ANEXO I. Solución de ejercicios propuestos.

Módulo 2. Creación de curvas cónicas, triángulos y piezas industriales en Python.

Ejercicio 2.1.1.6. Rotación y traslación de una elipse.

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

# Parámetros de la elipse original
a = 5   # semieje mayor
b = 3   # semieje menor

# Ángulos para parametrización
theta = np.linspace(0, 2*np.pi, 300)

# Ecuaciones paramétricas de la elipse centrada en el origen
x = a * np.cos(theta)
y = b * np.sin(theta)

# --- Transformaciones ---
# Traslación
Tx, Ty = 4, 2   # vector de traslación

# Rotación
phi = np.deg2rad(30)  # ángulo en radianes
R = np.array([[np.cos(phi), -np.sin(phi)],
              [np.sin(phi),  np.cos(phi)]])

# Aplicar rotación a cada punto
puntos = np.vstack((x, y))  # matriz 2xN
puntos_rot = R @ puntos

# Aplicar traslación
x_rot_tras = puntos_rot[0, :] + Tx
y_rot_tras = puntos_rot[1, :] + Ty

# --- Graficar ---
fig, ax = plt.subplots(figsize=(6,6))
ax.set_aspect('equal')

# Elipse original
ax.plot(x, y, 'b--', label='Elipse original')

# Elipse rotada y trasladada
```

```
ax.plot(x_rot_tras, y_rot_tras, 'r-', lw=2, label='Elipse rotada y
trasladada')

# Ejes y centro de traslación
ax.axhline(0, color='gray', lw=0.5)
ax.axvline(0, color='gray', lw=0.5)
ax.scatter(Tx, Ty, color='k', marker='x', label='Centro trasladado')

ax.legend()
plt.title("Rotación y Traslación de una Elipse")
plt.grid(True)
plt.show()
```

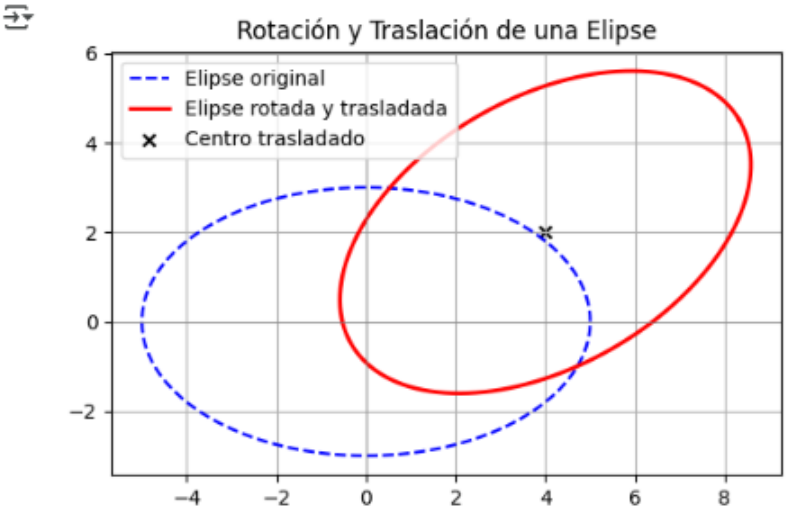


Figura 36. Código y resultado para rotar y trasladar una elipse.

Ejercicio Propuesto 2.1.1.7.

- Modificar el ángulo de giro de la elipse del ejercicio anterior (Ejercicio 2.1.1.6.).
- Reducir el tamaño de la elipse.
- Trasladar a otra distancia la elipse.

```
# Ejercicio Propuesto 2.1.1.7.
import numpy as np
import matplotlib.pyplot as plt

# --- Parámetros modificados de la elipse ---
a = 3   # semieje mayor reducido
b = 2   # semieje menor reducido

theta = np.linspace(0, 2*np.pi, 300)

# Elipse original centrada en el origen
x = a * np.cos(theta)
y = b * np.sin(theta)
```

```
# --- Transformaciones ---
# Traslación modificada
Tx, Ty = -2, 5   # nueva traslación

# Rotación modificada
phi = np.deg2rad(60)  # nuevo ángulo de rotación en grados
R = np.array([[np.cos(phi), -np.sin(phi)],
              [np.sin(phi),  np.cos(phi)]])

# Aplicar rotación a cada punto
puntos = np.vstack((x, y))
puntos_rot = R @ puntos

# Aplicar traslación
x_rot_tras = puntos_rot[0, :] + Tx
y_rot_tras = puntos_rot[1, :] + Ty

# --- Graficar ---
fig, ax = plt.subplots(figsize=(6,6))
ax.set_aspect('equal')

# Elipse original (más pequeña)
ax.plot(x, y, 'b--', label='Elipse original reducida')

# Elipse rotada y trasladada
ax.plot(x_rot_tras, y_rot_tras, 'r-', lw=2, label='Elipse rotada y
trasladada')

# Ejes y centro de traslación
ax.axhline(0, color='gray', lw=0.5)
ax.axvline(0, color='gray', lw=0.5)
ax.scatter(Tx, Ty, color='k', marker='x', label='Centro trasladado')

