ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
https://doi.org/10.17559/TV-20230518000644
Preliminary communication

A Comprehensive Evaluation of the DFP Method for Geometric Constraint Solving Algorithm Using PlaneGCS

Yunlei SUN, Yucong LI*, Kangping LIU

Abstract: The development of open-source geometric constraint solvers is a pressing research topic, as commercially available solvers may not meet the research requirements. In this paper, we examine the use of numerical methods in PlaneGCS, an open-source geometric constraint solver within the FreeCAD CAD software. Our study focuses on PlaneGCS's constraint solving algorithms and the three built-in single-subsystem solving methods: BFGS, LM, and Dogleg. Based on our research results, the DFP method was implemented in PlaneGCS and was successfully verified in FreeCAD. To evaluate the performance of the algorithms, we used the solving state of the constraint system as a test criterion, and analysed their solving time, adaptability, and number of iterations. Our results highlight the performance differences between the algorithms and provide empirical guidance for selection of constraint solving algorithms and research based on open-source geometric constraint solvers.

Keywords: algorithm testing; DFP method; numerical constraint solving algorithm; open-source geometric constraint solver

1 INTRODUCTION

Computer-aided design (CAD) is the most widely used modelling approach for engineering design. The typical starting point in these designs is 2D sketches which can later be extruded and combined to obtain complex three-dimensional assemblies [1]. The geometric constraint solver is a crucial component in this process and is often the key technology in parametric CAD design. However, as the engineering implementation of geometric constraint solvers is often a proprietary technology in commercial CAD, it can be difficult to understand the technical details. This has led to the need for the development of open-source geometric constraint solvers.

In this paper, we explore the application of numerical methods in open-source geometric constraint solvers using the FreeCAD CAD platform as an example. Samy Ait-Aoudia et al. [2] classified geometric constraint solving techniques into three categories: algebraic, rule-oriented, and graph construction. Among these, algebraic methods, including numerical and symbolic methods, are widely used due to their fast solving speed and applicability to almost all systems. Therefore, we focus on the application of numerical methods in open-source geometric constraint solvers.

However, despite some existing research focusing on geometric constraint solving, there is still relatively little research on the DFP method and geometric constraint solving algorithms based on PlaneGCS. The selection of geometric constraint solving algorithms still lacks criteria in engineering. And the experiments in these studies only simulate the work of geometric constraint solving algorithms in solvers by solving systems of equations, and are rarely actually applied to widely used solvers. Therefore, the objective of this study is to specifically investigate the application of the DFP method in PlaneGCS and comprehensively evaluate its performance using the real engineering cases as test cases. At the same time, the selection criteria of geometric constraint solving algorithm proposed by us are verified. Specifically, our research aims to answer the following questions:

- How is the DFP method implemented and verified in PlaneGCS?
- How does the geometric constraint solving algorithm based on the DFP method compare to other algorithms? How to choose an algorithm in engineering practice?
- What are the strengths and limitations of the DFP method for solving geometric constraints?

By explicitly stating these research objectives and questions, we can better guide the investigation of the DFP method and geometric constraint solving algorithms based on PlaneGCS and help to choose a more suitable algorithm, leading to a deeper understanding of the performance and applicability of open-source geometric constraint solvers.

We explore the application of numerical methods in open-source geometric constraint solvers using the FreeCAD CAD platform as an example. To investigate the application of the DFP method in PlaneGCS, we implemented and verified the DFP method in the PlaneGCS solver. We collected data through experiments to evaluate the performance of the geometric constraint solving algorithms, focusing on the solving state, solving time, adaptability, and number of iterations. The experimental data was obtained by testing the algorithms on a set of representative geometric constraint systems.

In the following sections, we present our research results and analysis.

Section 2 provides a literature review, summarising the existing research on geometric constraint solving methods and the solvers.

Section 3 describes the research methodology, including an overview of the PlaneGCS constraint solving algorithms and the implementation and verification of the DFP method.

Section 4 presents the results of our performance evaluation, including the experimental setup, data collection, and comparative analysis of solving state, solving time, adaptability, and number of iterations.

In Section 5, we discuss the implications of the results, interpreting and comparing the performance of the algorithms, highlighting their strengths and limitations.

Finally, in Section 6, we summarise our main research findings, answer the research objectives, discuss the contributions and significance of the study, acknowledge the limitations, and provide suggestions for future research.

2 RELATED WORKS

In many CAD modelling systems, 3D shapes are created by sweeping 2D profiles. In parametric systems, it is common to add geometric constraints and dimensions to the approximate profile so that the final accurate shape can be calculated [3]. This section provides an overview of geometric constraint solving methods and the solvers.

2.1 Overview of Geometric Constraint Solving Methods

Geometric constraint solving methods are techniques used to resolve a geometric constraint system, as shown in Fig. 1.

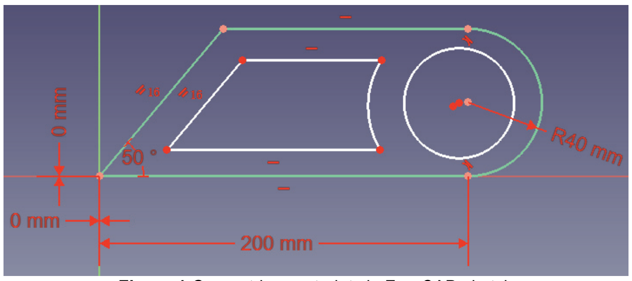


Figure 1 Geometric constraints in FreeCAD sketch

According to the classification proposed by Samy Ait-Aoudia et al. [2], these methods can be broadly categorized into three categories: algebraic methods, rule-oriented methods, and graph construction methods. Algebraic methods, in turn, encompass both numerical methods and symbolic methods, as illustrated in Fig. 2.

The numerical method transforms a geometrically constrained system into a set of nonlinear equations and solves it using a numerical algorithm. This approach was initially introduced by Hillyard [4] at the University of Cambridge and further developed and improved by Gossard [5] at MIT, known as Variational Geometry. Although these equations are often multi-solvable, numerical methods typically find only one solution. As a result, they are used as a last resort when all other methods have failed. Borning [6], Hillyard and Braid [7], and Sutherland [8], use a relaxation method. This method perturbs the values assigned to the variables and minimizes some measure of the global error. In general, convergence to a solution is slow. The method most widely used is the Newton-Raphson iteration. It is used in the solvers described in Refs. [9-11]. Newton-Raphson is a local method and converges much faster than relaxation. The method does not apply to consistently over-constrained systems of equations unless special steps are taken, such as solving a least squares problem.

The symbolic method, like the numerical method, also transforms the geometric constraint system into a set of nonlinear equations. However, it employs symbolic algebra to solve these constraints, using methods like the Grobner basis method [12] or Wu-Ritt characteristic column method [13]. If symbolic parameters are used in the system of nonlinear equations, the symbolic method can find a general solution to the geometric constraint system, making it a highly effective approach. However, this method has the disadvantage of being slower, with higher time and space complexity, so there are restrictions on the types of geometric elements and constraints that can be used.

