PULSOTEC: REVISTA CIENTÍFICA MULTIDISCIPLINARIA
AÑO 2 / VOLUMEN 2 / AGOSTO 2026 ISSN:3122-4032

Representación Geométrica: Curvas y Superficies Mediante Spyder

Geometric Representation: Curves And Surfaces Using Spyder

Luis Gabriel Mateo Mejía[1] , Karla Georgina Olivo Beltrán[2],

[1] Tecnológico Superior P´urhépecha, Cherán, Michoacán, México.
Doctorado en Sistemas Computacionales, Universidad de Ciencias Digitales, (UDVINCI), México.
[2] Instituto Tecnológico Superior P´urhépecha, Cherán, Michoacán, México.
Doctorado en Sistemas Computacionales, Universidad del Sur, (UNISUR), Tuxtla Gutiérrez, Chiapas, México.

ORCID: [1]https://orcid.org/0000-0001-8289-1377, [2] https://orcid.org/0000-0003-4673-300X

Autores de correspondencia: gabriel.mm@purhepecha.tecnm.mx, karla.ob@purhepecha.tecnm.mx

Resumen:
La finalidad de este artículo de divulgación es ayudar a los profesionales e investigadores en el uso de la programación en 'Spyder', que es de código abierto y gratuito, específicamente Python; funcionando como IDE. Además de señalar la importancia de las ciencias básicas en el proceso educativo, en este documento se pretende vincular su carácter de utilidad y aplicabilidad a los fenómenos de ingeniería que se expresan mediante ecuaciones matemáticas integradas al cálculo vectorial, integral, diferencial y de ecuaciones diferenciales, así como a sus bases en álgebra lineal. Para tal efecto se muestran ejemplos y explicaciones de código en matemáticas aplicadas a los siguientes elementos: cálculo de curvas en el plano, curvas dependientes de un parámetro, curvas en coordenadas polares, curvas en el espacio tridimensional, y gráfica de una superficie. Se observa que su explicación del uso de código así como las formas de integración con IA, contribuye, tanto a docentes como alumnos, a trabajar de forma más autónoma y congruente, así como al aprendizaje significativo e inserto en la productividad y desarrollo social.

Palabras Clave: Entorno de desarrollo integrado, programación, código Python, ciencias básicas, Investigación tecnológica y vinculación.

Abstract:
The purpose of this article is to assist professionals and researchers in using 'Spyder', an open-source and free Python programming language, as an IDE (Integrated Development Environment) with AI capabilities. In addition to highlighting the importance of basic sciences in the educational process, this document aims to connect their usefulness and applicability to engineering phenomena expressed through mathematical equations integrated with vector, integral, and differential calculus, as well as their foundations in linear algebra. To this end, examples and explanations of applied mathematical code are presented for the following elements: calculating curves in the plane, curves dependent on a parameter, curves in polar coordinates, curves in three-dimensional space, and surface graphing. It is observed that the explanation of code usage and its integration with artificial intelligence contributes to more autonomous and coherent work for both teachers and students, fostering meaningful learning embedded in the surrounding social productivity and development.

Keywords: Integrated development environment, programming, Python code, basic sciences, technological research and outreach.

Introducción

Siguiendo los parámetros de crecimiento y aplicación integral del nuevo modelo educativo del TecNM, [Tecnológico Nacional de México. 2024], se presenta este documento para continuar con el trabajo de concientización hacia una justicia social e integrativa, como es el caso particular que permite hacerlo las ingenierías, las ciencias básicas y su interacción interdisciplinar con la sociedad moderna y globalizada que vive nuestro país actualmente. Por otra parte, este trabajo viene a consolidar los proyectos integradores al interior de las academias de ciencias básicas que plantean la necesidad de complementar el aprendizaje de las matemáticas con su aplicación a la programación y a la ingeniería, [Tecnológico Nacional de México. 2015]. Por tal motivo, en el presente proyecto se propone como objetivo y meta la divulgación del uso de una herramienta de programación que permite coadyuvar en el proceso de la enseñanza de las matemáticas. [Guerrero de la Cruz. 2020]. Este es el entorno Spyder que usa código abierto Python, además de otras muchas funcionalidades de gran versatilidad, su descarga se encuentra en Spyder.org, que es un organismo con apoyo económico internacional y certificaciones. [Spyder. 2026]. Las ciencias básicas han sido en todas las ingenierías un reto para el aprendizaje y la enseñanza, recurriendo a herramientas de programación y calculadoras graficadoras como elementos clave para su explicación, comprensión y aplicación. [Radhakrishnan. 2023]. Para este caso, la herramienta que se propone obedece al desarrollo tecnológico del software para investigación y ciencia de datos. Como se observa en la siguiente figura 1, el IDE es muy intuitivo, sin embargo, el proceso de elaboración de un caso particular de programación es el siguiente: Una vez abierto el IDE, tomamos un nuevo archivo, ctrl+n, guardar archivo como, ejecutar archivo con F5.