ax.legend()
plt.title("Ejercicio Propuesto 2.1.1.7 - Rotación, Reducción y
Traslación de Elipse")
plt.grid(True)
plt.show()
```

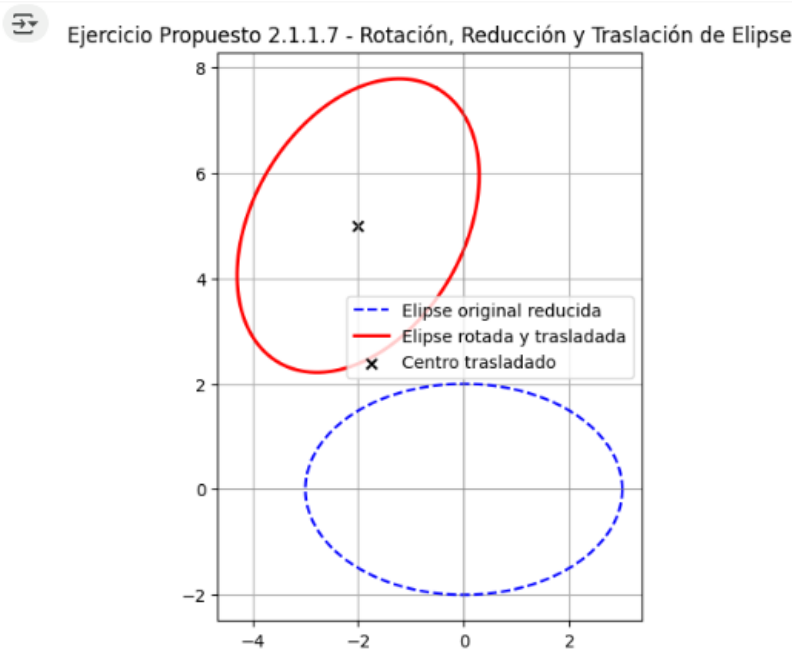


Figura 36.2. Código y resultado para rotar y trasladar una elipse.

Ejercicio 2.1.2.2. Diseño de una hipérbola script en Python para construir una hipérbola mediante su forma canónica y ecuaciones paramétricas.

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

# Parámetros de la hipérbola
a = 5   # semieje real
b = 3   # semieje imaginario

# Rango de ángulos (para rama derecha e izquierda)
theta = np.linspace(-2, 2, 400)

# Ecuaciones paramétricas de la hipérbola (rama derecha)
x1 = a * np.cosh(theta)
y1 = b * np.sinh(theta)

# Ecuaciones paramétricas de la hipérbola (rama izquierda)
x2 = -a * np.cosh(theta)
y2 = b * np.sinh(theta)

# Focos
d = np.sqrt(a**2 + b**2)
f1 = (d, 0)
f2 = (-d, 0)

# Gráfica
fig, ax = plt.subplots()
ax.plot(x1, y1, 'b', label='Rama derecha')
```

```
ax.plot(x2, y2, 'r', label='Rama izquierda')

# Focos
ax.scatter(*f1, color='k', marker='x', label='Foco F1')
ax.scatter(*f2, color='k', marker='x', label='Foco F2')

# Configuración
title = "Construcción de la Hipérbola (Forma canónica)"
ax.set_title(title)
ax.axhline(0, color='gray', lw=0.5)
ax.axvline(0, color='gray', lw=0.5)
ax.set_aspect('equal')
ax.legend()
plt.grid(True)
plt.show()
```

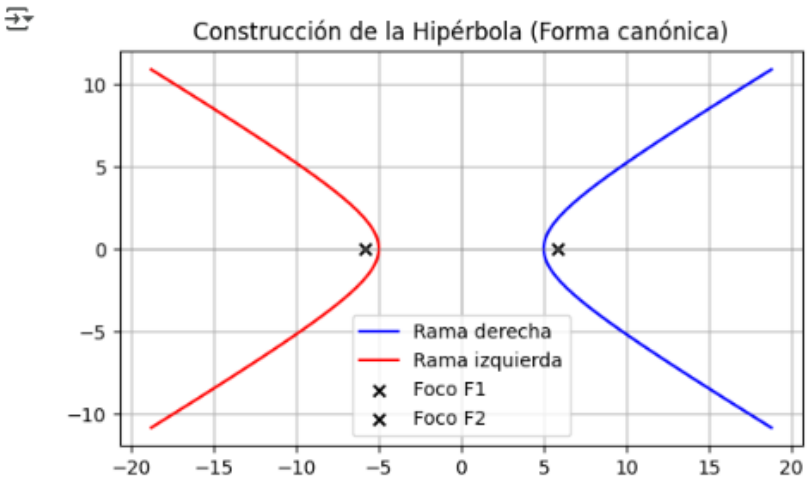


Figura 39. Código y salida de la hipérbola resultado.

Ejercicio Propuesto 2.1.2.3.