The rule-oriented geometric constraint solving method utilizes rules to define and carry out the construction process, hence the name "rule-construction solving method". This approach allows for clear representation of geometric knowledge, separating it from the processing stage, and makes it easy to expand the rule base. However, this method also has its drawbacks: the rules are often incomplete, the system is bulky, the solving speed is slow, and it is unable to solve cyclic constraints.

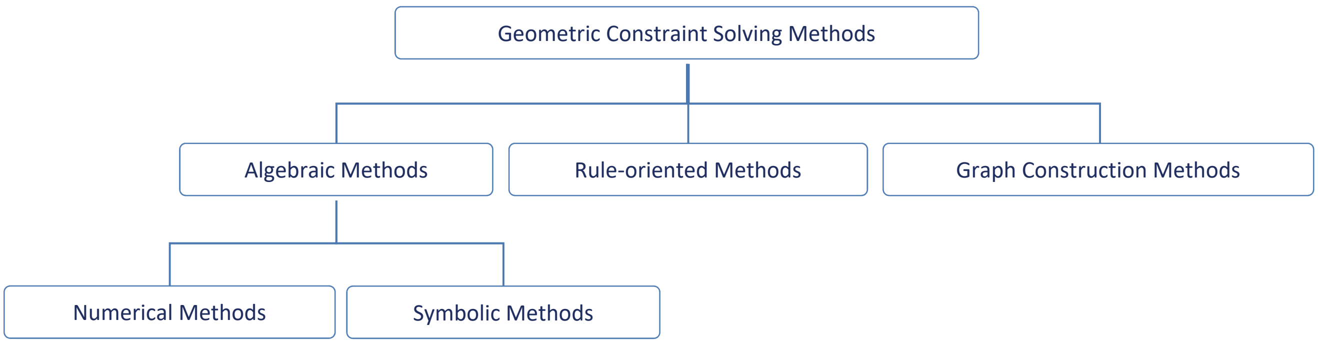


Figure 2 Classification of geometric constraint solving methods

The graph construction method transforms the geometric constraint system into a graph, deduces the construction process through analysis of the geometric constraint graph, and generates the geometric graphics based on the construction steps. Currently, some researchers use machine learning training datasets (such as SketchGraphs [14]) and frameworks (such as SketchGen [1]) to train models to automatically generate sketches, reducing design time and enabling new design workflows. This method is based on graph theory and is theoretically rigorous, fast, and efficient. However, it can only solve closed-loop constraints through numerical methods. Additionally, it is sensitive to the types of geometric elements and constraints used, so modifications to the solving algorithm are required when adding new geometric elements or constraints, making this method less general.

The summary of geometric constraint solving methods is presented in Tab. 1. It is evident that the numerical method is widely used in practical geometric constraint solving due to its general applicability, and the focus of the research in this field is to improve its solving efficiency and stability.

Table 1 Summary of geometric constraint solving methods

Methods		Advantages	Disadvantages	Applicable scene
Algebraic	Numerical	Fast, general	Sensitive to initial value, only one solution	Most systems
	Symbolic	Effective, generic solution	High time and space complexity	Small systems with restricted elements and constraints
Rule-oriented		Separate knowledge and processing, avoid numerical instability	Slow, difficult implementation	Small systems
Graph Construction		Rigorous theory, fast solving	Sensitive to types of elements and constraints	Systems without new types of elements and constraints

2.2 Limitations of Commercial Solvers

Commercial geometric constraint solvers have played a prominent role in the field of computer-aided design (CAD). Tab. 2 shows some famous commercial geometric constraint solvers:

Table 2 Some famous commercial geometric constraint solvers

Name	Developer	Applied to CAD
DCM (Dimensional Constraint Manager) [15]	D-Cubed (a subsidiary of Siemens PLM Software)	AutoCAD, SolidWorks, Creo, NX, Solid Edge etc.
LGS (LEDAS Geometric Solver) [16]	LEDAS (currently owned by Bricsys)	Cimatron E, BricsCAD etc.
DCS (Dimensional Constraint Solver) [17]	Huatian Software	CROWNCAD etc.

However, they also possess certain limitations that have prompted the exploration and development of open-source alternatives.

Proprietary Nature: Commercial solvers are often proprietary technologies, meaning that their underlying algorithms and implementation details are not openly accessible to researchers and developers. This lack of transparency hinders a comprehensive understanding of the solver's inner workings, limiting customization and advanced research efforts.

Limited Extensibility: Commercial solvers may have limited extensibility options, making it challenging for researchers and developers to integrate new algorithms or modify existing functionalities. The closed nature of these solvers restricts the flexibility required to adapt them to specific research needs or unique application domains.

Cost and Licensing: Commercial solvers typically require a substantial financial investment due to licensing fees and maintenance costs. This cost factor can be prohibitive for individual researchers, small research groups, or organizations with limited budgets, limiting their access to advanced geometric constraint solving capabilities.

Lack of Community Collaboration: The closed-source nature of commercial solvers often hampers collaboration and knowledge sharing among researchers and developers. This limitation prevents the community from collectively addressing challenges, improving algorithms, and contributing to the solver's overall development.

Dependency on Vendor Support: Commercial solvers rely heavily on vendor support for updates, bug fixes, and technical assistance. The dependency on a single vendor for ongoing maintenance and support may lead to delays in obtaining critical bug fixes or feature enhancements, potentially affecting productivity and research progress.

It is important to note that while commercial solvers have these limitations, they have also made significant contributions to the advancement of geometric constraint solving technology. However, the emergence of open-source solvers has provided an opportunity to address these limitations and foster a more collaborative and customizable environment for geometric constraint solving research and development.

2.3 Advantages of Open-Source Geometric Constraint Solvers

Open-source geometric constraint solvers have emerged as a significant alternative to proprietary solvers, offering several advantages in terms of development and functionality:

Accessibility: Open-source solvers are freely available, eliminating the need for costly licenses. This accessibility democratizes access to geometric constraint solving technology, making it available to a broader community of researchers, developers, and users. The open nature of the source code allows researchers and developers to scrutinize and understand the underlying algorithms and implementation details. This transparency facilitates code refinement, bug fixing, and customization according to specific requirements.

Extensibility: Open-source solvers can be easily extended to incorporate additional features and algorithms, enabling researchers and developers to continually enhance their capabilities. This extensibility promotes innovation and facilitates the integration of novel techniques into the solver framework.

Community Support: The open-source community surrounding these solvers often provides active support through forums, documentation, and collaborative problem-solving. This support network facilitates knowledge sharing and troubleshooting, fostering a vibrant and dynamic community.

Overall, the development and advantages of open-source geometric constraint solvers have revolutionized the field, enabling researchers, developers, and users to actively participate in the evolution and improvement of the solver technology. These solvers offer accessibility, extensibility, and a collaborative environment, empowering the CAD community to advance geometric constraint solving capabilities.

