Journal of Computing and Information Technology - CIT 21, 2013, 2, 125–135
doi:10.2498/cit.1002186

A Derivative-free Algorithm for Finding Least Squares Solutions of Quasi-linear and Linear Systems

Nikica Hlupić[1], Ivo Beroš[2] and Danko Basch[3]

[1] Department of Applied Computing, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
[2] Department of Applied Mathematics and Computer Science, VERN' University of Applied Sciences, Zagreb, Croatia
[3] Department of Control and Computer Engineering, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia

A novel derivative-free algorithm for solving quasi-linear systems is presented. It resembles "classical" optimization approach but greatly simplifies computation, resulting in fast execution and numerical stability. Though the global convergence cannot be guaranteed, it turns out that the presented algorithm finds a solution as successfully as other commonly accepted methods. The algorithm is clearly developed and mathematically founded, and its properties are examined by comparisons with other methods.

Keywords: derivative-free algorithm, optimization, quasi-linear systems, linear systems

1. Introduction

Quasi-linear systems are important because they are common in various fields of science, like mechanical and civil engineering or robotics and control theory, to mention a few of them. Although there are few theoretical results, like Bezout theorem [12], that provide some knowledge about existence and number of solutions of such systems, there is no known analytical procedure to find them. Therefore, we try to compute solutions numerically and in practice we do not know in advance whether an exact solution exists at all, so we wish to find the best possible solution. Thus, we set the problem like the problem of minimization of residual sum of squares (*rss*) in the system and apply an optimization method to find solution according to least squares (LS). Because equations in quasi-linear system are easily differentiable, practically all optimization methods can be tried and the choice depends on the insight into the problem and prior knowledge about possible solutions. So-called *line search* methods like gradient, Newton's (Levenberg-Marquardt variant), Quasi-Newton and possibly conjugate gradient [1],[13] will be good choice if we can find starting point not too distant from solution for otherwise they will likely find just a local optimum or not converge at all. If we do not have any clue about solution (i.e., we cannot even estimate it), the alternative is to apply derivative-free methods like Nelder-Mead [1],[7],[13]. Finally, if the feasible set (possible values of unknowns) is closed (see, for example, [1]) or if we have some prior knowledge about the feasible set, we can assign probabilities to candidate points. This opens the possibility for using various heuristics (simulated annealing, particle swarm, ant colony etc.; also called *stochastic* algorithms) and genetic algorithms, which can be effective as well [3],[7].

The main drawback of all these methods is that they search for better point only locally in some limited area around the current point (limited either by the boundaries of feasible set or by knowledge about probability distribution of candidate points). In this paper we present an algorithm that overcomes this limitation [6] and finds the best point on the whole line of search direction, that is, it finds the global minimizer along search direction. Having set the search direction, the computation is easy and straightforward, and the only cost paid for this

benefit is a restricted set of search directions. This is quite a weak restriction because suitable search directions are directions of coordinate axes, which are exactly the ones we would probably try as the first choice if we could freely choose direction. In the case that some other direction is unconditionally required, there is always the possibility to rotate the coordinate system so that one of coordinate axes becomes coincident with the desired direction. This is, in fact, what happens in so-called adaptive coordinate descent method [9]. Hence, arbitrary search directions are possible, but with a more involved computation. On the other hand, simplicity, numerical stability and success-fulness in solving systems of up to moderate complexity, indicated by simulations in the later sections of the paper, confirm considerable practical value of the proposed algorithm, even in its simplest form.

2. Mathematical Background and Formulation of the Algorithm

Quasi-linear systems are systems of nonlinear equations in which equations can consist not only of sums of unknowns multiplied by coefficients, but also of mutual products of unknowns in all combinations. Formally, with vector of unknowns $\mathbf{x} = [x_1 x_2 \ldots x_n]^{\mathrm{T}} \in \mathbf{R}^n$, quasi-linear system is a system

$$\begin{aligned} F_1(\mathbf{x}) &= f_{11}(\mathbf{x}) + f_{12}(\mathbf{x}) + \ldots + f_{1q}(\mathbf{x}) = b_1 \\ F_2(\mathbf{x}) &= f_{21}(\mathbf{x}) + f_{22}(\mathbf{x}) + \ldots + f_{2q}(\mathbf{x}) = b_2 \\ &\vdots \\ F_m(\mathbf{x}) &= f_{m1}(\mathbf{x}) + f_{m2}(\mathbf{x}) + \ldots + f_{mq}(\mathbf{x}) = b_m \end{aligned} \tag{1}$$

in which all $f(\mathbf{x})$ are of the form [12]

$$f(\mathbf{x}) = a \cdot x_1^{p1} x_2^{p2} \ldots x_n^{pn}, \tag{2}$$

where a are real numbers and powers p_i are either zero or one ($p_i = 0$ means that variable x_i does not figure in f). Introducing notation $\mathbf{F} : \mathbf{R}^n \to \mathbf{R}^m$, $\mathbf{F}(\mathbf{x}) = [F_1(\mathbf{x}) F_2(\mathbf{x}) \ldots F_m(\mathbf{x})]^{\mathrm{T}}$ and $\mathbf{b} = [b_1\ b_2 \ldots b_m]^{\mathrm{T}} \in \mathbf{R}^m$, we write the problem to be solved as

$$\mathbf{F}(\mathbf{x}) = \mathbf{b}. \tag{3}$$

The measure of the balance of each equation in the system is the difference $r_i = b_i - F_i$ between its left and right sides, called residual. For convenience, we collect all residuals in a vector of residuals $\mathbf{r} = [r_1\ r_2 \ldots r_m]^{\mathrm{T}} \in \mathbf{R}^m$. To solve the system means to find vector $\mathbf{x}$ for which all residuals will be zero. Since we do not know in advance whether such an $\mathbf{x}$ exists at all, the objective is to balance the system as well as possible, that is, to minimize residual sum of squares (rss)

$$rss = \sum (b_i - F_i)^2 = ||r||^2 \geq 0, \tag{4}$$

where the exact solution is found when $rss = 0$.

