A Globalization Approach for the Constrained Gauss-Newton Method

Naya Baslan
Department of Microsystems Engineering
University of Freiburg, Germany
naya.baslan@hotmail.com

Jonas Hall
Department of Mathematics
University of Freiburg, Germany
jonas.hall@uranus.uni-freiburg.de

***Abstract*—This project studies the Gauss-Newton algorithm and provides a functioning algorithm based on published literature. The main idea is to solve nonlinear least squares problems subject to an equality constraint. The flow of the algorithm is described, as well as the use of the merit function and its linearization to ensure global convergence. The algorithm is then tested on an example obtained from simulated data. The simulated model is described and the results are presented. Comparisons with currently available MATLAB functions is made and shows where the implemented algorithm is more efficient**

***Index Terms*—nonlinear least squares problem, generalized Gauss-Newton, optimization.**

I. Introduction

This project studies the Gauss-Newton algorithm and provides a functioning algorithm based on [1]. The main idea is to solve a nonlinear least squares problem subject to the equality constraint G. In the literature, such problem formulation is used for parameter estimation problems whose applications range from pharmaceutical to economic spectra. Most encountered problems of this type can be formulated as nonlinear least square problems. The book about numerical optimization [3] explains the main use of such problems as measuring the accuracy of a model at estimating data given different time steps. By formulating such an objective function and minimizing it using efficient algorithms, one would be able to give accurate parameter estimate values that result in the model closely resembling the collected data. This, in fact, is the main of the algorithm presented in this project.

Several numerical optimization concepts will be used in this report. They are all mainly concerned with implementing a Gauss-Newton algorithm that is globally convergent. In terms of this report and in numerical optimization generally, globalization refers to the ability of an algorithm to converge efficiently given remote starting points that are far away from the exact solution. Results presented at the end of this report, show that the algorithm will accept different starting points and still generate accurate results during an acceptable computation time. However, it can be easily noted that convergence rate increase rapidly with the number of iterations i.e. as the values approach the exact solution, the algorithm becomes faster.

The structure of the report is as follows: In the section II, the theoretical background of the project is described along with a summary of similar work. Further, it describes how the algorithm works in practice and provides a pseudo-code describing its flow. Section III presents the results of testing the algorithm on a set of data that simulates the trajectory of a volleyball. Global convergence is tested and the behaviour of the algorithm is analyzed at different starting values. A comparison to a well-known MATLAB solver can be found in IV. Lastly, in section V the report is concluded with a short discussion and a final summary. All figures and tables can be found in section VII at the very end of this report.

II. Theory

Before giving a step-by-step explanation of the algorithm's functionality, let us formulate the applicable problem class, and mention the notations used throughout this report.

A. The basics

Let $R : \mathbb{R}^n \to \mathbb{R}^m$, and $G : \mathbb{R}^n \to \mathbb{R}^l$ be twice continuously differentiable functions, assume $l \leq n \leq m$, and let $\|\cdot\|$ denote the Euclidean norm. Then we can formulate the *equality constrained non-linear least squares optimization problem*

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \|R(x)\|^2 \\ \text{subject to} \quad & G(x) = 0. \end{aligned} \tag{LSQNL}$$

We will refer to R as the *residual function*. Further, let us assume that the rank conditions

$$\operatorname{rank}\left(G'(x)\right) = l, \quad \operatorname{rank}\begin{pmatrix} R'(x) \\ G'(x) \end{pmatrix} = n \tag{1}$$

hold for all $x \in \mathbb{R}^n$.

For any function R we will denote its jacobian matrix by R', and abbreviate function evaluation at iteration steps x_k by $R_k = R(x_k)$. To avoid confusion between dimension and iteration step, we will denote dimension in the upper index, e.g. the identity matrix of dimension n is I^n. Finally, the Moore-Penrose Pseudo Inverse of a matrix M will be denoted by M^+.

