Combining Uneliminated Algebraic Formulations With Sparse Linear Solvers to Increase the Speed and Accuracy of Homotopy Path Tracking for Kinematic Synthesis

Jeffrey Glabe
Department of Aerospace and Mechanical Engineering,
University of Notre Dame,
Notre Dame, IN 46556
e-mail: jeffrey.glabe@pnnl.gov

Mark Plecnik[1]
Department of Aerospace and Mechanical Engineering,
University of Notre Dame,
Notre Dame, IN 46556
e-mail: plecnikmark@nd.edu

The method of kinematic synthesis requires finding the solution set of a system of polynomials. Parameter homotopy continuation is used to solve these systems and requires repeatedly solving systems of linear equations. For kinematic synthesis, the associated linear systems become ill-conditioned, resulting in a marked decrease in the number of solutions found due to path tracking failures. This unavoidable ill-conditioning places a premium on accurate function and matrix evaluations. Traditionally, variables are eliminated to reduce the dimension of the problem. However, this greatly increases the computational cost of evaluating the resulting functions and matrices and introduces numerical instability. We propose avoiding the elimination of variables to reduce required computations, increasing the dimension of the linear systems, but resulting in matrices that are quite sparse. We then solve these systems with sparse solvers to save memory and increase speed. We found that this combination resulted in a speedup of up to 250× over traditional methods while maintaining the same accuracy. [DOI: 10.1115/1.4055241]

Keywords: homotopy continuation, sparse systems, predictor–corrector methods, kinematic synthesis, computational speed, computational accuracy

1 Introduction

Many problems arising in science and engineering require solving a system of polynomial equations. The equations relevant to this paper are complex and with many roots. To this end, there are many possible methods for finding these roots: Newton's method [1], interval analysis [2], and Gröbner bases [3] to name a few. However, the method of homotopy continuation [4] is often chosen for large polynomial systems due to its ability to be split into a series of independent subproblems. This makes homotopy continuation an attractive choice for parallel computing.

Homotopy continuation solves a polynomial system by first solving a similar polynomial system but with easy to find roots, then continuously deforms these start points into the roots of a target system. The deformation of each root can occur independently of every other root, leading to the method being termed "embarrassingly parallel" and affording itself attractively to multithreaded computing.

Kinematic synthesis of robotic devices and mechanisms requires finding the values of physical dimensions that satisfy a number of constraints specified on position, velocity, acceleration, or some combination of the three. Finding the dimension values (i.e., roots of polynomial systems that form constraints) is typically achieved through either optimization or solving the algebraic equations directly. Homotopy continuation is often applied in the exact synthesis of mechanisms. The synthesis equations associated with such problems grow exponentially as more sophisticated mechanical systems are designed [5].

Traditionally in kinematic synthesis, the rotation operators are algebraically eliminated to reduce the dimensionality of the problem. The result is fewer equations and variables, but with the tradeoff of increasing the complexity of the resulting equations. For this paper, we will refer to "uneliminated" as forgoing algebraic elimination of the original kinematic synthesis equations. We refer to "eliminated" as having performed the necessary algebraic elimination to reduce the number of equations and variables of the polynomial systems to be solved.

When computing all roots to a system of polynomials for the first time, the prior art has devised several sophisticated methods of constructing start systems and start points. Some of these include multihomogeneous homotopy [6], polyhedral homotopies [7], regeneration [8], statistics-based accumulations of roots [9], or monodromy methods [10]. However, once an initial minimal set of roots has been found, then the method of parameter homotopy continuation is known to be the most efficient [11]. This method first selects a parameterization of the polynomial variety of interest (which may be the monomial coefficient themselves, but generally is not), then forms a homotopy directly on those parameters. This is opposed to forming a homotopy on the constituent polynomials directly (which would be identical to selecting monomial coefficients as parameters). The efficiency of such an approach stems from the fact that it applies to homotopy paths corresponding with the minimal number of finite start points that, for *numerically general* [12] parameters, maintain their finiteness and multiplicity over the entirety of the homotopy. To start with such a minimal number of paths, parameter homotopy leverages solutions found from other forms of homotopy to simplify proceeding computations, constituting what is termed a numerical reduction of the polynomial system. Systems having the same structure with different

[1]Corresponding author.

Contributed by the Computers and Information Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received January 27, 2022; final manuscript received July 25, 2022; published online September 15, 2022. Assoc. Editor: Anurag Purwar.

parameters occur often in science and engineering, making parameter homotopy continuation suitable for many applications.

A common practical usage of homotopy continuation involves two phases of computation. In the first phase, an ab initio computation is performed using a generic set of parameters. This initial computation might be a multi-homogeneous homotopy, a polyhedral homotopy, regeneration homotopy, or a monodromy-based accumulation. This step has been generally understood as a resource-intense, one-time computation. In the second phase, the end points from an ab initio computation are repurposed as start points for repeated parameter homotopy computations. These parameter homotopies are repeated for different target parameters as necessarily required by some science or engineering application. These ensuing parameter homotopies are expected to compute quickly.

This expectation is true for systems that track from numerically general start parameters to numerically general final parameters. Numerically general refers to randomly generated complex numbers that have no special relation to one another. But science and engineering applications often require tracking to less general final parameters, stemming from physical problems that impart some pattern on these parameters. For these systems, using final parameters that have physical meaning leads to severe ill-conditioning that overwhelms the accuracy of path tracking when native precision [13]. Higher precision may overcome this, but at the cost of increasing computation time.

In homotopy continuation, ill-conditioning can occur when paths become sufficiently close in the tracking space when two or more points are merging to create a solution of higher multiplicity. Special techniques such as the gamma trick [14] have been developed to avoid ill-conditioning at midpoints during path tracking. Additionally, it can be shown with the Theorem of Sard [15] that careful choice of start parameters yields paths that have a very low chance of being ill-conditioned at a midpoint. Most of the ill-conditioning, however, occurs towards the end of the path, near the target system. Traditionally, special endgame methods [16] are used to successfully approximate these singular solutions and have been implemented in softwares such as HOMPACK90 [17], PHCPACK [18], POLSYS_PLP [19] and POLSYS_GLP [20]. The methodology presented in this paper applies to finding ill-conditioned, but nonetheless, nonsingular and isolated roots relevant to kinematic synthesis with 64-bit precision. This differs from the endgame techniques [16], which were developed to compute singular roots of high multiplicity.

Endgame methods function in an "operating zone" between a ball defined by the nearest branch point and a zone of ill-conditioning [21,22]. When the ill-conditioning zone expands beyond the nearest branch point, then the operating zone is nonexistent and the endgame will fail. Increasing working precision effectively decreases the size of ill-conditioning zone. However, using non-native precision (>64 bits on modern CPUs) leads to a huge decrease in floating point throughput, about $10\times$ slower. One of the contributions of this paper is to compute these endpoints in the absence of higher precision libraries.

1.1 The Use of Polynomial Homotopy Continuation in Kinematic Synthesis.

For kinematic synthesis, the number of paths required to track for problems affects the amount of ill-conditioning. For problems with fewer paths, the degree of ill-conditioning seems to be much less. Problems in kinematics such as those studied in Ref. [23] have been able to successfully track all paths. However, for problems such as those found in this paper, both the number of paths required to track and the nonlinearity of the equations increase, where this ill-conditioning reaches the point that practical usage for engineering design becomes intractable. The problem studied by Newkirk et al. required tracking 128 paths for a system of seven quadratic equations, whereas for the scale for one of the problems investigated in this paper, we require tracking 1,122,022 paths for a system of seven quartic and 14 quintic equations. High failure rates have additionally been previously observed in similar problems with a large number of paths in kinematic synthesis [10,24,25]. The correlation between the number of paths required to track and condition number is special to kinematic synthesis. The number of paths to track and conditioning are two independent properties. It is entirely possible to conceive of small, ill-conditioned problems as well as large, well-conditioned problems outside the realm of kinematic synthesis.

We make this point in the conceptual diagrams of Fig. 1: parameter homotopy continuation works well when working with problems with fewer paths, but when pulling from data in exercises taken from Sec. 4 of this paper, numerical failures increased by 26% and compute time increased by 63% when parameter homotopies were executed with physical parameters rather than numerically generic parameters. This effect greatly hinders the efficiency that parameter homotopy is expected to have.

To further prove this point, we performed a perturbation analysis study on the same problem. Starting with two sets: one generic set of parameters selected at random and one set of physical parameters. We found the solutions to these sets and then generated 10 sets of randomly perturbed parameters using a random noise of $\pm$ 0.001 selected from a uniform distribution, with each parameter being individually perturbed. As tracking all 1,122,022 solutions of the original sets would prove infeasable, a representative set of 1000 solutions were chosen at random and used for all runs in this analysis instead. Solutions were found with Bertini [22] in quad precision and using the power-series endgame method. The solutions of the perturbed parameters were then identified and compared element-wise against the unperturbed solutions and the average norm of the deviation between the two was calculated. This average deviation was taken over all complex solutions for both generic and physical parameters.

The result was that, for the generic set of parameters, an average deviation of 2.6 was calculated between the perturbed and unperturbed sets. To construct this for sets of physical parameters, an average deviation of 1.612×10^{11} was observed. It is clear that, for the same problem, changing from generic to physical parameters greatly increased the sensitivity even though the fundamental polynomial structure of the problem was unchanged.

