Bull. Nov. Comp. Center, Comp. Science, 32 (2011), 57–75
© 2011 NCC Publisher

Properties of nonlinear systems and convergence of the Newton-Raphson method in geometric constraint solving

S. Gatilov

Abstract. The paper describes an application of a variant of the Newton-Raphson method to solution of geometric constraint problems. Sparsity and rank deficiency of the corresponding nonlinear systems are emphasized and statistical data are presented. Several ways of handling underdeterminancy and overdeterminancy in solving the Newton linear systems are considered. The behavior of Newton's method is shown on some examples of nonlinear systems. Two algorithms for solving linear systems are proposed based on rank-revealing LU and QR factorizations. The paper is concluded with a numerical comparison of the proposed linear solvers.

1. Geometric constraint problem

A geometric constraint problem (GCP) consists of a finite set of objects and a finite set of constraints on them. Either two-dimensional or three-dimensional space is considered. Each object has a type, for example: point, line, circle, plane, sphere, cylinder, etc. An "engineering variable" is an important special type of objects that has no geometrical meaning. Each constraint is imposed on a subset of objects of the problem. Constraints also have certain types: incidence, tangency, distance, angle, etc. Each type can have a different meaning depending on the types of objects involved. An "Engineering equation" is a special type of constraints that allows us to impose an arbitrary algebraic equation as a constraint on any set of engineering variables.

A GCP is called underconstrained if it has continuum many solutions. If the problem is not underconstrained itself but becomes underconstrained after removing any single constraint, it is called well-constrained. In other cases, a GCP is considered overconstrained. It should be noted that the number of solutions is found up to the movement and rotation of the whole model.

CAD users are generally advised to make their geometric models well-constrained. However, it is often difficult to avoid overconstraining. Moreover, if a user wants to see the intermediate result of modelling, an underconstrained problem is likely to appear since not all the necessary constraints have been created by that time. Therefore, a geometric solver should not neglect the underconstrained and overconstrained problems.

To solve a GCP, the decomposition and simplification methods are normally used. A lot of approaches are based on the graph structure of a GCP [13], [14]. The benefits of using a decomposition for solving GCPs are shown in [8].

The direct way of solving a GCP involves solving a nonlinear system. The problem is reformulated as a system of nonlinear equations in the modelization process. Each object generates variables and perhaps additional equations on them, and each constraint turns into some equations and may also add some helper variables. For example, a 2D point is usually represented with its coordinates x and y, and a 2D line can be represented as three variables c, s, d and an equation $c^2 + s^2 = 1$. Incidence of a point and a line produces an equation $sx - cy = d$.

Geometric constraint solvers often utilize several modelizations to increase the success rate and to improve efficiency. Modelizations must be correct: given a solution of a nonlinear system, it should be possible to compute a GCP solution efficiently. At least one of the modelizations used should also be complete: if a GCP is solvable, then the generated nonlinear system should also be solvable.

The systems of nonlinear equations can be simplified and decomposed too. An example of a decomposition heuristic for nonlinear systems is the Dulmage-Mendesohn decomposition (see [1]). One way of simplifying a nonlinear system is substituting a variable with its direct expression obtained from some equation. Nonlinear systems can be solved either symbolically or numerically. Symbolic approaches work only with polynomial systems and require computation of the Groebner basis, so they are at least exponential in time [3]. Among numerical approaches, the Newton-like methods are highly attractive in practice despite some serious disadvantages.

The Ledas Geometric Solver (**LGS**) is a commercial solver for geometric constraint problems used in several CAD systems. It is available in both two-dimensional and three-dimensional variants. Both versions utilize the graph-based decomposition approaches to reduce a complex problem to a sequence of smaller ones [11]. The atomic problems that cannot be decomposed further are modelized and solved directly by the numerical solver. A variant of the Newton-Raphson method is the core of this solver. This paper summarizes the research work intended to analyze and improve the efficiency of the numerical solver component of **LGS**. The implementation of linear solvers described in Section 4 is used in the up-to-date version of **LGS**. The statistical data obtained and convergence analysis may be useful for further improvement of the numerical solver.

2. Nonlinear system

2.1. Definitions

A system of m nonlinear equations and n variables can be defined as $F(x) = 0$, where

$$F : \mathbb{R}^n \to \mathbb{R}^m, \tag{1}$$

$$x = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}^T, \tag{2}$$

$$F(x) = \begin{bmatrix} f_1(x) & f_2(x) & \dots & f_m(x) \end{bmatrix}^T. \tag{3}$$

A nonlinear (or linear) system is considered consistent if and only if there exists a solution that satisfies all the equations. A point $z \in \mathbb{R}^n$ is called a solution or a root of the nonlinear system whenever $F(z) = 0$, i.e. it satisfies all the equations. The set of all roots of the nonlinear system is denoted by Z.

We will assume that F is at least continuously differentiable everywhere. In practice most of the equations in F are polynomial. Non-polynomial (and non-analytical) equations are necessary to support correctly splines and user-defined curves and surfaces.

The Jacobian of the nonlinear system at a point p is an $m \times n$ matrix $J(p)$:

$$J(p) = \frac{dF}{dx}(p) = \left[\tfrac{\partial f_i}{\partial x_j}(p)\right]_{i=1\dots m, j=1\dots n} \in M_{m,n}(\mathbb{R}). \tag{4}$$

The Jacobian is used to define several properties of nonlinear systems like overdeterminancy and underdeterminancy.