• Modificar el script anterior (del ejercicio 2.1.2.2) para crear una **hipérbola** con semieje a = 60 y marcar sus **focos**. Exportar el resultado a **DXF** para abrir en AutoCAD.
• Añadir una rotación y traslación de la hipérbola.

```
# Ejercicio Propuesto 2.1.2.3.
!pip install ezdxf

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

# Parámetros de la hipérbola
```

```
a = 60   # semieje real
b = 30   # semieje imaginario

# Rango de valores para el parámetro
theta = np.linspace(-2, 2, 400)

# Ecuaciones paramétricas de la hipérbola (rama derecha)
x1 = a * np.cosh(theta)
y1 = b * np.sinh(theta)

# Ecuaciones paramétricas de la hipérbola (rama izquierda)
x2 = -a * np.cosh(theta)
y2 = b * np.sinh(theta)

# Cálculo de focos
d = np.sqrt(a**2 + b**2)
f1 = (d, 0)
f2 = (-d, 0)

# Rotación y traslación
phi = np.radians(30)  # ángulo de rotación (30°)
Tx, Ty = 20, 40       # traslación

R = np.array([[np.cos(phi), -np.sin(phi)],
              [np.sin(phi),  np.cos(phi)]])

# Aplicar transformación a los puntos de la hipérbola
coords1 = np.dot(R, np.vstack((x1, y1))) + np.array([[Tx], [Ty]])
coords2 = np.dot(R, np.vstack((x2, y2))) + np.array([[Tx], [Ty]])

# Transformar focos
f1_rot = np.dot(R, np.array([[f1[0]], [f1[1]]])) + np.array([[Tx],
[Ty]])
f2_rot = np.dot(R, np.array([[f2[0]], [f2[1]]])) + np.array([[Tx],
[Ty]])

# Graficar
fig, ax = plt.subplots()
ax.plot(coords1[0], coords1[1], 'b', label='Rama derecha')
ax.plot(coords2[0], coords2[1], 'r', label='Rama izquierda')
ax.scatter(f1_rot[0], f1_rot[1], color='k', marker='x', label='Foco
F1')
ax.scatter(f2_rot[0], f2_rot[1], color='k', marker='x', label='Foco
F2')

ax.set_title("Hipérbola con rotación y traslación")
ax.axhline(0, color='gray', lw=0.5)
ax.axvline(0, color='gray', lw=0.5)
ax.set_aspect('equal')
```

```
ax.legend()
plt.grid(True)
plt.show()

# Exportar a DXF
doc = ezdxf.new()
modelspace = doc.modelspace()

# Dibujar ramas de la hipérbola en DXF
for i in range(len(coords1[0]) - 1):
    modelspace.add_line((coords1[0][i], coords1[1][i]),
(coords1[0][i+1], coords1[1][i+1]))
    modelspace.add_line((coords2[0][i], coords2[1][i]),
(coords2[0][i+1], coords2[1][i+1]))

# Dibujar focos
modelspace.add_point((f1_rot[0][0], f1_rot[1][0]))
modelspace.add_point((f2_rot[0][0], f2_rot[1][0]))

# Guardar archivo DXF
doc.saveas("hiperbola.dxf")
```

```
Collecting ezdxf
  Downloading ezdxf-1.4.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.8 kB)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.12/dist-packages (from ezdxf) (3.2.3)
Requirement already satisfied: typing_extensions>=4.6.0 in /usr/local/lib/python3.12/dist-packages (from ezdxf) (4.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from ezdxf) (2.0.2)
Requirement already satisfied: fonttools in /usr/local/lib/python3.12/dist-packages (from ezdxf) (4.59.1)
Downloading ezdxf-1.4.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 40.3 MB/s eta 0:00:00
Installing collected packages: ezdxf
Successfully installed ezdxf-1.4.2
```

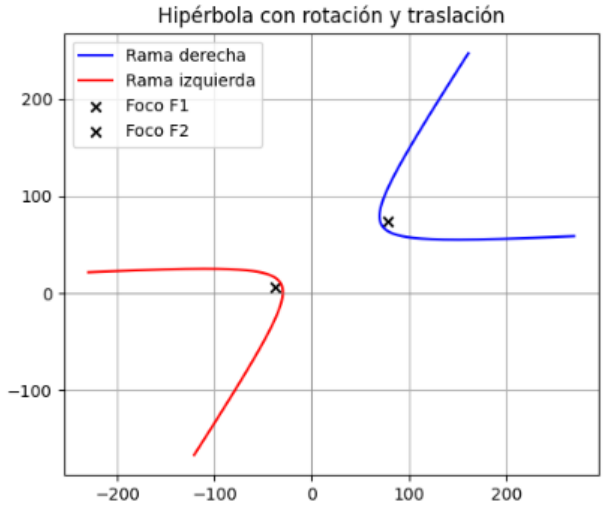


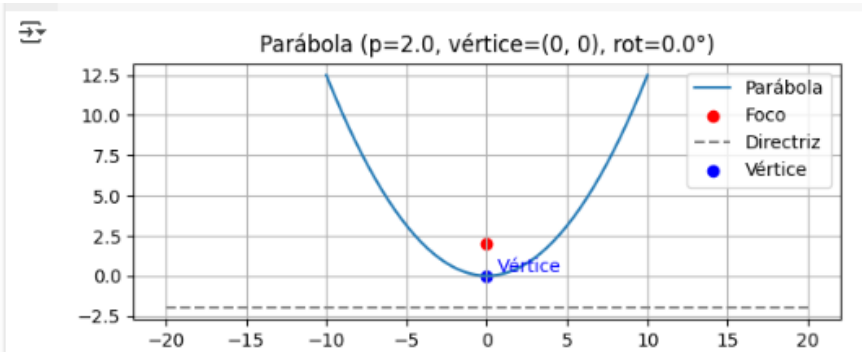
Figura 39.2. Código y salida esperada de hipérbola rotada.