The effect is pronounced enough that many would-be practical implementations of parameter homotopy are pushed out of reach based on the need to integrate non-native higher precision floating point operations (FLOPS). By our experience, the transition from 64-bit (double) to 128-bit (quad) arithmetic accounts for at least a $10\times$ increase in compute times. The contribution of this paper is to show how to surmount such barriers through the combination of a specific algebraic setup and sparse linear algebra routines. Our resulting algorithms compute with the speed of doubles, but the success of quads. Kinematic synthesis is one class of engineering problems that stand to benefit from this work.

2 Background

2.1 Parameter Homotopy Continuation. Consider a well-determined polynomial system $\mathbf{f}(\mathbf{z};\mathbf{q})$ where $\mathbf{z}$ is a vector of n variables and $\mathbf{q}$ is a vector of m parameters. We seek to find the solutions

$$\{\mathbf{z}^* \mid \mathbf{f}(\mathbf{z}^*;\mathbf{q})=\mathbf{0}\} \tag{1}$$

We define $\mathbf{q_0}$ as the set of start parameters, whose solutions have been already calculated in a previous "ab initio" step, and $\mathbf{q_1}$ as the set of target parameters for the system whose roots we wish to find. To do this, we must continuously deform each point $\mathbf{z_0}$ of the start system that satisfies $\mathbf{f}(\mathbf{z_0};\mathbf{q_0})=0$ into a point of the target system that satisfies $\mathbf{f}(\mathbf{z_1};\mathbf{q_1})=0$. To that end, we must construct the homotopy function:

$$\mathbf{f}(\mathbf{z};\mathbf{q}(t))=\mathbf{0} \tag{2}$$

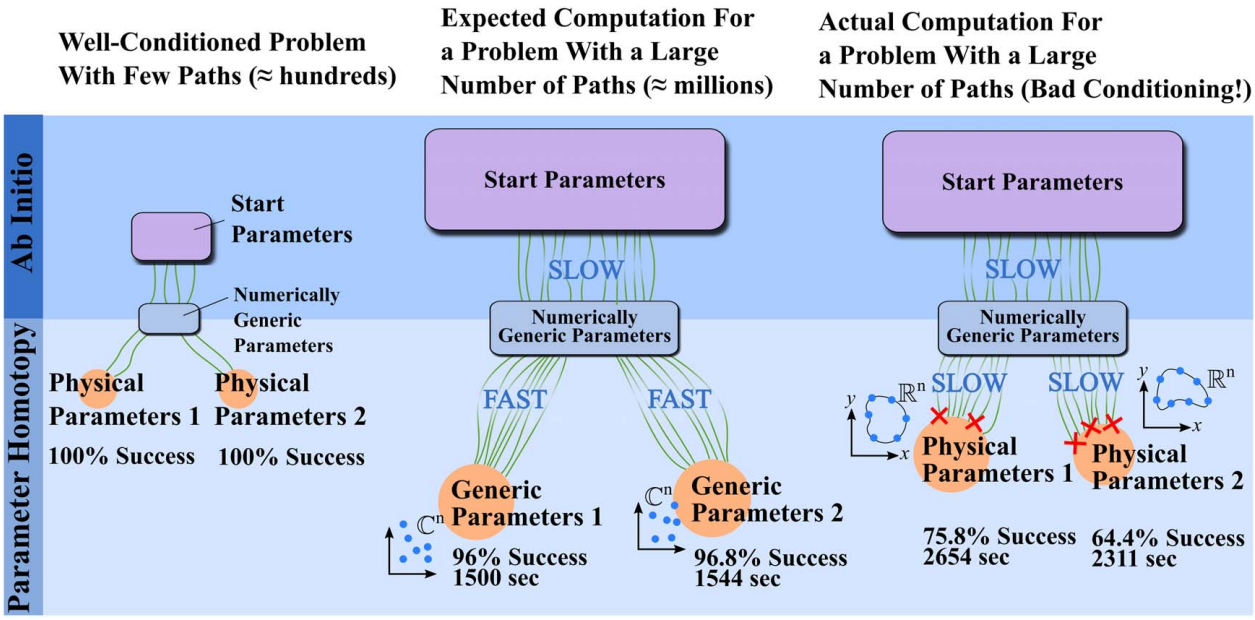

Fig. 1 Comparing the expected and actual performance of the parameter homotopy using generic parameters selected at random versus parameters chosen that have physical meaning to the designer. When using generic parameters, the process computes relatively quickly with a high success rate (number of solutions). To contrast, when using physical parameters, both the speed and accuracy of the parameter homotopy are impacted for problems with millions of paths, resulting in a marked drop in success rate and an increase in runtime. The numbers displayed in this figure were obtained from numerical exercises corresponding to the six-bar problem (Sec. 4) of this paper.

$$\text{where} \quad \mathbf{q}(t) = (1-t)\mathbf{q_0} + t\mathbf{q_1} \tag{3}$$

The tracking variable t is introduced to change from the start system at $t=0$ to the target system at $t=1$. For $0 \le t \le 1$, we have a linear combination of the parameters defining the two systems.

2.2 Path Tracking. To understand the process of tracking $\mathbf{z}$ from the start to the target system, we must derive the method of *path tracking* by understanding how a point $\mathbf{z}$ changes with respect to t. Taking the derivative of (2) with respect to t yields

$$\frac{\partial \mathbf{f}}{\partial \mathbf{z}}\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} + \frac{\partial \mathbf{f}}{\partial \mathbf{q}}\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = 0 \tag{4}$$

where $\frac{\partial \mathbf{f}}{\partial \mathbf{z}} \in \mathbb{C}^{n\times n}$, $\frac{\partial \mathbf{f}}{\partial \mathbf{q}} \in \mathbb{C}^{m\times n}$, and $\frac{\partial \mathbf{q}}{\partial t} \in \mathbb{C}^{m\times 1}$. For brevity, rename $\frac{\partial \mathbf{f}}{\partial \mathbf{z}} = \mathbf{J_v}$ and $\frac{\partial \mathbf{f}}{\partial \mathbf{q}} = \mathbf{J_p}$, that is the variable Jacobian and parameter Jacobian, respectively. The problem of path tracking therefore involves integrating the ordinary differential equation:

$$\mathbf{J_v}\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = -\mathbf{J_p}\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} \tag{5}$$

$$\text{where} \quad \frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = (\mathbf{q_1} - \mathbf{q_0}) \tag{6}$$

in conjunction with correction steps based on applying Newton's method to (2). The choice of integrator depends on the complexity of the polynomial system. For the polynomials described in this paper, a choice of Runge–Kutta–Fehlberg [26] and Dormand–Prince [27] fourth-order methods was used.

The process of path tracking involves iteratively predicting the next point $\mathbf{z}_{k+1}$ as t_k increments from 0 to 1. Path tracking is performed with a predictor–corrector technique. Nominally, at t_k, a Runge–Kutta iteration computes an estimate of $\mathbf{z}_{k+1}$ at t_{k+1}. This estimate is corrected by cycling it through a few iterations of Newton's method applied to (2).

Good prediction points are usually favored over increasing the number of correction steps due to path jumping. Path jumping occurs when a prediction point is placed in the zone of convergence of a different yet nearby path. The correction steps then cause the current path to converge to the solution curve of the nearby path, resulting in a repeated solution rather than two distinct solutions. Placing emphasis on good predictions helps mitigate this risk.

Predicted points that are not a good estimate of a root at t_{k+1} can be corrected by observing that the constraint (2) must be obeyed for all $t \in [0, 1]$. Therefore, it is possible to implement a root finding method to place the predicted point back onto the curve defined by (2). One popular choice is to iteratively execute Newton's method:

$$\mathbf{z}_{c+1} = \mathbf{z}_c - \mathbf{J_v}^{-1}\mathbf{f}(\mathbf{z}_c; \mathbf{q}(t_{k+1})) \tag{7}$$

until the predicted point has converged sufficiently to a solution of $\mathbf{f}(\mathbf{z}_c; \mathbf{q}(t_k))$. Various heuristic steps are taken to adaptively adjust step sizes and in case path tracking deviates from the nominal predictor–corrector process.

Path tracking is therefore the process of executing prediction and correction steps iteratively as t is perturbed from $t=0$ to $t=1$. At $t=1$, there are three possible outcomes for a path:

(1) The path becomes a distinct root $\mathbf{f}(\mathbf{z}^*; \boldsymbol{q}(1)) = 0$.
(2) Two or more paths converge to the same root to become a root of multiplicity > 1.
(3) Variable values tend toward infinity, termed points at infinity.

A path that does not reach one of these outcomes is considered to be a failed path. Paths fail for a variety of reasons. Relevant to this work, the most common reason is an inability for Newton's method to converge during a corrector step due to a shortage of floating point precision.

Most of computation spent during path tracking is spent solving linear systems of equations. Depending on the choice of integrator, (5) must be solved multiple times for each prediction step: six times for Runge–Kutta–Fehlberg and eight times for Dormand–Prince. Additionally, the correction equation (7) must be solved once for

every iteration. To give an estimate, a path tracker taking 200 prediction–correction steps using Dormand–Prince for the prediction with two iterations of Newton's method to correct must solve $200 \times (8+2) = 2000$ linear systems to track a path from t from 0 to 1. Solving these linear systems therefore is the heart of path tracking.