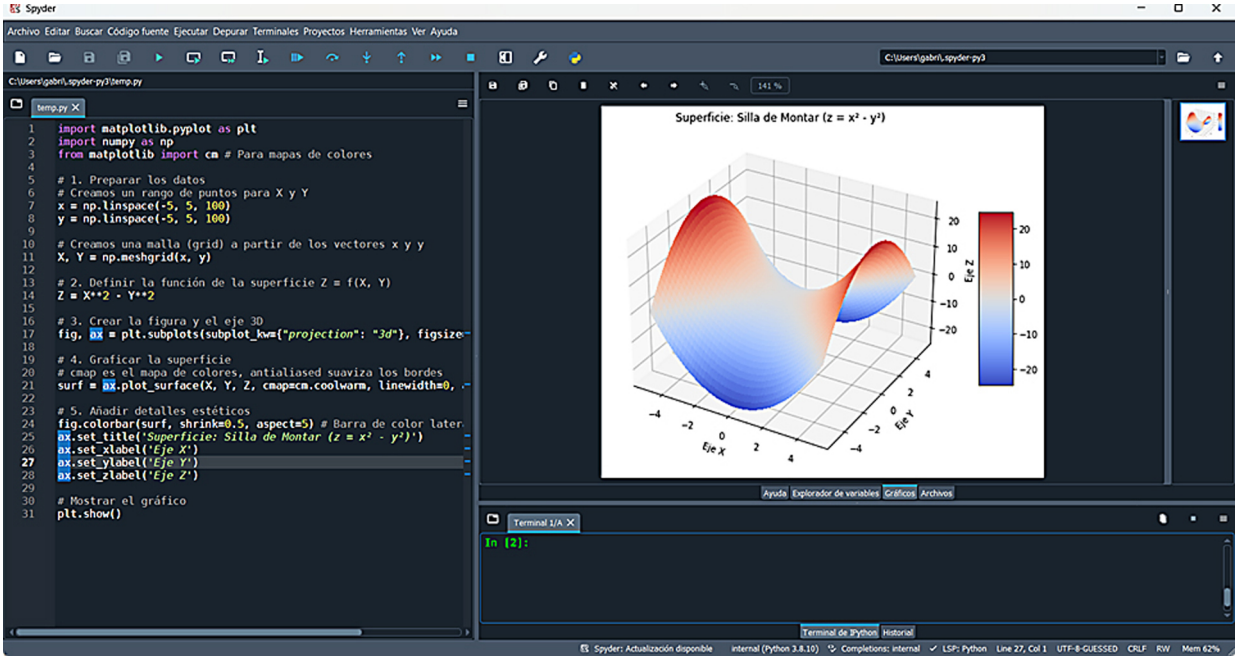

Figura 1. Abertura del IDE Spyder y creación de archivo.

Ya desde la metodología del proyecto integrador podemos observar que la vinculación directa de las materias de matemáticas con todas las ingenierías, por lo que una síntesis que abarca las estructuras de cálculo que requieren una base en la programación son principalmente: cálculo de curvas en el plano, curvas dependientes de un parámetro, curvas en coordenadas polares, curvas en el espacio tridimensional, y gráfica de una superficie que requieren la aplicación del cálculo integral y por consiguiente el conocimiento del diferencial. [Lara, 2019]. Su relación directa con las ecuaciones diferenciales se encuentra en las curvas integral, isoclina, ortogonal y de base. [Larson. 2016]. Las ciencias básicas de forma particular contienen el reto de programar sus ecuaciones, no solo para resolverlas a manera de calculadora, sino que se requiere graficar en 2D y 3D. Vemos que existen software como GeoGebra gratuito y otras versiones de paga como GPAI, que implican IA y por ende, costos, pero su aspecto de programación es poco didáctico y requiere bases teóricas en el funcionamiento de los algoritmos que se efectúan. Por tal motivo, este IDE Spyder es ideal para ilustrar el uso de programas cortos-pequeños en cantidad de líneas de código, explotados con las numerosas librerías del lenguaje Python y aplicadas a la solución de problemas matemáticos, [Moreno. 2020]. Sus principales instancias son numpy(), y matplotlib(), que son las librerías especializadas en matemáticas, cálculo y graficación lineal y 3D, además de la función graficadora plot(). [Moreno. 2020]. Para este caso de divulgación nos proponemos las siguientes preguntas que guiarán nuestra investigación exploratoria: a) ¿Qué librerías y versiones específicas del ecosistema científico de Python se utilizan? b) ¿Cómo se configura el entorno de desarrollo Spyder para la visualización interactiva? c) ¿Cuál es el criterio para definir la discretización del parámetro t (paso de tiempo)? d) ¿Qué método se emplea para la generación de la malla (Meshgrid) en las superficies? Como podemos observar las preguntas, nos permiten de forma específica diseñar programas apropiados al desarrollo de cálculos muy particulares, sobre todo para expresar ecuaciones en forma lineal, implícita y paramétrica. La lógica de congruencia del proyecto de este artículo que obedece tanto al desarrollo de ciencias básicas como a su proyecto integrador se presentan en la siguiente figura 2.

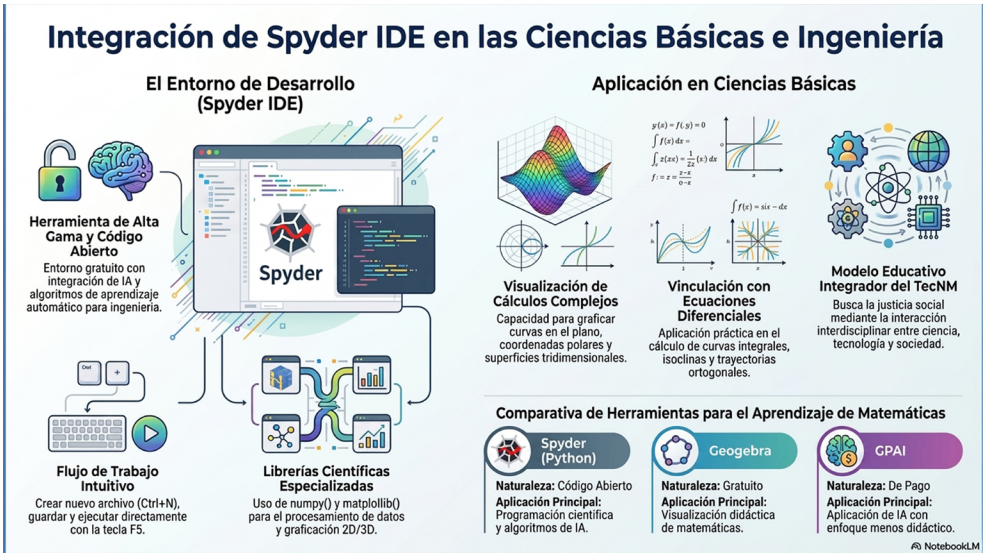

Figura 2. Lógica de congruencia del planteamiento del proyecto.

Metodología

Con las preguntas generadas en el apartado anterior, se propone usar esquemas y plantillas de programación con esa secuencia codificada, por tal motivo se exponen a continuación las ventanas principales a usar en el Spyder y los códigos analizados para esta caracterización de ejemplos, los cuales, siguen a una descripción de sus principales libertarias y con comentarios insertados dentro del código a programar. [Moreno. 2020]. Comenzamos con la apertura del IDE y su configuración principal, la cual permite agregar nuevo documento para insertar código y escribirlo, así como una compilación primaria de forma lineal, con gráficas en 2D, como se muestra en la siguiente figura 3.

