incorporation of each datum. Since each input datum gives rise to a summmand in (1), online inference corresponds to solving the sequence of sparse least-squares problems

\[
\min_{x_t \in \mathbb{R}^n} S^{(t)}(x_t)
\]

\[
S^{(t)}(x_t) = \sum_{i=1}^{m_t} r_i(x_t)^2 = \|r^{(t)}(x_t)\|^2
\]

for \( r^{(t)}: \mathbb{R}^{n_t} \to \mathbb{R}^m \) and \( t = 1, 2, \ldots \), where:

1) \( m_t, n_t \in \mathbb{N}^+ \) are monotonically non-decreasing in \( t \),
2) \( m_t \geq n_t \) for all \( t \),
3) \( x_t \in \mathbb{R}^{n_t} \) for all \( i \) and \( x_i \subseteq x_j \) for all \( i \leq j \).

Condition 1 above expresses the fact that the summation in (2) evolves over time only through the addition of new terms. Condition 2 is necessary in order for the minimization problem in (2) to have a unique solution. Condition 3 formalizes the idea that we also allow the vector of states \( X \) that we wish to estimate to be augmented online by the addition of new quantities of interest (for example, as in the case of robotic mapping when exploring previously unseen areas, cf. Fig. 2).

Our goal in this paper is to develop a fully incremental algorithm capable of robustly solving online sparse least-squares minimization problems of the form (2) in real-time.

III. REVIEW OF NEWTON-TYPE OPTIMIZATION METHODS AND iSAM

The RISE algorithm that we develop in Section V exploits iSAM’s incremental computation of the Gauss-Newton step in order to solve the sequential sparse least-squares problem (2) efficiently in the online case. In this section, we review the general class of Newton-type optimization methods, their specialization to the Gauss-Newton method, and Gauss-Newton’s incremental implementation in iSAM.

A. Newton’s method and its approximations

Newton’s method [8], [11] is an iterative numerical method for estimating a solution \( x^* \) of the general nonlinear minimization problem

\[
\min_{x \in \mathbb{R}^n} f(x), \quad f \in C^2(\mathbb{R}^n).
\]
Given an initial estimate \( x^{(i)} \) for \( x^* \), the function \( f \) is locally approximated at \( x^{(i)} \) by its second-order Taylor expansion \( q^{(i)} \):

\[
q^{(i)}(x^{(i)} + h) = f(x^{(i)}) + \nabla f(x^{(i)})^T h + \frac{1}{2} h^T \frac{\partial^2 f}{\partial x^2}(x^{(i)}) h, \tag{4}
\]

and a revised estimate

\[
x^{(i+1)} = x^{(i)} + h_N^{(i)} \tag{5}
\]

is computed by choosing the Newton step \( h_N^{(i)} \) to be any increment to \( x^{(i)} \) that minimizes the value of the local approximation (4):

\[
h_N^{(i)} \in \text{argmin}_{h \in \mathbb{R}^n} q^{(i)}(x^{(i)} + h), \tag{6}
\]

Provided that \( \frac{\partial^2 f}{\partial x^2}(x^{(i)}) > 0 \), there is a unique minimizer \( h_N^{(i)} \) in (6), which is determined as the solution of

\[
\frac{\partial^2 f}{\partial x^2}(x^{(i)}) h_N^{(i)} = -\nabla f(x^{(i)}). \tag{7}
\]

Assuming that each of the steps \( h_N^{(i)} \) in (6) exists, Newton’s method consists of iteratively applying equations (4), (6), and (5), in that order, to generate a sequence of estimates \( x^{(0)}, x^{(1)}, \ldots \) for \( x^* \) until some stopping criterion is satisfied.

Newton’s method has several attractive theoretical properties, in particular a fast (quadratic) convergence rate when initialized with an estimate \( x^{(0)} \) that is close to a minimizer \( x^* \) of a sufficiently regular function \( f \) [8]. However, in application it may not always be practical or computationally feasible to evaluate the gradient \( \nabla f(x^{(i)}) \) or Hessian \( \frac{\partial^2 f}{\partial x^2}(x^{(i)}) \) in the quadratic model (4) at every iteration (depending upon the dimension \( n \) and analytical complexity of the function \( f \)). In these cases, the local model (4) is often relaxed to

\[
q^{(i)}(x^{(i)} + h) = f(x^{(i)}) + (g^{(i)})^T h + \frac{1}{2} h^T B^{(i)} h, \tag{8}
\]

where \( g^{(i)} \in \mathbb{R}^n \) and \( B^{(i)} \in \mathbb{R}^{n \times n} \) symmetric are chosen such that

\[
g^{(i)} \approx \nabla f(x^{(i)}), \quad B^{(i)} \approx \frac{\partial^2 f}{\partial x^2}(x^{(i)}), \tag{9}
\]

and the corresponding update step \( h_N^{(i)} \) is computed as a solution of

\[
B^{(i)} h_N^{(i)} = -g^{(i)}. \tag{10}
\]

The selection of different methods for performing the approximations (9) gives rise to a broad class of optimization algorithms collectively referred to as Newton-type or approximate Newton methods. With a careful choice of approximation scheme in (9) it is possible to preserve many of the desirable properties of Newton’s method (most importantly a superlinear end-stage convergence rate) while dramatically reducing the computational burden of computing the update steps \( h_N^{(i)} \).

B. The Gauss-Newton method

The Gauss-Newton method [8], [11] is an approximate Newton method for solving the minimization problem (3) in the special case (1) in which the objective function is a sum of squared nonlinear terms. In this case we have

\[
\frac{\partial S}{\partial x_j} = 2 \sum_{i=1}^m r_i \frac{\partial r_i}{\partial x_j}, \tag{11a}
\]

\[
\frac{\partial^2 S}{\partial x_j \partial x_k} = 2 \sum_{i=1}^m \frac{\partial r_i}{\partial x_j} \frac{\partial r_i}{\partial x_k} + r_i \frac{\partial^2 r_i}{\partial x_j \partial x_k}, \tag{11b}
\]

and the Gauss-Newton method is obtained as an approximate Newton method by ignoring the effects of the second-order partial derivatives of \( r \) when forming the approximate Hessian \( B^{(i)} \) in (9):

\[
\frac{\partial^2 S}{\partial x_j \partial x_k} \approx 2 \sum_{i=1}^m \frac{\partial r_i}{\partial x_j} \frac{\partial r_i}{\partial x_k}, \tag{12}
\]

(the exact gradient \( \nabla S(x^{(i)}) \) corresponding to (11a) is used for \( g^{(i)} \)). Using the function \( r : \mathbb{R}^n \to \mathbb{R}^m \), we can write this approximation more conveniently in matrix notation as

\[
g^{(i)} = 2J(x^{(i)})^T r(x^{(i)}), \tag{13a}
\]

\[
B^{(i)} = 2J(x^{(i)})^T J(x^{(i)}), \tag{13b}
\]

where \( J(x^{(i)}) \) denotes the Jacobian of \( r \) evaluated at \( x^{(i)} \):

\[
J(x^{(i)}) = \frac{\partial r}{\partial x} \mid_{x=x^{(i)}} \in \mathbb{R}^{m \times n}. \tag{14}
\]

We observe that the approximation (12) is equivalent to the assumption that \( r \) is locally linear. Indeed, substitution of (13) into (8) produces

\[
q^{(i)}(x^{(i)} + h) = r(x^{(i)})^T r(x^{(i)}) + 2r(x^{(i)})^T J(x^{(i)}) h + h^T J(x^{(i)})^T J(x^{(i)}) h \tag{15}
\]

\[
= \left\| L^{(i)}(x^{(i)} + h) \right\|^2, \tag{16}
\]

where

\[
L^{(i)}(x^{(i)} + h) = r(x^{(i)}) + J(x^{(i)}) h \tag{17}
\]

is the first-order Taylor expansion (i.e. linearization) of \( r \) about \( x^{(i)} \). Consequently, by virtue of (11b), (12), and (15), we expect the Gauss-Newton method to produce the best results when applied to functions \( r \) that have relatively modest nonlinearities (as quantified by the magnitudes of their second partial derivatives) and small magnitudes \( \| r \| \).

To compute the Gauss-Newton step \( h_{GN}^{(i)} \), we observe that if \( J(x^{(i)}) \) is full-rank, then \( B^{(i)} \) as defined in (13b) is positive definite, and thus \( h_{GN}^{(i)} \) is uniquely determined by

\[
2J(x^{(i)})^T J(x^{(i)}) h_{GN}^{(i)} = -2J(x^{(i)})^T r(x^{(i)}), \tag{17}
\]

following (10). Letting

\[
Q^{(i)} \begin{pmatrix} R^{(i)} \\ 0 \end{pmatrix} = J(x^{(i)}) \tag{18}
\]

be a QR decomposition [19] of the Jacobian \( J(x^{(i)}) \) and

\[
\begin{pmatrix} q^{(i)} \\ e^{(i)} \end{pmatrix} = \begin{pmatrix} Q^{(i)} \\ 0 \end{pmatrix} r(x^{(i)}) \tag{19}
\]

for \( d^{(i)} \in \mathbb{R}^n \) and \( e^{(i)} \in \mathbb{R}^{m-n} \), we can simplify (17) to

\[
R^{(i)} h_{GN}^{(i)} = -d^{(i)}. \tag{20}
\]

Since \( R^{(i)} \) is upper-triangular, equation (20) can be efficiently solved for \( h_{GN}^{(i)} \) by back-substitution.
C. iSAM: Incrementalizing Gauss-Newton

As shown in Section II, the arrival of new data corresponds to augmenting the function \( r = r_{old} : \mathbb{R}^n \rightarrow \mathbb{R}^m \) on the right-hand side of (1) to the function

\[
\bar{r} : \mathbb{R}^{n+n_{new}} \rightarrow \mathbb{R}^{m+m_{new}}
\]

\[
\bar{r}(x_{old}, x_{new}) = \left( r_{old}(x_{old}) \quad r_{new}(x_{old}, x_{new}) \right),
\]

where here \( r_{new} : \mathbb{R}^{n+n_{new}} \rightarrow \mathbb{R}^{m_{new}} \) is the set of new measurement functions and \( x_{new} \in \mathbb{R}^{n_{new}} \) is the set of new system variables introduced as a result of the new observations.

In the naïve application of the Gauss-Newton algorithm of Section III-B, the solution \( x^* = (x^*_{old}, x^*_{new}) \) for the augmented least-squares problem determined by (21) would be found by performing Gauss-Newton iterations until convergence. However, in the context of the sequential estimation problem (2) we already have a good estimate \( \hat{x}_{old} \) for the values of the previously-introduced variables \( x_{old} \) obtained by solving the least-squares minimization problem (1) prior to the introduction of the new measurement functions \( r_{new} \). Thus, given any good initialization \( \hat{x}_{new} \) for the newly-introduced system variables \( x_{new} \), \( \hat{x} = (\hat{x}_{old}, \hat{x}_{new}) \) provides a good initialization for the augmented state \( x = (x_{old}, x_{new}) \) for the Gauss-Newton algorithm.