There are several common open-source geometric constraint solvers:

SolveSpace [18] is a software for 2D and 3D geometric modelling that allows the creation and modification of parametric or constraint-driven models, as well as export to a variety of formats. It uses a graph-based approach to solving geometric constraint systems.

PlaneGCS [19] is a geometric constraint solver for 2D sketching, which is mainly used in open-source software such as FreeCAD and SALOME Shaper. It supports many types of constraints such as distance, angle, parallel, perpendicular, tangent, etc. It uses numerical algorithms for solving systems of nonlinear equations such as BFGS.

GeoSolver [20] is a software for 3D geometric constraint solving that handles constraint relationships between basic elements such as points, lines, surfaces, and bodies, as well as user-defined variables and functions. It uses a method based on interval analysis to solve geometric constraint systems.

2.4 PlaneGCS and its Application in FreeCAD CAD Software

The PlaneGCS is the geometric constraint solver used in FreeCAD, as illustrated in Fig. 3. The solver consists of several key modules, including:

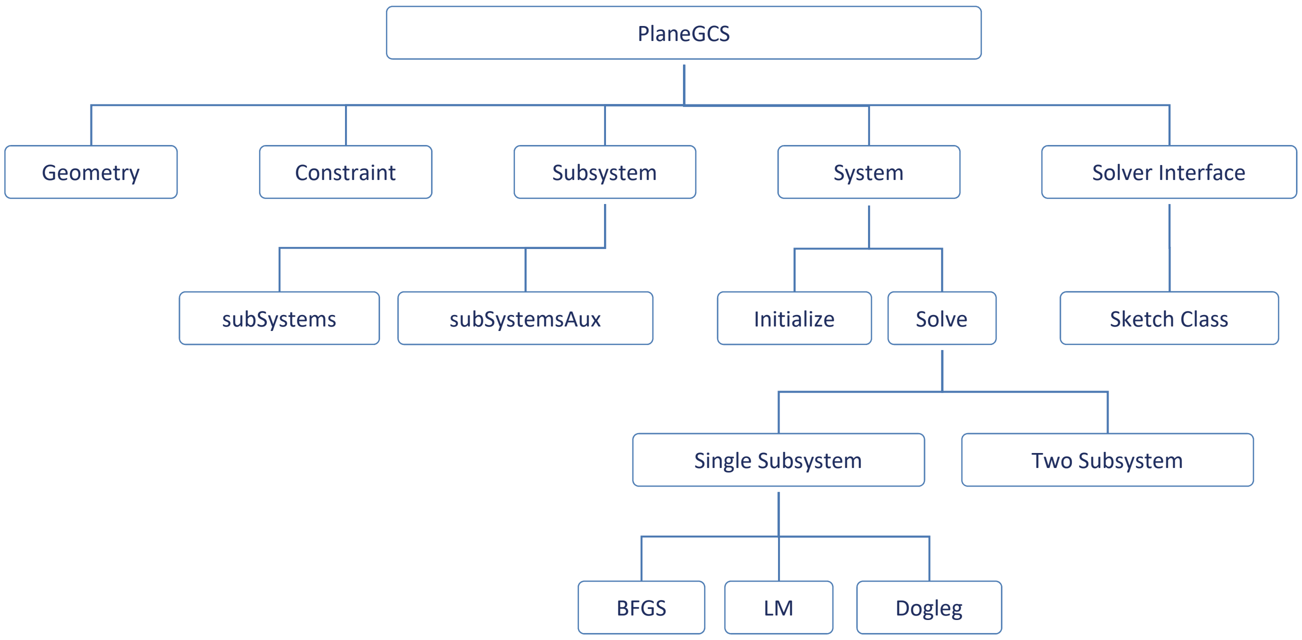


Figure 3 PlaneGCS frame

(1) Geometry Module: As defined in the *Geo.h* file, this module serves several key functions:

- It stores the parameters associated with the geometry
- Maintains solver-specific information about the geometry
- Assists in the construction of systems of parametric equations
- Facilitates the definition of complex geometries.

(2) Constraint Module: This module, defined in the *Constraints.h* file, performs a number of critical tasks:

- It stores the parameters of the constraint
- Determines the error caused by satisfying the constraint
- Computes the gradient with respect to a given parameter
- Limits the step size of the numerical algorithm
- Manages the priority of constraints
- Maintains the properties of the sketch workbench constraints.

(3) Subsystem. As defined in *SubSystem.h*, the subsystems are classified into two categories: primary subsystems and auxiliary subsystems. The primary subsystems arise from constraints with no priority, while the auxiliary subsystems arise from priority constraints. The priority is utilized in certain procedures that involve extended solution or programmatic movement of geometric shapes. The class *GCS::SubSystem* in *SubSystem.h* performs various computational operations on the parameters and constraints of the subsystem, such as calculation of residuals, the Jacobian matrix, or gradient. These calculations are used by the solver algorithms defined in *GCS:System*, such as the Dogleg algorithm.

(4) System. The class *GCS::System* defined in *GCS.h* is the core of the solver. It contains all the parameters and constraints for components with decoupled parameters. This system is divided into subsystems, which are subsets of parameters and constraints that can be solved independently. The *GCS::System* has two primary functions: initializing the geometric constraint system and solving it.

The geometric constraint system initialization includes the following operations:

- Diagnosis: to detect redundant and conflicting constraints, calculate the degrees of freedom of the system of equations, and determine parameters without complete constraints.
- Dividing the system into decoupled components: using graphs to simplify the size of the problem.
- Reducing component parameters and constraints: the solver looks for equivalent constraints and simplifies the parameters and constraints accordingly.
- Organizing subsystems: based on the simplified parameters and constraints of the decoupled components, each component may create a *subSystem*, a *subSystemsAux*, or both.

The geometric constraint system solving includes single-subsystem solving and dual-subsystems solving:

- Single-subsystem solving uses subsystems without priority constraints and is used to perform redundancy solving during diagnosis. The most frequent solving operation, it depends on the actual implementation of the solving algorithm, such as BFGS, LM, or Dogleg.

- Dual-subsystem solving is the implementation of the SQP algorithm, which is the only algorithm that can solve two subsystems simultaneously. This algorithm gives priority to one subsystem over another and is used for programmatic movement operations, such as dragging.

(5) Solver Interface. Defined in *Sketch.h*, the solver interface is responsible for communicating and exchanging data between the sketch workbench and the solver. It also facilitates programmatic movement of geometric shapes. Importantly, the solving algorithm of the solver is called here. The calling logic of the algorithm is as follows: first, the default algorithm is attempted. If the default algorithm fails, the other algorithms are called in a specific order. In PlaneGCS, the default solving algorithm is the Dogleg method, and the algorithm call order is Dogleg, LM, and BFGS.

3 METHODOLOGY

3.1 Selection and Explanation of Performance Evaluation Metrics

In practical engineering applications, geometric constraint solvers lack criteria for selecting algorithms. We proposed the following four evaluation metrics to evaluate the performance of the DFP method and the other three algorithms in solving geometric constraint systems, providing an example to solve this problem

Solving Time: This metric measures the amount of time required by the DFP method to solve a given constraint system. It provides a direct indication of the efficiency of the algorithm.

