# main

August 22, 2024

## 1 Taller semana 04, Matemáticas aplicadas 1

[51]:
```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import clear_output, display
import numpy.random as random
```

### 1.1 1

Serie Fibonacci: Escribe un programa en Python que genere los primeros n números de la serie de Fibonacci y visualízalos en un gráfico de líneas utilizando Matplotlib. El usuario debe poder elegir el número n

$F(n) = \frac{1}{\sqrt{5}} \left( \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right)$

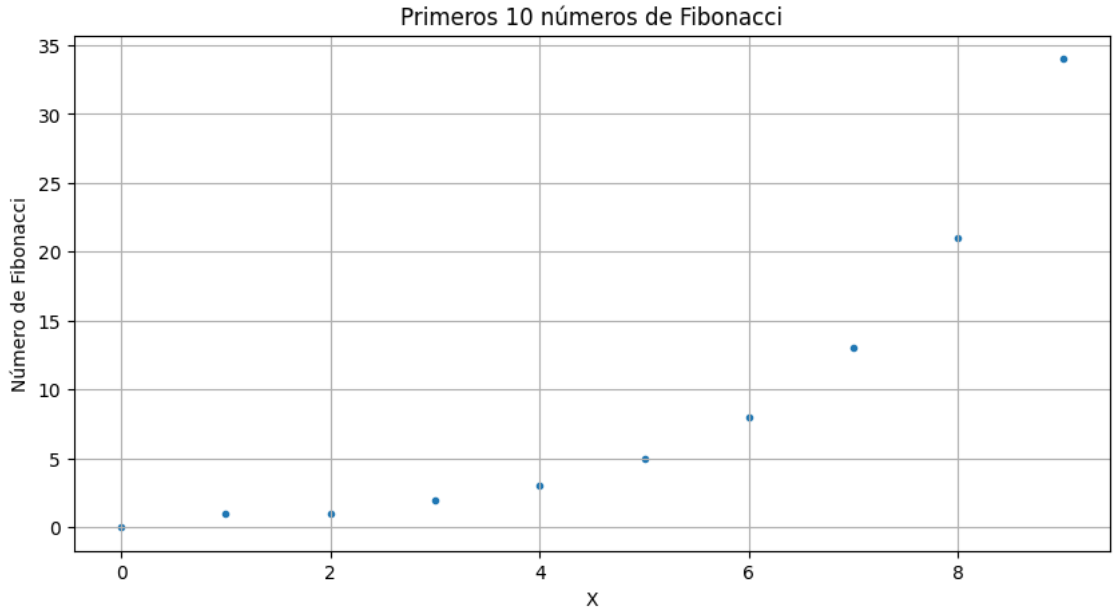
[52]:
```
def fibonacci(n):
        if n == 0:
                return 0
        elif n == 1:
                return 1
        else:
                a = 0
                b = 1
                for i in range(2, n+1):
                        c = a + b
                        a = b
                        b = c
                return c
```

[53]:
```
def fibonacci_math(n):
    return int(
                (1/(5**(1/2)))*(((1+(5**(1/2)))/2)**n - ((1-(5**(1/2)))/2)**n)
        )
```

[54]:
```
def get_n_fibonacci (n):
    return [fibonacci_math(i) for i in range(n)]
```

[55]:
```
# graph_fibonacci
def graph_fibonacci(n):
    entries = get_n_fibonacci(n)
    x = np.arange(n)
    plt.figure(figsize=(10, 5))
    plt.scatter(x, entries, marker='.')
    plt.title(f'Primeros {n} números de Fibonacci')
    plt.xlabel('X')
    plt.ylabel('Número de Fibonacci')
    plt.grid(True)
    plt.show()
```

[56]:
```
n = int(input('Ingrese el número de elementos de la serie de Fibonacci: '))
graph_fibonacci(n)
```



