Autora: María Inmaculada Rodríguez García Universidad de Cádiz.

Curso OCW Computación Aplicada a la Expresión Gráfica en Ingeniería. Diseño de piezas industriales, figuras y resolución de curvas cónicas en Python, para la Ingeniería Civil e Ingeniería Industrial.

María Inmaculada Rodríguez García

UNIVERSIDAD DE CÁDIZ

2025

Departamentos de Ingeniería Civil e Ingeniería Industrial. Departamento de Informática.

ETSIA/ESI

Descripción general

- **Resumen del curso:**

Este curso OCW fomenta la interdisciplinariedad e introduce al alumnado en la aplicación de la computación para la representación gráfica en el ámbito de la ingeniería, con un enfoque práctico en el uso de Python para el diseño de piezas industriales, figuras geométricas y la resolución de curvas cónicas. Se abordan tanto los fundamentos teóricos de la expresión gráfica como su implementación computacional, permitiendo al estudiante adquirir destrezas en la generación, análisis y exportación de modelos para su uso en entornos CAD y de ingeniería.

A lo largo de los módulos, se desarrollarán ejemplos aplicados a la Ingeniería Civil e Ingeniería Industrial, con especial atención al diseño de elementos constructivos, piezas mecánicas y la resolución geométrica de elipses, parábolas e hipérbolas, integrando métodos matemáticos y de programación para lograr resultados precisos y optimizados.

- **Objetivos generales:**
  1. Desarrollar competencias en el uso de Python para la representación y el diseño gráfico de piezas, figuras y curvas de interés en la ingeniería gráfica.
  2. Integrar conocimientos de geometría, matemáticas y programación para resolver problemas gráficos y exportar resultados a formatos compatibles con software de diseño asistido por ordenador (CAD).
- **Competencias específicas y transversales:**
  Específicas:
  - Comprender y aplicar los fundamentos de la expresión gráfica en ingeniería.
  - Utilizar herramientas de programación para el modelado geométrico y la resolución de problemas gráficos.
  - Generar e interpretar modelos digitales exportables a entornos CAD.

  Transversales:
  - Capacidad de resolución de problemas técnicos mediante el pensamiento computacional.
  - Autonomía en el aprendizaje y gestión de recursos digitales.
  - Comunicación técnica eficaz a través de representaciones gráficas y documentación estructurada.
- **Requisitos previos recomendados:**
  - Conocimientos básicos de geometría y trigonometría.
  - Fundamentos de programación (preferiblemente en Python).
  - Manejo básico de software CAD o familiaridad con entornos de diseño técnico.

Keywords: Expresión gráfica, Ingeniería Civil, Modelado Inteligente, Ingeniería industrial, Python, Informática, Programación, Ciencia de Datos.

Curso OCW Computación Aplicada a la Expresión Gráfica en Ingeniería. Diseño de piezas industriales, figuras y resolución de curvas cónicas en Python, para la Ingeniería Civil e Ingeniería Industrial.

Índice

Ejercicio Extra 3.1.1.: Escribe un script en Python que dibuje un cuadrado de 100 mm de lado utilizando matplotlib.
- *Pistas:* Usa listas o arrays para coordenadas x e y; recuerda cerrar la figura repitiendo el primer punto.

Ejercicio Extra 3.1.2.: Modifica el programa anterior para que el tamaño del cuadrado se pueda introducir por teclado.

Ejercicio Extra 3.1.3.: Crea un script que dibuje un hexágono regular de radio dado y lo exporte a un archivo **DXF** utilizando ezdxf.

Ejercicio Extra 3.1.4.: Programa una función poligono_regular(n, radio) que genere y muestre cualquier polígono regular dado su número de lados y radio circunscrito.

Ejercicio Extra 3.1.5.: Dibuja una elipse de semiejes a = 60 mm y b = 40 mm utilizando sus ecuaciones paramétricas, marcando sus focos en la figura. Solicita al usuario los valores de a y b y genera el archivo **DXF** de la elipse.

Ejercicio Extra 3.1.6.: Representa una parábola con vértice en el origen y ecuación $y^2 = 4px$ para un valor de p dado. Modifica el código para que el vértice y el eje de simetría se puedan desplazar a coordenadas arbitrarias.

Ejercicio Extra 3.1.7.: Dibuja la hipérbola $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$ con $a = 50\,mm$ y $b = 30\,mm$ utilizando un parámetro t en radianes. Programa una versión que trace también las asíntotas de la hipérbola.

Ejercicio Extra 3.1.8.: Diseña una brida circular con 6 taladros equidistantes y genera su archivo DXF. Modela en Python un perfil en "L" con medidas dadas y exporta el dibujo a DXF para su apertura en AutoCAD.

Ejercicio Extra 3.1.9.: Desarrolla un script que permita:
- Dibujar una pieza con al menos una curva cónica y dos figuras poligonales.
- Exportar el resultado a DXF.

Ejercicio Extra 3.1.10.: Desarrollo de un proyecto completo que integre el diseño y exportación de una pieza o figura compleja utilizando Python.

3.2. Archivos solución

Solución Ejercicio 3.1.1.ipynb
Solución Ejercicio 3.1.2.ipynb
Solución Ejercicio 3.1.3.ipynb
Solución Ejercicio 3.1.4.ipynb
Solución Ejercicio 3.1.5.ipynb
Solución Ejercicio 3.1.6.ipynb
Solución Ejercicio 3.1.7.ipynb
Solución Ejercicio 3.1.8.ipynb
Solución Ejercicio 3.1.9.ipynb
Solución Ejercicio 3.1.10.ipynb

ANEXO I. Solución de ejercicios propuestos.

Módulo 1. Introducción a la computación aplicada a la Expresión Gráfica a través del entorno de Python en GoogleColab.

1.1. Fundamentos de geometría para ingeniería

La **Expresión Gráfica en Ingeniería** es el lenguaje que permite transmitir ideas, diseños y procesos de forma visual y precisa.
A lo largo de la historia, esta disciplina ha pasado por tres etapas principales:

1. Dibujo manual – reglas, compás y papel.
2. CAD (Computer-Aided Design) – uso de software como AutoCAD o SolidWorks.
3. CAD programado – generación de diseños mediante código, que permite automatizar y parametrizar los procesos.

Este curso combina:

- Fundamentos de geometría aplicada a ingeniería.
- Programación en Python para la creación de figuras técnicas.
- Exportación a formatos CAD (DXF) para edición profesional.

El objetivo es que el alumnado pueda integrar la programación en su flujo de trabajo de diseño técnico, ganando rapidez, flexibilidad y precisión.

La expresión gráfica engloba técnicas y normas para representar objetos de manera comprensible y estandarizada.
Conceptos clave:

- Dibujo técnico → representación exacta y normalizada de piezas o infraestructuras.
- Vistas ortogonales → alzado, planta, perfil.
- Perspectivas → isométrica, caballera, cónica.
- Escalas → relación entre tamaño real y representado.
- Curvas y superficies → líneas rectas, circunferencias, elipses, parábolas, hipérbolas, etc.

Normas y estándares: se siguen regulaciones como ISO, UNE o ANSI, que garantizan que cualquier ingeniero pueda interpretar los planos sin ambigüedad.

Papel de la programación en el diseño técnico

El dibujo técnico tradicional requiere tiempo y repetición.

Mediante programación, podemos:

- Automatizar la creación de piezas repetitivas.
- Generar geometrías a partir de fórmulas matemáticas.
- Parametrizar dimensiones y formas.
- Integrar cálculos y representación gráfica en un mismo entorno.

Ejemplos reales:

- Generar automáticamente series de agujeros en bridas.
- Crear curvas cónicas para análisis de estructuras o vías.
- Modelar piezas mecánicas a partir de datos experimentales.

Herramientas básicas que usaremos:

- Python (en GoogleColab) como lenguaje de programación.

- NumPy para cálculos numéricos y manejo de vectores/matrices.
- Matplotlib para visualización y trazado de figuras.
- ezdxf para exportar resultados a formato DXF y abrirlos en AutoCAD.

1.2. Introducción a Python en GoogleColab para el diseño gráfico.

La integración de **Python en Google Colab** como herramienta para explorar la **generación computacional de geometría en Ingeniería Gráfica** proporciona un enfoque didáctico, contemporáneo y visual que favorece la motivación y el aprendizaje del alumnado.
Para poder comenzar es necesario seguir los siguientes pasos iniciales.

Las ventajas de trabajar en este entorno es que, al trabajar en la nube, además que no necesitamos instalar el programa Python, siempre tendremos disponible el trabajo y ampliando así la accesibilidad a éstos.

1.2.1. Tener una cuenta de Google.

Si ya usas Gmail, YouTube o Drive, ya la tienes. Si no, crea una:

https://accounts.google.com

1.2.2. Acceder a Google Colab.

Ve a: https://colab.research.google.com

1.2.3. Crear un nuevo notebook.

- Haz clic en **"Archivo"** → **"Nuevo cuaderno"** (o "New notebook").
- Se abrirá una nueva pestaña con una celda lista para escribir código Python.

1.2.4. Escribir y ejecutar código.

Para comenzar a escribir código el Google Colab, por defecto aparece un cuaderno en blanco. Podríamos abrir un cuaderno que ya tengamos en nuestro Drive, crear un nuevo cuaderno o subirlo desde nuestro PC si no necesitamos (Archivo →Cuaderno nuevo en Drive o Abrir cuaderno o Subir cuaderno).

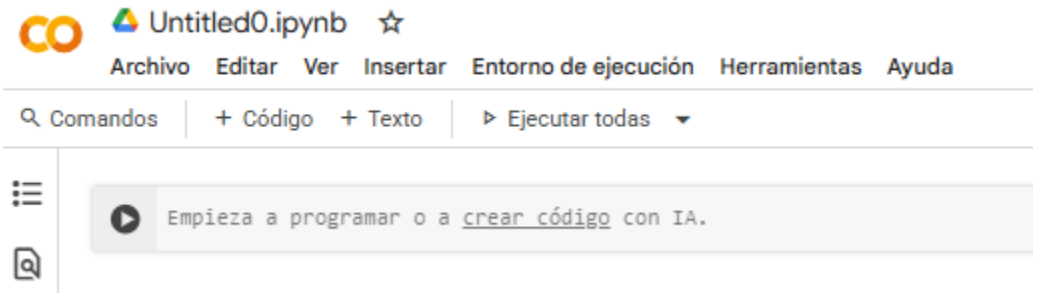


Figura 1. Cuaderno por defecto.
Para escribir los scripts, podemos ir agregando Código en **+ Código**:

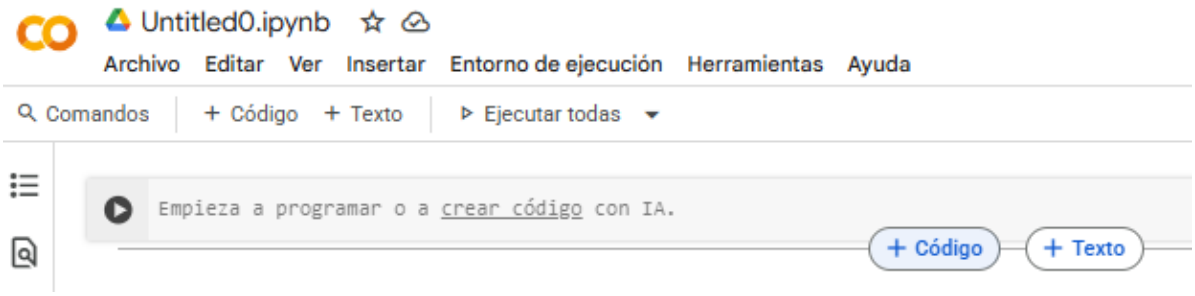


Figura 2. Agregamos código en **+ Código**.

Prueba:

- **Escribe código Python en la celda, por ejemplo 1.4.1 y 1.4.2:**
- Pulsa **Shift + Enter** o haz *clic* en el botón ▶ para ejecutar.

Ejercicio Resuelto 1.2.4.1. Salida por pantalla en Python.

```
print("Hola, Mundo")

Hola, Mundo
```

Ejercicio Resuelto 1.2.4.2. Salida por pantalla en Python con salto de línea \n.

```
print("salida", "de", "datos", "en", "Python", sep="\n") # si no se indica sep="\n" se usa espacio en blanco como separador.

salida
de
datos
en
Python
```

Se recomienda tener la sesión iniciada en Google Drive para poder trabajar en ambos entornos, cargar archivos desde el drive a Google Colab.

Cómo cargar archivos desde Google Drive a Google Colab

1. Monta tu Google Drive en Colab.
2. Ejecuta esta celda al principio: Pulsa **Shift + Enter** o haz *clic* en el botón ▶ para ejecutar.

Ejercicio 1.2.4.3. Monta tu Google Drive en Colab.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

Figura 3. Interfaz de Google Cola. Ejecuta esta celda al principio haciendo *clic* en el botón ▶.

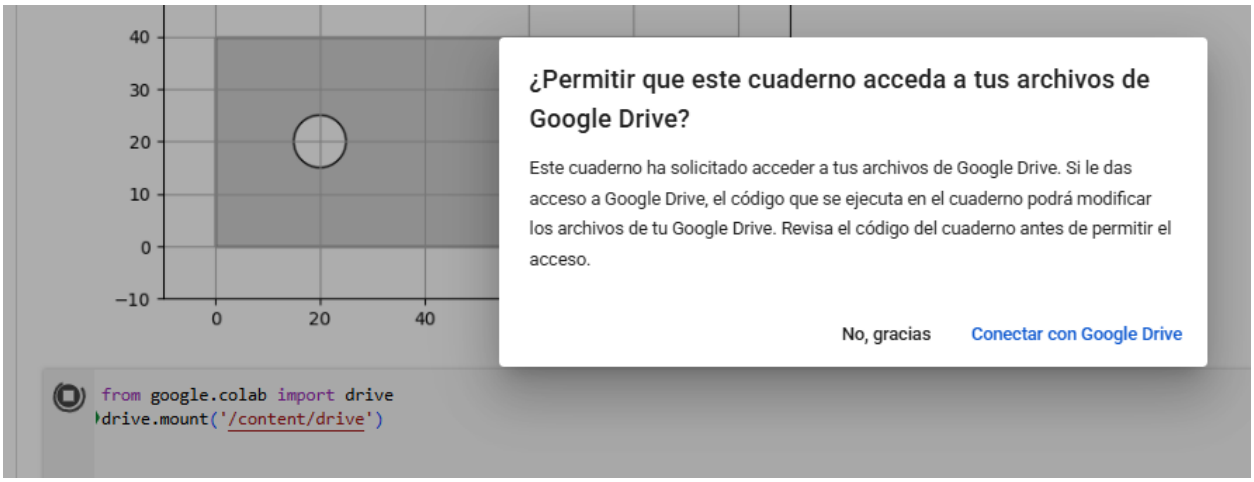


Figura 4. Permitir Conectar con Google Drive.

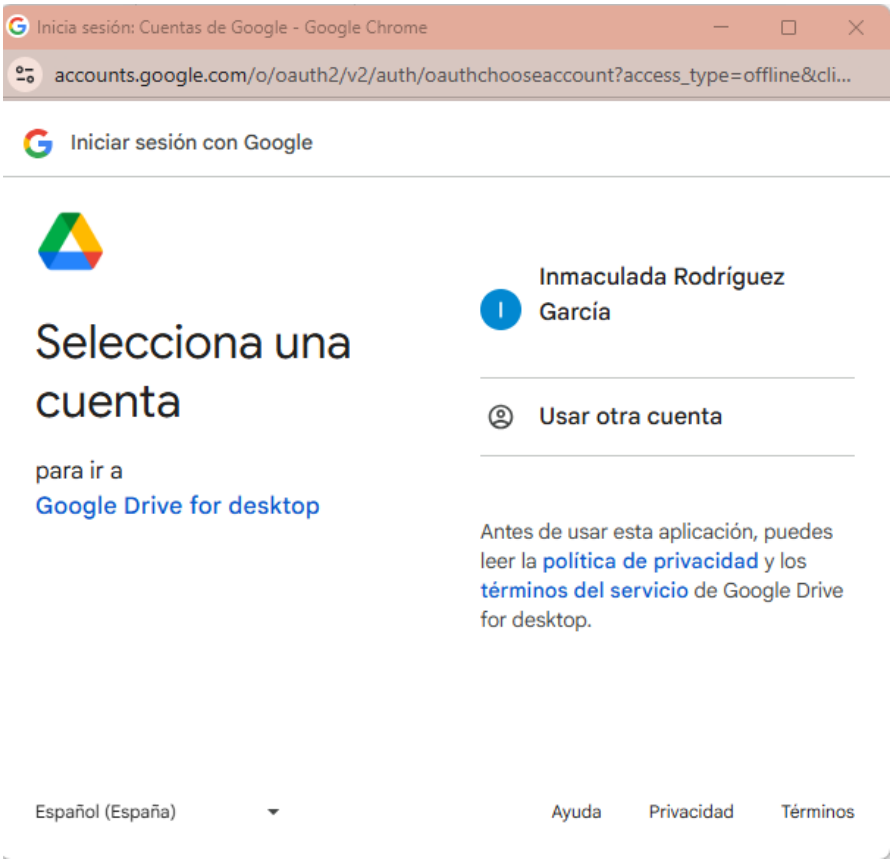


Figura 5. Seleccionar cuenta. **Continuar → continuar**

Accede a tus archivos

Tu Drive estará montado en:

`/content/drive/MyDrive/`

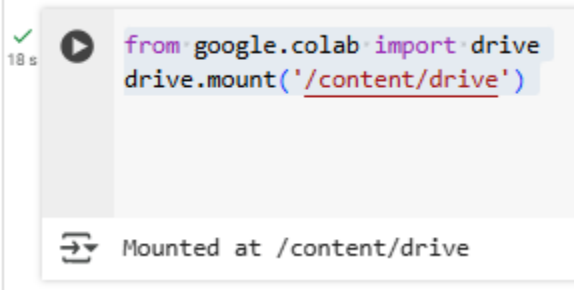


Figura 6. Drive montado en `/content/drive/MyDrive/`

Al ejecutarla:

- Se abrirá un enlace.
- Haz clic, autoriza con tu cuenta de Google (elígela, continuar, continuar). Puede que necesites un código de autorización, en ese caso pégalo en el cuadro que aparece en Colab.

Para acceder a tus archivos, por ejemplo, si tienes un archivo llamado **datos.csv** en la raíz de tu Drive necesitarás la siguiente instrucción:

```python
from google.colab import drive
drive.mount('/content/drive')

ruta = '/content/drive/MyDrive/datos.csv'

import pandas as pd
df = pd.read_csv(ruta)
```

Figura 7. Creamos la ruta hacia el archivo (datos.csv) que queremos leer. En este caso no hemos creado una carpeta donde trabajar, sino que estamos trabajando en la carpeta general del Drive. Esto realmente no se aconseja, hay que crear una carpeta de trabajo.

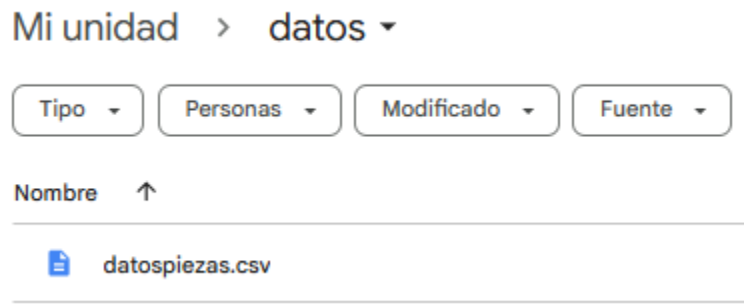


Figura 8. Añadimos archivo con los datos a nuestro Google Drive para poder leerlo. Conviene crear una carpeta dentro de nuestra unidad donde trabajaremos. En este caso la he llamado "datos" (es un *folder*) y dentro de ésta tendremos el archivo de **datospiezas.csv**.

Consejo:

Puedes navegar por tus carpetas con el explorador de archivos de Colab (icono de carpeta a la izquierda) y hacer clic derecho en cualquier archivo → Copiar ruta para pegarla en el código.

1.2.5. Guardar tu notebook.

- Google lo guarda automáticamente en tu Google Drive.
- Puedes cambiarle el nombre arriba donde dice *Untitled0.ipynb*.