Minimization of rss is common optimization approach, but from here on, we continue in a different direction. While other methods try to determine all unknowns simultaneously, i.e., they search in all n dimensions, we compute only one variable at a time. This simplifies the problem by reducing its dimensionality at the cost of getting a better value of only one variable in the system, instead of all of them.

The idea of the algorithm is to update one by one variable and the key for this idea to be useful is a special computation which, as we shall see shortly, guarantees that updating any of the variables will improve the solution or, in the worst case, it will remain the same. The computation is based on considering all but one variable known and fixed in each step of the algorithm. Because the variable considered unknown changes, let us denote it by λ. To explain the computation, let x_1, for example, be considered the only unknown and $x_2, \ldots, x_n$ known and fixed at their current values. Substituting known values into the system (1) and writting λ instead of x_1 we get a system of the form

$$\begin{aligned} a_{11} \cdot \lambda + a_{12} \cdot \lambda + \ldots + a_{1q} \cdot \lambda &= b_1 \\ a_{21} \cdot \lambda + a_{22} \cdot \lambda + \ldots + a_{2q} \cdot \lambda &= b_2 \\ &\vdots \\ a_{m1} \cdot \lambda + a_{m2} \cdot \lambda + \ldots + a_{mq} \cdot \lambda &= b_m. \end{aligned} \tag{5}$$

Coefficients a_{ij} in (5) contain contributions of all known variables $x_2, \ldots, x_n$ as well as of original coefficients in the system. Reduced system

(5) is nothing but

$$\begin{aligned}(a_{11}+a_{12}+\ldots+a_{1q})\cdot\lambda{=}v_1\cdot\lambda{=}b_1\\(a_{21}+a_{22}+\ldots+a_{2q})\cdot\lambda{=}v_2\cdot\lambda{=}b_2\\\vdots\\(a_{m1}+a_{m2}+\ldots+a_{mq})\cdot\lambda{=}v_m\cdot\lambda{=}b_m\end{aligned}\tag{6}$$

or, with $\mathbf{v}=[v_1\ v_2\ \ldots\ v_n]^{\mathrm{T}}\in\mathbf{R}^n$, simply

$$\mathbf{v}\cdot\lambda=\mathbf{b}.\tag{7}$$

System (6), i.e., (7) is a linear system of m equations in one unknown λ and its LS solution can be relatively simply computed as [5],[10],[14]

$$\lambda_{\mathrm{LS}}=(\mathbf{v}^{\mathrm{T}}\mathbf{v})^{-1}\mathbf{v}^{\mathrm{T}}\mathbf{b}=\frac{\sum_{i=1}^{m}v_ib_i}{\sum_{i=1}^{m}v_i^2}=\frac{\sum_{i=1}^{m}v_ib_i}{|\mathbf{v}|^2}.\tag{8}$$

Formula (8) yields the best possible value (in the whole set $\mathbf{R}$) of the variable considered unknown, with other variables fixed at their current values. The new value of the selected variable will certainly be better than the previous one because by Gauss-Markov theorem [5],[10],[14] it is guaranteed that new *rss* will necessarily be less or, in the worst case, equal to the one obtained by the old value of the selected variable. The explained procedure is an efficient computational engine for minimizing *rss* in a single dimension that we shall use in higher-level algorithms, so let us call it *quasi-linear computational engine* (QLCE) and summarize here for convenience.

Algorithm 1: QLCE

1. Given the current point $\mathbf{x}$ and one of coordinate axes as search direction (let it be j^{th} axis), consider the corresponding variable x_j unknown.
2. In system (1) substitute current values for all variables considered known and denote x_j as λ. This will render the system into a linear system in λ of the form (6), i.e., (7).
3. If $||v||\neq 0$, solve (7) for λ according to (8). Let us denote the result as λ_{LS}. If $||v||=0$, just return some indication of failure to the calling routine.
4. Compose new point $\mathbf{x}_{\mathrm{new}}$ by replacing x_j in $\mathbf{x}$ by λ_{LS}. This $\mathbf{x}_{\mathrm{new}}$ minimizes the *rss* of the original system (1) in selected dimension (i.e., along the line of selected coordinate axis).

Having QLCE computation at disposal, we immediately arrive to a natural extension of its underlying idea. The approach is to consider one by one variable unknown, with all other variables fixed at their current values, and to update them sequentially in order to gradually approach the solution, i.e., *rss* minimizer. Such an algorithm certainly exhibits descent property [1] because *rss* is necessarily reduced in each step (in the worst case it remains the same). Let us name it *quasy-linear solver* (QLS) and formulate it precisely.

Algorithm 2: quasi-linear solver (QLS)

1. Set $\mathbf{x}$ =starting point $\mathbf{x}_0$.
2. `for` $j=1$ to n
   - Consider x_j unknown and apply QLCE to find its better value. In the case of QLCE failure just return to the incremental part of the loop, i.e., take another variable as unknown and repeat QLCE computation.
   - Here QLCE returned $\mathbf{x}_{\mathrm{new}}$. To use it in further computation, set $\mathbf{x}=\mathbf{x}_{\mathrm{new}}$.
3. Check stopping conditions and if none is satisfied, continue from the step 2.

Remark: in step 2b, it is sufficient to take only j^{th} component of $\mathbf{x}_{\mathrm{new}}$, that is, QLCE can be stopped as soon as λ_{LS} is obtained and only this λ_{LS} has to be substituted for x_j to construct new $\mathbf{x}$ in step 2b of QLS.