Furthermore, since we generally expect the initial estimate \( \hat{x} \) to be close to the true minimizing value \( x^* \), it is not usually necessary to iterate the Gauss-Newton algorithm until convergence after integration of every new observation; instead, a single Gauss-Newton step is computed and used to correct the initial estimate \( \hat{x} \). The advantage to this approach is that it avoids having to recompute the Jacobian \( \bar{J}(\hat{x}) \) for \( \bar{r} \) and its QR decomposition anew each time new observations arrive; instead, iSAM efficiently obtains \( \bar{J}(\hat{x}) \) together with its QR decomposition by updating the Jacobian \( \bar{J}(\hat{x}_{old}) \) and its QR decomposition, as we now describe.

Letting \( x = (\hat{x}_{old}, x_{new}) \), the Jacobian \( \bar{J}(\hat{x}) \) for the augmented system (21) can be decomposed into block form as

\[
\bar{J}(\hat{x}) = \frac{\partial \bar{r}}{\partial x} = \begin{pmatrix}
\frac{\partial r_{old}}{\partial x_{old}} & 0 \\
J_{new}(x) & 0
\end{pmatrix}
\]

\[
\left( \begin{array}{c}
J(\hat{x}_{old}) \\
J_{new}(x)
\end{array} \right) = 0
\]

where

\[
J(\hat{x}_{old}) = \frac{\partial r_{old}}{\partial x_{old}} \in \mathbb{R}^{m \times n}
\]

is the Jacobian of the previous function \( r_{old} \) and

\[
J_{new}(x) = \frac{\partial r_{new}}{\partial x} \in \mathbb{R}^{m_{new} \times (n+n_{new})}
\]

is the Jacobian of the new measurement function \( r_{new} \). Letting

\[
J(\hat{x}_{old}) = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix}
\]

be a QR decomposition for \( J(\hat{x}_{old}) \), where \( Q_1 \in \mathbb{R}^{m \times n} \) and \( Q_2 \in \mathbb{R}^{m \times (m-n)} \), we have

\[
\begin{pmatrix} Q_1 & 0 & Q_2 \\
0 & I & 0
\end{pmatrix}
\]

\[
\left( \begin{array}{c}
R \\
J_{new}(\hat{x})
\end{array} \right) = \left( \begin{array}{c}
Q_1 R \\
J_{new}(\hat{x})
\end{array} \right) = \bar{J}(\hat{x})
\]

by (22) and (25), which gives a partial QR decomposition of \( \bar{J}(\hat{x}) \). This partial decomposition can be completed by using Givens rotations to zero out the remaining nonzero elements below the main diagonal. Let \( G \in \mathbb{R}^{(n+m_{new}) \times (n+m_{new})} \) denote a matrix of Givens rotations such that

\[
G \left( \begin{array}{c}
R \\
J_{new}(\hat{x})
\end{array} \right) = \left( \begin{array}{c}
\hat{R} \\
0
\end{array} \right)
\]

where \( \hat{R} \in \mathbb{R}^{(n+m_{new}) \times (n+m_{new})} \) is upper-triangular. Defining

\[
\bar{G} = \begin{pmatrix} G & 0 \\
0 & I \end{pmatrix}, \quad \bar{Q} = \begin{pmatrix} Q_1 & 0 & Q_2 \\
0 & I & 0
\end{pmatrix} \bar{G}^T,
\]

(28)

we can use (19) and (20) to compute the Gauss-Newton step \( \bar{h}_{GN} \) for the augmented system:

\[
\bar{Q}^T \bar{r}(\hat{x}) = \begin{pmatrix} G & 0 \\
0 & I \end{pmatrix} \begin{pmatrix} Q_1 & 0 & Q_2 \\
0 & I & 0
\end{pmatrix} \begin{pmatrix} r_{old}(\hat{x}_{old}) \\
\end{pmatrix}
\]

\[
= \begin{pmatrix} G & 0 \\
0 & I \end{pmatrix} \begin{pmatrix} d_{old} \\
0
\end{pmatrix}
\]

\[
= \begin{pmatrix} \bar{d} \end{pmatrix}
\]

\[
= \begin{pmatrix} \bar{d} \\
0
\end{pmatrix}
\]

\[
\bar{d}_{new}
\]

\[
\bar{e}_{old}
\]

\[
\bar{e}
\]

\[
= \begin{pmatrix} \bar{d} \\
0
\end{pmatrix}
\]

\[
= \begin{pmatrix} \bar{d} \\
0
\end{pmatrix}
\]

\[
\bar{h}_{GN} \]

(31)

Equations (24), (27) and (31) show how to obtain the \( \hat{R} \) factor of the QR decomposition of \( \bar{J}(\hat{x}) \) and the corresponding linear system (32) by updating the \( \hat{R} \) factor and linear system for the previous Jacobian \( \bar{J}(\hat{x}_{old}) \) using Givens rotations. Since the updated factor \( \hat{R} \) and the new right-hand side vector \( \bar{d} \) are obtained by applying \( \bar{G} \) directly to the augmented factor \( R \) in (27) and the augmented right-hand side vector \( \bar{d} \) in (31), it is not necessary to explicitly form the orthogonal matrix \( \bar{Q} \) in (28). Nor is it necessary to form the matrix \( G \) explicitly either; instead, the appropriate individual Givens rotations can be applied sequentially directly to the matrix in (27) and the right-hand side vector in (31). Under the assumption of sparsity, only a few elements of the the Jacobian \( J_{new}(\hat{x}) \) will be nonzero, and therefore only a small number of Givens rotations will be needed to perform the update in (27). Obtaining the linear system (32) by this method is thus a computationally efficient operation.
Finally, we observe that while relinearization is not needed after every new observation, the system should be periodically relinearized about its corrected estimate in order to perform a full Gauss-Newton iteration and obtain a better estimate of the local minimum (this is particularly true after observations which are likely to significantly alter the estimates of system variables). When relinearizing the system about the corrected estimate, the incremental updating method outlined above is no longer applicable; instead, the QR factorization of the Jacobian needs to be recomputed anew. While this is an expensive batch operation, the factorization step can be combined with a variable reordering step [21] in order to reduce the fill-in in the resulting factor $R$, thereby maintaining sparsity and speeding up subsequent incremental computations.

IV. FROM GAUSS-NEWTON TO POWELL’S DOG-LEG

The incrementalized Gauss-Newton method outlined in Section III-C is computationally efficient, straightforward to implement, and enjoys rapid (up to quadratic [11, pg. 113]) convergence near the minimum. However, the assumption of the local linearity of $r$ (which underlies the approximations (13b) and (15)) means that the Gauss-Newton method can exhibit poor behavior when either $r$ itself or its second partial derivatives $\frac{\partial^2 r}{\partial x^2}$ have large magnitude. Indeed, convergence of the Gauss-Newton method is not guaranteed, not even locally; this is in marked contrast with the behavior of Newton’s method, which (although not globally convergent) can at least guarantee fast local convergence for a class of functions that is not excessively restrictive in practice (cf. Theorem 3.5 of [8]). Furthermore, it is not difficult to construct simple (even quadratic) examples of $r$ where the sequence of Gauss-Newton iterates $\{x^{(i)}\}_{i=1}^\infty$ is in fact globally divergent (i.e. fails to converge when initialized with any point $x^{(0)} \neq x^\ast$). For example, consider the minimization (1) determined by

$$r : \mathbb{R} \to \mathbb{R}^2$$

$$r(x) = \left( \frac{x^2 + 1}{\lambda x^2 + x - 1} \right). \quad (33)$$

For $\lambda = -2$, the function $S(x)$ is $C^\infty$ and strictly convex, and has a single global minimum at $x^\ast = 0$ with $\frac{\partial^2 S}{\partial x^2}(x^\ast) > 0$, hence satisfies all of the standard regularity conditions customarily assumed in numerical optimization (cf. e.g. [8], [11]). Nevertheless, the Gauss-Newton algorithm is in fact globally divergent in this case [11, pg. 113]. This lack of robustness even under ideal conditions (a well-behaved objective function with strong regularity properties and good initialization, cf. Fig. 3) is a serious shortcoming of the Gauss-Newton method.

To address this shortcoming, in this paper we adopt the Powell’s Dog-Leg algorithm [8], [12] as the method of choice for performing the sparse least-squares minimization (1). This algorithm combines the superlinear end-stage convergence speed of the Newton-type methods with the excellent global convergence properties of gradient descent approaches. Indeed, when applied to sparse least-squares minimization problems, Powell’s Dog-Leg performs significantly faster than Levenberg-Marquardt (the current method of choice in the robotics and computer vision communities) while maintaining comparable levels of accuracy [16].

A. The trust-region approach

As shown in Section III-A, in each iteration Newton’s method constructs a local model $q^{(i)}$ for the objective function $f$ on a neighborhood $U^{(i)}$ of the current estimate $x^{(i)}$, and then determines an update step $p_N^{(i)}$ that minimizes $q^{(i)}$ in place of $f$. However, the model $q^{(i)}$ in (4) is constructed using information about $f$’s first- and second-order derivatives at $x^{(i)}$, which depend only upon the local behavior of $f$ near $x^{(i)}$. Therefore, since the Newton update is constructed using information about $f$’s derivatives at $x^{(i)}$, the model $q^{(i)}$ depends only upon the first-order and second-order derivatives of $f$ at $x^{(i)}$. This is in marked contrast with the behavior of Newton’s method, which (although not globally convergent) can at least guarantee fast local convergence for a class of functions that is not excessively restrictive in practice (cf. Theorem 3.5 of [8]). Furthermore, it is not difficult to construct simple (even quadratic) examples of $r$ where the sequence of Gauss-Newton iterates $\{x^{(i)}\}_{i=1}^\infty$ is in fact globally divergent (i.e. fails to converge when initialized with any point $x^{(0)} \neq x^\ast$). For example, consider the minimization (1) determined by

$$r : \mathbb{R} \to \mathbb{R}^2$$

$$r(x) = \left( \frac{x^2 + 1}{\lambda x^2 + x - 1} \right). \quad (33)$$

For $\lambda = -2$, the function $S(x)$ is $C^\infty$ and strictly convex, and has a single global minimum at $x^\ast = 0$ with $\frac{\partial^2 S}{\partial x^2}(x^\ast) > 0$, hence satisfies all of the standard regularity conditions customarily assumed in numerical optimization (cf. e.g. [8], [11]). Nevertheless, the Gauss-Newton algorithm is in fact globally divergent in this case [11, pg. 113]. This lack of robustness even under ideal conditions (a well-behaved objective function with strong regularity properties and good initialization, cf. Fig. 3) is a serious shortcoming of the Gauss-Newton method.