A matrix is underdetermined if its columns are linearly dependent. Analogously, a matrix is overdetermined if its rows are linearly dependent. For a given linear system, the underdeterminancy and overdeterminancy properties are determined by examining its matrix. The same concepts are defined for a nonlinear system at a given point: a nonlinear system is underdetermined (overdetermined) at a given point if and only if the same holds true for the Jacobian of the system at this point.

A nonlinear system is said to have a constant rank point p if its Jacobian has a constant rank in the vicinity of p:

$$\exists r > 0 \quad \forall x \in B(p, r) \quad \operatorname{rank} J(x) = \operatorname{rank} J(p). \tag{5}$$

The set Z of all the roots has a simple structure near any constant rank root $z \in Z$: due to the constant rank theorem, Z is a differentiable manifold of dimension $n - r$ near z, where $r = \operatorname{rank} J(z)$.

2.2. Properties

If a GCP is underconstrained and modelization is complete, then the nonlinear system also has continuum many roots. Hence, some of them are not isolated, and the nonlinear system is underdetermined there. On the other hand, correct modelization of an overconstrained GCP is likely to generate a nonlinear system which is overdetermined at some points. Modelization of well-constrained problems may also lead to underdetermined and overdetermined systems. This happens because of the difficulties in expressing some of the constraints as nonlinear equations. Due to these reasons, a numerical solver should be able to handle underdetermined and overdetermined problems in the context of geometric constraint solving.

A nonlinear system is also likely to have a sparse Jacobian. It depends heavily on the modelization used. Here we consider only relatively simple modelizations which adhere to the following rules:

1. For each object $\mathcal{G}_i$ of GCP, a set $V(\mathcal{G}_i)$ of variables and a set $E(\mathcal{G}_i)$ of equations are generated. The equations in $E(\mathcal{G}_i)$ depend only on the variables from $V(\mathcal{G}_i) \cup V_0$.
2. For each constraint $\mathcal{C}_j$ of GCP, a set $V(\mathcal{C}_j)$ of variables and a set $E(\mathcal{C}_j)$ of equations are generated. Denote the set of objects involved in the constraint $\mathcal{C}_j$ as $\mathfrak{O}(\mathcal{C}_j)$. The equations in $E(\mathcal{C}_j)$ depend only on the variables from $\bigcup\limits_{H \in \mathfrak{O}(\mathcal{C}_j)} V(H) \;\cup\; V(\mathcal{C}_j) \cup V_0$.
3. All other variables and equations of the nonlinear system constitute the sets V_0 and E_0, respectively. These variables and equations are global and do not correspond to any object or constraint.

Now suppose that $|V_0|, |E_0|, |V(\mathcal{G}_i)|, |V(\mathcal{C}_j)|, |\mathfrak{O}(\mathcal{C}_j)| = O(1)$, i.e. these quantities are bounded by small constants. Then each equation depends on no more than $O(1)$ variables except for the equations in E_0. The number of equations in E_0 is small ($|E_0| = O(1)$). So the total number of equation-variable dependencies is $O(m+n)$, where n is the total number of variables and m is the total number of equations. It means that at any point p the Jacobian $J(p)$ has no more than $O(m+n)$ nonzero elements and is perfectly sparse. Note that the proposed assumption is often valid in practice: atomic geometric objects can be parametrized with a small number of variables (like two coordinates and a radius for a circle), the arity of constraints is small (tangency involves exactly two objects), and the number of global variables and equations is usually small and bounded (they can be used to establish global coordinates).

Statistical evidence of sparsity and rank deficiency is shown in Table 1. Data from the two-dimensional and three-dimensional versions of **LGS** have been collected. A large amount of GCP tests has been solved, and information about all the evaluated Jacobian matrices has been stored in the process. The Jacobians collected are further classified as small ($n_i + m_i \leqslant 40$), medium ($40 < n_i + m_i \leqslant 400$), and large ($400 < n_i + m_i$). The average quantities for all the Jacobians and for each of these groups are given separately.

Table 1. Statistical data for evaluated Jacobian matrices in nonlinear systems. Here $\sum X_i$ is the sum of X_i; $\overline{X_i}$ is the average of X_i; $\mathrm{nnz}(J_i)$ is the number of nonzero elements in Jacobian J_i; "FR" is the portion of Jacobians that have full rank, i.e. $\mathrm{rank}(J_i) = \min(m_i, n_i)$; "ND" is the portion of mumerically deficient Jacobians, i.e. $\mathrm{rank}(J_i) < \mathrm{sprank}(J_i)$, where sprank is the structural rank of the nonzero pattern, as defined in MATLAB

Version(group)	$\overline{m_i}$	$\overline{n_i}$	$\frac{\sum \mathrm{nnz}(J_i)}{\sum(n_i+m_i)}$	$\frac{\sum \mathrm{nnz}(J_i)}{\sum n_i m_i}$	FR	ND	$\frac{\sum \dim\ker J_i}{\sum n_i}$
3D(all)	10.06	12.10	1.86	0.030	0.7828	0.1323	0.266
3D(small)	4.49	6.35	1.30	0.306	0.8341	0.0792	0.361
3D(medium)	55.87	59.44	2.26	0.050	0.2584	0.6753	0.194
3D(large)	530.34	550.51	2.54	0.007	0.0053	0.9358	0.117
2D(all)	12.79	12.24	1.45	0.058	0.7835	0.1133	0.051
2D(small)	6.48	6.36	1.13	0.227	0.8075	0.0899	0.072
2D(medium)	51.77	48.51	1.70	0.050	0.6295	0.2646	0.034
2D(large)	327.79	315.80	1.79	0.008	0.4956	0.2855	0.046

The number of nonzero elements is less than $3(n+m)$ on average, even for large **LGS3D** tests. It confirms the hypothesis of a perfectly sparse Jacobian. The portion of full rank Jacobians is about 78% and goes down for larger sizes. The portion of numerically deficient matrices is about 12% and goes up for larger sizes. This data confirms that underdetermined and overdetermined systems often appear in practice.