Unfortunately, improving one by one dimension does not guarantee convergence to a stationary point. Moreover, simulations show that occasionally even the order of axes taken as search directions has a major impact on the final result. Sometimes just the change of the order means the difference between falling into a local optimum and finding an exact solution. In the worst case, the algorithm can enter an infinite loop in which updating one (let us say, the first) variable causes such changes of the other ones that the next update of the first variable yields its previous value with which the cycle begun. There is not much we can do about it, but such difficulties motivate somewhat different variant of the algorithm. Namely, instead of

sequential updating of unknowns in some fixed order, we apply greedy [2] strategy. This means that in each iteration we check all dimensions (every variable is once taken as the unknown and resulting *rss* is computed), but eventually we update only the variable for which achievable *rss* reduction is maximal. This is the underlying logic of the algorithm we call *greedy quasi-linear solver* (QLSG).

Algorithm 3: greedy quasi-linear solver (QLSG)

1. Given the current point $\mathbf{x}$, compute referential *rss*, that is, set $rss_{\text{ref}} = rss(\mathbf{x})$. Also, set $rss_{\min} = rss_{\text{ref}}$ and $\mathbf{x}_{\text{best}} = \mathbf{x}$.
2. `for` $j = 1$ to n
   - Consider x_j unknown and apply QLCE to find its better value. In the case of QLCE failure just return to the incremental part of the loop, i.e., take another variable as unknown and repeat QLCE computation.
   - Here QLCE returned $\mathbf{x}_{\text{new}}$ so compute $rss_{\text{new}} = rss(\mathbf{x}_{\text{new}})$.
   - If $rss_{\text{new}} < rss_{\min}$ set $rss_{\min} = rss_{\text{new}}$ and $\mathbf{x}_{\text{best}} = \mathbf{x}_{\text{new}}$.
3. Set $\mathbf{x} = \mathbf{x}_{\text{best}}$.
4. Check stopping conditions and if none is satisfied, continue from the beginning.

Remark: in steps 2a and 2b again we only need to work with λ_{LS}, rather than with the whole $\mathbf{x}$.

There are several possible stopping conditions and we discuss those that we use in the section on comparative study. Most of them are common to the majority of optimization methods, but QLSG algorithm, specifically, has to care about $||\mathbf{v}||$ and prevent potential problem that $||\mathbf{v}|| = 0$ in (8), which is the task of step 3 in QLCE. In the early stages of QLSG this is a very unlikely situation because zero norm of $\mathbf{v}$ (with nonzero $\mathbf{b}$) means that the problem is not well defined and this, in turn, means that in the selected dimension improvement is not possible at all. However, as we approach the solution or a stationary point, numerical limitations can cause inability to solve (8) in any dimension, which is then indication for stopping the execution. In fact, in that case $\mathbf{x}$ will not change at all so this can be detected by watching $\mathbf{x}$, as is done routinely in other methods as well. In practice, such situations are relatively rare and in the next section we present comparative study that confirms the merit of QLSG.

3. Comparative Study

We shall compare QLSG with two commonly accepted methods. Specifically, with Newton's method (Levenberg-Marquardt modification) that uses simple backtracking for *line-search* and with Nelder-Mead method as a representative of derivative-free methods. Because solving quasi-linear systems is challenging only for systems with three or more equations and unknowns, visualization of algorithms advance is not feasible. Therefore, the main criterion for comparison will be successfulness of the algorithms in finding a solution, i.e., the quantity of interest in our work is the number of solved systems out of systems tried, that is, proportion of solved systems which we denote by θ.

Reliable establishment of differences among the algorithms (i.e., proportions of solved systems) requires some kind of analysis of variance (ANOVA) [4],[11],[14], but to facilitate analysis of results, we construct confidence intervals for estimates of proportions and present them graphically. In experiments like ours, where we work with so-called count data [11], proportion is a variable having the binomial distribution. However, for large number of samples (more than 30, and we always have many more than that) we can use the normal approximation to the binomial distribution so we may compute confidence intervals according to [11]

$$\hat{\theta} - z_{\alpha/2}\sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}} < \theta < \hat{\theta} + z_{\alpha/2}\sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}}, \tag{9}$$

where $\hat{\theta}$ is the observed proportion of successes in n trials (n systems) and $z_{\alpha/2}$ is the value of a standard normal variable corresponding to the degree (level) of confidence $(1-\alpha)$. We use $\alpha = 0,05$, which means that there is 95% probability that true proportion lies within the confidence interval.

In all simulations we randomly generate quasi-linear systems with at least one known solution (also randomly generated) and try to solve them by all three compared algorithms. Coefficients in the system and components (variables) of

the known solution are random integers in the range $[-10, 10]$. Nelder-Mead simplex method is identical to Matlab® variant implemented in `fminsearch` function, while QLSG and Newton's method (Levenberg-Marquardt) are the authors' custom routines. Stopping conditions are the same for both:

1. sufficiently small gradient components (smaller than 10^{-6})
2. insufficient change of solution compo-nents (relative change smaller than 10^{-15})
3. insufficient reduction of *rss* (relative reduction smaller than 10^{-14})
4. maximal number of iterations exceeded.

Starting point in all simulations is always the origin of coordinate system (zero-vector) and the system is considered solved if the final *rss* drops below 10^{-6}. This is an absolute threshold, though "the optimal" threshold might be expected to be related to initial *rss* and the structure of the system. However, absolute tresholds are justified and common in optimization because in the case that we find "the right search path", the algorithm can reduce *rss* arbitrarily, down to numerical precision of computer, so we can freely use any absolute threshold. On the other hand, if we take wrong direction, *rss* reduction will be minor and even after thousands of iterations it would remain far above any reasonable threshold anyway. In such sitations it is much better to restart the algorithm (from some other starting point).