To address this shortcoming, in this paper we adopt the Powell’s Dog-Leg algorithm [8], [12] as the method of choice for performing the sparse least-squares minimization (1). This algorithm combines the superlinear end-stage convergence speed of the Newton-type methods with the excellent global convergence properties [13–15] of gradient descent approaches. Indeed, when applied to sparse least-squares minimization problems, Powell’s Dog-Leg performs significantly faster than Levenberg-Marquardt (the current method of choice in the robotics and computer vision communities) while maintaining comparable levels of accuracy [16].

A. The trust-region approach

As shown in Section III-A, in each iteration Newton’s method constructs a local model $q^{(i)}$ for the objective function $f$ on a neighborhood $U^{(i)}$ of the current estimate $x^{(i)}$, and then determines an update step $p_N^{(i)}$ that minimizes $q^{(i)}$ in place of $f$. However, the model $q^{(i)}$ in (4) is constructed using information about $f$’s first- and second-order derivatives at $x^{(i)}$, which depend only upon the local behavior of $f$ near $x^{(i)}$. Therefore, since the Newton update is constructed using information about $f$’s derivatives at $x^{(i)}$, the model $q^{(i)}$ depends only upon the first-order and second-order derivatives of $f$ at $x^{(i)}$. This is in marked contrast with the behavior of Newton’s method, which (although not globally convergent) can at least guarantee fast local convergence for a class of functions that is not excessively restrictive in practice (cf. Theorem 3.5 of [8]). Furthermore, it is not difficult to construct simple (even quadratic) examples of $r$ where the sequence of Gauss-Newton iterates $\{x^{(i)}\}_{i=1}^\infty$ is in fact globally divergent (i.e. fails to converge when initialized with any point $x^{(0)} \neq x^\ast$). For example, consider the minimization (1) determined by

$$r : \mathbb{R} \to \mathbb{R}^2$$

$$r(x) = \left( \frac{x^2 + 1}{\lambda x^2 + x - 1} \right). \quad (33)$$

For $\lambda = -2$, the function $S(x)$ is $C^\infty$ and strictly convex, and has a single global minimum at $x^\ast = 0$ with $\frac{\partial^2 S}{\partial x^2}(x^\ast) > 0$, hence satisfies all of the standard regularity conditions customarily assumed in numerical optimization (cf. e.g. [8], [11]). Nevertheless, the Gauss-Newton algorithm is in fact globally divergent in this case [11, pg. 113]. This lack of robustness even under ideal conditions (a well-behaved objective function with strong regularity properties and good initialization, cf. Fig. 3) is a serious shortcoming of the Gauss-Newton method.

To address this shortcoming, in this paper we adopt the Powell’s Dog-Leg algorithm [8], [12] as the method of choice for performing the sparse least-squares minimization (1). This algorithm combines the superlinear end-stage convergence speed of the Newton-type methods with the excellent global convergence properties [13–15] of gradient descent approaches. Indeed, when applied to sparse least-squares minimization problems, Powell’s Dog-Leg performs significantly faster than Levenberg-Marquardt (the current method of choice in the robotics and computer vision communities) while maintaining comparable levels of accuracy [16].

A. The trust-region approach

As shown in Section III-A, in each iteration Newton’s method constructs a local model $q^{(i)}$ for the objective function $f$ on a neighborhood $U^{(i)}$ of the current estimate $x^{(i)}$, and then determines an update step $p_N^{(i)}$ that minimizes $q^{(i)}$ in place of $f$. However, the model $q^{(i)}$ in (4) is constructed using information about $f$’s first- and second-order derivatives at $x^{(i)}$, which depend only upon the local behavior of $f$ near $x^{(i)}$. Therefore, since the Newton update is constructed using information about $f$’s derivatives at $x^{(i)}$, the model $q^{(i)}$ depends only upon the first-order and second-order derivatives of $f$ at $x^{(i)}$. This is in marked contrast with the behavior of Newton’s method, which (although not globally convergent) can at least guarantee fast local convergence for a class of functions that is not excessively restrictive in practice (cf. Theorem 3.5 of [8]). Furthermore, it is not difficult to construct simple (even quadratic) examples of $r$ where the sequence of Gauss-Newton iterates $\{x^{(i)}\}_{i=1}^\infty$ is in fact globally divergent (i.e. fails to converge when initialized with any point $x^{(0)} \neq x^\ast$). For example, consider the minimization (1) determined by

$$r : \mathbb{R} \to \mathbb{R}^2$$

$$r(x) = \left( \frac{x^2 + 1}{\lambda x^2 + x - 1} \right). \quad (33)$$

For $\lambda = -2$, the function $S(x)$ is $C^\infty$ and strictly convex, and has a single global minimum at $x^\ast = 0$ with $\frac{\partial^2 S}{\partial x^2}(x^\ast) > 0$, hence satisfies all of the standard regularity conditions customarily assumed in numerical optimization (cf. e.g. [8], [11]). Nevertheless, the Gauss-Newton algorithm is in fact globally divergent in this case [11, pg. 113]. This lack of robustness even under ideal conditions (a well-behaved objective function with strong regularity properties and good initialization, cf. Fig. 3) is a serious shortcoming of the Gauss-Newton method.

To address this shortcoming, in this paper we adopt the Powell’s Dog-Leg algorithm [8], [12] as the method of choice for performing the sparse least-squares minimization (1). This algorithm combines the superlinear end-stage convergence speed of the Newton-type methods with the excellent global convergence properties [13–15] of gradient descent approaches. Indeed, when applied to sparse least-squares minimization problems, Powell’s Dog-Leg performs significantly faster than Levenberg-Marquardt (the current method of choice in the robotics and computer vision communities) while maintaining comparable levels of accuracy [16].
Consequently, while $q^{(i)}$ is a good approximation of $f$ near $x^{(i)}$ (in a sense made precise by Taylor’s Theorem), its global behavior may be quite different from $f$’s. This can become problematic if the update step $h_{N}^{(i)}$ computed using $q^{(i)}$ leaves the region $U^{(i)}$ in which $f$ is well-approximated by $q^{(i)}$.

Trust-region methods [8] address this hazard by maintaining an explicit trust-region, an open ball of radius $\Delta^{(i)}$ centered on the current estimate $x^{(i)}$ within which $f$ is considered well-approximated by the local model $q^{(i)}$. The trust-region update step $h_{tr}^{(i)}$ is then obtained as an increment to $x^{(i)}$ that minimizes the value of $q^{(i)}$, subject to the condition that the update step does not leave the trust-region:

$$ h_{tr}^{(i)} \in \arg\min_{\|h\| \leq \Delta^{(i)}} q^{(i)}(x^{(i)} + h). \quad (34) $$

The following theorem gives necessary and sufficient optimality conditions characterizing the trust-region step $h_{tr}^{(i)}$ in (34) when $q^{(i)}$ is a quadratic model (as in approximate Newton methods). Its proof follows from a straightforward application of the Karush-Kuhn-Tucker first-order optimality conditions together with the enforcement of the necessary second-order conditions for a minimizer (cf. [8, Chps. 4 and 12]).

**Theorem 1** (Optimality conditions for the trust-region step). Let $f, g \in \mathbb{R}^{n}$, let $B \in \mathbb{R}^{n \times n}$ be symmetric, and define

$$ q : \mathbb{R}^{n} \rightarrow \mathbb{R} $$

$$ q(h) = f + g^{T}h + \frac{1}{2}h^{T}Bh. \quad (35) $$

Then $h^{*} \in \mathbb{R}^{n}$ is a solution of the constrained minimization

$$ \min_{h \in \mathbb{R}^{n}} q(h) \quad \text{s.t.} \quad \|h\| \leq \Delta \quad (36) $$

for $\Delta > 0$ if and only if $\|h^{*}\| \leq \Delta$ and there exists $\lambda^{*} \geq 0$ such that the following conditions are satisfied:

$$ (B + \lambda^{*}I)h^{*} = -g, \quad (37a) $$

$$ \lambda^{*}(\Delta - \|h^{*}\|) = 0, \quad (37b) $$

$$ (B + \lambda^{*}I) \geq 0. \quad (37c) $$

As the trust-region method proceeds, the radius of the trust region $\Delta^{(i)}$ is varied adaptively according to the gain ratio:

$$ \rho^{(i)} = \frac{\text{ared}^{(i)}(h_{tr}^{(i)})}{\text{pred}^{(i)}(h_{tr}^{(i)})}, \quad (38) $$

where

$$ \text{ared}^{(i)}(h_{tr}^{(i)}) = f(x^{(i)}) - f(x^{(i)} + h_{tr}^{(i)}), $$

$$ \text{pred}^{(i)}(h_{tr}^{(i)}) = q^{(i)}(x^{(i)}) - q^{(i)}(x^{(i)} + h_{tr}^{(i)}); \quad (39) $$

this compares the actual reduction in the objective function’s value obtained by taking the proposed trust-region step $h_{tr}^{(i)}$ with the predicted reduction in function value using the local model $q^{(i)}$. Values of $\rho$ close to 1 indicate that the local model $q^{(i)}$ is performing well near the current iterate $x^{(i)}$, so the trust-region radius $\Delta^{(i)}$ can be increased in the next iteration (to allow longer steps and hence faster convergence), while values close to 0 indicate that the local model is performing poorly, and $\Delta^{(i)}$ should be reduced accordingly (Algorithm 1). The entire trust-region method is summarized as Algorithm 2:

**Algorithm 1** Updating the trust-region radius $\Delta$

1: procedure UPDATE_DELTA($\rho, \Delta, \eta_{1}, \eta_{2}, \gamma_{1}, \gamma_{2}$)
2: if $\rho \geq \eta_{2}$ then
3: $\Delta \leftarrow \gamma_{2}\Delta$
4: else if $\rho < \eta_{1}$ then
5: $\Delta \leftarrow \gamma_{1}\Delta$
6: end if
7: return $\Delta$
8: end procedure

**Algorithm 2** The trust-region minimization method

1: procedure TRUST-REGION($f, x_{0}, \Delta_{0}, \eta_{1}, \eta_{2}, \gamma_{1}, \gamma_{2}$)
2: Initialize $x \leftarrow x_{0}, \Delta \leftarrow \Delta_{0}$.
3: repeat
4: Construct local model $q$ for $f$ about $x$.
5: Compute trust-region step $h_{tr}$ by solving (34).
6: Set $x_{proposed} \leftarrow (x + h_{tr})$.
7: Compute $\rho$ using (38).
8: if $\rho \geq \eta_{1}$ then
9: Set $x \leftarrow x_{proposed}$.
10: end if
11: $\Delta \leftarrow $ UPDATE_DELTA($\rho, \Delta, \eta_{1}, \eta_{2}, \gamma_{1}, \gamma_{2}$).
12: until (stopping criteria)
13: return $x$
14: end procedure

Here $0 < \eta_{1} < \eta_{2} < 1$ and $0 < \gamma_{1} < 1 < \gamma_{2}$ are user-supplied parameters specifying the gain-ratio thresholds and scaling factors used in the update of the trust-region radius.