Ejercicio 2.1.3.1. Diseño de una parábola

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

```
def rotate_points(x, y, theta):
    """Rota puntos (x,y) un ángulo theta (rad) alrededor del origen."""
    ct, st = np.cos(theta), np.sin(theta)
    xr = ct * x - st * y
    yr = st * x + ct * y
    return xr, yr

def  plot_parabola(p=1.0,  vertex=(0,0),  theta=0.0,  x_range=(-5,  5),
points=400, ax=None):
    """
    Dibuja una parábola definida por:
        y = (1/(4p)) * x^2    (p > 0, vértice en el origen, eje vertical
hacia arriba)

    Parámetros:
      p        : distancia focal (vértice a foco)
      vertex   : (h,k) coordenadas del vértice
      theta    : rotación en radianes (antihorario)
      x_range  : rango de x para trazar la curva antes de rotar
      points   : número de puntos
      ax       : eje de matplotlib opcional
    """
    if p == 0:
        raise ValueError("p debe ser distinto de cero.")

    # Coordenadas de la parábola antes de rotación y traslación
    x = np.linspace(x_range[0], x_range[1], points)
    y = (x**2) / (4 * p)

    # Foco y directriz (sin rotar)
    foco = np.array([0, p])
    directriz_y = -p  # y constante

    # Rotar la parábola
    xr, yr = rotate_points(x, y, theta)
    # Rotar foco
    xf, yf = rotate_points(foco[0], foco[1], theta)
    # Rotar la directriz (representada como línea horizontal)
    # Generamos un segmento largo
    dir_x = np.linspace(x_range[0]*2, x_range[1]*2, 2)
    dir_y = np.full_like(dir_x, directriz_y)
    dir_xr, dir_yr = rotate_points(dir_x, dir_y, theta)

    # Trasladar todo al vértice
    h, k = vertex
    xr += h; yr += k
    xf += h; yf += k
    dir_xr += h; dir_yr += k
```

```
    if ax is None:
        fig, ax = plt.subplots(figsize=(7,7))

    ax.plot(xr, yr, label='Parábola')
    ax.scatter([xf], [yf], color='red', label='Foco')
    ax.plot(dir_xr, dir_yr, '--', color='gray', label='Directriz')

    # Marcar vértice
    ax.scatter([h], [k], color='blue', label='Vértice')
    ax.text(h, k, "  Vértice", va='bottom', color='blue')

    ax.set_aspect('equal', adjustable='box')
    ax.grid(True)
    ax.legend()
            ax.set_title(f"Parábola      (p={p},      vértice={vertex},
rot={np.degrees(theta):.1f}°)")
    plt.show()


# --- EJEMPLOS ---
# 1) Parábola estándar vertical, p=2
plot_parabola(p=2.0, vertex=(0,0), theta=0.0, x_range=(-10, 10))

# 2) Parábola rotada 45° con vértice desplazado
plot_parabola(p=1.5, vertex=(3,-2), theta=np.deg2rad(45), x_range=(-8,
8))
```



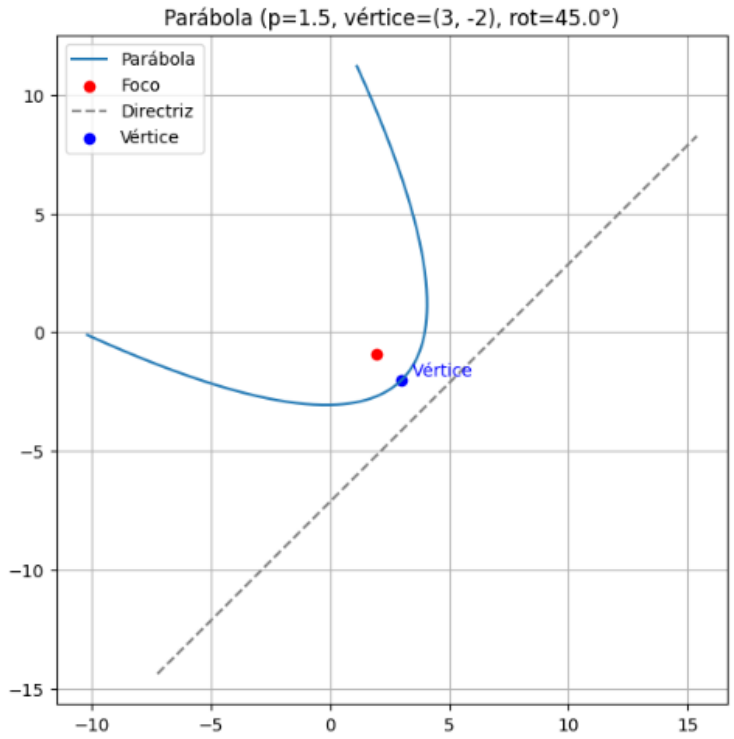


Figura 40. Código y resultado esperado de parábola.

Ejercicio Propuesto 2.1.3.2.