First we consider small systems of three equations in three unknowns. We use notation e3x3q4, which means "equations 3", "*x*es 3" and maximally 4 combinatorial terms (with products of unknowns) in an equation. Maximal number of iterations is 20000 and we generate 500 systems. Result of comparison is in Figure 1. We see that with these small systems QLSG is as successful as "classical" Levenberg-Marquardt (LM) algorithm, while both are better than Nelder-Mead method. Successfulness of QLSG and LM algorithm is about 75%, but insight into simulation output reveals that LM algorithm on average needs just 40 and QLSG about 3600 iterations. If we take into account the fact that LM in each iteration updates all unknowns and QLSG updates only one, fair comparison requires that we divide total number of QLSG iterations by the number of variables. With this correction, the average number of QLSG iterations is 1200, which is still much more than LM needs. This is a typical outcome because LM search direction follows from the second-order Taylor approximation of the objective function, which is quite a good approximation, while QLSG search directions are merely directions of coordinate axes so QLSG cannot take "shortcuts" on its way to the solution. On the other hand, QLSG does not need derivatives and, more important, it does not require equal number of equations and unknowns in the system, which is its second major advantage over Newton's and similar methods.

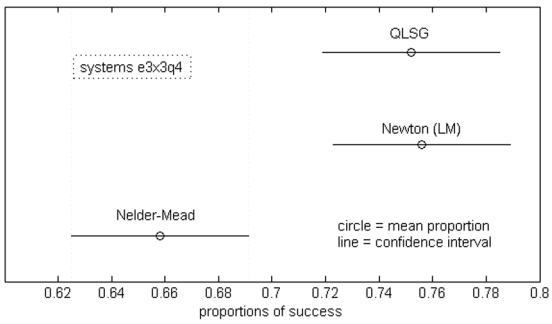

Figure 1. Proportions of successes with systems e3x3q4 and 95% confidence intervals of proportions.

In real life applications, we usually encounter overdetermined systems [5],[10] (systems with more equations than unknowns) and they rarely have exact solution. Nevertheless, it is usually desirable to obtain the best possible, i.e., least squares, solution, even when we know that an exact one does not exist. QLSG (or QLS) is very usefully in such situations because it is designed to reduce *rss* as much as possible and in that way, in effect, it tries to find least squares solution. There is, of course, no guarantee that QLSG will yield true least squares solution because it can finish in a local optimum or fail for some other reason. Also, there is no known closed form formula or algorithm that is guaranteed to yield least squares solution of a quasi-linear system, so it is not possible to verify QLSG result anyway. Nonetheless, the fact is that QLSG reduces *rss* and thereby it approaches true least squares solution, whatever it be.

Increasing complexity of the system quickly deteriorates all methods, which is apparent in Fig-

ure 2 and Figure 3 that show results for 500 systems e5x5q4 and e10x10q4, respectively.

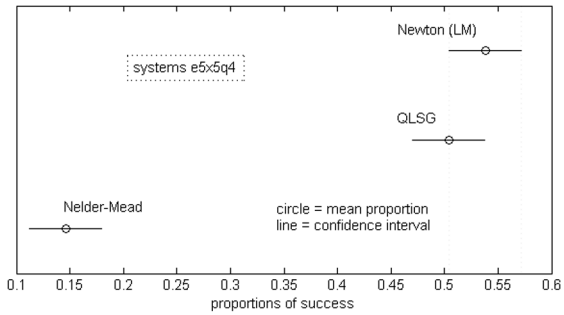

Figure 2. Proportions of successes with systems e5x5q4 and 95% confidence intervals of proportions.

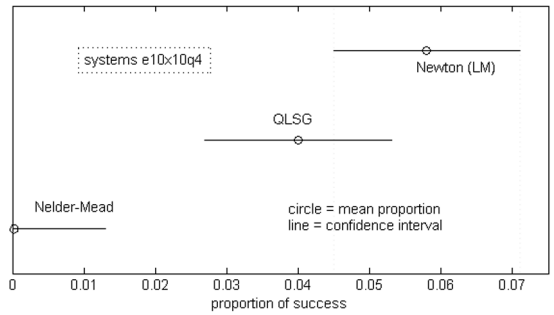

Figure 3. Proportions of successes with systems e10x10q4 and 95% confidence intervals of proportions.

We see that proportion of success (values on abscissa) rapidly decreases as complexity of system increases. Observing these figures, it could be concluded that QLSG continually becomes worse than LM as complexity of systems increases. This would, however, be a hasty conclusion because the simulations allowed limited number of iterations, which caused QLSG to end before it exhausted all its capabilities. When QLSG is let running until some other stopping condition (not number of iterations) breaks the execution, it becomes as successful as LM, if not better. Unfortunately, in bad circumstances it can easily reach a few tens of thousands iterations before it finds solution, but it mostly does find it. This is illustrated in Figure 4, which shows QLSG performance for one of the systems e5x5q4 which has not been solved in previous simulation due to limited number of iterations. The limit was 25000 iterations and, as before, the end point (the last $\mathbf{x}$ vector) was considered a solution if the final *rss* was less than 10^{-6}. As we see in the figure, for the system under consideration this *rss* value is reached at 26330^{th} iteration and though QLSG can solve it, in the previous simulation it was counted as unsolved.

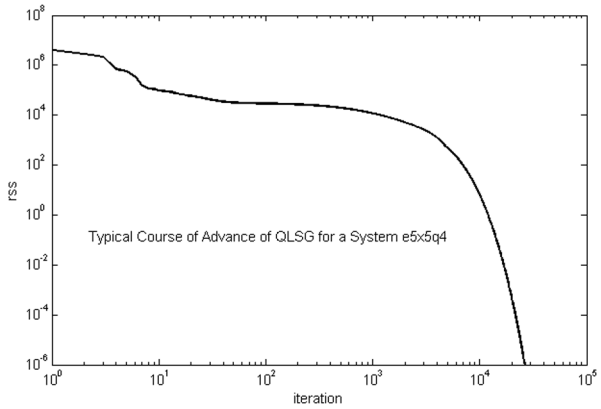

Figure 4. Log-log graph of typical course of solving e5x5q4 system, i.e., *rss* reduction by QLSG algorithm.

