Numerical Solutions for Kinematics of Multi-bar Mechanisms Using Graph Theory and Computer Simulations

*Brandon Torres & Mahdi Farahikia**

Division of Engineering Programs, State University of New York at New Paltz, New Paltz, NY

https://doi.org/10.33697/ajur.2024.118

Students: torresb4@newpaltz.edu
*Mentor: farahikm@newpaltz.edu**

ABSTRACT
A method is presented to demonstrate the application of computer simulations in the kinematic analysis of planar mechanisms, emphasizing its use in teaching the latter topic in a corresponding undergraduate course. Concepts of rigid-body dynamics are utilized in the kinematics of machines to analyze the motions (and forces in dynamics) transmitted within multiple interconnected links that make a mechanism, such as a car engine, airplane landing gear, press machine, door closer, and so on. Due to the tediousness of the analytical solutions, most textbooks limit the derivation of the equations to four-bar linkages like crank-rocker and crank-slider mechanisms. Benefiting from the advancements in computer programs, such as MATLAB, and their efficiency in solving large systems of linear and nonlinear equations, a method is proposed to facilitate teaching kinematic analysis of multi-bar linkages to undergraduate students while fostering the application of computational engineering via real-life examples. The results obtained from this method are shown to be in excellent agreement with the algebraic solution of the relative motion equations for each element in the mechanism.

KEYWORDS
Kinematics of Machines; Linkage; Numerical Solutions; Graph Theory; Machine Dynamics; Computer Simulation; Mechanism; Linkage Kinematics

INTRODUCTION

Mechanisms play important roles in a wide range of technologies, such as aircraft, steam engines, car engines, robotics, satellites, door closers, and so on[1–3]. At their core, these mechanisms are multi-body systems of interconnected links with the purpose of transmitting motion and force.

The design and synthesis of mechanisms to generate certain outputs (e.g., motion, path, or function) depend significantly on kinematics[1, 4]. The modern-day knowledge of kinematics of mechanisms and multi-body systems is based to a great extent on the works of Franz Reuleaux (German engineering scientist, considered as the "father of kinematics" by some) and his two major books, "*The Kinematics of Machinery*", and "*The Constructor*"[1, 5, 6]. Reuleaux's books and advocacy in mathematical approaches to mechanical engineering and machine kinematics inspired further development of texts in the late 19^{th} and early 20^{th} centuries, such as Kennedy, Hartmann, and Grübler[1, 7–9].

While the importance of understanding and studying the kinematics of machines is undeniable, derivation of standalone sets of governing equations for each type of linkage (four-bar, five-bar, etc.) can be mathematically "tedious", as shown by Norton[4]. As a result, the concentration of such techniques available in the literature (a summary of which

is given in the proceeding paragraphs) is limited mainly to four-bar (like crank-rocker and crank-slider mechanisms) and very few specific multi-bar linkages.

Norton and Uicker demonstrated the graphical analysis of the positions, velocities, and accelerations of linkages[4, 10]. Vector sums of the corresponding relative motion equations for two points on each link are represented graphically at a chosen scale. Although applicable, this method is not ideal for linkages with more than four bars or kinematic analysis of a linkage for its full range of motion, especially when analyzing accelerations composed of normal and tangential components.

Algebraic solutions for partial or complete kinematic analysis of specific four-bar mechanisms were explained by Norton, Uicker, and Martin[4, 10, 11]. Similar to the graphical method, vector sums of the relative motion equations are solved here for two points (typically joints) on each link based on the constraints imposed on each joint by the neighboring links. Although computer programming enables kinematic analysis of mechanisms for their full range of motion, extending this approach to linkages with more than four bars is time-consuming. Additionally, neither the graphical nor the algebraic solutions are specific to machine kinematics but an extension of the concepts of kinematics to interconnected bodies.

Analytical methods using vector loops and complex-number algebra were introduced by Norton, Uicker, Russell, Vinogradov, and Martin[4, 10–13]. At least one vector loop is constructed for a mechanism. The vector-loop equations are written in the complex-number format and expanded using Euler's identity to develop a set of n equations and n unknowns. For example, 1-DOF (Degree-of-Freedom) four-bar mechanisms lead to a set of two equations with two unknowns. The computation of the degrees of freedom (mobility) of a mechanism is demonstrated in the following section. The analytical solutions of these equations can be challenging and virtually impractical as the number of links increases[4]. Also, the number of unknowns for linkages with more than one degree of freedom exceeds the number of equations.

The vector loop approach and complex-number algebra combined with the capabilities of scientific computer programs, such as MATLAB and Python, provide an opportunity to develop a robust and efficient approach for the kinematic analysis of machines. Uicker mentioned position analysis of multi-bar linkages using numerical methods, such as Newton-Raphson, but mostly focused on abstract analytical solutions[10]. Russell introduced the application of MATLAB specific to a few examples but did not generalize it to arbitrary configurations of links[12]. A computational simulator for a five-bar "Gantry" mechanism was created using Python, Javascript, and Coördinator[14].

Despite recent advancements in scientific computer programs, for example, in solving large systems of linear and nonlinear equations and differential equations, their incorporation in relevant engineering courses remains inadequate. The current work aims to show the possibility of filling this gap in the context of developing a purely computational approach to the kinematics of machines by proposing a technique that merges classical multi-body dynamics and computer programming. The fundamentals of this method are based on generating the vector loop equations for mechanisms. However, instead of attempting to solve the large system of equations analytically, the goal here is to utilize the functionalities of MATLAB.

The advantage of the proposed method is twofold. First, it promotes the kinematic analysis of multi-bar mechanisms through a computational process of solving the vector loop equations instead of relying on commercial programs, like *Working Model*. Second, a stand-alone computer program can be produced by incorporating graph theory that identifies the vector loops automatically based on the provided input and performs a complete kinematic analysis for the full range of motion of the mechanism.