B. Related Work

The book [3] describes the use of merit functions to ensure the global convergence of an algorithm and divides the approach into two types. The first one is based on the use of line search methods, which alter the step size to ensure convergence. The second method, which is merely studied,

but not fully implemented in this project, is the trust region method. Here the step is evaluated as being acceptable or not. Furthermore, the book relies on the use of l1 merit functions, while this project uses l2 merit functions. Similar ideas and explanations are also summarized in the lecture notes provided to us in this course [5] The importance of the use of a merit function is motivated in details in [4]. To summarize a few important points, we refer to the following points from that paper: Step sizes are modified per iterate to ensure the merit function is reduced at each iteration Constrained minimization problems require a merit function to be defined to be comparable with the merit function that is naturally available in unconstrained problems (namely the objective function itself) It is impossible that to devise an algorithm that is globally convergent at any arbitrary starting point. Therefore, some simplifications and assumptions are made and accepted as means of global convergence. Furthermore, the terms merit functions and penalty functions are interchangeably used in the literature. The book [3] describes these two terms as being equivalent.

C. Describing the globalization

The algorithm is based on the generalized Gauss-Newton Method and uses a line search that achieves global convergence. Let us motivate these two key parts of the algorithm. We follow [1].

Step 1: Finding an adequate step direction. This is done in a common manner for generalized Gauss-Newton methods. We obtain an improved direction d_k as a solution of the linearized least squares problem, which results from approximating both the residual and the constraint by a first order Taylor expansion at the given iteration point x_k. Formally, we shall solve

$$\begin{aligned} \min_{d_k \in \mathbb{R}^n} \quad & \|R'(x_k)d_k + R(x_k)\| \\ \text{subject to} \quad & G'(x_k)d_k + G(x_k) = 0. \end{aligned} \tag{LSQL}$$

By rank assumption (1), we find a unique solution

$$d_k^* = -{G'_k}^+ G_k + (R'_k E_k)^+ (R'_k {G'_k}^+ G_k - R_k), \tag{2}$$

where $E_k = (I^n - {G'_k}^+ G'_k)$ is called the orthoprojector. The representation (2) follows from first order optimality conditions for the linear least squares problem, and was deduced in [1].

The next two steps will guarantee globalization. For $\mu > 0$, let us define the penalty function

$$\psi(x, \mu) = \|R(x)\| + \mu \|G(x)\|, \tag{3}$$

together with its linearized model

$$\phi(x, p, \mu) = \|R'(x)p + R(x)\| + \mu \|G'(x)p + G(x)\|.$$

As stated in [1], a minimizer for (LSQNL) can be found by locally minimizing the penalty function, given μ large enough. We will describe the line search proposed in the paper.

Step 2: Line search For non stationary x we get $d \neq 0$, and choose μ to satisfy

$$\psi(x, \mu) - \phi(x, d, \mu) > 0. \tag{4}$$

If this condition holds, d is a descent direction for ψ. Now let $\alpha \in (0, 1]$ be small enough, such that

$$\psi(x, \mu) - \psi(x + \alpha d, \mu) \geq \delta \left(\psi(x, \mu) - \phi(x, \alpha d, \mu)\right) \tag{5}$$

where $\delta \in (0, 1)$ is fixed. That this is always possible given μ satisfying (4) will be stated as follows.

Theorem 1. *Let $x, s \in \mathbb{R}^n$, and assume μ satisfies* (4)*. Further, let R', G' be Lipschitz continuous with constants L_F and L_G, respectively. Then there exists a number $eta \in (0, 1]$ such that 5 holds for all $\alpha \in (0, \eta]$.*

We assume R and G to be twice continuously differentiable on $\mathbb{R}^n$, so if we consider R' and G' on a bounded subset we will have Lipschitz continuity. This means that for practicable problems the assumptions of this theorem will be satisfied. Now let us make sure that such a μ can be found. The proof of the theorem will be omitted here.