The "trajectory" of *rss* in Figure 4 shows a typical course of advance of QLSG. In the beginning it quickly reduces *rss* for few orders of magnitude, but then it usually needs many iterations to make many small movements until it comes into "the basin of attraction" of a solution. Eventually, once it gets close enough to the solution, the algorithm rapidly descends to it. Thus, if it is important to solve the system and time is not a limiting factor, QLSG might be the best algorithm for this task. This follows from the fact that QLCE computation yields the best point on the whole line of search direction, while other methods can search only locally in some limited area around the current point. Due to this property, QLSG has good chances to get out of local optimums and purely accidentally find solution, even when this is apparently not likely.

4. QLS Applied to Linear Systems

Because linear systems are a subclass of quasi-linear systems defined by the constraint that each term of the form (2) can contain only one unknown, it is clear that QLSG algorithm can also be used for solving linear systems. There is nothing to be changed in the algorithm formulation and all its properties observed in the section

on comparative study remain in effect when it is applied to linear systems as well. However, applying the basic (non-greedy) variant, i.e. QLS algorithm, to a linear system reveals a surprising connection between QLS algorithm and Gauss-Seidel method [5],[8] for solving linear systems. Namely, it turns out that, when applied to a pure linear system $\mathbf{Ax} = \mathbf{b}$, the QLS algorithm is nothing but another form of Gauss-Seidel method applied to the system $\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$. Thus, we can establish the following result.

Proposition: *When applied to a linear system* $\mathbf{Ax} = \mathbf{b}$*, QLS algorithm is equivalent to Gauss-Seidel method applied to the system* $\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$. □

Proof: We prove the statement by showing that QLCE computation of new value of a single unknown, when applied to a linear system and sequentially to all unknowns, is algebraically equivalent to Gauss-Seidel computation.

In each iteration, QLS algorithm applied to $\mathbf{Ax} = \mathbf{b}$ tries to reduce $rss(\mathbf{x}) = \mathbf{r}(\mathbf{x})^{\mathrm{T}}\mathbf{r}(\mathbf{x}) = ||\mathbf{b} - \mathbf{Ax}||^2$ by changing only one x_k. To this aim, it fixes all other unknowns and solves $d/dx_k[rss(x_k)] = 0$. Expanded, the condition $d/dx_k[rss(x_k)] = 0$ is

$$2(\mathbf{b} - x_1 \cdot \mathbf{a}_1 - \ldots - x_k \cdot \mathbf{a}_k - \ldots - x_n \cdot \mathbf{a}_n)^{\mathrm{T}} \cdot \mathbf{a}_k = 0,$$

where $\mathbf{a}_j$ means j^{th} column of matrix $\mathbf{A}$. By isolating x_k it follows that new x_k by QLS is

$$\begin{aligned} x_k = & \frac{(\mathbf{b} - x_1\mathbf{a}_1 - \ldots - x_{k-1}\mathbf{a}_{k-1}}{(\mathbf{a}_k)^{\mathrm{T}}\mathbf{a}_k} \\ & + \frac{-x_{k+1}\mathbf{a}_{k+1} - \ldots - x_n\mathbf{a}_n)^{\mathrm{T}}\mathbf{a}_k}{(\mathbf{a}_k)^{\mathrm{T}}\mathbf{a}_k}. \end{aligned} \tag{10}$$

If we introduce notation $\mathbf{C} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$ and $\mathbf{d} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$ and have in mind that matrix $\mathbf{C}$ is symmetric, formula (10) can be written as

$$\begin{aligned} x_k = & \frac{d_k - x_1c_{k1} - \ldots - x_{k-1}c_{k,k-1}}{c_{kk}} \\ & + \frac{-x_{k+1}c_{k,k+1} - \ldots - x_nc_{kn}}{c_{kk}}. \end{aligned} \tag{11}$$

On the other hand, new value of y_k yielded by Gauss-Seidel method, when applied to the system $\mathbf{Cy} = \mathbf{d}$, is given by

$$\sum_{i=1}^{k-1} y_i c_{ki} + y_k c_{kk} + \sum_{i=k+1}^{n} y_i c_{ki} = d_k \tag{12}$$

wherefrom we get

$$\begin{aligned} y_k &= \frac{d_k}{c_{kk}} - \sum_{i=1}^{k-1} y_i c_{ki} - \sum_{i=k+1}^{n} y_i c_{ki} \\ &= \frac{d_k - y_1c_{k1} - \ldots - y_{k-1}c_{k,k-1}}{c_{kk}} \\ &+ \frac{-y_{k+1}c_{k,k+1} - \ldots - y_nc_{kn}}{c_{kk}}. \end{aligned} \tag{13}$$

Comparing new x_k by QLS in (11) with new y_k by Gauss-Seidel method in (13), we see that they are identical, so for any starting point $\mathbf{x}^{(0)} = \mathbf{y}^{(0)}$ the sequence of new values of unknowns produced by either method will be identical. ■

The equivalence of QLS and Gauss-Seidel method sheds a new light on Gauss-Seidel method, but more importantly, it enables transferring of all theoretical results developed for Gauss-Seidel to QLS. Hence, because the convergence of Gauss-Seidel method, when applied to positive definite systems [5],[10], is proven [5],[8], and $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ certainly is positive definite [5],[8],[10], we may assert that convergence of QLS algorithm applied to any linear system is guaranteed as well. In the case that system has no solution at all, QLS algorithm will converge to a least squares solution $\mathbf{x} = (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$, which is a desired property.