**B. Powell’s Dog-Leg**

While the canonical trust-region method (Algorithm 2) is intellectually pleasing and enjoys excellent convergence and robustness properties (cf. Section IV-D), it is not commonly used in practice because of the computational cost of finding the exact trust-region step $h_{tr}$ in (34). (Briefly, this entails solving a root-finding problem for the Lagrange multiplier $\lambda^{*}$ in (37a) that generally requires multiple factorizations of matrices of the form $B + \lambda I$ [22], a prohibitively expensive operation when appearing as a single step within another iterative algorithm.) Instead, most practical trust-region implementations solve (34) approximately using a computationally cheap approximation that is nevertheless accurate enough to preserve the desirable properties of the canonical algorithm. In this section we describe Powell’s Dog-Leg [8, 12], one such approximate method that can be used whenever the matrix $B$ in the local quadratic model $q$ in (34) is positive definite.

The dog-leg approximation is derived by regarding the trust-region step computed in (34) as a function $h_{tr}(\Delta)$ of the trust-region radius and considering the effect of varying $\Delta$. For $B \in \mathbb{R}^{n \times n}$ symmetric and positive definite, the local model $q$ has a unique unconstrained minimizer: the Newton step

$$ h_{N} = -B^{-1}g. \quad (40) $$

If the trust-region radius $\Delta$ is greater than the length of this step, then the constraint in (34) is inactive, and therefore

$$ h_{tr}(\Delta) = h_{N}, \quad \|h_{N}\| \leq \Delta. \quad (41) $$
Conversely, when the trust-region radius $\Delta$ is small, the (second-order) quadratic term in the local model (8) is dominated by the linear term, and therefore the trust-region step computed in (34) will be well-approximated by the maximum step length allowable in the direction of steepest descent:

$$h_{tr}(\Delta) \approx -\frac{\Delta}{\|g\|}g, \quad \Delta \text{ small.} \quad (42)$$

As $\Delta$ increases through the range $[0, \infty)$, the exact trust-region step $h_{tr}(\Delta)$ traces a smooth path interpolating these cases; Powell’s Dog-Leg approximates this path using a piecewise linear path with two segments (Fig. 4). The first segment extends from $x$ to the gradient descent step $h_{gd}$, which we define as the unconstrained minimizer of the local model $q$ along the steepest descent direction:

$$h_{gd} = -ag, \quad \alpha = \arg\min_{a \in \mathbb{R}^+} q(x - ag). \quad (43)$$

Using the definition of the local model (8) and the fact that $B > 0$ by hypothesis, the gradient descent step defined in (43) can be written in closed-form as:

$$h_{gd} = -ag, \quad \alpha = \frac{g^Tg}{g^TBg}. \quad (44)$$

The second segment linearly interpolates the gradient descent step $h_{gd}$ and the Newton step $h_N$. Parameterizing the dog-leg path as $p_{dl} : [0, 1] \to \mathbb{R}^n$, we have

$$p_{dl}(t) = \begin{cases} 2th_{gd}, & 0 \leq t \leq \frac{1}{2}, \\
 h_{gd} + (2t - 1)(h_N - h_{gd}), & \frac{1}{2} \leq t \leq 1. \quad (45) \end{cases}$$

A direct computation using the definitions (40), (43), and (45) proves the following result (cf. e.g. [8, Sec. 4.1]).

**Lemma 1.** The dog-leg path $p_{dl}$ defined in (45) satisfies the following properties:

(i) $\|p_{dl}(t)\|$ is monotonically increasing for all $t \in [0, 1]$.

(ii) $q(p_{dl}(t))$ is monotonically decreasing for all $t \in [0, 1]$.

By virtue of Lemma 1, the Powell’s Dog-Leg step $h_{dl}$ is defined to be the (unique) farthest point along the dog-leg path $p_{dl}$ lying inside the trust-region boundary (cf. Fig. 1):

$$h_{dl} = p_{dl}(\tau), \quad \tau = \max\{t \in [0, 1] \mid \|p_{dl}(t)\| \leq \Delta\}. \quad (46)$$

The algorithmic computation of the dog-leg step $h_{dl}$ corresponding to definition (46) is given in Algorithm 3. The scalar $\beta$ appearing in line 7 is chosen to satisfy

$$\|h_{gd} + \beta(h_N - h_{gd})\|^2 = \Delta^2, \quad (47)$$

which is quadratic in $\beta$. By Lemma 1, (47) has exactly one solution in $(0, 1)$, which can be written in closed form as

$$v = h_N - h_{gd}, \quad \beta = \frac{-(h_{gd}^Tv + \sqrt{(h_{gd}^Tv)^2 + (\Delta^2 - \|h_{gd}\|^2\|v\|^2)})}{\|v\|^2}. \quad (48)$$

The complete Powell’s Dog-Leg algorithm is obtained from Algorithm 2 by replacing the computation of the trust-region step $h_{tr}$ in line 5 with the computation of the dog-leg step $h_{dl}$ defined by equations (40), (44), and Algorithm 3.

![Fig. 4](image-url)

**Algorithm 3** Computing the dog-leg step $h_{dl}$

1: procedure **COMPUTE** _DOG-LEG_$(h_N, h_{gd}, \Delta)$
2: if $\|h_N\| \leq \Delta$ then
3: $h_{dl} \leftarrow h_N$
4: else if $\|h_{gd}\| \geq \Delta$ then
5: $h_{dl} \leftarrow \frac{\|h_{gd}\|}{\|h_{gd}\|} h_{gd}$
6: else
7: $h_{dl} \leftarrow h_{gd} + \beta(h_N - h_{gd})$, where $\beta \in (0, 1)$ is chosen such that $\|h_{dl}\| = \Delta$ (cf. equation (48)).
8: end if
9: return $h_{dl}$
10: end procedure

C. **Indefinite Gauss-Newton-Powell’s Dog-Leg**

In this subsection we derive an approximate trust-region algorithm based on Powell’s Dog-Leg with Gauss-Newton steps for solving minimization problems of the form (1).

To begin, we observe that by virtue of the Hessian approximations (13b) used in the Gauss-Newton local model $q$, $B \geq 0$ always. If $J(x)$ is full-rank, then $B > 0$ and the dog-leg step $h_{dl}$ defined in Section IV-B exists and can be computed in terms of the Gauss-Newton step $h_{GN}$ and the gradient descent step $h_{gd}$. Equations (18)–(20) already provide a closed-form solution for computing the Gauss-Newton step, so it suffices to provide a closed-form solution for the gradient descent step $h_{gd}$ in (44). Substituting the expressions for $B$ and $g$ from (13) into (44), we find

$$g = 2J(x)^Tr(x), \quad \alpha = \frac{\|g\|^2}{2\|J(x)g\|^2}. \quad (49)$$

Equations (18)–(20), (49), and Algorithm 3 thus enable the computation of the dog-leg step $h_{dl}$ when $B > 0$.

In the case that $J(x)$ is not full-rank, $B$ is not positive definite and the dog-leg step is not defined. However, the Cauchy step $h_C$ (the constrained minimizer of the local model...
Algorithm 4: The indefinite Gauss-Newton-Powell’s Dog-Leg algorithm

1: procedure IGN-PDL(r, x₀, Δ₀, η₁, η₂, γ₁, γ₂)
2: Initialize x ← x₀, Δ ← Δ₀.
3: repeat
4:   Set g ← J(x)ᵀ r(x).
5:   Compute R factor and right-hand side vector d as in equations (18) and (19).
6:   if R is nonsingular then
7:     Compute Gauss-Newton step h_GN using (20).
8:     Set α ← ∥g∥² / ∥J(x)g∥².
9:     Set h_{gd} ← −α g.
10:    Set h ← COMPUTE_DOG-LEG(h_GN, h_{gd}, Δ).
11:  else
12:    Compute κ using equation (53c).
13:    Set h ← −κ g.
14:  end if
15:  Set x_{proposed} ← (x + h).
16:  Compute ρ using (38).
17:  if ρ ≥ η₁ then
18:    Set x ← x_{proposed}.
19:  end if
20:  Δ ← UPDATE_DELTA(ρ, Δ, η₁, η₂, γ₁, γ₂).
21: until (stopping criteria)
22: return x
23: end procedure

Algorithm 4: The indefinite Gauss-Newton-Powell’s Dog-Leg algorithm

Theorem 2 (Global convergence of trust-region methods). Let f : ℜⁿ → ℜ, x₀ ∈ ℜⁿ, and let Ω ⊆ ℜⁿ be any convex open set containing the sublevel set Λ_f(x₀) of f at x₀:

\[ Λ_f(x₀) = \{ x ∈ ℜⁿ | f(x) ≤ f(x₀) \}. \]  

Assume further that f is lower-bounded on Ω, f ∈ C¹(Ω), and that ∇f is Lipschitz continuous on Ω.

Fix constants 0 < η₁ < η₂ < 1 and 0 < γ₁ < 1 < γ₂, ζ < 1 − η₂, β, σ ∈ (0, ∞), and ε ∈ (0, 1], and let \( \{ x^{(i)} \}_{i=0}^{∞} \) be the sequence of iterates obtained in Algorithm 2 using local models g⁽¹⁾ and approximate trust-region update steps \( h_{tr}⁽¹⁾ \) in line 5 that satisfy the following criteria:

(i) Feasibility: \( ∥h_{tr}⁽¹⁾∥ ≤ Δ⁽¹⁾ \) for all \( i ≥ 0 \).

(ii) Gradient approximation error: the approximate gradients g⁽¹⁾ used in each of the local models q⁽¹⁾ satisfy the bounded relative error criterion:

\[ \frac{∥g⁽¹⁾ − ∇f(x⁽¹⁾)∥}{∥g⁽¹⁾∥} ≤ ζ. \]  

(iii) Uniform boundedness of approximate Hessians: each \( B⁽¹⁾ \) in ℜⁿˣⁿ is symmetric and satisfies \( ∥B⁽¹⁾∥ ≤ β \).