It is important to note that the goal here is not to undermine the significance of or eliminate the graphical, algebraic, and analytical methods described in the literature. An understanding of the fundamentals of classical dynamics is essential to validate the results obtained from numerical methods using in-house programs or commercial applications.

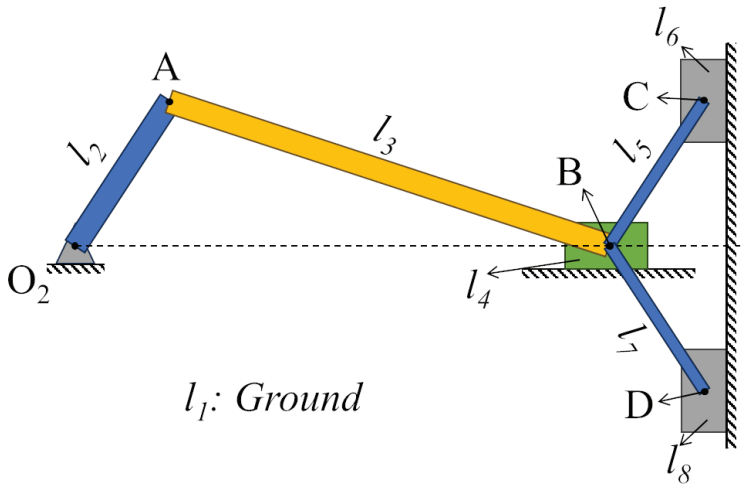

Figure 1. A schematic of an eight-bar linkage with one horizontal and two vertical sliders.

METHODS AND PROCEDURES

To demonstrate the proposed method, take the example of an eight-bar linkage shown in **Figure 1**. This mechanism consists of one horizontal slider (l_4) and two vertical sliders (l_6 and l_8.) The rotational motion of the input crank (l_2) is transferred to the horizontal slider by the connecting rod l_3. The connecting rods l_5 and l_7 transmit the horizontal translation of l_4 to the vertical translations of l_6 and l_8, respectively.

The revolute full joints O_2, A, C, and D are of Order one, whereas the Order of the revolute full joint B is three. There are three prismatic joints connecting the three sliders to the ground. In total, there are ten full joints and no half joints in this eight-bar mechanism. The mobility (Degree-of-Freedom) of this mechanism is one, given by **Equation 1**[4]

$$M = 3(L-1) - 2J_1 - J_2 = 3(8-1) - 2(10) - 0 = 1 \qquad \textbf{Equation 1.}$$

where M, L, J_1, and J_2 are the mobility, number of links, number of full joints, and number of half joints, respectively.

Vector Loop Generation

Vectors connect two joints on the same link so long as both joints connect the link to two adjacent links. The tail of each vector falls on the head of the vector representing an adjacent link. The ground link in each loop also has a vector. A local coordinate system is associated with each vector at its tail to demonstrate its direction. It is reasonable to choose a unique type of coordinate system, such as Cartesian, for all the local coordinate systems.

Ideally, a mechanism should be divided into multiple 1-DOF vector loops consisting of up to four links or two unknowns. The number of vector loops in a 1-DOF linkage is equal to the number of different joints connecting a link to the ground minus one. For example, in the eight-bar mechanism of **Figure 1**, four links (l_2, l_4, l_6, and l_8) have ground-connecting joints. Hence, this linkage can be represented by three vector loops (explained in the following), each consisting of three vectors, as shown in **Figure 2**. It is important to note that the fourth vector in each of the loops is the "offset" vector associated with sliders, all of which are set to zero for convenience but not necessity.

The first vector loop is O_2AB. This loop consist of vectors $\overrightarrow{O_2A}$, $\overrightarrow{AB}$, and $\overrightarrow{BO_2}$ whose complex-number notations are $ae^{j\theta_2}$, $be^{j\theta_3}$, and $ce^{j\theta_4}$, respectively, where $j = \sqrt{-1}$. In these equations, a, b, and c are magnitudes of the vectors, whereas θ_2, θ_3, and θ_4 represent the direction of each vector. In this loop, a, b, and $\theta_4 (= 180°)$ are constant, θ_2 is the input angular position of l_2, θ_3 is the unknown angular position of l_3, and the unknown rectilinear motion of slider l_4 is given by c.

The second vector loop, BCE, consists of vectors $\overrightarrow{BC}$, $\overrightarrow{CE}$, and $\overrightarrow{EB}$ with respective complex-number notations $de^{j\theta_5}$,

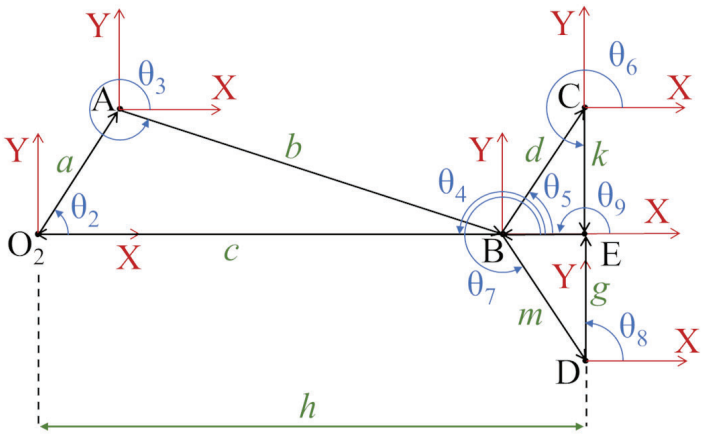

Figure 2. Corresponding vector loops for the eight-bar linkage.

$ke^{j\theta_6}$, and $(h-c)e^{j\theta_9}$. A virtual point E is chosen for this vector loop because it consists of two sliders. This virtual point can be placed anywhere; however, it is reasonable to choose a location that leads to convenient angles for the vectors. For instance, the constant angles θ_6 and θ_9 are 270° and 180°, respectively. The input to this vector loop is c, calculated in the first loop, and the two unknowns are the angular position, θ_5, of l_5 and the vertical length k. The length of the $\overrightarrow{EB}$ equals the constant h minus the input c.