3. Newton-Raphson method

Here we consider the classical Newton-Raphson method for solving nonlinear systems. It is an iterative method that calculates series of points (p_k) (approximate solutions). The first approximate solution p_0 in series can be chosen arbitrarily and is usually determined from the current state of the geometric model. To obtain the next solution, the system is linearized at the current point, and the produced linear system is solved:

$$J(p_k) \triangle p_k = -F(p_k), \tag{6}$$

$$p_{k+1} = p_k + \triangle p_k. \tag{7}$$

The matrix of the linear system is equal to the Jacobian matrix $J(p_k)$ at the current point p_k. If it is invertible, then the inductive relations can be rewritten as:

$$p_{k+1} = p_k - J^{-1}(p_k) F(p_k). \tag{8}$$

If the linear system is overdetermined, then it can be inconsistent. On the other hand, the underdetermined linear system can have multiple solutions.

This is why it is necessary to clarify the meaning of the linear system solution $\triangle p_k$ in (6) to define completely the Newton-Raphson iteration.

3.1. Underdetermined linear system

If a linear system is underdetermined and consistent, then the set of its solutions is a linear manifold of nonzero dimension. In this case, the solution with the minimal norm is a sensible choice for Newton's method. The Newton iteration with this choice is more likely to produce a solution of the nonlinear system which is close to the start point p_0. This choice is also beneficial for convergence of Newton's method.

Consider a very simple example (a circle equation):

$$x^2 + y^2 = 1. \tag{9}$$

The linear system in Newton's method is:

$$2x_k \triangle x_k + 2y_k \triangle y_k = 1 - x_k^2 - y_k^2. \tag{10}$$

Here $\triangle x_k$ and $\triangle y_k$ are the unknowns. Obviously, this linear system is always underdetermined.

First suppose that the minimum norm solution of the system is always chosen. Then in polar coordinates the Newton recurrent formula is:

$$\begin{cases} R_{k+1} = \frac{1+R_k^2}{2R_k}, \\ \varphi_{k+1} = \varphi_k. \end{cases} \tag{11}$$

The R_k series converges quadratically to 1 from any start point except $R_0 = 0$. Hence, the Newton iteration converges quadratically to the closest root from any start point except $p_0 = 0$.

On the other hand, the solution of the linear system can be chosen in a simple way. Suppose that the Gaussian elimination is used to solve the system. Then one variable (let it be $\triangle y_k$) is free and the other one ($\triangle x_k$) is dependent. The free variable can be set to any value. The most natural choice is zero. With this choice of the solution, the following recurrent relation is obtained:

$$\begin{cases} x_{k+1} = \frac{1-y_k^2+x_k^2}{2x_k}, \\ y_{k+1} = y_k. \end{cases} \tag{12}$$

If $|y_0| < 1$, then there is quadratic convergence to some root. It is obvious that, if $|y_0| > 1$, no solution can be found because y_k is constant and no solution with $|y| > 1$ exists. An important note: even the local convergence is missing for the roots $x = 0$ and $y = \pm 1$.

So even this simple example shows that choosing the minimum norm solution of a linear system is a good idea. A question yet unanswered here is what norm to use. It is better to use a norm induced by an inner product because there are efficient linear solvers for this case. The ordinary dot product is enough.

Note that the choice of the minimum norm solution is sensible to variable scaling. Generally, in geometric problems all the quantities are expressed in terms of some units of measurement. Each variable represents either some length, some angle or some derived quantity (like area, volume, etc). The dimensional analysis states that there is no sense in the standard dot product over variables if they have different units of measurement. It is necessary to determine the characteristic size of the problem for each of the independent units of measurement: length and, optionally, angle. The characteristic length can be chosen as the maximal absolute initial value of all the variables representing length. The characteristic angle can be set to 2π. Then all the variables can be replaced with their dimensionless equivalents, e.g. x replaced with $\hat{x} = x/L$. Such scaling improves convergence and leads to more natural solutions. Poor scaling can force a nonlinear solver to favor changing some quantities over the others in order to satisfy the constraints. For example, the solution may contain heavily rotated objects even if only a slight change is enough.

The ideas of this section have been tested on the 3D version of Ledas Geometric Solver. The test base of **LGS3D** consists of 21 243 geometric problems; some of them are inherently inconsistent. The nonlinear systems generated for these tests are in general highly underdetermined: the ratio of the Jacobian kernel dimension to the number of variables is about 0.27 on average. The nonlinear solver with the Gaussian elimination that sets free variables to zeros failed to solve 4 024 problems. After the linear solver had been modified to produce a solution with minimal L2-norm, it failed to solve 3 483 problems. Implementation of the appropriate variable scaling reduced the number of unsolved problems to 3 264. These improvements are considered substantial for this test base.