(iv) Asymptotic step direction: assuming that \( ∥B⁽¹⁾∥ ≤ β \) for all \( i ≥ 0 \) and that

\[ \liminf_{i→∞} g⁽¹⁾ > 0 \quad \text{and} \quad \lim_{i→∞} Δ⁽¹⁾ = 0, \]  

the step direction \( Θ⁽¹⁾ \) satisfies

\[ \lim_{i→∞} \cos Θ⁽¹⁾ = 1, \]  

where

\[ \cos Θ⁽¹⁾ = -\frac{(g⁽¹⁾)^T h_{tr}⁽¹⁾}{∥g⁽¹⁾∥∥h_{tr}⁽¹⁾∥}. \]
(v) Uniform predicted decrease: each of the proposed update steps $h_{tr}^{(i)}$ satisfies:

$$\text{pred}^{(i)}(h_{tr}^{(i)}) \geq \frac{1}{2} c [g^{(i)}] \min \left\{ \Delta^{(i)}, \frac{||g^{(i)}||}{\sigma} \right\}. \quad (59)$$

Then either $\nabla f(x^{(i)}) = 0$ for some iterate $x^{(i)}$, or the infinite subsequence $\{x^{(i)}\}_{i=0}^{\infty} \subseteq \{x^{(i)}\}_{i=0}^{\infty}$ of iterates that are accepted in line 9 of Algorithm 2 satisfies

$$\lim_{k \to \infty} \|\nabla f(x^{(i)})\| = 0. \quad (60)$$

With the aid of Theorem 2, we prove the following convergence result for the IGN-PDL algorithm (Algorithm 4).

Theorem 3 (Global convergence of the IGN-PDL algorithm). Let $r: \mathbb{R}^n \to \mathbb{R}^m$ be $C^2$, define

$$S(x) = \|r(x)\|^2, \quad (61)$$

and fix $0 < \eta_1 < \eta_2 < 1$ and $0 < \gamma_1 < 1 < \gamma_2$. Given any $x^{(0)} \in \mathbb{R}^n$, if the sublevel set $\mathcal{L}_S(x^{(0)})$ is bounded, then the sequence of iterates $\{x^{(i)}\}$ accepted in line 18 of Algorithm 4 either terminates at some $x^{(k)}$ with $\nabla S(x^{(k)}) = 0$ or satisfies

$$\lim_{i \to \infty} \|\nabla S(x^{(i)})\| \to 0. \quad (62)$$

This theorem is proved in the Appendix.

In practice, the hypotheses of Theorem 3 are quite general; intuitively, the bounded sublevel set condition simply prohibits the cost function $S$ from assigning the same quality to arbitrarily large regions of the state space. Any reasonable inferential cost function arising in practice will satisfy this condition.

2) Robustness to numerical ill-conditioning: In addition to their strong convergence properties, trust-region methods are also naturally robust against numerical ill-conditioning in the linear system (10) used to solve for the Newton step.

Recall that for a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$, the condition number $\kappa_2(A)$ (cf. [19, pg. 230]) is

$$\kappa_2(A) = \left\{ \begin{array}{ll} \frac{\sigma_{\text{max}}(A)}{\sigma_{\text{min}}(A)} & \text{rank}(A) = n, \\
\infty & \text{rank}(A) < n, \end{array} \right. \quad (63)$$

where $\sigma_{\text{max}}(A)$ and $\sigma_{\text{min}}(A)$ give the maximum and minimum singular values of $A$, respectively. Matrices $A$ with $\kappa_2(A) \gg 1$ are called ill-conditioned; these matrices tend to have poor numerical properties when used as the coefficient matrix for inverse problems of the form

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2 \quad (64)$$

for $b \in \mathbb{R}^m$ (of which (10) and (15) are special cases). To see why this is so, observe that if $\text{rank}(A) = n$, then (64) has the unique minimizer

$$x_{LS} = \text{argmin}_{x \in \mathbb{R}^n} \|Ax - b\|^2 = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i, \quad (65)$$

where $\sigma_1 \geq \cdots \geq \sigma_n > 0$ are the singular values of $A$, and $u_i \in \mathbb{R}^{m \times 1}$ and $v_i \in \mathbb{R}^{n \times 1}$ are the left- and right-singular vectors (respectively) corresponding to $\sigma_i$ for all $1 \leq i \leq n$ (cf. [19, Sec. 5.5.3]). If $\kappa_2(A) \gg 1$, then since $\{u_i\}_{i=1}^{n}$ and $\{v_i\}_{i=1}^{n}$ are orthonormal sets, (65) shows that $x_{LS}$ will tend to be dominated by the effects of those (generally few) components of $b$ lying in subspaces spanned by left-singular vectors $u_i$ corresponding to small singular values $\sigma_i$. In practice, this commonly manifests as the least-squares solution $x_{LS}$ “exploding” (as measured by $\|\cdot\|$) through subspaces of $\mathbb{R}^n$ spanned by right-singular vectors $v_i$ corresponding to small singular values $\sigma_i$ whenever $A$ is ill-conditioned.

One standard approach for addressing poor numerical conditioning in (64) is Tikhonov regularization [23]. In this approach, the original problem (64) is replaced by

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2 + \lambda \|\Gamma x\|^2, \quad (66)$$

where $\Gamma \in \mathbb{R}^{n \times n}$ is a conditioning matrix designed to control some property of interest of the Tikhonov-regularized solution, and $\lambda \geq 0$ is a parameter controlling the applied degree of regularization. The minimizer $x_\lambda$ of (66) can be formally computed in closed form as

$$x_\lambda = \text{argmin}_{x \in \mathbb{R}^n} \|Ax - b\|^2 + \lambda \|\Gamma x\|^2 = \min_{x \in \mathbb{R}^n} \|Ax - b\|^2, \quad (67)$$

where

$$\bar{A} = \left( \begin{array}{c} A \\ \sqrt{\lambda} \end{array} \right), \quad \bar{b} = \left( \begin{array}{c} b \\ 0 \end{array} \right). \quad (69)$$

The standard form of Tikhonov regularization has $\Gamma = I$, which simply controls the norm of $x_\lambda$. More interestingly, this choice also has the effect of improving the numerical conditioning of (66) versus (64). To see this, observe that (66) with $\Gamma = I$ can be expressed as an instance of (64):

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|^2 + \lambda \|\Gamma x\|^2 = \min_{x \in \mathbb{R}^n} \|\bar{A}x - \bar{b}\|^2, \quad (68)$$

where

$$\bar{A} = \left( \begin{array}{c} A \\ \sqrt{\lambda} \end{array} \right), \quad \bar{b} = \left( \begin{array}{c} b \\ 0 \end{array} \right). \quad (69)$$

The $n \times n$ block $\sqrt{\lambda} I$ of the augmented coefficient matrix $\bar{A}$ ensures that $\sigma_{\text{min}}(\bar{A}) \geq \sqrt{\lambda}$, and the Pythagorean theorem implies that $\sigma_{\text{max}}(\bar{A}) = \|A\| \leq \sqrt{\lambda + \|\Gamma\|^2}$. These two inequalities together imply that

$$\kappa_2(\bar{A}) \leq \sqrt{\frac{\lambda + \|\Gamma\|^2}{\lambda}} = \sqrt{1 + \frac{\|\Gamma\|^2}{\lambda}}. \quad (70)$$

Equation (70) shows that the regularization parameter $\lambda$ controls the effective conditioning of the Tikhonov-regularized system (66)–(69) when $\Gamma = I$.

We now highlight a remarkable connection between trust-region methods and Tikhonov regularization. Consider the optimality condition (37a) for the trust-region step in Theorem 1, and suppose that $B \geq 0$. Since $B$ is symmetric, it can be diagonalized by an orthogonal matrix $Q$:

$$B = Q \text{diag}(\lambda_1, \ldots, \lambda_n) Q^T \quad (71)$$

where $\lambda_1 \geq \cdots \geq \lambda_n \geq 0$ since $B \geq 0$; (71) is thus also a singular value decomposition for $B$, and $\kappa_2(B) = \lambda_1/\lambda_n$. The same $Q$ appearing in (71) also diagonalizes the matrix

$$\bar{B} = B + \lambda^* I \quad \text{appearing in (37a)}: \quad \bar{B} = Q \text{diag}(\lambda_1 + \lambda^*, \ldots, \lambda_n + \lambda^*) Q^T, \quad (72)$$

and therefore

$$\kappa_2(\bar{B}) = \frac{\lambda_1 + \lambda^*}{\lambda_n + \lambda^*} \leq \frac{\lambda_1}{\lambda_n} = \kappa_2(B), \quad (73)$$
where the inequality in (73) is strict if \( \lambda^* > 0 \).

Theorem 1 and equations (71)–(73) show that the trust-region step \( h_t \) determined by (34) can be interpreted as a kind of Tikhonov-regularized solution of the system (10) defining the (approximate) Newton step \( h_N \); indeed, in the specific case of the Gauss-Newton system defined by (13), the corresponding first-order optimality condition (37a) is

\[
(J(x)^T J(x) + \lambda^* I) h_t = -J(x)^T r(x),
\]

which is an instance of the Tikhonov-regularized system (67).

In particular, equation (73) shows that \( h_t \) is the solution of a system whose conditioning is always at least as good as that of the system (10) defining \( h_N \).

This analysis shows that trust-region methods are **innately robust** to ill-conditioning, and we therefore expect them to be particularly effective (as compared to pure Newton-type methods) whenever the approximate Hessians \( B_i \) used in the local quadratic models (8) are ill-conditioned.

V. RISE: INCREMENTALIZING POWELL’S DOG-LEG

In this section we present **Robust Incremental least-Squares Estimation (RISE)**, an incrementalized version of the IGN-PDL algorithm (Algorithm 4). For pedagogical clarity, we begin by following the original derivation of RISE as given in [24], in which IGN-PDL is incrementalized with the aid of iSAM. We then derive RISE2, a new and improved version of the RISE algorithm obtained by using iSAM2 [10] in place of the original iSAM.

A. Derivation of RISE

The IGN-PDL algorithm computes the approximate trust-region update step \( h \) from the Gauss-Newton step \( h_{GN} \) and the gradient descent step \( h_{gd} \) when \( R \) is nonsingular, and the Cauchy step \( h_C \) when \( R \) is singular (cf. lines 7–10 and 13 of Algorithm 4). As iSAM already implements an efficient incremental algorithm for computing \( h_{GN} \) in the nonsingular case, it remains only to develop efficient methods to compute \( h_{gd} \) and \( h_C \). In turn, lines 9 and 13 of Algorithm 4 show that \( h_{gd} \) and \( h_C \) are computed in terms of the gradient direction vector \( g \) and the scale factors \( \alpha \) and \( \kappa \) defined in (53). Thus, it suffices to determine efficient incrementalized versions of equations (53a)–(53c).

