Revista Brasileira de Ensino de Física, vol. 47, e20250401 (2025)
www.scielo.br/rbef
DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

Roullete curves, coin paradox and Aristotle's wheel paradox

Osvaldo L. Santos-Pereira*[1]

[1]Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Received on September 05, 2025. Revised on October 18, 2025. Accepted on October 30, 2025.

This work discusses the concept of roulette, the generated curves that occur when one curve rolls without slipping along another, tracing the path of a fixed point. The coin paradox and Aristotle's wheel paradox are used as pedagogical motivations to discuss the parametric equations of epicycloids and hypocycloids, providing a geometrical intuition for the mathematical derivations and computational implementation of those curves. Python code is provided as motivation to apply the derived parametric equations, resulting in concrete visualizations and animations.

Keywords: Roullete, Aristotle, paradox, coin, wheel, epicycloid, hypocycloid.

1. Introduction

Roulettes are generated when one curve rolls without slipping along another, tracing the path of a fixed point. Classic examples include cycloids, epicycloids, hypocycloids, and cardioids [1–7]. Despite their simple construction, roulette often yields surprising and counterintuitive results.

Two famous cases illustrate how misconceptions can arise from the analysis of roulette curves, examples are the coin paradox and the Aristotle wheel paradox. An epicycloid consists of an exterior circle rolling about another fixed circle, and the coin paradox was first defined as a particular case of an epicycloid: a coin rolling about another coin of equal radius completing two full turns, not one, as intuition suggests [8–10]. In Aristotle's wheel paradox, concentric wheels of different radii seem to traverse the same distance, apparently contradicting the relationship between circumference and radius [11–13].

The coin paradox has an analogue in celestial mechanics. The concept of sidereal time. A sidereal day [14] represents the time for one rotation about the planet's axis relative to the distant stars, just like the rolling coin in the paradox. An Earth year has around 365.25 solar days, but 366.25 sidereal days. As a solar day has 24 hours, a sidereal day has approximately 23 hours, 56 minutes, and 4.1 seconds.

The coin paradox is often illustrated through the epicycloid curve, traced by a rolling coin on a fixed coin of the same size. An epicycloid is a geometric curve traced by a point on a circle, an epicycle, that rolls without slipping around the outside of a fixed circle. The curve features points where it comes into contact with the fixed circle, and the number of cusps N is equal to the ratio of the radius R of the fixed circle to the radius of the rolling circle r, so $N = R/r$ when this ratio is an integer.

Abad [15] presented an interesting application of roulette curves, where he calculated the magnetic field generated by a steady current that takes the shape of two types of special curves: hypocycloids and epicycloids with a specified number of sides. The computation was performed in the center of the referred curves. He used the Biot-Savart law, and the result was shown to be general because it is obtained as a function of the number of sides of the curve and in terms of a parameter that identifies the type of curve considered.

This paper provides a pedagogical introduction to roulette curves, highlighting their mathematical formulation, physical interpretation, and educational potential. Special attention is given to epicycloids, hypocycloids, and the paradoxes of rolling motion: the coin paradox and the Aristotle wheel paradox. One of the intentions of this paper is to demonstrate how these paradoxes connect with the intuitive aspects of geometry and mechanics in the context of physics teaching.

This work is organized as follows: the second section presents and discusses the coin paradox, solving it using simple circular kinematics, which can be learned from fundamental physics textbooks [16]. The third section presents the epicycloid curves, deriving the parametric equations for the curve, and provides several examples of such curves. The fourth section introduces the hypocycloid, where a simple change of signs in r is made as an analogy to the derivation of the parametric equations for the epicycloids. Several examples of curves are given for different values of k. Section five presents and explains Aristotle's wheel paradox in the context of fundamental

*Correspondence email address: olsp@if.ufrj.br

Editor-in-Chief: Marcello Ferreira

physics [16]. Section six presents all the Python codes used in this work to generate images and animations. The last section is the conclusion of this paper.

2. Coin Paradox

The coin paradox was originally proposed by Martin Gardner in 1975 [8]. Gardner was a mathematician known for writing recreational mathematical puzzles. This apparent paradox gain notoriety in 1982 when was proposed as a question in a SAT question. The SAT is a standardized test widely used for college admissions in the United States. The problem was that there was no correct answer among the five alternatives. Someone noticed that the question could not be answered correctly and reported the error to the company that administered the SAT and there was a backlash which was widely reported in respected newspaper in the United States [17].

The coin paradox was again brought up in 2015 the streaming platform Youtube by a channel named *MindYourDecisions* with a video named "Why did everyone miss this SAT Math question?" [18] uploaded in july 5th of 2015. As of October of 2025 the video has more than three million views.

An article named "The SAT Problem That Everybody Got Wrong" was written by Jack Murtagh and edited by Jeanna Bryner and published in an article in the Scientific American Journal in June 20 of 2023 [19]. The article was about the 1982 SAT problem and the coin paradox.

In November 30th of 2023 the YouTube channel *Veritassium* also uploaded a video regarding the coin paradox and the SAT question. As of October of 2025 the Veritassium video named "The SAT Question Everyone Got Wrong" [20] has over sixteen million views. A fun fact, Dr. Doug Jungreis, who is now a researcher in mathematics and one of the people who reported the error in the SAT question in 1982 appeared in the video and was interviewd by Dr. Derek Muller the owner of the Veritassium channel.

The coin paradox is as follows: start with two identical coins on a table, both showing their heads side up and aligned. Keep coin B fixed, and roll coin A around it without slipping. When coin A reaches the opposite side, the heads are again parallel, and coin A has completed one full revolution. Continuing the motion back to the starting point adds a second revolution. Paradoxically, coin A seems to have rolled a distance twice its circumference.

Fig. 1 below illustrates one of the reasons why this paradox seems to appear in people's reasoning. The image shows five snapshots of the rolling coin on a fixed coin: the first one in the starting position, with its head in an upright position; the second snapshot, after the rolling coin has turned its head upside down; the third snapshot shows the rolling coin after completing half of the circular path with its head in an upright position again, the fourth snapshot shows the coin again with its head upside down but on the left of the fixed coin, and the final snapshot where the rolling coin returns to its starting point. In general, people will count the number of times the top of the rolling coin touches the fixed coin, imagining that this is half of the total roll.

Figure 1: Illustration of the coin rotation paradox: a coin rolling externally around another identical coin completes two full rotations before returning to its initial position. The image shows a Brazilian one real coin (1 BRL).

The explanation is that coin A's circumference equals the distance rolled, but an additional rotation arises because its path is circular rather than straight. Flattening coin B's circumference into a line shows only one actual rotation, while the second corresponds to the transport of A around B.

The center of the rolling coin follows a circular path of radius equal to the sum of both radii, so its trajectory has twice the circumference of either coin. Covering this distance without slipping forces the rolling coin to complete two full revolutions. Details of the coin's orientation or rotation direction do not affect this result.

This paradox is easily explained using simple kinematics of circular motion. Consider a circle A of radius r rolling without slipping on the outside of a fixed circle B of radius R, as shown in Fig. 2, the center of circle A traces a circular path of radius $R + r$.

Let the linear speed of the center of A be v, and its angular speed about its own center be ω. The no-slipping condition at the point of contact between the two circles implies that the linear speed of the circle A's center is

$$v = \omega r. \tag{1}$$

The time taken for the center of A to complete one full revolution around B is

$$t = \frac{2\pi(R+r)}{v} = \frac{2\pi}{\omega}\left(\frac{R}{r} + 1\right). \tag{2}$$

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

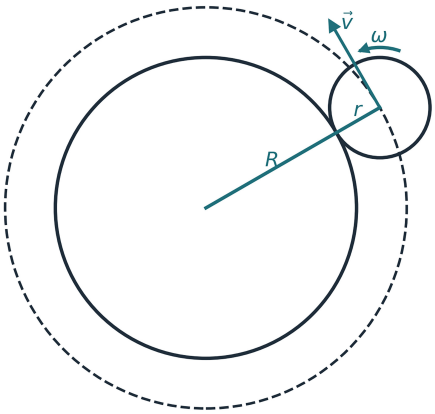

Figure 2: Geometry of a circle of radius r rolling without slipping around a fixed circle of radius R. The center of the moving circle traces a circular path of radius $R+r$, leading to $(R+r)/r$ full rotations about its own axis.

During this interval, the angular distance $\theta = \omega t$ traversed by circle A about its own center is given by the following expression

$$\theta = 2\pi\left(\frac{R}{r}+1\right). \tag{3}$$

Hence, the number of complete rotations of A about its own axis is

$$N = \frac{\theta}{2\pi} = \frac{R}{r}+1. \tag{4}$$

As a particular case of the coin paradox, if $R = r$, then

$$N = \frac{r}{r}+1 = 2, \tag{5}$$

meaning that circle A completes two full rotations about its center while rolling once around circle B.