The third vector loop, BDE, consists of vectors $\overrightarrow{BD}$, $\overrightarrow{DE}$, and $\overrightarrow{EB}$ represented by the complex-number notations $me^{j\theta_7}$, $ge^{j\theta_8}$, $(h-c)e^{j\theta_9}$, respectively. It is observed that vector $\overrightarrow{EB}$ is common between the second and third loops. The two unknowns in this loop are the angular position, θ_7, of l_7 and the vertical length g, while the input is the horizontal position, c, of slider l_4, as in the second vector loop. The angle θ_8 is constant at 90°.

Vector Loop Equations

For each loop in the linkage, it can be written that, due to the method of generating the vector loops, the sum of the vectors equals zero. Since there are three vector loops in the mechanism shown in **Figure 2**, three vector loop equations can be written for its position analysis in the complex-number notation.

$$\begin{cases} ae^{j\theta_2} + be^{j\theta_3} + ce^{j\theta_4} = 0 \\ de^{j\theta_5} + ke^{j\theta_6} + (h-c)e^{j\theta_9} = 0 \\ me^{j\theta_7} + ge^{j\theta_8} + (h-c)e^{j\theta_9} = 0 \end{cases} \quad \textbf{Equation 2.}$$

Equation 2 can be simplified to **Equation 3** by applying the known constants described above. Further simplification of this equation is not necessary in the development of the computational algorithm.

$$\begin{cases} ae^{j\theta_2} + be^{j\theta_3} + ce^{j\pi} = 0 \\ de^{j\theta_5} + ke^{j\frac{3\pi}{2}} + (h-c)e^{j\pi} = 0 \\ me^{j\theta_7} + ge^{j\frac{\pi}{2}} + (h-c)e^{j\pi} = 0 \end{cases} \quad \textbf{Equation 3.}$$

The six unknowns of this system of equations are θ_3, θ_5, θ_7, c, g, and k. Taking the first and second time-derivatives of **Equation 3** gives the necessary systems of equations to compute the velocity and acceleration of each link. Hence, for the three vector loops considered here, the velocity and acceleration equations are given by **Equations 4** and **5**, respec-

tively.

$$\begin{cases} ja\omega_2 e^{j\theta_2} + jb\omega_3 e^{j\theta_3} + \dot{c}e^{j\pi} = 0 \\ jd\omega_5 e^{j\theta_5} + \dot{k}e^{j\frac{3\pi}{2}} - \dot{c}e^{j\pi} = 0 \\ jm\omega_7 e^{j\theta_7} + \dot{g}e^{j\frac{\pi}{2}} - \dot{c}e^{j\pi} = 0 \end{cases} \quad \textbf{Equation 4.}$$

$$\begin{cases} ja\alpha_2 e^{j\theta_2} - a\omega_2^2 e^{j\theta_2} + jb\alpha_3 e^{j\theta_3} - b\omega_3^2 e^{j\theta_3} + \ddot{c}e^{j\pi} = 0 \\ jd\alpha_5 e^{j\theta_5} - d\omega_5^2 e^{j\theta_5} + \ddot{k}e^{j\frac{3\pi}{2}} - \ddot{c}e^{j\pi} = 0 \\ jm\alpha_7 e^{j\theta_7} - m\omega_7^2 e^{j\theta_7} + \ddot{g}e^{j\frac{\pi}{2}} - \ddot{c}e^{j\pi} = 0 \end{cases} \quad \textbf{Equation 5.}$$

Each system of equations above has two unknowns per equation, for a total of six unknowns. These unknowns are ω_3, ω_5, ω_7, $\dot{c}$, $\dot{g}$, and $\dot{k}$ for **Equation 4** and α_3, α_5, α_7, $\ddot{c}$, $\ddot{g}$, and $\ddot{k}$ for **Equation 5**. The input angular position (θ_2) and angular velocity (ω_2) of link l_2 can be calculated using dynamics principles ($\omega = \int_{t_0}^{t} \alpha dt + \omega_0$, and $\theta = \int_{t_0}^{t} \omega dt + \theta_0$.) To solve for the six unknowns, the *real* ($\mathfrak{Re}$) and *imaginary* ($\mathfrak{Im}$) components of its complex equations are set equal to zero, leading to two individual equations for each vector loop equation. For instance, the expansion of **Equation 3** is given in **Equation 6**. **Equations 4** and **5** can also be expanded similarly.

$$\begin{cases} \mathfrak{Re}[ae^{j\theta_2} + be^{j\theta_3} + ce^{j\pi}] = 0 \\ \mathfrak{Im}[ae^{j\theta_2} + be^{j\theta_3} + ce^{j\pi}] = 0 \\ \mathfrak{Re}[de^{j\theta_5} + ke^{j\frac{3\pi}{2}} + (h-c)e^{j\pi}] = 0 \\ \mathfrak{Im}[de^{j\theta_5} + ke^{j\frac{3\pi}{2}} + (h-c)e^{j\pi}] = 0 \\ \mathfrak{Re}[me^{j\theta_7} + ge^{j\frac{\pi}{2}} + (h-c)e^{j\pi}] = 0 \\ \mathfrak{Im}[me^{j\theta_7} + ge^{j\frac{\pi}{2}} + (h-c)e^{j\pi}] = 0 \end{cases} \quad \textbf{Equation 6.}$$

Although the components of a complex number can be derived using Euler's identity ($e^{j\theta} = cos(\theta) + jsin(\theta)$), corresponding built-in MATLAB functions are utilized in this method.

RESULTS

Vector Loop Generation Algorithm

The problem of automatically detecting loops for mechanisms is very similar to finding cycles (vector loops) in an undirected graph. The vectors for any cycle inherently form edges between connection points or pins on a link, which are called nodes. Any graph has a finite number of unique cycles, starting and ending with the same node. The proposed method requires that each edge in a cycle is traversed only once.