Solving State: This metric reflects the adaptability of the DFP method, and it is the best indication of the strengths and weaknesses of the algorithm in practical use, because it reflects whether a typical test case has been solved successfully. PlaneGCS defines four solving states:

- Success indicates that the algorithm has found a solution and the error of the constraint system tends to 0.
- Converged means that the algorithm has found a solution that minimizes the error of the constraint system, but the error may not meet the required tolerance level, resulting in a failure.
- Failed state refers to the inability of the algorithm to find a solution.
- SuccessfulSolutionInvalid means that the algorithm has found a solution, but the solution is not accepted by OCE (Open Cascade Community Edition).

The solving state metric is a reflection of the adaptability of the DFP method, with Success > SuccessfulSolutionInvalid > Failed = Converged.

Number of Iterations: This metric measures the number of iterations required by the DFP method to solve a given constraint system. It reflects the convergence performance of the algorithm.

Systematic Error: This metric represents the difference between the results obtained by the DFP method and the expected results, and is a reflection of the accuracy of the algorithm.

It is important to note that in our theoretical analysis and discussion, we have set an acceptable error threshold of $1e^{-20}$ for the solution. This means that we will analyse the number of iterations required by the DFP method when the systematic error is small enough to provide insight into the fundamental behaviour of the algorithm.

3.2 Overview of Constraint Solving Algorithms in PlaneGCS

The focus of this paper is on the single-subsystem solving algorithm of PlaneGCS. A brief overview of the principles of these three preset algorithms is provided below, and the detailed steps of the algorithms implemented in PlaneGCS can be found in the appendix.

The BFGS method is an optimization algorithm that was independently introduced by Broyden [21], Fletcher [22], Goldfarb [23], and Shanno [24]. The formula for the BFGS method is given by Eq. (1), where B_k represents the inverse matrix of the approximate Hessian matrix H_k of the nonlinear systems of equations derived from the geometric constrained systems. The variables s_k and y_k in the equation are defined as $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$, respectively.

$$B_{k+1}^{BFGS} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} \tag{1}$$

The BFGS method is a quasi-Newton optimization method that uses the BFGS formula to correct the matrix. It is based on the assumption that both B_{k+1}^{BFGS} and B_k^{BFGS} are reversible. Using the Shermann-Morrison-Woodbury formula, a modified formula for H_k can be derived from Eq. (1), as shown in Eq. (2).

$$H_{k+1}^{BFGS} = H_k + \left(1 + \frac{y_k^T H_k y_k}{y_k^T s_k}\right) \frac{s_k s_k^T}{y_k^T s_k} - \left(\frac{s_k y_k^T H_k + H_k y_k s_k^T}{y_k^T s_k}\right) \tag{2}$$

The PlaneGCS algorithm uses the BFGS formula (Eq. (2)) to calculate the iteration direction, and then utilizes line search to determine the step size.

The Gauss-Newton method may encounter difficulties when the matrix $J_k^T J_k$ becomes singular. To resolve this issue, Levenberg [25] proposed using Eq. (3) to calculate the iteration direction, with $v_k > 0$. The Levenberg-Marquardt (LM) method, which has been widely adopted, was the result of the efforts of Marquardt [26] in 1964. The equation (Eq. (3)) is referred to as the LM equation, where J_k is the Jacobian matrix of the residual function in the system, I is the identity matrix, d is the iteration direction, and r_k is the residual.

$$\left(J_k^T J_k + v_k I\right) d = -J_k^T r_k \tag{3}$$

The correction of the parameter v_k is a crucial aspect of the LM method. PlaneGCS employs the method proposed by Nielsen [27]:

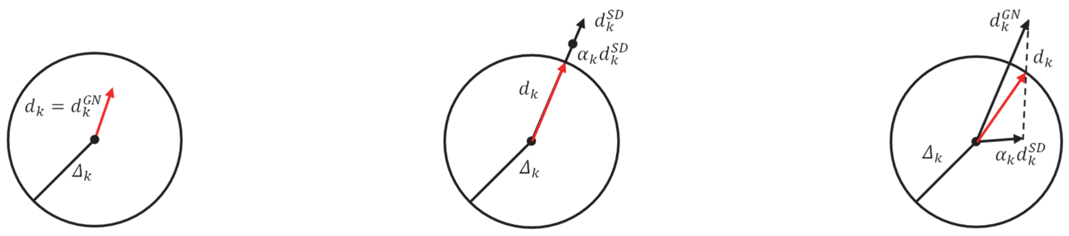
$$\begin{cases} \text{If } \gamma_k > 0,\ v_{k+1} = \max\left\{1/3, 1-(2\gamma_k - 1)^3\right\} v_k \\ \text{If } \gamma_k \le 0,\ v_{k+1} = c v_k, c := 2c \end{cases} \tag{4}$$

where, γ_k is defined as the ratio of the actual reduction of $f(x)$ from x_k to $x_k + d_k$, to the reduction of the quadratic approximation function $q_k(d) = \frac{1}{2}(J_k d + r_k)^T (J_k d + r_k)$ of $f(x_k + d)$.

The Dogleg method, proposed by Powell [28], solves the trust region subproblem Eq. (5) in 1970. It is inspired by the influence of parameter v_k in the LM method on the direction d_k^{LM}. In the Dogleg method, the trust region radius is denoted by Δ_k.

$$\begin{aligned} &\min \frac{1}{2} \| J_k d + r_k \|^2, \\ &\text{s.t. } \| d \| \le \Delta_k,\ \Delta_k > 0 \end{aligned} \tag{5}$$

The Dogleg method optimizes the iteration direction selection of the LM method. Its selection principle is illustrated in Fig. 4.



If$\| d_k^{GN} \| \le \Delta_k$,
make $d_k = d_k^{GN}$

Else, if$\alpha_k \| d_k^{SD} \| \ge \Delta_k$,
make $d_k = \dfrac{\Delta_k}{\| d_k^{SD} \|} d_k^{SD}$

Else，calculate
$d_k = (1-\beta)\alpha_k d_k^{SD} + \beta d_k^{GN}$,
where make sure β makes $d_k = \Delta_k$

Figure 4 Schematic diagram of the Dogleg method

3.3 Implementation and Verification of the DFP Method

The DFP formula, shown in Eq. (6), which was first proposed by Davidon [29] in 1959 and later developed by Fletcher and Powell [30], is the first quasi-Newton method that lays the foundation for the establishment of quasi-Newton methods. The DFP method is a quasi-Newton method that uses the DFP formula to correct the matrix. When compared to Eq. (1), it can be seen that B_k and H_k, y_k and s_k are swapped in the two equations, making BFGS method and DFP method dual to each other. This is also the reason why we choose the DFP method, which is similar in nature to the BFGS and is used to verify that our selection criteria are more representative.

$$H_{k+1}^{DFP} = H_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} \tag{6}$$

Assuming that both H_{k+1}^{DFP} and H_k^{DFP} are reversible, the correction formula for B_k can be derived from Eq. (6) using the Shermann-Morrison-Woodbury formula. Eq. (7) shows the correction formula, which can also be seen as a duality formula with respect to Eq. (2).

$$\begin{aligned} B_{k+1}^{DFP} = B_k + \left(1 + \frac{s_k^T B_k s_k}{s_k^T y_k}\right) \frac{y_k y_k^T}{s_k^T y_k} - \\ -\left(\frac{y_k s_k^T H_k + H_k s_k y_k^T}{s_k^T y_k}\right) \end{aligned} \tag{7}$$

The steps of the DFP method are as follows:

Step 1 Start with an initial point x_0 in R^n and a symmetric positive definite matrix H_0 in $R^{n \times n}$, set a tolerance value $\epsilon > 0$, and initialize the iteration counter $k = 0$.