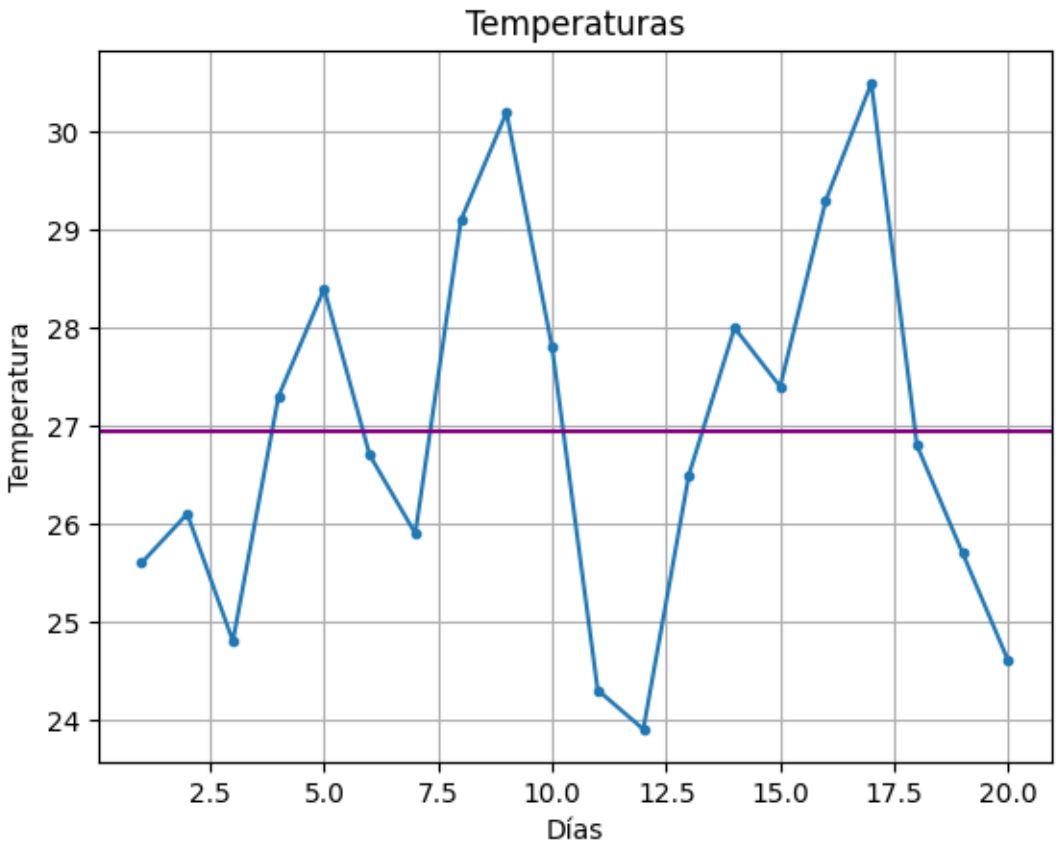

## 1.2 2 Análisis de datos meteorológicos

Dado un conjunto de datos con las temperaturas diarias de una ciudad durante un año, escribe un programa que calcule la media, la mediana y la desviación estándar de las temperaturas. Luego, crea un gráfico de líneas que muestre la temperatura media mensual

[57]:
```
temperaturas = [25.6, 26.1, 24.8, 27.3, 28.4, 26.7, 25.9, 29.1, 30.2, 27.8, 24.
↪3, 23.9,
                26.5, 28.0, 27.4, 29.3, 30.5, 26.8, 25.7, 24.6]


def media(temperaturas):
```

```
    suma = 0
    for x in temperaturas:
        suma += x
    return (suma/len(temperaturas))


promedio = media(temperaturas)
print(f'La desviación estándar es de: {promedio}')


def mediana(temperaturas):
    return np.median(temperaturas)


mediana = mediana(temperaturas)
print(f'La desviación estándar es de: {mediana}')


def desviacion_estandar(temperaturas):
    varianza = np.var(temperaturas)
    return np.sqrt(varianza)


desviacion = desviacion_estandar(temperaturas)
print(f'La desviación estándar es de: {desviacion}')

x = range(1, len(temperaturas) + 1)
plt.plot(x, temperaturas, marker = ".")
plt.axhline(promedio, color="purple")
plt.title('Temperaturas')
plt.xlabel('Días')
plt.ylabel('Temperatura')
plt.grid(True)

plt.show()
```