3.2. Overdetermined linear system

Consider one more example of a nonlinear system:

$$\begin{cases} x^2 + y^2 = 2, \\ x = 1, \\ y = 1. \end{cases} \tag{13}$$

Obviously it is always overdetermined and has a unique root $x = y = 1$. The linear system produced by Newton's method is:

$$\begin{bmatrix} 2x_k & 2y_k \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \triangle x_k \\ \triangle y_k \end{bmatrix} = \begin{bmatrix} 2 - x_k^2 - y_k^2 \\ 1 - x_k \\ 1 - y_k \end{bmatrix}. \tag{14}$$

Now we are going to analyze the consistency of this linear system. The values of both unknown variables can be uniquely determined from the last two equations. So the system is consistent if and only if these values satisfy the first equation. By substituting $\triangle x_k = 1 - x_k$, $\triangle y_k = 1 - y_k$ into the first equation, after some simplification we get:

$$(x_k - 1)^2 + (y_k - 1)^2 = 0. \tag{15}$$

So the linear system produced by Newton's method is consistent if and only if $x_k = y_k = 1$. Since this point is an exact solution of the nonlinear system, it is of no interest to us. At any other point, the linear system is inconsistent, although the nonlinear system is consistent.

So even a consistent overdetermined nonlinear system can produce inconsistent linear systems in the Newton-Raphson method. In order to solve such problems, the linear solver should be able to treat inconsistent systems too. It cannot produce a correct solution of the linear system, but it can give an approximate solution which can be used as a step in Newton's method.

The solution of the linear least squares problem can be used:

$$\triangle p_k \in \operatorname*{argmin}_{d \in \mathbb{R}^n} \|J(p_k)d + F(p_k)\|. \tag{16}$$

The set of the linear least squares solutions is known to be equal to the set of solutions of the normal equations, so (16) is equivalent to:

$$J^T(p_k)J(p_k)\triangle p_k = -J^T(p_k)F(p_k). \tag{17}$$

This linear system is always consistent and gives one way to compute the desired solution. The Newton-Raphson method utilizing the least squares solution of a linear system is also known as the Gauss-Newton method.

There is also another favorable choice of the approximate solution. Choose any $r = \operatorname{rank} J(p_k)$ linearly independent rows of the Jacobian matrix $J(p_k)$. Solve r equations corresponding to these rows and simply ignore all the other equations. The chosen rows can be called base rows since they completely determine the solution of the linear system. Such a solution can be easily obtained from the Gaussian elimination. If row pivoting is used, then the set of satisfied equations may differ from iteration to iteration.

There is intuition behind this choice. First of all, satisfying the base r linear equations is equivalent to performing a Newton step for the reduced nonlinear system that consists only of the equations corresponding to the base rows:

$$\overline{F}(x) = (f_{b_1}(x), f_{b_2}(x), \dots, f_{b_r}(x)) = 0. \tag{18}$$

And second, as the current approximate solution p_k in Newton's method tends to the constant rank root p, the residues of the non-base $m - r$ linear equations tend to zeros (follows from the constant rank theorem). So the error of this approximate solution gets small in terms of residue.

For both choices of the approximate solution, Newton's method can converge to a non-root attractor. Consider the following overdetermined system ($a > 0$ is a constant scaling parameter):

$$\begin{cases} x^2 - 1 = 0, \\ a(x-1) = 0. \end{cases} \tag{19}$$

The Newton linear system is:

$$\begin{bmatrix} 2x_k \\ a \end{bmatrix} \triangle x_k = \begin{bmatrix} 1 - x_k^2 \\ a(1-x_k) \end{bmatrix}. \tag{20}$$

If the LU-based linear solver from Section 4.1 is used, then only one linear equation is satisfied at each iteration. Row pivoting ensures that the equation with the maximum absolute value of derivative is satisfied.

$$x_{k+1} = \begin{cases} \frac{1+x_k^2}{2x_k}, & \text{if } |2x_k| \geqslant a, \\ 1, & \text{else.} \end{cases} \tag{21}$$

The second branch immediately produces the root of the system. The first branch converges to -1 from $x_0 < 0$ and to 1 from $x_0 > 0$. If $a \leqslant \frac{1}{2}$ and $x_0 \leqslant -2a$, then Newton's method converges to -1, which is a root of the first equation but is not a root of the system.

Choosing the least squares solution for the linear system solves this problem but generates a new one. The recurrent expression is:

$$x_{k+1} = x_k + \frac{(x_k - 1)(2x_k^2 + 2x_k + a^2)}{4x_k^2 + a^2} = \frac{2x_k^3 + 2x_k + a^2}{4x_k^2 + a^2}. \tag{22}$$

The root $x_k = 1$ is always a fixed point of the iteration. However, two additional fixed points appear when $a < \frac{\sqrt{2}}{2}$. They correspond to the local minima of L2-norm $\|F(x)\|$ of the nonlinear system.

$$\begin{cases} x_1^* = \frac{-1+\sqrt{1-2a^2}}{2}, \\ x_2^* = \frac{-1-\sqrt{1-2a^2}}{2}. \end{cases} \tag{23}$$

It can be checked that $0 < \frac{dx_{k+1}}{dx_k}(x_2^*) < 1$ for $0 < a < \frac{\sqrt{2}}{2}$. Hence, x_2^* is an attracting fixed point for such values of the constant a. Newton's method locally converges to this point though it is not a root of the system.