Figure 3 shows the decomposition of the mechanism in **Figure 1** in terms of edges and neighbors. Pin A on link l_2 is defined by nodes 3 (on l_2) and 4 (on l_3), and so on. Each node is attributed by the link to which it belongs and the nodes with which it neighbors. Node 3 belongs to link l_3 and neighbors nodes 2 (on l_2) and 4 (on l_3). In other words, one of the neighboring nodes belongs to the same link, and the other one belongs to an adjacent link. One edge is defined between nodes 2 and 3, both of which belong to link l_2. Another edge is defined between nodes 3 and 4, which belong to links l_2 and l_3, respectively. The edges that connect nodes on adjacent links are shown with dashed lines in **Figure 3**.

The method described here requires the minimum number of cycles to cover every node and edge in the graph that represents the mechanism. For instance, the eight-bar mechanism of **Figure 1** can be represented by six possible loops, but only three cycles are needed to cover all the nodes and edges. The *cyclebasis* function in MATLAB gives the minimum number of cycles for a given graph. This information is used to find the order of the links and directions of the vectors and is necessary when generating the vector loop equations ensuring that every link is included without redundancy.

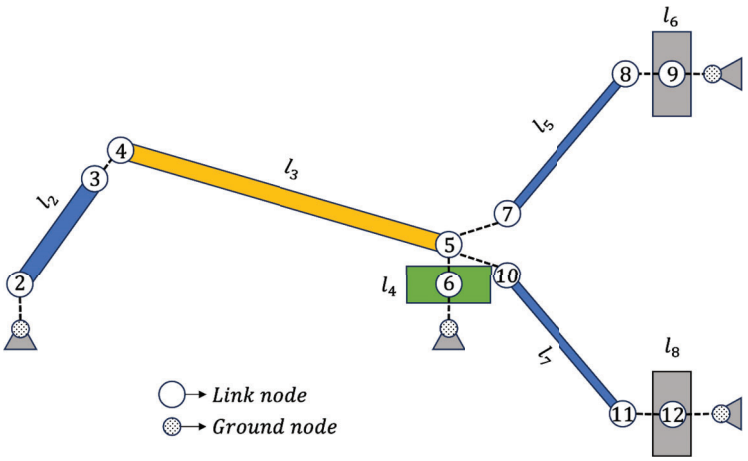

Figure 3. For each link, a node is created for points that connect to a neighboring link. Edges are formed at these connections and within the link itself. For example, l_2 would contain nodes 2 and 3, and node 3 would be neighbors with nodes 2 and 4. When defined this way, each loop can be found starting from and ending at the ground reference point, node 1. This method can be generalized for any mechanism.

For the numerical method presented here, an adjacency matrix is defined first. This is a square matrix whose rows represent a node of interest and whose columns refer to the neighboring nodes to that node. If the nodes represented by a row-column combination are connected by an edge, the corresponding element of the adjacency matrix takes a value of one. Otherwise, that element of the matrix will have a zero value. The adjacency matrix can be developed in a *for loop* as shown in the snippet below.

When the adjacency matrix is formed, it is provided as an input to the *graph* function that converts the matrix into a graphical object. The graphical object is used as the input to the *cyclebasis* function to generate the minimum number of unique cycles that represent the graph corresponding to the mechanism. The following snippet shows the MATLAB code for the steps explained above.

```
A = zeros(num_points+1); % Adjacency matrix as a square matrix of zeros
% We use the number of points +1 to account for the ground node

% Use nested for-loops to populate the elements of A
% First the link, second for each node on the link
for l = 1:length(LinkData)
   for p = 1:(LinkData(l).Point)
      currentNode = LinkData(l).Point{p}.NodeIndex;
      neighbors = LinkData(l).Point{p}.Neighbor;
      A(currentNode,neighbors) = 1; % If the nodes are connected, set the value to 1.
   end
end

G = graph(A); % Convert the matrix to graphical object.
cycles = cyclebasis(G); % Obtain the minimum unique cycles for the graph.
```

Figure 4 demonstrates the graph network defined for the eight-bar mechanism shown in **Figure 1** using the *cyclebasis* function in MATLAB. Three vector loops are automatically created for this mechanism because it can be divided into three 1-DOF linkages. Each loop has an area that does not overlap with the others and is essentially called a basis cycles.

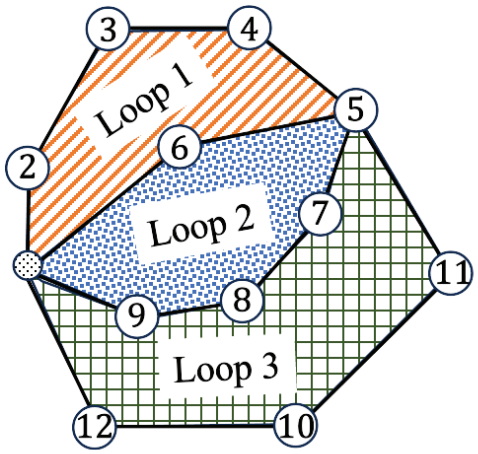

Figure 4. Graph network representation of the eight-bar linkage.

Fundamentals of Numerical Solution to the Vector Loop Equation

In this section, the steps in developing a computational solution of **Equation 6** (and similar expansion of **Equations 4** and **5**) are explained. It is assumed that the initial angular and linear positions of the links are obtained using graphical or analytical linkage synthesis methods and that the initial angular velocity and angular position of the input crank, l_2, are known.

Time values are stored in an array, e.g., `t=0:del_t:t_end`, where the variables for time interval (`del_t`) and final time (`t_end`) are previously defined. The amplitudes and angles of the vectors are stored in two arrays with the number of rows equal to the number of time points and the number of columns equal to the number of links, e.g., `L=zeros(length(t),8)` and `theta=zeros(length(t),8)`. Arrays of the same order are defined to store the velocities and accelerations. The first row of these arrays corresponds to the initial condition of the linkage at $t = 0$.