Step 2.1: Updating penalty parameter. This is done by introducing a lowest bound function ω that μ needs to exceed. Let us call $P = R'E(R'E)^+$ the projector. If $(\|R\| + \|R'd + R\|)\|G\| = 0$ we will set $\omega(x) = 0$, else define

$$\omega(x) = \frac{[R + (I^m - P)(R - R'G'^+G)]^\top (I^m - P)R'G'^+G}{(\|R\| + \|R'd + R\|)\|G\|}. \tag{6}$$

Lemma 2. *Let μ satisfy*

$$\mu > |\omega(x)|, \tag{7}$$

and assume x is not stationary. Then the condition (4) *holds.*

The proof of this lemma can be found in [1]. In the algorithm we take more precaution in updating μ. Let μ_k, ω_k describe the respective values in iteration k. We introduce $0 < \bar{\mu_1} < \bar{\mu_2} < \infty$. We set $\mu_{k+1} = |\omega_k| + \bar{\mu_2}$ if $\mu_k < |\omega_k| + \bar{\mu_1}$, and else we keep the old value. Finally, we can combine this result with the Theorem, and finish the line search.

Step 2.2: Updating damping factor. After having ensured that the penalty parameter is chosen well, we can find an adequate damping factor according to Theorem 1. This is simply done by using a scalar $\gamma \in (0, 1)$, and taking the smallest $j \in \mathbb{N}$, such that $\gamma^j \alpha$ satisfies (5). Then let the damping factor for this iteration be $\gamma^i \alpha$. The factor γ should not be chosen too small, since a large damping factor may negatively impact the speed of convergence.

D. Pseudo-code

When writing the algorithm we included two external functionalities to compute jacobians and Moore-Penrose pseudo inverse matrices. For computing jacobians we decided to use the sophisticated MATLAB software `CasADi`, which computes precise jacobian matrices via algorithmic differentiation.

Details about this can be read in [2]. The latter was done using the MATLAB method `pinv`. We present the pseudo code that shows the rough structure of the algorithm. Explicit computations have been omitted.

```
% Initialize
delta = 0.4; gamma = 0.8;
mu_bar = [1,2]; mu = 0;
alpha = 1; k = 1;
x = x_init;

% THE ALGORITHM.
while(true)
  % Step (1)
  Solve LSQL and update d_k.
  if (norm(d_k) <= tolerance)
    break;
  end

  % Step (2.1)
  Update penalty parameter mu

  % Step (2.2)
  alpha = min(alpha/gamma, 1);
  while( lineSearchCondition )
    alpha = gamma*alpha;
  end

  % Step (3)
  x = x + alpha*d; k = k + 1;
end
```

E. Convergence

We give the in [1] stated and proven convergence theorem regarding this algorithm.

Theorem 3.

(*i*) *Let R, G be twice continuously differentiable functions satisfying the rank condition* (1)*. Then the algorithm is well defined for any starting point $x \in \mathbb{R}^n$, i.e., it either terminates after a finite number of steps with a stationary point x^*, or else it is infinite, and there holds $d_k \neq 0$ for all k.*

(*ii*) *If the algorithm is infinite, and additionally all iterates x_k belong to a compact set W on which R' and G' are both Lipschitz continuous, then each accumulation point of $(x_k)_{k\in\mathbb{N}}$ is stationary.*

III. Testing the Algorithm

The example used to test the behaviour of the algorithm was obtained from a set of simulated data. The model example is a system with two cameras that are located one meter apart. Figure 1 will give intuition of the problem setup. Both cameras are recording the trajectory of a volleyball being thrown. The data obtained resembles the projection of the two dimensional coordinates of the volleyball as projected on the image plane of each camera. The optimization problem is formulated in such as way that the algorithm should predict the original three dimensional position of the volleyball, given the two dimensional coordinates from each camera.

Figure 2 shows the recorded ball positions from the two cameras separately. We will use the algorithm to find the best fitting curve using a state model and a measurement model that will be described below. As we will see, the state model will depend on the time step that two points are apart, and the acceleration due to gravity. Our goal will be to find the curve that best fits the scattered data, and additionally

III-A estimating the size of the time step given the value of the acceleration due to gravity,

III-B estimating the value of acceleration due to gravity, given the size of the time step.