2.3 Path Failures and Numerical Precision. Paths whose outcomes do not fall under the conditions listed in the previous section can be considered as failures. Path failures can happen for a variety of reasons, but the most common and relevant reason is ill-conditioning of the matrices involved in solving the linear systems. In homotopy continuation, ill-conditioning occurs at t^* when $\mathbf{f}(\mathbf{z}; \mathbf{q}(t^*))$ approaches near singular conditions, even if an actual singularity does not occur. It is important to note that in most homotopy continuation settings, the choice of start parameters $\mathbf{q_0}$ is specially chosen such that there is a very low probability that the polynomial system truly has a root of multiplicity higher than one [12].

In the correction step, solving the linear system with a matrix of condition number C, it is generally expected that $\log_{10}(C)$ digits of precision are lost [13]. Numbers stored in double precision can expect 15–17 significant decimal digits [28]. When path tracking, ill-conditioned matrices often prevent a path from converging back to the curve (2) by restricting the number of digits that Newton's method is able to correct. The path is then not a good estimate of a root for the next prediction step. This causes the path to rapidly diverge from (2), preventing it from tracking successfully.

A natural remedy for this issue is to simply increase the working precision. Quad precision numbers can expect 33–36 significant decimal digits [28], with the possibility of extending that number further using multiprecision libraries. However, on the current hardware, special routines are required in order to perform arithmetic in precision higher than that of a double. As a general rule, using quad math results in a runtime approximately five times longer than double math [29].

As solving linear systems is a major computational cost of path tracking, it is important to observe the structure of the linear systems to be solved. A generic system of equations has a complexity of $\mathcal{O}(n^3)$, where n is the number of variables. Therefore, the number of variables of the system has a great impact on the time it takes to solve them. For the problem of kinematic synthesis, it is possible to influence the number of equations of the resulting system of equations.

2.4 Application: Kinematic Synthesis of Mechanisms. Kinematics is the geometry of motion. The goal of kinematic synthesis of linkages is to discover the dimensions, the lengths, and angles of a mechanism that produces a desired motion designated by a set of *tasks*. These tasks are defined as constraints on position, velocity, acceleration, or some combination of the three.

The dimensions of the mechanism are found by first defining the *loop equations*. These are a set of vector equations that define the geometry that must be conserved throughout the motion of the linkage. This geometry consists of a sequence of rotation and translation operations with sets of moving and fixed points. There are multiple ways to define these rotations and translations. One way is to use the addition of vectors to define translations, and elements of $SO(2)$ for rotations. Alternatively, Wampler [30] used isotropic coordinates which reduce the problem to the addition and multiplication of complex numbers, leading to a more direct formulation of polynomials. This paper uses isotropic coordinates to define all of the loop equations.

In essence, the isotropic coordinates $(z, \bar{z})$ of a planar point are related to the (x, y) coordinates of a planar point by the invertible linear transformation

$$\begin{Bmatrix} z \\ \bar{z} \end{Bmatrix} = \begin{bmatrix} 1 & i \\ 1 & -i \end{bmatrix} \begin{Bmatrix} x \\ y \end{Bmatrix}$$

In other words, a complex number and its conjugate are used in lieu of its real and imaginary components. The same goes for complex rotation operators, which are simply unit complex numbers, $e^{i\phi} = \cos\phi + i\sin\phi$. Defining $Q = e^{i\phi}$, see that Qz rotates z by ϕ about the origin of the complex plane. Note that Q is related to its complex conjugate through the algebraic equation $Q\bar{Q} = 1$, meanwhile, no such algebraic equation exists for z and z^*. Therefore, as a necessary condition for complex numbers to represent rotations, equations of the form $Q\bar{Q} = 1$, termed normalization conditions, are used in conjunction with isotropic coordinates.

Rotation operators can be eliminated by solving the loop equations for a rotation operator and its conjugate, then substituting the result into the normalization conditions. Rotation operators are usually eliminated whenever possible to reduce the number of variables, and thus the time it takes to solve the corresponding linear systems.

However, the resulting equations grow in complexity as operators are eliminated, rapidly increasing the cost of evaluating $\mathbf{f}(\mathbf{z}; \mathbf{q}(t))$, $\mathbf{J_v}$, and $\mathbf{J_p}$. Additionally, the exponential increase in operations introduces instability due to the intrinsic propagating error of floating point operations. Leaving the equations uneliminated reduces the number of floating point operations to evaluate Jacobians but increases the dimension of the linear systems that need to be solved (computation time scales by a cube of dimension for dense linear solving). For the problem of kinematic synthesis in particular, we know that leaving the equations in their uneliminated form means that $\mathbf{J_v}$ and $\mathbf{J_p}$ are particularly sparse matrices. Much work has been done in creating special routines for solving sparse linear systems that save memory and operations by working only with the nonzero elements of the matrix [31].

In this paper, we demonstrate how to overcome the numerical failures and slow compute times of parameter homotopies aimed at physical parameters. This is done by opting for equivalent formulations of polynomial systems that comprise of more equations and variables, and writing predictor–corrector tracking routines that implement sparse libraries. Such a setup carries more information from step to step across a homotopy through more variables rather than by representing few variables with more bits. Our technique appreciably reduces numerical failures without needing to resort to higher precision arithmetic. The latter point is important because the throughput of floating point operations on non-native (>64-bit) numbers is dramatically slower than native precision. The usage of more equations and variables naturally leads to sparse Jacobians, which we handle inside our path tracking algorithm with appropriate sparse routines. Else, our proposed benefit would be more than canceled by the cubic scaling of compute times to solve linear systems of increasing dimension.

2.5 Sparse Linear Solvers. Matrices with a large percentage of zeros occur in many areas of science and engineering. These include structural analysis, analysis of network and power distribution systems, and numerically solving differential equations [32]. Typically these matrices are very large as to justify special methods to deal only with nonzero elements.

Whether dense or sparse, a linear system of the form

$$\mathbf{Ax} = \mathbf{b} \tag{8}$$

is typically solved directly by decomposing $\mathbf{A}$ into the product of an upper $\mathbf{U}$ and lower $\mathbf{L}$ matrix:

$$\mathbf{PA} = \mathbf{LU} \tag{9}$$

where $\mathbf{P}$ is a permutation matrix that re-arranges the rows or columns of $\mathbf{A}$. This yields the decomposed linear system:

$$\mathbf{LUx} = \mathbf{Pb} \tag{10}$$

which can be solved consecutively as two triangular linear systems, $\mathbf{Ly} = \mathbf{Pb}$ and $\mathbf{Ux} = \boldsymbol{y}$. For dense solving, linear systems with triangular matrices are a special case which has a complexity of

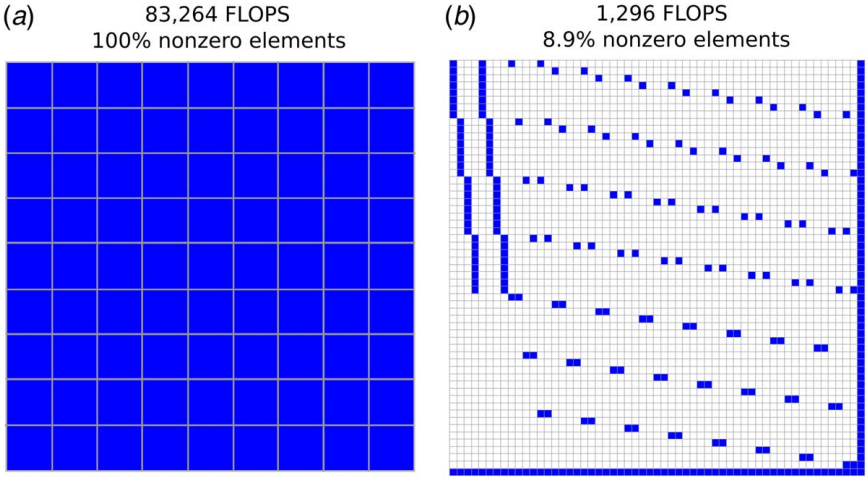

Fig. 2 Percentage of nonzero elements and patterns for $\mathbf{J_v}$ for both the eliminated and uneliminated formulation of the four-bar problem as well as the total number of FLOPS required to evaluate the matrices: (*a*) eliminated problem and (*b*) uneliminated problem

$\mathcal{O}(n^2)$, while performing an LU decomposition which has a complexity of $\mathcal{O}((2/3)n^3)$. Thus, solving a dense linear system is $\mathcal{O}((2/3)n^3 + 2n^2)$ or $\mathcal{O}(n^3)$ for large n.

Sparse matrices are typically described in *triplet form* where the row, column, and value of all nonzero entities are specified. Any row and column not specified is assumed to be zero. This data structure is easily converted into other data structures such as *compressed-column* or *compressed-row* forms to allow for memory savings. Special algorithms are tuned for these data structures to facilitate matrix–matrix or matrix–vector computations [31].

Factoring a sparse matrix into **L** and **U** components must be done carefully to avoid *fill-ins*. A *fill-in* is the generation of a new nonzero element in a position in either **L** or **U** that is otherwise zero in the corresponding position of **A**. Sparse matrices gain their speedup in computation in part by avoiding superfluous fill-in operations. Rose and Tarjan [33] were first to describe a process for factoring **A** to reduce fill-ins. To do this, they created a directed graph from **A** where edge (i, j) exists if $a_{ij} \neq 0$. This *adjacency graph* can then be re-ordered which has the effect of permuting the rows and columns of **A**. This re-ordering is done in a way that the number of fill-ins is reduced, but not necessarily minimal as finding an ordering that produces a minimal amount of fill-ins is an NP-complete problem [33,34].