3. Epicycloid

An epicycloid is a type of roulette curve generated by a fixed point on the circumference of a circle of radius r that rolls without slipping around the outside of a fixed circle of radius R, just like the example we explored in Sec. 2 about the coin paradox. The locus of the point traces a curve with a rich variety of shapes depending on the ratio $k = R/r$. For $k = 3$ as shown in Fig. 3, there is a trefoiloid [6]. In this example, it would be the case of a circle of radius r rolling without slipping around a circle B of radius $R = 3r$.

Epicycloids are encountered in various physical and engineering contexts. They describe the motion of planetary gears [21, 22] and caustic curves in optics [23], with applications in kinematics and computer-aided design. They also provide elegant didactic examples of how rolling motion translates into complex yet tractable geometrical trajectories.

It is straightforward to demonstrate the parametric equations of the epicycloid by analyzing the geometry

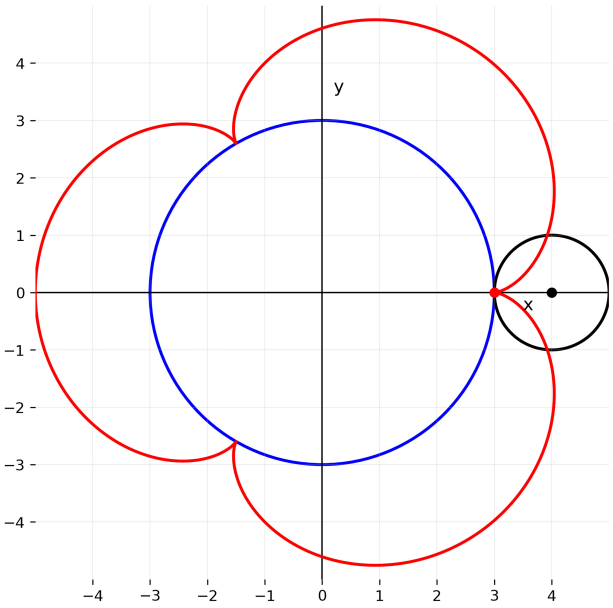

Figure 3: Example of an epicycloid generated by a circle of radius r rolling around a fixed circle of radius R. The number of cusps depends on the ratio R/r. In this figure, $R = 3r$, and the number of turns completed by the smaller circle equals 4. The name of this epicycloid is trefoiloid [6].

of rolling without slipping. Consider a circle of radius r rolling externally without slipping on a fixed circle of radius R, as illustrated in Fig. 4. Consider the origin of the coordinate system at the center of the larger circle, at $O_R(0,0)$. A point $B = (x,y)$ on the circumference of the rolling circle traces the epicycloid.

Ref. [6] provides a visual proof of the parametric equations of a general roulette (*epitrochoid*), traced by a point P attached to a circle that is rolling about a fixed circle. See Figs. 56-58 in chapter 6 of Ref. [6]. Fig. 4 was based on the derivation provided by Ref. [6], however the reader can find several other proofs on the internet, in

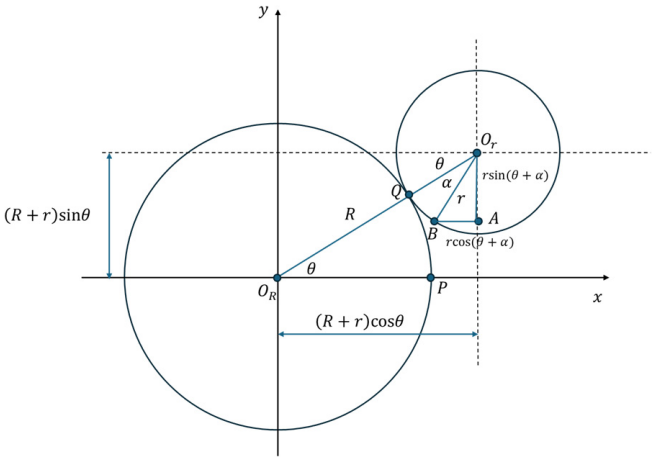

Figure 4: Construction of the epicycloid. A generating circle of radius r rolls externally around a fixed circle of radius R, and a point B on its circumference traces the epicycloid curve.

personal webpage of Professors, public archives or free online encyclopedias such as Wikipedia. It is customary for calculus and analytical geometry textbooks to leave this proof as exercise for students. It is a rather trivial exercise, but it must be stated that Fig. 4 it is not an original proof, since the study of roulette date from 1525 (Dürer) [6].

Let θ denote the angle corresponding to the arc length on the fixed circle measured from the starting tangent point of point $B_0 = (x_0, y_0)$, and let α be the angle corresponding to the arc length on the rolling circle measured from the contact point to the point B. Notice that at $\theta = 0$ the initial point is at $B_0 = (R, 0)$.

The coordinates of the point B on the rolling circle can be derived from geometric and trigonometric relations shown in Fig. 4.

$$x(\theta) = (R + r)\cos\theta - r\cos(\theta + \alpha), \tag{6}$$

$$y(\theta) = (R + r)\sin\theta - r\sin(\theta + \alpha). \tag{7}$$

Since the rolling occurs without slipping, the two arc lengths of the fixed circle $\widehat{PQ} = \theta R$ and the arc length of the rolling circle $\widehat{BQ} = \alpha r$ must be equal. Thus, the no-slipping condition gives the following relation

$$\alpha = \frac{R}{r}\,\theta. \tag{8}$$

Substituting Equation (8) in Equations (6) and (7) results in the following formulas

$$x(\theta) = (R + r)\cos\theta - r\cos\left(\frac{R + r}{r}\theta\right), \tag{9}$$

$$y(\theta) = (R + r)\sin\theta - r\sin\left(\frac{R + r}{r}\theta\right). \tag{10}$$

These are the parametric equations of the epicycloid, expressed in terms of the rolling angle θ. The structure of the curve, including the number of cusps, depends directly on the ratio $k = R/r$.

The ratio $(R + r)\theta/r$ can be physically interpreted as the relative angular speed of the rolling circle compared to its translational motion around the fixed circle. For every increase of 2π in θ, one complete revolution of the center around the fixed circle, the rolling circle rotates through an angle of $2\pi(R+r)/r$. Thus, the rolling circle completes not one, but $(R + r)/r$ complete rotations about its axis while making a single revolution around the fixed circle. This is a source of the paradox observed with the rolling coin, since the rolling motion and the transport of the circle's center is the combined motion of a circle of radius $R+r$ and not only r, producing more rotations than intuition alone might suggest.

Figure 5 illustrates several examples of epicycloids for different values of the ratio $k = R/r$, where R is the radius of the fixed circle and r is the radius of the rolling circle. When k is an integer, the epicycloid closes with $k + 1$ cusps, producing well-known curves such as the cardioid ($k = 1$), the nephroid ($k = 2$), the trefoiloid ($k = 3$), and the quatrefoiloid ($k = 4$). When k is non-integer or rational, the curve is more intricate and may not close after a single revolution, generating dense, lace-like structures.

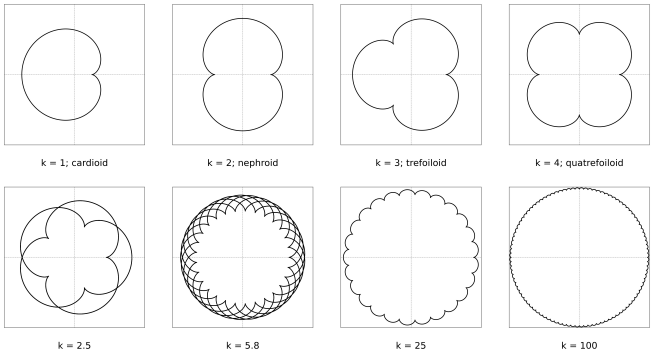

Figure 5: Examples of epicycloids for different values of $k = R/r$.

Fig. 6 shows two points of view for the coin paradox. Fig. (a) on the left indicates that the small coin is drawn with an arrow that initially points towards the center of the large coin. From the point of view of the large coin, the small coin will have completed a full rotation when the arrow once again points to the center of the large coin. This happens after 120 degrees, then again after 240 degrees, and finally after a full rotation of 360 degrees. From this viewpoint, the small coin rotates three times, and each rotation corresponds to the coin rotating along one-third of the large coin's circumference. So there is no paradox here. However, when we view the situation from an external perspective, we see an additional rotation, because while rolling along the edge of the large coin, the small coin also moves in a large circle. Fig. (b) on the right shows that the small coin actually rotates 4 times before it returns to its initial position. This might seem very strange. If the bigger coin has 3 times the circumference of the small coin, and the small coin rotates without slipping, how can it possibly

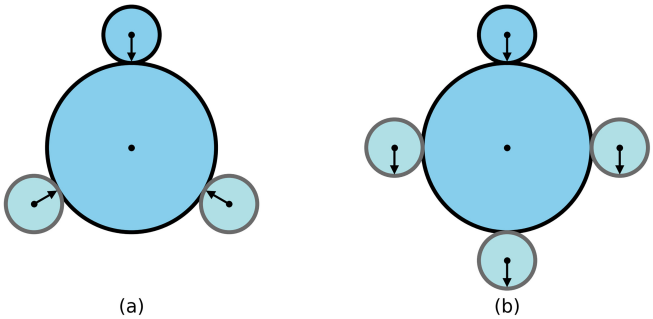

Figure 6: Illustration of the rolling paradox. **Left:** the intuitive but incorrect reasoning, where the small circle is assumed to rotate only R/r times while rolling around the larger circle. **Right:** the correct reasoning: the circular path of the rolling circle's center adds one extra rotation, giving $N = (R + r)/r$.

