TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 69, Issue I, March, 2026

A CLASSIFICATION SYSTEM FOR THE FOUR-BAR LINKAGE MECHANISM TRAJECTORIES

Tiberiu Alexandru ANTAL

Abstract: *The paper aims to provide a way to classify trajectories generated the four-bar linkage mechanism. Essentially, it attempts to answer the question: "If the crank of the mechanism is rotated, does the point where the motion is collected move in a straight line, a circle, or a parabola?" The classification system works by treating the trajectory as a set of statistical data and performs regressions to find the geometric model with the smallest error.*
Key words: *AI, classification, mechanism, Python, trajectory.*

1. INTRODUCTION

In AI, classification is a type of supervised learning where the model is trained on a labeled dataset. The goal is to predict the "class" or category of new, unseen data points. Classification is a way of sorting things into groups based on how they are similar. The goal is to look at a single item and decide which category/class it belongs in. A classification system organizes items into specific categories based on shared distinguishing qualities or characteristics or relationships. Its main purpose is to evaluate an input and place it into a matching, predetermined group. Before AI, classification systems were static. If you wanted to add a new category or handle more complex data, you had to rewrite the rules. AI makes these systems dynamic - they improve as they see more data, becoming more accurate and adaptable over time. At first glance, AI (especially modern Machine Learning) and regression (a classic statistical technique) might seem like separate worlds. However, they are deeply and fundamentally connected as regression is one of the elementary building blocks of AI and Machine Learning as it can be used to analyze evolution. Classification constitutes a foundational cognitive and computational framework for organizing information. By categorizing phenomena through manual heuristics or algorithmic models, it translates the detection of patterns into a structured basis for systemic decision-making.

2. MATHEMATICAL MODEL AND PROGRAMMING OF THE SIMULATION

Analyzing the trajectory of a four-bar linkage - specifically the path traced by a point on the coupler link or an extension of the coupler - is a classic problem in kinematics known as path generation [1], [2], [5]. To model these trajectories (coupler curves) the code relies on vector equations and coordinate geometry implemented in the Python programming language using the object oriented paradigm presented in [3] and [4]. This class represents the individual links of the mechanism. It handles the kinematics (the movement). The `SolveInt()` method calculates where two links (circles) intersect. Since a four-bar linkage is essentially two bars rotating around fixed pivots and meeting at a moving point, this method finds that moving point. The `rotateP1()` simulates the input crank turning and `record_trajectory_point()` it saves the (x,y) position of the coupler point (v4) into a list.

3. TRAJECTORY CLASSIFICATION MATHEMATICS

Received: 29.01.26; **Similarities**: 17.02.26: **Reviewed**:16.02./28.02.26: **Accepted** 12.03.26.

In the context the path generation the "trajectory" is a set of *n* points (x_i, y_i) generated by the mechanism's extension of the coupler link that converts simple rotation into complex motion. The trajectory is mathematically defined as a sextic curve (a sixth-order algebraic curve). Despite the complexity this can be traced to some more simple functional categories under certain circumstances:

- **Cusp Trajectories**: The trajectory is close to a teardrop or a path with a sharp "V" (cusp). It is used to pick up an object at a specific point and moving it away. The cusp represents a momentary zero-velocity point in certain directions.
- **"D" Shapes or Flat-Top Curves**: The trajectory contains a nearly straight-line segment. It is used in mechnisma like Hoekens or Chebyshev linkages. It's used in cranes or walking mechanisms where a foot must stay flat against the ground.
- **Figure - Eights**: The trajectory intersects itself, forming two loops. It is used in complex timing mechanisms where a component needs to pass through the same spatial coordinate twice during a single input cycle.
- **Ovals and Kidneys**: The trajectory is smooth forming non-intersecting closed loops. Used in agitators or mixers where continuous, smooth sweeping motion is required without abrupt changes in acceleration.

In mechanical engineering, we rarely need a point to move in a sextic curve. We usually need a machine to do something simple: push a box in a straight **line**, hold a door open at a specific **arc**, or mimic a natural movement. Approximating these simple shapes is important for three main reasons: cost, complexity, and control. Form a mathematical view we are interested to approximate the trajectory with simple geometric primitives (lines, arcs, or parabolas) to provide a classification of the mechanism trajectories based on elementary shapes that can be associated with simple actions.