There exists many methods of re-ordering the adjacency graph to find an ordering that reduces fill-ins. Some popular orderings include Cuthill–McKee [35], minimum degree [36,37], minimum deficiency [38], and nested dissection [39]. This re-ordering is also known as symbolic factorization, after which numerical factorization is performed to actually compute the **L** and **U** components.

Numerical factorization routines include supernodal [40], frontal [41], and multifrontal [42] methods. Numerical factorization must be performed for each decomposition, however, if multiple **A** matrices have the same sparsity pattern but different coefficients, the symbolic factorization only needs to be performed once.

3 Implementation

In this section, we detail the implementation of our proposed method. The code was written in C++, using Eigen 3 [43] as the linear algebra library for the matrices and solvers, both sparse and dense. We chose the SuperLU [44] package for our sparse solver using a minimum degree ordering for symbolic factorization. This ordering is performed once, and then used in all subsequent numerical factorizations. Microsoft's Visual Studio was used as the development environment. The code was compiled using the Microsoft Visual C++ 14.2 compiler to run on Windows x64 architecture.

The elements of $\mathbf{J_v}$ were calculated using Mathematica 12, with the resulting expressions being converted to C++. From these expressions, we were able to evaluate the number of FLOPS required to evaluate $\mathbf{J_v}$. This was done by computing the number of mathematical operations (addition, subtraction, multiplication, division, and power) in each expression. Then, knowing the number of FLOPS required for each operation when using complex numbers (two for addition/subtraction and six for multiplication/division), we calculated the total number of FLOPS from all the operations. The results and sparsity pattern for the four-bar problem can be found in Fig. 2. For the first example problem below (four-bar synthesis), for the four-bar problem the eliminated $\mathbf{J_v}$ required 64× more FLOPS to evaluate than its uneliminated counterpart. For the second example problem below (six-bar synthesis), the eliminated $\mathbf{J_v}$ required 3× operations to evaluate (see Fig. 3).

In order to shorten homotopy path lengths and easily represent points at infinity where they should arise [6], all the path tracking was done in projective space. This was accomplished by introducing a homogeneous coordinate to the variable set. The tracker then tracks the ratio of all other variables $\{z_1, z_2, \ldots, z_n\}$ to the reference coordinate by generating the $n+1$ vector:

$$\mathbf{Z} = \{Z_0, Z_1, \ldots, Z_n\} \quad \text{where} \quad \begin{Bmatrix} 1 \\ \mathbf{z} \end{Bmatrix} = \frac{\mathbf{Z}}{Z_0} \tag{11}$$

A point is considered to be at infinity when $Z_0 = 0$. Numerically, this is evaluated as the metric

$$Z_0 < \epsilon \tag{12}$$

for a sufficiently small ϵ. The homogeneous coordinate must also be introduced to the variables of $\mathbf{f(z; q}(t))$ by replacing each variable z_j with $z_j = \dfrac{Z_j}{Z_0}$ and clearing the denominator to generate the new homogeneous polynomial system $\mathbf{F(Z; q}(t))$. This is a system of n equations in $n+1$ variables, including Z_0.

To square the system up, the Euclidean projective patch equation is added

$$\mathbf{u} \cdot \begin{Bmatrix} \mathbf{Z} \\ t \end{Bmatrix} - 1 = 0 \tag{13}$$

to the end of $\mathbf{F(Z\ q}(t))$, where $\mathbf{u}$ is an $n+2$ vector of random complex coefficients. This equation defines a linear form that all other points are referenced to in projective space [21]. For convenience, we can combine $\mathbf{Z}$ and t into a single vector $\mathbf{Y} = \{\mathbf{Z}, t\}$

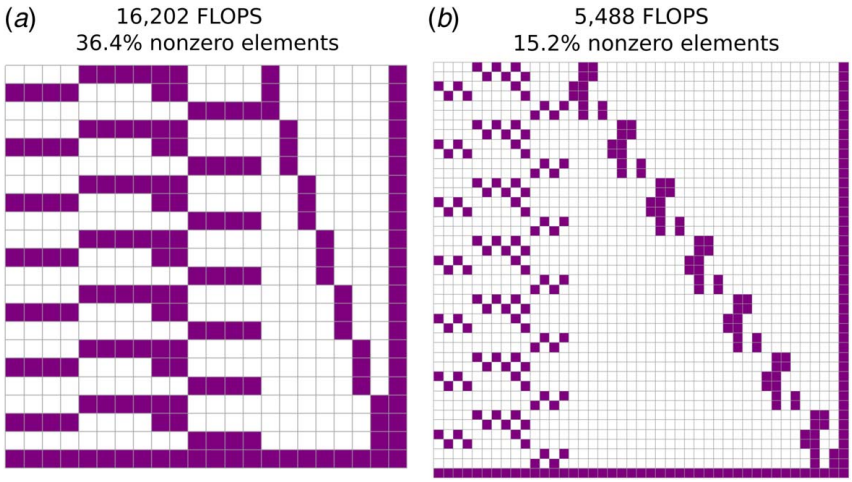

Fig. 3 Percentage of nonzero elements and patterns for J_v for both the eliminated and uneliminated formulation of the six-bar problem as well as the total number of FLOPS required to evaluate the matrices: (*a*) eliminated problem and (*b*) uneliminated problem

and create the new projective homotopy system:

$$\mathbf{H}(\mathbf{Y}) = \begin{Bmatrix} \mathbf{F}(\mathbf{Z};\, \mathbf{q}(t)) \\ \mathbf{u}\cdot\mathbf{Y} - 1 \end{Bmatrix} = 0 \tag{14}$$

Rather than working with t, we consider $\mathbf{Y}$ as a function of the path's arc length s. Evaluating the derivative of $\mathbf{H}$ with respect to s yields

$$\frac{\partial \mathbf{H}}{\partial \mathbf{Y}}\frac{\mathrm{d}\mathbf{Y}}{\mathrm{d}s} = 0 \tag{15}$$

where $\dfrac{\partial \mathbf{H}}{\partial \mathbf{Y}}$ is an $(n+1)\times(n+2)$ matrix with the form:

$$\frac{\partial \mathbf{H}}{\partial \mathbf{Y}} = \begin{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{F}}{\partial \mathbf{Z}} \end{bmatrix} & \dfrac{d\mathbf{F}}{dt} \\ \cdots \mathbf{u}^{\mathbf{T}} \cdots & \end{bmatrix} \quad \text{where} \quad \frac{\mathrm{d}\mathbf{F}}{\mathrm{d}t} = \frac{\partial \mathbf{F}}{\partial \mathbf{q}}\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} \tag{16}$$

$\dfrac{\mathrm{d}\mathbf{q}}{\mathrm{d}t}$ is defined in (3). In the projective case, $\left[\dfrac{\partial \mathbf{F}}{\partial \mathbf{Z}}\right]$ is an $n\times(n+1)$ matrix, $\dfrac{d\mathbf{F}}{dt}$ an $n\times 1$ vector, $\left[\dfrac{\partial \mathbf{F}}{\partial \mathbf{q}}\right]$ an $n\times m$ matrix, and $\dfrac{\mathrm{d}\mathbf{q}}{\mathrm{d}t}$ an $m\times 1$ vector.

Equation (15) is solved to calculate the next predicted point $\tilde{\mathbf{Y}}_n$ in the sequence using a chosen integration method for a given step size Δs. A row of $n+1$ random coefficients is appended to $\dfrac{\partial \mathbf{H}}{\partial \mathbf{Y}}$ to square it up, and an extra 1 is appended to the vector of zeros on the right-hand side.

The predicted point is then corrected by solving (14) using Newton's method with $\tilde{\mathbf{Y}}_n$ as an initial value. To square up (14), we append the equation

$$\hat{\mathbf{H}}(\mathbf{Y}) = \begin{bmatrix} \mathbf{H}(\mathbf{Y}) \\ \tilde{\mathbf{V}}_{\mathrm{n}}\cdot(\mathbf{Y} - \delta\tilde{\mathbf{Y}}_{\mathrm{n}}) \end{bmatrix} \tag{17}$$

where $\tilde{\mathbf{V}}_n$ is an estimate of the tangent vector obtained by solving (15) with $\tilde{\mathbf{Y}}_n$. The appended equation describes a plane that is almost coincident with $\tilde{\mathbf{Y}}_n$ and is normal to $\tilde{\mathbf{V}}_n$. The correction factor δ is a small complex number used to ensure that t monotonically increases to $t=1$.

Newton's method is therefore iteratively executed as

$$\mathbf{Y}_{k+1} = \mathbf{Y}_k - \left[\boldsymbol{\Gamma}\right]_k \hat{\mathbf{H}}(\mathrm{Y})_k \tag{18}$$

$$\text{where} \quad \left[\boldsymbol{\Gamma}\right]_k = \begin{bmatrix} \begin{bmatrix} \dfrac{\partial \mathbf{H}}{\partial \mathbf{Y}_k} \end{bmatrix} \\ \tilde{\mathbf{V}}^{T}{}_{\mathrm{n}} \end{bmatrix}^{-1} \tag{19}$$

The last element of (18) can be isolated and written as

$$t_{k+1} = t_k - \boldsymbol{\Gamma}_{k,n+2}\cdot\left\{\frac{\mathbf{H}(\mathbf{Y_k})}{\tilde{\mathbf{V}}_{\mathrm{n}}\cdot(\mathbf{Y}_k - \delta\tilde{\mathbf{Y}}_{\mathrm{n}})}\right\} \tag{20}$$

where $\boldsymbol{\Gamma}_{k,n+2}$ is that last row of $\boldsymbol{\Gamma}_k$. At t_k, everything in (21) is known except for t_{k+1} and δ. Setting t_{k+1} equal to the last element of $\tilde{\mathbf{Y}}_n$ yields δ. Next, we can solve (18) for $\mathbf{Y}_{k+1}$ and iteratively execute Newton's method until convergence. It is in this way that the prediction and correction steps are computed in projective space.