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

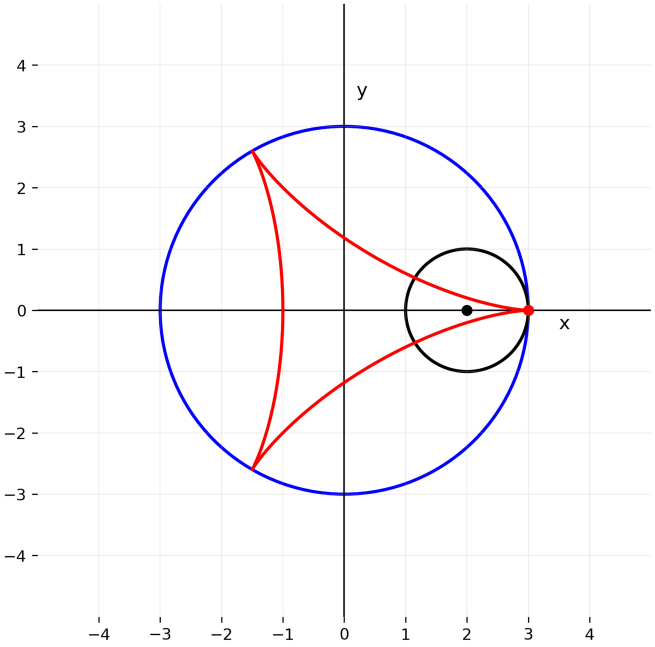

Figure 7: Examples of hypocycloids for $k = 3$, named deltoid [6].

take four rotations to get back to its starting point? But it really does.

4. Hypocycloid

A hypocycloid is generated by the trace of a fixed point on a small circle that rolls within a larger circle [6]. This would be analogous to a coin rolling on the inside of a loop. As the radius of the larger circle is increased, the hypocycloid becomes more like the cycloid, which is the curve traced by a point on a circle as it rolls along a straight line without slipping. A cycloid is a specific form of trochoid. Fig. 7 below illustrates one example of a hypocycloid for $k = 3$, named deltoid [6].

The derivation of the parametric equations for the hypocycloid is analogous to the derivation of the Epicycloid, using a similar figure as the one in Fig. 4, but there is a simple way to get the equations immediately: use the change $r \to -r$ instead to get the following expressions

$$x(\theta) = (R - r)\cos\theta + r\cos\left(\frac{R - r}{r}\theta\right), \qquad (11)$$

$$y(\theta) = (R - r)\sin\theta - r\sin\left(\frac{R - r}{r}\theta\right). \qquad (12)$$

Now the number of complete turns is diminished by one instead of the addition of a turn, as the rolling circle completes the 2π cycle. One interesting property of a hypocycloid is that for any value of r, it is a brachistochrone for the gravitational potential inside a homogeneous sphere of radius R, see pp. 230–2 of Ref. [24].

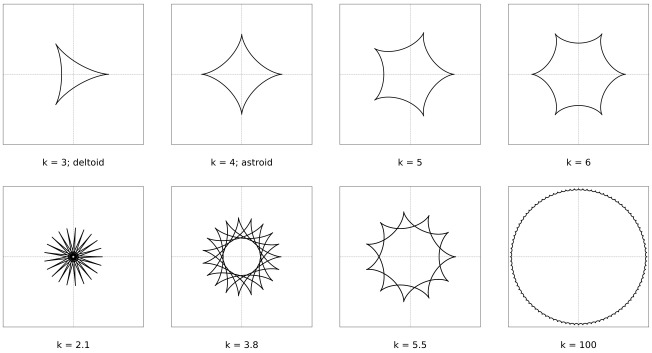

Figure 8: Examples of hypocycloids for different ratios $k = R/r$.

As illustrated in Fig. 8, hypocycloids exhibit a wide variety of shapes depending on the ratio $k = R/r$. When k is an integer, the curve is closed and presents exactly k cusps: for $k = 3$, it gives the deltoid, $k = 4$, the astroid, and larger integers produce more complex star-shaped forms. If k is not an integer, the curve no longer closes after a single revolution of the rolling circle. Instead, the trajectory densely fills a region with intricate star-like patterns. This sensitivity to the value of k highlights the richness of hypocycloids as pedagogical examples, providing a natural way to connect geometry, periodicity, and the role of rational versus irrational ratios in mathematics.

5. Aristotle's Wheel Paradox

A classical example of a geometrical paradox is known as *Aristotle's wheel paradox*. The setup consists of two concentric circles of different radii, rigidly connected, as if they were part of the same wheel. The larger circle rolls without slipping along a straight line.

Intuitively, one might expect that if the larger circle of radius R covers a distance equal to its circumference $2\pi R$, then the smaller circle of radius $r < R$ should simultaneously cover a distance equal to its own circumference $2\pi r$. This reasoning seems natural from the point of view of the individual kinematics of each circle. However, since the smaller circle is rigidly attached and rotates together with the larger circle, both circumferences appear to travel the same linear distance, despite their different radii and circumferences, leading to the paradoxical conclusion that arcs of different sizes are equal in length.

The resolution of the paradox lies in observing the path traced by the smaller circle. Although it is concentric with the larger one, the smaller circle does not *roll* on the ground. Instead, each point on its circumference describes a curve that is no longer a straight line but a *roulette*. Consequently, the smaller circle does not actually unroll its full circumference along the ground. The apparent contradiction arises only if one assumes, incorrectly, that both circles roll in the same manner.

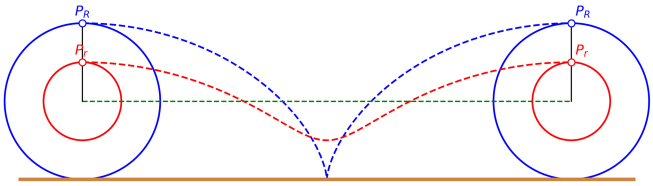

Figure 9: Illustration of the solution of Aristotle's wheel paradox for $k = 2$.

Fig. 9 illustrates the wheel paradox: the circles before and after rolling one revolution, showing the motions of the center of point P_R on the circle of radius R, and P_r of the circle of radius r, with P_R and P_r starting and ending at the top of their respective circles. The green dashed line is the center's motion. The blue dashed curve shows P_R motion. The red dash curve shows P_r motion. Notice that the P_r path is clearly shorter than the path of P_R. The ratio between the two circles in Fig. 9 is $k = R/r = 2$.

The closer P_r is to the center, the shorter, more direct, and closer to the green line its path is. Fig. 10 below shows the ratio $k = 10$

The *trochoids* [5, 25, 26] is the name of the curve described by points P_R in the bigger circle of radius R and P_r in the smaller circle of radius r, seen in Figs.9 and 10 above. The trochoid is a curve formed when a circle rolls without slipping along a straight line. The kinematic constraint of rolling without slip imposes that the horizontal displacement (x_c) of the circle's center is proportional to the arc length traversed on its circumference, $x_c(\theta) = R\theta$, and the vertical coordinate of the center remains constant, $y_c(\theta) = R$ as the circle rolls over the line taken as the x-axis. If a point is located at a fixed distance a from the center, its position in the laboratory frame results from the superposition of the translation of the circle's center and the rotation of the point around the center. The parametric representation of its trajectory is therefore

$$x(\theta) = R\theta + a\sin\theta \tag{13}$$

$$y(\theta) = R + a\cos\theta. \tag{14}$$

The geometrical aspects of the trochoid curve depend on the relation between the distance a and the circle's radius R. If $a = R$, the traced curve reduces to the *cycloid*, the roulette generated by a circle rolling without slipping on a line. If $a < R$, the trajectory is a *curtate trochoid*, characterized by arches that do not reach the baseline; while for $a > R$, one obtains a *prolate trochoid*, whose lobes extend beyond the rolling line. The cycloid, as a particular instance, has played a fundamental role in the history of mathematics and physics, for example, the brachistochrone problem [27, 28].

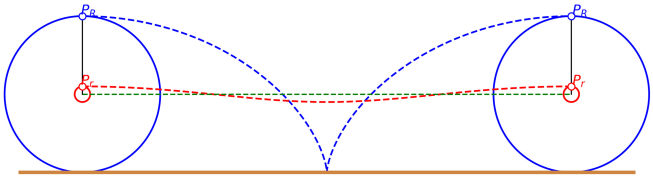

Figure 10: Illustration of the solution of Aristotle's wheel paradox for $k = 10$.