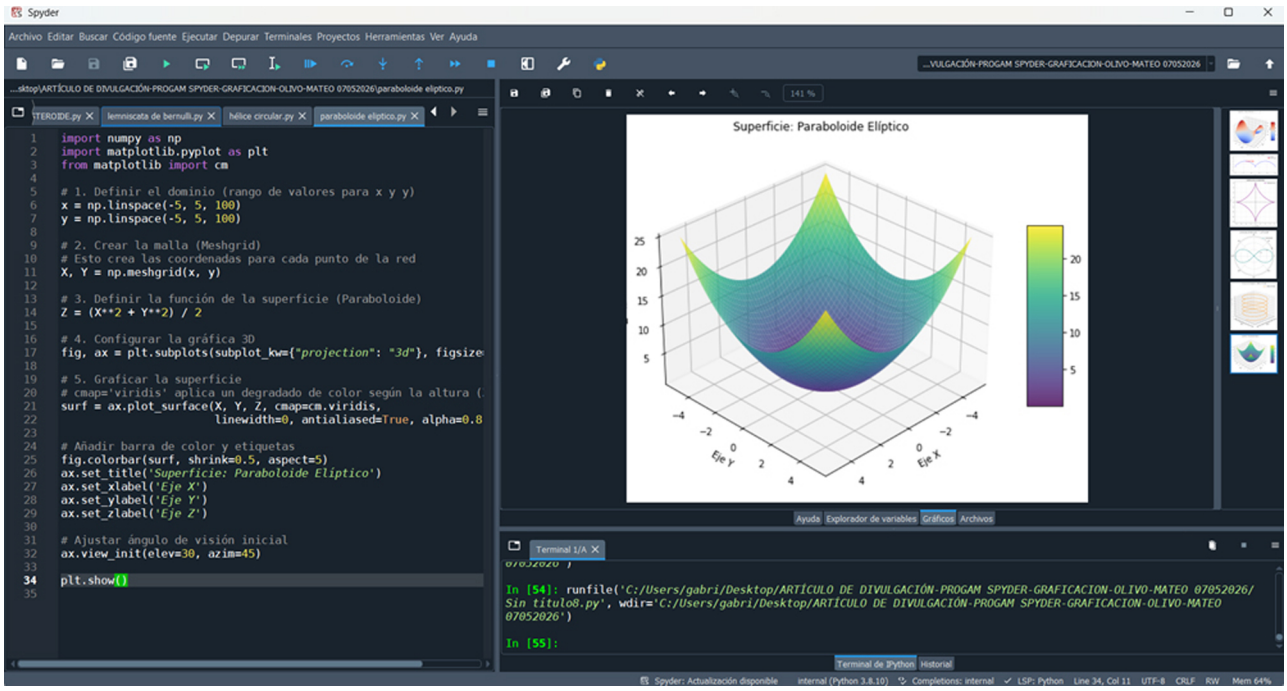

Figura 3. Configuración primaria y compilación principal.

Posteriormente se observan los ajustes para realizar las gráficas en 3D con las preferencias en herramientas ‘Qt5’ y ‘automatic’, para realizar gráficas de recorrido y comportamiento, como se muestra en la figura 4 a continuación.

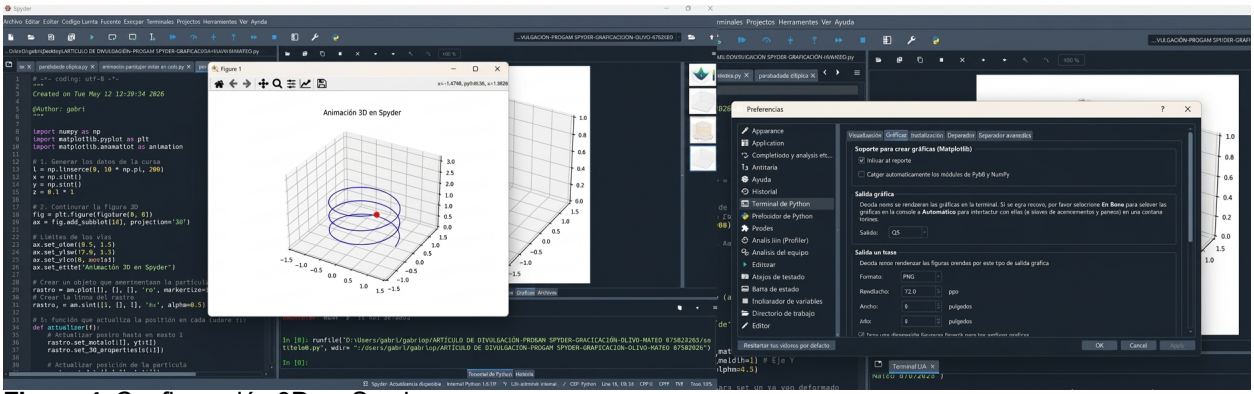

Figura 4. Configuración 3D en Spyder.

Lógica de proceso de programación: Con el objetivo de contestar las preguntas planteadas en la introducción, vemos que es factible ir desarrollando cada uno de los tipos de curvas planteadas con su respectivo código. Las partes esenciales de estos programas obedecen a la siguiente lógica crítica estructural: a) Definición del Dominio: Para curvas: Usar np.linspace para crear un solo vector de puntos (el parámetro t o theta). Para superficies: Usar np.meshgrid para crear una red de puntos sobre un plano. b) Modelado Matemático: Traducir la ecuación matemática a código Python usando las funciones de np (como np.sin, np.cos, np.sqrt). c) Configuración del Escenario (Ejes): Si es 2D: Un eje normal basta. Si es Polar: Especificar projection='polar'. Si es 3D: Especificar projection='3d'. d) Estética y Proporción: Indispensable: Usar plt.axis('equal') o ax.set_aspect('equal') para que las formas no se vean deformadas. Visualización: Usar mapas de colores (cmap) en superficies para entender la profundidad. Como se sintetiza en la siguiente tabla 1 a continuación:

Tabla 1. Guía de programación curvas y superficies.