4. Linear solvers

A linear solver solves the linearized system (6) at each iteration of the Newton-Raphson method:

$$Ax = b, \tag{24}$$

where $A \in M_{m,n}(\mathbb{R}), b \in \mathbb{R}^m$. In this paper only direct methods are considered. The choice of the linear solver algorithm is affected by the following facts:

1. The linear solver should be able to handle rank deficient matrices. It is important because the nonlinear systems in a GCP are often overdetermined, underdetermined, or both (noted in Section 2).

   This means that matrix factorization should have rank-revealing capabilities, which is a rather tough requirement for sparse factorizations.

2. In case of an underdetermined system, the linear solver should produce a solution with a small norm. The benefits of the minimum norm solution choice were discussed in Section 3.1.

   All the solvers described below produce minimum L2-norm solutions for consistent systems. Moreover, if the system is inconsistent, then the solution has the minimum norm among all the solutions with the same residue.

3. The linear solver should exploit sparsity of the Jacobian matrix for good performance.

   High-performance factorization packages designed specifically for dense matrices are not suitable for this problem. For instance, using **LAPACK** routines **DGETRF** and **DGEQP3** will most likely lead to wasting a lot of flops and time.

Note that it is better to avoid solving the normal equations (17). The number of nonzero elements in the normal matrix $J^T J$ is generally greater than in the original matrix J. Moreover, the condition number of the normal matrix is the square of the condition number of the original matrix. Therefore, it is harder to correctly detect the rank of the normal matrix, and performance may decrease. Despite these problems, some approaches have been proposed for solving the augmented equations [16].

In the algorithm descriptions below, the sign "=" is used to indicate a mathematical equality, and the sign ":=" is used for assignment and involves all the necessary computational steps.

4.1. Solver based on LU factorization

4.1.1. Algorithm

In this type of a linear solver, the rank-revealing LU factorization is applied to the Jacobian matrix:

$$L, P_r, P_c, U := \textbf{FactorizeLU}(A), \tag{25}$$

$$LP_rAP_c = U = \begin{bmatrix} G & H \\ 0 & S \end{bmatrix}, \tag{26}$$

where $P_c \in M_{n,n}(\mathbb{R})$ and $P_r \in M_{m,m}(\mathbb{R})$ are permutation matrices produced by pivoting, $L \in M_{m,m}(\mathbb{R})$ is a lower triangular matrix with units on its diagonal, $G \in M_{r,r}(\mathbb{R})$ is an upper triangular invertible matrix, $H \in M_{r,n-r}(\mathbb{R})$ is an arbitrary matrix, and $S \in M_{m-r,n-r}(\mathbb{R})$ is a matrix consisting of small values (usually $\max\limits_{i,j} |S_{i,j}| < \varepsilon$).

The matrix S is considered to be numerically zero, so r is the computed numerical rank of the matrix A. Now given the right side vector b of the system, we apply row transformations to it:

$$c := LP_rb. \tag{27}$$

Ignoring numerical errors, the original linear system (24) is equivalent to:

$$\begin{cases} Uy = c, \\ x = P_cy. \end{cases} \tag{28}$$

Now recall the structure of the matrix U. The last $m - r$ rows are considered to be numerically zero. If any of the last $m-r$ elements of the vector c are nonzero, then the system is inconsistent. To handle inconsistency, the values of the last $m - r$ elements of c are simply ignored:

$$\tilde{c} := (c_1, c_2, \ldots, c_r, 0, 0, \ldots, 0)^T. \tag{29}$$

After replacing c with $\tilde{c}$, the system (28) becomes consistent:

$$Uy = \tilde{c}. \tag{30}$$

And we continue to solve the corrected system instead of the original one:

$$Ax = \tilde{b} = {P_r}^{-1}L^{-1}\tilde{c}. \tag{31}$$

Assuming that $x = P_cy$, the set of solutions of (31) is equal to the set of solutions of (30) due to invertibility of LP_r. Since P_c is orthogonal, $\|x\| = \|y\|$. Hence, if the minimum norm solution y of (30) is found, then the corresponding x will be the minimum norm solution of the corrected system (31).

To obtain a solution of (30), we set the last $n - r$ variables to zeros and run triangular solve to compute the first r variables:

$$y^o := \left[G^{-1}\tilde{c}_{1..r} \quad 0\right]. \tag{32}$$

Then the basis of the kernel of U is computed. The kernel basis is stored in the matrix $K \in M_{n,n-r}(\mathbb{R})$. Each column of K contains a single base vector of the kernel space.

$$K := \begin{bmatrix} G^{-1}H \\ E \end{bmatrix}, \tag{33}$$

where E is the identity matrix. It is easy to validate that $UK = 0$ (if the S part is considered to be zero).

The kernel basis is orthonormalized with the modified Gram-Schmidt algorithm:

$$K^Q := \textbf{OrthonormalizeMGS}(K). \tag{34}$$

This kernel basis can be used to compute the orthogonal projection of the solution y^o on $\ker U$. The difference between y^o and its projection is the minimum norm solution of (30):

$$\alpha := (K^Q)^T y^o, \tag{35}$$

$$y^* := y^o - K^Q\alpha. \tag{36}$$

Obviously, y^* is a correct solution of (30) because it differs from the solution y^o by a vector from $\ker U$. It has minimum norm because it is orthogonal to $\ker U$.