3.1 Linear Fit Test

The goal is to find the slope (m) and intercept (b) of the $y = mx + b$ line that minimizes the vertical distance between the actual points and the line with:

$$m = \frac{n\sum_{i=1}^{n}(x_i y_i) - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \quad (1)$$

$$b = \frac{\sum_{i=1}^{n} y_i - m\sum_{i=1}^{n} x_i}{n} \quad (2)$$

This approach calculates the "best" line by minimizing the sum of the squared vertical distances (residuals) between each point on your coupler curve and the proposed line. R-squared (R^2) or the goodness of fit calculation is made following the steps:

- For every x_i in the path we calculate the predicted position $\hat{y}_i$ using your new line $\hat{y}_i = mx_i + b$;
- We calculate the errors:
  - Residual Sum of Squares (SS_{res}): The actual deviation of the linkage from the straight line is calculated by $SS_{res} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$;
  - Total Sum of Squares (SS_{tot}): The total spread of the linkage's vertical movement relative to its average height ($\bar{y}_i$) computed as $SS_{tot} = \sum_{i=1}^{n}(y_i - \bar{y}_i)^2$;
  - $R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$

A metric measures how well a specific task it is performing. It is the number we look at to make a selection between several solutions. With the help of *m* and R^2 we introduce linearity metric as described in Table 1.

Table1

The linearity metric table

Metric	What it tells	Mechanism context
m (Slope)	The direction of the "straight" path	Tells you the angle of the output motion
R^2	The "straightness" quality.	Tells how much the "straight" line vibrates or bows."

Based on Table 1 abstract concepts like "straightness" or "smoothness" turn into a number that we can compare, optimize, or use to make a "pass/fail" decision. Although the numerical part seems complicated, the implementation in Python is very simple because there are already library functions that perform the operations described like:

```
# Residual sum of squares
ss_res = np.sum((y - y_pred) ** 2)

# Total sum of squares
ss_tot = np.sum((y - np.mean(y)) ** 2)

r_squared = 1 - (ss_res / ss_tot) if
ss_tot != 0 else 0
```

3.2 Circular/Arc Fit Test

The goal is to find if the trajectory points follow a circular path given by the equation $(x-a)^2+(y-b)^2=r^2$ where (a, b) are the center coordinates and r is the radius. For this we rewrite the equation as:

$$x^2+y^2=2a\,x+2\,b\,y+c \tag{3}$$

where

$$c=r^2-a^2-b^2 \tag{4}$$

Because the linkage doesn't trace a perfect circle (it's a sextic curve), no single set of a, b, c will fit every point perfectly. Every point will have a tiny bit of error as:

$$E_i=2a\,x_i+2b\,y_i+c-(x_i^2+y_i^2) \tag{5}$$

To find the "best" fit, we want to minimize the Sum of Squared Errors (S). We square them so that negative errors don't cancel out positive ones:

$$S=\sum_{i=1}^{n}[2a\,x_i+2b\,y_i+c-(x_i^2+y_i^2)]^2 \tag{6}$$

To find the minimum of S, we use calculus. We take the partial derivative of S with respect to our three unknowns (a, b, c) and set them to zero. We get the following set of equations for a (the x center - (7), b the y center - (8) and for the c constant - (9)):

$$\sum_{i=1}^{n}2x_i(2a\,x_i+2\,b\,y_i+c)=\sum_{i=1}^{n}2x_i(x_i^2+y_i^2) \tag{7}$$

$$\sum_{i=1}^{n}2y_i(2a\,x_i+2\,b\,y_i+c)=\sum_{i=1}^{n}2y_i(x_i^2+y_i^2) \tag{8}$$