The time array is used to calculate the angular velocity and position of the input crank for all time points using a known angular acceleration, α_2. In case of a constant angular acceleration, the angular velocity and position of the input crank are stored in the first columns of their arrays using

```
alpha(:,1)=alpha_2;
omega(:,1)=alpha_2*t+omega_2i;
theta(:,1)=0.5*alpha_2*t.^2+omega(:,1).*t+theta_2i;
```

in which `omega_2i` and `theta_2i` represent variables for the initial angular velocity and position of l_2.

Since the amplitudes of the vectors representing rotating rigid rods are constant, their values are stored in all the rows of the corresponding columns of the amplitude array. Here, the constant lengths of links l_2, l_3, l_5, and l_7 are stored in the first, second, fourth, and sixth columns of the magnitude array (e.g., `L(:,1) = a;` where `a` is determined through linkage synthesis.)

Similarly, the angles of the vectors representing the sliding links are constant and can be stored in the angle array. In this case, the links l_4, l_6, and l_8 are sliders, and their angles are stored in the third, fifth, and sixth columns (e.g., `theta(:,3)=pi;`). The constant angle of the vector $\overrightarrow{EB}$ is stored in all the rows of the eighth column of this array.

The remaining elements of the arrays mentioned above are determined through an iterative numerical method which requires an initial guess as the solution for the unknowns. Here, the six unknowns in **Equation 6** are the angular positions of l_3, l_5, and l_7, and the linear positions of l_4, l_6, and l_8. Hence, the values obtained for these variables in linkage synthesis are used as the initial guess to facilitate a quick convergence for the position analysis solver.

```
pos_ini = [theta(1,2) L(1,3) theta(1,4) L(1,5) theta(1,6) L(1,7)];
```

After the successful completion of the position analysis, an initial guess for velocity analysis is made using the forward-difference method.

```
vel_ini = [(theta(2,2)-theta(1,2))/del_t (L(2,3)-L(1,3))/del_t ...];
```

Similarly, an initial guess for acceleration analysis is generated via numerical differentiation of velocity using the forward-difference method.

```
acl_ini = [(omega(2,2)-omega(1,2))/del_t (V(2,3)-V(1,3))/del_t ...];
```

Position analysis is performed, first, in a *for loop* with as many iterations as the number of time points as explained here (`for cnt=1:length(t)`). An anonymous function corresponding to the system of equations for position analysis, such as Eqn. 6, must be defined within the *for loop*. The anonymous function takes one input, which is an array with the same number of elements as the number of unknowns (six here.) The key to setting up the anonymous function properly is to represent the unknowns of the problem correctly. For instance, the complex notations of the position vectors for rotating rigid rods, such as l_3 ($l_3 e^{j\theta_3}$), and sliders, such as l_4 ($ce^{j\pi}$), are as follows.

```
L(cnt,2)*exp(x(1)*1i)
x(2)*exp(1i*pi)
```

Here, `x` is the input variable for the anonymous function whose first and second elements represent the angular position of l_3 and linear position of l_4, respectively. Consequently, the anonymous function corresponding to **Equation 6** for position analysis is constructed as shown below.

```
pos = @(x) [real(L(cnt,1)*exp(theta(cnt,1)*1i)+ ...
    L(cnt,2)*exp(x(1)*1i)+ ...
    x(2)*exp(theta(cnt,3)*1i));
      imag(L(cnt,1)*exp(theta(cnt,1)*1i)+ ...
    L(cnt,2)*exp(x(1)*1i)+ ...
    x(2)*exp(theta(cnt,3)*1i));
      real(L(cnt,4)*exp(x(3)*1i)+ ...
    x(4)*exp(theta(cnt,5)*1i)+ ...
    (h-x(2))*exp(theta(cnt,8)*1i));
      imag(L(cnt,4)*exp(x(3)*1i)+ ...
    x(4)*exp(theta(cnt,5)*1i)+ ...
    (h-x(2))*exp(theta(cnt,8)*1i));
      real(L(cnt,6)*exp(x(5)*1i)+ ...
    x(6)*exp(theta(cnt,7)*1i)+ ...
    (h-x(2))*exp(theta(cnt,8)*1i));
      imag(L(cnt,6)*exp(x(5)*1i)+ ...
    x(6)*exp(theta(cnt,7)*1i)+ ...
    (h-x(2))*exp(theta(cnt,8)*1i))];
```

This anonymous function and the initial guess are used as inputs of the *fsolve* command in MATLAB to compute the unknowns at every iteration of the *for loop*. While the initial guess may remain unchanged, updating it with the solution from the previous iteration helps with a quick convergence of the *fsolve* command.

```
if cnt > 1
   pos_ini = sol;
end
sol = fsolve(pos,pos_ini);
```

Here, `sol` is an array of six elements storing the solution of the numerical solver. At the end of each iteration, this variable is used to update the arrays that were defined for angular and linear positions of links.

```
theta(cnt,2) = sol(1);
L(cnt,3) = sol(2);
L(cnt,8) = h - sol(2);
theta(cnt,4) = sol(3);
L(cnt,5) = sol(4);
theta(cnt,6) = sol(5);
L(cnt,7) = sol(6);
```

Similar to the position analysis, an anonymous function corresponding to the expanded format of **Equation 4** is defined inside a *for loop*. The angular velocities of rotating rods and linear velocities of the sliders are unknown. All the position data are calculated in the previous step. The anonymous function for the velocities of the example here is shown below.

```
vel = @(x) [real(L(cnt,1)*omega(cnt,1)*exp(theta(cnt,1)*1i)*1i+ ...
  L(cnt,2)*x(1)*exp(theta(cnt,2)*1i)*1i+ ...
  x(2)*exp(theta(cnt,3)*1i));
    imag(L(cnt,1)*omega(cnt,1)*exp(theta(cnt,1)*1i)*1i+ ...
  L(cnt,2)*x(1)*exp(theta(cnt,2)*1i)*1i+ ...
  x(2)*exp(theta(cnt,3)*1i));
    ... % Continues like the position vector
```

Again, updating the initial guess solution at every iteration of the solution to facilitate a quick convergence is reasonable. The initial guess and the anonymous function are provided as inputs to the *fsolve* command. The results obtained from this command are used to update the corresponding elements of the velocity arrays similar to the position analysis.