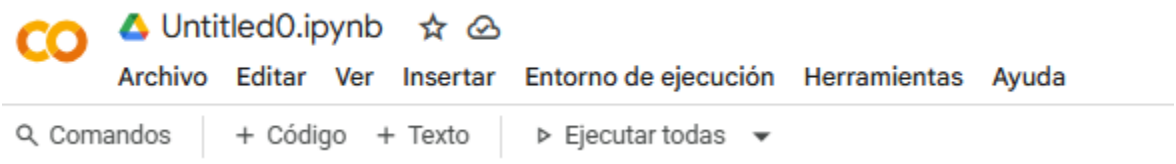


Figura 9. Nombre de archivo Python por defecto.

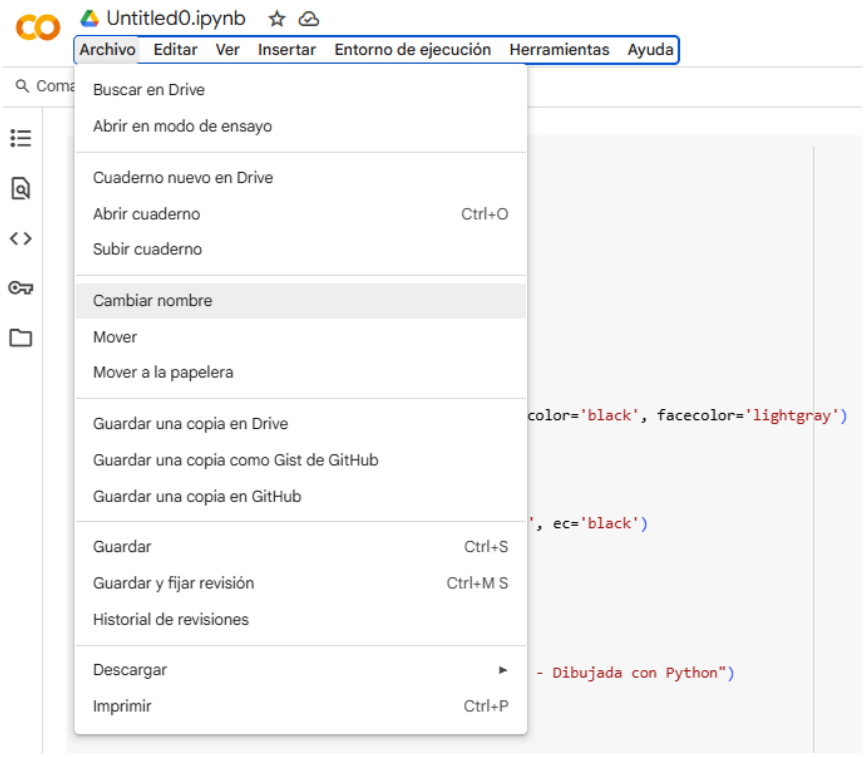


Figura 10. Cambiar nombre al archivo.

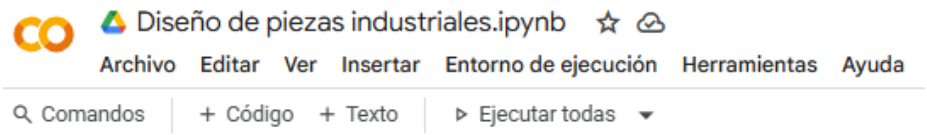


Figura 11. Nombre de archivo *"Diseño de piezas industriales.ipynb"*.

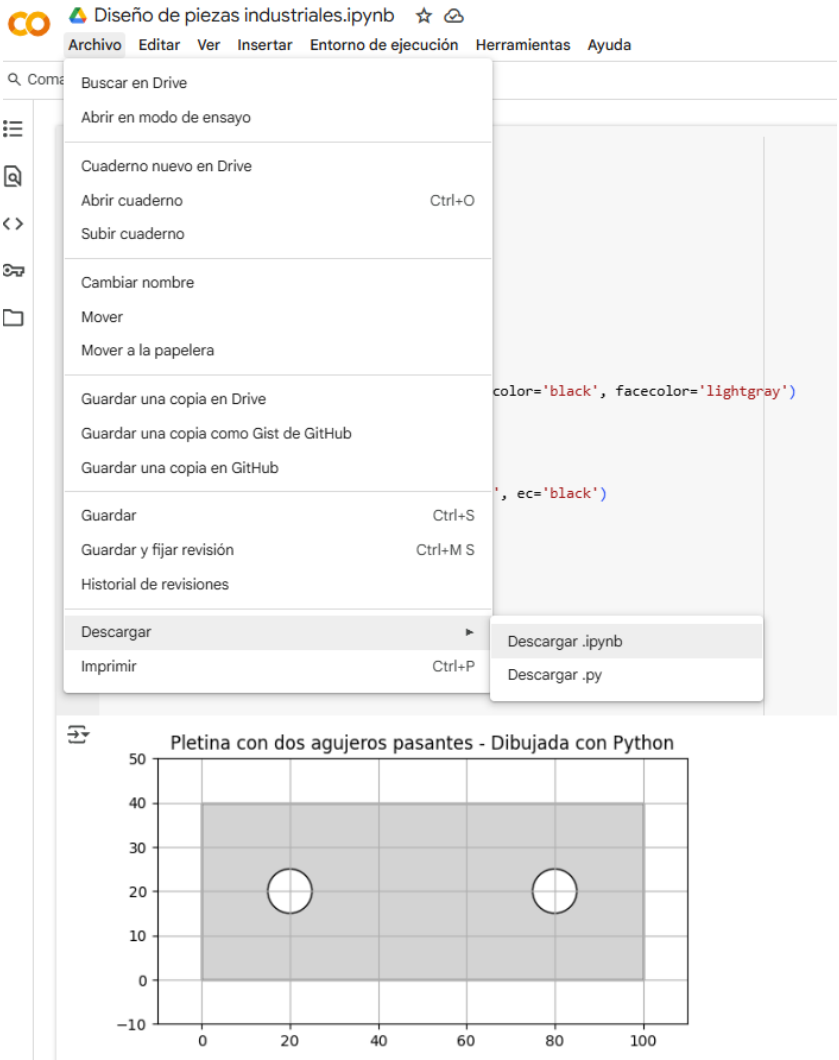


Figura 12. Descargar el archivo Python al PC personal (extensión *.ipnb*).

Guardar un archivo.

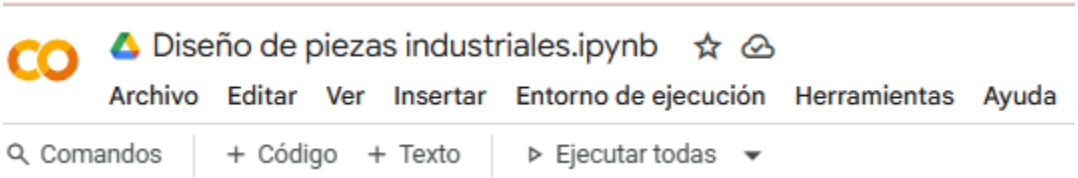


Figura 13. Cuando aparece el símbolo de la nube con un tick dentro, indica que el archivo ha sido guardado correctamente en la nube.

Para evitar perder el trabajo, se recomienda descargar el archivo en formato *.ipynb* para tener una copia de seguridad, incluso incorporarlo a nuestra carpeta de trabajo y cargarlo desde ahí si se nos hubiera cerrado.

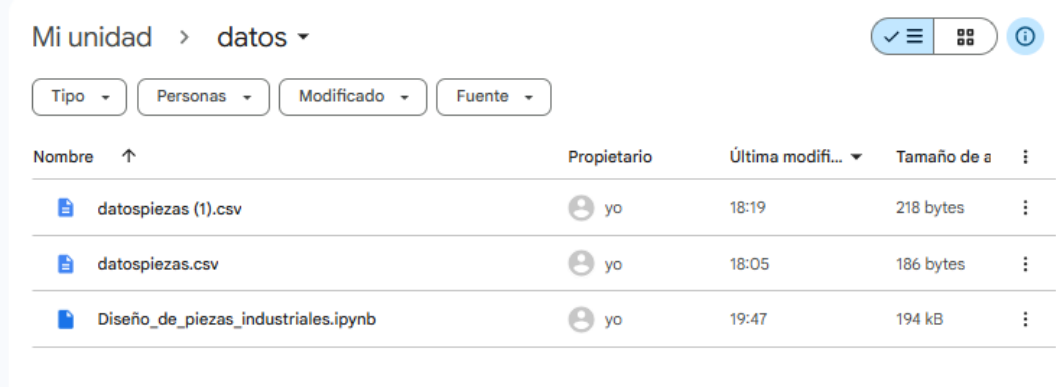


Figura 14. Archivos dentro de la carpeta *datos* de trabajo.

Cuando tardamos mucho en trabajar sobre el cuaderno de GoogleColab éste se desconectará y nos pedirá volver a conectar.

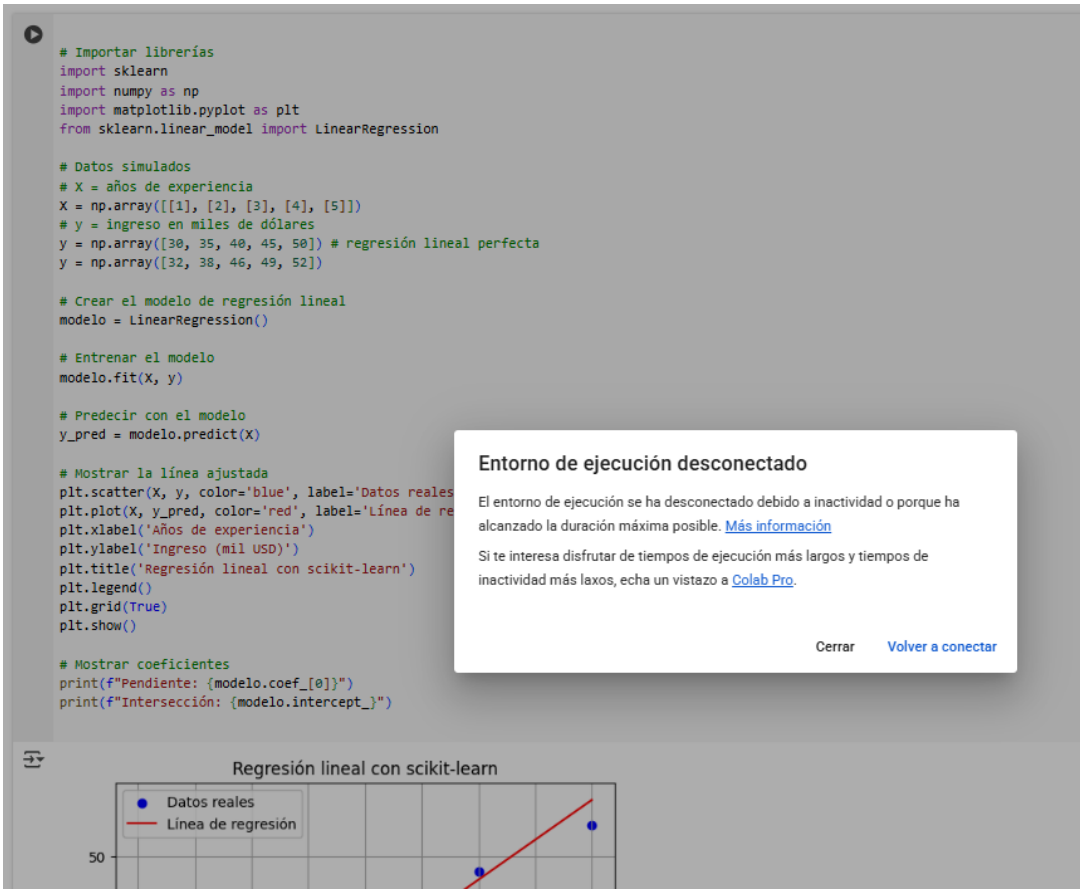


Figura 15. Entorno desconectado. Volver a conectar.

1.2.6. Importar librerías.

Puedes usar muchas librerías como *matplotlib*, *numpy*, *pandas*, *scikit-learn*, etc. directamente:

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

Si necesitas instalar una nueva librería en Jupyter o Google Colab:

!pip install *nombre_librería*

Ejemplo:

!pip install *trimesh*

!pip install *sklearn*

Si necesitas instalar una nueva librería en Python:

pip install *ezdxf*

```
Collecting ezdxf
  Downloading ezdxf-1.4.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.8 kB)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.11/dist-packages (from ezdxf) (3.2.3)
Requirement already satisfied: typing_extensions>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from ezdxf) (4.14.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from ezdxf) (2.0.2)
Requirement already satisfied: fonttools in /usr/local/lib/python3.11/dist-packages (from ezdxf) (4.59.0)
Downloading ezdxf-1.4.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 46.8 MB/s eta 0:00:00
Installing collected packages: ezdxf
Successfully installed ezdxf-1.4.2
```

Figura 16. Al instalar la librería *ezdxf* aparecerá este mensaje.

Puedes instalar una nueva librería la primera vez, al inicio del cuaderno, y ya funcionará para el resto del código tan sólo con importarla (import ezdxf).

Ejercicio Resuelto 1.2.6.1. Importar librerías. Ejemplos

```
# Importar librerías
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import ast
import math
import sklearn

!pip install trimesh
```

¿Para qué se usa cada una de las librerías?

Librerías importadas

- *pandas* as pd
  - **¿Para qué se usa?:** Para leer, organizar y analizar datos en forma de tablas, como los archivos .csv.
  - **Ejemplo típico:** leer un fichero con dimensiones de piezas y convertirlo en una tabla de trabajo.

```
df = pd.read_csv('datospiezas.csv')
```

- *matplotlib.pyplot* as plt
  - **¿Para qué se usa?:** Para crear gráficos y dibujos en 2D, como el contorno de una pieza industrial.
  - **Ejemplo típico:** dibujar una pletina, un círculo (agujero), o una tuerca con plt.Circle y plt.Rectangle.

- *numpy* as np
  - **¿Para qué se usa?:** Para operaciones matemáticas y geométricas más avanzadas. Es muy útil para trabajar con ángulos, arrays y coordenadas.
  - **Ejemplo típico:** calcular los vértices de un octágono con np.cos y np.sin.

- *ast* **(Abstract Syntax Trees)**
  - **¿Para qué se usa?:** Para convertir cadenas de texto en estructuras de datos reales, como listas o diccionarios.
  - **Ejemplo típico:** leer de un archivo CSV una lista de coordenadas en formato texto ("(20, 10), (30, 20)") y transformarla a [(20, 10), (30, 20)] con:

```
coords = ast.literal_eval("(20, 10), (30, 20)")
```

- *math*
  - **¿Para qué se usa?:** La librería math en Python se usa para realizar operaciones matemáticas avanzadas que no están disponibles con las operaciones básicas (+, -, *, /).
  - **Ejemplo típico:** Función usada: math.sqrt()

```
c = math.sqrt(a**2 - b**2)
```

Ejercicio Resuelto 1.2.6.2. Ejemplo de uso librerías, primero se importa, después se puede usar.

```
import math

# Longitudes de los catetos
a = 3
b = 4

# Calcular la hipotenusa usando math.sqrt
c = math.sqrt(a**2 + b**2)

print(f"La hipotenusa del triángulo es: {c}")
```

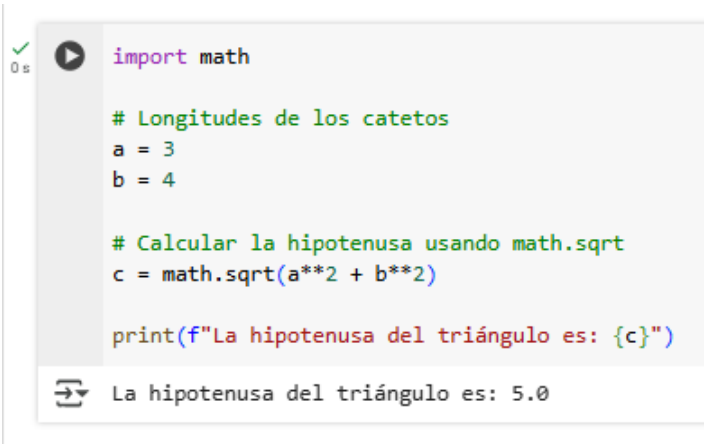


Figura 17. Código para importar librería *math* y salida esperada: "La hipotenusa del triángulo es: 5.0".

- scikit-learn

  - **¿Para qué se usa?:** La librería scikit-learn (también llamada sklearn para su importación) es una de las bibliotecas más populares en Python para *machine learning* (aprendizaje automático).
  - **Ejemplo típico:** leer

Ejercicio Resuelto 1.2.6.3. Importar librerías. Ejemplo de uso scikit-learn

```
# Importar librerías
import sklearn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Datos simulados
# X = años de experiencia
X = np.array([[1], [2], [3], [4], [5]])
# y = ingreso en miles de dólares
y = np.array([30, 35, 40, 45, 50]) # regresión lineal perfecta
y = np.array([32, 38, 46, 49, 52])

# Crear el modelo de regresión lineal
modelo = LinearRegression()

# Entrenar el modelo
modelo.fit(X, y)

# Predecir con el modelo
y_pred = modelo.predict(X)
```

```
# Mostrar la línea ajustada
plt.scatter(X, y, color='blue', label='Datos reales')
plt.plot(X, y_pred, color='red', label='Línea de regresión')
plt.xlabel('Años de experiencia')
plt.ylabel('Ingreso (mil USD)')
plt.title('Regresión lineal con scikit-learn')
plt.legend()
plt.grid(True)
plt.show()

# Mostrar coeficientes
print(f"Pendiente: {modelo.coef_[0]}")
print(f"Intersección: {modelo.intercept_}")
```

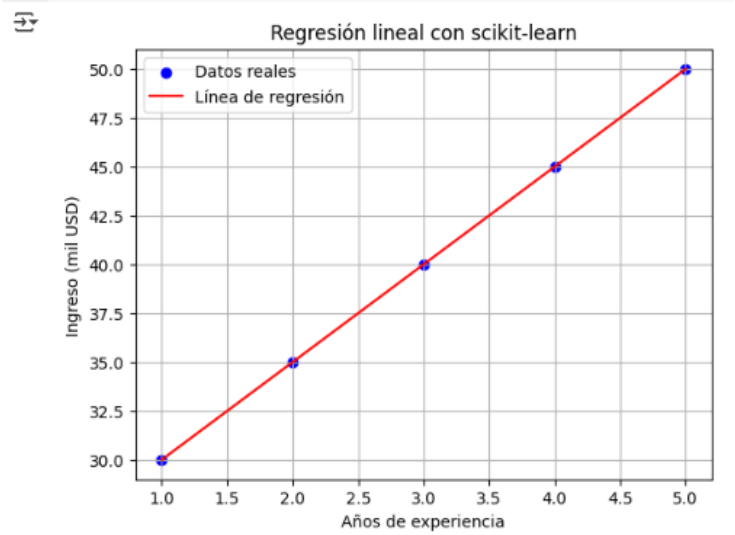


Figura 18. Usando librería `scikit-learn`. Nos ha dado una regresión lineal perfecta (en la realidad esto no suele suceder).

Ejercicio Propuesto 1.2.6.4.

• Cambiar los datos en las instrucciones para que la regresión lineal sea más realista (ejemplo en Figura 19).

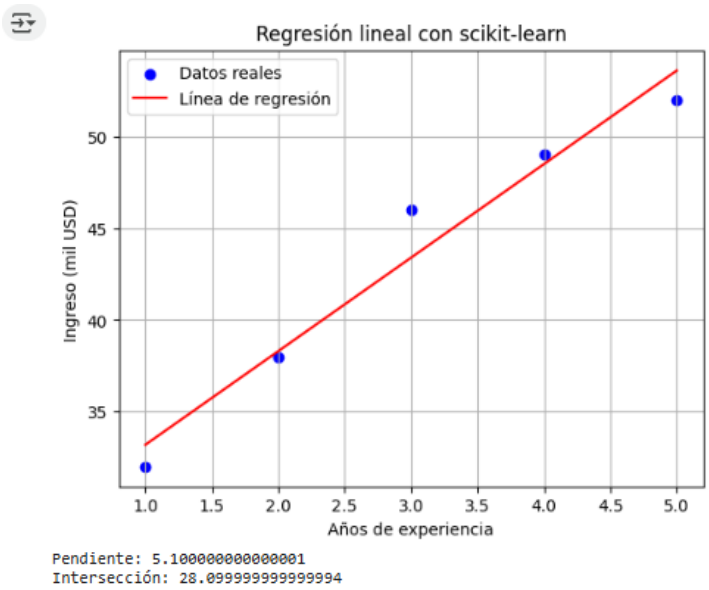


Figura 19. Usando librería scikit-learn. Aquí tenemos una salida un poco más realista.