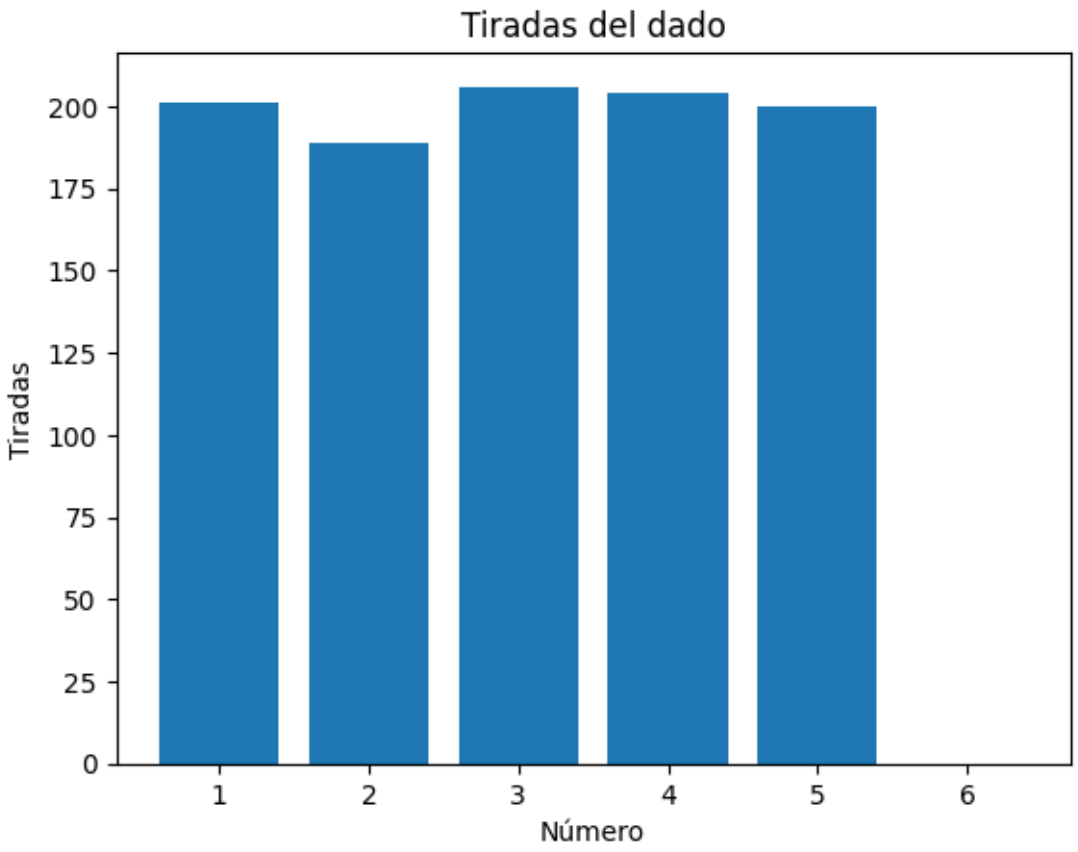
```
La desviación estándar es de: 26.945
La desviación estándar es de: 26.75
La desviación estándar es de: 1.864530772071086
```




### 1.3 3 Simulación de un dado

Simula 1000 lanzamientos de un dado tradicional y muestra la frecuencia de cada número en un gráfico de barras

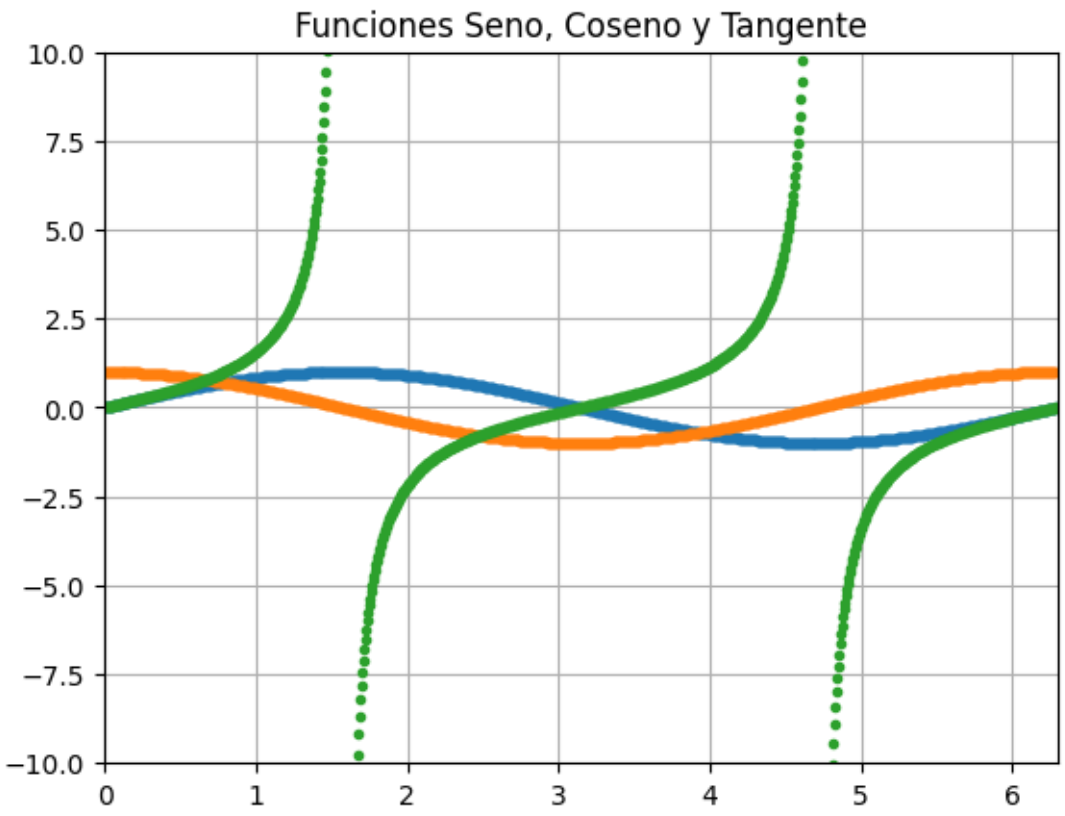
[61]:
```
def lanzar_dado():
    resultados = []
    for x in range(0, 1000):
        resultados.append(random.randint(1, 6))
    return resultados


resultados = lanzar_dado()
contadas = [resultados.count(i) for i in range(1, 7)]

plt.bar(range(1, 7), contadas)
plt.xlabel('Número')
plt.ylabel('Tiradas')
plt.title('Tiradas del dado')
plt.show()
```