From now on, let g represent the acceleration due to gravity, and h represent the time step. The state model

$$F(X,h,g) = F\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ v_1 \\ v_2 \\ v_3 \\ h \\ g \end{pmatrix} = \begin{pmatrix} p_1 + hv_1 \\ p_2 + hv_2 \\ p_3 + hv_3 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -\frac{1}{2}gh^2 \\ 0 \\ 0 \\ -gh \end{pmatrix}$$

is a mapping between state coordinates X, where (p_1, p_2, p_3) describe the position, and (v_1, v_2, v_3) describe the velocity. The measurement model $G : \mathbb{R}^6 \to \mathbb{R}^4$ is mapping a state point X to a measurement point $Y = (y_1, y_2, y_3, y_4)$, where (y_1, y_2) describe X in the coordinate system of the first camera, and (y_3, y_4) describe X in the coordinate system of the second camera. The explicit form of M will be omitted here. We can express the residual function of N measured points $Y_1, \ldots, Y_N$ as

$$R(X_1, \ldots, X_{N+1}, h, g) = \begin{pmatrix} X_2 - F(X_1, h, g) \\ \vdots \\ X_{N+1} - F(X_N, h, g) \\ Y_1 - M(X_1) \\ \vdots \\ Y_N - M(X_N). \end{pmatrix} \quad (8)$$

The residual function will vanish if there exists neither state nor measurement noise. Since we are interested in solving a non linear least squares optimization problem with equality constraint, we will in fact assume, that there exists no state noise, i.e. we will omit the first N rows of R, and include the constraint

$$G(X_1, \ldots, X_{N+1}, h, g) = \begin{pmatrix} X_2 - F(X_1, h, g) \\ \vdots \\ X_{N+1} - F(X_N, h, g). \end{pmatrix} \quad (9)$$

Finally we can formulate our two experiments.

A. Experiment 1 – Estimating the value of h

We use the algorithm with residual function $R_A(X_1, \ldots, X_{N+1}, h, 9.81)$, and equality constraint $G_A(X_1, \ldots, X_{N+1}, h, 9.81)$. Table I shows the results of the algorithm at three different tests. The exact value of `h` is `0.05`, which is precisely approximated by the algorithm. The values `|R_0|` and `|R_star|` represent the initial and final value of the objective function respectively. We can see the estimated model positions in figure 3. With `|G_0|`, and `|G_star|` we describe the initial and final value of the constraint, and we see that the constraint does in fact vanish at the minimizer. By `k` we denote the number of iterations required for the algorithm to converge to the final solution. The total time spent until the solution is found is `T_total`, while `T_CasADi` is the time spent using CasAdi to calculate the jacobians of R and G.

B. Experiment 2 – Estimating the value of g

The same experiment is repeated, but with g as the unknown instead of h. Similarly as above, we choose the residual function $R_B(X_1, \ldots, X_{N+1}, 0.05, g)$, and equality constraint $G_B(X_1, \ldots, X_{N+1}, 0.05, g)$. It can be depicted from table II, that the algorithm was much faster for smaller initial values of g. However, unexpectedly, it did not show a faster rate of convergence when the initial g value was 10 in comparison to when it was 1, even though 10 is a closer value to the exact solution. Studying figure 4, we find that the objective function is decreasing extremely quickly, which results in convergence after just a few steps.

IV. Comparison to other solvers

The algorithm is not restricted to solving constrained non-linear least squares problems. One can give a trivial constraint, i.e. $G = 0$ in order to have an unconstrained non-linear least squares problem (this will violate the rank assumption, which is ignored here as a special case). The function `lsqnonlin` is a well known, and in practice often used MATLAB tool for solving such problems. We therefore reformulated our problem III-A by excluding the equality constraint, i.e. we will consider the residual function as formulated in (8). We should remind ourselves, that the constraint implied that there would not exist state noise, which lead to an accurate approximation of the time step. After removing this assumption, we will no longer expect a precise value. The comparison results presented in table III show that the proposed algorithm is around five to ten times faster than lsqnonlin. This suggests that `lsqnonlin` was originally for local convergence, where the initial value is very close to the final solution. However, as the results from Test 1 show, the proposed algorithm is still about 4.85 times faster at conditions which are very close to local convergence. When other remote starting points are chosen, a higher difference in speed is observed. The graph in figure 5 shows the ratio of the speeds of both functions at different starting points. To further prove that the algorithm used here outperforms lsqnonlin, we calculated the difference between the norms of the optimal value of R when using both algorithms. At all tested points, the value achieved by the generalized Gauss-Newton algorithm was lower than that achieved by lsqnonlin. This proves that the former is better at solving minimization problems.