Finally, the column permutation is applied to obtain the minimum norm solution of (31):

$$x^* := P_c y^*. \tag{37}$$

And this solution serves as an approximate solution of the original system (24) in Newton's method.

4.1.2. Discussion

The algorithm described above does not use the last $m-r$ rows of the matrix U and of the vector c at all. Recall that L is a lower-triangular invertible matrix. Then each i-th row of U is a linear combination of the first i rows of P_rAP_c and vice versa. The same is true for the vectors c and P_rb. Hence, the algorithm does not use the last $m-r$ rows of the matrix P_rAP_c and the vector P_rb.

Define the set of indices:

$$I = {P_r}^{-1}\{1, 2, \dots, r\}. \tag{38}$$

The rows with indices from I are the base rows. All the other rows of the vector b are not used in the algorithm at all. It means that the obtained solution satisfies the base r equations and ignores the rest $m - r$ equations. This is exactly the way of handling inconsistency that was discussed at the end of Section 3.2.

The most computationally expensive steps of the algorithm are LU factorization (25) and kernel orthonormalization (34). If the implementation does not use sparsity at all, then LU takes $\Theta(rmn)$ time and MGS takes $\Theta((n-r)^2 n)$ time. It is highly beneficial to exploit sparsity of the matrix A at least to some degree in LU factorization. Complete pivoting in LU should be avoided at all costs since it requires a full scan of all the remaining nonzero elements at each step of LU factorization. The simplest approach of exploiting sparsity is to avoid computations for the eliminations with zero coefficients in LU. It is generally enough to decrease dramatically the computational time for the problems of size about $1\,000 \times 1\,000$.

The MGS step takes $\Theta(k^2 n)$ time, where $k = n - r$ is the dimension of the kernel. If the degree of underdeterminancy of the Jacobian is very small, i.e. $\frac{k}{n} \ll 1$, then the MGS step takes a negligible time. Therefore, the LU-based solver fits the problems with low underdeterminancy. However, as k and n become comparable, the time taken by MGS starts to dominate totally.

It should be noted though that exploiting sparsity in MGS is not very helpful. In fact, MGS computes QR factorization of the matrix K^T. The Q-factor is calculated explicitly, and the R-factor is discarded in the process. The MGS algorithm is seldom used for sparse matrices in practice because the number of nonzero elements in Q-factor is usually large [5, section 5.6]. Besides, the K matrix itself can be denser than the original matrix A. Therefore, the usage of the QR-based linear solver can improve performance on the problems with a high underdeterminancy degree.

4.2. Solver based on QR factorization

4.2.1. Algorithm

To find the minimum norm solution, we start with rank-revealing QR factorization of the transposed matrix A^T:

$$Q, P_r, P_c, R := \mathbf{FactorizeQR}(A^T), \tag{39}$$

$$QP_r A^T P_c = R = \begin{bmatrix} G & H \\ 0 & S \end{bmatrix}, \tag{40}$$

where all the matrices are defined in a way similar to LU in (26), except for $Q \in M_{n,n}(\mathbb{R})$, which is an orthogonal matrix.

After transposing (40), the problem can be turned into the equivalent one:

$$R^T y = c, \tag{41}$$

$$y = QP_r x, \tag{42}$$

$$c := {P_c}^{-1} b. \tag{43}$$

Since Q and P_r are orthogonal, L2-norms of x and y are equal. Now let us examine system (41) closer. Recall that we assume S to be numerically zero. Then

$$\begin{bmatrix} G^T & 0 \\ H^T & S \approx 0 \end{bmatrix} y = c. \tag{44}$$

The values of the first r variables in this system are uniquely determined from the first r equations. The system does not depend on the last $n - r$ variables, so their values can be chosen arbitrarily. Obviously, setting them all to zeros gives the minimum norm solution of the system. Triangular solve is performed here:

$$y^* := \begin{bmatrix} (G^T)^{-1} c_{1..r} \\ 0 \end{bmatrix}. \tag{45}$$

System (44) is consistent if and only if the solution y^* satisfies the last $m - r$ equations. First, suppose that it is true. Then, the minimum norm solution of the original system (24) is computed as:

$$x^* := {P_r}^{-1} Q^{-1} y^*. \tag{46}$$

Otherwise, the system is inconsistent. In this case we try to obtain the minimum norm least squares solution. To achieve this, QR factorization of the matrix A can be used:

$$\hat{Q}, \hat{P}_r, \hat{P}_c, \hat{R} := \mathbf{FactorizeQR}(A), \tag{47}$$

$$\hat{Q}\hat{P}_r A \hat{P}_c = \hat{R} = \begin{bmatrix} \hat{G} & \hat{H} \\ 0 & \hat{S} \end{bmatrix}. \tag{48}$$

Then the right-side vector is corrected as follows:

$$\hat{c} := \hat{Q}\hat{P}_r b, \tag{49}$$

$$\tilde{c} := (\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_r, 0, 0, \ldots, 0), \tag{50}$$

$$\tilde{b} := \hat{P_r}^{-1}\hat{Q}^{-1}\tilde{c}. \tag{51}$$

The right-side vector $\tilde{c}$ lies in the image of the matrix $\hat{R}$ and is the closest one to $\hat{c}$ in the sense of L2-distance. Since $\hat{Q}$ and $\hat{P_r}$ are orthogonal, $\tilde{b}$ is also the closest right-side to b that gives a consistent system. This is why if the vector $\tilde{b}$ is used as the right-side instead of b in the remaining part of the algorithm, the x^* is the least squares minimum norm solution of system (24). So, after computing the corrected right-side $\tilde{b}$, go to step (43) and continue until the solution x^* is found.