Guaranteed convergence motivates further optimization of the algorithm with respect to specifics of linear systems in order to accelerate the algorithm as much as possible. Indeed, it is possible to reduce the number of operations by taking a deeper insight into a two consecutive steps, so let us follow what QLS does with system $\mathbf{Ax} = \mathbf{b}$, which we expand here for convenience. We have m equations in n unknowns

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m. \end{aligned} \tag{14}$$

QLS takes one unknown, let us say x_1, and constructs overdetermined system with m equations in a single unknown x_1 by subtracting all terms

that do not contain x_1 from the right sides in all equations. Hence, QLS constructs the system

$$\begin{aligned} a_{11}x_1 &= b_1 - a_{12}x_2 - \ldots - a_{1n}x_n \\ a_{21}x_1 &= b_2 - a_{22}x_2 - \ldots - a_{2n}x_n \\ &\vdots \\ a_{m1}x_1 &= b_m - a_{m2}x_2 - \ldots - a_{mn}x_n \end{aligned} \tag{15}$$

or in common matrix-vector notation

$$\mathbf{a}_1 x_1 = \mathbf{b} - \sum_{j=2}^{n} \mathbf{a}_j x_j. \tag{16}$$

New x_1 is obtained by solving (16) according to (8), where $\mathbf{a}_1$ from (16) has the role of $\mathbf{v}$ in (7) and the right side of (16) has the role of $\mathbf{b}$ in (7). In the next step QLS takes x_2 as the only unknown and constructs

$$\mathbf{a}_2 x_2 = \mathbf{b} - \sum_{\substack{j=1 \\ j\neq 2}}^{n} \mathbf{a}_j x_j, \tag{17}$$

where it uses new x_1 obtained in the previous step. Looking at (16) and (17), we notice that by switching from x_1 to x_2, the right sides of the systems to be solved contain mainly the same terms. The difference is only that in (16) we subtract $\mathbf{a}_2 x_2$, while in (17) this term does not contribute to the right side and in (17) we substract $\mathbf{a}_1 x_1$ (new x_1), which was not subtracted in (16). Hence, once we compute the right side in (16), computing the right side in (17) does not require all calculations as given by the formula. It is sufficient to add $\mathbf{a}_2 x_2$ to the right side obtained by (16) in order to cancel subtraction of $\mathbf{a}_2 x_2$ in the previous step, and to subtract $\mathbf{a}_1 x_1$ (with new x_1) from the result. If we denote the right sides of these systems when computing x_i as $\mathbf{t}_i$, we can write all calculations needed to transform the right side in two consecutive steps as a recursive formula

$$\mathbf{t}_i = \mathbf{t}_{i-1} + \mathbf{a}_i x_i - \mathbf{a}_{i-1} x_{i-1}. \tag{18}$$

Relation (18), along with the fact that the left sides of systems of the form (16) and (17) are simply i^{th} columns of system matrix $\mathbf{A}$, enables significant reduction of the number of needed operations and leads to a highly optimized variant of the algorithm designed specifically for solving linear systems. Of course, because (18) is a recursive formula, the algorithm has to be started properly as specified in the following list.

Algorithm 4: optimized linear solver

1. Set $\mathbf{x}$ =starting point $\mathbf{x}_0$.
2. Compute all possible denominators in (8) that can appear (there will be n of them) and store them appropriately, let us say in array den.
3. Compute the right side of (16) taking $\mathbf{x}_n$ as the unknown, that is, compute
$$\mathbf{t}_n = \mathbf{b} - \sum_{j=1}^{n-1} \mathbf{a}_j x_j.$$
4. `for` $i = 1$ to n
   - Use recursive formula (18) to compute the right side of (16) for x_i as the unknown, that is, compute
   $$\mathbf{t}_i = \mathbf{t}_{i-1} + \mathbf{a}_i x_i - \mathbf{a}_{i-1} x_{i-1}.$$
   If $i = 1$, use n instead of $i - 1$ to properly handle the circularity of sequence of variables.
   - Solve $\mathbf{a}_i x_i = \mathbf{t}_i$ for x_i according to (8), whereas the value of denominator is den[i]. Denote new x_i as λ_{LS}.
   - Replace x_i in the current $\mathbf{x}$ by λ_{LS}.
5. Check stopping conditions and, if none is satisfied, start new sequence of updates, i.e., continue from step 4.

Step 2 is not necessary, but it accelerates calculation (8) in step 4b at the negligible cost of some extra storage. Step 3 is the required initialization of recursion as it simulates the non-existing previous state of recursion (state as would be before repeated arrival to x_1).

To illustrate the practical value of QLS and its properties when applied to a linear system, let us consider a simple example. System

$$\begin{aligned} x_1 + x_2 + x_3 &= 3 \\ x_1 - x_2 + x_3 &= 3 \\ x_1 - x_2 - x_3 &= 1 \end{aligned} \tag{19}$$

is neither diagonally dominant nor positive definite and Gauss-Seidel method fails to solve it, though the solution $\mathbf{x} = [2\ 0\ 1]^{\text{T}}$ exists and can

be easily found. In contrast, QLS (all variants) does not have any problem with this system and the course of the algorithm's advance (i.e., reduction of *rss*) is shown in Figure 5.

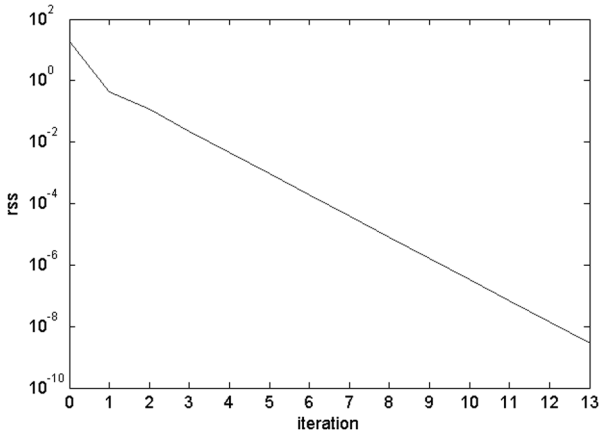

Figure 5. The course of *rss* reduction when solving system (19) by QLS algorithm.

The equivalence of QLS and Gauss-Seidel method applied to normalized system has been proven algebraically and illustration, either graphical or tabular, cannot show anything but the same curves or numbers because both algorithms yield identical estimates of unknowns up to the numerical precision of the computer. Unfortunately, precise calculation in the next section reveals that even optimized QLS algorithm requires about three times more basic operations (additions, multiplications) than Gauss-Seidel method to solve system $\mathbf{Ax} = \mathbf{b}$, so there is not much reason to use it in practice. It is true that QLS can be directly applied to any system, while Gauss-Seidel method imposes certain constraints, but if we wish LS solution of a system, there are other methods (like QR or SVD, see [5],[10]) at disposal and they will likely be faster than QLS when applied to full systems. With sparse systems QLS algorithm might be competitive, but this strongly depends on implementation and is subject to additional examination out of the scope of this paper.