Librería adicional: trimesh

!pip install *trimesh*

- ¿Para qué se usa? trimesh sirve para trabajar con geometría 3D, como mallas, modelos .STL o .OBJ, y escenas 3D.
- Importante en contextos de diseño industrial porque permite:
  - Crear y visualizar piezas en 3D
  - Exportar diseños a formatos imprimibles en 3D
  - Analizar volumen, área, normales y más propiedades geométricas

No se usa directamente para gráficos 2D, pero es esencial si se quiere llevar los diseños planos a modelos 3D o impresión 3D.

Ejemplo Básico:

```
import trimesh

# Cargar un modelo STL
mesh = trimesh.load('pieza.stl')

# Mostrar información de la malla
print(mesh)
print("Área superficial:", mesh.area)
print("Volumen:", mesh.volume)

# Visualización interactiva
mesh.show()
```

```
# Exportar a otro formato
mesh.export('pieza_convertida.obj')
```

1.2.7. Subir archivos.

Haz clic en el icono de carpeta (barra lateral izquierda) → botón "Subir".

1.2.8. Errores típicos.

Python incorpora una herramienta de IA que explica el error qu aparece en pantalla.

Figura 20. Error debido a que aún no existe el archivo *datos.csv* en mi Drive.

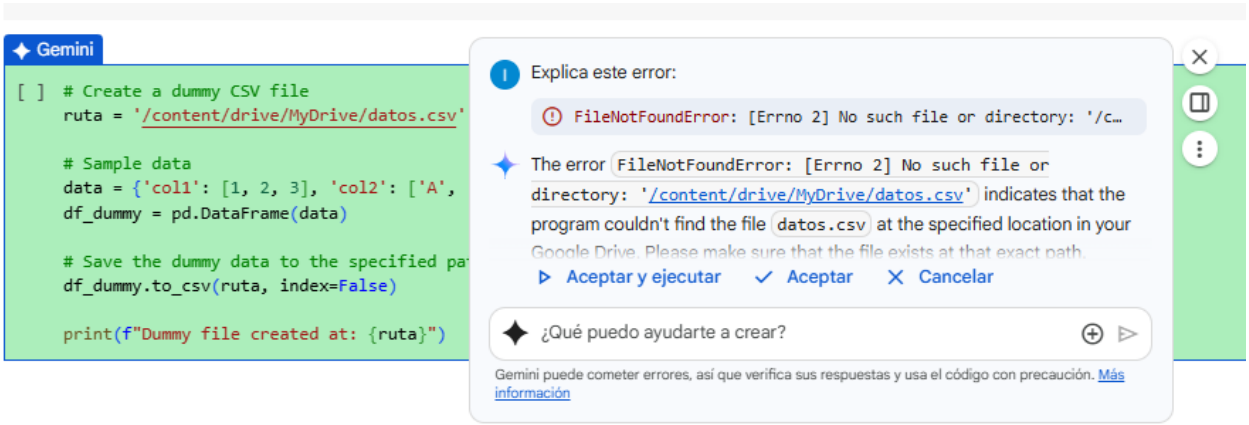


Figura 21. Explicación del error. La IA te propone una solución para corregir el error.

Cómo poder corregir este error:

Paso 1: Guarda tu archivo en Google Drive

Primero, sube datos.csv a tu carpeta de Drive (por ejemplo, en la raíz o en una carpeta como Colab Notebooks).

Paso 2: Script completo en Colab

```
from google.colab import drive
drive.mount('/content/drive')

ruta = '/content/drive/MyDrive/datos.csv'

import pandas as pd
df = pd.read_csv(ruta)
```

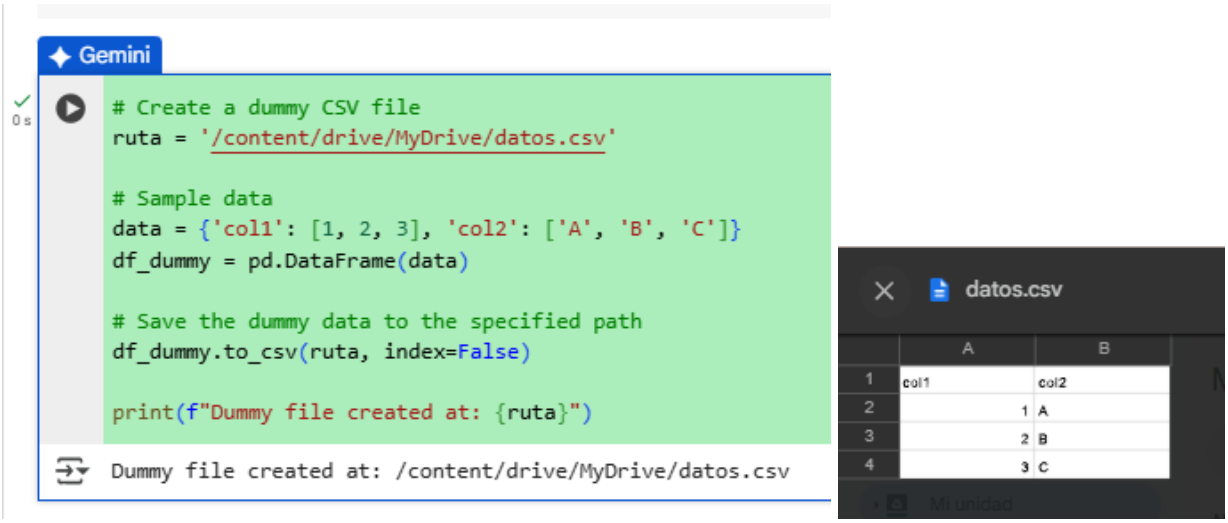


Figura 21. La IA te sugiere un cambio que si lo aceptas incluso te crea un archivo de muestra para corregir el error. Aparece en la carpeta "Mi unidad" de tu drive principal con el nombre que le indicaste en el *script*.

1.2.9. Compartir tu notebook.

Haz clic en el botón **"Compartir"** (arriba a la derecha), igual que en Google Docs. Puedes dar permisos de solo lectura o edición.

1.3. Generación de Figuras Geométricas Básicas en Python.

En el diseño técnico, las figuras geométricas básicas (puntos, segmentos, polígonos y circunferencias) son los elementos fundamentales a partir de los cuales se construyen piezas más complejas.
En este módulo aprenderemos a **programar y visualizar** estas figuras en Python, utilizando librerías de cálculo y trazado.

Herramientas que vamos a usar:

- **numPy** → para generar coordenadas y realizar cálculos.
- **matplotlib** → para representar las figuras en pantalla.
- (Opcional) **ezdxf** → para exportar los resultados a formato DXF y abrirlos en AutoCAD.

1.3.1. Representación de un punto

Un punto se representa como un par ordenado (x, y) en el plano.

```
import matplotlib.pyplot as plt

# Coordenadas del punto
x = [3]
y = [4]

plt.figure()
plt.plot(x, y, 'ro', label="Punto (3,4)") # 'ro' = red circle marker
plt.axis('equal')
plt.legend()
plt.title("Representación de un punto")
plt.show()
```

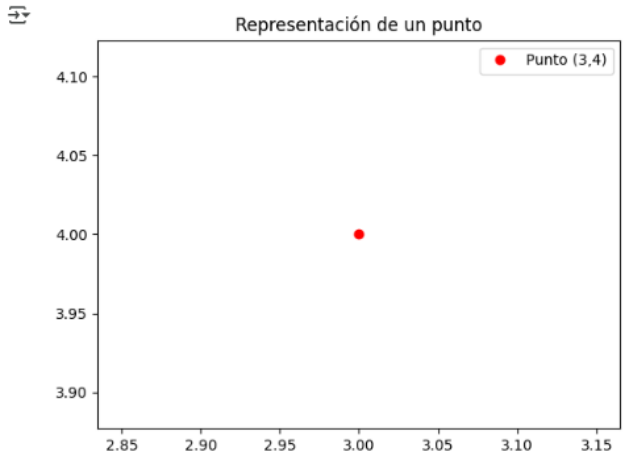


Figura 22. Salida esperada de un punto.

Ejercicio Propuesto 1.3.1.1.

- Cambiar los datos en las instrucciones para que el punto tenga otra posición.
- Agregar varios puntos en la misma gráfica.

1.3.2. Dibujo de un segmento

Un segmento se define por dos puntos (x1, y1) y (x2, y2).

```
x = [1, 5]
y = [2, 6]

plt.figure()
plt.plot(x, y, 'b-', linewidth=2, label="Segmento AB")
plt.scatter(x, y, color='red') # puntos extremos
plt.axis('equal')
plt.legend()
plt.title("Segmento AB")
plt.show()
```

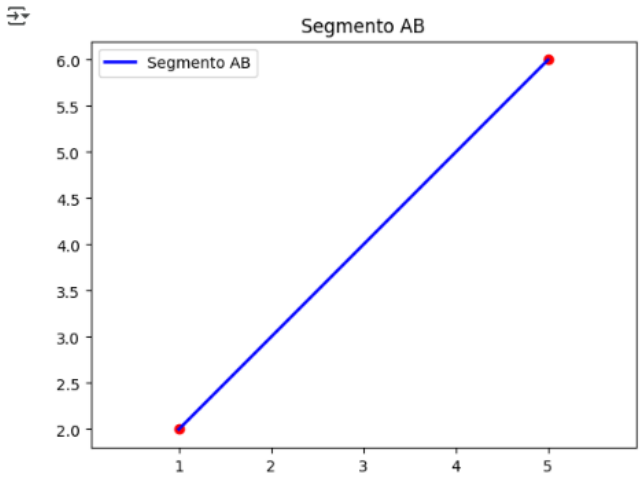


Figura 23. Salida esperada de un segmento.

Ejercicio Propuesto 1.3.2.1.

- Cambiar los datos en las instrucciones para que el segmento sea diferente.
- Usa el script del Ejercico Propuesto 1.3.1.1. y unir los puntos.

1.3.3. Creación de polígonos

Un polígono se genera uniendo varios puntos en orden y cerrando la figura repitiendo el primero al final.

Ejemplo: **Pentágono regular.**

```
import numpy as np

# Número de lados y radio
n = 5
r = 4
angulos = np.linspace(0, 2*np.pi, n+1)

x = r * np.cos(angulos)
y = r * np.sin(angulos)

plt.figure()
plt.plot(x, y, 'g-', linewidth=2, label="Pentágono")
plt.axis('equal')
plt.legend()
plt.title("Pentágono regular")
plt.show()
```

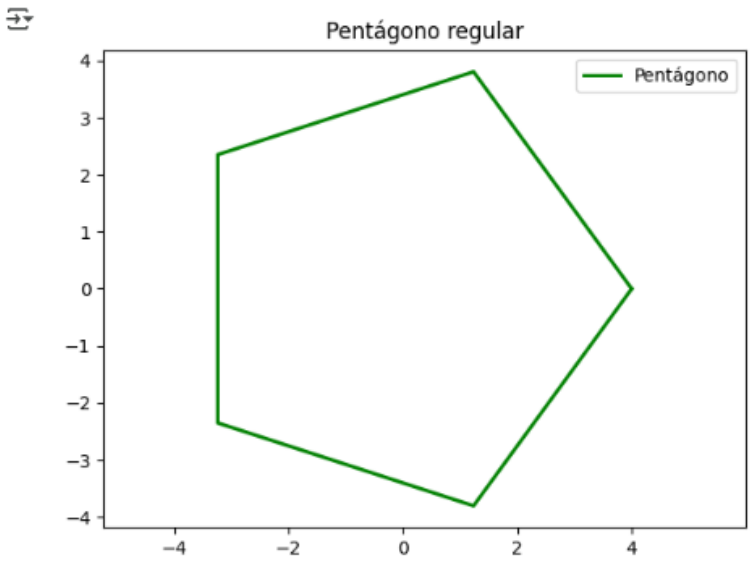


Figura 24. Salida esperada de un pentágono regular.

Ejercicio Propuesto 1.3.3.1.

- Cambiar los datos en las instrucciones para formar otro polígono regular, de 7 y 8 lados.
- Modificar el color del polígono.

1.3.4. Dibujo de circunferencias

La circunferencia se representa con ecuaciones paramétricas en el primer cuadrante (Ecuaciones 1-2):

$$x = r \cdot \cos(t) \quad (1)$$

$$y = r \cdot \text{sen}(t) \quad (2)$$

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

r = 5
t = np.linspace(0, 2*np.pi, 300)

x = r * np.cos(t)
y = r * np.sin(t)

plt.figure()
plt.plot(x, y, 'm-', linewidth=2, label="Circunferencia")
plt.axis('equal')
plt.legend()
plt.title("Circunferencia")
plt.show()
```

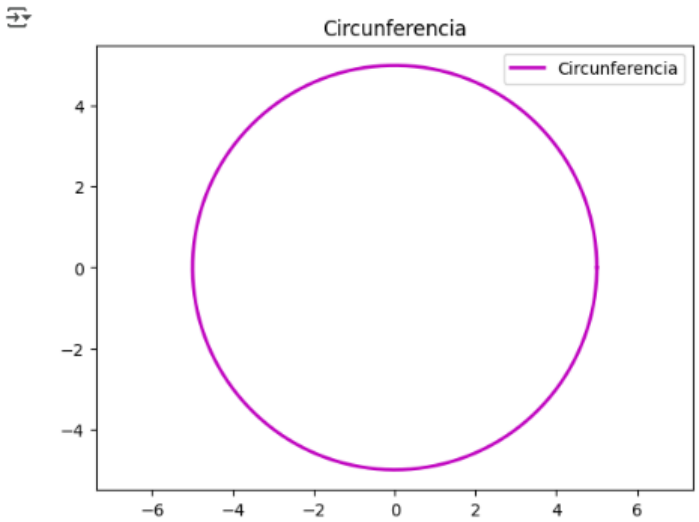


Figura 25. Salida esperada de un círculo.

Explicación de este código:

Ese código con **numPy** está creando **los puntos (x, y) de una circunferencia completa de radio 5** centrada en el origen.

```
r = 5 # Radio de la circunferencia = 5 unidades.

t = np.linspace(0, 2*np.pi, 300) # Genera 300 valores de t igualmente espaciados desde 0 hasta 2π radianes. Esto cubre toda la vuelta alrededor del círculo completo(360°).

x = r * np.cos(t)
y = r * np.sin(t)

# Aplica las ecuaciones paramétricas de la circunferencia para cada valor de t. x e y serán arrays con 300 puntos que, al graficarlos, forman un círculo perfecto.

# Interpretación: Si quisiéramos solo la parte de la circunferencia del primer cuadrante, cambiaríamos la línea de t a:

t = np.linspace(0, np.pi/2, 300)
```

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

r = 50  # radio
t = np.linspace(0, np.pi/2, 300)
x = r * np.cos(t)
y = r * np.sin(t)

plt.figure()
plt.plot(x, y)
plt.axis('equal')
plt.title("Circunferencia técnica generada con Python")
plt.show()
```

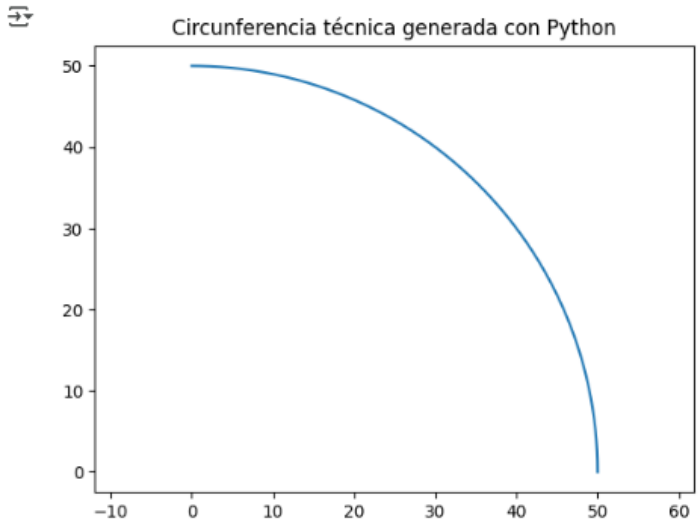


Figura 26. Salida esperada.

Ejercicio Propuesto 1.3.4.1.

- Dibujar en la misma figura:
  - Un punto en (2, 3).
  - Un segmento de (0,0) a (4,5).
  - Un triángulo equilátero de lado 5.
  - Una circunferencia de radio 3 centrada en el origen.
- Usar diferentes colores y estilos de línea.
- Mantener proporciones reales (plt.axis('equal')).

Ejercicio Propuesto Avanzado 1.3.4.2.

- Dibujar en Crear una función en Python que reciba:
  - Tipo de figura ("punto", "segmento", "polígono", "circunferencia").
  - Parámetros (coordenadas, número de lados, radio…).
- La función deberá dibujar la figura correspondiente.
- Opcional: añadir la opción de exportar a DXF para su uso en AutoCAD.

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

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

Primero de todo, tan sólo comentar, que antes de seguir con este curso para aquellos estudiantes que quieran conocer el entorno básico de Python, si es que nunca lo han usado, se recomienda acceder al **curso OCW *Aplicaciones de Machine Learning en Matlab y Python para la Ingeniería Civil e Ingeniería Industrial*** de la UCA a través de https://ocw.uca.es/ .

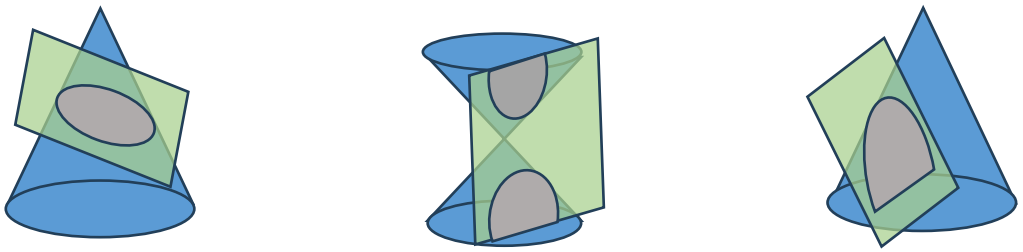
Se recomienda ir haciendo los ejemplos en un script y hacer los ejercicios propuesto de cada apartado.

Para sintetizar el Dibujo Técnio con la computación actual, vamos a crear una **pieza técnica sencilla** (como una tuerca, una pletina con agujeros, o una pieza simétrica) usando programación en Python, con el propósito de:

- Explorar cómo se **representa la geometría** a través de coordenadas en Python.
- Introducir **Python como herramienta creativa y técnica** para el desarrollo de **Dibujo Técnico**.
- Establecer conexiones entre programación, matemáticas, visualización y diseño técnico.

2.1. Diseño de curvas cónicas

Se denominan secciones cónicas a aquellas superficies que son producidas por la intersección de un plano con una superficie cónica de revolución. Según la posición del plano secante respecto al eje del cono, en relación con el ángulo en el vértice, se obtienen tres curvas diferentes: elipse, hipérbola y parábola.



(a) (b) (c)

Figura 27. Curvas de revolución. (a) elipse, (b) hipérbola, (c) parábola.

La elipse y la hipérbola tienen dos directrices y dos focos; la parábola tiene un solo foco y, con ello, una sola directriz, puesto que el plano secante es paralelo a una de las generatrices de la superficie cónica y corta sólo a una de las ramas del cono.

Teorema de Dandelin. El foco o los focos de una curva cónica se encuentran en los puntos de tangencia del plano secante con las esferas inscritas en la superficie cónica que lo sean, a su vez, tangentes al plano que produce la sección. Los focos son puntos notables de las cónicas. Representación gráfica en las Figuras 7. (a),(b) y (c).

Directriz de una curva cónica. Se llama directriz a la recta intersección del plano secante (en rojo) que produce la curva cónica y el plano de contacto (en morado) que contiene a los puntos de tangencia de las circunferencias en 2D (en verde, realmente en 3D son esferas inscritas a las paredes del cono). Estas circunferencias son además tangentes al plano secante, dando ligar a los focos en los puntos de tangencia. En las Figuras 7.(a),(b) y (c), se muestran representadas en proyección frontal, las superficies cónicas por las dos generatrices de su contorno aparente, tangentes a las esferas. Podemos observar la vista abatida, en verdadera magnitud, producida sobre el plano secante donde se muestra la curva cónica en 2D, sus rectas directrices, que al ser perpendiculares al plano del dibujo en proyección frontal, están representadas por las rectas abatidas d_x.