One of the advantages of projective space is that the projective patch can be exchanged for another at any point. Switching patches alter the condition number of the Jacobian in (16). As the projective patch only defines one row of the Jacobian, it only has so much sway over its condition number. But at times when Newton's method is struggling to converge, altering it can improve conditioning enough to enable convergence. When Newton's method fails to converge, the path can be modified by generating a new random patch $\mathbf{u}_{\mathrm{new}} = \{u_0,\, u_1,\, \ldots,\, u_n,\, u_t\}$ and rescaling $\mathbf{Z}$ so that it satisfies the patch equation (13):

$$\mathbf{u}_{\mathrm{new}}\cdot\begin{Bmatrix} \sigma\mathbf{Z} \\ t \end{Bmatrix} - 1 = 0 \tag{21}$$

Solving (22) for σ yields

$$\sigma = \frac{1 - u_t t}{u_{\mathrm{new}}\cdot\begin{Bmatrix} \mathbf{Z} \\ 0 \end{Bmatrix}} \tag{22}$$

The point that failed to converge is then updated as

$$\mathbf{Y} = \begin{Bmatrix} \sigma\mathbf{Z} \\ t \end{Bmatrix} \tag{23}$$

and path tracking can attempt to continue after switching patches. Using this method, paths that would otherwise fail can be potentially salvaged to continue on to become solutions.

For the predictor, we chose to implement Dormand–Prince [27] from the Runge–Kutta family of integrators. This method provides a fourth-order $\mathbf{R}_4$ as well as a fifth-order $\mathbf{R}_5$ prediction of the next point. We define the *local truncation error* (LTE) for the prediction

to be

$$\text{LTE} = \|\mathbf{R}_5 - \mathbf{R}_4\| \tag{24}$$

The LTE is a metric for the confidence of a prediction. In our implementation, we define ξ to be the maximum allowable LTE for the prediction step. If LTE$>\xi$, then the prediction is rejected and a new prediction is made using a smaller step size.

For the correction step, we test for convergence by setting a convergence minimum ν and using the metric

$$\nu < \left\| \left[\mathbf{\Gamma} \right]_k \hat{\mathbf{H}}(\mathrm{Y})_k \right\| \tag{25}$$

to test for convergence. The right-hand side of the equation comes from (18). The number of Newton's method iterations is logged and the correction step is rejected if a specified maximum number of iterations has occurred. For this paper, we decided on a maximum of 10 Newton's method iterations. If a correction step is rejected, then a new patch is randomly generated and the point is updated using (24) and Newton's method is attempted again. If Newton's method still fails to converge after five consecutive path switches, the path is rejected and tracking is terminated.

Finally, the number of prediction–correction steps is logged and if a path exceeds a prescribed number of steps, tracking halts and the path are declared a failure. This is to keep very poorly behaving paths from using excessive computation time.

4 Demonstration

In this section, we demonstrate our method on two kinematic synthesis problems. The first is the four-bar synthesis problem for nine path points. This is a classical problem in kinematic synthesis, making it a nice benchmark. The first ab initio solution to this problem found 8652 roots and was computed in 1992 by Wampler et al. [14]. Despite this landmark result, subsequent parameter homotopies directed at physical parameters still present a computational bog, as we illustrate below, albeit at a smaller scale. The second problem is the Stephenson IIB timed curve synthesis, which seeks to design a planar six-bar mechanism. This problem is larger (estimated to have 1,122,022 roots) and was solved for the first time in 2020 [10].

For both these problems, the number of solutions found and the runtime are compared for dense and sparse implementations, eliminated (small) and uneliminated (large) equation sets. Additionally, we will compare our method to that of BERTINI [22], a sophisticated homotopy continuation solver that represents the state of the art. All computations referenced in this section were executed on an Intel i7-8700K 3.7GHz processor running on a single thread.

4.1 Four-Bar Synthesis for Nine Path Points.

The goal is to design a four-bar linkage that guides a trace point attached to its coupler link through a set of N points P_j, $j=0, \ldots, N-1$. In a complex plane, we seek to find the two points of fixed pivots $A = A_x + iA_y$ and $B = B_x + iB_y$ as well as the locations of the moving pivots $C = C_x + iC_y$ and $D = D_x + iD_y$ that define the mechanism. The points C and D connect to the first task position P_0 to define how the trace point connects to the coupler link.

4.1.1 Loop Equations. The displacement of the linkage from its original configuration at P_0 requires moving the three links AC, BD, and CDP. We define the orientation of each link in the jth position by the angles ϕ_j, ψ_j, and θ_j, respectively. In the complex plane, these rotations are represented by complex numbers as

$$Q_j = e^{i\phi_j}, \quad S_j = e^{i\psi_j}, \quad T_j = e^{i\theta_j}, \quad j = 1, \ldots, N-1 \tag{26}$$

These orientations are relative to the first position, thus $Q_0 = S_0 = T_0 = 1$. This yields the two loop equations that define the geometry

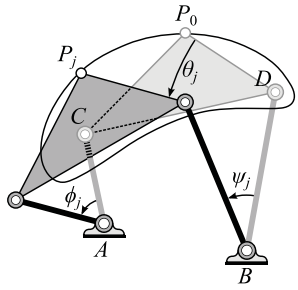

Fig. 4 A four-bar linkage displacing from P_0 to P_j

that must be preserved throughout the motion as

$$\begin{aligned} A + Q_j(C - A) + T_j(P_0 - C) - P_j &= 0 \\ B + S_j(D - B) + T_j(P_0 - D) - P_j &= 0 \end{aligned} \tag{27}$$

$$j = 1, \ldots, N-1 \tag{28}$$

See Fig. 4. Since points are represented as isotropic coordinates, we must also append the conjugate loop equations:

$$\begin{aligned} \bar{A} + \bar{Q}_j(\bar{C} - \bar{A}) + \bar{T}_j(\bar{P}_0 - \bar{C}) - \bar{P}_j &= 0 \\ \bar{B} + \bar{S}_j(\bar{D} - \bar{B}) + \bar{T}_j(\bar{P}_0 - \bar{D}) - \bar{P}_j &= 0 \end{aligned} \tag{29}$$

$$j = 1, \ldots, N-1 \tag{30}$$

where the overbar denotes the complex conjugate. Additionally, the rotation operators Q_j, S_j, and T_j are required to have unit magnitude, forming the normalization conditions:

$$Q_j \bar{Q}_j = 1 \tag{31}$$

$$S_j \bar{S}_j = 1 \tag{32}$$

$$T_j \bar{T}_j = 1, \quad j = 1, \ldots, N-1 \tag{33}$$

Equations (28)–(34) form a system of $7(N-1)$ equations in $8 + 6(N-1)$ variables, which is well defined when $N = 9$ task positions. This square uneliminated system results in 56 equations and variables: $\{A, \bar{A}, B, \bar{B}, C, \bar{C}, D, \bar{D}, Q_j, \bar{Q}_j, S_j, \bar{S}_j, T_j, \bar{T}_j\}$, $j = 1, \ldots, 8$, parameterized by 18 values, $\{P_0, \bar{P}_0, P_1, \bar{P}_1, \ldots, P_8, \bar{P}_8\}$.

4.1.2 Algebraic Elimination. The rotation operators $(Q_j, \bar{Q}_j)$ are eliminated by solving (28) and (30) for Q_j and $\bar{Q}_j$, respectively, and substituting the results into (32). Similarly, $(S_j, \bar{S}_j)$ are eliminated by solving (29) and (31) for S_j and $\bar{S}_j$ with the results substituted into (33) to obtain the equations:

$$\begin{aligned} (C - A)(\bar{C} - \bar{A}) &= (A - P_j + T_j(P_0 - C)) \\ &\quad (\bar{A} - \bar{P}_j + \bar{T}_j(\bar{P}_0 - \bar{C})) \\ (D - B)(\bar{D} - \bar{B}) &= (B - P_j + T_j(P_0 - D)) \\ &\quad (\bar{B} - \bar{P}_j + \bar{T}_j(\bar{P}_0 - \bar{D})) \end{aligned} \tag{34}$$

$$j = 1, \ldots, N-1 \tag{35}$$

The final rotation operators $(T_j, \bar{T}_j)$ are eliminated by expanding (35)–(36) and writing the result as the linear system:

$$\begin{bmatrix} a\bar{b}_j & \bar{a}b_j \\ c\bar{d}_j & \bar{c}d_j \end{bmatrix} \begin{Bmatrix} T_j \\ \bar{T}_j \end{Bmatrix} = \begin{Bmatrix} f\bar{f} - a\bar{a} - b_j\bar{b}_j \\ g\bar{g} - c\bar{c} - d_j\bar{d}_j \end{Bmatrix} \tag{36}$$

where

$$\begin{aligned} a &= P_0 - C, \quad & b_j &= A - P_j, \quad & f &= C - A \\ c &= P_0 - D, \quad & d_j &= B - P_j, \quad & g &= D - B \end{aligned}$$

Solving (37) for $(T_j, \bar{T}_j)$ and substituting the results into (34) yields

$$\begin{aligned}&(a\bar{b}_j(g\bar{g} - c\bar{c} - d_j\bar{d}_j) - c\bar{d}_j(f\bar{f} - a\bar{a} - b_j\bar{b}_j))\\&\quad\times(\bar{a}b_j(g\bar{g} - c\bar{c} - d_j\bar{d}_j) - \bar{c}d_j(f\bar{f} - a\bar{a} - b_j\bar{b}_j))\\&\quad+(a\bar{b}_j\bar{c}d_j - \bar{a}b_jc\bar{d}_j)^2 = 0\end{aligned} \tag{37}$$

This eliminated system (38) forms a group of $N-1$ equations in variables $\{A, \bar{A}, B, \bar{B}, C, \bar{C}, D, \bar{D}\}$, which again is well defined when $N=9$ task positions. This forms a system of eight equations in eight variables, parameterized by 18 values $\{P_0, \bar{P}_0, P_1, \bar{P}_1, \ldots, P_8, \bar{P}_8\}$.