Letting \( x = (x_{old}, x_{new}) \) as before and substituting the block decompositions (21) and (22) into (53a) produces

\[
\hat{g} = \bar{J}(\hat{x})^T \bar{r}(\hat{x}) = \begin{pmatrix} J(\hat{x}_{old})^T J_{new}(\hat{x})^T \\ 0 \end{pmatrix} \begin{pmatrix} r_{old}(\hat{x}_{old}) \\ r_{new}(\hat{x}) \end{pmatrix} = \begin{pmatrix} J(\hat{x}_{old})^T r_{old}(\hat{x}_{old}) \\ 0 \end{pmatrix} + J_{new}(\hat{x})^T r_{new}(\hat{x}).
\]

Comparing the right-hand side of (75) with (53a), we recognize the product \( J(\hat{x}_{old})^T r_{old}(\hat{x}_{old}) \) as nothing more than \( \bar{g} = g_{old} \), the gradient direction vector of the original (i.e., unaugmented) system at the linearization point \( \hat{x}_{old} \). Thus, (75) can be reduced to

\[
\hat{g} = \begin{pmatrix} g_{old} \\ J_{new}(\hat{x})^T r_{new}(\hat{x}) \end{pmatrix}.
\]

Algorithm 5 The RISE algorithm

1: procedure RISE
2: Initialization: \( \hat{x}_{old}, \hat{x}_{estimate} \leftarrow x_0, \Delta \leftarrow \Delta_0 \).
3: while (\( \exists \) new data \( (\hat{x}_{new}, r_{new}) \)) do
4: if (relinearize) then
5: Update linearization point: \( \hat{x}_{old} \leftarrow \hat{x}_{estimate} \).
6: Construct Jacobian \( \bar{J}(\hat{x}_{old}, \hat{x}_{new}) \).
7: Perform complete QR decomposition on \( \bar{J}(\hat{x}) \), cache \( \bar{R} \) factor and right-hand side vector \( \bar{d} \) as in equations (18) and (19).
8: Set \( \hat{g} \leftarrow \bar{R}^T \bar{d} \).
9: else
10: Compute the partial Jacobian \( J_{new}(\hat{x}) \) in (24).
11: Obtain and cache the new \( \bar{R} \) factor and new right-hand side vector \( \bar{d} \) by means of Givens rotations as in equations (27) and (31).
12: Set \( \hat{g} \leftarrow \hat{g} + J_{new}(\hat{x})^T r_{new}(\hat{x}) \).
13: end if
14: if \( \bar{R} \) is nonsingular then
15: Compute Gauss-Newton step \( h_{GN} \) using (32).
16: Set \( \alpha \leftarrow \| \hat{g} \|^2 / \| \bar{R} \hat{g} \|^2 \).
17: Set \( h_{gd} \leftarrow -\alpha \hat{g} \).
18: Set \( \hat{x} \leftarrow \text{COMPUTE}_\text{DOG-LEG}(h_{GN}, h_{gd}, \Delta) \).
19: else
20: Compute \( \kappa \) using (80b).
21: Set \( \hat{x} \leftarrow -\kappa \hat{g} \).
22: end if
23: Set \( \hat{x}_{proposed} \leftarrow (\hat{x} + h) \).
24: Compute \( \rho \) using (38).
25: if \( \rho \geq \eta_1 \) then
26: Update estimate: \( \hat{x}_{estimate} \leftarrow \hat{x}_{proposed} \).
27: else
28: Retain current estimate: \( \hat{x}_{estimate} \leftarrow \hat{x} \).
29: end if
30: Set \( \Delta \leftarrow \text{UPDATE}_\text{DELTA}(\rho, \Delta, \eta_1, \eta_2, \gamma_1, \gamma_2) \).
31: Update cached variables: \( \hat{x}_{old} \leftarrow \hat{x}, \hat{r} \leftarrow \bar{r}, \hat{g} \leftarrow \hat{g}, \hat{R} \leftarrow \bar{R}, \hat{d} \leftarrow \bar{d} \).
32: end while
33: return \( \hat{x}_{estimate} \)
34: end procedure

Since the matrix \( J_{new}(\hat{x}) \) is sparse and its row dimension is equal to the (small) number of new measurements added when the system is extended, equation (76) provides an efficient method for obtaining the gradient direction vector \( \hat{g} \) for the augmented system by *incrementally updating* the previous gradient direction vector \( g \), as desired.

Furthermore, in addition to obtaining \( \hat{g} \) from \( g \) using (76) in incremental update steps, we can also exploit computations already performed by iSAM to more efficiently batch-compute \( \hat{g} \) during relinearization steps, when incremental updates cannot be performed. Substituting (29) into (53a), we obtain:

\[
\hat{g} = \begin{pmatrix} Q \bar{R} \\ 0 \end{pmatrix}^T \bar{r}(\hat{x}) = (\bar{R}^T 0) \bar{Q}^T \bar{r}(\hat{x}).
\]
Comparing (77) with (30) then shows that
\[
   \bar{g} = (\bar{R}^T \ 0) \begin{pmatrix} \bar{d} \\ \bar{e} \end{pmatrix} = \bar{R}^T \bar{d}.
\] (78)

The advantage of equation (78) versus equation (53a) is that \( \bar{R} \) is a sparse matrix of smaller dimension than \( \bar{J}(\hat{x}) \), so that the matrix-vector multiplication in (78) will be faster. Moreover, since iSAM already computes the factor \( \bar{R} \) and the right-hand side vector \( \bar{d} \), the factors on the right-hand side of (78) are available at no additional computational expense.

Having shown how to compute the vector \( \bar{g} \), it remains only to determine the scaling factors \( \alpha \) and \( \kappa \) as in (53b) and (53c). The magnitude of \( \bar{g} \) can of course be computed efficiently directly from \( \bar{g} \) itself, which leaves only the denominator \( \| \bar{J}(\hat{x})\bar{g} \|^2 \). To compute this quantity we again exploit the fact that iSAM already maintains the \( \bar{R} \) factor of the QR decomposition for \( \bar{J}(\hat{x}) \); for since \( \bar{Q} \) is orthogonal, then
\[
   \| \bar{J}(\hat{x})\bar{g} \|^2 = \left\| \begin{pmatrix} \bar{Q} & 0 \end{pmatrix} \bar{g} \right\|^2 = \left\| \begin{pmatrix} \bar{R} \end{pmatrix} \bar{g} \right\|^2 = \| \bar{R}\bar{g} \|^2,
\] (79)
and equations (53b) and (53c) are therefore equivalent to
\[
   \alpha = \frac{\| \bar{g} \|^2}{\| \bar{R}\bar{g} \|^2}, \tag{80a}
\]
\[
   \kappa = \begin{cases} \min \left\{ \frac{\Delta}{\| \bar{g} \|^2}, \frac{\| \bar{R}\bar{g} \|^2}{\| \bar{R}\bar{g} \|^2} \right\} , & \| \bar{R}\bar{g} \| > 0, \\ \frac{\Delta}{\| \bar{g} \|^2}, & \| \bar{R}\bar{g} \| = 0. \end{cases} \tag{80b}
\]

Again, since \( \bar{R} \) is sparse, the matrix-vector multiplication appearing in (80) is efficient.

Equations (76), (78), and (80) enable the implementation of RISE, a fully incrementalized version of the IGN-PDL algorithm that integrates directly into the existing iSAM framework (Algorithm 5).

### B. RISE2: Enabling fluid relinearization

In this section we present RISE2, an improved version of RISE obtained by replacing iSAM with iSAM2 [10] in the derivation given in Section V-A. iSAM2 improves upon iSAM by eliminating the need to periodically reevaluate and refactor the entire Jacobian (two very expensive batch operations) in order to relinearize the system about its corrected estimate (cf. Section III-C); instead, iSAM2 efficiently relinearizes the system at every iteration by applying direct updates only to those (few) rows of \( \bar{R} \) and \( \bar{d} \) that are modified when relinearization occurs, a process known as fluid relinearization. Similarly, RISE2 eliminates expensive periodic batch factorizations (cf. lines 4 to 8 in Algorithm 5) by applying iSAM2’s fluid relinearization in order to efficiently update \( \bar{R} \), \( \bar{d} \), and the gradient direction vector \( \bar{g} \) to their values at the updated estimate \( \hat{x}_{\text{estimate}} \) at every iteration.

Internally, iSAM2 achieves efficient computation by means of the Bayes Tree [25], which encodes a symbolic factorization of the \( \bar{R} \) factor and right-hand side vector \( \bar{d} \) of the linear system (20) obtained from the QR decomposition (18), (19). Each node in the tree stores pairs of the form \([\bar{R}_i, \bar{d}_i]\), where

\[
   \frac{\Delta}{\| \bar{R}\bar{g} \|^2}
\] (80b)

\[
   \alpha = \frac{\| \bar{g} \|^2}{\| \bar{R}\bar{g} \|^2}, \tag{80a}
\]

\[
   \kappa = \begin{cases} \min \left\{ \frac{\Delta}{\| \bar{g} \|^2}, \frac{\| \bar{R}\bar{g} \|^2}{\| \bar{R}\bar{g} \|^2} \right\} , & \| \bar{R}\bar{g} \| > 0, \\ \frac{\Delta}{\| \bar{g} \|^2}, & \| \bar{R}\bar{g} \| = 0. \end{cases} \tag{80b}
\]

Again, since \( \bar{R} \) is sparse, the matrix-vector multiplication appearing in (80) is efficient.

Equations (76), (78), and (80) enable the implementation of RISE, a fully incrementalized version of the IGN-PDL algorithm that integrates directly into the existing iSAM framework (Algorithm 5).

**Algorithm 6 The RISE2 algorithm**

1: **procedure** RISE2
2: Initialization: \( \hat{x}_{\text{estimate}} \leftarrow x_0, \Delta \leftarrow \Delta_0. 
3: \textbf{while} \ (\exists \text{ new data } (\hat{x}_{\text{new}}, r_{\text{new}})) \ \textbf{do}
4: \quad \text{Add new nodes for } r_{\text{new}} \text{ to Bayes Tree and set initial estimated update: } \hat{x} = (\hat{x}_{\text{estimate}}, \hat{x}_{\text{new}}). 
5: \quad \text{Apply Algorithm 6 of [10] to update the Bayes Tree, } R, \text{ and } d, \text{ and the linearization point } \hat{x}_{\text{lin}}. 
6: \quad \text{Initialize gradient } \bar{g} \text{ for augmented system:}
7: \quad \qquad \bar{g} \leftarrow \begin{pmatrix} \bar{g} \\ 0 \end{pmatrix}. 
8: \quad \textbf{for all} (\text{pairs } [R_i, d_i] \text{ modified in Step 5}) \ \textbf{do}
9: \quad \quad \text{Compute the updated contribution } \bar{g}_i \text{ to the gradient direction vector } \bar{g} \text{ coming from row } i:
10: \quad \quad \quad \bar{g}_i \leftarrow d_i R_i^T. 
11: \quad \text{Update gradient direction vector:}
12: \quad \quad \bar{g} \leftarrow \bar{g} - \bar{g}_i + \bar{g}_i. 
13: \quad \text{Cache updated value of } g_i: g_i \leftarrow \bar{g}_i. 
14: \quad \textbf{end for}
15: \quad \text{Cache updated value of gradient: } g \leftarrow \bar{g}. 
16: \quad \textbf{if} \ R \text{ is nonsingular then}
17: \quad \quad \text{Compute the Gauss-Newton step } h_{\text{GN}} \text{ using Algorithm 7 of [10].}
18: \quad \quad \text{Set } \alpha \leftarrow \frac{\| g \|^2}{\| R g \|^2}.
19: \quad \quad \text{Set } h_{gd} \leftarrow -\alpha g.
20: \quad \quad \text{Set } h \leftarrow \text{COMPUTE-DOG-LEG}(h_{\text{GN}}, h_{gd}, \Delta). 
21: \quad \textbf{else}
22: \quad \quad \text{Compute } \kappa \text{ using (80b).}
23: \quad \quad \text{Set } h \leftarrow -\kappa g.
24: \quad \textbf{end if}
25: \quad \text{Set } \hat{x}_{\text{proposed}} \leftarrow (\hat{x}_{\text{lin}} + h).
26: \quad \text{Compute } \rho \text{ using (38).}
27: \quad \textbf{if} \ \rho \geq \eta_1 \ \textbf{then}
28: \quad \quad \text{Update estimate: } \hat{x}_{\text{estimate}} \leftarrow \hat{x}_{\text{proposed}}.
29: \quad \quad \textbf{else}
30: \quad \quad \quad \text{Retain current estimate: } \hat{x}_{\text{estimate}} \leftarrow \hat{x}_{\text{lin}}.
31: \quad \quad \textbf{end if}
32: \quad \Delta \leftarrow \text{UPDATE_DELTA}(\rho, \Delta, \eta_1, \eta_2, \gamma_1, \gamma_2). 
33: \quad \textbf{end while}
34: \quad \textbf{return} \ \hat{x}_{\text{estimate}} 
35: \textbf{end procedure}