5. Complexity Analysis

Because all presented algorithms are iterative routines for which required number of iterations is not known in advance, the appropriate measure of complexity is the number of operations in a single iteration.

Obviously, the complexity of the basic QLS variant (Algorithm 2) is n times the complexity of its computational engine QLCE and the number of operations in QLCE depends on the structure of equations and their terms. We know the largest number q of terms in a single equation in the system, but to facilitate the analysis and get the worse possible situation, we assume that all equations have q terms and that all terms contain all unknowns. Then, when we fix all except one variable and substitute their values in a term like (2), calculation of the resulting coefficient a_{rs} in (5) takes exactly $(n-1)-1$ multiplications (M) for multiplying variables plus one multiplication with coefficient, which yields the total of $n-1$ multiplications. Processing all q terms in an equation requires $q(n-1)$ multiplications and collecting the resulting coefficients according to (6) requires $q-1$ additions (A). Hence, processing a single equation is $q(n-1)\mathrm{M}+(q-1)\mathrm{A}$ operation and processing the whole system to complete step 2 of QLCE requires $m[q(n-1)\mathrm{M}+(q-1)\mathrm{A}]$ basic operations. Solving (8) in the third step of QLCE will take additional m multiplications and $m-1$ additions in both nominator and denominator, and finally one division, which we shall (for simplicity) account for as multiplication. Thus, solving (8) requires totally $2[m\mathrm{M}+(m-1)\mathrm{A}]+\mathrm{M}$ basic operations. All in all, one QLCE iteration takes

$$[mq(n-1)+2(m+1)] \text{ multiplications}$$

and

$$[m(q-1)+2(m-1)] \text{ additions.} \qquad (20)$$

In "big Oh" notation [2] this becomes $\mathrm{O}(mqn)$ multiplications and $\mathrm{O}(mq)$ additions, so it follows that QLCE has linear complexity with respect to all relevant parameters. Consequently, one iteration of QLS is

$$\begin{aligned} &\mathrm{O}(n[\mathrm{O}(mqn)\mathrm{M}+\mathrm{O}(mq)\mathrm{A}]) \\ &\quad = \mathrm{O}(mqn^2)\mathrm{M}+\mathrm{O}(mqn)\mathrm{A}. \end{aligned} \qquad (21)$$

For $m = n$ and $q \ll n$, this becomes

$$\mathrm{O}(n^3)\mathrm{M}+\mathrm{O}(n^2)\mathrm{A}. \qquad (22)$$

Thus, it can be declared that in common situations QLS is roughly $\mathrm{O}(n^3)$ algorithm.

Since linear systems are much simpler than quasi-linear, QLS solves them in an order of

magnitude smaller number of operations. This is easy to obtain by the analysis analogous to the preceding one. The only difference is that $q = n$ and there will be only one multiplication per term during construction of system (5). However, with linear systems we should use the optimized linear solver. Though its big Oh complexity is of the same order as that of general QLS when applied to linear systems, the asymptotic complexity is smaller and can be important for large size problems. Let us now calculate it precisely.

Algorithm 4 has preprocessing phase (steps 2 and 3) and iterative phase (steps 4 and 5). In step 2 all denominators in (8) are computed and we have already found that this will take $m\mathrm{M} + (m-1)\mathrm{A}$ per variable, that is, step 2 is $n[m\mathrm{M} + (m-1)\mathrm{A}]$. Initialization of recursion in step 3 requires $(n-1)n\mathrm{M}$ for $n-1$ products $\mathbf{a}_j x_j$, then $[(n-1)-1]\mathrm{A}$ for their summation and finally $n\mathrm{A}$ for subtraction of the sum from $\mathbf{b}$ or totally $(n-1)n\mathrm{M} + 2(n-1)\mathrm{A}$. On the whole, the preprocessing phase requires

$$\begin{aligned} &n[m\mathrm{M}+(m-1)\mathrm{A}]+(n-1)n\mathrm{M}+2(n-1)\mathrm{A} \\ &=(nm+n^2-n)\mathrm{M}+[(m-1)+2(n-1)]\mathrm{A}. \end{aligned} \quad (23)$$

Complexity of iterative phase depends on steps 4a and 4b. Clearly, recursion (18) in step 4a is $2n\mathrm{M} + 2n\mathrm{A}$. In step 4b we only have to compute nominator, which takes $m\mathrm{M}$ and $(m-1)\mathrm{A}$. Denominator is looked-up from the storage prepared in advance, so step 4b is completed by just one additional division (accounted for as multiplication). Thus, step 4b is $(m+1)\mathrm{M} + (m-1)\mathrm{A}$ and it follows that the total complexity of one pass through the for loop is $2n\mathrm{M} + 2n\mathrm{A} + (m+1)\mathrm{M} + (m-1)\mathrm{A}$ or $(2n+m+1)\mathrm{M} + (2n+m-1)\mathrm{A}$. Updating all n variables is, consequently,

$$(2n^2 + nm + n)\mathrm{M} + (2n^2 + nm - n)\mathrm{A}. \quad (24)$$

According to (23) and (24), the complexity of preprocessing and iterative phase is similar, determined by terms $(nm + n^2 - n)\mathrm{M}$ and $(2n^2+nm+n)\mathrm{M}$, respectively, but iterative term is a little larger. Of course, the more iterations, the more important iterative term so in practice it will be dominant. Therefore, in a common situation, when $m = n$, the complexity of linear solver is roughly $3n^2$ and it can be declared to be $\mathrm{O}(n^2)$ algorithm.