4.1.3 Ab Initio. In a previous work, Plecnik and Fearing [45] were able to generate the complete starting set of 8652 roots using their finite root generation method. This set was used as the $\{A, \bar{A}, B, \bar{B}, C, \bar{C}, D, \bar{D}\}$ starting values for the eliminated problem, rather than re-computing the set again in an ab initio run. Additionally, we note that although it is well known that solutions may be organized into sets of six that reduce the required number of startpoints to 1442, we chose to benchmark on the full set of 8652.

The uneliminated problem used the same set as its starting values. However, it was necessary to find the $\{Q_j, \bar{Q}_j, S_J, \bar{S}_j, T_j, \bar{T}_j\}$, $j = 1, \ldots, 8$, for each root as well. To do this, we solved the inverse kinematics problem by substituting the known starting values into (35) as well as the starting parameters $\{P_0, P_j, \bar{P}_j\}$, $j = 1, \ldots, 8$. We then solved the substituted system and (34) for the corresponding set of two $\{T_j, \bar{T}_j\}$ values. Similar substitutions were performed on (36) and solved with (34) to find a second set of $\{T_j, \bar{T}_j\}$ pairs. The $(T_j, \bar{T}_j)$ values that matched between the two sets were chosen as the correct value for the jth parameter, $j = 1, \ldots, 8$.

Then, for each $(T_j, \bar{T}_j)$ value, the corresponding $(Q_j, \bar{Q}_j)$ values were found by substituting the known values into (28) and (30) and solving the resulting equations for $(Q_j, \bar{Q}_j)$. Similar substitutions into (29) and (31) yielded $(S_j, \bar{S}_j)$. It was in this way that the 8652 roots of the starting uneliminated system were generated from the eliminated set.

4.1.4 Path Tracking Results. Both the eliminated and uneliminated polynomial systems were solved using the same set of target parameters, taken from problem 1 in Ref. [14]. Each of the 8652 paths were tracked single-threaded on the same hardware. A path at z^* was declared a success at $t=1$ if $\|f(z^*; q(1))\| < \sigma$ with a cutoff value of $\sigma = 1 \times 10^{-12}$. The results can be found in Table 1.

Table 1 Performance results for the four-bar path synthesis example

Equation type	Linear solver type	Precision	Success rate (%)	Time (s)
Eliminated	Dense	Double	89.4%	2413
Eliminated	Dense	Quad	98.4%	751,305
Eliminated	Sparse	Double	82.8%	9536
Uneliminated	Dense	Double	95.1%	16,001
Uneliminated	Sparse	Double	96.4%	3002

Notes: The baseline performance of using the eliminated equations with a dense linear solver in double precision yielded a modest success rate. This success rate was improved by increasing precision from double to quads, but doing so greatly increased the runtime. Using the eliminated equations with a sparse solver did not improve performance or accuracy, likely due to the associated linear systems being solved being fully dense. To contrast this, using the uneliminated formulation with a dense linear solver had a good success rate, but a longer runtime due to the increase in dimensionality over the eliminated formulation. Finally, our proposed methodology of using the uneliminated formulation with a sparse linear solver (highlighted) yielded a success rate comparable to using the eliminated formulation and quad math, but with a greatly improved runtime.

Solving the eliminated system with a dense linear solver achieved an 89.4% success rate when using double precision in 2413 s ($\approx$ 40.2 min). In contrast, solving the uneliminated system with a sparse solver took longer of 3002 s ($\approx$ 50 min) but achieved a success rate of 96.4%.

4.2 Stephenson IIB Timed Curve Generator. The ab initio solutions of all six-bar timed curve path generators were first computed by Baskar and Plecnik [10]. We select one six-bar timed curve generator, the Stephenson IIB, to further benchmark our technique contributed in this work. The synthesis of timed curve path generation refers to finding linkage dimensions that move a trace through desired points as coordinated with the angle of an input link. Such a synthesis approach enables a designer to prescribe the transmission ratio of forces as well as the motion of the mechanism.

4.2.1 Loop Equations. The goal of the synthesis is to find the location of fixed pivots $\{F, H\}$ as well as the set of moving pivots $\{A, B, C, D, G\}$. Similar to the four-bar path synthesis problem, these points are also complex numbers. In this case, the designer specifies N task positions P_j, as well as N input angles, ρ_j, $j=0, \ldots, N$. We consider displacements relative to the initial position P_0. As this is a more complicated mechanism, three loop equations are required to define the geometry that must be preserved. These equations are defined as

$$H + V_j(G - H) + S_j(P_0 - G) = P_j \tag{38}$$

$$\begin{aligned}F + R_j(A - F) + Q_j(B - A) + S_j(P_0 - B) &= P_j\\F + R_j(C - F) + T_j(D - C) + S_j(P_0 - D) &= P_j\end{aligned} \tag{39}$$

$$j = 1, \ldots, N-1 \tag{40}$$

where $\{Q_j, R_j, S_j, T_j, V_j\}$ are rotation operators defined as

$$Q_j = e^{i\phi_j},\ R_j = e^{i\rho_j},\ S_j = e^{i\psi_j},\ T_j = e^{i\theta_j},\ V_j = e^{i\nu_j}$$

See Fig. 5. Without loss of generality, P_0 is eliminated by considering all displacements to be relative to the first position.

$$W_j = P_j - P_0, \quad j = 1, \ldots, N-1 \tag{41}$$

This results in the set of simpler equations

$$H + V_j(G - H) - S_jG = W_j \tag{42}$$

$$\begin{aligned}F + R_j(A - F) + Q_j(B - A) - S_jB &= W_j\\F + R_j(C - F) + T_j(D - C) - S_jD &= W_j\end{aligned} \tag{43}$$

$$j = 1, \ldots, N-1 \tag{44}$$

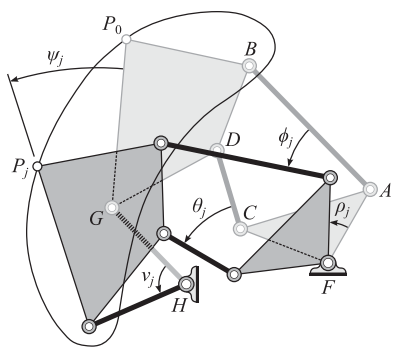

Fig. 5 A Stephenson II-B six-bar linkage displacing from P_0 to P_j

Using isotropic coordinates, the conjugate equations must also be added

$$\bar{H} + \bar{V}_j(\bar{G} - \bar{H}) - \bar{S}_j\bar{G} = \bar{W}_j \tag{45}$$

$$\begin{aligned} \bar{F} + \bar{R}_j(\bar{A} - \bar{F}) + \bar{Q}_j(\bar{B} - \bar{A}) - \bar{S}_j\bar{B} &= \bar{W}_j \\ \bar{F} + \bar{R}_j(\bar{C} - \bar{F}) + \bar{T}_j(\bar{D} - \bar{C}) - \bar{S}_j\bar{D} &= \bar{W}_j \end{aligned} \tag{46}$$

$$j = 1, \ldots, N-1 \tag{47}$$

As well as the normalization conditions:

$$Q_j\bar{Q}_j = 1 \tag{48}$$

$$S_j\bar{S}_j = 1 \tag{49}$$

$$T_j\bar{T}_j = 1 \tag{50}$$

$$V_j\bar{V}_j = 1, \quad j = 1, \ldots, N-1 \tag{51}$$

Note that $(R_j, \bar{R}_j)$ do not appear in the normalization equations as they are part of the $\{P_0, \bar{P}_0, P_j, \bar{P}_j, R_j, \bar{R}_j\}$ parameters specified by the designer. The complex conjugates of the rotation operators $\{\bar{Q}_j, \bar{S}_j, \bar{T}_j, \bar{V}_j\}$ are eliminated in this problem first by establishing the identities

$$\bar{Q}_j = \frac{1}{Q_j}, \quad \bar{S}_j = \frac{1}{S_j}, \quad \bar{T}_j = \frac{1}{T_j}, \quad \bar{V}_j = \frac{1}{V_j} \tag{52}$$

from (49) to (52) and substituting the results into (46)–(48) and clearing the denominators. This elimination results in the equations:

$$\bar{H}S_jV_j + S_j(\bar{G} - \bar{H}) - V_j\bar{G} = \bar{W}_jS_jV_j \tag{53}$$

$$\bar{F}Q_jS_j + \bar{R}_jQ_jS_j(\bar{A} - \bar{F}) + S_j(\bar{B} - \bar{A}) - Q_j\bar{B} = \bar{W}_jQ_jS_j \tag{54}$$

$$\bar{F}S_jT_j + \bar{R}_jS_jT_j(\bar{C} - \bar{F}) + S_j(\bar{D} - \bar{C}) - T_j\bar{D} = \bar{W}_jS_jT_j \tag{55}$$

Equations (42)–(44) and (53)–(55) represent a system of $6(N-1)$ equations in $14 + 4(N-1)$ variables $\{A, \bar{A}, B, \bar{B}, C, \bar{C}, D, \bar{D}, F, \bar{F}, G, \bar{G}, H, \bar{H}, Q_j, S_j, T_j, V_j\}$, which is square for N=8 task positions. This yields the uneliminated system of 42 equations in 42 variables parameterized by 28 values $\{W_1, \bar{W}_1, \ldots, W_7, \bar{W}_7, R_1, \bar{R}_1, \ldots, R_7, \bar{R}_7\}$.