Step 2 Check the termination criteria. If satisfied, output the relevant information and stop the iteration.

Step 3 Compute H_k using Eq. (6), and calculate the search direction d_k as $d_k = -H_k g_k$.

Step 4 Use a line search method to find a positive step size α_k, and update the current estimate $x_{k+1} = x_k + \alpha_k d_k$.

Step 5 Correct H_k using Eq. (6) to obtain H_{k+1}, increment the iteration counter $k = k + 1$, and go back to Step 2.

According to the steps of DFP method, we implemented the DFP algorithm in PlaneGCS and verified its effectiveness. We have added options for the DFP method to the solver control panel on the left side of the FreeCAD interface, as shown in Fig. 5.

Next, the time and space complexities of the DFP method are analyzed. We assume that the size of the geometric constraint system is n.

The time complexity can be analyzed as follows:

a. Finding the system error has a time complexity of $O(n)$.

b. The line search has a loop condition of $f1 > f2 < f3$, meaning that the number of iterations is constant with regards to the acceptable error. The function for finding err is called in each iteration, resulting in a time complexity of $O(n)$.

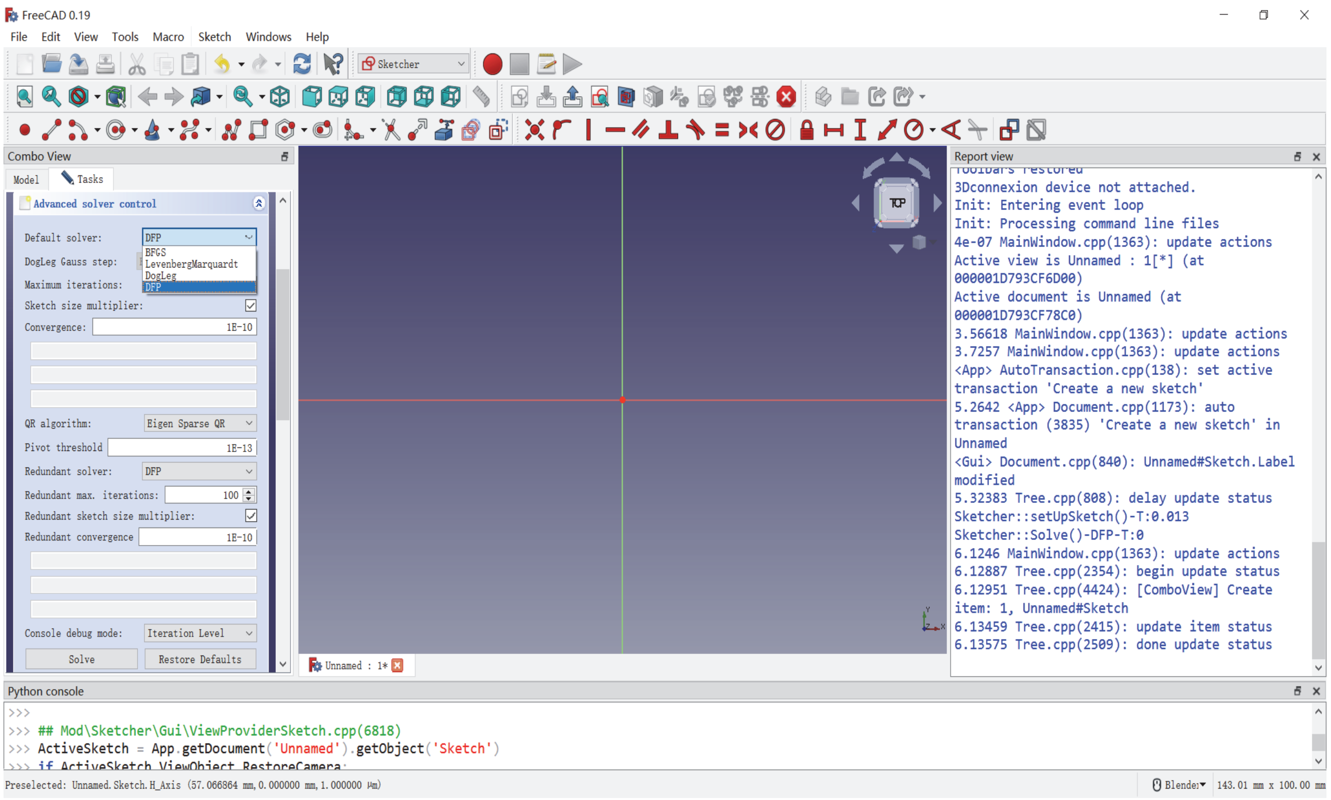
c. The termination condition of the DFP method is also tied to the error, and therefore, the number of iterations is constant. Among the internal functions called, finding error and the line search function both have a time complexity of $O(n)$, while the time complexity for the gradient calculation function is $O(n^2)$. As a result, the overall time complexity of the DFP method is $O(n^2)$.

The space complexity is determined by the data structures defined in the code for solving the problem, which are shown in Tab. 3.

The space complexity is calculated according to the space occupied by H, namely $O(n^2)$.