- Modificar el script anterior (Ejercicio 2.1.3.1.) para crear una **parábola** con semieje a = 80, modificar su curvatura y marcar su **foco**. Exportar el resultado a **DXF** para abrir en AutoCAD.
- Añadir una rotación y traslación de la parábola.

```
# Ejercicio Propuesto 2.1.3.2.

!pip install ezdxf

import numpy as np
import matplotlib.pyplot as plt

def rotate_points(x, y, theta):
    """Rota puntos (x,y) un ángulo theta (rad) alrededor del
origen."""
    ct, st = np.cos(theta), np.sin(theta)
    xr = ct * x - st * y
    yr = st * x + ct * y
    return xr, yr

def plot_parabola(p=1.0, vertex=(0,0), theta=0.0, x_range=(-10,10),
points=400, ax=None, color='b', label='Parábola'):
    """
```

```
    Dibuja parábola: y = (1/(4p)) * x^2
    - p > 0: abre hacia arriba
    - p < 0: abre hacia abajo
    """
    x = np.linspace(x_range[0], x_range[1], points)
    y = (x**2) / (4*p)   # cambia concavidad con el signo de p

    # Foco y directriz
    foco = np.array([0, p])
    directriz_y = -p
    dir_x = np.linspace(x_range[0]*2, x_range[1]*2, 2)
    dir_y = np.full_like(dir_x, directriz_y)

    # Rotar
    xr, yr = rotate_points(x, y, theta)
    xf, yf = rotate_points(foco[0], foco[1], theta)
    dir_xr, dir_yr = rotate_points(dir_x, dir_y, theta)

    # Trasladar
    h, k = vertex
    xr += h; yr += k
    xf += h; yf += k
    dir_xr += h; dir_yr += k

    # Graficar
    if ax is None:
        fig, ax = plt.subplots(figsize=(7,7))
    ax.plot(xr, yr, color=color, label=label)
    ax.scatter([xf], [yf], color='red', s=40, label='Foco')
    ax.plot(dir_xr, dir_yr, '--', color='gray', label='Directriz')
    ax.scatter([h],[k], color='black', marker='x', label='Vértice')

    ax.set_aspect('equal', adjustable='box')
    ax.grid(True)
    ax.legend()

# --- EJEMPLOS ---
fig, ax = plt.subplots(figsize=(7,7))

# Parábola con p>0 (abre hacia arriba)
plot_parabola(p=5, vertex=(0,0), theta=0, x_range=(-20,20), ax=ax,
color='blue', label='Convexa')

# Parábola con p<0 (abre hacia abajo)
plot_parabola(p=-5, vertex=(0,-30), theta=0, x_range=(-20,20), ax=ax,
color='green', label='Cóncava')

ax.set_title("Parábolas con distinta concavidad")
plt.show()
```

```
Collecting ezdxf
  Downloading ezdxf-1.4.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.8 kB)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.12/dist-packages (from ezdxf) (3.2.3)
Requirement already satisfied: typing_extensions>=4.6.0 in /usr/local/lib/python3.12/dist-packages (from ezdxf) (4.15.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from ezdxf) (2.0.2)
Requirement already satisfied: fonttools in /usr/local/lib/python3.12/dist-packages (from ezdxf) (4.59.1)
Downloading ezdxf-1.4.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 29.9 MB/s eta 0:00:00
Installing collected packages: ezdxf
Successfully installed ezdxf-1.4.2
```

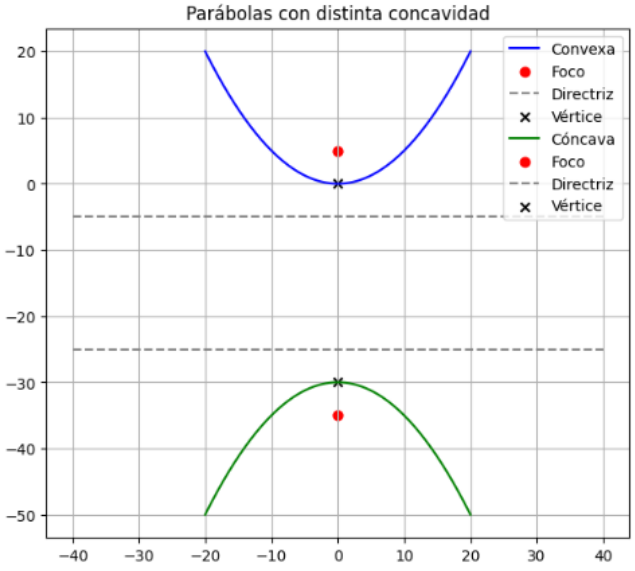


Figura 40.2. Código y resultado esperado de parábola modificada.

Ejercicio Propuesto 2.2.4.

- **Pide o define las coordenadas de los vértices de un triángulo.**
- **Calcula las rectas de las alturas.**
- **Obtiene el ortocentro como intersección de dos de ellas.**
- **Dibuja el triángulo, las alturas y el ortocentro.**

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

def line_from_points(p1, p2):
    """Devuelve coeficientes (A,B,C) de la recta Ax + By + C = 0 que
pasa por p1 y p2."""
    x1, y1 = p1
    x2, y2 = p2
    A = y1 - y2
    B = x2 - x1
    C = x1*y2 - x2*y1
    return A, B, C

def intersection(L1, L2):
    """Devuelve la intersección de dos rectas dadas por coeficientes
(A,B,C)."""
    A1, B1, C1 = L1
```

```
    A2, B2, C2 = L2
    det = A1*B2 - A2*B1
    if det == 0:
        return None  # Son paralelas
    x = (B2*(-C1) - B1*(-C2)) / det
    y = (A1*(-C2) - A2*(-C1)) / det
    return (x, y)

def ortocentro(A, B, C):
    """
    Calcula el ortocentro de un triángulo con vértices A, B, C.
    """
    # Recta BC
    L_BC = line_from_points(B, C)
    # Recta AC
    L_AC = line_from_points(A, C)
    # Recta AB
    L_AB = line_from_points(A, B)

    # Altura desde A -> perpendicular a BC que pasa por A
    A1, B1, _ = L_BC
    altura_A = (B1, -A1, -(B1*A[0] + -A1*A[1]))

    # Altura desde B -> perpendicular a AC que pasa por B
    A2, B2, _ = L_AC
    altura_B = (B2, -A2, -(B2*B[0] + -A2*B[1]))

    # Intersección de dos alturas
    H = intersection(altura_A, altura_B)
    return H, altura_A, altura_B

# --- Ejemplo ---
A = (1, 1)
B = (7, 2)
C = (3, 6)

H, alt_A, alt_B = ortocentro(A, B, C)

# --- Gráfica ---
fig, ax = plt.subplots(figsize=(6,6))

# Triángulo
triangle_x = [A[0], B[0], C[0], A[0]]
triangle_y = [A[1], B[1], C[1], A[1]]
ax.plot(triangle_x, triangle_y, 'b-', lw=2, label='Triángulo')