Table 2 Performance results for the six-bar path synthesis example using the same randomly chosen subset of 10,000 paths of the original 1,122,022 roots

Equation type	Linear solver type	Precision	Success rate (%)	Runtime (s)
Eliminated	Dense	Double	75.8%	2654
Eliminated	Dense	Quad	97.8%	184,569
Eliminated	Sparse	Double	72.2%	1923
Uneliminated	Dense	Double	88.1%	12,033
Uneliminated	Sparse	Double	97.2%	3394

Notes: The baseline performance of using the eliminated equations with a dense linear solver in double precision yielded a poor success rate. This success rate was improved by increasing precision from double to quads, but doing so greatly increased the runtime. Using the eliminated equations with a sparse solver did not improve accuracy but had a slight improvement in runtime, likely due to the semi-sparse structure of the linear systems for the eliminated problem (see Fig. 3(*a*)). Using the uneliminated formulation with a dense linear solver had improved success rate, but a longer runtime due to the increase in dimensionality over the eliminated formulation. Finally, our proposed methodology of using the uneliminated formulation with a sparse linear solver (highlighted) again yielded a success rate comparable to using the eliminated formulation and quad math, but with a greatly improved runtime.

4.2.2 Algebraic Elimination. Similar to the previous problem, we seek to further simplify the system by eliminating the rotation operators $\{Q_j, T_j, V_j\}$. This is done by solving (42) and (53) for V_j and setting the results equal to each other to eliminate V_j. Q_j is eliminated in a similar manner using (43) and (54). Finally, T_j is eliminated by equating the results of (44) and (55). This results in the set of eliminated equations:

$$(W_j - H + S_jG)(\bar{H}S_j - \bar{G} - \bar{W}_jS_j) = -S_j(\bar{G} - \bar{H})(G - H) \tag{56}$$

$$\begin{aligned} &(W_j - F - R_j(A - F) + S_jB) \\ &\quad \times (\bar{F}S_j + \bar{R}_jS_j(\bar{A} - \bar{F}) - \bar{B} - \bar{W}_jS_j) \\ &\quad = -S_j(\bar{B} - \bar{A})(B - A) \end{aligned} \tag{57}$$

$$\begin{aligned} &(W_j - F - R_j(C - F) + S_jB) \\ &\quad \times (\bar{F}S_j + \bar{R}_jS_j(\bar{C} - \bar{F}) - \bar{D} - \bar{W}_jS_j) \\ &\quad = -S_j(\bar{D} - \bar{C})(D - C) \qquad j = 1, \ldots, (N-1) \end{aligned} \tag{58}$$

Equations (56)–(58) represent a system of $3(N-1)$ equations in $14 + (N-1)$ variables: $\{A, \bar{A}, B, \bar{B}, C, \bar{C}, D, \bar{D}, F, \bar{F}, G, \bar{G}, H, \bar{H}, S_j\}$, which again is square for N=8 task positions. This results in an eliminated polynomial system of 21 equations in 21 variables in 28 parameters $\{W_1, \bar{W}_1, \ldots, W_7, \bar{W}_7, R_1, \bar{R}_1, \ldots, R_7, \bar{R}_7\}$.

4.2.3 Ab Initio. The set of starting roots for the uneliminated problem has been previously computed in Ref. [10]. This set was found using a monodromy method to find 1,122,022 roots. For the eliminated problem, the same set can be used directly by omitting $\{Q_j, T_j, V_j\}$ from each root of the uneliminated start set. This subset satisfies the eliminated equations (56)–(58).

4.2.4 Path Tracking Results. To quantify the advantage of our approach, it is sufficient to select a representative subset from the 1,122,022 roots and extrapolate. A random selection of 10,000 roots was made to serve as start points for parameter homotopies for benchmarking. This randomized set was used for both the uneliminated and the eliminated problems. All 10,000 paths were tracked single-threaded on the same hardware. The task positions were taken from problem T-1 in Ref. [10]. As in the previous problem, a path at $\mathbf{z}^*$ was declared a success at $t = 1$ if $\|f(\mathbf{z}^*; \mathbf{q}(1))\| < \sigma$ with a cutoff value of $\sigma = 1 \times 10^{-12}$. The results can be found in Table 2.

The smaller eliminated system running a dense linear solver in double precision took 2654 s (≈44.2 min) to successfully track 75.8% of the 10,000 paths. Running the uneliminated system with a sparse solver took 3394 s (≈56.6 min) but achieved a success rate of 97.2%. Quad precision was required to achieve a similar success rate for the eliminated system, achieving a 97.8% success rate in 184,569 s (≈2.1 days) of computation time. Our approach combining uneliminated algebraic systems with sparse linear solvers created a 54× speedup over quad precision while maintaining a comparable accuracy level.

5 Discussion

For both problems, the eliminated version ran faster than the uneliminated version when both used dense linear solvers. This is expected as the linear systems in the uneliminated problems are larger than those of the eliminated. However, path tracking the uneliminated problem with sparse linear solvers was more accurate than tracking the eliminated problem with dense methods, with the difference becoming more pronounced in the more complex Stephenson IIB timed curve problem. Achieving a similar level of accuracy in the eliminated problem required moving to quad precision, but doing so paid a heavy penalty in runtime.

To highlight the necessity of using both the uneliminated equations and sparse routines, the success rates and times for eliminated/sparse and uneliminated/dense implementations have also been reported. For both problems, simply using a sparse solver did not increase the accuracy and runtime. The eliminated/sparse implementation saw a decrease in accuracy over its eliminated/dense counterpart. For the four-bar problem, there was an increase in runtime while for there was a decrease in runtime for the six-bar problem. This is likely due to the fact that the eliminated $\mathbf{J}_\mathbf{v}$ in the six-bar problem is still relatively sparse, while in the four-bar problem the eliminated matrix is fully dense. Additionally, for the uneliminated implementation of both problems, switching from a sparse to a dense linear solver resulted in a decrease in success rate and an increase in runtime.

When comparing the performance of our implementation with that of BERTINI, we found that for the four-bar path synthesis problem, BERTINI was able to successfully track 83% of paths in 218 s ($\approx$3.6 min) using double precision. Using quad precision, BERTINI successfully tracked 96% of paths in 19237 s ($\approx$5.3 h).

For the Stephenson IIB timing problem, BERTINI was able to find 8523 solutions in 338 s ($\approx$5.6 min) for a success rate of 85% using double precision, and 9670 solutions in 55,652 s ($\approx$15.5 h) for a success rate of 97% using quad precision. At this higher success rate, our uneliminated/sparse implementation had a speedup over BERTINI of 6.4× for the four-bar problem and 16.4× for the Stephenson IIB problem.

Extrapolating the results for 10,000 paths to the full 1,122,022 paths required for the Stephenson IIB problem, we estimate that our method of sparse solvers and uneliminated systems distributed over an high-performance computing cluster of 192 processors would take 1983 s ($\approx$33 min). Similarly, we estimate that BERTINI would take 6,244,277 ($\approx$9 h) to solve the same set to a comparable accuracy using quad precision.

It is important to note that Bertini implements additional sophisticated features that boost speed and accuracy which are not included in our basic sparse/unelimminated implementation. These features include automatic differentiation and endgame algorithms [22]. In light of this, the comparisons made in Tables 1 and 2 are experimentally more controlled and meaningful. BERTINI exclusively uses dense linear solvers as it is intended to solve any general system of polynomials with no guarantee on any special structure.

6 Conclusion

This paper presents a novel methodology for quickly and accurately solving kinematics equations without requiring non-native precision by both solving the original, uneliminated equations and using sparse linear solvers on the systems associated with path tracking. This provides a novel method for calculating the roots of a polynomial system using parameters associated with physical problems. These parameters are known to cause associated matrices to become ill-conditioned enough such that practical implementations of parameter homotopy on physical problems is greatly compromised by speed and precision setbacks. The increase in numerical failures can be alleviated by using 128-bit precision, but this greatly increases runtime as current hardware is not designed to natively run quad precision. For the problem of kinematic synthesis, the rotation operators of the loop equations are traditionally algebraically eliminated to decrease the size of the linear system to be solved. This has the effect of increasing the complexity and thus the number of floating point operations in evaluating the function and its corresponding Jacobian matrices. Leaving the rotation operators uneliminated decreases the cost of evaluating the functions and Jacobians but increases the size of the corresponding linear system. These uneliminated linear systems, while larger in dimension, end up being quite sparse in nature over their eliminated counterparts. By coupling the uneliminated kinematic synthesis equations with a sparse linear solver, we obtained a path tracking success rate comparable to running the eliminated system in quad math while keeping the speed of performing all calculations in double precision. This resulted in a maximum of 54.4× speedup over conventional methods.