6. Python Codes

6.1. Figure 1: circles with arrows

To produce a proxy for Fig. 1, but instead of coins, the figure shows circles with arrows, run the Python code provided below using a Jupyter Notebook.

keywordstyle

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle,
    FancyArrowPatch

def draw_equal_quincunx_with_arrows(radius=1.0,
     arrow_len=None, savepath=None):
    """
    Five equal circles (radius = radius) in
    quincunx.
    Outer four are tangent to the center circle
     (centers at distance d=2*radius).
    Dashed guide circle goes through the outer
    centers.
    Arrows start at the CENTER of each outer
    circle:
        top & bottom: upwards; left & right:
    downwards.
    The dashed guide is rendered BEHIND the
    disks (hidden where overlapped).
    """
    s = float(radius)
    if s <= 0:
        raise ValueError("radius must be
    positive.")

    d = 2.0 * s
    if arrow_len is None:
        arrow_len = 1.1 * s

    centers = {
        "top":    (0.0,  d),
        "right":  ( d,  0.0),
        "bottom": (0.0, -d),
        "left":   (-d,  0.0),
        "center": (0.0,  0.0),
    }

    # z-order: guide (1) < circles (3) < arrows
     (4)
    z_guide, z_circles, z_arrows = 1, 3, 4

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.set_aspect('equal')
    ax.axis('off')

    # --- dashed guide circle FIRST (behind)
    ---
    ax.add_patch(Circle((0.0, 0.0), d, ec="#4
    a5568", fc="none",
                        lw=1.8, ls="--", zorder
    =z_guide))

```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

```
    # --- disks on top of guide (opaque facecolors hide the dashed where overlapped) ---
    for key in ["top", "right", "bottom", "left", "center"]:
        cx, cy = centers[key]
        ax.add_patch(Circle((cx, cy), s, ec="#2b6cb0", fc="#e6eefc",
                            lw=2, zorder=z_circles))

    # --- arrows on top ---
    def vertical_arrow_from_center(cx, cy, direction):
        if direction == "up":
            start, end = (cx, cy), (cx, cy + arrow_len)
        elif direction == "down":
            start, end = (cx, cy), (cx, cy - arrow_len)
        else:
            raise ValueError("direction must be 'up' or 'down'")
        a = FancyArrowPatch(
            start, end, arrowstyle='-|>',
            mutation_scale=16,
            lw=2.0, color='#44515c', zorder=z_arrows
        )
        ax.add_patch(a)

    vertical_arrow_from_center(*centers["top"], "up")
    vertical_arrow_from_center(*centers["bottom"], "up")
    vertical_arrow_from_center(*centers["left"],   "down")
    vertical_arrow_from_center(*centers["right"],  "down")

    # limits include arrows
    pad = 0.3 * s
    x_pad = s + pad
    y_pad = max(s, arrow_len) + pad
    ax.set_xlim(-d - x_pad, d + x_pad)
    ax.set_ylim(-d - y_pad, d + y_pad)

    if savepath:
        fig.savefig(savepath, dpi=300, bbox_inches='tight', pad_inches=0.1)
    plt.show()
    plt.close(fig)

# Example:
draw_equal_quincunx_with_arrows(radius=1.0, arrow_len=1.1)
```

6.2. Figure 1: circles with arrows animated

To produce a proxy for Fig. 1, but instead of coins, the figure shows circles with arrows, in animation style, run the Python code provided below using a Jupyter Notebook.

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, FancyArrowPatch
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

def animate_two_circles_with_roulette_and_arrow(radius=1.0, frames=360, interval=25,
                                                clockwise=False, savepath=None, fps=30, dpi=150):
    R = float(radius); r = R
    sgn = -1.0 if clockwise else 1.0
    phi0 = np.pi / 2
    phi  = phi0 + sgn * np.linspace(0.0, 2*np.pi, frames, endpoint=False)
    psi  = 2.0 * (phi - phi0)           # external rolling, R=r -> factor 2
    cx   = (R + r) * np.cos(phi)
    cy   = (R + r) * np.sin(phi)
    alpha0 = phi0 + np.pi               # marker initially toward origin
    Px = cx + r * np.cos(psi + alpha0)
    Py = cy + r * np.sin(psi + alpha0)

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.set_aspect('equal'); ax.axis('off')
    center_disk  = Circle((0, 0), R, ec="#2b6cb0", fc="#e6eefc", lw=2, zorder=3)
    rolling_disk = Circle((cx[0], cy[0]), r, ec="#d65f5f", fc="#fdecec", lw=2, zorder=3)
    ax.add_patch(center_disk); ax.add_patch(rolling_disk)
    path_line, = ax.plot([], [], '-', lw=2.0, color='red', zorder=2)
    P_dot,     = ax.plot(Px[0], Py[0], 'o', ms=6, mfc='white', mec='#222', zorder=4)
    arrow = FancyArrowPatch((cx[0], cy[0]), (Px[0], Py[0]),
                            arrowstyle='-|>', mutation_scale=16, lw=2.0,
                            color='#44515c', zorder=4)
    ax.add_patch(arrow)

    pad = 0.4 * R
    ax.set_xlim(-3*R - pad, 3*R + pad)
    ax.set_ylim(-3*R - pad, 3*R + pad)

    def update(i):
        rolling_disk.center = (cx[i], cy[i])
        P_dot.set_data(Px[i], Py[i])
        path_line.set_data(Px[:i+1], Py[:i+1])
        ang = psi[i] + alpha0
        arr_len = 0.9 * R
        start = (cx[i], cy[i])
        end   = (cx[i] + arr_len*np.cos(ang), cy[i] + arr_len*np.sin(ang))
        arrow.set_positions(start, end)
        return rolling_disk, P_dot, path_line, arrow

    anim = FuncAnimation(fig, update, frames=frames, interval=interval, blit=True)

    # ---- Save if requested ----
    if savepath is not None:
        ext = savepath.split(".")[-1].lower()
        if ext in {"mp4", "m4v", "mov"}:
            from matplotlib.animation import FFMpegWriter
            anim.save(savepath, writer=FFMpegWriter(fps=fps), dpi=dpi)
        elif ext in {"gif"}:
            from matplotlib.animation import PillowWriter
```

```
            anim.save(savepath, writer=PillowWriter(fps=fps), dpi=dpi)
        else:
            raise ValueError("Unsupported extension; use .mp4 or .gif")

    plt.close(fig)   # close figure to avoid duplicate display in notebooks
    return HTML(anim.to_jshtml())

# Show inline only
animate_two_circles_with_roulette_and_arrow(radius=1.0)
# Save as MP4 (needs ffmpeg)
animate_two_circles_with_roulette_and_arrow(radius=1.0, savepath="roll.mp4", fps=30, dpi=150)
# Save as GIF (uses Pillow)
animate_two_circles_with_roulette_and_arrow(radius=1.0, savepath="roll.gif", fps=20)
```

6.3. Figure 2: solving the paradox with physics

To produce Fig. 2, run the Python code provided below using a Jupyter Notebook, Google Colab Notebook, or your preferred development environment.

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, FancyArrow, Arc

def draw_rolling_diagram(R=3, r=1, theta_deg=30, save_path=None):
    """
    Rolling-circle diagram with:
      - v drawn from the small-circle center (longer, tangent to center path),
      - w curved arrow placed on TOP of the small circle (outside) to avoid crossings,
      - R and r shown as teal radius segments with midpoint labels.
    """
    theta = np.deg2rad(theta_deg)
    RR = R + r
    O = (0.0, 0.0)
    C = (RR*np.cos(theta), RR*np.sin(theta))

    fig, ax = plt.subplots(figsize=(6, 6))
    col, teal = '#1f2d3a', '#1f6f7a'

    # Dashed path of the rolling center
    ax.add_patch(Circle(O, RR, fill=False, ls='--', lw=1.5, color=col))

    # Circles
    ax.add_patch(Circle(O, R, fill=False, lw=2, color=col))
    ax.add_patch(Circle(C, r, fill=False, lw=2, color=col))

    # Direction from O to C
    ang = np.arctan2(C[1]-O[1], C[0]-O[0])

    # Radius R with midpoint label
    R_end = (O[0] + R*np.cos(ang), O[1] + R*np.sin(ang))
    ax.plot([O[0], R_end[0]], [O[1], R_end[1]], color='#1f6f7a', lw=2)
    R_mid = (O[0] + 0.5*R*np.cos(ang), O[1] + 0.5*R*np.sin(ang))
    ax.text(R_mid[0], R_mid[1], r"$R$", color='#1f6f7a', fontsize=12, ha='center', va='bottom')

    # Radius r with midpoint label
    r_end = (C[0] - r*np.cos(ang), C[1] - r*np.sin(ang))
    ax.plot([C[0], r_end[0]], [C[1], r_end[1]], color='#1f6f7a', lw=2)
    r_mid = (C[0] - 0.5*r*np.cos(ang), C[1] - 0.5*r*np.sin(ang))
    ax.text(r_mid[0], r_mid[1], r"$r$", color='#1f6f7a', fontsize=12, ha='center', va='bottom')