An anonymous representing the expanded acceleration equations such as **Equation 5**, shown below, is used inside a *for loop* similar to position and velocity analysis to compute all the accelerations in successive iterations.

```
acl = @(x) [real(-L(cnt,1)*omega(cnt,1)^2*exp(theta(cnt,1)*1i)+ ...
  L(cnt,1)*alpha(cnt,1)*exp(theta(cnt,1)*1i)+ ...
  L(cnt,2)*x(1)*exp(theta(cnt,2)*1i)*1i- ...
  L(cnt,2)*omega(cnt,2)^2*exp(theta(cnt,2)*1i)+ ...
  x(2)*exp(theta(cnt,3)*1i));
    imag(-L(cnt,1)*omega(cnt,1)^2*exp(theta(cnt,1)*1i)+ ...
  L(cnt,1)*alpha(cnt,1)*exp(theta(cnt,1)*1i)+ ...
  L(cnt,2)*x(1)*exp(theta(cnt,2)*1i)*1i- ...
  L(cnt,2)*omega(cnt,2)^2*exp(theta(cnt,2)*1i)+ ...
  x(2)*exp(theta(cnt,3)*1i));
  ... % Continues like the position vector
```

The initial guess solution for acceleration is updated in each iteration and is provided as an input for the *fsolve* function. The output of this function is used to populate the acceleration arrays similar to the position analysis.

Algorithm for Numerical Solution

A generalized form of the anonymous functions is used to define the vectors of each link. Each link has four attributes. The first attribute is called "*Type*" and determines if the link is a rotational rod, slider, etc. The magnitude and angle of the vector for each link are used to define "*Magnitude*" and "*Angle*" attributes of the link. Then, the anonymous function is attributed to the link, called "*Fun*", which depends on the link type. This is demonstrated in the following snippet.

```
euler = @(R, theta) R*exp(1i*theta);
Vector = @(R, theta) [real(euler(R,theta)); imag(euler(R,theta))];
for l = 1:length(Link)
  ang = Link(l).Angle;
  mag = Link(l).Magnitude;
  if Link(l).Type == 0 % Rigid
      Link(l).Fun = @(x) Vector(mag,and);
  elseif Link(l).Type == 1 % Rotating Bar
      Link(l).Fun = @(x) Vector(mag,x);
  elseif Link(l).Type == 2 % Slider
      Link(l).Fun = @(x) Vector(x,mag);
  else
      % reserved for other link types
  end
end
```

In a *for loop*, the anonymous functions are summed for their real and imaginary parts in separate vector rows, resulting in a new anonymous function that is compatible with MATLAB's *fsolve* function. Like before, the *fsolve* function is called repeatedly for each time step, using results from the previous step for initial guesses, as explained in the previous section. The automatic method of generating the "*anonymous function*" is shown in the snippet below.

```
F = @(x) 0;
for l = Order{c}
  F = @(x) F(x) + Direction{c}(l) * Link(l).Fun(x(l));
end
```

Besides being shorter, the primary benefit of this approach is its ability to be generalized. The same code used to solve the eight-bar of this example linkage would also solve any other 1-DOF multi-bar linkages.

DISCUSSION

The results from the numerical algorithm are verified using the algebraic solution of the relative motion equations for interconnected rigid bodies and applying the motion constraints at each joint. For instance, in the present example, $v_{By} = v_{Cx} = v_{Dx} = 0$ and $a_{By} = a_{Cx} = a_{Dx} = 0$. The lengths and angles of the links for the eight-bar linkage shown in **Figure 1** at $t = 0$ are given in **Table 1**.

Kinematic analysis of the linkage is performed for $0 \leq t \leq 2.35$ seconds with an interval of 0.05 seconds. This ensures the input link completes at least one cycle. The angular acceleration of the input crank, l_2, is zero, and its initial angular velocity is three rad/s.

Link	Length (m)	Angle (rad)
l_2	a = 3	$\theta_2 = \pi/2$
l_3	b = 5	$\theta_3 = 5.6397$
l_4	c = 4	$\theta_4 = \pi$
l_5	d = 6.15	$\theta_5 = 0.8627$
l_6	k = 4.6715	$\theta_6 = 3\pi/2$
l_7	m = 6.15	$\theta_7 = 5.4205$
l_8	g = 4.6715	$\theta_8 = \pi/2$

Table 1. Initial values for lengths and angles of the links of the eight-bar linkage.

Angular positions, velocities, and accelerations of links l_3, l_5, and l_7 are shown in **Figure 5**. An excellent agreement between the results from the numerical algorithm and the algebraic analytical solution is observed. As expected, the kinematic parameters of links l_5 and l_7 are vertically symmetrical.

Figure 6 also shows that the results for displacements, velocities, and accelerations for sliders l_6 and l_8 agree with those derived from the analytical solution. Due to the setup of the eight-bar linkage in this example, the profiles of the kinematic parameters of these links are also vertically symmetric.

It is observed in **Figure 7** that the results for the horizontal slider l_4 obtained from the numerical algorithm match well with the analytical solution. The deviation between the analytical and numerical results shown in **Figures 5-7** is much smaller than 1% for all parameters. The largest difference may occur in reporting zero values, which is caused by rounding errors in the numerical method showing very small values such as 10^{-16} instead of absolute zero. Such an excellent agreement partially stems from choosing appropriate initial guess solutions for the numerical method in each step of the iterations, as explained earlier.