Cicloide	Curva Paramétrica Plana	2D (x, y)	numpy, matplotlib	Parámetro de tiempo/ángulo t.	plt.plot(x, y)
Astroide	Curva Paramétrica (Hipocicloide)	2D (x, y)	numpy, matplotlib	Potencias trigonométricas (cos^3, sin^3).	plt.plot(x, y)
Lemniscata	Curva Polar	2D (r, theta)	numpy, matplotlib	Rango de theta para evitar raíces negativas.	fig.add_ subplot(projection='polar')
Hélice	Curva en el Espacio	3D (x, y, z)	numpy, matplotlib	Un parámetro t que afecta a los tres ejes.	ax.plot(x, y, z)
Paraboloide	Superficie Cuadrática	3D (x, y, z)	numpy, matplotlib, cm	Creación de malla con np.meshgrid(x, y).	ax.plot_surface(X, Y, Z)
Recorrido Partícula	Animación Dinámica	3D + Tiempo	numpy, matplotlib, animation	Función update(i) que refresca cada cuadro.	animation.FuncAnimation()

Programación Python en IDE Spyder

Desarrollamos los siguientes programas con base a la tabla guía y ejecutamos la depuración y compilación de errores, quedándonos con los ejecutables y sus gráficas tanto en 2D como en 3D, que será de gran utilidad para la gráfica de curva de recorrido. [Moreno. 2020]. Recorrido de partícula como curva dependiente de un parámetro en el espacio tridimensional. Sus principales librerías son: import numpy as np: Se encarga de generar la trayectoria matemática. Crea el parámetro 't' y las funciones trigonométricas para que la partícula sepa por dónde debe pasar. import matplotlib.pyplot as plt: Crea la infraestructura visual (la ventana, los ejes 3D y los objetos gráficos como el punto y la línea). import matplotlib.animation as animation: Esta es la librería clave y nueva. Proporciona las herramientas para actualizar la gráfica repetidamente, creando la ilusión de movimiento (como un proyector de cine que pasa fotogramas). Y para la animación tenemos: frames: Define cuántas "fotos" tiene la película. interval: Define la velocidad (en milisegundos) entre cada cuadro. Ver el código propuesto y la figura 5, a continuación:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
# 1. Generar los datos de la curva
t = np.linspace(0, 10 * np.pi, 500)
x = np.cos(t)
y = np.sin(t)
z = 0.1 * t
# 2. Configurar la figura 3D
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
# Límites de los ejes
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-1.5, 1.5)
ax.set_zlim(0, max(z))
ax.set_title("Animación 3D en Spyder")
# Crear el objeto que representará la partícula (un punto rojo)
punto, = ax.plot([], [], [], 'ro', markersize=10)
# Crear la línea del rastro
rastro, = ax.plot([], [], [], 'b-', alpha=0.5)
# 3. Función que actualiza la posición en cada cuadro (i)
def actualizar(i):
    # Actualizar rastro hasta el punto i
    rastro.set_data(x[:i], y[:i])
    rastro.set_3d_properties(z[:i])
    # Actualizar posición de la partícula
    punto.set_data([x[i]], [y[i]])
    punto.set_3d_properties([z[i]])
    return rastro, punto
# 4. Crear la animación
# interval=20 es la velocidad en milisegundos (ajusta a 50 o 100 para ir más lento)
ani = animation.FuncAnimation(fig, actualizar, frames=len(t),
                              interval=20, blit=False)
plt.show()
```

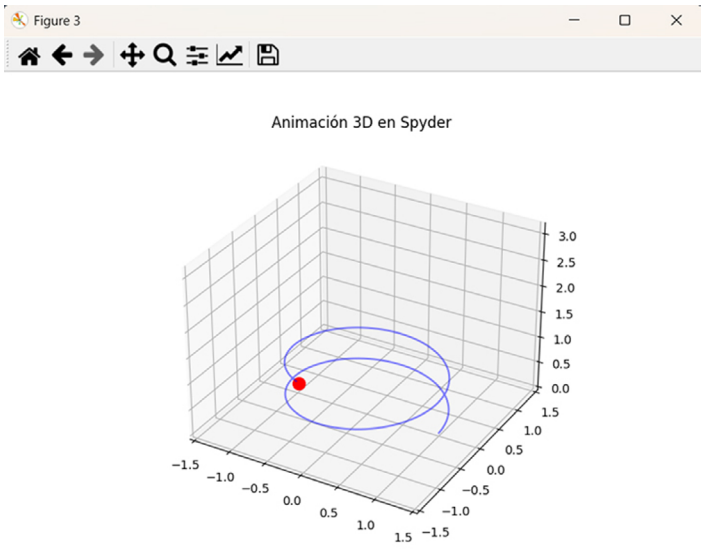

Figura 5. Recorrido de partícula.

Cicloide es ejemplo de curva dependiente de un parámetro: Para esta curva, tenemos las principales librerías, numpy (as np), que s el motor matemático. Se usa para crear los arreglos de datos (vectores), realizar operaciones matemáticas complejas (como senos y cosenos) sobre todo el conjunto de datos a la vez, y calcular derivadas o integrales numéricas. El matplotlib.pyplot (as plt), es la herramienta de visualización. Permite generar el gráfico, dibujar la curva, los vectores y personalizar la estética (títulos, colores, rejillas). Su visualización queda a cargo de plt.plot dibuja la línea azul de la trayectoria. Así plt. quiver es el comando clave aquí, dibuja una flecha (vector) que muestra la dirección del movimiento en un punto específico. plt.axis('equal') es vital para que el círculo no se vea como un óvalo y las proporciones sean reales. Vemos el ejemplo y su gráfico respectivo, figura 6:

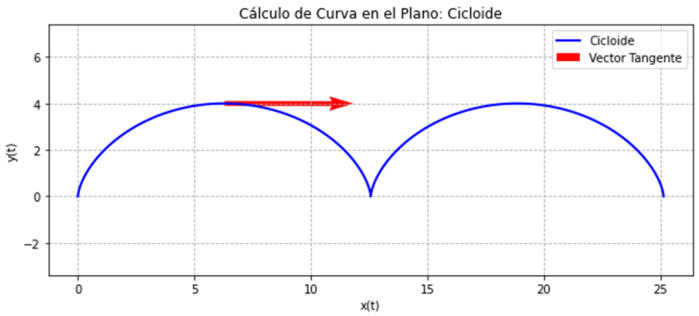

Figura 6. Cicloide, curva dependiente de un parámetro.