# Dibujar alturas
def draw_line(line, x_range, style, label):
    A, B, C = line
```

```
    x_vals = np.linspace(x_range[0], x_range[1], 100)
    y_vals = (-A*x_vals - C) / B
    ax.plot(x_vals, y_vals, style, label=label)

draw_line(alt_A, (0,8), 'r--', 'Altura desde A')
draw_line(alt_B, (0,8), 'g--', 'Altura desde B')

# Ortocentro
ax.scatter(*H, color='purple', zorder=5, label='Ortocentro H')

# Vértices
for P, name in zip([A,B,C], ['A','B','C']):
    ax.scatter(*P, color='black')
    ax.text(P[0]+0.1, P[1]+0.1, name, fontsize=12)

ax.set_aspect('equal')
ax.grid(True)
ax.legend()
ax.set_title("Triángulo y su Ortocentro")
plt.show()
```

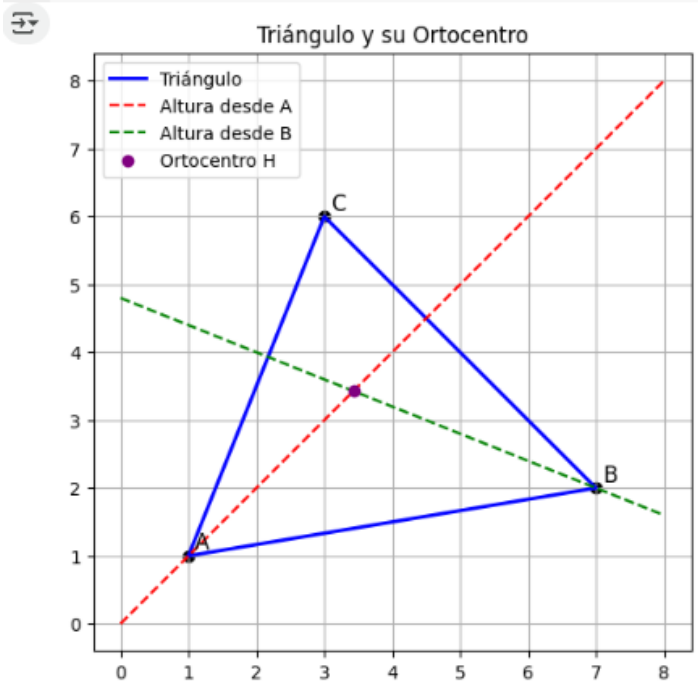


Figura 43.2. Código y salida de triángulo y su ortocentro.

Ejercicio 2.3.3. Diseño de un engranaje en Python

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

# ---------- utilidades geométricas ----------
def involute_xy(rb, t):
```

```
    """Curva involuta de círculo: x=rb(cos t + t sin t), y=rb(sin t -
t cos t)."""
    x = rb * (np.cos(t) + t * np.sin(t))
    y = rb * (np.sin(t) - t * np.cos(t))
    return x, y

def rotate(x, y, ang):
    ca, sa = np.cos(ang), np.sin(ang)
    return ca*x - sa*y, sa*x + ca*y

def arc_xy(r, th1, th2, n=50):
    """Arco CCW entre th1 y th2 (rad)."""
    if th2 < th1:
        th2 += 2*np.pi
    th = np.linspace(th1, th2, n)
    return r*np.cos(th), r*np.sin(th)

# ---------- construcción de un diente involuta ----------
def tooth_segments(z, m, alpha_deg=20.0, n_iv=80):
    """
    Devuelve los segmentos (arrays x,y) de UN diente (CCW):
      - flanco derecho (de raíz a punta)
      - arco de cabeza (addendum)
      - flanco izquierdo (de punta a raíz)
      - arco de pie (root/base)
    """
    alpha = np.deg2rad(alpha_deg)
    rp = 0.5*m*z                # radio primitivo
    rb = rp*np.cos(alpha)       # radio base
    ra = rp + m                 # radio de cabeza (addendum)
    rr = max(rp - 1.25*m, 0.0)  # radio de pie (dedendum aprox. ISO)

    inv = np.tan(alpha) - alpha                          # función
involuta
    half_tooth = np.pi/(2*z) + inv                       # desfase
angular de cada flanco
    r_start = max(rr, rb)                                # si rr < rb,
se empieza en la base

    # parámetro t tal que r = rb*sqrt(1+t^2)
    t_for_r = lambda r: np.sqrt(max((r/rb)**2 - 1.0, 0.0))
    t0 = 0.0 if r_start == rb else t_for_r(r_start)
    ta = t_for_r(ra)
    t = np.linspace(t0, ta, n_iv)

    # involuta base (apuntando al +X), flanco derecho e izquierdo
    x0, y0 = involute_xy(rb, t)
    xr, yr = rotate(x0, y0, +half_tooth)                # flanco
derecho
```

```
    xl, yl = rotate(x0, -y0, -half_tooth)              # flanco
izquierdo (espejo + rotación)

    # puntas (en el círculo de cabeza)
    th_r_tip = np.arctan2(yr[-1], xr[-1])
    th_l_tip = np.arctan2(yl[-1], xl[-1])
    xa_head, ya_head = arc_xy(ra, th_r_tip, th_l_tip, n=40)

    # arranques en raíz/base
    th_r_root = np.arctan2(yr[0], xr[0])
    th_l_root = np.arctan2(yl[0], xl[0])
    xa_root, ya_root = arc_xy(r_start, th_l_root, th_r_root, n=35)

    segs = [
        (xr, yr),                  # flanco derecho (subiendo)
        (xa_head, ya_head),        # arco de cabeza
        (xl[::-1], yl[::-1]),      # flanco izquierdo (bajando)
        (xa_root, ya_root),        # arco de pie/base
    ]
    return segs, (rb, rp, ra, rr)

# ---------- dibujo de todo el engranaje ----------
def draw_involute_gear(z=24, m=2.0, alpha_deg=20.0, bore_diam=10.0,
show_guides=False):
    segs_one, (rb, rp, ra, rr) = tooth_segments(z, m, alpha_deg)
    fig = plt.figure(figsize=(7,7))
    ax = plt.gca()

    # dibuja cada diente por rotación de sus segmentos
    for i in range(z):
        ang = 2*np.pi*i/z
        for (x, y) in segs_one:
            xs, ys = rotate(x, y, ang)
            ax.plot(xs, ys, linewidth=1.8)

    # guías (opcionales): primitiva, base, raíz y agujero
    th = np.linspace(0, 2*np.pi, 400)
    if show_guides:
        ax.plot(rp*np.cos(th), rp*np.sin(th), linewidth=0.6)
        ax.plot(rb*np.cos(th), rb*np.sin(th), linewidth=0.6)
        ax.plot(rr*np.cos(th), rr*np.sin(th), linewidth=0.6)

    # agujero central
    if bore_diam > 0:
        r = bore_diam/2.0
        ax.plot(r*np.cos(th), r*np.sin(th), linewidth=1.8)

    ax.set_aspect('equal', 'box')
    ax.set_xticks([]); ax.set_yticks([])
```

```
    ax.set_title(f"Engranaje involuta (z={z}, m={m}, α={alpha_deg}°)")
    plt.show()

# ejemplo de uso
draw_involute_gear(z=24, m=2.0, alpha_deg=20.0, bore_diam=12.0,
show_guides=False)
```