Table 3 The data structure used by the DFP method

Matrix	Vector					
H: approximate Hessian matrix	x: parameter vector	$xdir$: iteration direction	$grad$: gradient vector	s: $x_{k+1} - x_k$	y: $g_{k+1} - g_k$	Hy: $H * y$



Default solver: DFP
DogLeg Gauss step:
Maximum iterations:
Sketch size multiplier: ✓
Convergence: 1E-10
QR algorithm: Eigen Sparse QR
Pivot threshold 1E-13
Redundant solver: DFP
Redundant max. iterations: 100
Redundant sketch size multiplier: ✓
Redundant convergence 1E-10
Console debug mode: Iteration Level

```
3Dconnexion device not attached.
Init: Entering event loop
Init: Processing command line files
4e-07 MainWindow.cpp(1363): update actions
Active view is Unnamed : 1[*] (at
000001D793CF6D00)
Active document is Unnamed (at
000001D793CF78C0)
3.56618 MainWindow.cpp(1363): update actions
3.7257 MainWindow.cpp(1363): update actions
<App> AutoTransaction.cpp(138): set active
transaction 'Create a new sketch'
5.2642 <App> Document.cpp(1173): auto
transaction (3835) 'Create a new sketch' in
Unnamed
<Gui> Document.cpp(840): Unnamed#Sketch.Label
modified
5.32383 Tree.cpp(808): delay update status
Sketcher::setUpSketch()-T:0.013
Sketcher::Solve()-DFP-T:0
6.1246 MainWindow.cpp(1363): update actions
6.12887 Tree.cpp(2354): begin update status
6.12951 Tree.cpp(4424): [ComboView] Create
item: 1, Unnamed#Sketch
6.13459 Tree.cpp(2415): update item status
6.13575 Tree.cpp(2509): done update status
```

```
>>>
>>> ## Mod\Sketcher\Gui\ViewProviderSketch.cpp(6818)
>>> ActiveSketch = App.getDocument('Unnamed').getObject('Sketch')
>>> if ActiveSketch.ViewObject.RestoreCamera:
```

Preselected: Unnamed.Sketch.H_Axis (57.066864 mm, 0.000000 mm, 1.000000 μm)　143.01 mm x 100.00 mm

Figure 5 FreeCAD interface

4 EXPERIMENT

4.1 Experimental Setup and Data Collection

The tests were conducted using the test environment and tools listed in Tab. 4. The development environment used was FreeCAD 0.19.1, compiled using CMake 3.20.1 and Visual Studio 2019, and the operating system used was Windows 10.

The testing was performed using the Test Framework workbench of FreeCAD, as shown in Fig. 6. The testing module selected was TestSketcherApp, which is used to test the functions of the Sketch workbench. Clicking the Start button executes the setup test cases, which are examples of actual user use, and Fig. 7 shows one of the test cases ("BasicFillet"). The advantage of testing in this way is that it is closer to the actual engineering application scenarios than simulation in software such as Matlab by constructing a system of equations.

Table 4 Test/development environment

Source Code	Library Pack	Compilation Tool	Development Tool	Interface Development Plugin	Operating System
FreeCAD-0.19.1	FreeCADLibs_ 12.5.3_x64_VC17	CMake 3.20.1	Visual Studio 2019	Qt VS Tools 2.7.1	Windows 10

To collect the data to evaluate the performance of the algorithms, two methods were used. The first method involved changing the default solving algorithm in PlaneGCS, and the second method involved changing the order in which the algorithms are called in PlaneGCS. As an example, the DFP method was used to modify the default algorithm, and the order of the algorithms was changed to DFP, BFGS, LM, and Dogleg. The test information was recorded in the Report View on the right side of Fig. 5, which could be output as a log. However, the logs were long and cumbersome, so Python was used to extract and process the information.

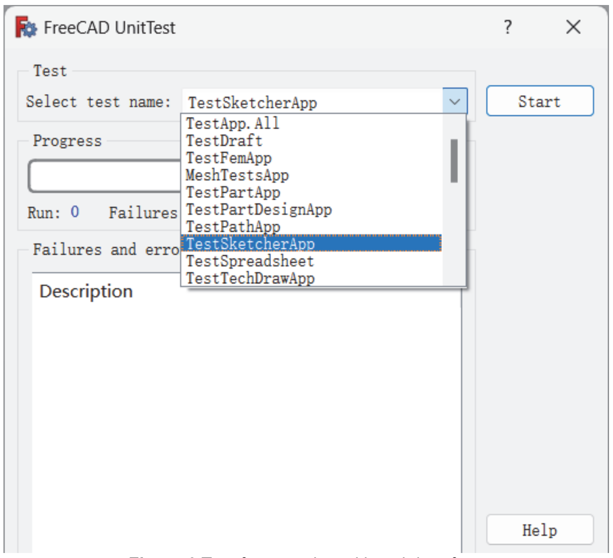


Figure 6 Test framework workbench interface

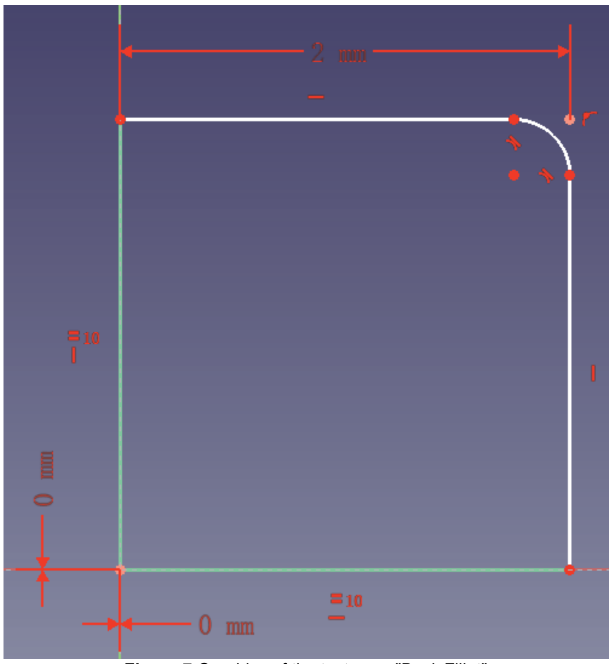


Figure 7 Graphics of the test case "BasicFillet"

4.2 Comparative Analysis of Algorithm Performance

Tab. 5 provides an overview of the average solving time of multiple experiments of each algorithm for each test case. As can be seen: the DFP solves the "BasicFillet" in the shortest time of the four algorithms, the BFGS has the shortest time for four test cases, the LM has two, and the Dogleg is the fastest for most of the test cases. It is important to note that the LM method failed to solve the test case "Curve" due to the results being rejected by the OCE (Open Cascade community version). The DFP and BFGS methods were unable to solve the "Sketchslot" test case as the number of iterations exceeded the pre-set maximum of 100. Further analysis of the "Sketchslot" test case will be discussed in a later section of this paper.

Table 5 The solving time of four algorithms for every test case (Unit: sec.)

Test Case	DFP	BFGS	LM	Dogleg
BasicFillet	**0.0064**	0.0074	0.0092	0.012
Coincident	0.006	**0.004**	0.0048	0.0046
Curve	0.0236	0.0176	0.025	**0.0174**
Distance	0.0082	0.0068	**0.0058**	**0.0058**
HorizontalVertical	0.0074	0.0052	**0.004**	0.0056
OriginalCorner	0.007	0.0064	0.0056	**0.004**
PointOnObject	0.0104	**0.008**	0.0096	0.0106
Symmetric	0.0176	0.0126	0.0118	**0.0114**
Tangent	0.0512	0.03	0.0218	**0.016**
Unconnected	0.0076	0.0034	0.003	**0.002**
UnconnectedCurve	0.0104	0.0052	0.0032	**0.0028**
BlockConstraintTests	0.065	0.045	0.0384	**0.0352**
SketchBox	0.0052	**0.0046**	0.0056	**0.0046**
Issue3245	0.0062	0.003	0.0032	**0.0028**
Issue3245_2	0.0008	**0.0002**	0.0004	0.0004
SketchSlot	1.1428	0.2144	0.177	**0.0632**

The data in Tab. 6 shows that the space complexity of LM and Dogleg methods is higher compared to the other two methods, DFP and BFGS. This may indicate that these two methods require more memory to solve the constraint system compared to the others. It should also be noted that the space complexity of an algorithm is an important factor in determining its performance and scalability, especially for large-scale geometric constraint solving problems. Thus, it is crucial to take this into consideration when choosing a suitable algorithm for a particular problem.