$$\sum_{i=1}^{n}(2a\,x_i+2\,b\,y_i+c)=\sum_{i=1}^{n}(x_i^2+y_i^2) \tag{9}$$

By solving these three simultaneous equations (which can be done with basic substitution or elimination), we find the values for a, b, and c that represent the average center of that trajectory segment, and the value of r from (4). Again the numerical part is greatly simplified by the functions implemented in the Python libraries, which can directly implement the sequence:

```
z = x**2 + y**2
A = np.column_stack([2*x, 2*y,
np.ones_like(x)])

# Solve linear system: A·[a, b, c]ᵀ = z
result = np.linalg.lstsq(A, z,
rcond=None)[0]

# Extract parameters
a, b, c = result
r = np.sqrt(a**2 + b**2 + c)

# Calculate R-squared (goodness of fit)
    distances = np.sqrt((x - a)**2 + (y -
b)**2)
residuals = distances - r
ss_res = np.sum(residuals**2)
ss_tot = np.sum((distances -
np.mean(distances))**2)
r_squared = 1 - (ss_res / ss_tot) if
ss_tot != 0 else 0
```

3.3 Quadratic Fit Test

The goal is to find if the trajectory points follow a parabolic path $y=ax^2+bx+c$

where a, b, and c coefficients are calculated using polynomial regression. The test is started by creating a design matrix with three columns:

- Column 1: x² values (for the quadratic term)
- Column 2: x values (for the linear term)
- Column 3: All ones (for the constant term)

The algorithm solves the linear least squares problem, or it finds the coefficients [a, b, c] that minimize: $\sum_{i=1}^{n}[y_i - (a\,x_i^2 + b\,x_i + c)]^2$ This is done using linear algebra (`np.linalg.lstsq`), which finds the optimal coefficients that make the predicted parabola pass as close as possible to all points. After obtaining the coefficients, the algorithm:

- Predicts y-values for each x using the fitted equation: `y_pred=a·x²+b·x+c`
- Calculates residuals: differences between actual y and predicted y
- Computes R-squared.

```
x = points[:, 0]
y = points[:, 1]

# Create design matrix for quadratic regression: y = a·x² + b·x + c
A = np.column_stack([x**2, x, np.ones_like(x)])

# Solve using least squares: A·[a, b, c]ᵀ = y
coeff = np.linalg.lstsq(A, y, rcond=None)[0]

# Extract coefficients
a, b, c = coeff

# Calculate predicted y values
y_pred = A @ coeff
# Same as: a*x² + b*x + c

# Calculate R-squared
ss_res = np.sum((y - y_pred) ** 2)

# Sum of squared residuals
ss_tot = np.sum((y - np.mean(y)) ** 2)

# Total sum of squares
r_squared = 1 - (ss_res / ss_tot) if ss_tot != 0 else 0
```

The goodness of fit metric is explained by R^2 as:

- $R^2 = 1$: Perfect fit (model explains 100% of variation);
- $R^2 = 0$: Model explains none of the variation (no better than using the mean);
- R^2 between 0 and 1: Percentage of variation explained by the model.

3.4 Curvature Analysis

Curvature (κ) measures how sharply a curve bends at a point. It's the reciprocal of the radius of the osculating circle (the circle that best approximates the curve at that point). To measure how much the trajectory bends locally we consider three consecutive points P_0, P_1, and P_2. The area A of the triangle with the given vertices is:

$$A = 0.5[x_0(y_1 - y_2) + x_1(y_2 - y_0) + x_2(y_0 - y_1)] \quad (10)$$

The side lengths can be calculates as: $a = |P_0 - P_1|$, $b = |P_1 - P_2|$, and $c = |P_2 - P_0|$ and the radius of circumcircle as: $R = (a\,b\,c)/(4\,A)$ and the curvature as: $\kappa = 1/R$.

The curvature is important because a low variation (<0.1) suggests constant curvature (line or perfect circle), while a high variation suggests changing curvature (complex curves). In this context it helps distinguish between arcs and lines.

3.5 Classification Logic

The classification logic is a hierarchical, comparative decision-making system that determines trajectory type based on multiple criteria by:

- Test all models: Linear, circular, quadratic;
- Sort by R-squared: Best fit gets top priority;
- Curvature validation: Override if arc has very low curvature variation;
- Minimum points check: Needs at least 20 points for reliable classification.