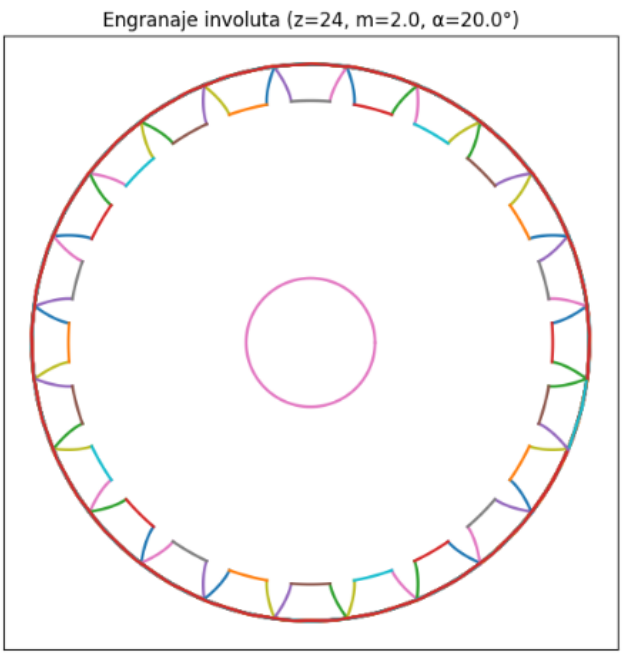


Figura 48. Código y salida esperada del engranaje.

Ejercicio Propuesto 2.3.4.