Table 6 The data structure stored by the algorithm

Data Structure	DFP	BFGS	LM	Dogleg
Matrix	1	1	2	2
Vector	6	6	7	8

The solving results for the 16 test cases using the four algorithms are displayed in Tab. 7, including the illegal (OCE-Invalid) state, referred to as the SuccessfulSolutionInvalid state in Section 4.1. The results show that these four algorithms have a range of applicability with Dogleg performing the best, followed by LM, BFGS, and DFP, which is in line with the default algorithm call order of PlaneGCS.

Table 7 The solving state of four algorithms for the 16 test cases

State	DFP	BFGS	LM	Dogleg
Success	15	15	15	16
OCE-Invalid (SuccessfulSolutionInvalid)	0	0	1	0
Failed	1	1	0	0

Table 8 The number of iterations and system error

DFP		BFGS		LM		Dogleg	
ite	err	ite	err	ite	err	ite	err
23	2.94E−22	14	8.58E−26	14	3.53E−19	4	1.07E−28
56	1.61E−23	23	1.83E−21	14	1.16E−20	7	2.88E−25
82	7.43E−28	8	5.10E−21	14	1.22E−19	5	1.07E−28
74	8.04E−22	27	4.76E−22	14	3.38E−19	6	1.24E−25
177	8.58E−22	45	9.03E−21	17	4.78E−19	7	2.96E−23
125	9.40E−23	22	1.19E−23	15	3.40E−18	7	1.03E−27
151	9.42E−21	27	2.18E−24	16	2.59E−20	6	2.10E−21
168	3.90E−22	28	1.05E−21	16	9.52E−18	7	2.50E−24
79	1.15E+00	168	1.21E−21	26	5.27E−18	10	7.39E−22
214	1.17E−24	24	1.08E−22	15	1.78E−20	6	7.36E−25

The results of the number of iterations and systematic error for the four algorithms used in PlaneGCS for the test case "Sketchslot" are presented in Tab. 8. The maximum number of iterations was increased during the test. As can be seen, the LM and Dogleg methods were more efficient than the quasi-Newton methods DFP and BFGS. Furthermore, the Dogleg method performed better than LM, and BFGS performed better than DFP.

4.3 Analysis of the Results

From the test data, it can be found that: the DFP method is the best at handling fillet operations, it has the shortest solving time, and when the sketch designed by the user contains a large number of fillets, DFP will significantly reduce the sketch solving time. Another noteworthy phenomenon is that the Dogleg method has the best performance among the 16 test cases, it has the shortest solving time in solving most of the test cases, and has fewer iterations than the other three algorithms, but it uses the most storage space, and may get bottlenecked when dealing with larger scale sketch solving problems. A good idea for solving geometric constraints is to have DFP, BFGS and LM handle the operations that they are good at, such as DFP for fillet, BFGS for coincident, and LM for distance, and Dogleg as an alternative, which is applied when the other methods do not work well to make the problem solvable by using its wide applicability.

5 DISCUSSION

The superiority of the Dogleg method over the quasi-Newton methods (DFP and BFGS) can be attributed to its trust region technique. Unlike the quasi-Newton methods, which determine the direction of iteration first and then the step size, the trust region method first sets limits on the step size and then determines both the direction and step size of iteration. This approach helps the iteration converge closer to the optimal solution, as it restricts the iteration from venturing too far in the wrong direction.

As stated by Nocedal [31], the BFGS method is known to have a highly effective self-correcting property, which allows it to correct misestimations of the curvature in the objective function within a few steps. On the other hand, the DFP method is less effective in correcting such misestimations, which is considered to be the reason for its poor practical performance.

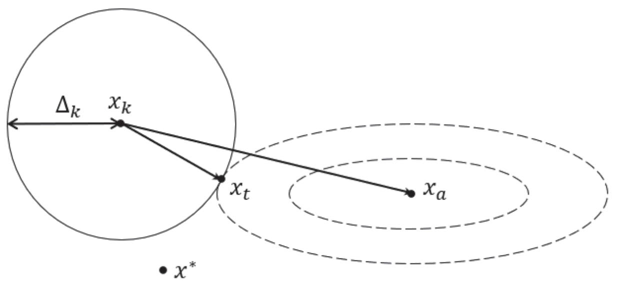


Figure 8 Comparison of results obtained by line search method and trust region method

The comparison between the Dogleg method and the two quasi-Newton methods is shown in Fig. 8, where x_k represents the current iteration point, x^* represents the optimal solution of the problem, and the dashed line represents the contour line of $q_k(d)$. The next iteration point obtained by the line search method (quasi-Newton) is $x_a = x_k + d_k$, while the next iteration point obtained by the trust region method (Dogleg) is x_t. It can be observed that x_t is closer to the minimum point x^* of the original problem than x_a, demonstrating the effectiveness of the trust region in directing the iteration towards the optimal solution.

We can see from the experimental results: the DFP method takes the shortest time among the four algorithms to solve the fillet problem, but it does not have an advantage in solving the other test cases, which also reflects that different algorithms are good at different areas in practical application scenarios.

The objectives of our study were to investigate and evaluate the application of the DFP methodology in PlaneGCS and to comprehensively evaluate its performance in solving test cases as an empirical guide for subsequent research outlines based on open-source geometric constraint solvers. And help to choose the appropriate geometric constraint solving algorithm in different engineering scenes. Based on the results of our study, we succeeded in achieving these objectives.

It is worth noting that although our findings are consistent with the purpose of the study, there are still some limitations. For example, our study was evaluated based on PlaneGCS only, and the application of other CAD platforms may be different. More algorithms need to be introduced to test the reliability of the selection criteria. In addition, our study could be further extended to explore the potential room for improvement and wider applicability of the DFP approach.

6 CONCLUSIONS

The conclusion of this paper highlights the study of the PlaneGCS geometric constraint solver of the open-source CAD software FreeCAD. The focus of the study was on the BFGS, LM, and Dogleg methods, with the addition of the DFP method. The research results were compared and analyzed in terms of solving time, adaptability, number of iterations, and systematic error. The study successfully explored the application and selection of numerical methods in an open-source geometric constraint solver and obtained the following main research findings:

- The DFP method is successfully integrated into PlaneGCS and its effectiveness in geometric constraint solving is verified.
- The DFP method shows faster solving speed in solving the fillet operation, which is advantageous compared with other algorithms.
- Different geometric constraint solving methods are good at handling different problems, which can refer to our standards for selection in the real engineering scene.