```
import numpy as np
import matplotlib.pyplot as plt
# 1. Definición de parámetros
r = 2  # Radio del círculo
t = np.linspace(0, 4 * np.pi, 1000)  # Parámetro t (dos vueltas)
# 2. Ecuaciones paramétricas de la curva
x = r * (t - np.sin(t))
y = r * (1 - np.cos(t))
# 3. Cálculo de la derivada (Vector Tangente) usando diferencias finitas
dx = np.gradient(x, t)
dy = np.gradient(y, t)
# 4. Cálculo de la Longitud de Arco aproximada
# L = integral de sqrt(dx^2 + dy^2) dt
ds = np.sqrt(dx**2 + dy**2)
longitud = np.trapz(ds, t)
print(f"Longitud de arco aproximada: {longitud:.4f}")
# 5. Visualización
plt.figure(figsize=(10, 4))
plt.plot(x, y, label='Cicloide', color='blue', lw=2)
# Dibujar un vector tangente en un punto específico (ej. t = pi)
idx = 250 # Índice cercano a t = pi
plt.quiver(x[idx], y[idx], dx[idx], dy[idx], color='red', scale=20, label='Vector Tangente')
plt.title('Cálculo de Curva en el Plano: Cicloide')
plt.xlabel('x(t)')
plt.ylabel('y(t)')
plt.axis('equal')
plt.grid(True, linestyle='--')
plt.legend()
plt.show()
```

Paraboloide elíptico como gráfica de una superficie: Sus principales librerías son import numpy as np, sigue siendo el motor de cálculo. Aquí es fundamental para crear la malla (meshgrid). A diferencia de la cicloide, donde solo necesitabas una lista de puntos, aquí necesitas una red de coordenadas (x, y) para cubrir un área. El import matplotlib.pyplot as plt, es la librería base para crear la ventana de la gráfica y los ejes. El from matplotlib import cm, (Color Map), es crucial para superficies, proporciona escalas de colores (como viridis, magma, etc.) que ayudan a visualizar la profundidad y la altura (Z) del objeto, haciendo que se vea "3D" y no como un bloque plano. Como se observa en la siguiente figura 7 y ejemplo de código propuesto:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
# 1. Definir el dominio (rango de valores para x y y)
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
# 2. Crear la malla (Meshgrid)
# Esto crea las coordenadas para cada punto de la red
X, Y = np.meshgrid(x, y)
# 3. Definir la función de la superficie (Paraboloide)
Z = (X**2 + Y**2) / 2
# 4. Configurar la gráfica 3D
fig, ax = plt.subplots(subplot_kw={"projection": "3d"},
figsize=(10, 7))
# 5. Graficar la superficie
# cmap='viridis' aplica un degradado de color según la
altura (Z)
surf = ax.plot_surface(X, Y, Z, cmap=cm.viridis,
                       linewidth=0, antialiased=True,
alpha=0.8)
# Añadir barra de color y etiquetas
fig.colorbar(surf, shrink=0.5, aspect=5)
ax.set_title('Superficie: Paraboloide Elíptico')
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
# Ajustar ángulo de visión inicial
ax.view_init(elev=30, azim=45)
plt.show()
```

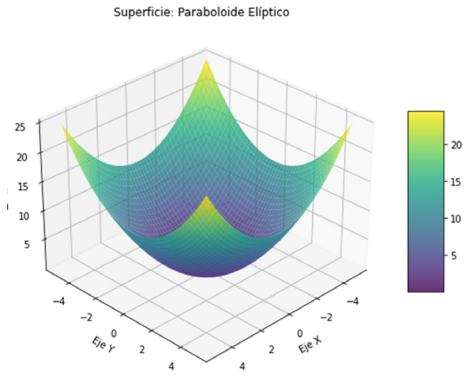

Figura 7. Gráfica de superficie, paraboloide elíptico.

Hélice circular, del tipo curva en el espacio o tridimensional. Sus principales librerías son import numpy as np, que se encarga de la generación de los datos. Aquí es vital para crear el parámetro ‘t’ (el tiempo o ángulo) y aplicar las funciones trigonométricas (sin x y cos x) que generan el círculo en la base. Luego import matplotlib.pyplot as plt, se utiliza para construir la infraestructura visual. Aunque es la misma librería que usaste para la cicloide, aquí se le pide que maneje un entorno de tres ejes. Su código propuesto es el siguiente y su gráfica es la figura 8, a continuación:

```
import numpy as np
import matplotlib.pyplot as plt
# 1. Definición de parámetros
R = 1       # Radio
c = 0.5     # Factor de ascenso
t = np.linspace(0, 10 * np.pi, 1000) # De 0 a 10pi
(varias vueltas)
# 2. Ecuaciones paramétricas en 3D
x = R * np.cos(t)
y = R * np.sin(t)
z = c * t
# 3. Creación de la figura 3D
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
# 4. Graficar la curva
ax.plot(x, y, z, label='Hélice Circular', color='dar-
korange', lw=2)
# Estética y etiquetas
ax.set_title('Curva en el Espacio: Hélice Circular')
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
ax.legend()
# Ajustar la vista para una mejor perspectiva
ax.view_init(elev=20, azim=45)
plt.show()
```

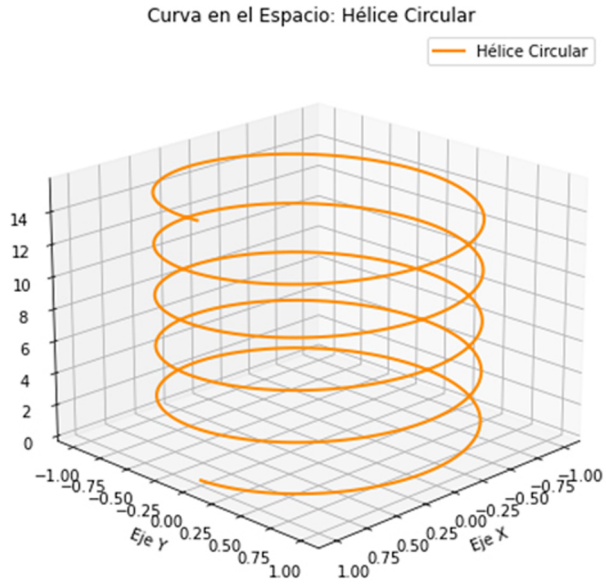

Figura 8. Curva en el espacio tridimensional.

Lemniscata de Bernoulli, como ejemplo de curva en coordenadas polares. Sus principales librerías son import numpy as np, que resulta esencial para manejar los cálculos trigonométricos. Se usa para generar el rango de ángulos y para calcular la raíz cuadrada (sqrt) y el coseno necesario para la ecuación. Luego import matplotlib.pyplot as plt, se encarga de la parte visual. Lo más importante aquí es que permite cambiar el sistema de coordenadas de cartesianas (rectangulares) a polares. Así projection='polar', cambia por completo el gráfico. En lugar de una cuadrícula de cuadritos, crea una telaraña circular donde los datos se posicionan por ángulo y radio. Luego la simetría, como la fórmula solo dibuja un lazo, el código añade theta + np.pi para "reflejar" el dibujo y completar el famoso símbolo de infinito (el segundo lazo). Veamos el código de ejemplo y la figura 9 ilustrativa a continuación:

```
import numpy as np
import matplotlib.pyplot as plt
# 1. Definición de parámetros
a = 2
# El rango de theta debe cuidarse donde cos(2*theta)
sea positivo para evitar raíces imaginarias
theta = np.linspace(-np.pi/4, np.pi/4, 500)
# 2. Ecuación de la Lemniscata (r^2 = 2a^2 * cos(-
2theta))
r = np.sqrt(2 * a**2 * np.cos(2 * theta))
# 3. Graficación
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(111, projection='polar')
# Graficamos ambos lóbulos (el positivo y el reflejo)
ax.plot(theta, r, color='teal', lw=2, label='Lemnis-
cata')
ax.plot(theta + np.pi, r, color='teal', lw=2) # Se-
gundo lazo
# Configuración visual
ax.set_title(r'Lemniscata de Bernoulli: $r^2 = 2a^2 \
cos(2\theta)$', va='bottom')
ax.set_rticks([1, 2, 2.8])  # Menos anillos radiales
ax.grid(True)
plt.legend()
plt.show()
```