V. Conclusion and future development

To sum it all up, this project implemented and tested the Gauss-Newton algorithm and used it to solve an nonlinear constrained least squares problem. The problem was solved numerically and line search was used in the algorithm. The step-size used to approach the solution changes with each subsequent iteration until the algorithm converges to the final solution. Several tests were performed using the aforementioned example and it was shown that the algorithm converges to a solution not only when it starts with an initial guess that is close to the exact solution, but also when it starts at remote points that are much further away.

One could improve the efficiency of this algorithm by somewhat normalizing the residual and constraint functions. Extremely flat or steep functions are a numerical burden, and multiplying the functions by a scalar will not impact the minimizer.

Another suggestion is to use trust region method instead of line search and compare to the convergence rate of both methods. Initial tests were conducted using the trust region method. However, by the time of submitting this report it was not yet fully implemented.

VI. Acknowledgement

This project was done under the supervision of Professor Dr. Moritz Diehl as part of the course in Numerical Optimization. Sincere thanks are extended to Florian Messerer for his continuous guidance throughout our work on the project, and to Katrin Baumgrtner for providing us with the data that was used in the example. We would like to express our great appreciation to them for their valuable assistance and advice.

References

[1] O. Knoth, , "A globalization scheme for the generalized Gauss-Newton method," Numerische Mathematik, vol. 56, pp. 591–607, June 1989.
[2] J. Anderson, J. Gillis, and M.Diehl, "User Documentation for CasADi v3.3.0-194.aldla5d78."
[3] J. Nocedal and S. J. Wright, "Numerical optimization," New York: Springer, 2006.
[4] "P. T. Boggs and J. W. Tolle, Sequential Quadratic Programming, Acta Numerica, vol. 4, p. 1, 1995.
[5] M. Diehl, "Lecture Notes on Numerical Optimization (Preliminary Draft)," March 2016

VII. Figures and tables

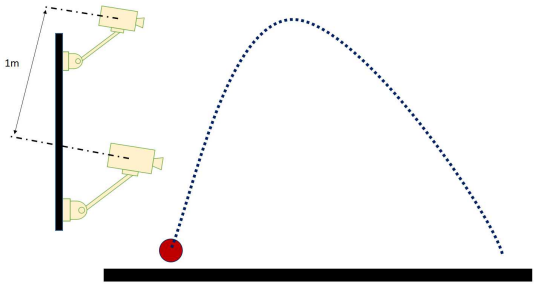

Fig. 1. Depicting the setup of the simulation.

	Test 1	Test 2	Test 3
h_0	0.010	0.743	5.721
h_star	0.0499	0.0499	0.0499
|R_0|	3.0E2	5.0E3	4.3E4
|R_star|	1.3E1	1.3E1	1.3E1
|G_0|	0.050	0.050	0.050
|G_star|	8.6E-15	9.3E-15	7.9E-15
k	11	13	20
T_total [s]	1.8080	1.502	1.6990
T_CasADi [s]	0.6020	0.620	0.640

TABLE I
TESTING THE ALGORITHM WITH PROBLEM III-A WITH THREE DIFFERENT INITIAL VALUES FOR THE TIME STEP.