Excentricidad de una curva cónica (e). La razón entre las distancias de un punto cualquiera de una curva cónica al foco y a la recta directriz correspondiente a ese foco, es una cantidad constante llamada *excentricidad*. Gráficamente podemos observarla en las Figuras 27 (a),(b) y (c) en naranja, cuyas distancias las proporciona directamente el programa *Geogebra*. En la elipse $\boldsymbol{e < 1}$, en la hipérbola $\boldsymbol{e > 1}$ y en la parábola $\boldsymbol{e = 1}$. Este concepto es muy útil y fundamental para la construcción de curvas cónicas.

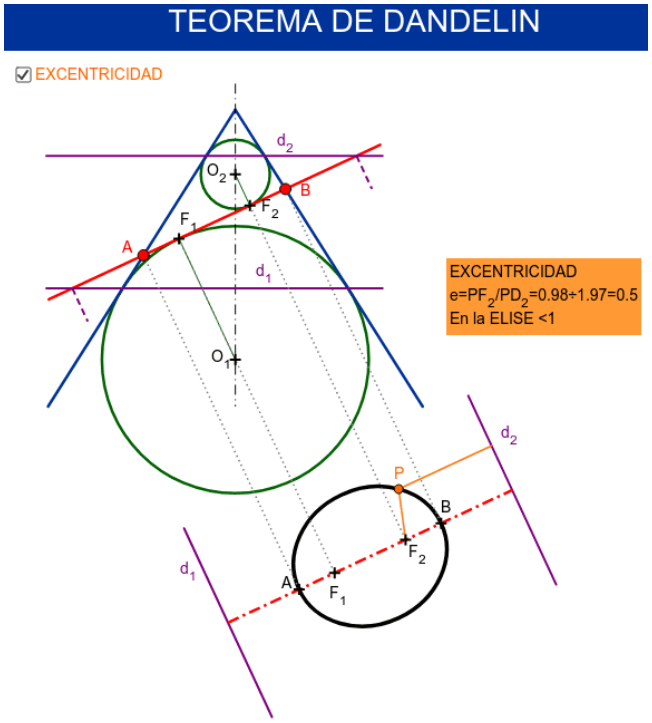


Figura 27. (a)

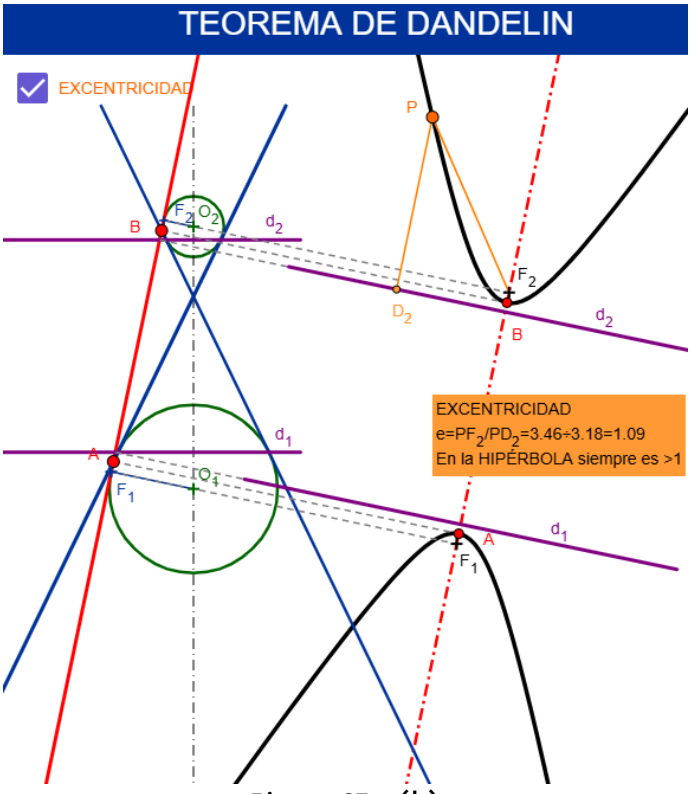


Figura 27. (b)

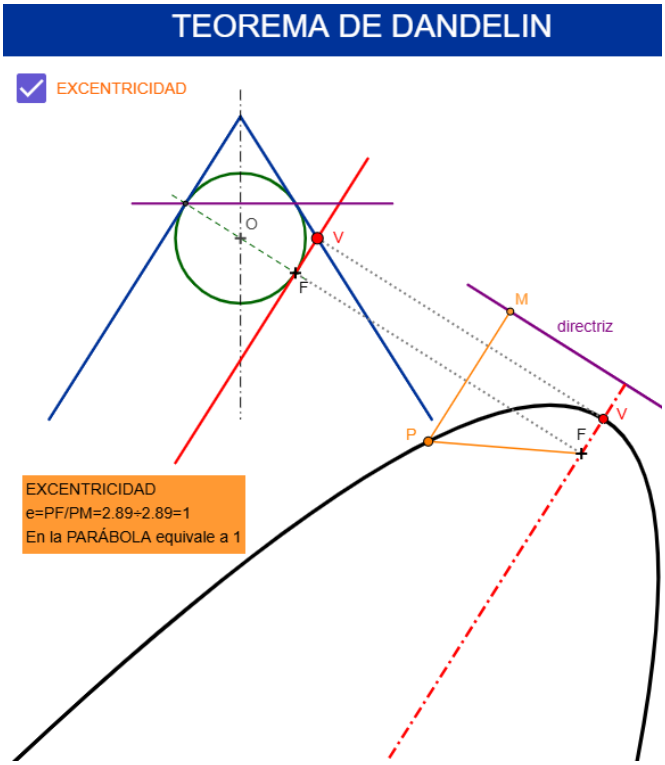


Figura 27. (C)

Figura 28. Teorema de Dandelin en curvas de revolución. (a) elipse, (b) hipérbola, (c) parábola. Fuente: *Geogebra*.

2.1.1. Diseño de elipses.

La **construcción de elipses** se puede hacer perfectamente en Python, y es un excelente ejercicio para trabajar:

- Geometría analítica (construcción por ecuación paramétrica).
- Visualización gráfica.
- Uso de herramientas computacionales en ingeniería gráfica.

Se abordará la elipse desde su formulación matemática hasta su modelado computacional en Python y la exportación a formatos CAD (DXF). Se tratarán las formas canónicas y paramétricas, el cálculo de focos, la excentricidad, transformaciones (traslación y rotación) y ejemplos aplicados a la Ingeniería Civil e Industrial.

Objetivos de la unidad:

- Entender las definiciones geométricas y algebraicas de la elipse.
- Calcular semiejes, focos y excentricidad a partir de parámetros a y b.
- Implementar en Python la generación y visualización de elipses (posicionadas y rotadas).
- Exportar la elipse a un archivo DXF para su uso en AutoCAD.

Fundamentos matemáticos. Definición geométrica

Una **elipse** es una curva cerrada y plana, lugar geométrico de los puntos del plano cuya suma de distancias a dos puntos fijos llamados focos, es constante e igual a ***2a***, siendo ésta la longitud del eje mayor (o eje real) de la elipse. El eje menor o imaginario se

representa por 2b. Ambos ejes son perpendiculares entre sí y son ejes de simetría, cortándose en el centro. Si los focos son F_1 y F_2 y P un punto cualquiera de la curva, $|F_1P|+|F_2P| = 2a$. La distancia entre ambos focos recibe el nombre de distancia focal y se representa por $2c$.

Forma canónica de la elipse centrada en el origen (0,0):

Para una elipse centrada en el origen y con eje mayor en el eje x:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1, \qquad a \geq b > 0$$

- Semieje mayor: a.
- Semieje menor: b.
- Distancia focal (desde el centro al foco): $c = \sqrt{a^2-b^2}$
- Focos: $(\pm c, 0)$
- **Excentricidad**: medida de la "elongación" de la elipse:

$$e = \frac{c}{a} = \sqrt{1-\frac{b^2}{a^2}} \quad , \; 0 < e < 1$$

Si el eje mayor está en el eje y, la ecuación es:

$$\frac{x^2}{b^2}+\frac{y^2}{a^2}=1, \qquad a \geq b > 0$$

Formas paramétricas de la elipse:

Una representación muy útil para graficar es paramétrica (centro en el origen):

$$x(t) = a\cos t, \;\; y(t) = b\, sen\, t, \quad t\, \epsilon\, [0, 2\pi]$$

Esta parametrización recorre la elipse en sentido antihorario.

Traslación y rotación de una elipse:

- **Traslación**: si el centro es (h,k), entonces:

$$x(t) = h + a\cos t, \;\; y(t) = k + b\, sen\, t$$

- **Rotación**: para rotar la elipse un ángulo (antihorario) se aplica la matriz de rotación a los puntos paramétricos:

Rotación en 2D. Para rotar un punto (x,y) un ángulo θ (positivo = antihorario) alrededor del **origen (0,0)**:

$$R(\theta)_+ = \begin{pmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & \cos(\theta) \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = R(\theta)\begin{pmatrix} x \\ y \end{pmatrix}$$

O lo que es lo mismo:

$$X = x\cos(\theta) - ysen(\theta)$$
$$Y = x\,\mathrm{sen}(\theta) + ycos(\theta)$$

NOTA: matriz de rotación en sentido horario o negativo:

$$R(\theta)_{-} = \begin{pmatrix} \cos(\theta) & sen(\theta) \\ -sen(\theta) & \cos(\theta) \end{pmatrix}$$

Propiedades

- Det $R = 1$, $R^{-1} = R^T = R(-\theta)$
- Conserva distancias y ángulos (isometría euclídea).

Traslada (x, y) al origen, rota, y vuelve a trasladar:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} h \\ k \end{pmatrix} + R(\theta)\left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} h \\ k \end{pmatrix}\right)$$

Expresado componente a componente:

$$x' = h + (x-h)\cos(\theta) - (y-k)sen(\theta)$$
$$y' = k + (x-h)sen(\theta) - (y-k)\cos(\theta)$$

Código en Python:

```
import numpy as np

def rotar_2d(x, y, theta, h=0.0, k=0.0):
    ct, st = np.cos(theta), np.sin(theta)
    xr = h + (x - h)*ct - (y - k)*st
    yr = k + (x - h)*st + (y - k)*ct
    return xr, yr
```

Rotación en 3D. Para rotar un punto (x, y, z) alrededor de los ejes cartesianos:

Alrededor de **X** un ángulo α: $R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -sen(\alpha) \\ 0 & sen(\alpha) & \cos(\alpha) \end{pmatrix}$

Alrededor de **Y** un ángulo β: $R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & sen(\beta) \\ 0 & 1 & 0 \\ -sen(\beta) & 0 & \cos(\beta) \end{pmatrix}$

Alrededor de **Z** un ángulo φ: $R_z(\phi) = \begin{pmatrix} \cos(\phi) & -sen(\phi) & 0 \\ sen(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Composición. La rotación general puede escribirse como producto de matrices $R = R_z \cdot R_y \cdot R_x$ (u otro orden según convención).

Para rotar un vector $\boldsymbol{v}$ un ángulo $\boldsymbol{\theta}$ alrededor de un eje unitario $\boldsymbol{u} = (u_x, u_y, u_z)$,

$$R(\boldsymbol{u}, \theta) = I\, cos\theta + (1 - cos\theta)\boldsymbol{u}\boldsymbol{u}^{\boldsymbol{T}} + [\boldsymbol{u}]_{\mathrm{x}}\, sen\theta$$

Siendo $[\boldsymbol{u}]_{\mathrm{x}} = \begin{pmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{pmatrix}$

Ejemplo de uso en la elipse de rotación y traslación. Si $x(t) = a\cos t$, $y(t) = b\, sen\, t$,

$$(x_r(t), y_r(t)) = (h, k) + R(\theta)(a\cos t, b\, sen\, t)$$

- **Ejemplo básico: Elipse con Python**

Ejercicio 2.1.1.1. Diseño de una elipse

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

# Parámetros de la elipse
a = 50  # semieje mayor
b = 30  # semieje menor

# Ángulo en radianes para trazar la curva
t = np.linspace(0, 2 * np.pi, 300)

# Ecuaciones paramétricas de la elipse
x = a * np.cos(t)
y = b * np.sin(t)

# Crear la figura
plt.figure(figsize=(6, 6))
plt.plot(x, y, label="Elipse")
plt.gca().set_aspect('equal')  # Ejes proporcionales
plt.title("Construcción de una Elipse en Python")
plt.grid(True)
plt.legend()
plt.show()
```

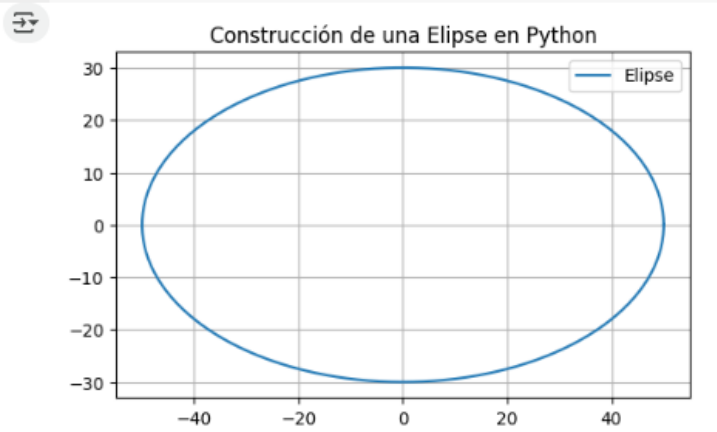


Figura 29. Código y resultado esperado de la elipse.

Ejercicio 2.1.1.2. Diseño de una elipse y guardar el archivo en .dxf

```
!pip install ezdxf

import math
import ezdxf
import numpy as np

# Parámetros de la elipse
a = 50  # semieje mayor
b = 30  # semieje menor
cx = 0  # centro en x
cy = 0  # centro en y

# Ángulo en radianes para trazar la curva
t = np.linspace(0, 2 * np.pi, 300)

# Ecuaciones paramétricas de la elipse
x = cx + a * np.cos(t)
y = cy + b * np.sin(t)

# Crear un nuevo archivo DXF
doc = ezdxf.new('R2010')
modelspace = doc.modelspace()

# Añadir la elipse como polilínea
points = list(zip(x, y))
modelspace.add_lwpolyline(points, close=True)

# Guardar el archivo DXF
doc.saveas('elipse.dxf')
```

```
print("El archivo elipse.dxf se ha creado correctamente.")
```

```
Collecting ezdxf
  Downloading ezdxf-1.4.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.8 kB)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.11/dist-packages (from ezdxf) (3.2.3)
Requirement already satisfied: typing_extensions>=4.6.0 in /usr/local/lib/python3.11/dist-packages (from ezdxf) (4.14.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from ezdxf) (2.0.2)
Requirement already satisfied: fonttools in /usr/local/lib/python3.11/dist-packages (from ezdxf) (4.59.0)
Downloading ezdxf-1.4.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 60.4 MB/s eta 0:00:00
Installing collected packages: ezdxf
Successfully installed ezdxf-1.4.2
El archivo elipse.dxf se ha creado correctamente.
```

Figura 30. Código y resultado esperado.

El archivo *elipse.dxf* se guarda en la carpeta desde la que estás ejecutando tu script o notebook de Python.

- En **Jupyter Notebook**, normalmente se guarda en la misma carpeta donde está el notebook.
- En un **script Python (.py)**, se guarda en la carpeta actual donde ejecutas python script.py.

Para verificar la ruta exacta, se puede usar este código:

```
import os
print(os.getcwd())  # Muestra la carpeta actual
```

Figura 31. Código y resultado esperado.

Esto te permitirá localizar *elipse.dxf* y abrirlo en AutoCAD u otro programa CAD. Para convertirlo a .dwg habrá que usar alguna herramienta o conversor.

Si quieres guardarlo directamente en tu Google Drive (en Colab), debes montar tu drive así:

```
# Para guardarlo en tu Drive, lo montas primero
from google.colab import drive
drive.mount('/content/drive')

# Luego lo guardas en una ruta de tu drive, por ejemplo:
doc.saveas('/content/drive/My Drive/elipse.dxf')
```

```
Mounted at /content/drive
```

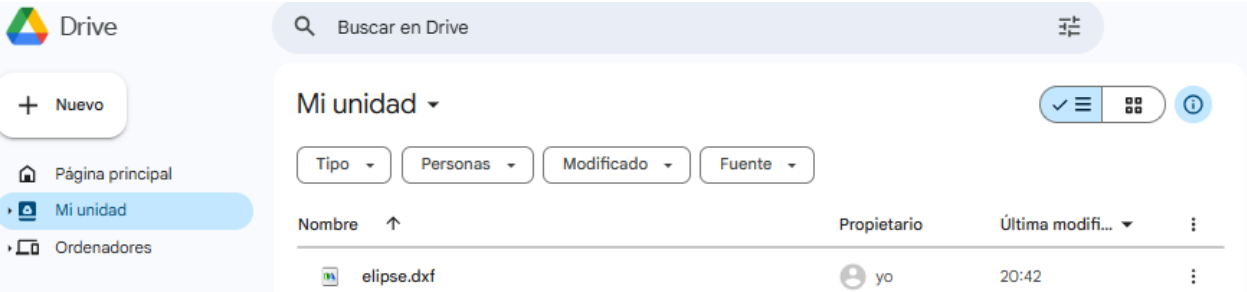


Figura 32. Código y resultado esperado. El archivo aparece en nuestra carpeta general de Drive.

Construcción geométrica de la elipse de forma tradicional con cuerda y con circunferencias.

1) Método de la cuerda (o "del jardinero").

Se fijan dos clavos en los focos F_1, F_2 y se ata una cuerda de longitud total del eje mayor $\mathbf{2a}$. Tensando la cuerda con un lápiz, la punta recorre la elipse: $|PF_1|+|PF_2| = 2a$.
Datos: semiejes $a \geq b$, focos en $(\pm c, 0)$, con $c = \sqrt{a^2 - b^2}$.
String (cuerda): longitud = 2a; puntos focos: $F_1 = (-c, 0)$, $F_2 = (c, 0)$.

Ejercicio 2.1.1.3. Diseño y dibujo de una elipse por el método de la cuerda (o "del jardinero")

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

# Parámetros
a, b = 6.0, 4.0
c = np.sqrt(a**2 - b**2)
F1, F2 = (-c, 0.0), (c, 0.0)

# Elipse paramétrica (visualización de la curva)
t = np.linspace(0, 2*np.pi, 600)
x = a*np.cos(t)
y = b*np.sin(t)

# Elegimos un punto P de la elipse para dibujar la cuerda tensada
ang = np.deg2rad(40)
Px, Py = a*np.cos(ang), b*np.sin(ang)

plt.figure(figsize=(7,5))
plt.plot(x, y, 'b', label='Elipse')
plt.scatter([F1[0], F2[0]], [F1[1], F2[1]], c='r', label='Focos')
plt.plot([F1[0], Px], [F1[1], Py], 'g--', lw=1.5, label='Cuerda F1-P')
plt.plot([F2[0], Px], [F2[1], Py], 'g--', lw=1.5, label='Cuerda F2-P')
plt.scatter([Px],[Py], c='k', zorder=5, label='Punto P')
```

```
# Ejes y formato
plt.axhline(0, color='k', lw=0.5); plt.axvline(0, color='k', lw=0.5)
plt.gca().set_aspect('equal', adjustable='box')
plt.title('Elipse por método de la cuerda (jardinero)')
plt.legend(loc='upper right', fontsize=8)
plt.grid(True, ls=':', alpha=0.5)
plt.show()
```

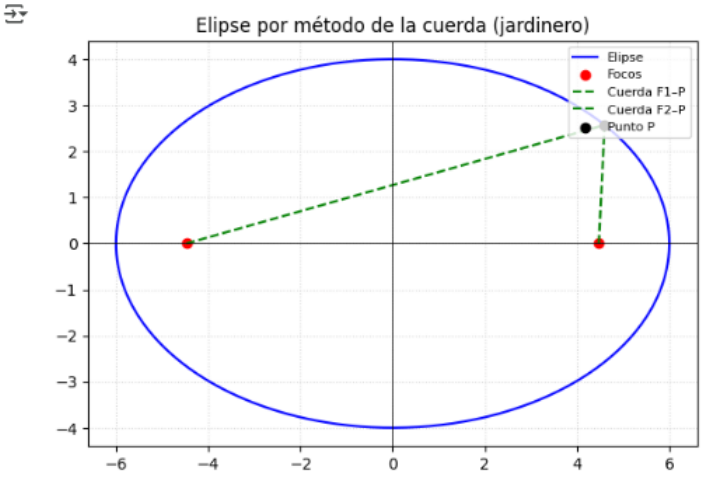


Figura 33. Código y resultado esperado de la elipse por el método del jardinero.

2) Método de proyección de puntos. Afinidad.