\( R_i \in \mathbb{R}^{1 \times n} \) is the \( i \)th row of the matrix \( R \) and \( d_i \in \mathbb{R} \) is the \( i \)th element of the right-hand side vector \( d \):

\[
   R = \begin{pmatrix} R_1 \\ \vdots \\ R_n \end{pmatrix} \in \mathbb{R}^{n \times n}, \quad d = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix} \in \mathbb{R}^{n \times 1} \tag{81}
\]

(cf. Fig. 3(c) in [10]). Updating \( R \) and \( d \) by adding new measurements to the system or relinearizing the system about its updated estimate \( \hat{x}_{\text{estimate}} \) can be implemented as simple, computationally efficient editing operations on the tree itself (cf. Algorithms 4 and 5 of [10], respectively); similarly, solv-
ing the updated system for the Gauss-Newton step is achieved by means of a simple single-pass algorithm flowing from the root towards the leaves (Algorithm 7 of [10]). As in the case of the original RISE algorithm, we obtain RISE2 by exploiting these computations to produce an efficient incremental update rule for the gradient direction vector \( g \).

According to (78), \( g \) can be computed from \( R \) and \( d \) as:

\[
g = R^T d. \tag{82}
\]

Substituting the block decomposition (81) into (82) produces

\[
g = R^T d = \left( R_1^T \cdots R_n^T \right) \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} = \sum_{i=1}^n d_i R_i^T, \tag{83}
\]

which expresses \( g \) as a linear combination of the rows of \( R \).

When fluid relinearization and updating are applied, some of the values \([R_i, d_i]\) in the nodes of the Bayes Tree may be modified. By means of (83), we can recover the updated value \( \bar{g} \) of the gradient direction vector corresponding to the new linearization point \( \bar{x}_{\text{estimate}} \) using only those quantities that the Bayes Tree already computes during this update. Specifically, we initialize \( \bar{g} \) according to:

\[
\bar{g} \leftarrow \begin{bmatrix} g \\ 0 \end{bmatrix}, \tag{84}
\]

and for each pair \([R_i, d_i]\) that was modified during the update of the Bayes Tree, we likewise update \( \bar{g} \) according to:

\[
\bar{g} \leftarrow \bar{g} - \begin{bmatrix} g_i \\ 0 \end{bmatrix} + \bar{g}_i, \tag{85}
\]

where

\[
g_i = d_i R_i^T, \quad \bar{g}_i = d_i \bar{R}_i^T \tag{86}
\]

here \([R_i, d_i]\) gives the values in row \( i \) prior to their update, and \([\bar{R}_i, \bar{d}_i]\) gives these values after their update. Equations (85) and (86) indicate that the update to \( \bar{g} \) due to row \( i \)'s update simply consists of subtracting off the contribution \( g_i \) to the gradient direction vector coming from row \( i \) prior to its update and replacing it with the updated contribution \( \bar{g}_i \).

Replacing the computation of \( g \) in the original RISE algorithm (lines 4 to 13 of Algorithm 5) with the new incremental update procedure (84)–(86) produces RISE2 (Algorithm 6).

We point out that in both cases RISE(2)'s efficiency and incrementality are a direct result of exploiting iSAM(2)'s pre-existing functionality for incrementally updating \( R \) and \( d \). In addition to being a purely intellectually pleasing result, this also means that any other computations depending upon pre-existing iSAM(2) functionality (for example, online covariance extraction for data association [26]) can proceed with RISE(2) without modification.

VI. RELATED WORK

There is a vast body of prior work on least-squares problems (1) and their solution; indeed, entire books have been devoted to this subject alone [27]. However, since our own interest in this problem is motivated by visual mapping applications, we will restrict our attention to prior work in the SLAM and bundle adjustment literature that attempts to solve (1) exactly.

In the SLAM literature, the first formulation of the full or smoothing problem as an instance of (1) is due to Lu & Milios [28], who proposed to solve it using the Gauss-Newton algorithm. This approach remains perhaps the most popular, with many well-known algorithms [10], [29]–[33] differing only in how they solve the linear system (10) corresponding to (13). Lu & Milios themselves originally proposed to solve (10) directly using dense matrix inversion, but the \( O(n^3) \) computational cost of this technique is only tractable for fairly small problems. Subsequent work achieved improved computational speeds by exploiting symmetric positive definiteness and sparsity in the approximate Hessian (13b) using iterative methods such as sparse preconditioned conjugate gradient [29] or (multiresolution) Gauss-Seidel relaxation [30]. Thrun & Montemerlo [31] exploited the sparse block structure of the Hessian in a direct method by using the Schur complement to solve first the (dense but low-dimensional) system corresponding to the robot pose update and then the (large but sparse) system corresponding to the landmark update. Grisetti et al. [32] exploited sparsity by implementing a hierarchical version of Gauss-Newton on a multiresolution pyramid and using lazy evaluation when propagating updates downward.

To further exploit sparsity, Dellaert & Kaess [34] conducted a thorough investigation of direct linear-algebraic techniques for efficiently solving sparse linear systems of the form (10). One surprising result of this analysis was the primacy of variable ordering strategies as a factor in the computational cost of solving these systems; indeed, the use of fast sparse matrix factorization algorithms (such as sparse multifrontal QR [35] or Cholesky [36] decompositions) coupled with good variable ordering heuristics [21] enables the linear system (10) to be solved in a fraction of the time needed to simply construct it (i.e., evaluate the Jacobian or Hessian). This insight in turn eventually led to the development of the Bayes Tree [25] and iSAM2 [10], which directly update the reduced linear system (20) at each iteration rather than reevaluating and refactorizing the Hessian in (10); this completely incremental approach is the current state of the art amongst Gauss-Newton-based optimization algorithms in robotics.

Alternatively, some SLAM approaches propose to solve (1) using first-order methods. These methods have excellent robustness properties, but are limited to a linear convergence rate since they do not take advantage of curvature information. Olson et al. [37] proposed to overcome this slow convergence by using a deterministic variation of Jacobi-preconditioned stochastic gradient descent with a clever parameterization that enables the algorithm to make more rapid progress through the state space. While originally conceived as a batch algorithm, Olson's method was later adapted to the online setting through the use of spatially-varying learning rates and lazy evaluation [38]. Grisetti et al. [39] improved upon Olson's original parameterization by ordering variables using a spanning tree through the network of constraints rather than temporally. These approaches were later unified to produce TORO [40].

In bundle adjustment, the optimization (1) is typically solved in the batch (offline) case, and therefore BA solutions generally emphasize good robustness and convergence properties rather than real-time computational speed. To that end,
many popular software packages used for BA (e.g. g2o [33], SBA [41] and sBA [42]) implement sparse batch versions of Levenberg-Marquardt to take advantage of its desirable robustness and numerical properties. One notable online approach is the \textit{continuable LM} method of [43], which (like iSAM and RISE) applies a single update step per iteration and (also like RISE) maintains the value of the damping parameter $\lambda$ across iterations; however, unlike RISE (which incrementally updates the linear system (10)) it completely reconstructs and refactors the modified normal equations (74) in each iteration, which are the most expensive parts of computing each update step.

In summary, while prior work has addressed subsets of the following criteria, we believe that the RISE algorithm is the first to satisfy all three simultaneously:

- **Speed**: We implement a numerical optimization method with a superlinear convergence rate and incrementally update the linear system (32) across iterations rather than recomputing it, thus achieving computational speeds comparable to state-of-the-art online incremental sparse least-squares solvers.

- **Robustness**: Unlike purely Gauss-Newton-based methods, RISE is provably robust to highly nonlinear systems and numerically ill-conditioned Jacobians and is globally convergent for a broad class of objective functions.

- **Generality**: We place no restrictions on the admissible parameterizations of the state variables, nor on the number of arguments to each of the functions $r_i$.

### VII. Experimental Results

In this section we illustrate the performance of the Powell’s Dog-Leg, Gauss-Newton, and Levenberg-Marquardt batch methods and the iSAM and RISE incremental methods on 1000 randomly-generated instances\(^3\) of the sphere2500 dataset \([10]\), a standard 6DOF pose-graph SLAM benchmark. In these experiments we use the implementations of these algorithms available in the iSAM v1.6 library with their default settings, and apply the pseudo-Huber robust cost function (cf. \([2\), Sec. A6.8]) with parameter $b = .5$.

#### A. Simulated data: sphere2500

Here we consider the performance of the Powell’s Dog-Leg in Section IV as the sparse least-squares optimization method of choice. All algorithms use the same stopping criteria, and (in the interest of time) are limited to a maximum of 500 iterations. The initial estimate of the robot path is obtained by integrating the simulated raw odometry measurements. Results from the experiment are summarized in Table I, and the solutions computed by each algorithm for a single representative problem instance are shown in Fig. 5.

As expected, Powell’s Dog-Leg and Levenberg-Marquardt obtain solutions of comparable quality, significantly outperforming Gauss-Newton. The superior performance of these algorithms can be understood in terms of their robust convergence properties and high tolerance for nonlinearity; it is clear from Table I that Gauss-Newton makes only marginal progress towards the minima in these examples.