- **Modificar el código anterior del Ejercicio 2.3.3. para conseguir que el engranaje tenga 8 dientes.**

```
!pip install cadquery

import cadquery as cq
import numpy as np
import matplotlib.pyplot as plt

# ---------- utilidades geométricas ----------
def involute_xy(rb, t):
    """Curva involuta de círculo: x=rb(cos t + t sin t), y=rb(sin t -
t cos t)."""
    x = rb * (np.cos(t) + t * np.sin(t))
    y = rb * (np.sin(t) - t * np.cos(t))
    return x, y

def rotate(x, y, ang):
    ca, sa = np.cos(ang), np.sin(ang)
```

```
    return ca*x - sa*y, sa*x + ca*y

def arc_xy(r, th1, th2, n=50):
    """Arco CCW entre th1 y th2 (rad)."""
    if th2 < th1:
        th2 += 2*np.pi
    th = np.linspace(th1, th2, n)
    return r*np.cos(th), r*np.sin(th)

# ---------- construcción de un diente involuta ----------
def tooth_segments(z, m, alpha_deg=20.0, n_iv=80):
    alpha = np.deg2rad(alpha_deg)
    rp = 0.5*m*z                  # radio primitivo
    rb = rp*np.cos(alpha)         # radio base
    ra = rp + m                   # radio de cabeza (addendum)
    rr = max(rp - 1.25*m, 0.0)    # radio de pie (dedendum aprox.)

    inv = np.tan(alpha) - alpha
    half_tooth = np.pi/(2*z) + inv
    r_start = max(rr, rb)

    # parámetro t tal que r = rb*sqrt(1+t^2)
    t_for_r = lambda r: np.sqrt(max((r/rb)**2 - 1.0, 0.0))
    t0 = 0.0 if r_start == rb else t_for_r(r_start)
    ta = t_for_r(ra)
    t = np.linspace(t0, ta, n_iv)

    # involuta base (apuntando al +X), flanco derecho e izquierdo
    x0, y0 = involute_xy(rb, t)
    xr, yr = rotate(x0, y0, +half_tooth)               # flanco
derecho
    xl, yl = rotate(x0, -y0, -half_tooth)              # flanco
izquierdo (espejo + rotación)

    # puntas (en el círculo de cabeza)
    th_r_tip = np.arctan2(yr[-1], xr[-1])
    th_l_tip = np.arctan2(yl[-1], xl[-1])
    xa_head, ya_head = arc_xy(ra, th_r_tip, th_l_tip, n=40)

    # arranques en raíz/base
    th_r_root = np.arctan2(yr[0], xr[0])
    th_l_root = np.arctan2(yl[0], xl[0])
    xa_root, ya_root = arc_xy(r_start, th_l_root, th_r_root, n=35)

    segs = [
        (xr, yr),                 # flanco derecho (subiendo)
        (xa_head, ya_head),       # arco de cabeza
        (xl[::-1], yl[::-1]),     # flanco izquierdo (bajando)
        (xa_root, ya_root),       # arco de pie/base
```

```
    ]
    return segs, (rb, rp, ra, rr)

# ---------- dibujo de todo el engranaje ----------
def draw_involute_gear(z=8, m=2.0, alpha_deg=20.0, bore_diam=10.0,
show_guides=False):
    segs_one, (rb, rp, ra, rr) = tooth_segments(z, m, alpha_deg)
    fig = plt.figure(figsize=(7,7))
    ax = plt.gca()

    # dibuja cada diente por rotación de sus segmentos
    for i in range(z):
        ang = 2*np.pi*i/z
        for (x, y) in segs_one:
            xs, ys = rotate(x, y, ang)
            ax.plot(xs, ys, linewidth=1.8)

    # guías (opcionales)
    th = np.linspace(0, 2*np.pi, 400)
    if show_guides:
        ax.plot(rp*np.cos(th), rp*np.sin(th), linewidth=0.6,
linestyle="--", label="Pitch circle")
        ax.plot(rb*np.cos(th), rb*np.sin(th), linewidth=0.6,
linestyle="--", label="Base circle")
        ax.plot(rr*np.cos(th), rr*np.sin(th), linewidth=0.6,
linestyle="--", label="Root circle")

    # agujero central
    if bore_diam > 0:
        r = bore_diam/2.0
        ax.plot(r*np.cos(th), r*np.sin(th), linewidth=1.8)

    ax.set_aspect('equal', 'box')
    ax.set_xticks([]); ax.set_yticks([])
    ax.set_title(f"Engranaje involuta (z={z}, m={m}, α={alpha_deg}°)")
    plt.show()

# ------- ejecutar -------
draw_involute_gear(z=8, m=2.0, alpha_deg=20.0, bore_diam=12.0,
show_guides=True)
```

```
Collecting cadquery
  Downloading cadquery-2.5.2-py3-none-any.whl.metadata (16 kB)
Collecting cadquery-ocp<7.8,>=7.7.0 (from cadquery)
  Downloading cadquery_ocp-7.7.2-cp312-cp312-manylinux_2_35_x86_64.whl.metadata (1.6 kB)
Collecting ezdxf (from cadquery)
  Downloading ezdxf-1.4.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.8 kB)
Collecting multimethod<2.0,>=1.11 (from cadquery)
  Downloading multimethod-1.12-py3-none-any.whl.metadata (9.6 kB)
Collecting nlopt<3.0,>=2.9.0 (from cadquery)
  Downloading nlopt-2.9.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.1 kB)
Collecting typish (from cadquery)
  Downloading typish-1.9.3-py3-none-any.whl.metadata (7.2 kB)
Collecting casadi (from cadquery)
  Downloading casadi-3.7.1-cp312-none-manylinux2014_x86_64.whl.metadata (2.2 kB)
Collecting path (from cadquery)
  Downloading path-17.1.1-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: numpy<3,>=2 in /usr/local/lib/python3.12/dist-packages (from nlopt<3.0,>=2.9.0->cadquery) (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.12/dist-packages (from ezdxf->cadquery) (3.2.3)
Requirement already satisfied: typing_extensions>=4.6.0 in /usr/local/lib/python3.12/dist-packages (from ezdxf->cadquery) (4.15.0)
Requirement already satisfied: fonttools in /usr/local/lib/python3.12/dist-packages (from ezdxf->cadquery) (4.59.1)
Downloading cadquery-2.5.2-py3-none-any.whl (163 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 163.9/163.9 kB 4.2 MB/s eta 0:00:00
Downloading cadquery_ocp-7.7.2-cp312-cp312-manylinux_2_35_x86_64.whl (143.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.0/143.0 MB 6.3 MB/s eta 0:00:00
Downloading multimethod-1.12-py3-none-any.whl (10 kB)
Downloading nlopt-2.9.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (438 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 438.6/438.6 kB 24.6 MB/s eta 0:00:00
Downloading casadi-3.7.1-cp312-none-manylinux2014_x86_64.whl (75.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 MB 7.6 MB/s eta 0:00:00
Downloading ezdxf-1.4.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 53.4 MB/s eta 0:00:00
Downloading path-17.1.1-py3-none-any.whl (23 kB)
Downloading typish-1.9.3-py3-none-any.whl (45 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.1/45.1 kB 2.6 MB/s eta 0:00:00
Installing collected packages: typish, cadquery-ocp, path, nlopt, multimethod, ezdxf, casadi, cadquery
Successfully installed cadquery-2.5.2 cadquery-ocp-7.7.2 casadi-3.7.1 ezdxf-1.4.2 multimethod-1.12 nlopt-2.9.1 path-17.1.1 typish-1.9.3
```

Figura 48.2. Código y salida esperada del engranaje propuesto.