Método que usa circunferencias auxiliares concéntricas, que consiste en construir una elipse utilizando dos circunferencias concéntricas de radios iguales a los semiejes mayor ($\boldsymbol{a}$) y menor ($\boldsymbol{b}$) de la elipse y proyectando puntos entre ellas.

Fundamento geométrico de "Afinidad".

Esta construcción se basa en que la elipse y la circunferencia cuyo diámetro mide igual que el eje mayor de la elipse, son afines, siendo el eje de afinidad el propio eje mayor y la dirección de afinidad perpendicular al eje.

Construcción

- Se trazan dos circunferencias concéntricas con centro en un punto O:
  - Radio mayor = semieje mayor $\boldsymbol{a}$
  - Radio menor = semieje menor $\boldsymbol{b}$
- Se trazan varios diámetros comunes a ambas circunferencias, con lo que habrán quedado divididas en cualquier número par de partes iguales o desiguales.
- Por los puntos de división obtenidos en la circunferencia mayor, se trazan perpendiculares al eje mayor y por los de la circunferencia menor paralelas al eje mayor.
- Las intersecciones mutuas de estas rectas auxiliares determinan los puntos de la elipse.

Ejercicio 2.1.1.4. Diseño de una elipse por método de circunferencias auxiliares concéntricas

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

# Parámetros de la elipse
a = 5   # semieje mayor
b = 3   # semieje menor
n = 12  # divisiones de la circunferencia mayor

# Ángulos de división
theta = np.linspace(0, 2*np.pi, n, endpoint=False)

# Puntos en circunferencia mayor y menor
x_circ = a * np.cos(theta)
y_circ = a * np.sin(theta)

x_circ_menor = b * np.cos(theta)
y_circ_menor = b * np.sin(theta)

# Puntos de la elipse (proyección)
x_elipse = a * np.cos(theta)
y_elipse = b * np.sin(theta)

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

# Circunferencias auxiliares
circle_mayor = plt.Circle((0,0), a, fill=False, ls='--', color='gray')
circle_menor = plt.Circle((0,0), b, fill=False, ls='--', color='gray')
ax.add_artist(circle_mayor)
ax.add_artist(circle_menor)

# Radios
for i in range(n):
    ax.plot([0, x_circ[i]], [0, y_circ[i]], color='lightgray', lw=0.8)

# Elipse final
ax.plot(x_elipse, y_elipse, 'b-', lw=2, label='Elipse')
ax.scatter(x_elipse, y_elipse, color='red')

ax.legend()
```

```
plt.title("Método de circunferencias auxiliares concéntricas")
plt.grid(True)
plt.show()
```

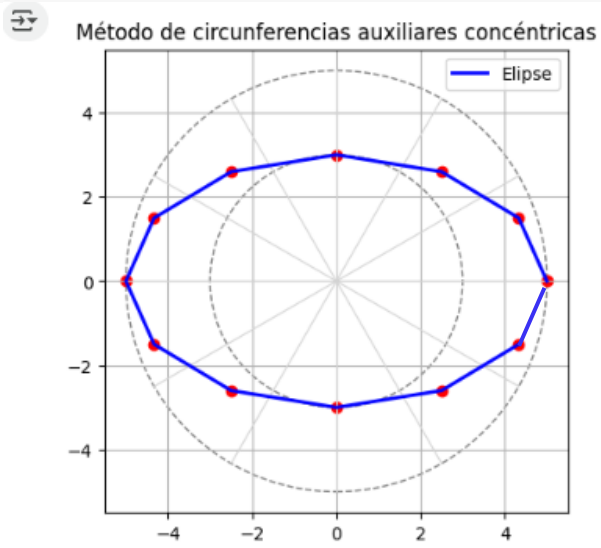


Figura 34. Código y resultado, sin líneas auxiliares.

Ejercicio 2.1.1.5. Diseño de una elipse por ambos métodos: Jardinero y afinidad.

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

# Parámetros de la elipse
a = 6.0  # semieje mayor
b = 4.0  # semieje menor
c = np.sqrt(a**2 - b**2)  # distancia focal

# Focos
F1 = (-c, 0.0)
F2 = (c, 0.0)

# Parámetros para muestreo
t = np.linspace(0, 2*np.pi, 400)
# Puntos reales de la elipse
x_e = a * np.cos(t)
y_e = b * np.sin(t)

# --- 1) Método de la cuerda (jardinero) ---
# Seleccionamos algunos ángulos para mostrar ejemplos de cuerdas
t_samples = np.linspace(0.1, 2*np.pi-0.1, 12)
P_samples = np.column_stack((a*np.cos(t_samples), b*np.sin(t_samples)))
```

```
# comprobación: |PF1| + |PF2| debe ser 2a
sums = np.linalg.norm(P_samples - np.array(F1), axis=1) +
np.linalg.norm(P_samples - np.array(F2), axis=1)

# --- 2) Método de circunferencias auxiliares concéntricas ---
n = 24
theta = np.linspace(0, 2*np.pi, n+1)  # cerramos la discretización
A_pts = np.column_stack((a*np.cos(theta), a*np.sin(theta)))  # puntos en
circunf. mayor
B_pts = np.column_stack((b*np.cos(theta), b*np.sin(theta)))  # puntos en
circunf. menor
# Construcción: P = (x_A, y_B)
P_construct = np.column_stack((A_pts[:,0], B_pts[:,1]))

# --- Dibujar ambas construcciones lado a lado ---
fig, axes = plt.subplots(1, 2, figsize=(14,6))

# 1) Dibujo método de la cuerda
ax = axes[0]
ax.set_aspect('equal')
ax.plot(x_e, y_e, 'b-', lw=2, label='Elipse (teórica)')
ax.scatter([F1[0], F2[0]], [F1[1], F2[1]], c='red', label='Focos')

# Dibujar muestras y cuerdas tensadas (segmentos F1-P y F2-P)
for i, P in enumerate(P_samples):
    ax.plot([F1[0], P[0]], [F1[1], P[1]], 'g--', lw=1)
    ax.plot([F2[0], P[0]], [F2[1], P[1]], 'g--', lw=1)
    ax.scatter(P[0], P[1], c='k')
    ax.text(P[0]*1.02, P[1]*1.02, f"P{i}", fontsize=8)
    # mostrar la suma de distancias (comprobación)
    ax.text(P[0]*1.02, P[1]*1.02 - 0.4, f"s={sums[i]:.3f}", color='green',
fontsize=7)

ax.set_title('Método de la cuerda (jardinero)')
ax.grid(True, ls=':', alpha=0.6)
ax.legend()
ax.axhline(0, color='k', lw=0.5); ax.axvline(0, color='k', lw=0.5)

# 2) Dibujo método de circunferencias auxiliares
ax2 = axes[1]
ax2.set_aspect('equal')
# Dibujar las circunferencias auxiliares
circle_big = plt.Circle((0,0), a, fill=False, ls='--', color='gray')
circle_small = plt.Circle((0,0), b, fill=False, ls='--', color='gray')
ax2.add_artist(circle_big)
```

```
ax2.add_artist(circle_small)

# Dibujar puntos A (big), B (small) y P construidos
ax2.plot(A_pts[:,0], A_pts[:,1], '.', color='orange', label='A en r=a')
ax2.plot(B_pts[:,0], B_pts[:,1], '.', color='green', label='B en r=b')
ax2.plot(P_construct[:,0], P_construct[:,1], 'r--', lw=1, label='P
construidos (discretos)')
ax2.scatter(P_construct[:,0], P_construct[:,1], c='red')

# Para cada división, dibujar las proyecciones horizontales/verticales que
llevan a P
for i in range(len(theta)):
    Ax, Ay = A_pts[i]
    Bx, By = B_pts[i]
    Px, Py = Ax, By
    # dibujar radio a Ax
    ax2.plot([0, Ax], [0, Ay], color='lightgray', lw=0.8)
    # dibujar radio a B
    ax2.plot([0, Bx], [0, By], color='lightgray', lw=0.8)
    # líneas de proyección
    ax2.plot([Ax, Px], [Ay, Py], 'b--', lw=0.8)
    ax2.plot([Bx, Px], [By, Py], 'b--', lw=0.8)
    # marcar puntos A, B y P
    ax2.scatter([Ax], [Ay], color='orange')
    ax2.scatter([Bx], [By], color='green')
    ax2.scatter([Px], [Py], color='red')

# Dibujar la elipse teórica para comparación
ax2.plot(x_e, y_e, 'k-', lw=1, alpha=0.7)
ax2.set_title('Método de circunferencias auxiliares (proyección)')
ax2.grid(True, ls=':', alpha=0.6)
ax2.legend()
ax2.axhline(0, color='k', lw=0.5); ax2.axvline(0, color='k', lw=0.5)

plt.suptitle('Construcción clásica de la elipse: cuerda y circunferencias
auxiliares')
plt.tight_layout()
plt.show()
```

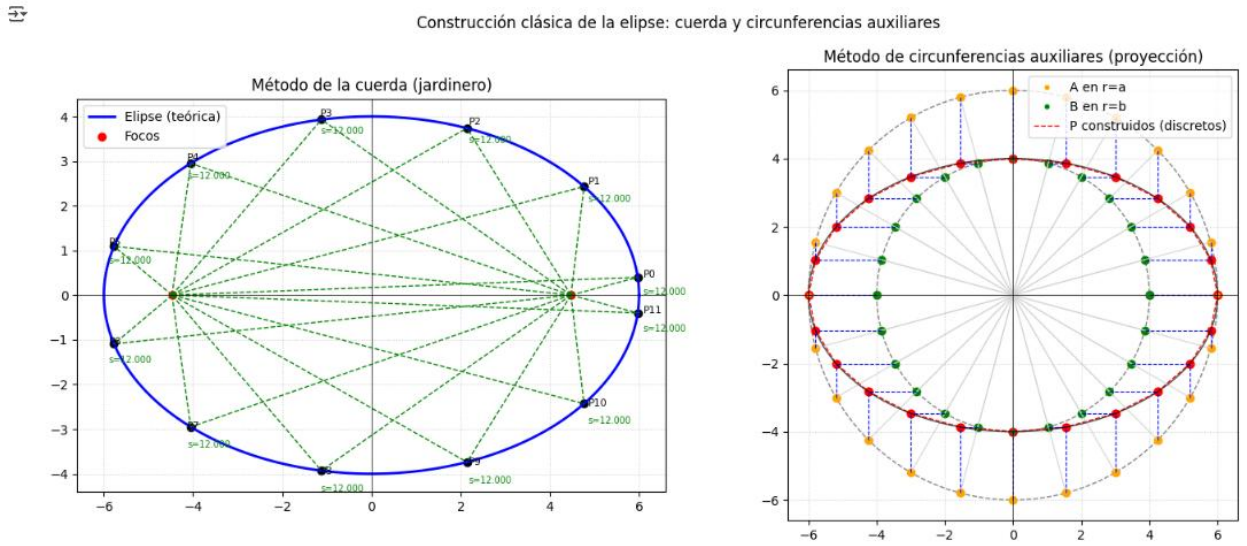


Figura 35. Código y resultado para diseño de elispes por ambos métodos: Jardinero y circunferencias auxiliares.

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()
```

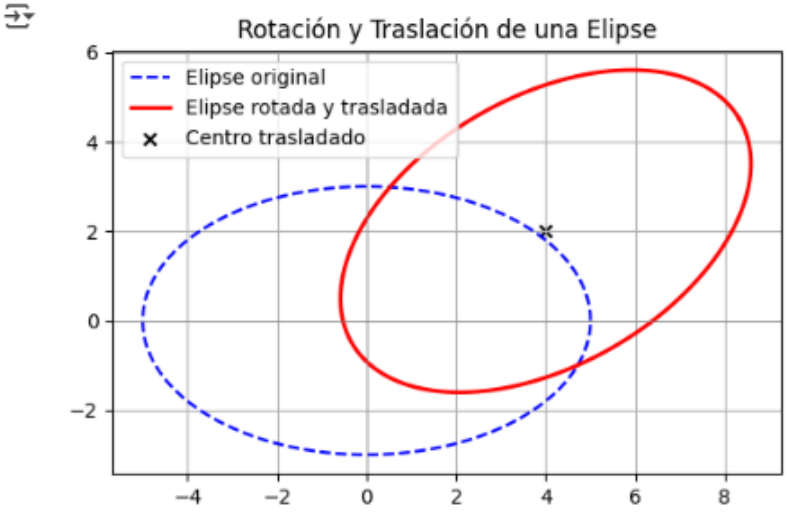


Figura 36. Código y resultado para rotar y trasladar una elipse.

Ejercicio Propuesto 2.1.1.7.

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

Ejercicio 2.1.1.8. Diseño de una elipse y sus focos dados sus semiejes

```
import math

# Datos (puedes cambiar estos valores)
a = 5  # semieje mayor
b = 3  # semieje menor

# Calcular c
c = math.sqrt(a**2 - b**2)

# Coordenadas de los focos
foco1 = (c, 0)
foco2 = (-c, 0)

print("Focos de la elipse:")
print(f"Foco 1: {foco1}")
print(f"Foco 2: {foco2}")


import matplotlib.pyplot as plt
import numpy as np

# Crear los puntos de la elipse
theta = np.linspace(0, 2*np.pi, 100)
x = a * np.cos(theta)
y = b * np.sin(theta)

plt.figure(figsize=(8,6))
plt.plot(x, y, label='Elipse')
plt.scatter([foco1[0], foco2[0]], [foco1[1], foco2[1]], color='red',
label='Focos')
plt.axhline(0, color='gray', lw=0.5)
plt.axvline(0, color='gray', lw=0.5)
plt.gca().set_aspect('equal', adjustable='box')
plt.title('Elipse y sus focos')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

```
Focos de la elipse:
Foco 1: (4.0, 0)
Foco 2: (-4.0, 0)
```

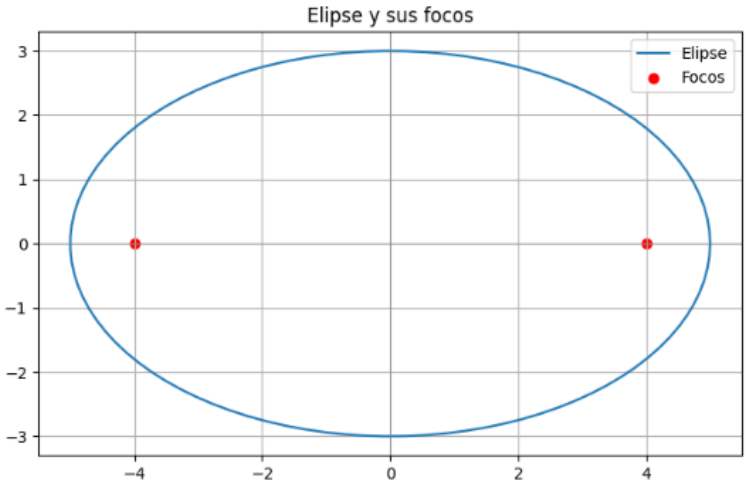


Figura 37. Código y resultado de una elipse y sus focos.

2.1.2. Diseño de hipérbolas.

La hipérbola es una curva abierta, plana y con dos ramas, que se define como el lugar geométrico de los puntos del plano cuya diferencia de distancias a dos puntos fijos llamados focos es constante. Tiene dos vértices y dos ejes de simetría, perpendiculares entre sí que se cortan en el punto medio del eje mayor, siendo éste el centro de la curva. El eje mayor es ***2a*** siendo la distancia comprendida entre vértices de la curva. El eje menor es perpendicular al eje mayor en el centro de la curva. La distancia focal sigue siendo ***2c***. Los radios vectores son los segmentos que unen cualquier punto de la curva con los dos focos. Al igual que en la elipse la circunferencia focal es la que tiene centro en el centro de la curva y diámetro la distancia focal, y la circunferencia principal, mismo centro y diámetro el eje mayor. Una asíntota en la hipérbola es la tangente a la curva en el infinito, siendo hipérbola equilátera cuando las asíntotas forman 45º respecto a los ejes de la curva. Toda asíntota pasa por el centro de la curva. La hipérbola no tiene eje menor sino eje imaginario, ***2b*** define el eje conjugado (no tiene puntos de la curva, pero sirve para trazar las asíntotas).

Forma canónica de la hipérbola centrada en el origen (0,0):

a.1) Hipérbola con eje real en el eje x (horizontal):

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$$

- Semieje mayor/real: a.
- Semieje imaginario: b.
- Distancia focal (desde el centro al foco): $c = \sqrt{a^2 + b^2}$
- Focos: $(\pm c, 0)$
- Asíntotas: $y = \pm \frac{b}{a} x$

- **Excentricidad**: medida de la "apertura" de la hipérbola. Cuanto mayor es e, más abierta es la hipérbola.:

$$e = \frac{c}{a} = \sqrt{1 - \frac{b^2}{a^2}} \quad , \; e > 1$$

b.1) Hipérbola con eje real en el eje y (vertical):

$$\frac{y^2}{a^2} - \frac{x^2}{b^2} = 1$$

- Focos: $(0, \pm c)$
- Asíntotas: $y = \pm \frac{a}{b} x$

Forma paramétrica de la hipérbola centrada en el origen (0,0):

a.2) Hipérbola con eje real en el eje x

$$\begin{cases} x = a \cosh(t) \\ y = b \, senh(t) \end{cases}$$

b.2) Hipérbola con eje real en el eje y

$$\begin{cases} x = b \, senh(t) \\ y = a \cosh(t) \end{cases}$$

Donde $senh(t)$ y $\cosh(t)$ son funciones hiperbólicas.

- La forma canónica es útil para análisis matemático y ubicación de focos y asíntotas.
- La forma paramétrica es práctica para **graficar en Python**, ya que permite generar puntos de la curva fácilmente.
- Igual que con la elipse, se pueden aplicar **rotaciones y traslaciones** para representar hipérbolas en distintas posiciones.

Traslación y rotación de una hipérbola:

- **Traslación**: si el centro es (h, k), entonces:

$$\frac{(x-h)^2}{a^2} - \frac{(y-k)^2}{b^2} = 1$$

- **Rotación**: para rotar la hipérbola un ángulo (antihorario) se aplica la matriz de rotación a los puntos paramétricos:

$$R(\theta)_+ = \begin{pmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & \cos(\theta) \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}$$

O lo que es lo mismo:

$$x' = x\cos(\theta) - ysen(\theta)$$
$$y' = x\,\mathrm{sen}(\theta) + ycos(\theta)$$

Forma paramétrica con rotación y traslación de la hipérbola:

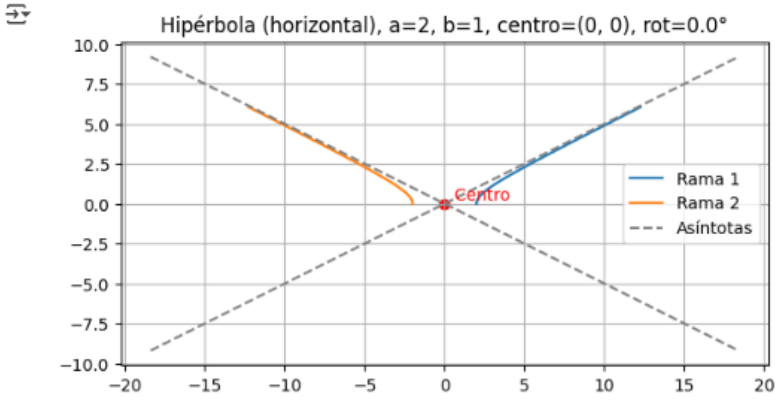
$$\begin{cases} X(t) = h + (a\cosh(t))\cos(\theta) - (\,b\,senh(t))sen(\theta) \\ Y(t) = k + (a\cosh(t))sen(\theta) + (b\,senh(t))\cos(\theta) \end{cases}$$

Ejercicio 2.1.2.1. Diseño de una hipérbola

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

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

def plot_hyperbola(a, b, center=(0,0), theta=0.0,
                   orientation='horizontal', t_max=3.0, points=400,
                   show_asymptotes=True, ax=None):
    """
    Dibuja una hipérbola de la forma:
      - horizontal:  x^2/a^2 - y^2/b^2 = 1
      - vertical:    y^2/b^2 - x^2/a^2 = 1

    Parámetros:
      a, b         : semiejes (a>0, b>0)
      center       : (h,k) centro de la hipérbola
      theta        : rotación en radianes (sentido antihorario)
      orientation : 'horizontal' o 'vertical'
      t_max       : parámetro máximo para cosh/sinh (mayor -> ramas más largas)
      points       : resolución (número de puntos por rama)
      show_asymptotes : dibujar asíntotas (True/False)
       ax           : ejes matplotlib opcionales (si no se pasa, se crea uno
nuevo)
    """
    if a <= 0 or b <= 0:
        raise ValueError("a y b deben ser positivos.")
    if orientation not in ('horizontal', 'vertical'):
```

```
        raise ValueError("orientation debe ser 'horizontal' o 'vertical'.")