4.2.2. Discussion

The QR-based linear solver computes the minimum norm least squares solution, which can be expressed in terms of the Moore-Penrose pseudoinverse matrix [15] as $x^* = A^\dagger b$. In this case the Newton-Raphson method turns into:

$$p_{k+1} = p_k - J^\dagger(p_k)F(p_k). \tag{52}$$

Convergence of the Newton-Raphson method defined as (52) has been thoroughly studied for the constant rank (5) case. Perhaps it was considered in [2] for the first time. Since then numerous local and semilocal convergence theorems have been given in [9], [17], [10]. Some of them provide quadratic convergence. Convergence to singular roots has also been studied in [4]. Since convergence to singular roots is generally only linear, several acceleration techniques have been proposed to restore superlinear convergence [7].

The most computationally expensive steps in the QR-based solver are obviously QR factorizations. QR factorization is more computationally expensive than LU factorization. It requires more computation per each nonzero element, and in the sparse case it leads to more fill-in [12]. The QR factorization of A can be skipped if the linear system is consistent. However, if the linear system is inconsistent, two factorizations are required.

The QR factorization exploits sparsity of the matrix A^T much better than the kernel orthonormalization does. Therefore, it is much more suitable for problems with a high degree of underdeterminancy. Overdetermined systems often produce inconsistent linear systems, and two factorizations are computed in this case.

It is possible to compute the least squares minimum norm solution with a single SVD factorization. The SVD factorization of the matrix A is defined as:

$$UAV = \operatorname{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k), \tag{53}$$

where $U \in M_{m,m}(\mathbb{R})$ and $V \in M_{n,n}(\mathbb{R})$ are orthonormal matrices, σ_i are nonnegative singular values in descending order and $k = \min(m, n)$. This factorization is much more computationally expensive than QR and has problems with exploiting sparsity.

5. Numerical results

We compare the linear solvers described above on large GCP tests from the **LGS3D** geometric solver test base. The geometric solver considered uses correct and complete modelization and solves the nonlinear system by the Newton-Raphson iteration. To improve global convergence, the line search is used in Newton's method. This is the only major difference from the classical Newton's method. The linear solvers behave precisely as described in Section 4.

The solvers named "LU" and "QR" use in-house LU and QR factorizations, respectively. Both factorizations are left-looking, use only row pivoting (no column pivoting). QR factorization is based on householder reflectors. The factorizations are semi-sparse: they store input/output matrices in the compressed sparse column format, but they do not try to predict any sparsity patterns during factorization. The factorization skips a column if all the potential pivots in it are less than the given threshold. This helps to determine approximately the rank of the matrix. Columns are grouped into blocks of size 16 to reduce sparsity overhead when applying transformations. Multi-threading and SSE2 are not used by the in-house factorizations. Since multicore workstations become increasingly popular, parallelization and vectorization are planned for future improvements.

The solver named "QRL" is a QR-based solver that uses **LAPACK** routines **DGEQP3** and **DORMQR** to perform factorization and multiplication on Q-factor. **DGEQP3** performs a dense householder QR factorization with column pivoting via **BLAS3** operations. The **AMD ACML** vendor-optimized implementation of **LAPACK** and **BLAS** is used. SSE2 is used in this library to improve performance. The rank is determined by comparing diagonal values of the R-factor with the same threshold as in in-house factorizations.

The hardware setup is Intel Core2 Quad Processor Q9300 (2.50 GHz, 6MB L2 cache). All the runs are performed in a single-threaded mode. Note that the "QRL" solver relies on **ACML** which has a parallelized version. So it would potentially run up to 4 times faster if launched in a multi-threaded mode on this hardware.

It should be emphasized that we do not intend to compare available linear solvers or factorization implementations here. The only purpose of including the "QRL" solver in the comparison is to show high benefits of exploiting sparsity for the linear systems considered. Fully dense **LAPACK** QR solver is significantly slower than its semi-sparse equivalent. Even if **ACML** utilized all 4 CPU cores, the performance gap would still remain.

Table 2. The testing results on **LGS3D** GCP tests. The overall time spent by the geometric solver (in seconds) and whether the test was solved is shown for each combination "test + linear solver". Note that solved tests generally consume significantly less time than the unsolved ones

Test	Upd		Arb		Var		Diam		Box	
$m \times n$	67	45	163	252	126	214	778	1 268	421	518
LU	0.112	N	0.487	Y	0.475	N	1.863	Y	3.901	N
QR	0.096	N	0.098	Y	0.109	Y	1.066	Y	1.502	Y
QRL	0.111	N	0.168	Y	0.113	Y	2.542	Y	2.329	Y

Table 3. Results of testing linear solvers on several linear systems. Here m, n are the sizes of a matrix, r is its approximate rank as determined by the "QRL" solver, NNZ is the number of nonzero values in it. slv is 'Y' for consistent systems. All the timings are measured in seconds

Test	m	n	r	NNZ	slv	LU	QR	QRL
Box1	421	518	397	3 785	N	0.036	0.076	0.274
Box2	923	921	881	4 663	N	0.074	0.337	2.244
Var1	472	505	444	1 972	Y	0.011	0.008	0.013
Var2	569	609	534	1 879	N	0.013	0.036	0.462
Diam1	484	320	9	1 903	N	0.058	0.005	0.065
Diam2	778	1 268	778	5 428	Y	0.790	0.073	1.165

The semi-sparse version also requires much less memory and can easily get into L2 cache unlike the dense solver.