For comparison, the update of a single variable by Gauss-Seidel method, according to (13), is precisely $(n-1)\mathrm{M}+[(n-1)-1]\mathrm{A}+\mathrm{M}+2\mathrm{A} = n\mathrm{M} + n\mathrm{A}$, so the update of all n variables, i.e., one iteration, is $n^2\mathrm{M} + n^2\mathrm{A}$. Because multiplication is much more time consuming than addition, we can neglect addition term and it follows that complexity of Gauss-Seidel method is n^2 or about three times less than of the optimized linear solver.

6. Conclusion

Quasi-linear solver presented in the paper is computationally one of the simplest algorithms for solving systems of quasi-linear equations applying optimization approach but, at the same time, it shows to be one of the most robust and effective ones as well. Conceptually, the idea of the presented algorithm resembles coordinate descent method [9], but determining only one variable at a time is all that is common to these two algorithms. The main difference follows from the fact that QLS benefits from the special form of equations, which eventually leads to completely different computation. Comparisons show that it is as successful as Levenberg-Marquardt variant of Newton's method and they are both more successful than Nelder-Mead algorithm, when dealing with quasi-linear systems. However, QLSG has three major advantages over Newton's method when we deal with quasi-linear systems.

First, it is derivative-free algorithm and as such it is much simpler (though computing derivatives of quasi-linear equations is not problematic). As the consequence, its second major advantage is that it can work with systems that do not have equal number of equations and unknowns, which significantly widens its applicability compared to Newton's method. Finally, its third advantage is that its computational engine QLCE "sees" along the whole line of search direction whereby it can discover very distant points that reduce *rss*. This improves the algorithm's ability to "jump out" of local optimums and makes it less dependent on starting point. Moreover, QLCE computes new point by direct calculation, i.e., it does not need line-search, which greatly simplifies program

implementation and, together with simple and well defined computation of LS solution, reduces numerical requirements.

All these advantages have been gained at a relatively low cost that search directions are restricted to directions of coordinate axes and that we update only one dimension in a single iteration. Mutual effect of these restrictions is prolonged execution compared to LM, but there is a possible modification that might reduce the average required number of iterations. Namely, we can apply some kind of coordinate adaptation, like in adaptive coordinate descent method [9], and this is certainly one of the main directions for further development of the QLSG algorithm. However, the presented results show that, even in its simplest form, QLSG is robust and effective, which makes it a good choice for practical purposes.

References

[1] E. K. P. CHONG, S. H. ZAK, *An Introduction to Optimization*. Wiley, 3rd Edition, 2008.

[2] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, C. STEIN, *Introduction to Algorithms*. McGraw-Hill, 2nd edition, 2003.

[3] Y. DOĞAN, F. ÖRÜCÜ, A. KUT, V. RADEVSKI, Optimizing the Equation for a Dataset with Corresponding Attributes by Hybrid Genetic Algorithm. *Proceedings of the ITI 2011 33rd International Conference on Information Technology Interfaces*, (June 27–30, 2011), Cavtat, Croatia.

[4] N. R. DRAPER, H. SMITH, *Applied Regression Analysis*. 3rd Edition, John Wiley & Sons, 1998.

[5] G. H. GOLUB, C. F. VAN VAN LOAN, *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[6] N. HLUPIĆ, I. BEROŠ, D. BASCH, A Derivative-free Algorithm for Solving Quasi-linear Systems. *Proceedings of the ITI 2013 35th International Conference on Information Technology Interfaces*, (June 2013), Cavtat, Croatia.

[7] E. M. T. HENDRIX, B. G.-TOTH, *Introduction to Nonlinear and Global Optimization*. Springer Science + Business Media, LLC 2010.

[8] I. IVANŠIĆ, *Numerička matematika*. Element, Zagreb (in Croatian), 1998.

[9] I. LOSHCHILOV, M. SCHOENAUER, M. SEBAG, Adaptive Coordinate Descent. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation GECCO'11*, (July 12–16, 2011), Dublin, Ireland.

[10] C. D. MEYER, Matrix Analysis and Applied Linear Algebra, SIAM 2000

[11] I. MILLER, M. MILLER, *John E. Freund's Mathematical Statistics with Applications*. 7th ed., Prentice Hall, 2004.

[12] A. MORGAN, Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems, SIAM 2009

[13] J. NOCEDAL, S. J. WRIGHT, *Numerical Optimization*. 2nd edition, Springer Science and Business Media, 2006.

[14] G. A. F. SEBER, A. J. LEE, *Linear Regression Analysis*. 2nd edition, Wiley, 2003.

Received: July, 2013
Revised: September, 2013
Accepted: September, 2013

Contact addresses:

Nikica Hlupić
Department of Applied Computing
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3
Zagreb 10000
Croatia
e-mail: nikica.hlupic@fer.hr

Ivo Beroš
VERN' University of Applied Sciences
Trg bana J. Jelačića 3
Zagreb 10000
Croatia
e-mail: ivo.beros@vern.hr

Danko Basch
Department of Control and Computer Engineering
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3
Zagreb 10000
Croatia
e-mail: danko.basch@fer.hr

NIKICA HLUPIĆ received his Ph.D. degree in 2001 from the Faculty of Electrical Engineering and Computing, Zagreb, Croatia, where he is at present associate professor at the Department of Applied Computing. He lectures several undergraduate and graduate courses and his scientific interests include optimization, estimation theory, statistics and algorithm design.

IVO BEROŠ received his M.Sc. degree in 2001 from the Department of Mathematics, Faculty of Science, Zagreb, Croatia. At present, he is a senior lecturer at the VERN' University of Applied Sciences, Zagreb, Croatia. His scientific interests include numerical mathematics, optimization and statistics.

DANKO BASCH is a full time professor in the Department of Control and Computer Engineering at the Univeristy of Zagreb Faculty of Electrical Engineering and Computing. He received his PhD and MSc in computer science from the same faculty in 1991 and 2000 respectively. His research areas include modelling and simulation, modelling languages, and programming language design.