    # parámetro t para cosh/sinh (t>=0), generamos puntos para ramas positiva
y negativa
    t = np.linspace(0.0, t_max, points)

    if orientation == 'horizontal':
        # rama derecha
        x_r = a * np.cosh(t)
        y_r = b * np.sinh(t)
        # rama izquierda (simétrica)
        x_l = -a * np.cosh(t)
        y_l = b * np.sinh(t)
        # asíntotas en sistema no rotado: y = ±(b/a) x
        m1, m2 = b / a, -b / a
    else:  # vertical
        # rama superior
        y_r = b * np.cosh(t)
        x_r = a * np.sinh(t)
        # rama inferior
        y_l = -b * np.cosh(t)
        x_l = a * np.sinh(t)
         # asíntotas en sistema no rotado: y = ±(b/a) x  (pero para vertical
interpretamos de forma simétrica)
        m1, m2 = b / a, -b / a

    # unir las ramas (aplicar rotación y traslación)
    xr_r, yr_r = rotate_points(x_r, y_r, theta)
    xr_l, yr_l = rotate_points(x_l, y_l, theta)
    h, k = center
    xr_r += h; yr_r += k
    xr_l += h; yr_l += k

    # Preparar ejes
    if ax is None:
        fig, ax = plt.subplots(figsize=(7,7))

    ax.plot(xr_r, yr_r, label='Rama 1')
    ax.plot(xr_l, yr_l, label='Rama 2')

    # Dibujar asíntotas (en espacio no rotado son líneas a través del origen)
    if show_asymptotes:
            # construiremos un segmento largo para cada asíntota y lo
rotamos/trasladamos
        L = max(np.max(np.abs(xr_r)), np.max(np.abs(yr_r)), 1.0) * 1.5
```

```
        x_line = np.linspace(-L, L, 2)
        # asíntota 1: y = m1 * x
        y_line1 = m1 * x_line
        y_line2 = m2 * x_line
        # rotamos y trasladamos:
        xr_a1, yr_a1 = rotate_points(x_line, y_line1, theta)
        xr_a2, yr_a2 = rotate_points(x_line, y_line2, theta)
        xr_a1 += h; yr_a1 += k
        xr_a2 += h; yr_a2 += k
        ax.plot(xr_a1, yr_a1, '--', color='gray', label='Asíntotas')
        ax.plot(xr_a2, yr_a2, '--', color='gray')

    # Marcar centro y configurar
    ax.scatter([h], [k], color='red')
    ax.text(h, k, '  Centro', color='red', va='bottom')
    ax.set_aspect('equal', adjustable='box')
    ax.grid(True)
    ax.legend()
     ax.set_title(f"Hipérbola ({orientation}), a={a}, b={b}, centro={center},
rot={np.degrees(theta):.1f}°")
    plt.show()
# --- EJEMPLOS de uso ---

# 1) Hipérbola horizontal centrada en el origen:
plot_hyperbola(a=2, b=1, center=(0,0), theta=0.0, orientation='horizontal',
t_max=2.5)

# 2) Hipérbola horizontal, centro desplazado y rotada 30 grados:
plot_hyperbola(a=3, b=1.5, center=(1.0, -0.5), theta=np.deg2rad(30),
               orientation='horizontal', t_max=3.0)

# 3) Hipérbola vertical (transversa en eje y):
plot_hyperbola(a=1.0,     b=2.0,     center=(0,0),     theta=np.deg2rad(-20),
orientation='vertical', t_max=3.0)
```



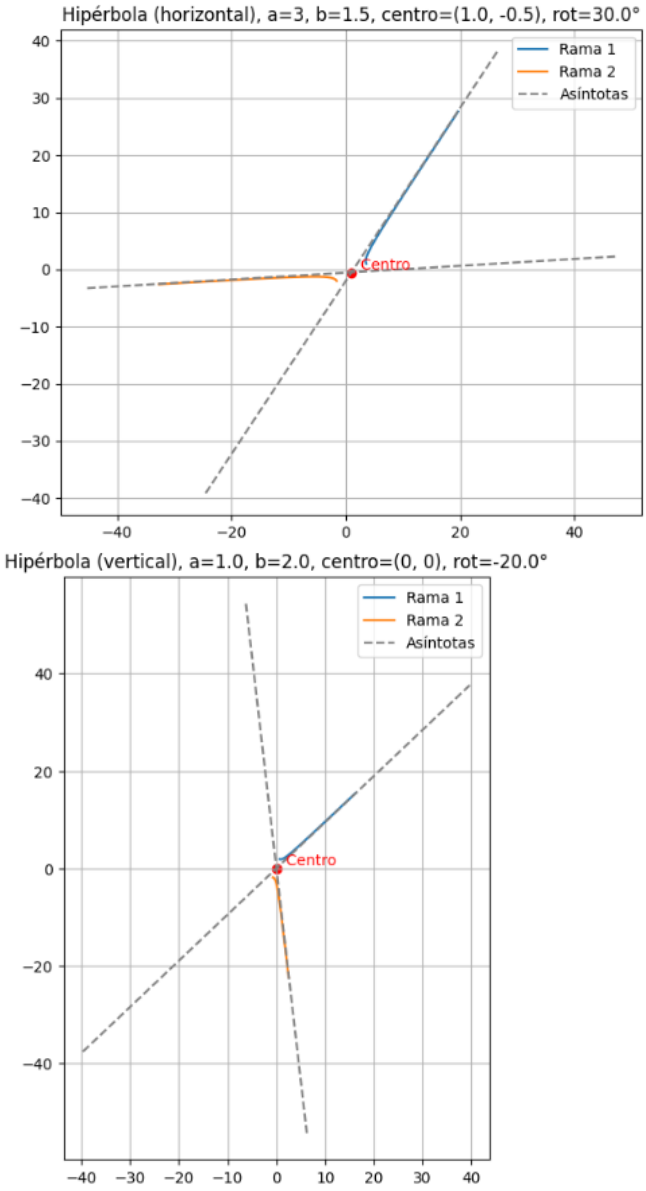


Figura 38. Código y salida en pantalla de las hipérboles diseñadas con sus asíntotas, sin rotar y rotadas.

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()
```

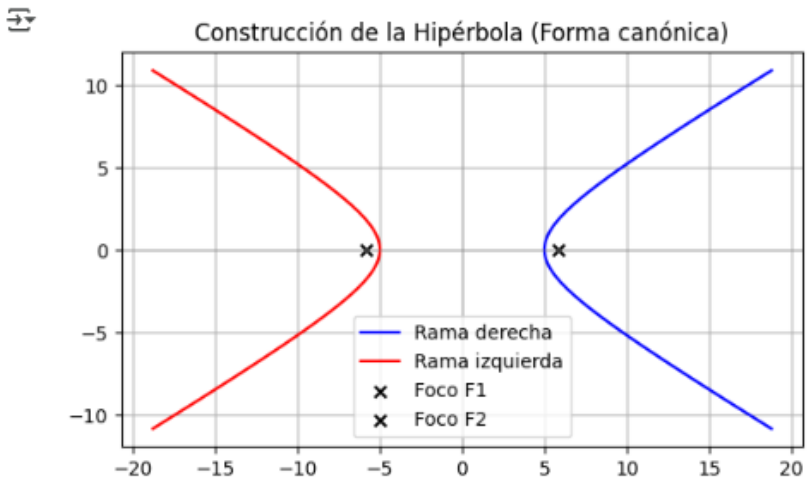


Figura 39. Código y salida de la hipérbola resultado.

Ejercicio Propuesto 2.1.2.3.

- Modificar el script anterior 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.

2.1.3. Diseño de parábolas.

La parábola es una curva abierta y plana, de una sola rama, definiéndose como el lugar geométrico de los puntos del plano que equidistan de un punto fijo llamado **foco** y de una recta fija llamada **directriz**. Tiene un solo eje de simetría, perpendicular a la directriz y que contiene al foco. El punto de intersección entre el eje y la curva es el vértice, siendo la tangente en el mismo, paralela a la directriz y por tanto, perpendicular al eje. El vértice, al ser un punto de la curva equidista del foco y de la directriz. En esta curva, se conoce como parámetro a la longitud de la cuerda que pasando que es paralela a la directriz y pasa por el foco. La circunferencia principal es de radio infinito y es tangente a la curva en su vértice. La circunferencia focal de la parábola, también de radio infinito al existir un solo foco, convirtiéndose por tanto en una recta y por ello coincide con la directriz.

Forma canónica centrada en el origen (0,0):

a.1) Parábola con eje de simetría en el eje X.

$$y^2 = 4ax$$

- El vértice está en el origen $(0,0)$.
- El foco se encuentra en $(a,0)$.
- La directriz es la recta $x = -a$.
- La distancia **foco-vértice** es $\boldsymbol{a}$.

- La distancia entre el foco y la directriz es $\boldsymbol{2a}$.

b.1) Parábola con eje de simetría en el eje Y.

$$x^2 = 4ay$$

- Vértice en el origen.
- Foco: $(0, a)$.
- Directriz: $y = -a$.
- Igual que antes, la distancia **foco-directriz** es $\boldsymbol{2a}$.

Forma paramétrica de la parábola:

Para la parabola $y^2 = 4ax$:

$$\begin{cases} x = at^2 \\ y = 2at \end{cases}$$

Para la parabola $x^2 = 4ay$:

$$\begin{cases} x = 2at \\ y = at^2 \end{cases}$$

- El parámetro $\boldsymbol{a}$ define la **apertura de la parábola**.
- $\boldsymbol{2a}$ corresponde a la distancia entre el foco y la directriz.
- La forma paramétrica es muy útil para **graficar en Python**.

Traslación y rotación de una parábola:

- **Traslación**: si el centro es (h, k), entonces:

$$(y - k)^2 = 4a(x - h)$$

- **Rotación**: para rotar la parábola un ángulo (antihorario) se aplica la matriz de rotación a los puntos paramétricos:

$$R(\theta)_+ = \begin{pmatrix} \cos(\theta) & -sen(\theta) \\ sen(\theta) & \cos(\theta) \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}$$

O lo que es lo mismo:

$$X = x\cos(\theta) - ysen(\theta)$$
$$Y = x\,\mathrm{sen}(\theta) + ycos(\theta)$$

Forma paramétrica con rotación y traslación de la parábola:

$$\begin{cases} X(t) = h + (at^2)\cos(\theta) - (\,2at)sen(\theta) \\ Y(t) = k + (at^2)sen(\theta) + (2at)\cos(\theta) \end{cases}$$

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))
```

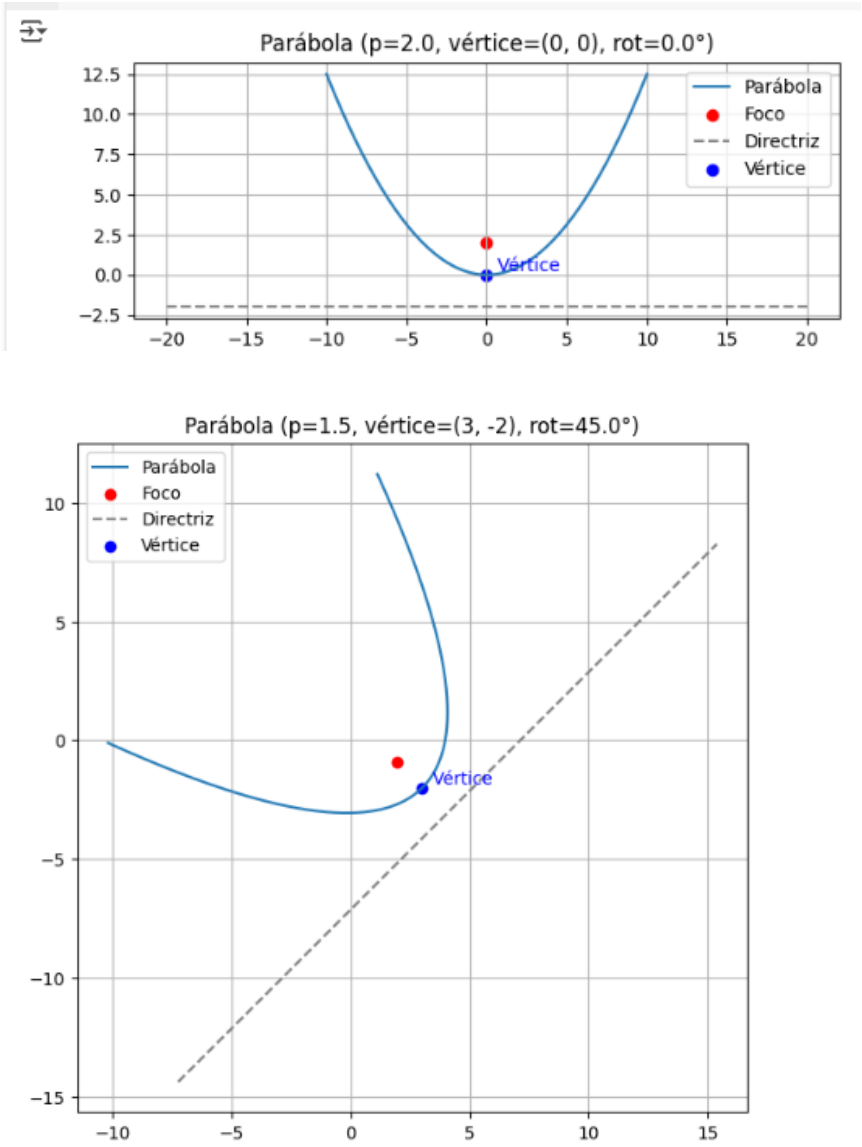


Figura 40. Código y resultado esperado de las parábolas.

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.

2.2. Diseño de triángulos

Un triángulo es una superficie plana limitada por tres rectas que se cortan dos a dos, siendo los puntos de intersección los vértices del triángulo y los segmentos entre cada uno de los vértices, los lados del triángulo. Los lados se designan con letras minúsculas, los vértices con letras mayúsculas opuestas a cada uno de sus lados, y los ángulos con la misma letra mayúscula de cada vértice a la que se le añade un circunflejo ($\hat{A}, \hat{B}, \hat{C}$) o a también se usan letras del alfabeto griego (α, β, ϒ,…).

Clasificación según los lados del triángulo:

- **Equilátero** → los tres lados iguales y ángulos de 60º.
- **Isósceles** → dos lados iguales y los ángulos opuestos a esos lados son iguales.
- **Escaleno** → los tres lados diferentes, todos los ángulos distintos.

Clasificación según los ángulos del triángulo:

- **Acutángulo** → todos los ángulos menores de 90º.
- **Rectángulo** → un ángulo recto (90º).
- **Obtusángulo** → un ángulo mayor de 90º.

Alturas, medianas, bisectrices y mediatrices:

- **Altura** → perpendicular desde un vértice al lado opuesto.
- **Mediana** → une un vértice con el punto medio del lado opuesto.
- **Bisectriz** → divide en dos partes iguales un ángulo interior.
- **Mediatriz** → perpendicular en el punto medio de un lado.

Puntos notables del triángulo:

- **Circuncentro** → intersección de las mediatrices (centro de la circunferencia circunscrita).
- **Incentro** → intersección de las bisectrices (centro de la circunferencia inscrita).
- **Ortocentro** → intersección de las alturas.
- **Baricentro** → intersección de las medianas (centro de gravedad).

Triángulo complementario:

En geometría, el triángulo complementario de un triángulo ABC se define como el triángulo formado por los puntos medios de los segmentos que unen el ortocentro con los vértices.

Es decir, si H es el ortocentro de ABC, entonces el triángulo A'B'C'es el triángulo complementario de ABC.

$$A' = \frac{1}{2}(A + H),\ B' = \frac{1}{2}(B + H),\ C' = \frac{1}{2}(C + H)$$

Propiedades fundamentales de los triángulos:

- Un lado siempre es menor que la suma de los otros dos y mayor que su diferencia.
- A mayor lado se opone siempre mayor ángulo.
- La suma de los ángulos interiores de cada triángulo es de 180º.
- La suma de los ángulos exteriores de un triángulo es siempre de 360º.
- La suma de las longitudes de dos lados siempre es mayor que el tercero:

$$a + b > c, \quad a + c > b, \quad b + c > a$$

Estas propiedades son la base para resolver problemas de geometría plana, trigonometría y construcción geométrica.

Ejercicio 2.2.1. Diseño de un triángulo obtusángulo

```
# Paso 2: Importar librerías
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Nuevas coordenadas: este sí genera un triángulo obtusángulo
A = np.array([0, 0])
B = np.array([6, 0])
C = np.array([3, 1])  # Este da un ángulo obtuso en C

# Función para calcular ángulo en vértice central
def calcular_angulo(p1, p2, p3):
    v1 = p1 - p2
    v2 = p3 - p2
    cos_angle = np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
    angle = np.degrees(np.arccos(cos_angle))
    return angle

# Calcular ángulos internos
angulo_A = calcular_angulo(C, A, B)
angulo_B = calcular_angulo(A, B, C)
angulo_C = calcular_angulo(A, C, B)

# Mostrar ángulos
print(f"Ángulo en A: {angulo_A:.2f}°")
print(f"Ángulo en B: {angulo_B:.2f}°")
print(f"Ángulo en C: {angulo_C:.2f}°")  # Este debe ser obtuso
```

```
# Graficar el triángulo
x = [A[0], B[0], C[0], A[0]]
y = [A[1], B[1], C[1], A[1]]

plt.figure(figsize=(6,6))
plt.plot(x, y, marker='o', color='black')
plt.text(*A, 'A', fontsize=12, ha='right', va='top')
plt.text(*B, 'B', fontsize=12, ha='left', va='top')
plt.text(*C, 'C', fontsize=12, ha='center', va='bottom')
plt.title('Triángulo Obtusángulo (ángulo en C > 90°)')
plt.axis('equal')
plt.grid(True)
plt.show()
```

```
Ángulo en A: 18.43°
Ángulo en B: 18.43°
Ángulo en C: 143.13°
```

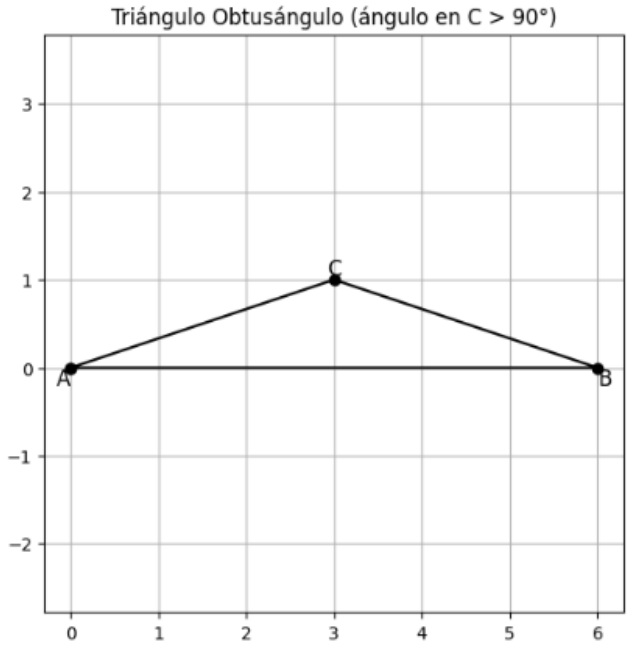


Figura 41. Código y salida esperada en el diseño de un triángulo obtusángulo.

Ejercicio 2.2.2. Diseño de un triángulo acutángulo isósceles:

- **Define los vértices de un triángulo acutángulo en el plano.**
- **Calcula sus lados y ángulos internos con trigonometría.**
- **Comprueba que los tres ángulos son agudos.**
- **Dibuja el triángulo.**

```
import numpy as np
```

```
import matplotlib.pyplot as plt

def distancia(P, Q):
    """Distancia entre dos puntos en el plano"""
    return np.sqrt((Q[0]-P[0])**2 + (Q[1]-P[1])**2)

def angulo(a, b, c):
    """Ley de cosenos: ángulo opuesto a lado a, dados lados a, b, c"""
    return np.arccos((b**2 + c**2 - a**2)/(2*b*c))

def triangulo_acutangulo(A, B, C):
    # Lados
    a = distancia(B,C)  # opuesto A
    b = distancia(A,C)  # opuesto B
    c = distancia(A,B)  # opuesto C

    # Ángulos
    ang_A = angulo(a,b,c)
    ang_B = angulo(b,a,c)
    ang_C = angulo(c,a,b)

    # Convertir a grados
    ang_A_deg = np.degrees(ang_A)
    ang_B_deg = np.degrees(ang_B)
    ang_C_deg = np.degrees(ang_C)

    print("=== Triángulo Acutángulo ===")
    print(f"Lados: a={a:.2f}, b={b:.2f}, c={c:.2f}")
    print(f"Ángulo A = {ang_A_deg:.2f}°")
    print(f"Ángulo B = {ang_B_deg:.2f}°")
    print(f"Ángulo C = {ang_C_deg:.2f}°")

    if all(ang < 90 for ang in [ang_A_deg, ang_B_deg, ang_C_deg]):
        print("Es un triángulo acutángulo.")
    else:
        print("No es acutángulo.")