    # Velocity vector from small-circle center (tangent)
    vx, vy = np.cos(ang + np.pi/2), np.sin(ang + np.pi/2)
    v_len = 1.8
    ax.add_patch(FancyArrow(C[0], C[1], v_len*vx, v_len*vy,
                            width=0.025, head_width=0.22, head_length=0.22,
                            color='#1f6f7a'))
    ax.text(C[0] + v_len*vx + 0.12, C[1] + v_len*vy, r"$\vec{v}$", color='#1f6f7a', fontsize=12)

    # w curved arrow OUTSIDE small circle, on top (absolute angles 70deg to 110 deg)
    arc_rad = 1.20 * r
    start_deg_abs, end_deg_abs = 70, 110
    arc = Arc(C, 2*arc_rad, 2*arc_rad, theta1=start_deg_abs, theta2=end_deg_abs, color='#1f6f7a', lw=2)
    ax.add_patch(arc)
    end_rad = np.deg2rad(end_deg_abs)
    hx = C[0] + arc_rad*np.cos(end_rad)
    hy = C[1] + arc_rad*np.sin(end_rad)
    tdx, tdy = -np.sin(end_rad), np.cos(end_rad)  # CCW tangent
    ax.add_patch(FancyArrow(hx - 0.12*tdx, hy - 0.12*tdy, 0.12*tdx, 0.12*tdy,
                            width=0.0, head_width=0.18, head_length=0.18,
                            color='#1f6f7a', length_includes_head=True))
    ax.text(hx + 0.22, hy + 0.18, r"$\omega$", color='#1f6f7a', fontsize=12)

    # Layout
    ax.set_aspect('equal', adjustable='box')
    ax.set_xlim(-RR-1.4, RR+1.8)
    ax.set_ylim(-RR-1.4, RR+1.8)
    ax.axis('off')

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')
    return fig, ax

# Example usage:
fig, ax = draw_rolling_diagram(R=3, r=1, theta_deg=30, save_path="fig2.png")
```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

6.4. Figure 3: the epicycloid

To produce Fig. 3, one must execute the Python code provided below.

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

def plot_epicycloid(R=3.0, r=1.0, points=2000, save_path=None, show=True):
    """
    Plot the epicycloid generated by a circle of radius r rolling
    around a fixed circle of radius R.

    Parameters
    ----------
    R : float
        Radius of the fixed circle.
    r : float
        Radius of the rolling circle.
    points : int
        Number of sample points for the epicycloid.
    save_path : str or None
        If provided, saves the figure to this path.
    show : bool
        If True, displays the figure. If False, closes it.

    Returns
    -------
    fig, ax : Matplotlib figure and axes objects
    """
    k = (R + r) / r
    theta = np.linspace(0, 2*np.pi, points)
    x = (R + r) * np.cos(theta) - r * np.cos(k * theta)
    y = (R + r) * np.sin(theta) - r * np.sin(k * theta)

    fig, ax = plt.subplots(figsize=(6,6))

    # Grid
    ax.set_xlim(-R-r-1, R+r+1)
    ax.set_ylim(-R-r-1, R+r+1)
    ax.set_aspect('equal', 'box')
    ax.set_xticks(np.arange(-R-r, R+r+1, 1))
    ax.set_yticks(np.arange(-R-r, R+r+1, 1))
    ax.grid(True, which='both', linestyle='-', linewidth=0.5, alpha=0.25)

    # Axes lines
    ax.axhline(0, color='k', linewidth=1)
    ax.axvline(0, color='k', linewidth=1)
    ax.text(R+r-0.5, -0.3, 'x', fontsize=12)
    ax.text(0.2, R+r-0.5, 'y', fontsize=12)

    # Fixed circle (blue)
    ax.add_patch(Circle((0,0), R, fill=False, edgecolor='blue', linewidth=2))

    # Rolling circle (black) at initial position
    rolling_center = (R + r, 0.0)
    ax.add_patch(Circle(rolling_center, r, fill=False, edgecolor='black', linewidth=2))

    # Center dot
    ax.plot(rolling_center[0], rolling_center[1], 'ko')

    # Epicycloid path
    ax.plot(x, y, color='red', linewidth=2)

    # Starting point
    ax.plot([R], [0], 'o', color='red')

    # Clean frame
    for spine in ax.spines.values():
        spine.set_visible(False)

    plt.tight_layout()

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')

    if not show:
        plt.close(fig)

    return fig, ax

fig, ax = plot_epicycloid(R=3.0, r=1.0, save_path="fig3.png")
```

6.5. Figure 3 animated: the epicycloid animation

The Python code below produces Fig. 3 in animation style. Run the code in your Jupyter Notebook to see an MP4 video of the epicycloid being traced.

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from matplotlib.animation import FuncAnimation
from matplotlib import rc

rc("animation", html="jshtml")   # for Jupyter animation display

def animate_epicycloid(R=3.0, r=1.0, frames=600, interval=20):
    """
    Animate the epicycloid generated by a circle of radius r rolling
    around a fixed circle of radius R.

    Parameters
    ----------
    R : float
        Radius of the fixed circle.
    r : float
        Radius of the rolling circle.
    frames : int
        Number of frames in the animation.
    interval : int
        Delay between frames in milliseconds.

    Returns
    -------
    ani : matplotlib.animation.FuncAnimation
        Animation object (renders inline in Jupyter).
    """
    k = (R + r) / r
    theta = np.linspace(0, 2*np.pi, frames)
```

```
    x = (R + r) * np.cos(theta) - r * np.cos(k * theta)
    y = (R + r) * np.sin(theta) - r * np.sin(k * theta)

    fig, ax = plt.subplots(figsize=(6,6))
    ax.set_xlim(-R-r-1, R+r+1)
    ax.set_ylim(-R-r-1, R+r+1)
    ax.set_aspect('equal', 'box')
    ax.set_xticks(np.arange(-R-r, R+r+1, 1))
    ax.set_yticks(np.arange(-R-r, R+r+1, 1))
    ax.grid(True, linestyle='-', linewidth=0.5, alpha=0.25)
    ax.axhline(0, color='k', linewidth=1)
    ax.axvline(0, color='k', linewidth=1)
    ax.text(R+r-0.5, -0.3, 'x')
    ax.text(0.2, R+r-0.5, 'y')
    for s in ax.spines.values():
        s.set_visible(False)

    # Fixed circle
    ax.add_patch(Circle((0,0), R, fill=False, edgecolor='blue', linewidth=2))

    # Rolling circle
    rolling_circle = Circle((R+r, 0), r, fill=False, edgecolor='black', linewidth=2)
    ax.add_patch(rolling_circle)

    # Traced point and path
    trace_dot, = ax.plot([], [], 'ro')
    trace_line, = ax.plot([], [], 'r-', linewidth=2)

    # Starting point
    ax.plot([R], [0], 'o', color='red')
    plt.close(fig)

    def init():
        rolling_circle.center = (R+r, 0)
        trace_dot.set_data([], [])
        trace_line.set_data([], [])
        return trace_dot, trace_line, rolling_circle

    def update(i):
        cx = (R + r) * np.cos(theta[i])
        cy = (R + r) * np.sin(theta[i])
        rolling_circle.center = (cx, cy)
        trace_dot.set_data(x[i], y[i])
        trace_line.set_data(x[:i+1], y[:i+1])
        return trace_dot, trace_line, rolling_circle

    ani = FuncAnimation(fig, update, init_func=init,
                        frames=frames, interval=interval, blit=True)
    return ani
ani = animate_epicycloid(R=3.0, r=1.0, frames=600, interval=30)
ani  # shows animation
```

6.6. Figure 5: several examples of epicycloids

The Python code below produces Fig. 5. Run the code in your Jupyter Notebook or Google Colab Notebook.