### 1.4 4 Gráfica de funciones trigonométricas

Escribe un programa que grafique las funciones seno, coseno y tangente en el mismo plano para valores entre 0 y 2 .

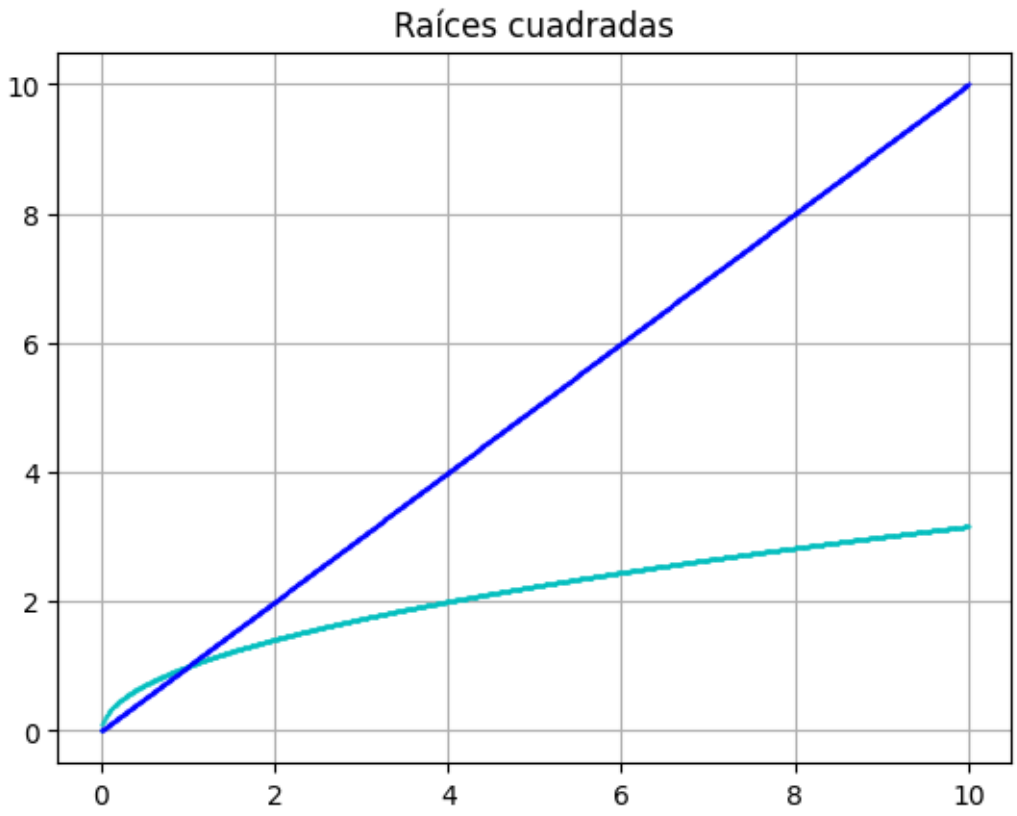
[63]:
```
x = np.linspace(0,(2*np.pi),1000)
s = [np.sin(x) for x in x]
c = [np.cos(x) for x in x]
t = [np.tan(x) for x in x]

plt.plot(x,s, '.')
plt.plot(x,c, '.')
plt.plot(x,t, '.')
plt.xlim(0,6.3)
plt.ylim(-10,10)
plt.grid('g')
plt.title("Funciones Seno, Coseno y Tangente")
plt.show()
```