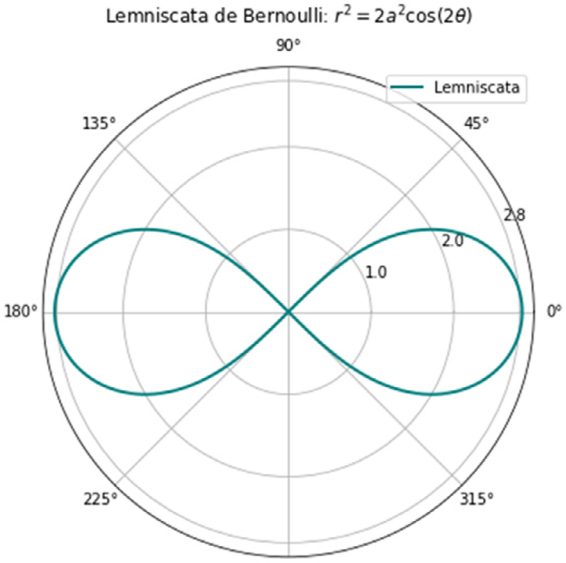

Figura 9. Curva en coordenadas polares.

Asteroide como ejemplo de curva plana o de superficie, muestra las siguientes librerías de base: numpy (as np), que es indispensable para manejar el parámetro ‘t’. Permite calcular las funciones trigonométricas elevadas al cubo (cos3 y sin3) de forma eficiente sobre todo un arreglo de puntos. Luego matplotlib.pyplot (as plt), se encarga de transformar los datos calculados en una imagen visual. Es la que dibuja la línea continua que forma la "estrella". Se utiliza plt.plot(x, y) y es muy importante añadir plt.axis('equal') para que la estrella no se vea deformada o aplastada. La gráfica obedece a la familia de las hipocicloides. Su código y gráfica se muestran en la siguiente figura 10.

```
import numpy as np
import matplotlib.pyplot as plt
# 1. Configuración del parámetro a (radio de la circunferencia mayor)
a = 4
# 2. Definición del parámetro t de 0 a 2pi
# Usamos 1000 puntos para que la curva se vea suave
t = np.linspace(0, 2 * np.pi, 1000)
# 3. Ecuaciones paramétricas del Asteroide
x = a * np.cos(t)**3
y = a * np.sin(t)**3
# 4. Creación de la gráfica
plt.figure(figsize=(7, 7))
plt.plot(x, y, label=f'Asteroide (a={a})', color='purple', lw=2.5)
# Estética de la gráfica
plt.title('Gráfica de un Asteroide', fontsize=14)
plt.xlabel('x = a·cos³(t)')
plt.ylabel('y = a·sin³(t)')
plt.axhline(0, color='black',linewidth=1) # Eje X
plt.axvline(0, color='black',linewidth=1) # Eje Y
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.axis('equal') # Importante para que no se vea deformado
# Mostrar resultado
plt.show()
```

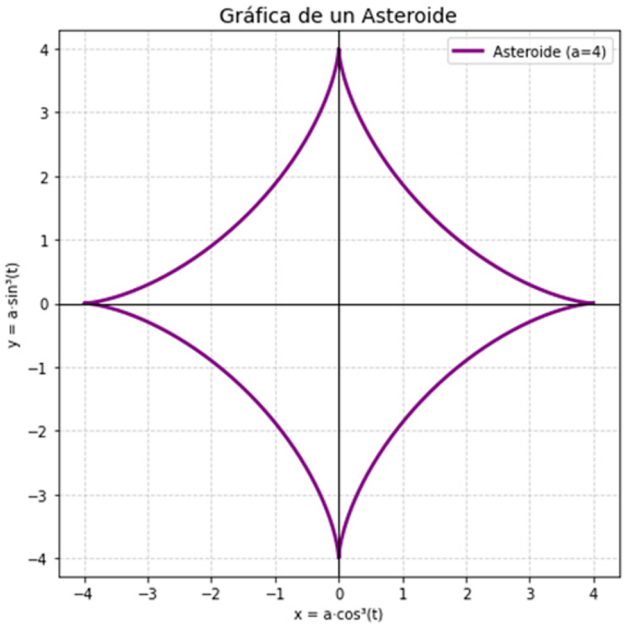

Figura 10. Curva plana,

La silla de montar como gráfica de una superficie. Sus principales librerías son import numpy as np, es la base para el manejo de las matrices. Aquí es vital para generar la malla (meshgrid) y realizar la resta de cuadrados (X2 - Y2) elemento por elemento en toda la cuadrícula. Luego import matplotlib.pyplot as plt es la librería principal para generar la ventana y los ejes. Proporciona el método plot_surface que conecta los puntos de la malla para crear la "tela" de la superficie. Así from matplotlib import cm, (Color Map), proporciona el mapa de colores coolwarm. En este tipo de gráficas es fundamental porque ayuda a distinguir visualmente las zonas altas (colores cálidos/rojos) de las zonas bajas o "caídas" (colores fríos/azules). Definición del Dominio y Malla (meshgrid), se crean los vectores x y y, y luego np.mesh-

grid los expande en matrices. Esto le dice a Python: "estos son todos los puntos sobre el suelo donde quiero calcular la altura". Función de Superficie (Z): Aquí se define la geometría. Al usar una resta (X2 - Y2), creas la curvatura doble, el eje X tiende a subir (parábola hacia arriba) y el eje Y tiende a bajar (parábola hacia abajo). Instanciación del Eje 3D: Mediante projection="3d", se prepara a Matplotlib para recibir tres coordenadas y permitir la rotación del gráfico. Mapeo de Color y Renderizado, cmap=cm. coolwarm, aplica el degradado de color. Luego antialiased=False, se usa para mejorar el rendimiento o dar un aspecto. El código y su gráfica respectiva, se ilustra en la figura 11.