```
  keywordstyle
import numpy as np
import matplotlib.pyplot as plt

def epicycloid_xy(k, num_points=2400, turns=1.0):
    r = 1.0
    R = k * r
    w = (R + r) / r
    t = np.linspace(0, 2 * np.pi * turns, num_points)
    x = (R + r) * np.cos(t) - r * np.cos(w * t)
    y = (R + r) * np.sin(t) - r * np.sin(w * t)
    return x, y

def plot_panel(ax, k, subtitle=None):
    turns = 1.0 if abs(k - round(k)) < 1e-9 else 12.0
    x, y = epicycloid_xy(k, num_points=4000, turns=turns)
    ax.axhline(0, color='0.6', lw=0.8, ls='--')
    ax.axvline(0, color='0.6', lw=0.8, ls='--')
    ax.plot(x, y, color='black', lw=1.8)
    ax.set_aspect('equal', 'box')
    lim = k + 3.0
    ax.set_xlim(-lim, lim)
    ax.set_ylim(-lim, lim)
    ax.set_xticks([]); ax.set_yticks([])
    for spine in ax.spines.values():
        spine.set_linewidth(1.0)
        spine.set_color('0.3')
    if subtitle:
        ax.text(0.5, -0.10, subtitle, transform=ax.transAxes,
                ha='center', va='top', fontsize=22)

# Parameters
k_list = [1, 2, 3, 4, 2.5, 5.8, 25, 100]
subs = ["k = 1; cardioid",
        "k = 2; nephroid",
        "k = 3; trefoiloid",
        "k = 4; quatrefoiloid",
        "k = 2.5",
        "k = 5.8",
        "k = 25",
        "k = 100"]

# Make subplot areas bigger
fig, axes = plt.subplots(2, 4, figsize=(22, 12))  # Bigger figure = bigger boxes
axes = axes.ravel()

for i in range(len(k_list)):
    plot_panel(axes[i], k_list[i], subtitle=subs[i])

# Make panels occupy more space
plt.subplots_adjust(left=0.00, right=1, top=1, bottom=0.1,
                    wspace=0.1, hspace=0.3)  # Smaller wspace/hspace = bigger boxes

# Show or save
plt.savefig("fig5.png", dpi=300, bbox_inches="tight")
plt.show()
```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

6.7. Figure 6: two different views of the coin paradox

To produce Fig. 6 run the Python code bellow

keywordstyle

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, FancyArrow
import numpy as np

def draw_coin(ax, center, radius, facecolor='#87CEEB', edgecolor='black',
              arm_angle=None, arm_len=None, armcolor='black', lw=3):
    """Draw a coin with an optional orientation arrow."""
    circ = Circle(center, radius, facecolor=facecolor, edgecolor=edgecolor, lw=lw)
    ax.add_patch(circ)
    # center dot
    ax.plot(center[0], center[1], 'o', color='black', ms=4)
    # orientation arrow
    if arm_angle is not None:
        if arm_len is None:
            arm_len = radius * 0.9
        x0, y0 = center
        x1 = x0 + arm_len * np.cos(arm_angle)
        y1 = y0 + arm_len * np.sin(arm_angle)
        arr = FancyArrow(x0, y0, x1 - x0, y1 - y0,
                         width=0.03*radius, head_width=0.25*radius, head_length=0.25*radius,
                         color=armcolor, length_includes_head=True)
        ax.add_patch(arr)

def tangent_center(big_c, R, r, angle):
    """Center of a small circle tangent to the big circle at polar angle 'angle'."""
    return (big_c[0] + (R + r) * np.cos(angle),
            big_c[1] + (R + r) * np.sin(angle))

def draw_scene(ax, big_center, R, r, mode='left',
               big_color='#87CEEB', small_color='#B0E0E6'):
    """Draw one panel (left or right)."""
    # Big coin
    draw_coin(ax, big_center, R, facecolor=big_color, lw=3)

    # Top small coin (tangent at 90) with downward arrow
    top_c = tangent_center(big_center, R, r, np.pi/2)
    draw_coin(ax, top_c, r, facecolor=big_color, lw=3, arm_angle=-np.pi/2)

    if mode == 'left':
        # Two tangent small coins at 210 and 330 with angled arrows
        c1 = tangent_center(big_center, R, r, np.deg2rad(210))
        c2 = tangent_center(big_center, R, r, np.deg2rad(330))
        draw_coin(ax, c1, r, facecolor=small_color, edgecolor='#6b6b6b',
                  arm_angle=np.deg2rad(30))
        draw_coin(ax, c2, r, facecolor=small_color, edgecolor='#6b6b6b',
                  arm_angle=np.deg2rad(150))
    else:
        # Three tangent coins: left (180), right (0), bottom (270)
        left_c   = tangent_center(big_center, R, r, np.pi)
        right_c  = tangent_center(big_center, R, r, 0.0)
        bottom_c = tangent_center(big_center, R, r, -np.pi/2)

        # Left & right coins: arrows up
        draw_coin(ax, left_c,  r, facecolor=small_color, edgecolor='#6b6b6b',
                  arm_angle=-np.pi/2)
        draw_coin(ax, right_c, r, facecolor=small_color, edgecolor='#6b6b6b',
                  arm_angle=-np.pi/2)

        # Bottom coin: arrow DOWN (fixed)
        draw_coin(ax, bottom_c, r, facecolor=small_color, edgecolor='#6b6b6b',
                  arm_angle=-np.pi/2)

def print_image(r=0.6, k=3, center_distance=8.0, save_path=None):
    """Render the two panels and optionally save to file."""
    R = k * r  # enforce R = k r

    fig, ax = plt.subplots(figsize=(10, 5))
    left_center  = (-center_distance/2, 0.0)
    right_center = ( center_distance/2, 0.0)

    draw_scene(ax, left_center,  R, r, mode='left')
    draw_scene(ax, right_center, R, r, mode='right')

    # Labels (a) and (b) under each panel
    y_label = - (R + r) - 0.8
    ax.text(left_center[0],  y_label, '(a)', fontsize=14, ha='center', va='top')
    ax.text(right_center[0], y_label, '(b)', fontsize=14, ha='center', va='top')

    # Layout
    extent = center_distance/2 + (R + r) + 1.0
    ax.set_xlim(-extent, extent)
    ax.set_ylim(- (R + r) - 1.5, (R + r) + 1.5)
    ax.set_aspect('equal', adjustable='box')
    ax.axis('off')

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')
    plt.show()

# Example
print_image(r=0.6, k=3, center_distance=8.0, save_path="fig6.png")
```

6.8. Figure 7: the hypocycloid

To produce Fig. 7 run the Python code bellow

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from matplotlib.patches import Circle

def plot_epicycloid(R=3.0, r=1.0, points=2000, save_path=None, show=True):
    """
    Plot the epicycloid generated by a circle of radius r rolling
    around a fixed circle of radius R.

    Parameters
    ----------
    R : float
        Radius of the fixed circle.
    r : float
        Radius of the rolling circle.
    points : int
        Number of sample points for the epicycloid.
    save_path : str or None
        If provided, saves the figure to this path.
    show : bool
        If True, displays the figure. If False, closes it.

    Returns
    -------
    fig, ax : Matplotlib figure and axes objects
    """
    k = - (R - r) / r
    theta = np.linspace(0, 2*np.pi, points)
    x = (R - r) * np.cos(theta) + r * np.cos(k * theta)
    y = (R - r) * np.sin(theta) + r * np.sin(k * theta)

    fig, ax = plt.subplots(figsize=(6,6))

    # Grid
    ax.set_xlim(-R-r-1, R+r+1)
    ax.set_ylim(-R-r-1, R+r+1)
    ax.set_aspect('equal', 'box')
    ax.set_xticks(np.arange(-R-r, R+r+1, 1))
    ax.set_yticks(np.arange(-R-r, R+r+1, 1))
    ax.grid(True, which='both', linestyle='-', linewidth=0.5, alpha=0.25)

    # Axes lines
    ax.axhline(0, color='k', linewidth=1)
    ax.axvline(0, color='k', linewidth=1)
    ax.text(R+r-0.5, -0.3, 'x', fontsize=12)
    ax.text(0.2, R+r-0.5, 'y', fontsize=12)

    # Fixed circle (blue)
    ax.add_patch(Circle((0,0), R, fill=False, edgecolor='blue', linewidth=2))

    # Rolling circle (black) at initial position
    rolling_center = (R - r, 0.0)
    ax.add_patch(Circle(rolling_center, r, fill=False, edgecolor='black', linewidth=2))

    # Center dot
    ax.plot(rolling_center[0], rolling_center[1], 'ko')

    # Epicycloid path
    ax.plot(x, y, color='red', linewidth=2)

    # Starting point
    ax.plot([R], [0], 'o', color='red')

    # Clean frame
    for spine in ax.spines.values():
        spine.set_visible(False)

    plt.tight_layout()

    if save_path:
        plt.savefig(save_path, dpi=300, bbox_inches='tight')

    if not show:
        plt.close(fig)