It is not so evident which of the solvers is better: LU of QR. For the linear systems with a low degree of underdeterminancy (Box1, Box2, Var1, Var2), the LU-based solver is usually much faster. It confirms that LU factorization is generally faster than QR factorization, especially in sparse context. The usage of the second QR factorization in the case of an inconsistent system also significantly decreases performance of the QR-based solver. However, on tests with a high degree of underdeterminancy (Diam1, Diam2) LU-based solver spends almost all the time on the kernel orthonormalization by MGS, whereas the QR-based solver even benefits from the low rank of the system.

6. Conclusion

The nonlinear systems specific to geometric constraint solving are considered. The conducted analysis suggests that these systems are highly rank-deficient and sparse. Statistical data confirm these conclusions. Simple examples show that the usage of the minimum norm solution of linear systems is preferred in the underdetermined case. Handling inconsistent linear systems is clearly unavoidable for some overdetermined cases. Some possible ways of doing it are discussed.

Two linear solvers based on rank-revealing matrix factorizations are described. One of them is claimed inefficient for highly underdetermined cases because it utilizes kernel orthonormalization to minimize the solution norm. It is shown that fully dense factorizations are slow because the systems solved are very sparse. It is strongly advised to use factorization that exploits sparsity for better performance. Ideally, a high-performance sparse rank-revealing factorization should be used. For instance, **SuiteSparseQR** [6] can be used for the QR-based solver.

References

[1] Ait-aoudia S., Jegou R., Michelucci D. Reduction of constraint systems. – 1993.

[2] Ben-Israel A. A newton-raphson method for the solution of systems of equations // J. Math. Anal. Appl. – 1966. – Vol. 15. – P. 243–252.

[3] Buchanan S. A., de Pennington A. Constraint definition system: a computer-algebra based approach to solving geometric-constraint problems // Computer-Aided Design. – 1993. – Vol. 25, N 12. – P. 741 – 750. http://www.sciencedirect.com/science/article/pii/001044859390101S.

[4] Chen X., Nashed Z., Qi L. Convergence of newton's method for singular smooth and nonsmooth equations using adaptive outer inverses // SIAM J. on Optimization. – 1997. – February. – Vol. 7. – P. 445–462. http://dx.doi.org/10.1137/S1052623493246288.

[5] Davis T. A. Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2). – Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006.

[6] Davis T. A. Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization // ACM Transactions on Mathematical Software. – 2011. – Vol. 38, N 1.

[7] Decker D. W., Kelley C. T. Convergence acceleration for newton's method at singular points // SIAM Journal on Numerical Analysis. – 1982. – February. – Vol. 19, N 1. – P. 219–229.

[8] Decomposition plans for geometric constraint systems, part i: performance measures for cad // J. Symb. Comput. – 2001. – April. – Vol. 31. – P. 367–408. http://dl.acm.org/citation.cfm?id=374530.374535.

[9] Dedieu J.-P., Kim M.-H. Newton's method for analytic systems of equations with constant rank derivatives // J. Complex. – 2002. – March. – Vol. 18. – P. 187–209. http://portal.acm.org/citation.cfm?id=636282.636290.

[10] Dedieu J. P., Shub M. Newton's method for overdetermined systems of equations // Math. Comput. – 2000. – July. – Vol. 69. – P. 1099–1115. http://portal.acm.org/citation.cfm?id=349856.349871.

[11] Ershov A. G. Algorithms and software systems for parametric geometric modelling problems: Ph.D. thesis / A.P. Ershov Institute of Informatics System SBRAS. – 2007. http://www.iis.nsk.su/files/news/ershov.pdf.

[12] Gilbert J. R., Ng E. G., Ng G. Predicting structure in nonsymmetric sparse matrix factorizations // Graph Theory and Sparse Matrix Computation. – Springer-Verlag, 1992. – P. 107–139.

[13] Hoffmann C. M., Lomonosov A., Sitharam M. Planning geometric constraint decomposition via optimal graph transformations // Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance. – AGTIVE '99. – London, UK: Springer-Verlag, 2000. – P. 309–324. http://dl.acm.org/citation.cfm?id=646676.702135.

[14] Joan-Arinyo R., Tarrés-Puertas M., Vila-Marta S. Treedecomposition of geometric constraint graphs based on computing graph circuits // 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling. – SPM '09. – New York, NY, USA: ACM, 2009. – P. 113–122. http://doi.acm.org/10.1145/1629255.1629270.

[15] Penrose R. A generalized inverse for matrices // Proceedings of the Cambridge Philosophical Society 51. – 1955. – P. 406–413.

[16] Saunders M. A. Solution of sparse linear equations using cholesky factors of augmented systems: Tech. rep.: SOL, 1999.

[17] Xu X., Li C. Convergence of newton's method for systems of equations with constant rank derivatives // J. Comp. Math. – 2007. – Vol. 25. – P. 705–718.