Acknowledgment

This paper is based upon work supported by the National Science Foundation under Grant No. CMMI-2144732.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtainable from the corresponding author upon reasonable request.

References

[1] Lipson, J. D., 1976, "Newtons Method," Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation—SYMSAC 76, Yorktown Heights, NY, Aug. 10–12, ACM Press.
[2] Kolev, L., 2000, "An Interval Method for Global Nonlinear Analysis," IEEE Trans. Circ. Syst. I Fundam. Theory Appl., **47**(5), pp. 675–683.
[3] Boege, W., Gebauer, R., and Kredel, H., 1986, "Some Examples for Solving Systems of Algebraic Equations by Calculating Groebner Bases," J. Symbol. Comput., **2**(1), pp. 83–98.
[4] Freudenstein, F., and Roth, B., 1963, "Numerical Solution of Systems of Nonlinear Equations," J. ACM, **10**(4), pp. 550–556.
[5] Plecnik, M. M., 2015, "The Kinematic Design of Six-Bar Linkages Using Polynomial Homotopy Continuation," Ph.D. thesis, Department of Mechanical and Aerospace Engineering, University of California, Irvine, CA.
[6] Morgan, A., and Sommese, A., 1987, "A Homotopy for Solving General Polynomial Systems That Respects M-Homogeneous Structures," Appl. Math. Comput., **24**(2), pp. 101–113.
[7] Huber, B., and Sturmfels, B., 1995, "A Polyhedral Method for Solving Sparse Polynomial Systems," Math. Comput., **64**(212), pp. 1541–1541.
[8] Hauenstein, J. D., Sommese, A. J., and Wampler, C. W., 2010, "Regeneration Homotopies for Solving Systems of Polynomials," Math. Comput., **80**(273), pp. 345–377.
[9] Plecnik, M. M., and Fearing, R. S., 2017, "Finding Only Finite Roots to Large Kinematic Synthesis Systems," ASME J. Mech. Rob., **9**(2), p. 021005.
[10] Baskar, A., and Plecnik, M., 2020, "Synthesis of Six-Bar Timed Curve Generators of Stephenson-Type Using Random Monodromy Loops," ASME J. Mech. Rob., **13**(1), p. 011005.
[11] Morgan, A. P., and Sommese, A. J., 1989, "Coefficient-Parameter Polynomial Continuation," Appl. Math. Comput., **29**(2), pp. 123–160.
[12] Bates, D. J., Sommese, A. J., Hauenstein, J. D., and Wampler, C. W., 2013, *Numerically Solving Polynomial Systems With Bertini*, Society for Industrial and Applied Mathematics, Philadelphia, PA.
[13] Higham, N. J., 2002, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA.
[14] Wampler, C. W., Morgan, A. P., and Sommese, A. J., 1992, "Complete Solution of the Nine-Point Path Synthesis Problem for Four-Bar Linkages," J. Mech. Des., **114**(1), pp. 153–159.
[15] Schreiber, H., Meer, K., and Schmitt, B., 2002, "Dimensional Synthesis of Planar Stephenson Mechanisms for Motion Generation Using Circlepoint Search and Homotopy Methods," Mech. Mach. Theory, **37**(7), pp. 717–737.
[16] Morgan, A. P., Sommese, A. J., and Wampler, C. W., 1990, "Computing Singular Solutions to Nonlinear Analytic Systems," Numer. Math., **58**(1), pp. 669–684.
[17] Watson, L. T., Sosonkina, M., Melville, R. C., Morgan, A. P., and Walker, H. F., 1997, "Algorithm 777: HOMPACK90," ACM Trans. Math. Softw., **23**(4), pp. 514–549.
[18] Verschelde, J., 1999, "Algorithm 795," ACM Trans. Math. Softw., **25**(2), pp. 251–276.
[19] Wise, S. M., Sommese, A. J., and Watson, L. T., 2000, "Algorithm 801: POLSYS_PLP," ACM Trans. Math. Softw., **26**(1), pp. 176–200.
[20] Su, H.-J., McCarthy, J. M., Sosonkina, M., and Watson, L. T., 2006, "Algorithm 857," ACM Trans. Math. Softw., **32**(4), pp. 561–579.
[21] Sommese, A. J., and Wampler, C. W., 2005, *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science*, World Scientific, Singapore.
[22] Bates, D. J., Hauenstein, J. D., Sommese, A. J., and Wampler, C. W., 2007, "Bertini: Software for Numerical Algebraic Geometry," bertini.nd.edu.
[23] Newkirk, J. T., Watson, L. T., and Stanišić, M. M., 2010, "Determining the Number of Inverse Kinematic Solutions of a Constrained Parallel Mechanism Using a Homotopy Algorithm," ASME J. Mech. Rob., **2**(2), p. 024502.
[24] Plecnik, M. M., and Fearing, R. S., 2020, "Designing Dynamic Machines With Large-Scale Root Finding," IEEE Trans. Robot., **36**(4), pp. 1135–1152.

[25] Baskar, A., and Plecnik, M., 2021, "Synthesis of Watt-Type Timed Curve Generators and Selection From Continuous Cognate Spaces," ASME J. Mech. Rob., **13**(5), p. 051003.
[26] Fehlberg, E., 1968, "Classical Fifth-, Sixth-, Seventh-, and Eighth-Order Runge–Kutta Formulas With Stepsize Control," NASA Technical Report R-287.
[27] Dormand, J., and Prince, P., 1980, "A Family of Embedded Runge–Kutta Formulae," J. Comput. Appl. Math., **6**(1), pp. 19–26.
[28] IEEE Std 754-2019, 2019, "IEEE Standard for Floating-Point Arithmetic" (Revision of IEEE 754-2008), pp. 1–84.
[29] Bailey, D., 2005, "High-Precision Floating-Point Arithmetic in Scientific Computation," Comput. Sci. Eng., **7**(3), pp. 54–61.
[30] Wampler, C. W., 1996, "Isotropic Coordinates, Circularity, and Bezout Numbers: Planar Kinematics From a New Perspective," Proceedings of the ASME Design Engineering Technical Conference, Irvine, CA, Aug. 18–22.
[31] Davis, T. A., Rajamanickam, S., and Sid-Lakhdar, W. M., 2016, "A Survey of Direct Methods for Sparse Linear Systems," Acta Numer., **25**, pp. 383–566.
[32] Tewarson, R. P., 1970, "Computations With Sparse Matrices," SIAM Rev., **12**(4), pp. 527–543.
[33] Rose, D. J., and Tarjan, R. E., 1978, "Algorithmic Aspects of Vertex Elimination on Directed Graphs," SIAM J. Appl. Math., **34**(1), pp. 176–197.
[34] Gilbert, J. R., 1980, "A Note on the NP-Completeness of Vertex Elimination on Directed Graphs," SIAM J. Algebraic Discret. Methods, **1**(3), pp. 292–294.
[35] Cuthill, E., and McKee, J., 1969, "Reducing the Bandwidth of Sparse Symmetric Matrices," Proceedings of the 1969 24th National Conference, New York, NY, Aug. 26–28, ACM Press.
[36] Markowitz, H. M., 1957, "The Elimination Form of the Inverse and Its Application to Linear Programming," Manage. Sci., **3**(3), pp. 255–269.
[37] George, A., and McIntyre, D. R., 1978, "On the Application of the Minimum Degree Algorithm to Finite Element Systems," SIAM J. Numer. Anal., **15**(1), pp. 90–112.
[38] Berry, R., 1971, "An Optimal Ordering of Electronic Circuit Equations for a Sparse Matrix Solution," IEEE Trans. Circuit Theory, **18**(1), pp. 40–50.
[39] Kernighan, B. W., and Lin, S., 1970, "An Efficient Heuristic Procedure for Partitioning Graphs," Bell Syst. Tech. J., **49**(2), pp. 291–307.
[40] Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S., and Liu, J. W. H., 1999, "A Supernodal Approach to Sparse Partial Pivoting," SIAM J. Matrix Anal. Appl., **20**(3), pp. 720–755.
[41] Irons, B. M., 1970, "A Frontal Solution Program for Finite Element Analysis," Int. J. Numer. Methods Eng., **2**(1), pp. 5–32.
[42] Duff, I. S., and Reid, J. K., 1983, "The Multifrontal Solution of Indefinite Sparse Symmetric Linear," ACM Trans. Math. Softw., **9**(3), pp. 302–325.
[43] Guennebaud, G., and Jacob, B., 2010, "Eigen v3,"
[44] Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S., and Liu, J. W. H., 1999, "A Supernodal Approach to Sparse Partial Pivoting," SIAM J. Matrix Anal. Appl., **20**(3), pp. 720–755.
[45] Plecnik, M. M., and Fearing, R. S., 2017, "A Study on Finding Finite Roots for Kinematic Synthesis," Volume 5B: 41st Mechanisms and Robotics Conference, Cleveland, OH, Aug. 6–9, American Society of Mechanical Engineers.