### 1.5 5 Raíces cuadradas

Grafique la función $y = \sqrt{x}$ junto con la línea $y = x$ en el mismo gráfico para valores de x entre 0 y 10

[64]:
```
x = np.linspace(0,10,1000)
y = [np.sqrt(x) for x in x]


plt.plot(x,y, '.c', ms=1)
plt.plot(x,x, '.b', ms = 1)
plt.grid('g')
plt.title("Raíces cuadradas")
plt.show()
```



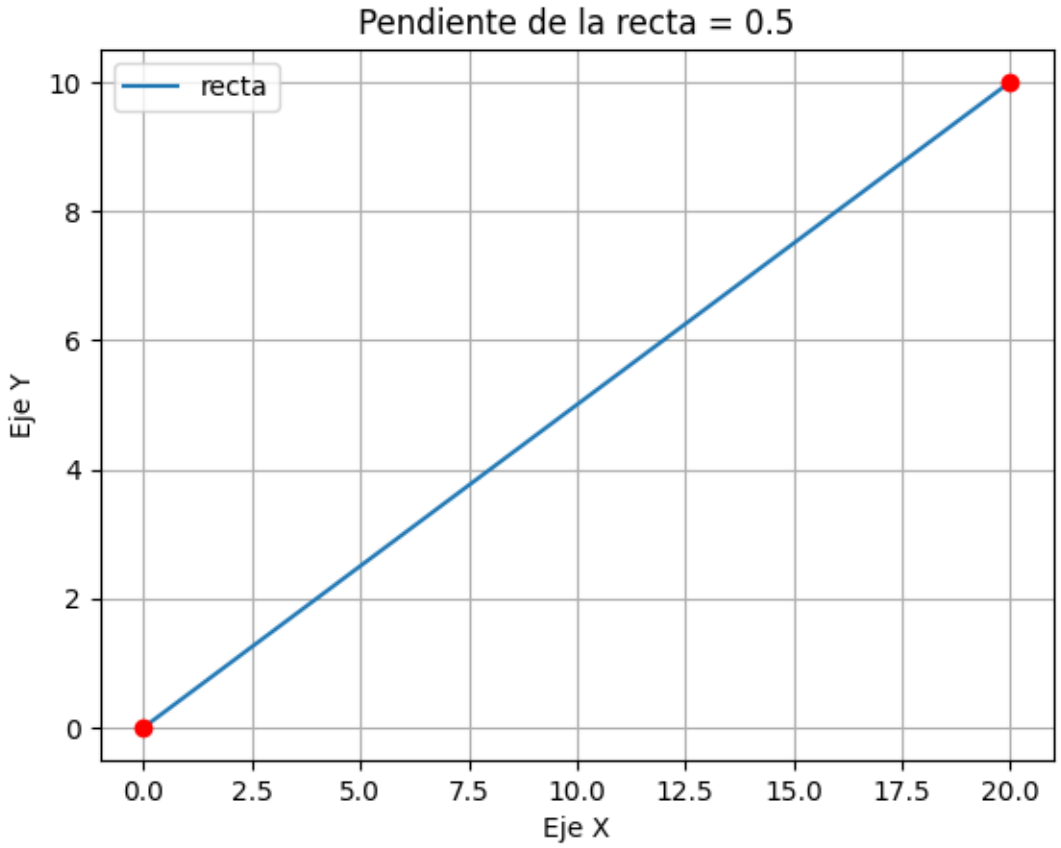

### 1.6 6 Pendiente de una recta

Escribe un programa que tome dos puntos en el plano (x1, y1) y (x2, y2), calcule la pendiente de la recta que los une, y grafique dicha recta