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm # Para mapas de colores
# 1. Preparar los datos
# Creamos un rango de puntos para X y Y
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
# Creamos una malla (grid) a partir de los vectores x y y
X, Y = np.meshgrid(x, y)
# 2. Definir la función de la superficie Z = f(X, Y)
Z = X**2 - Y**2
# 3. Crear la figura y el eje 3D
fig, ax = plt.subplots(subplot_kw={"projection": "3d"}, figsize=(10, 7))
# 4. Graficar la superficie
# cmap es el mapa de colores, antialiased suaviza los bordes
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm, linewidth=0, antialiased=False)
# 5. Añadir detalles estéticos
fig.colorbar(surf, shrink=0.5, aspect=5) # Barra de color lateral
ax.set_title('Superficie: Silla de Montar (z = x² - y²)')
ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')
# Mostrar el gráfico
plt.show()
```

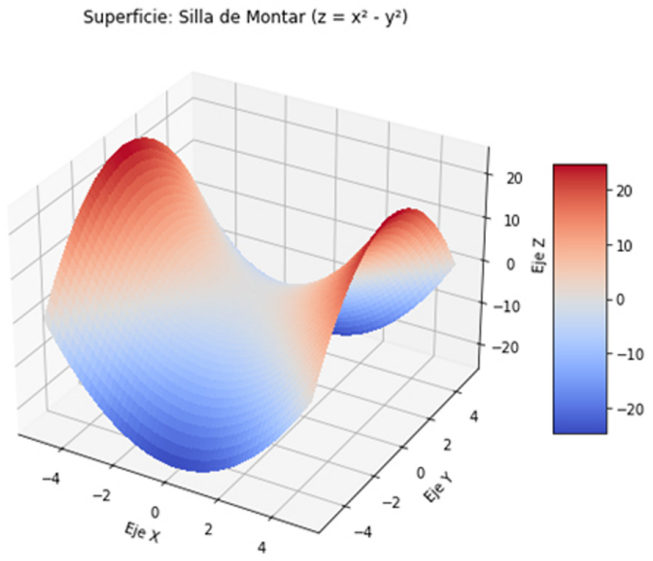

Figura 11. Silla de montar.

Integración de AI en el Spyder: Para integrar AI en el Spyder requerimos seguir la siguiente lógica, a) la generación de código, que consiste en la creación rápida de ecuaciones paramétricas complejas; b) la corrección de sintaxis, que evita errores comunes al usar meshgrid o dimensiones de matrices; c) la documentación, que genera automáticamente los comentarios y docstrings para tus funciones matemáticas; y d) la optimización, que sugiere el uso de funciones específicas de SciPy o NumPy que son más rápidas. El código seleccionado y copiado al Spyder pasa a ser revisado por el explorador de variables, usando el Kite de autocompletado inteligente, que es el motor de IA de Spyder, [Spyder. 2026]. En el siguiente mapa

conceptual se agrupa la metodología para desarrollar un programa en el IDE, tanto para curvas de nivel como estructuras matemáticas complejas, ver la figura 12.

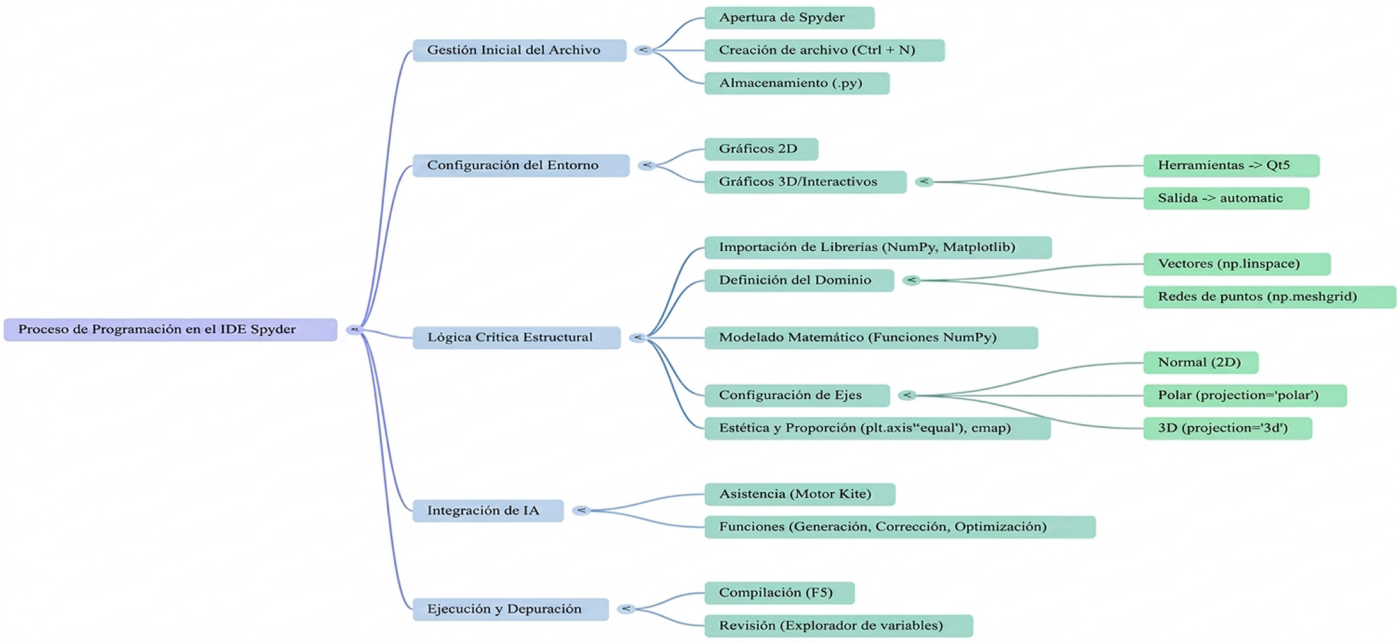

Figura 12. Mapa de metodología para la programación de estructuras matemáticas y curvas de nivel.

Resultados

Como se ha mostrado en los apartados anteriores, se ha reflejado la manera de ir desarrollando cada una de las preguntas planteadas bajo el uso del mismo IDE; vemos que los resultados se derivan directamente de la metodología aplicada a cada pregunta que en esencia consta de: manejo de librerías bajo el ecosistema científico para problemas de cálculo y graficación, como numpy(), con motor matemático para el manejo de variables, arreglos y funciones complejas, como matplotlib(), que incluye infraestructura visual, animation(), que no solo genera movimiento, sino clase de color, cm(), los colores usados fueron viridis y coolwarm. [Guerrero. 2020]. La configuración del Spyder no requiere de mayor esfuerzo dentro del entorno, mientras que las gráficas 2D tienen una compilación lineal primaria, el comportamiento interactivo 3D, que permite rotación y acercamiento, es posible con las salidas 'Qt5' y 'automatic', dentro de las herramientas de preferencias. [Radhakrishnan,2023]. De igual manera la discretización de parámetro (t), se logra bajo el uso de la función np.lispace(), que permite precisar curvas suaves y generar vectores con una alta densidad de puntos, utilizando comúnmente rangos de 500 a 1000 puntos (por ejemplo, de 0 a 2π o 4π) para evitar que las figuras se vean deformadas o pixeladas, [Radhakrishnan,2023]. Para la generación de mallas, (Meshgrid), hemos observado que es posible expandir los vectores en los ejes X, y Y en matrices, mediante np.meshgrid(), lo que permite crear una red de coordenadas sobre el suelo del gráfico para calcular la altura (z) en cada pun-

to, transformando ecuaciones matemáticas en una 'tela' tridimensional. [McKinney. 2022].

Conclusiones

En cuanto a las preguntas planteadas, podemos discutir las siguientes respuestas: las librerías de Python funcionan desde una óptica de programación orientada a objetos y a procesos, [Radhakrishnan. 2023]. La factibilidad técnica muestra el uso de pocas líneas de código que resuelven problemas complejos de cálculo vectorial y diferencial, haciendo del lenguaje Python una estructura y una herramienta versátil y de alta gama, pertinente y actual para el aprendizaje y la comprensión de las ciencias básicas, [McKinney. 2022]. Se observa que los motores del manejo de variables del Spyder autocompleta de manera inteligente y optimizan la sintaxis, agregando una generación de comentarios técnicos al margen del código, lo que da legibilidad a su lectura y comprensión del programa. El impacto educativo viene garantizado debido a que docentes y alumnos pueden trabajar de forma autónoma, congruente e incluso coherentemente, debido a que las herramientas Tic´s que se señalan son de uso gratuito. Observamos que se logra el aprendizaje significativo vinculado a los fenómenos reales de ingeniería. [Guerrero. 2020].