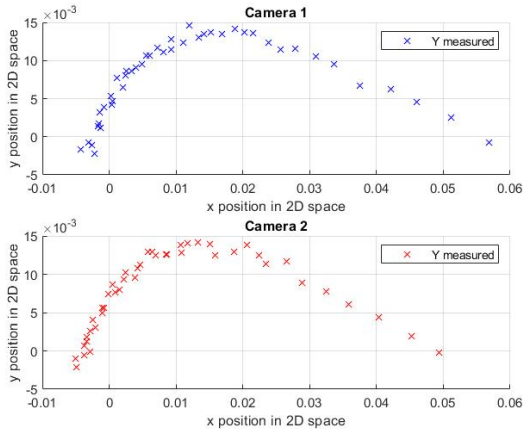

Fig. 2. Measured data points of the volleyball position on camera screen.

	Test 1	Test 2	Test 3
g_0	1	10	100
g_star	9.7917	9.7917	9.7917
|R_0|	3.2E2	1.5E2	2.4E3
|R_star|	1.3E1	1.3E1	1.3E1
|G_0|	0.050	0.050	0.050
|G_star|	8.7E-15	7.4E-15	1.3E-14
k	9	9	14
T_total [s]	1.6430	1.666	1.866
T_CasADi [s]	0.5490	0.616	0.603

TABLE II
TESTING THE ALGORITHM WITH PROBLEM III-B WITH THREE DIFFERENT INITIAL VALUES FOR THE ACCELARTION DUE TO GRAVITY.

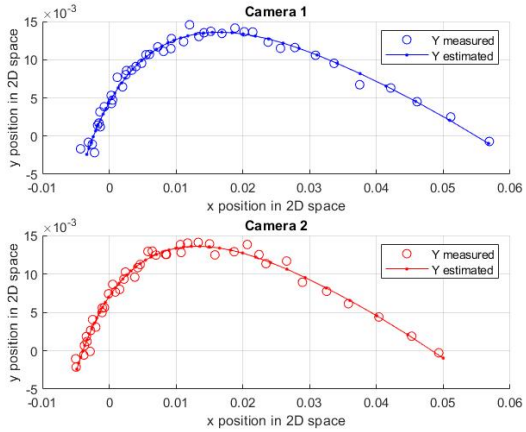

Fig. 3. Result of the fitted curve after projecting via measurement model.

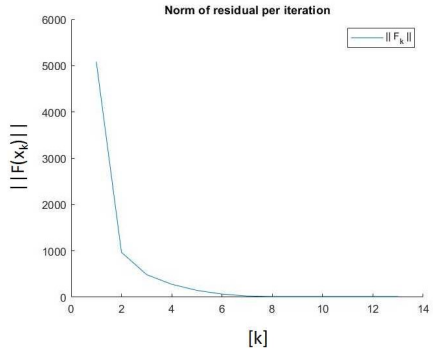

Fig. 4. Plot of residual norm against iteration steps.

	Test 1	Test 2	Test 3
h_0	0.05	0.91	6.25
h_star_ggn	0.0498	0.0498	0.0498
h_star_lsq	0.0498	0.0498	0.0498
|R_0|	1.56E2	6.35E3	4.72E4
|R_star_ggn|	1.33E1	1.33E1	1.33E1
|R_star_lsq|	1.33E1	1.33E1	1.33E1
T_ggn [s]	1.11	4.88	10.74
T_lsqnonlin [s]	5.34	41.37	73.29

TABLE III
COMPARING THE ALGORITHM TO LSQNONLIN USING THE UNCONSTRAINED VERSION OF III-A.

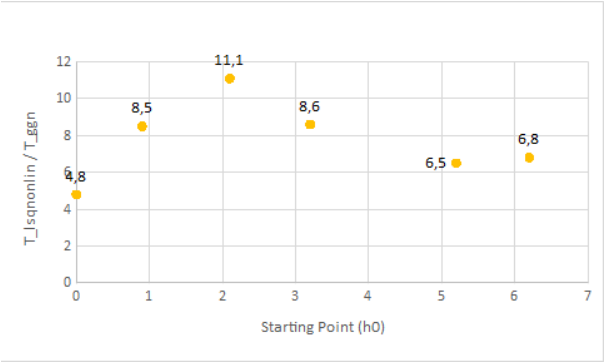

Fig. 5. Plotting $\frac{T\_lsqnonlin}{T\_ggn}$ against initial time step.