    # --- Gráfica ---
    fig, ax = plt.subplots()
    x = [A[0], B[0], C[0], A[0]]
    y = [A[1], B[1], C[1], A[1]]
    ax.plot(x,y,'b-',lw=2)

    # Vértices con etiquetas
    for P, name in zip([A,B,C], ["A","B","C"]):
```

```
        ax.scatter(*P, color="red")
        ax.text(P[0]+0.1, P[1]+0.1, name)

    ax.set_aspect("equal")
    ax.grid(True)
    ax.set_title("Triángulo Acutángulo")
    plt.show()


# --- Ejemplo ---
# Definimos un triángulo con coordenadas
A = (0,0)
B = (4,0)
C = (2,3)

triangulo_acutangulo(A,B,C)
```

```
=== Triángulo Acutángulo ===
Lados: a=3.61, b=3.61, c=4.00
Ángulo A = 56.31°
Ángulo B = 56.31°
Ángulo C = 67.38°
Es un triángulo acutángulo.
```

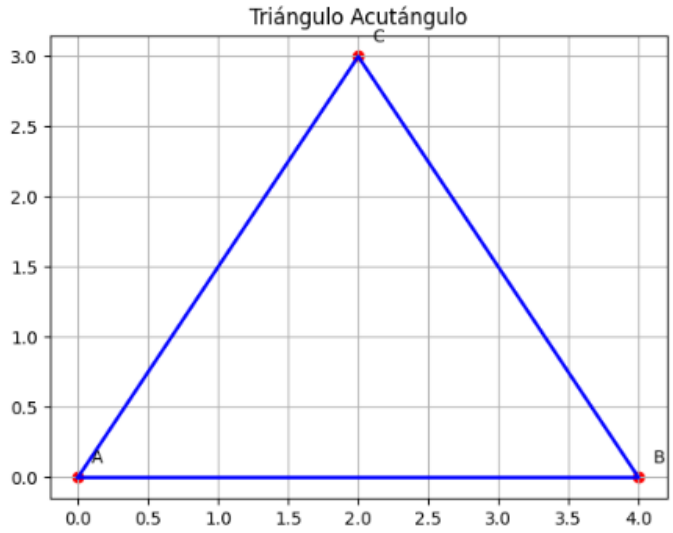


Figura 42. Código y salida de triángulo acutángulo.

Ejercicio 2.2.3. Diseño de un triángulo rectángulo:

- **Definir las longitudes de los catetos.**
- **Calcular la hipotenusa con el teorema de Pitágoras.**
- **Calcular ángulos internos.**
- **Mostrar los resultados numéricos.**
- **Gráfica del triángulo rectángulo.**

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

```
def triangulo_rectangulo(cateto_a, cateto_b):
    # Hipotenusa
    hipotenusa = np.sqrt(cateto_a**2 + cateto_b**2)

    # Ángulos en radianes
    ang_A = np.arctan2(cateto_b, cateto_a)  # ángulo en A (opuesto a cateto
b)
    ang_B = np.arctan2(cateto_a, cateto_b)  # ángulo en B (opuesto a cateto
a)
    ang_C = np.pi/2                         # recto

    # Convertir a grados
    ang_A_deg = np.degrees(ang_A)
    ang_B_deg = np.degrees(ang_B)

    print("=== Triángulo rectángulo ===")
    print(f"Cateto a = {cateto_a}")
    print(f"Cateto b = {cateto_b}")
    print(f"Hipotenusa c = {hipotenusa:.2f}")
    print(f"Ángulo A = {ang_A_deg:.2f}°")
    print(f"Ángulo B = {ang_B_deg:.2f}°")
    print(f"Ángulo C = 90°")

    # --- Gráfica ---
    A = (0,0)
    B = (cateto_a,0)
    C = (0,cateto_b)

    fig, ax = plt.subplots()
    x = [A[0], B[0], C[0], A[0]]
    y = [A[1], B[1], C[1], A[1]]
    ax.plot(x,y,'b-',lw=2)

    # Vértices
    for P, name in zip([A,B,C], ["A(0,0)", f"B({cateto_a},0)",
f"C(0,{cateto_b})"]):
        ax.scatter(*P, color="red")
        ax.text(P[0]+0.1, P[1]+0.1, name)

    # Cuadro para ángulo recto
    ax.plot([0.3,0.3,0,0], [0,0.3,0.3,0], 'k-')

    ax.set_aspect("equal")
    ax.grid(True)
```

```
    ax.set_title("Triángulo rectángulo")
    plt.show()

# --- Ejemplo ---
triangulo_rectangulo(3,4)
```

```
=== Triángulo rectángulo ===
Cateto a = 3
Cateto b = 4
Hipotenusa c = 5.00
Ángulo A = 53.13°
Ángulo B = 36.87°
Ángulo C = 90°
```

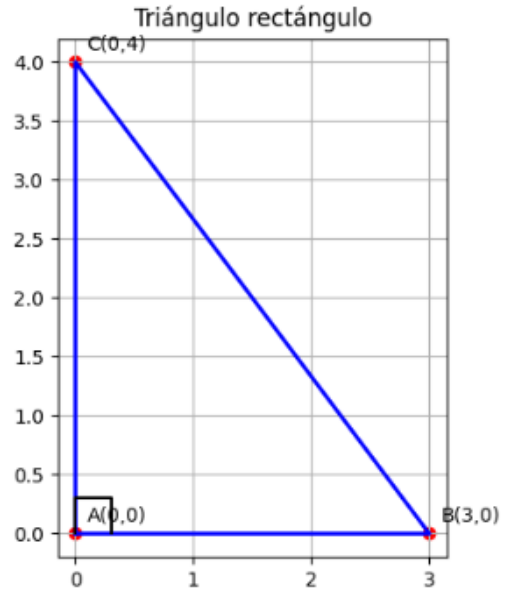


Figura 43. Código y salida de triángulo rectángulo.

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.**

Ejercicio 2.2.5. Cálculo del triángulo complementario de otro dado.

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

def pendiente(P, Q):
    if Q[0] == P[0]:
        return None
    return (Q[1]-P[1])/(Q[0]-P[0])

def pie_perpendicular(P, A, B):
```

```
    """Devuelve el pie de la perpendicular desde P a la recta AB"""
    x1, y1 = A; x2, y2 = B; x0, y0 = P
    if x1 == x2:  # AB vertical
        return np.array([x1, y0])
    if y1 == y2:  # AB horizontal
        return np.array([x0, y1])
    m = (y2-y1)/(x2-x1)
    n = y1 - m*x1
    # pendiente perpendicular
    mp = -1/m
    np_ = y0 - mp*x0
    # intersección de rectas
    xi = (np_ - n)/(m - mp)
    yi = m*xi + n
    return np.array([xi, yi])

def ortocentro(A,B,C):
    """Calcula el ortocentro de un triángulo ABC"""
    # Altura desde A hacia BC
    D = pie_perpendicular(A, B, C)
    # Altura desde B hacia AC
    E = pie_perpendicular(B, A, C)
    # Intersección de alturas: A-D y B-E
    def recta(P,Q):
        if P[0]==Q[0]: return ("x", P[0])
        m = (Q[1]-P[1])/(Q[0]-P[0])
        n = P[1]-m*P[0]
        return (m,n)
    rAD = recta(A,D)
    rBE = recta(B,E)
    if rAD[0]=="x":
        xh = rAD[1]; yh = rBE[0]*xh + rBE[1]
    elif rBE[0]=="x":
        xh = rBE[1]; yh = rAD[0]*xh + rAD[1]
    else:
        xh = (rBE[1]-rAD[1])/(rAD[0]-rBE[0])
        yh = rAD[0]*xh + rAD[1]
    return np.array([xh,yh])

def plot_ortico(A,B,C):
    A = np.array(A); B = np.array(B); C = np.array(C)
    H = ortocentro(A,B,C)

    # Pies de alturas
    D = pie_perpendicular(A,B,C)
```

```
    E = pie_perpendicular(B,A,C)
    F = pie_perpendicular(C,A,B)

    # Plot
    fig, ax = plt.subplots()
    # Triángulo principal
    ax.plot([A[0],B[0],C[0],A[0]], [A[1],B[1],C[1],A[1]], 'b-',
label="Triángulo ABC")
    # Ortocentro
    ax.scatter(*H, color='red', label="Ortocentro H")
    ax.text(H[0]+0.1,H[1]+0.1,"H", color="red")
    # Alturas
    ax.plot([A[0],D[0]],[A[1],D[1]],'g--',alpha=0.7)
    ax.plot([B[0],E[0]],[B[1],E[1]],'g--',alpha=0.7)
    ax.plot([C[0],F[0]],[C[1],F[1]],'g--',alpha=0.7)
    # Triángulo órtico
    ax.plot([D[0],E[0],F[0],D[0]],[D[1],E[1],F[1],D[1]],'m-',label="Triángulo
órtico DEF")

    # Marcar 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)
    for P,name in zip([D,E,F],[ "D","E","F"]):
        ax.scatter(*P, color='orange')
        ax.text(P[0]+0.1,P[1]+0.1,name)

    ax.set_aspect("equal")
    ax.grid(True)
    ax.legend()
    ax.set_title("Triángulo y su triángulo órtico (pies de las alturas)")
    plt.show()

# --- Ejemplo ---
A = (0,0)
B = (6,2)
C = (2,5)

plot_ortico(A,B,C)
```

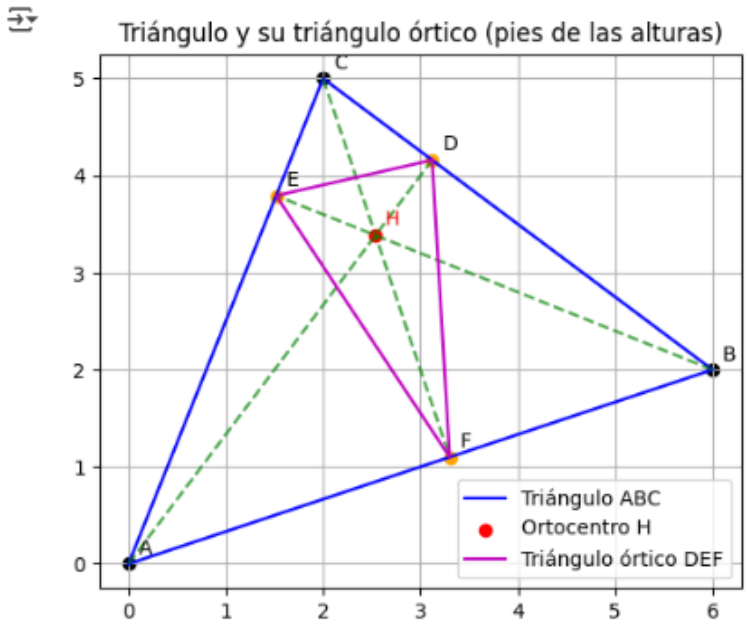


Figura 44. Código y salida de triángulo órtico.

Ejercicio 2.2.6. Hallar el cincuncentro y circunferencia circunscrita de un triángulo.

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

def circuncentro(A, B, C):
    """Devuelve el circuncentro y radio del triángulo ABC."""
    A, B, C = np.array(A), np.array(B), np.array(C)

    # Vectores de lados
    D = B - A
    E = C - A

    # Matriz para resolver sistema
    M = np.array([[D[0], D[1]], [E[0], E[1]]])
    b = np.array([np.dot(D, (B+A))/2 - np.dot(D, A),
                  np.dot(E, (C+A))/2 - np.dot(E, A)])

    try:
        # Resolver sistema lineal
        O = np.linalg.solve(M.T, b)
    except np.linalg.LinAlgError:
        raise ValueError("Los puntos son colineales, no existe
circuncentro.")
```

```
    # Radio (distancia de O a un vértice)
    R = np.linalg.norm(O - A)
    return O, R

def plot_circuncentro(A, B, C):
    """Dibuja triángulo, circuncentro y circunferencia circunscrita."""
    A, B, C = np.array(A), np.array(B), np.array(C)
    O, R = circuncentro(A,B,C)

    fig, ax = plt.subplots()
    # Triángulo
    ax.plot([A[0],B[0],C[0],A[0]],[A[1],B[1],C[1],A[1]],'b-',label="Triángulo
ABC")

    # Circuncentro
    ax.scatter(*O, color='red', label='Circuncentro O')
    ax.text(O[0]+0.1, O[1]+0.1, "O", color='red')

    # Circunferencia circunscrita
    circ = plt.Circle(O, R, color='green', fill=False, linestyle='--',
label="Circunferencia circunscrita")
    ax.add_patch(circ)

    # 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)

    ax.set_aspect("equal")
    ax.grid(True)

    # Mover leyenda fuera del gráfico
    ax.legend(loc="center left", bbox_to_anchor=(1, 0.5))

    ax.set_title("Circuncentro y Circunferencia Circunscrita")
    plt.show()

# --- Ejemplo ---
A = (0,0)
B = (6,2)
C = (2,5)

plot_circuncentro(A,B,C)
```

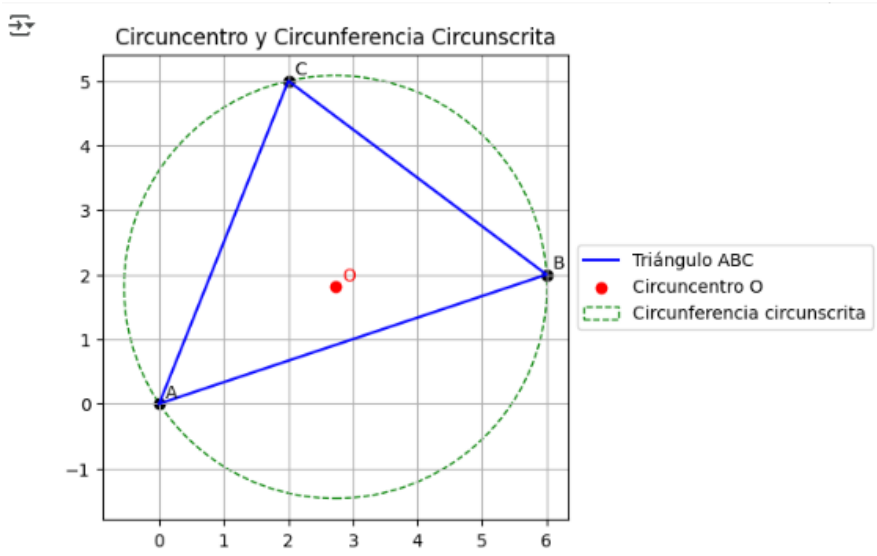


Figura 45. Código y salida de circunferencia circunscrita.

2.3. Diseño de piezas industriales

El modelado de componentes mecánicos es el proceso de **representar piezas o sistemas mecánicos en un entorno digital o gráfico**, generalmente usando software CAD (Computer-Aided Design). Permite diseñar, analizar y fabricar piezas de manera precisa antes de su producción real. Esto permite describir la forma, dimensiones, tolerancias de un componente, comprobar que la pieza cumple con su propósito dentro de un mecanismo, verificar esfuerzos, deformaciones, vibraciones y vida útil. Todo ello con el fin de generar planos técnicos normalizados para fabricación y montaje.

Componentes mecánicos comunes en el modelado:

Elementos de unión: pernos, tornillos, tuercas, arandelas, remaches.
Elementos de transmisión: engranajes, poleas, ejes, acoplamientos.
Elementos de soporte: cojinetes, rodamientos, chumaceras.
Piezas estructurales: placas, perfiles, bastidores.

El modelado debe seguir estándares internacionales para que los diseños sean interpretados en cualquier parte del mundo:

ISO (International Organization for Standardization).
ANSI/ASME (American National Standards Institute).
DIN (Deutsches Institut für Normung).

Ejercicio 2.3.1. Diseño de una pletina perforada en 2 dimensiones (2D).

```
# Dimensiones de la pletina (en milímetros)
import matplotlib.pyplot as plt
largo = 100
```

```
ancho = 40

# Coordenadas de los agujeros
agujeros = [(20, 20), (80, 20)]

# Crear la figura y los ejes
fig, ax = plt.subplots()

# Dibujar el rectángulo base
rect = plt.Rectangle((0, 0), largo, ancho, edgecolor='black',
facecolor='lightgray')
ax.add_patch(rect)

# Dibujar los agujeros
for (x, y) in agujeros:
    circle = plt.Circle((x, y), 5, color='white', ec='black')
    ax.add_patch(circle)

# Ajustar la vista
ax.set_aspect('equal')
ax.set_xlim(-10, largo + 10)
ax.set_ylim(-10, ancho + 10)
ax.set_title("Pletina con dos agujeros pasantes - Dibujada con Python")
plt.grid(True)
plt.show()
```

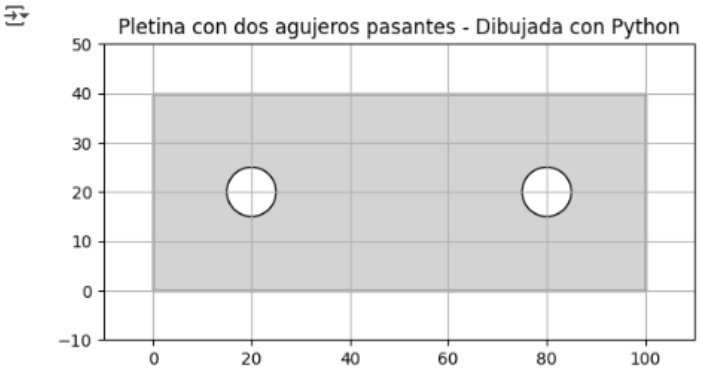


Figura 46. Salida esperada. Pletina con dos agujeros pasantes - Dibujada con Python.

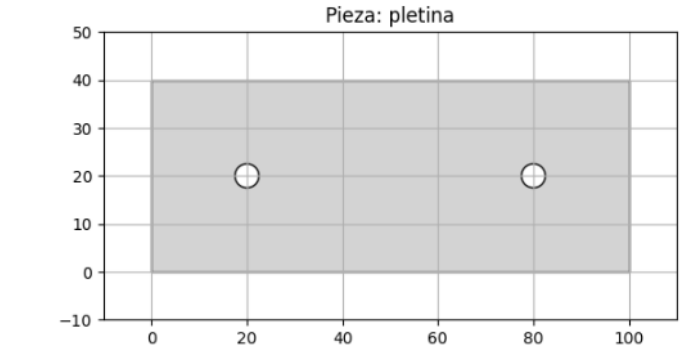
Ejercicio 2.3.2. Diseño de tres piezas en 2 dimensiones (2D) cargando archivo .csv.

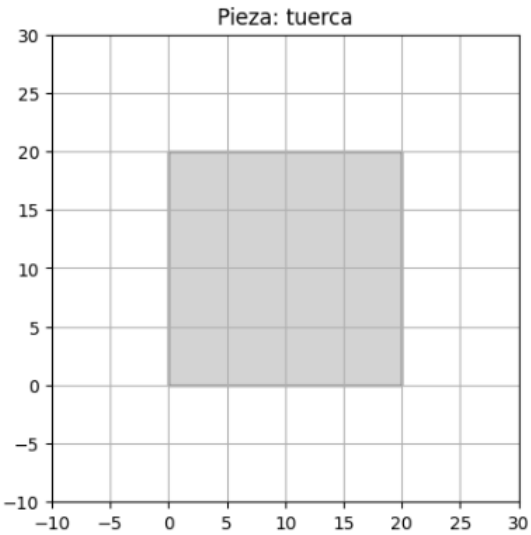
```
# Paso 1: Montamos nuestro Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Paso 2: Importamos librerías que necesitaremos
import pandas as pd
```

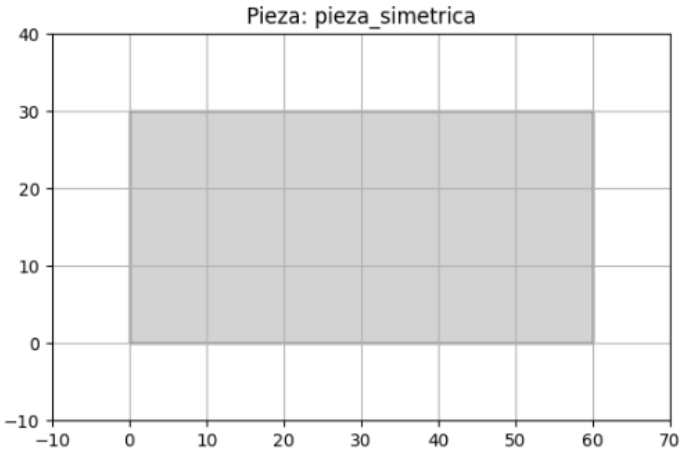
```
import matplotlib.pyplot as plt
import numpy as np
import ast