Resalta la efectividad de integrar herramientas tecnológicas avanzadas en la enseñanza de las matemáticas para ingeniería, en este caso particular Spyder. Sin embargo, diferentes Tic´s se incorporan constantemente al proceso de enseñanza aprendizaje, resultado de gran apoyo para la comprensión de las ciencias básicas. [Guerrero. 2020]. La vinculación con el modelo educativo es fundamental para cumplir con las metas del aprendizaje y aplicación de las ciencias básicas. Las cuales, quedan fortalecidas, no solo para resolver ecuaciones de forma aritmética, sino para lograr su visualización gráfica. Se pasa así de la aplicación de datos de calculadora, a un medio de comprensión para los fenómenos de ingeniería. La eficacia de Spyder se muestra como un entorno intuitivo ideal para trabajos empíricos, de investigación y aplicación de ciencia de datos, completamente apoyados en la esencia del código abierto, gratuito y con una creciente capacidad de incorporar librerías especializadas que facilitan la resolución de problemas de cálculo vectorial, diferencial y álgebra lineal con códigos cortos y eficientes. De igual manera la optimización mediante IA, (a través del motor Kite en Spyder), es un catalizador crítico que optimiza la generación de ecuaciones complejas, corrige la sintaxis en matrices y mejora la documentación de funciones matemáticas, acelerando el proceso de programación, [Johansson. 2019]. Se valida así la aplicabilidad práctica, ya que es viable diseñar programas específicos para representar curvas y superficies (como la cicloide, la hélice o la silla de montar), logrando una representación geométrica precisa que facilita el estudio de trayectorias e isoclinas en ingeniería, [Guerrero. 2020].

Citas y referencias

Guerrero de la Cruz, R., Guerrero de la Cruz, R. J., & Guerrero de la Cruz, A. (2020). *Programación con Python*. Universidad Autónoma de la Ciudad de México. https://repositorioinstitucionaluacm.mx/jspui/bitstream/123456789/3042/1/ProgramaconPythonWEB.pdf

Johansson, R. (2019). Computación científica numérica con Python: Usando NumPy, SciPy y Matplotlib (2.ª ed.). *Apress*.

Lara, B. L., Tolentino, F. O., López, V. C., Muñoz, J. E., (2019). Metodología para el desarrollo del proyecto integrador en programas del área de ingeniería mecánica eléctrica. A*NFEI Digital* (11), pp. 1-10. https://www.anfei.mx/revista/index.php/revista/article/view/586.

Larson, R., y Edwards, B. H. (2016). Cálculo de varias variables (10.ª ed., Vol. 2). Cengage Learning.

López Rodríguez, N. M., García Fraile, J. A., El proyecto integrador: Estrategia didáctica para la formación de competencias desde la perspectiva del enfoque socioformativo (Primera edición), *Gafra editores*, 2012.

McKinney, W. (2022). Python para análisis de datos: Manejo de datos con Pandas, NumPy y Jupyter (3.ª ed.). *O'Reilly Media*.

Moreno Muñoz, A., y Arroba Aguado, B. (2020). Python 3: Curso práctico. *Editorial Ra-Ma*.

Radhakrishnan, P., & Vairaprakash, G. (2023). Python with Spyder: An experiential learning perspective. *CRC Press*.

Spyder IDE. (2026.). Home. https://www.spyder-ide.org/

Tecnológico Nacional de México. (2015). Proyectos integradores para la formación y desarrollo de competencias profesionales (2.ª ed.). http://www.itsjuanrodriguezclara.edu.mx/Document/Transparencia/875/FRACCION_I/Proyectos_Integradores_2da_edicion.pdf

Tecnológico Nacional de México, (2024). Modelo Educativo del TecNM: Humanismo para la Justicia Social, México. Repositorio en: https://www.tecnm.mx/archivos/ slider/Modelo_Educativo_del_TecNM_digital_orig.pdf.

Tejeda, M. D., Polo, P. L., (2019). Los proyectos integradores: una perspectiva pedagógica de desarrollo del ingeniero. *ANFEI Digital* (11), pp. 1-11. https://www.anfei.mx/revista/index.php/revista/article/view/547.