    return fig, ax

fig, ax = plot_epicycloid(R=3.0, r=1.0, save_path="fig7.png")
```

6.9. Figure 7: the animated hypocycloid

The Python code below produces Fig. 7 in animation style. Run the code in your Jupyter Notebook to see an MP4 video of the epicycloid being traced.

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from matplotlib.animation import FuncAnimation
from matplotlib import rc

rc("animation", html="jshtml")   # for Jupyter animation display

def animate_epicycloid(R=3.0, r=1.0, frames=600, interval=20):
    """
    Animate the epicycloid generated by a circle of radius r rolling
    around a fixed circle of radius R.

    Parameters
    ----------
    R : float
        Radius of the fixed circle.
    r : float
        Radius of the rolling circle.
    frames : int
        Number of frames in the animation.
    interval : int
        Delay between frames in milliseconds.

    Returns
    -------
    ani : matplotlib.animation.FuncAnimation
        Animation object (renders inline in Jupyter).
    """
    k = - (R - r) / r
    theta = np.linspace(0, 2*np.pi, frames)
    x = (R - r) * np.cos(theta) + r * np.cos(k * theta)
    y = (R - r) * np.sin(theta) + r * np.sin(k * theta)

    fig, ax = plt.subplots(figsize=(6,6))
    ax.set_xlim(-R-r-1, R+r+1)
    ax.set_ylim(-R-r-1, R+r+1)
```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

```
    ax.set_aspect('equal', 'box')
    ax.set_xticks(np.arange(-R-r, R+r+1, 1))
    ax.set_yticks(np.arange(-R-r, R+r+1, 1))
    ax.grid(True, linestyle='-', linewidth=0.5,
     alpha=0.25)
    ax.axhline(0, color='k', linewidth=1)
    ax.axvline(0, color='k', linewidth=1)
    ax.text(R+r-0.5, -0.3, 'x')
    ax.text(0.2, R+r-0.5, 'y')
    for s in ax.spines.values():
        s.set_visible(False)

    # Fixed circle
    ax.add_patch(Circle((0,0), R, fill=False,
    edgecolor='blue', linewidth=2))

    # Rolling circle
    rolling_circle = Circle((R+r, 0), r, fill=
    False, edgecolor='black', linewidth=2)
    ax.add_patch(rolling_circle)

    # Traced point and path
    trace_dot, = ax.plot([], [], 'ro')
    trace_line, = ax.plot([], [], 'r-',
    linewidth=2)

    # Starting point
    ax.plot([R], [0], 'o', color='red')
    plt.close(fig)

    def init():
        rolling_circle.center = (R+r, 0)
        trace_dot.set_data([], [])
        trace_line.set_data([], [])
        return trace_dot, trace_line,
    rolling_circle

    def update(i):
        cx = (R - r) * np.cos(theta[i])
        cy = (R - r) * np.sin(theta[i])
        rolling_circle.center = (cx, cy)
        trace_dot.set_data(x[i], y[i])
        trace_line.set_data(x[:i+1], y[:i+1])
        return trace_dot, trace_line,
    rolling_circle

    ani = FuncAnimation(fig, update, init_func=
    init,
                        frames=frames, interval
    =interval, blit=True)
    return ani

ani = animate_epicycloid(R=3.0, r=1.0, frames
    =600, interval=30)
ani  # shows animation
```

6.10. Figure 8: several examples of hypocycloid

To produce Fig. 8 run the Python code bellow

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt

def epicycloid_xy(k, num_points=2400, turns
    =1.0):
    r = 1.0
    R = k * r
    w = - (R - r) / r
    t = np.linspace(0, 2 * np.pi * turns,
    num_points)
    x = (R - r) * np.cos(t) + r * np.cos(w * t)
    y = (R - r) * np.sin(t) + r * np.sin(w * t)
    return x, y

def plot_panel(ax, k, subtitle=None):
    turns = 1.0 if abs(k - round(k)) < 1e-9
    else 12.0
    x, y = epicycloid_xy(k, num_points=4000,
    turns=turns)
    ax.axhline(0, color='0.6', lw=0.8, ls='--')
    ax.axvline(0, color='0.6', lw=0.8, ls='--')
    ax.plot(x, y, color='black', lw=1.8)
    ax.set_aspect('equal', 'box')
    lim = k + 3.0
    ax.set_xlim(-lim, lim)
    ax.set_ylim(-lim, lim)
    ax.set_xticks([]); ax.set_yticks([])
    for spine in ax.spines.values():
        spine.set_linewidth(1.0)
        spine.set_color('0.3')
    if subtitle:
        ax.text(0.5, -0.10, subtitle, transform
    =ax.transAxes,
                ha='center', va='top', fontsize
    =22)

# Parameters
k_list = [3, 4, 5, 6, 2.1, 3.8, 5.5, 100]
subs = ["k = 3; deltoid",
        "k = 4; astroid",
        "k = 5",
        "k = 6",
        "k = 2.1",
        "k = 3.8",
        "k = 5.5",
        "k = 100"]

# Make subplot areas bigger
fig, axes = plt.subplots(2, 4, figsize=(22, 12)
    )  # Bigger figure = bigger boxes
axes = axes.ravel()

for i in range(len(k_list)):
    plot_panel(axes[i], k_list[i], subtitle=
    subs[i])

# Make panels occupy more space
plt.subplots_adjust(left=0.00, right=1, top=1,
    bottom=0.1,
                    wspace=0.1, hspace=0.3)  #
    Smaller wspace/hspace = bigger boxes

# Show or save
plt.savefig("fig8.png", dpi=300, bbox_inches="
    tight")
plt.show()
```

6.11. Figure 9: Aristotle's wheel paradox

To produce Fig. 9 run the Python code bellow

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt

def plot_aristotle_wheel(R=2.0, r=1.0, savepath
    =None):
    """
    Plot Aristotle's Wheel Paradox diagram for
    two concentric rolling circles.

```

```
    Parameters
    ----------
    R : float
        Radius of the larger circle.
    r : float
        Radius of the smaller circle.
    savepath : str or None
        If provided, saves the figure to the given path.
    """
    theta = np.linspace(0, 2 * np.pi, 600)

    # General trochoid for a point at distance a from center
    def trochoid(a):
        x = R * theta + a * np.sin(theta)
        y = R + a * np.cos(theta)
        return x, y

    # Trajectories for P_R (a=R) and P_r (a=r)
    x_R, y_R = trochoid(R)
    x_r, y_r = trochoid(r)

    # Center line at y = R, spanning exactly 2 pi R
    x_line = [0.0, 2 * np.pi * R]
    y_line = [R, R]

    # Figure/axes
    fig, ax = plt.subplots(figsize=(12, 4))
    ax.set_aspect('equal')
    ax.axis('off')

    # Paths
    ax.plot(x_R, y_R, 'b--', lw=1.5, label="Path of $P_R$")
    ax.plot(x_r, y_r, 'r--', lw=1.5, label="Path of $P_r$")
    ax.plot(x_line, y_line, 'g--', lw=1.2, label="Center line")

    # Vertical black lines at start and end (center to top of large circle)
    ax.plot([0, 0], [R, R + R], 'k', lw=1)
    ax.plot([2 * np.pi * R, 2 * np.pi * R], [R, R + R], 'k', lw=1)

    # Circles at start (x=0) and end (x=2 pi R)
    left_large  = plt.Circle((0.0, R), R, edgecolor='blue', fill=False, lw=1.5)
    left_small  = plt.Circle((0.0, R), r, edgecolor='red',  fill=False, lw=1.5)
    right_large = plt.Circle((2 * np.pi * R, R), R, edgecolor='blue', fill=False, lw=1.5)
    right_small = plt.Circle((2 * np.pi * R, R), r, edgecolor='red',  fill=False, lw=1.5)
    for c in (left_large, left_small, right_large, right_small):
        ax.add_patch(c)

    # Markers and labels: top points on each circle at start/end
    PR_left  = (0.0, R + R)
    Pr_left  = (0.0, R + r)
    PR_right = (2 * np.pi * R, R + R)
    Pr_right = (2 * np.pi * R, R + r)

    ax.plot(*PR_left,  'bo', markersize=6, mfc='white')
    ax.plot(*Pr_left,  'ro', markersize=6, mfc='white')
    ax.plot(*PR_right, 'bo', markersize=6, mfc='white')
    ax.plot(*Pr_right, 'ro', markersize=6, mfc='white')

    ax.text(PR_left[0]  - 0.2, PR_left[1]  + 0.2, r"$P_R$", fontsize=12, color='blue')
    ax.text(Pr_left[0]  - 0.2, Pr_left[1]  + 0.2, r"$P_r$", fontsize=12, color='red')
    ax.text(PR_right[0] + 0.1, PR_right[1] + 0.2, r"$P_R$", fontsize=12, color='blue')
    ax.text(Pr_right[0] + 0.1, Pr_right[1] + 0.2, r"$P_r$", fontsize=12, color='red')

    # Ground line
    x_pad = max(R, r) * 0.8
    ax.plot([-x_pad, 2 * np.pi * R + x_pad], [0, 0], color='peru', lw=3)

    # Robust limits: include full circles and labels for any R, r
    pad_y = max(R, r) * 0.3
    y_min = min(0.0, R - max(R, r)) - pad_y
    y_max = R + max(R, r) + pad_y
    x_min = -R - x_pad
    x_max = 2 * np.pi * R + R + x_pad

    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)

    if savepath:
        fig.savefig(savepath, dpi=300, bbox_inches='tight', pad_inches=0.1)

    plt.close(fig)  # close to avoid handle leaks
    return fig

# Save as PNG
fig = plot_aristotle_wheel(R=2, r=1, savepath="fig9.png")
fig
```

6.12. Figure 9: Aristotle's wheel paradox animated

To produce Fig. 9 in animation style, run the Python code below

keywordstyle

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

def animate_aristotle_wheel(R=2.0, r=1.0, frames=300, interval=30, show_trails=True):
    """
    Animate Aristotle's Wheel Paradox: two concentric circles of radii R and r
    roll without slipping over a distance 2piR. Points P_R (a=R) and P_r (a=r)
    trace trochoids. Returns HTML for inline display in Jupyter.
    """
    # Parameter over one full revolution
    theta = np.linspace(0.0, 2*np.pi, frames)

```

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2025-0401