# Paso 3: Leemos el archivo desde tu Google Drive
ruta = '/content/drive/MyDrive/datos/datospiezas (1).csv'  # Ajusta la ruta si
está en una subcarpeta
df = pd.read_csv(ruta)

# Paso 4: Función para dibujar una pieza rectangular
def   dibujar_pieza_rectangular(nombre,   largo,   ancho,   diametro_agujero,
coord_agujeros):
    fig, ax = plt.subplots()
       rect   =   plt.Rectangle((0,   0),   largo,   ancho,   edgecolor='black',
facecolor='lightgray')
    ax.add_patch(rect)

    if coord_agujeros and coord_agujeros != "[]":
        coords = ast.literal_eval(coord_agujeros)
        for (x, y) in coords:
             circle = plt.Circle((x, y), diametro_agujero / 2, color='white',
ec='black')
            ax.add_patch(circle)

    ax.set_aspect('equal')
    ax.set_xlim(-10, largo + 10)
    ax.set_ylim(-10, ancho + 10)
    ax.set_title(f"Pieza: {nombre}")
    plt.grid(True)
    plt.show()

# Paso 5: Función para dibujar una tuerca octogonal
def dibujar_tuerca_octogonal(nombre, largo, ancho, diametro_agujero):
    fig, ax = plt.subplots()
    centro_x, centro_y = largo / 2, ancho / 2
    radio = min(largo, ancho) / 2
    angulos = np.linspace(0, 2 * np.pi, 9)[:-1] + np.pi / 8
    x_pts = centro_x + radio * np.cos(angulos)
    y_pts = centro_y + radio * np.sin(angulos)
    ax.fill(x_pts, y_pts, edgecolor='black', facecolor='lightgray')

      agujero   =   plt.Circle((centro_x,   centro_y),   diametro_agujero   /   2,
color='white', ec='black')
    ax.add_patch(agujero)
```

```
    ax.set_aspect('equal')
    ax.set_xlim(centro_x - radio - 10, centro_x + radio + 10)
    ax.set_ylim(centro_y - radio - 10, centro_y + radio + 10)
    ax.set_title(f"Tuerca Octogonal")
    plt.grid(True)
    plt.show()

# Paso 6: Dibujar todas las piezas según su tipo
for _, row in df.iterrows():
    nombre = row["nombre_pieza"]
    if nombre == "tuerca_octogonal":
          dibujar_tuerca_octogonal(nombre, row["largo_mm"], row["ancho_mm"],
row["diametro_agujero_mm"])
    else:
         dibujar_pieza_rectangular(nombre, row["largo_mm"], row["ancho_mm"],
row["diametro_agujero_mm"], row["coordenadas_agujeros"])
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```







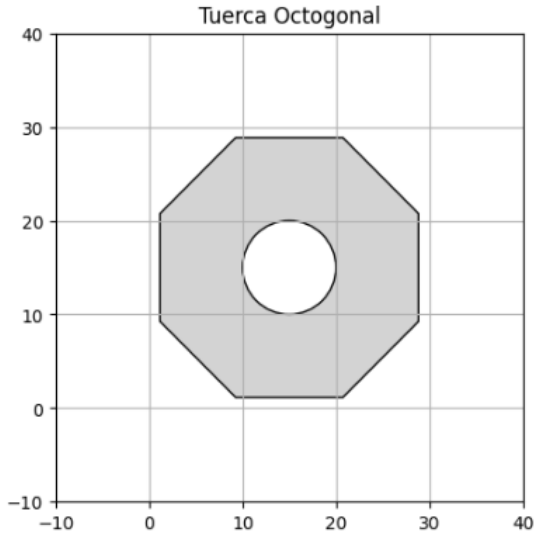


Figura 47. Salidas esperadas de los modelos cargados en .CSV.

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)
```

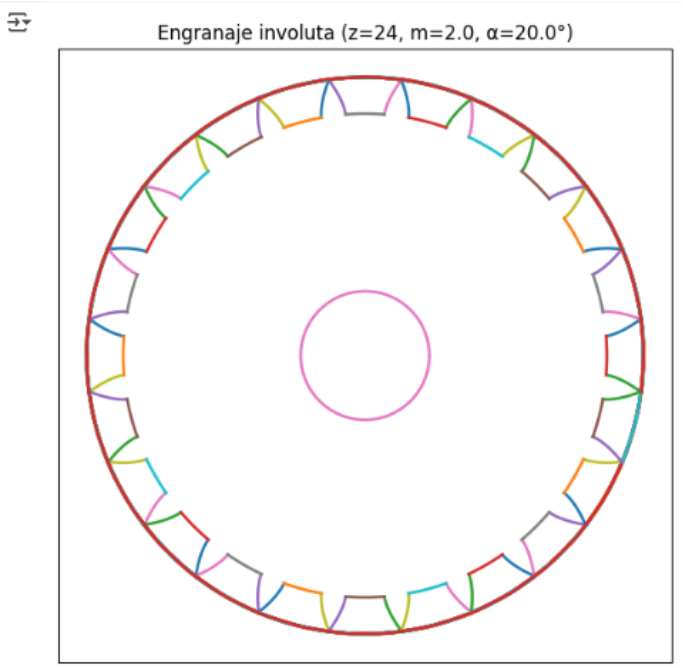


Figura 48. Código y salidda esperada del engranaje.

Ejercicio Propuesto 2.3.4.

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

Módulo 3. Ejercicios Extra Propuestos.

3.1. Ejercicios Extra:

Ejercicio Extra 3.1.1.: Escribe un script en Python que dibuje un cuadrado de 100 mm de lado utilizando matplotlib.
- *Pistas:* Usa listas o arrays para coordenadas x e y; recuerda cerrar la figura repitiendo el primer punto.

Ejercicio Extra 3.1.2.: Modifica el programa anterior para que el tamaño del cuadrado se pueda introducir por teclado.

Ejercicio Extra 3.1.3.: Crea un script que dibuje un hexágono regular de radio dado y lo exporte a un archivo **DXF** utilizando ezdxf.

Ejercicio Extra 3.1.4.: Programa una función poligono_regular(n, radio) que genere y muestre cualquier polígono regular dado su número de lados y radio circunscrito.

Ejercicio Extra 3.1.5.: Dibuja una elipse de semiejes a = 60 mm y b = 40 mm utilizando sus ecuaciones paramétricas, marcando sus focos en la figura. Solicita al usuario los valores de a y b y genera el archivo **DXF** de la elipse.

Ejercicio Extra 3.1.6.: Representa una parábola con vértice en el origen y ecuación $y^2 = 4px$ para un valor de p dado. Modifica el código para que el vértice y el eje de simetría se puedan desplazar a coordenadas arbitrarias.

Ejercicio Extra 3.1.7.: Dibuja la hipérbola $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$ con $a = 50\,mm$ y $b = 30\,mm$ utilizando un parámetro t en radianes. Programa una versión que trace también las asíntotas de la hipérbola.

Ejercicio Extra 3.1.8.: Diseña una brida circular con 6 taladros equidistantes y genera su archivo DXF. Modela en Python un perfil en "L" con medidas dadas y exporta el dibujo a DXF para su apertura en AutoCAD.

Ejercicio Extra 3.1.9.: Desarrolla un script que permita:
- Dibujar una pieza con al menos una curva cónica y dos figuras poligonales.
- Exportar el resultado a DXF.

Ejercicio Extra 3.1.10.: Desarrollo de un proyecto completo que integre el diseño y exportación de una pieza o figura compleja utilizando Python.

3.2. Archivos solución

Solución Ejercicio 3.1.1.ipynb
Solución Ejercicio 3.1.2.ipynb
Solución Ejercicio 3.1.3.ipynb
Solución Ejercicio 3.1.4.ipynb
Solución Ejercicio 3.1.5.ipynb
Solución Ejercicio 3.1.6.ipynb
Solución Ejercicio 3.1.7.ipynb
Solución Ejercicio 3.1.8.ipynb
Solución Ejercicio 3.1.9.ipynb
Solución Ejercicio 3.1.10.ipynb

ANEXO I. Solución de ejercicios propuestos.

SOLUCIÓN DE EJERCICIOS PROPUESTOS EN EL MÓDULO 2

Ejemplo: **Pentágono regular.**

```
import numpy as np

# Número de lados y radio
n = 5
r = 4
angulos = np.linspace(0, 2*np.pi, n+1)

x = r * np.cos(angulos)
y = r * np.sin(angulos)

plt.figure()
plt.plot(x, y, 'g-', linewidth=2, label="Pentágono")
plt.axis('equal')
plt.legend()
plt.title("Pentágono regular")
plt.show()
```

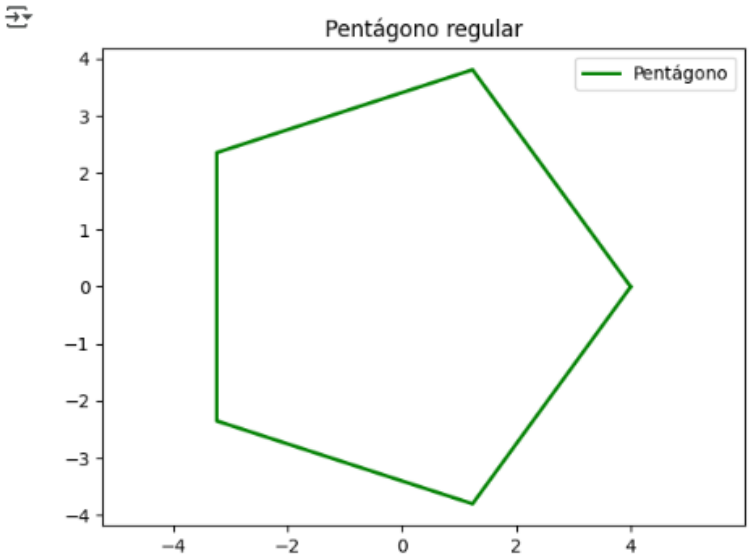


Figura 24. Salida esperada de un pentágono regular.

Ejercicio Propuesto 1.3.3.1.

• Cambiar los datos en las instrucciones del Ejemplo dado para formar otro polígono regular, de 7 y 8 lados.
• Modificar el color del polígono.

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

def dibujar_poligono_regular(lados=6, radio=1, centro=(0, 0)):
    if lados < 3:
        raise ValueError("Un polígono debe tener al menos 3 lados.")

    # Ángulos igualmente espaciados entre 0 y 2π
    angulos = np.linspace(0, 2 * np.pi, lados + 1)  # +1 para cerrar el
polígono

    # Coordenadas
    x = centro[0] + radio * np.cos(angulos)
    y = centro[1] + radio * np.sin(angulos)

    # Graficar
    plt.figure(figsize=(5, 5))
    plt.plot(x, y, marker='o', color='black')
    plt.title(f'{lados}-lados (polígono regular)')
    plt.axis('equal')
    plt.grid(True)
    plt.show()

# Ejemplo: dibujar un hexágono
dibujar_poligono_regular(lados=6, radio=50)
```

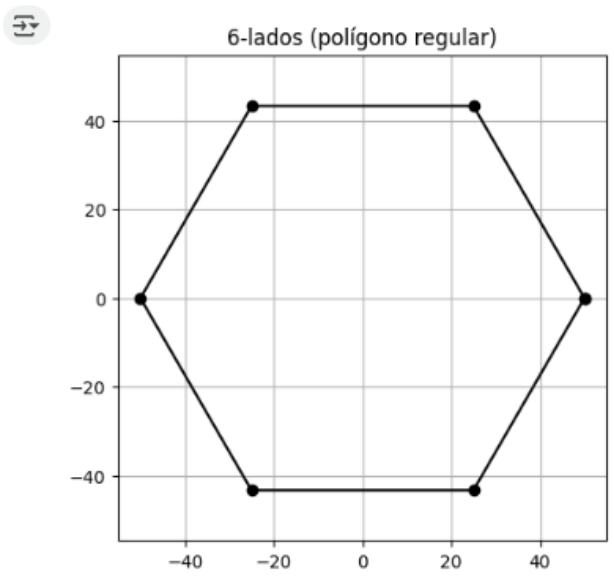


Figura 24.2. Código y salida esperada de polígono de 6 lados.

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()
```

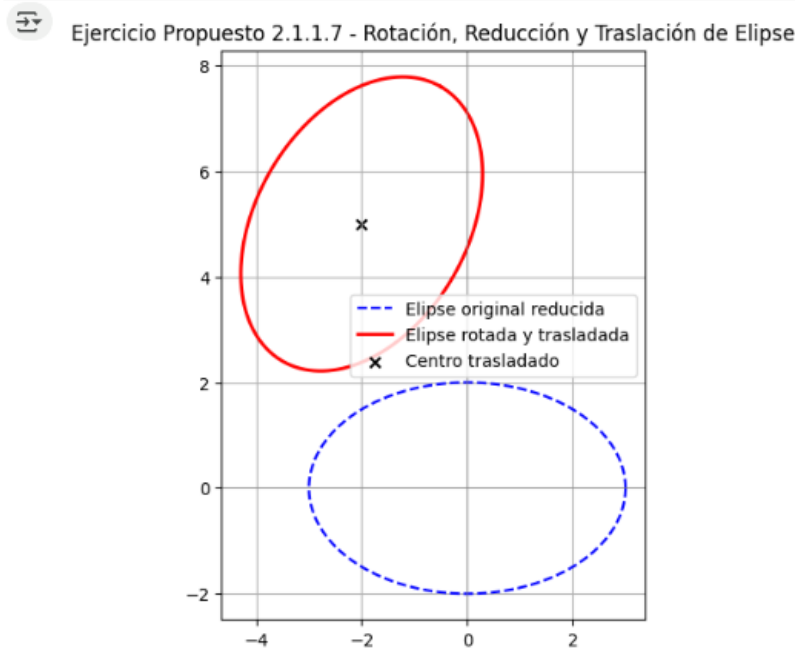


Figura 36.2. Código y resultado para rotar y trasladar una elipse.

Ejercicio 2.1.2.3. Diseño de una hipérbola script en Python para construir una hipérbola mediante su forma canónica y ecuaciones paramétricas.

```
# 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
```

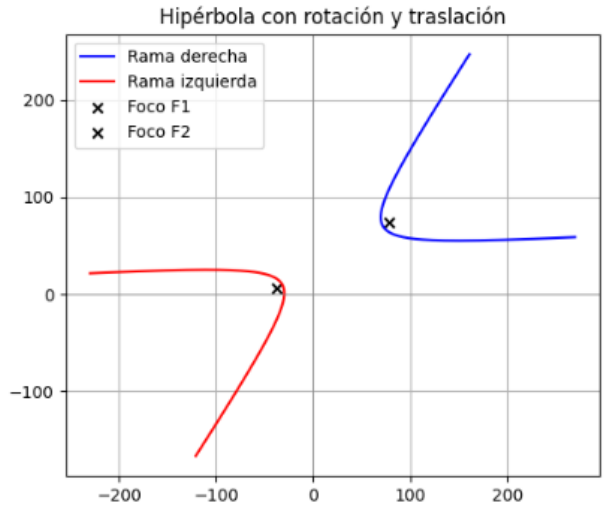


Figura 39.2. Código y salida esperada de hipérbola rotada.

```
# Ejercicio 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()
```

```
Requirement already satisfied: ezdxf in /usr/local/lib/python3.12/dist-packages (1.4.2)
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)
```

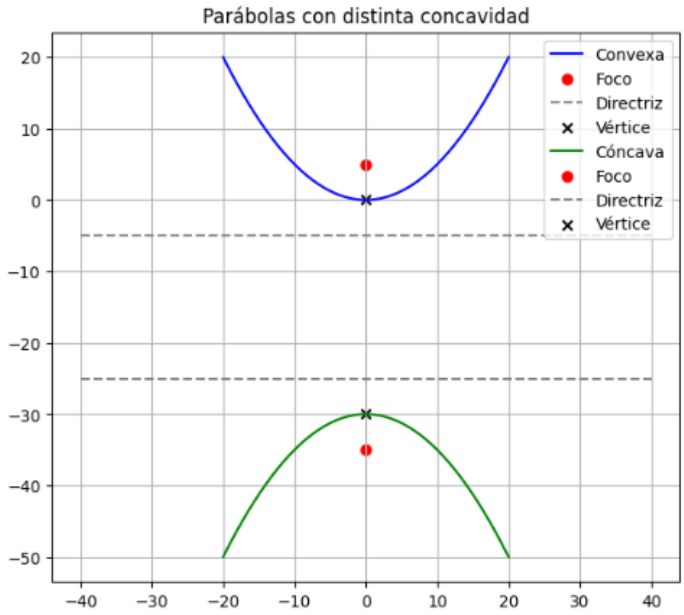


Figura 40.2. Código y resultado esperado de parábola modificada.

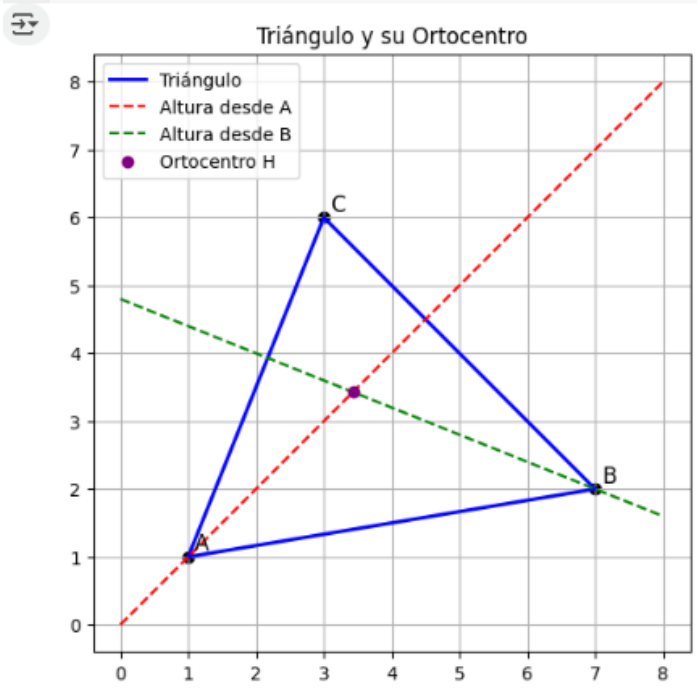
Ejercicio Propuesto 2.2.4.

- **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()
```



Ejercicio Propuesto 2.3.3.

- **Modificar el código anterior 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
```

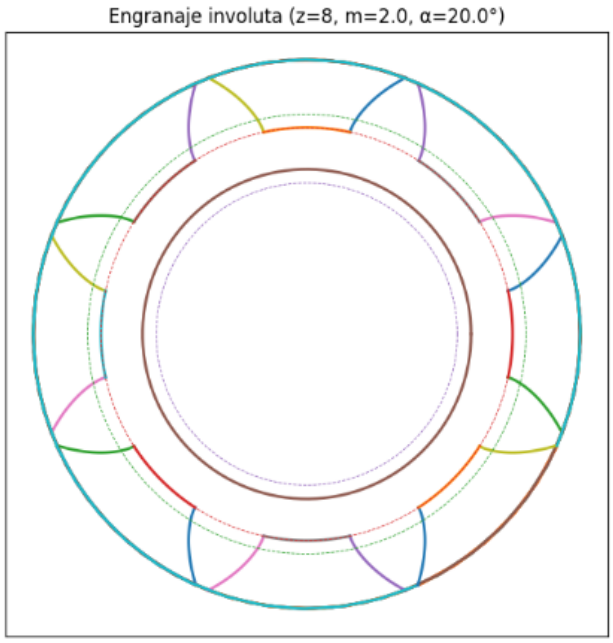


Figura 48.2. Código y salida esperada del engranaje propuesto.