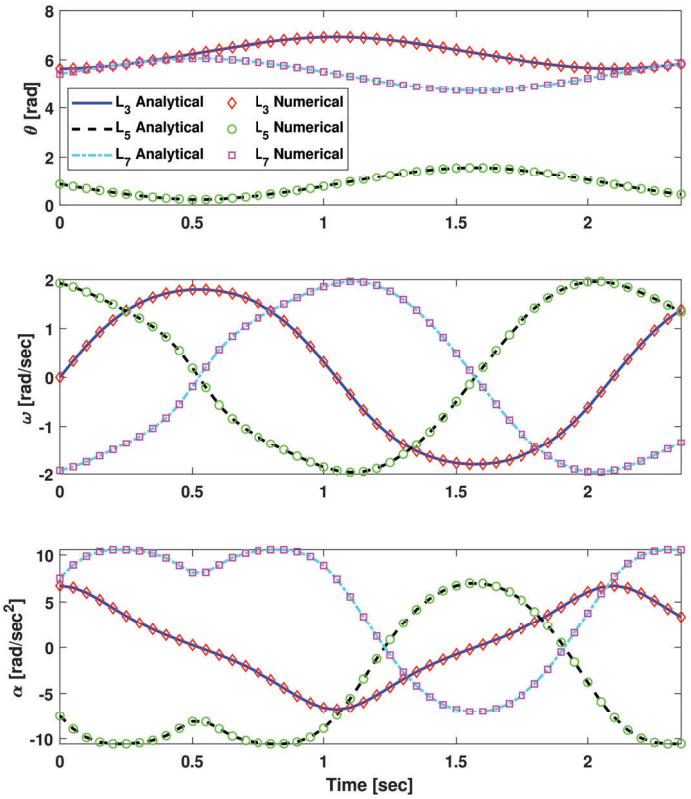

Figure 5. (Top) Angular positions, (Middle) angular velocities, and (Bottom) angular accelerations of links l_3, l_5, and l_7.

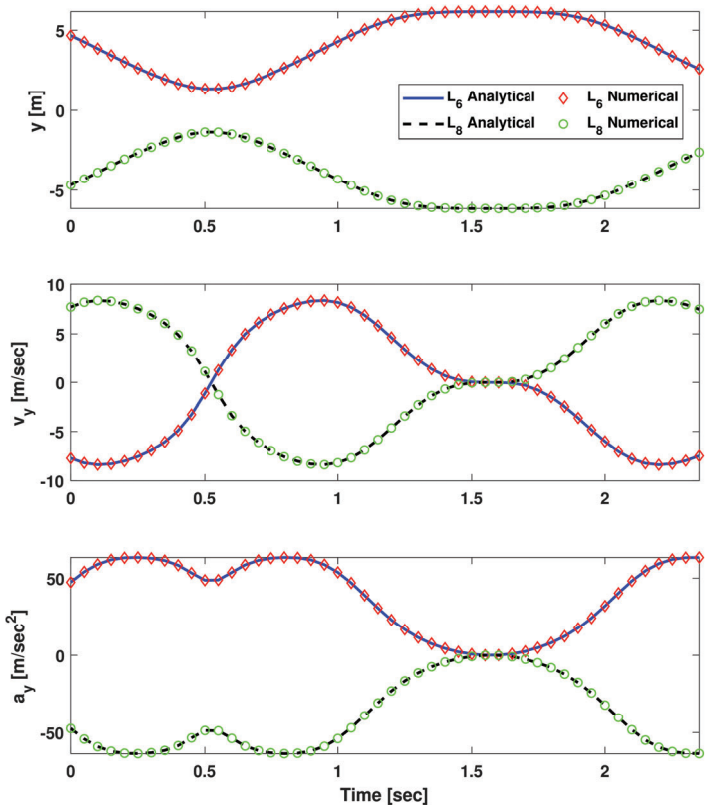

Figure 6. (Top) Positions, (Middle) velocities, and (Bottom) accelerations of sliders l_6, and l_8.

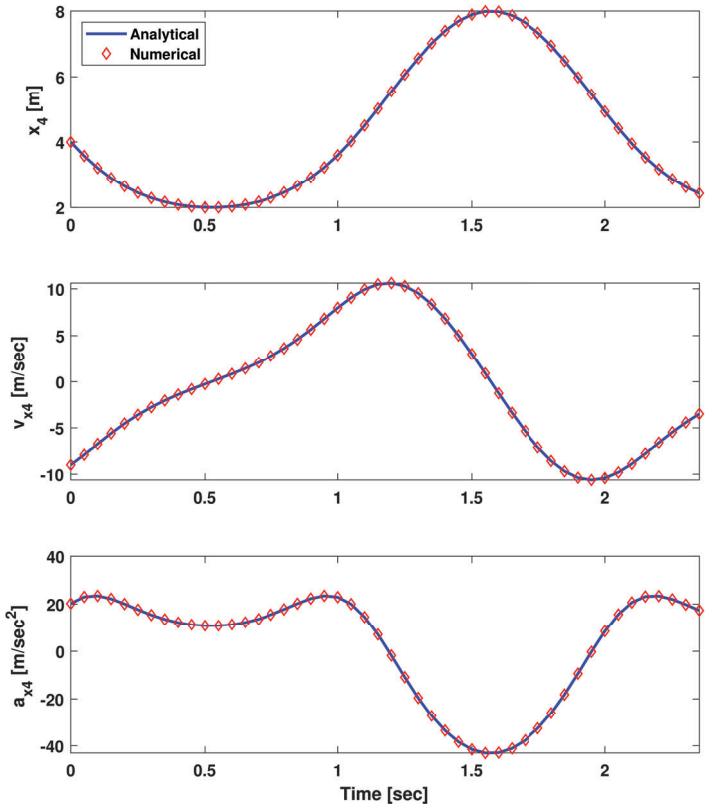

Figure 7. (Top) Position, (Middle) velocity, and (Bottom) acceleration of slider l_4.