4. RESULTS AND CONCLUSIONS

The classification logic follows a best approximate match approach:

- Let statistical fits compete (R^2 comparison);
- Apply physical reasoning (curvature validation override);
- Quantify uncertainty (confidence with curvature factor);
- Handle edge cases (minimum points, numerical failures).

The code is designed to be practical for engineering applications. The curvature check recognizes that in mechanical systems, a "circle with radius approaching infinity" is physically indistinguishable from a line, so the simpler model (line) should be preferred. The logic works well for clear-cut cases (obvious lines, arcs, parabolas) but has limitations for ambiguous or complex trajectories common in real-world mechanisms. The classification from Figure 1 was obtained for the following configuration:

```
v1 = Vector2D("a", 0, 0, -2, 0)
v2 = Vector2D("b", -2, 0, 2, 3.5)
v3 = Vector2D("c", 4, 0, 2, 3.5)
v4 = Vector2D("d", v2.midpoint[0],
v2.midpoint[1], 3.5, 6.5)
v4.setBase(v2)
```

with the φ-φ_{start} in [0^0, 360^0] with 3^0 step.

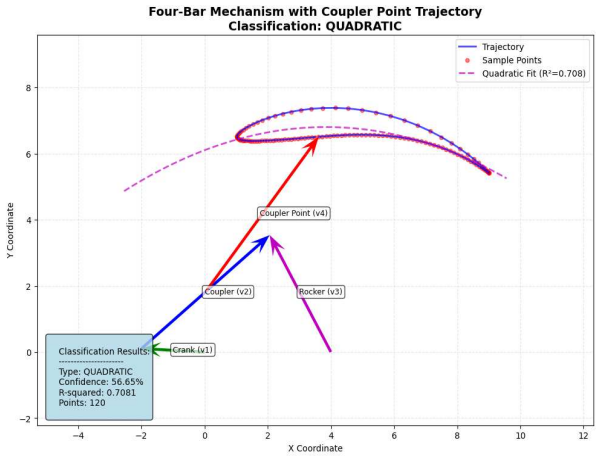

Fig. 1. - Four-bar linkage quadratic trajectory classification.

```
Type: QUADRATIC
Confidence: 56.65%
R-squared: 0.7081
Curvature Variation: 3.4813
Quadratic Equation: y = -0.047x² + 0.364x + 6.103
```

The classification from Figure 2 in obtained for φ-φ_{start} in [-90^0, 90^0] with 3^0 step.

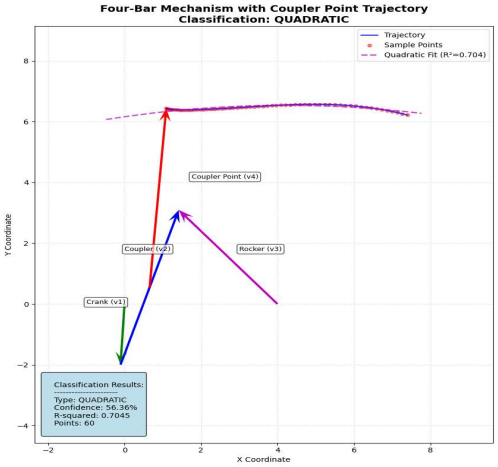

Fig. 2. - Four-bar linkage with another quadratic trajectory classification.

```
Type: QUADRATIC
Confidence: 56.36%
R-squared: 0.7045
Curvature Variation: 2.1011
Quadratic Equation: y = -0.021x² + 0.178x + 6.160
```

The classification from Figure 3 in obtained for φ-φ_{start} in [-0^0, 60^0] with 1^0 step.

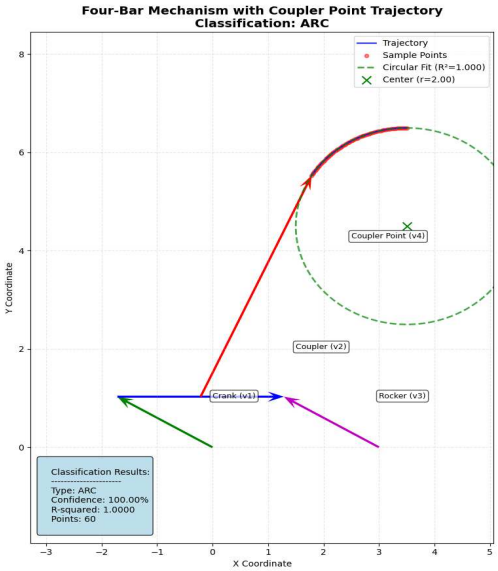

Fig. 3. - Four-bar linkage arc trajectory classification.