```
    # Helpers
    def trochoid(a, th):
        x = R*th + a*np.sin(th)
        y = R    + a*np.cos(th)
        return x, y

    # Precompute paths for trails
    xR_path, yR_path = trochoid(R, theta)   # P_R path
    xr_path, yr_path = trochoid(r, theta)   # P_r path

    # Figure and axes
    fig, ax = plt.subplots(figsize=(12, 4))
    ax.set_aspect('equal')
    ax.axis('off')

    # Static floor and center line
    x_total = 2*np.pi*R
    pad_x = max(R, r)*0.8
    ax.plot([-pad_x, x_total + pad_x], [0, 0], color='peru', lw=3)   # floor
    ax.plot([0, x_total], [R, R], 'g--', lw=1.2)                     # center line y=R

    # Static end markers: vertical lines at start and end
    ax.plot([0, 0], [R, R+R], 'k', lw=1)
    ax.plot([x_total, x_total], [R, R+R], 'k', lw=1)

    # Dynamic patches: rolling concentric circles (share same center)
    big = Circle((0, R), R, fill=False, lw=1.8, ec='blue')
    small = Circle((0, R), r, fill=False, lw=1.8, ec='red')
    ax.add_patch(big)
    ax.add_patch(small)

    # Moving points P_R and P_r
    PR_dot, = ax.plot([], [], 'o', ms=6, mfc='white', mec='blue')
    Pr_dot, = ax.plot([], [], 'o', ms=6, mfc='white', mec='red')

    # Labels near points
    PR_lbl = ax.text(0, 0, r"$P_R$", fontsize=12, color='blue',
                     ha='left', va='bottom')
    Pr_lbl = ax.text(0, 0, r"$P_r$", fontsize=12, color='red',
                     ha='left', va='bottom')

    # Optional trails
    PR_trail, = ax.plot([], [], 'b--', lw=1.2) if show_trails else (None,)
    Pr_trail, = ax.plot([], [], 'r--', lw=1.2) if show_trails else (None,)

    # Robust limits (avoid clipping for any R, r)
    pad_y = max(R, r)*0.35
    y_min = min(0.0, R - max(R, r)) - pad_y
    y_max = R + max(R, r) + pad_y
    x_min = -R - pad_x
    x_max = x_total + R + pad_x
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)

    # Animation update
    def update(i):
        th = theta[i]
        # Rolling center (translation only; rotation is encoded in trochoid points)
        center_x = R*th
        center_y = R
        big.center = (center_x, center_y)
        small.center = (center_x, center_y)

        # Current P_R and P_r
        xR, yR = trochoid(R, th)
        xr, yr = trochoid(r, th)

        PR_dot.set_data(xR, yR)
        Pr_dot.set_data(xr, yr)

        PR_lbl.set_position((xR + 0.08*R, yR + 0.08*R))
        Pr_lbl.set_position((xr + 0.08*R, yr + 0.08*R))

        # Update trails
        artists = [big, small, PR_dot, Pr_dot, PR_lbl, Pr_lbl]
        if show_trails:
            PR_trail.set_data(xR_path[:i+1], yR_path[:i+1])
            Pr_trail.set_data(xr_path[:i+1], yr_path[:i+1])
            artists += [PR_trail, Pr_trail]
        return artists

    anim = FuncAnimation(fig, update, frames=frames, interval=interval, blit=True)
    plt.close(fig)  # prevent duplicate static output
    return anim

# Example (in a Jupyter cell):
anim = animate_aristotle_wheel(R=2.0, r=1.0)
# Save to MP4 (requires ffmpeg installed)
anim.save("aristotle_wheel.mp4", writer="ffmpeg", dpi=150)
# Or save to GIF (requires ImageMagick or Pillow support)
anim.save("aristotle_wheel.gif", writer="pillow", dpi=100)
# Display animation
HTML(anim.to_jshtml())
```

7. Conclusion

This work demonstrated that the coin paradox, Aristotle's wheel paradox, and the geometry of epicycloids and hypocycloids follow directly from the no-slip rolling constraint, which yields the rotation count $N = (R \pm r)/r$ for a circle of radius r rolling on or inside a circle of radius R. The direct application of basic concepts in circular kinematics was used to solve apparent paradoxes and establish links between classical mechanics, geometry, and visualization.

Parametric equations for epicycloids and hypocycloids were derived and illustrated using Python-generated figures, providing a computational toolset for teaching kinematics and curve geometry. Because roulette connects rotation, constraint motion, and analytic geometry with minimal prerequisites, it offers a compact, visually compelling framework for undergraduate physics and

mathematics instruction. As a bonus, Python code for animated figures was included.

The use of visual aids, such as computer-generated animations and interactive computational tools, can significantly enhance the pedagogy of physics education. Platforms such as YouTube, with channels like Veritasium [29] and 3Blue1Brown [30], have become commonplace resources for undergraduate and graduate students seeking to complement traditional university instruction with accessible, visually engaging explanations.

Data Availability

No new data were created or analyzed in this study.

References

[1] J.C. Maxwell, XXXV on Trans. Roy. Soc. Edin. **16**, 519 (1849).

[2] E.H. Lockwood, in: *A Book of Curves* (Cambridge University Press, Cambridge, 1967).

[3] H. Cundy and A. Rollett, in: *Mathematical Models* (Tarquin Publications, Stradbroke, 1989), 3 ed.

[4] M. Gardner, *The Sixth Book of Mathematical Games from Scientific American* (University of Chicago Press, Chicago, 1984).

[5] R.C. Yates, *A Handbook on Curves and Their Properties* (J.W. Edwards, Ann Arbor, 1952).

[6] J.D. Lawrence, *A Catalog of Special Plane Curves* (Dover, New York, 1972).

[7] D. Zwillinger, in: *CRC Standard Mathematical Tables and Formulae* (CRC Press, Boca Raton, 1996), 3 ed.

[8] M. Gardner, in: *Mathematical Carnival* (Alfred A. Knopf, New York, 1975).

[9] T. Pappas, in: *The Joy of Mathematics* (World Public./Tetra, San Carlos, 1989).

[10] H. Steinhaus, *Mathematical Snapshots* (Dover, New York, 1999), 3 ed.

[11] I.E. Drabkin, Aristotle's Wheel: Notes on the History of a Paradox **9**, 162 (1950).

[12] D.W. Ballew, Math. Teach. **65**, 507 (1972).

[13] P. Costabel, Math. Teacher **61**, 527 (1968).

[14] A.K. Bartlett, Popular Astronomy **12**, 649 (1904).

[15] D.R. Abad, Rev. Bras. Ensino Fís. **43**, e20200482 (2021).

[16] D. Halliday, R. Resnick and J. Walker, *Fundamentals of Physics Extended* (Wiley, Hoboken, 2013), 10 ed.

[17] Error found in S.A.T. question, The New York Times Archives, May 25 1982, available in: https://www.nytimes.com/1982/05/25/us/error-found-in-sat-question.html.

[18] MINDYOURDECISIONS, *Why did everyone miss this SAT Math question?*, available in: https://www.youtube.com/watch?v=kN3AOMrnEUs.

[19] J. Murtagh, *The SAT Problem That Everybody Got Wrong*, available in: https://www.scientificamerican.com/article/the-sat-problem-that-everybody-got-wrong/.

[20] VERITASIUM YOUTUBE CHANNEL, *The SAT Question Everyone Got Wrong*, available in:https://www.youtube.com/watch?v=FUHkTs-Ipfg.

[21] J.J. Uicker, G.R. Pennock and J.E. Shigley, *Theory of Machines and Mechanisms* (Oxford University Press, New York, 2003).

[22] B. Paul, *Kinematics and Dynamics of Planar Machinery* (Prentice Hall, Englewood Cliffs, 1979).

[23] J.A. Boyle, arXiv:1406.1736v1 (2014).

[24] N.C. Rana and P.S. Joag, *Classical Mechanics* (Tata McGraw-Hill, Noida, 2001).

[25] E.A. Whitman, American Mathematical Monthly **50**, 309 (1943).

[26] S. Wagon, *Mathematica in Action* (Springer Science & Business Media, New York, 1999).

[27] R. Courant and H. Robbins, *What Is Mathematics?: An Elementary Approach to Ideas and Methods* (Oxford University Press, Oxford, 1996), 2 ed.

[28] S.T. Thornton and J.B. Marion, *Classical Dynamics of Particles and Systems* (Brooks/Cole, Pacific Grove, 2004), 5 ed.

[29] VERITASIUM YOUTUBE CHANNEL, available in: https://www.youtube.com/@veritasium, accessed in: 15/10/2025.

[30] 3BLUE1BROWN YOUTUBE CHANNEL, available in: https://www.youtube.com/c/3blue1brown, accessed in: 15/10/2025.