In this study, the way of applying DFP algorithm to PlaneGCS is put forward, which provides a new idea and method for improving the solution of geometric constraints in CAD field. By putting forward reasonable evaluation metrics and comparing DFP method with other algorithms, we provide an important reference for the selection of geometric constraint solving algorithms in CAD field. Our research results have important guiding significance for the further development and improvement of open-source geometric constraint solver in CAD field, and provide practical experience and suggestions for related work.

However, there are some limitations in this work that require improvement in the future. These include the need for more real engineering cases to test the adaptability of the solving algorithms and explore other sketch operations that DFP method is good at solving.

Acknowledgments

This work was supported in part by the Fundamental Research Funds for the Central Universities (Grant No. 18CX02019A).

7 REFERENCES

[1] Para, W., Bhat, S., Guerrero, P., Kelly, T., Mitra, N., Guibas, L. J., & Wonka, P. (2021). Sketchgen: Generating constrained cad sketches. *Advances in Neural Information Processing Systems*, *34,* 5077-5088.

[2] Ait-Aoudia, S., Bahriz, M., & Salhi, L. (2009, July). 2D geometric constraint solving: an overview. *2009 Second International Conference in Visualisation*, 201-206. https://doi.org/10.1109/VIZ.2009.29

[3] González-Lluch, C., Company, P., Contero, M., Pérez-López, D., & Camba, J. D. (2019). On the effects of the fix geometric constraint in 2D profiles on the reusability of parametric 3D CAD models. *International Journal of Technology and Design Education*, *29*(4), 821-841. https://doi.org/10.1007/s10798-018-9458-z

[4] Hillyard, R. C. & Braid, I. C. (1978). Analysis of dimensions and tolerances in computer-aided mechanical design. *Computer-Aided Design*, *10*(3), 161-166. https://doi.org/10.1016/0010-4485(78)90140-9

[5] Light, R. A. & Gossard, D. C. (1983). Variational geometry: a new method for modifying part geometry for finite element analysis. *Computers & Structures*, *17*(5-6), 903-909. https://doi.org/10.1016/0045-7949(83)90104-9

[6] Borning, A. (1981). The programming language aspects of thinglab, a constraint-oriented simulation laboratory. *Readings in Artificial Intelligence & Databases*, *3*(4), 353-387. https://doi.org/10.1145/357146.357147

[7] Hillyard, R. C. & Braid, I. C. (1978). Characterizing non-ideal shapes in terms of dimensions and tolerances. *ACM SIGGRAPH Computer Graphics*, *12*(3), 234-238. https://doi.org/10.1145/965139.807396

[8] Sutherland, I. E. (1963). Sketchpad: a man-machine graphical communication system. *Proceedings of the Spring Joint Computer Conference.* https://doi.org/10.1145/1461551.1461591

[9] Light, R. & Gossard, D. (1982). Modification of geometric models through variational geometry. *Computer-Aided Design*, *14*(4), 209-214. https://doi.org/10.1016/0010-4485(82)90292-5

[10] Lin, V. C., Gossard, D. C., & Light, R. A. (1981). Variational geometry in computer-aided design. *ACM*, *15*, 171-177. https://doi.org/10.1145/800224.806803

[11] Nelson, G. (1985). Juno, a constraint-based graphics system. *ACM*, 235-243. https://doi.org/10.1145/325334.325241

[12] Buchberger, B. (1985). *An Algorithmic Method in Polynomial Ideal Theory*. Reidel Publishing Co.

[13] Wu-Ritt (1984). *Basic principles of machine proof of geometric theorems*. Science Press.

[14] Seff, A., Ovadia, Y., Zhou, W., & Adams, R. P. (2020). Sketchgraphs: A large-scale dataset for modeling relational geometry in computer-aided design. arXiv preprint arXiv:2007.08506.

[15] Siemens PLM Software. (n.d.). D-Cubed 2D DCM. https://www.plm.automation.siemens.com/global/en/products/plm-components/2d-dcm.html

[16] LEDAS Ltd. (n.d.). LGS 2D/3D geometric constraint solvers. https://ledas.com/en/products/lgs-2d-3d/

[17] Huatian Software. (n.d.). DCS (Dimensional Constraint Solver) [Software]. Huatian Software. https://www.htcad.com/en/dcs.html

[18] SolveSpace. (n.d.). SolveSpace [Software]. SolveSpace. https://solvespace.com/index.pl

[19] PlaneGCS. (n.d.). PlaneGCS [Software]. PlaneGCS. https://github.com/PlaneGCS/PlaneGCS

[20] GeoSolver. (n.d.). GeoSolver [Software]. GeoSolver. https://geosolver.com/download.html

[21] Broyden, C. G. (1970). The convergence of a class of double-rank minimization algorithms: 2. the new algorithm. *IMA journal of applied mathematics*, *6*(3), 222-231. https://doi.org/10.1093/imamat/6.3.222

[22] Fletcher, R. (1970). A new approach to variable metric algorithms. *The computer journal*, *13*(3), 317-322. https://doi.org/10.1093/comjnl/13.3.317

[23] Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of computation*, *24*(109), 23-26. https://doi.org/10.1090/S0025-5718-1970-0258249-6

[24] Shanno, D. F. (1970). Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*, *24*(111), 647-656. https://doi.org/10.1090/S0025-5718-1970-0274029-X

[25] Levenberg, K. (1944). A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, *2*(2), 164-168. https://doi.org/10.1090/qam/10666

[26] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, *11*(2), 431-441. https://doi.org/10.1137/0111030

[27] Nielsen, H. B. (1999). Damping parameter in Marquardt's method. *IMM*, *248*.

[28] Powell, M. J. (1970). A hybrid method for nonlinear equations. Numerical methods for nonlinear algebraic equations, 87-161.

[29] Davidon, W. C. (1991). Variable metric method for minimization. *SIAM Journal on Optimization*, *1*(1), 1-17. https://doi.org/10.1137/0801001

[30] Fletcher, R., & Powell, M. J. (1963). A rapidly convergent descent method for minimization. *The computer journal*, *6*(2), 163-168. https://doi.org/10.1093/comjnl/6.2.163

[31] Wright, J. N. S. J. (2006). Numerical optimization.

Contact information:

Yunlei SUN, Associate Professor
Qingdao Institute of Software,
College of Computer Science and Technology,
China University of Petroleum (East China), Qingdao, 266580, China
E-mail: sunyunlei@upc.edu.cn

Yucong LI, Master
(Corresponding author)
Qingdao Institute of Software,
College of Computer Science and Technology,
China University of Petroleum (East China), Qingdao, 266580, China
E-mail: s21070036@s.upc.edu.cn

Kangping LIU, Master
Qingdao Institute of Software,
College of Computer Science and Technology,
China University of Petroleum (East China), Qingdao, 266580, China
E-mail: 2459625512@qq.com