Furthermore, in addition to its favorable accuracy, Powell’s Dog-Leg is also the fastest algorithm of the three by an order of magnitude, both in terms of the number of iterations necessary to converge to a local minimum and the total elapsed computation time. In this case the superior speed of Powell’s Dog-Leg versus Gauss-Newton is a consequence of its improved convergence; its superior speed versus Levenberg-Marquardt has been studied previously \([16]\), and is due in part to the fact that Powell’s Dog-Leg need only solve (10) for the Newton step once at each linearization point, whereas Levenberg-Marquardt must solve the modified normal equations (74) (an expensive operation) whenever the linearization point is updated or the damping parameter $\lambda$ is changed.

#### 2) Incremental methods

Next we compare the original iSAM algorithm with RISE (Algorithm 5); results are summarized in Table II (note that the statistics given for each method in the first and second rows of Table II are computed using only the set of problem instances for which that method ran to completion, as explained below).

\(^1\)The iSAM Library (version 1.6), available through http://people.csail.mit.edu/kaess/isam/isam_v1.6.tgz.

\(^2\)The GTSAM Library (version 2.1.0), available through https://research.cc.gatech.edu/borg/sites/edu.borg/files/downloads/gtsam-2.1.0.tgz.

\(^3\)Generated using the generateSpheresICRA2012.cpp executable in the iSAM v1.6 library.
As expected, RISE significantly outperformed iSAM in terms of final solution quality. In over half of the problem instances, the solution computed by iSAM diverged so far from the true minimum that the numerically-computed Jacobian became rank-deficient, forcing the algorithm to abort the computation (solving equation (20) for the Gauss-Newton step requires that the Jacobian be full-rank). Even for those problem instances in which iSAM ran to completion (which are necessarily the instances that are the “easiest” to solve), Table II shows that the solutions computed using the incremental Gauss-Newton approach have significantly greater costs than those computed using the incremental Powell’s Dog-Leg method. Indeed, RISE’s performance on all of the problem instances was, on the average, significantly better than iSAM’s performance on only the easiest instances.

RISE’s enhanced robustness versus iSAM does come at a slightly greater computational cost: each iteration of the RISE algorithm must compute the Gauss-Newton step (the output of iSAM) as an intermediate result in the computation of the dog-leg step. As shown in Algorithms 3 and 5, the cost of computing the dog-leg step given the Gauss-Newton step is dominated by the costs of the matrix-vector multiplications needed to compute the gradient descent step; since these have the same asymptotic time-complexity as the backsubstitution that iSAM already performs to compute the Gauss-Newton step, we expect that RISE will suffer at most a small constant-factor slowdown in speed versus iSAM. The results in Table II show that in practice this constant-factor slowdown has only a modest effect on RISE’s overall execution speed (an increase of about 20% versus iSAM) when the computational costs of manipulating the underlying data structures are also included: both iSAM and RISE are fast enough to run comfortably in real-time.

B. Visual mapping with a calibrated monocular camera

In this experiment we consider a significantly more challenging test scenario: visual mapping with a calibrated monocular camera via incremental bundle adjustment. Bundle adjustment is known to suffer from a litany of numerical challenges, including strong nonlinearities in the objective function (due to the nonlinear camera projection mappings, the rotational degrees of freedom in the camera pose estimates, and the use of robust cost functions) and poor numerical conditioning (which can be caused by unfavorable camera configurations or large variations across the uncertainties of the reconstructed camera poses and point positions) [3]. Successful bundle adjustment optimization methods must be able to robustly address these numerical difficulties.

The input for this experiment consists of a short (141 second) 640 × 480 monochrome video sequence recorded by the left camera of a Bumblebee2 stereocamera as it was hand-scanned over a room-sized static scene containing 46 AprilTags [44]. A set of keyframes and point observations was extracted from this video by sampling at 1 Hz (to provide a reasonable baseline between successive keyframes) and then
selecting as keyframes all images in which at least 3 AprilTags were detected; this yielded 104 keyframes containing 3876 observations of the 184 AprilTag corner points. These observations were used to estimate the 3D camera pose of each keyframe and the 3D position of each corner point via bundle adjustment (i.e. by minimizing the sum of reprojection errors under the Huber robust cost function (cf. [2, Sec. A6.8]) with parameter $b = 1$); the camera’s internal calibration was estimated using Zhang’s method [45] immediately prior to recording the video and treated as a fixed constant. No prior information about the AprilTags’ geometry was included in the adjustment; the tags were used only to solve the data association problem, which is beyond the scope of this work. Ground truth was also acquired by tracking the position of the camera using a Vicon system.

Incremental bundle adjustment was performed using the iSAM2 and RISE2 implementations available in the GTSAM v2.1.0 library (here one iteration comprised the incorporation of all observations from a single keyframe). Camera poses were initialized using the EPnP algorithm [46] (whenever at least four previously-initialized points were in view) or the two-view homogeneous DLT method of [2, Sec. 9.6]; points were initialized using either the homogeneous DLT triangulation method of [2, Sec. 12.2] or the two-view homogeneous method of [2, Sec. 9.6]. Results from the experiment are shown in Fig. 6 and Table III.

The RISE2 algorithm successfully processed all 104 frames, converging to a final solution with an objective function value of 558.8 and a raw RMS reprojection error of .3797 pixels. In contrast, the iSAM2 algorithm did not even run to completion on this example; in the 31st iteration it reported that the Jacobian $J(x^{(31)})$ was numerically rank-deficient and aborted the computation. A closer investigation reveals that the Gauss-Newton update step applied by iSAM2 in iteration 30 overshot the true objective function minimum by moving 5 points behind several of the cameras from which they were visible near the newly-initialized camera for keyframe 30 (cf. Fig. 6(c) and Table III); this led to a poor initialization of the camera pose for keyframe 31 and the subsequent failure of the algorithm.

Our experience has been that this failure mode (overshooting followed by the rank-deficiency of the Jacobian computed in the next iteration) is generic when iSAM(2) is applied to highly nonlinear or ill-conditioned systems. In contrast, RISE(2)’s use of the trust-region approach enables it to reliably overcome these numerical challenges.

### VIII. Conclusion

In this paper we presented RISE, an incremental trust-region method for robust online sparse least-squares estimation. RISE improves upon current state-of-the-art sequential sparse least-squares solvers by providing superior robustness to objective function nonlinearity and numerical ill-conditioning while simultaneously exploiting prior advances in incremental optimization to achieve fast online computation. In addition to deriving the RISE algorithm itself, we also provided a thorough theoretical analysis of its desirable numerical properties and proved its global convergence for a broad class of inferential cost functions (twice-continuously differentiable functions with bounded sublevel sets). Finally, we evaluated the algorithm empirically in simulation using standard 6DOF pose-graph SLAM benchmark datasets, and demonstrated its
superior performance on a more challenging real-world example arising in the context of online visual mapping.

In addition to its utility in SLAM and visual mapping tasks, we believe that the RISE algorithm can also be advantageously applied to other numerically challenging online inference problems sharing the common mathematical structure (2); we are particularly interested in its use as a superlinear optimization method for online machine learning with kernels [5]. Recent work in the robotics [47] and computer vision [48] literature has shown promising results in this direction, but utilizes Gauss-Newton-based approaches that (as demonstrated herein) limit the class of objective functions that can be reliably employed. It would be interesting to consider applications of RISE (and related techniques) in the context of online machine learning in future work.

APPENDIX

Proof of Theorem 3: We prove Theorem 3 by means of Theorem 2. Since \( \mathcal{L}_S(x(0)) \) is bounded, there exists \( D > 0 \) such that \( \|x - x(0)\| < D \) for all \( x \in \mathcal{L}_S(x(0)) \). Define the convex open set

\[
\Omega = \{ x \in \mathbb{R}^n \mid \|x - x(0)\| < D \}.
\]

Then \( \Omega \) is a closed and bounded (hence compact) subset of \( \mathbb{R}^n \) containing the sublevel set \( \mathcal{L}_S(x(0)) \). Equation (11b) gives the elements of the Hessian matrix \( \frac{\partial^2 S}{\partial x^2} \), which are continuous since \( r \in C^2 \), and therefore \( \frac{\partial^2 S}{\partial x^2} \) attains a maximum value \( L \) on the compact set \( \Omega \); \( L \) is thus a Lipschitz constant for \( \nabla S \) on \( \Omega \). The same argument applied to the approximate Hessians \( B \) whose elements are defined by equation (12) shows that \( \|B\| \) likewise attains a maximum \( \beta \) on \( \Omega \). Finally, \( S \) (as a sum-of-squares function) is clearly lower-bounded by 0 everywhere. This establishes the initial hypotheses of Theorem 2.

We now check each of the enumerated criteria (i)-(v) in turn. Condition (i) is immediate from the definitions of the dog-leg and constrained Cauchy steps in and (46) and (50). Condition (ii) is satisfied with \( \varsigma = 0 \) since the gradients used in Algorithm 4 are exact (cf. equation (13a)). Condition (iii) is satisfied using the value of \( \beta \) obtained in the preceding paragraph. Condition (iv) is trivially satisfied in the case of the Cauchy step \( h_C \), since this step is by definition always antiparallel to the gradient (cf. equation (50)). In the case of the dog-leg step, we observe using (44) that

\[
\|h_{dl}\| = \left\| \frac{\|g\|^2}{g^T B g} \right\| = \frac{\|g\|^3}{g^T B g} \geq \frac{\|g\|^3}{\beta \|g\|^2} = \frac{\|g\|}{\beta},
\]

so that (by virtue of the computation in Algorithm 3) \( h_{dl} \) is also antiparallel to the gradient whenever

\[
\Delta^{(i)} \leq \frac{\|g^{(i)}\|}{\beta} \leq \|h_{dl}^{(i)}\|.
\]

Equation (89) will hold for all \( i > N \) for some sufficiently large \( N \) by virtue of the additional hypotheses (56) in effect for the purposes of this condition.

Finally, for condition (v), consider the predicted decrease for the Cauchy step \( h_C = -\kappa g \) defined in (50) and (51):

\[
\text{pred}(h) = \kappa \|g\|^2 - \frac{1}{2} \kappa^2 g^T B g.
\]

The full (i.e. unconstrained) Cauchy step has \( \kappa = \frac{\|g\|^2}{g^T B g} \), in which case (90) becomes:

\[
\text{pred}(h) = \left( 1 - \frac{1}{2} \right) \frac{\|g\|^4}{g^T B g} g^T B g,
\]

Equations (91)–(93) show that the Cauchy step defined in (50) satisfies the uniform predicted decrease condition (59) with \( c = 1 \) and \( \sigma = \beta \). For the dog-leg step, we simply observe that by virtue of (45), (46) and Lemma 1, \( \text{pred}(h_{dl}) \geq \text{pred}(h_C) \) whenever \( h_{dl} \) exists. This proves condition (v), and completes the proof.

ACKNOWLEDGMENTS

The authors would like to thank F. Dellaert and R. Roberts for the RISE2 implementation in the GTSAM library.

REFERENCES