The classification results were obtained for the following configuration:

```
v1 = Vector2D("a", 0, 0, 0, 2)
v2 = Vector2D("b", 0, 2, 3, 2)
v3 = Vector2D("c", 3, 0, 3, 2)
```

```
v4 = Vector2D("d", v2.midpoint[0],
v2.midpoint[1], 3.5, 6.5)
```

```
Type: ARC
Confidence: 100.00%
R-squared: 1.0000
Curvature Variation: 0.0000
Circle Center: (3.500, 4.500)
Circle Radius: 2.000
```

Python excels at classification tasks compared to C and C++ for several interconnected reasons:

- **Data science ecosystem**: Python dominates machine learning with comprehensive libraries. Scikit-learn provides 200+ algorithms with consistent APIs. NumPy and pandas handle numerical and tabular data efficiently. Visualization libraries like Matplotlib and Seaborn enable instant feedback. These libraries have C/Fortran backends for performance but Python interfaces for simplicity;
- **Interactive development**: Python supports interactive computing through Jupyter notebooks and REPL environments. Data scientists can test multiple algorithms quickly, visualize results immediately, and iterate rapidly. The language's expressiveness allows complex operations in few lines of code. This accelerates the trial-and-error nature of model development.
- **Abstraction level:** Python operates at a higher abstraction level, letting developers focus on algorithms rather than implementation details. Memory management is automatic, unlike C/C++'s manual allocation. The language hides low-level complexities like pointers and memory addresses that distract from the classification logic.

5. REFERENCES

[1] ANTAL, Tiberiu Alexandru, *Python in the planar four-bar linkage mechanism simulation*. Acta Technica Napocensis - Series: Applied Mathematics, Mechanics, And Engineering**,** v. 68, n. 1 & 2, jun. 2025. ISSN 2393–2988.

[2] ANTAL, T. A., CSIBI, V., *Mechanizmus generálása SolidEdge-ben Visual Basic-ból/ Generarea mecanismelor în SolidEdge din Visual Basic / Mechanism generation in SolidEdge from Visual Basic*, Revista Műszaki Szemle, 32 szám, p.3-8, 2005, ISSN 1454-0746.

[3] ANTAL, Tiberiu Alexandru, *A machine learning approach in planar mechanism trajectory approximation*. Acta Technica Napocensis - Series: Applied Mathematics, Mechanics, And Engineering, v. 68, n. 2, 2025. ISSN 2393–2988.

[4] ANTAL, T .A., *Aspects concerning code reusabilty in planar mechanisms simulation, Acta Technica Napocensis*, Series: Applied Mathemathics and Mechanics, Nr. 48, Vol. I, 2005, p.45-50, ISSN 1221-5872.

[5] ANTAL, T .A., *An object based approach to planar mechanisms simulation, Acta Technica Napocensis*, Series: Applied Mathemathics and Mechanics, Nr. 48, Vol. I, 2005, p.37-44, ISSN 1221-5872.

Un sistem de clasificare pentru traiectoriile mecanismului bielă manivelă

Lucrarea își propune sa dea o modalitatea de clasificare a traiectoriilor generate de un mecanism bielă manivelă. În esență se încearcă a da un răspuns la întrebarea: „Dacă se realizează rotirea manivelei din mecanism punctul în care se culege mișcarea se mișcă se mișcă pe o linie dreaptă, pe un cerc sau pe o parabolă?” Sistemul de clasificare funcționează prin tratarea traiectoriei ca o mulțime de date statistice și realizează regresii pentru a găsi modelul geometric cu eroarea cea mai mică.

ANTAL Tiberiu Alexandru, Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanical System Engineering, antaljr@mail.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.