It must be noted that the goal here is to verify the accuracy of the results from the presented numerical algorithm when compared with the algebraic solution. The excellent match between the results derived from the two methods verifies the accuracy of the numerical solution. This algorithm was tested with several other 1-DOF multi-bar linkages and exact matches were observed between the numerical and algebraic solutions.

CONCLUSIONS

An algorithm was presented for numerical analysis of the kinematics of multi-bar linkages using computer simulations in MATLAB. The algorithm is based on developing the vector loop equations for the linkages, as in an analytical solution. The vector loop equations can be generated either manually or by generating a path-finding algorithm. The numerical solution benefits from the capabilities of scientific programming languages, such as MATLAB, to find the roots of systems of nonlinear and linear equations. It was demonstrated that the presented algorithm can handle links of any order (binary, ternary, etc.) as well as links whose kinematics are related to that of an adjacent link. The results obtained using the presented algorithm for an eight-bar linkage were in excellent agreement with the analytical solution.

The method presented here can be used in teaching relevant courses such as kinematics or dynamics of machines. The authors are currently developing a complete stand-alone program with a graphical user interface for kinematic analysis of mechanisms. The next step is to include force calculations to provide a complete dynamic analysis of any linkage mainly for educational purposed.

The algorithm can be further improved to capture errors that may occur if the mechanism is physically incapable of completing a full cycle. For example, if a non-Grashof four-bar linkage is pushed beyond its limits, the solver will produce an error. Another enhancement to the method is to enable the program to analyze linkages with inverse sliders.

ACKNOWLEDGEMENTS

The authors thank the Office of Research, Scholarship, and Creative Activities at SUNY New Paltz.

REFERENCES

1. Moon, F. C., (2003), Franz Reuleaux: Contributions to 19th Century Kinematics and Theory of Machines, *Applied Mechanics Review*, **56**(2), 261–285, *https://doi.org/10.1115/1.1523427*
2. Thurston, R. H., (1878), The Steam-Engine As A Simple Machine, in *A History of the Growth of the Steam-Engine*, Vol. 24, 1–20, D. Appleton And Company, New York.
3. The Editors of Encyclopedia, (2023), Steam Engine, December 2023 URL *https://www.britannica.com/technology/steam-engine* (accessed January 2024)
4. Norton, R. L., (1998), Position Analysis, in*Design of Machinery*, 5th ed., 174–214, McGraw-Hill, Massachusetts.
5. Reuleaux, F., (2013), Sketch of the History of Machine Development, in *The Kinematics of Machinery: Outlines of a Theory of Machines*, 1st ed., 201–242, Courier Corporation, New York.
6. Reuleaux, F., (1901), The Elements of Graphostatics, in *The Constructor*, 4th ed., 22–38, HH Suplee, New York.
7. Kennedy, A. B. W., (1907), The Machine, in *The Mechanics of Machinery*, 1st ed., 1–19, Macmillan and Company, New York.
8. Hartmann, W., (1913), Introduction, in *Die Maschinengetriebe*, 1st ed., 1–2, Macmillan and Company, New York.
9. Grübler, M. F., (1917), Die Elementenpaare, in *Getriebelehre: eine Theorie des Zwanglaufes und der ebenen Mechanismen*, 1st ed., 1–7, Springer, Berlin.
10. Uicker Jr, J. J., Pennock, G. R., and Shigley, J. E., (2023), The World of Mechanisms, in *Theory of machines and mechanisms*, 6th ed., 3–54, Cambridge University Press, Cambridge.
11. Martin, G. H., (2002), Linkages, in *Kinematics and dynamics of machines*, 2nd ed., 44–68, Waveland Press, Illinois.
12. Russell, K., Shen, J. Q., and Sodhi, R., (2018), Mathematical Concepts in Kinematics, in *Kinematics and Dynamics of Mechanical Systems: Implementation in MATLAB® and SimMechanics®*, 2nd ed., 13–37, CRC Press, New York.

13. Vinogradov, O., (2000), Kinematic Analysis of Mechhanisms, in *Fundamentals of kinematics and dynamics of machines and mechanisms*, 1st ed., 15–64, CRC Press, New York.
14. Gamble, B., (2010), Software Implementation, in 5-Bar Linkage Kinematic Solver and Simulator, 1st ed., 18–23, The University of Vermont, Vermont.

ABOUT THE STUDENT AUTHOR

Brandon Torres is a Class of 2024 Mechanical Engineering undergraduate student at the State University of New York (SUNY) at New Paltz. His area of interest for research is computational engineering and programming. Brandon has been working on projects that involve modeling physical systems with MATLAB, such as fluid motion and machine dynamics problems. Brandon has also been working as a student assistant for Computer Simulations in SUNY New Paltz, hoping to teach others about the benefits of using computer programming to solve real-world engineering problems.

PRESS SUMMARY

Machines and mechanisms are all around us, from door closers, to can openers, to car engines, to aircraft landing gear, and so on. These are a tools made up of interconnected links (e.g. rods) that transfer motion and force from one source and/or form to another point and/or form. For example, the energy stored in the spring component of a door closer is converted into the kinetic energy that moves the door back to its closing state.

Understanding the behavior of these mechanisms, such as the speed at which their components move, requires a dynamic analysis of the interconnected links. Although algebraic methods from classical dynamics can be used for this purpose, the process becomes tedious as the number of components in the mechanism increases. The increasing number of links in the mechanisms makes the derivation of specific equations for any given design virtually impossible. There is a lack of a general solution method for the dynamic analysis of mechanisms of any design that can be used for education purposes.

Thanks to the advancement of computational technologies, the authors have developed a numerical method that employs mathematical concepts such as graph theory and complex numbers to perform a complete kinematic analysis of mechanisms. The proposed method is capable of analyzing mechanisms with several components, and can be taught in classroom as a means to study the motions of mechanisms for which analytical solutions are challenging.