[69]:
```
x1= float ( input("Dame el valor de x1: "))
x2= float (input("Dame el valor de x2: "))
y1= float ( input("Dame el valor de y1: "))
y2= float (input("Dame el valor de y2: "))

m= (y2-y1)/(x2-x1)
print ("La pendiente de la recta es: ", m ,"\n")
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.title(f"Pendiente de la recta = {m}")
plt.grid()

plt.plot([x1,x2],[y1,y2], label='recta', )
plt.legend()
plt.plot(x1,y1,'ro')
plt.plot(x2,y2,'ro')
```

```
plt.show()
```

```
La pendiente de la recta es:  0.5
```

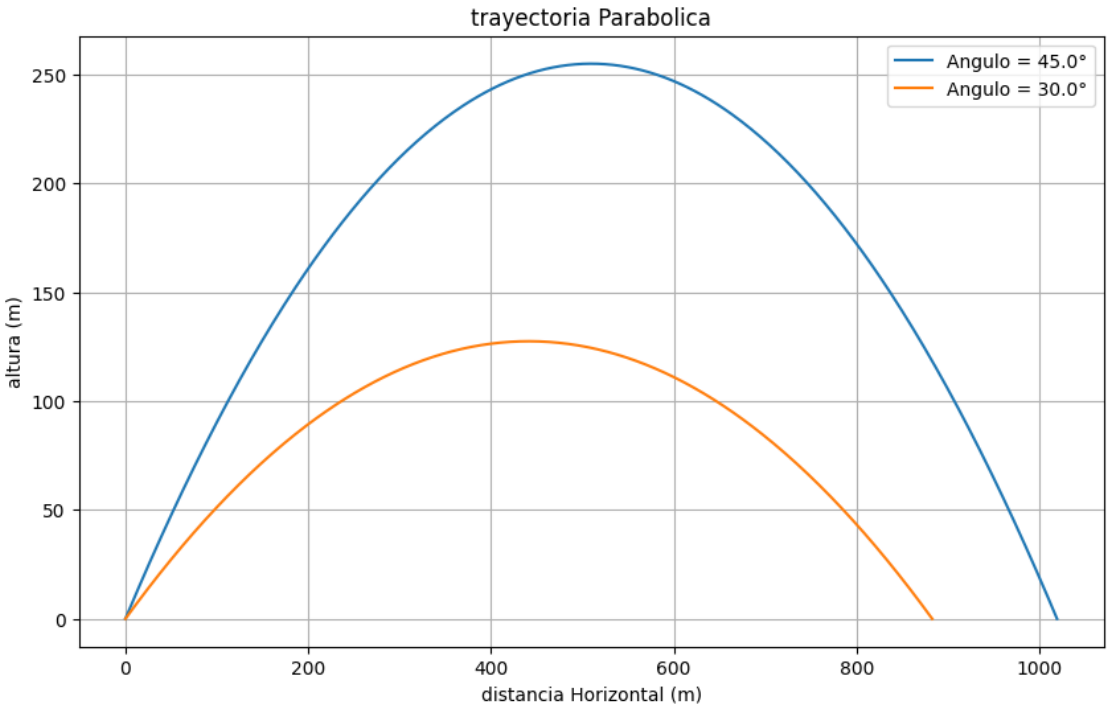



## 1.7 7 Movimiento parabólico

Escribe un programa que simule el movimiento de un proyectil bajo la influencia de la gravedad (sin resistencia del aire). Grafica la trayectoria parabólica para diferentes ángulos de lanzamiento

[70]:
```
import matplotlib.pyplot as plt

def obtener_datos_usuario():
    v0 = float(input("introduce la velocidad inicial del proyectil (m/s): "))
    g = 9.81
    num_angulos = int(input("cuantos angulos de lanzamiento quieres ingresar?␣
 ↪"))

    angulos = []
    for i in range(num_angulos):
```

```
        angulo = float(input(f"introduce el angulo de lanzamiento {i+1} (en␣
 ↪grados): "))
        angulos.append(angulo)

    return v0, g, angulos

def graficar_trayectorias(v0, g, angulos):
    plt.figure(figsize=(10, 6))

    for angulo in angulos:
          = np.radians(angulo)
        t_max = (2 * v0 * np.sin( )) / g
        t = np.linspace(0, t_max, num=500)

        x = v0 * np.cos( ) * t
        y = v0 * np.sin( ) * t - 0.5 * g * t**2

        plt.plot(x, y, label=f'Angulo = {angulo}°')

    plt.xlabel('distancia Horizontal (m)')
    plt.ylabel('altura (m)')
    plt.title('trayectoria Parabolica')
    plt.grid(True)
    plt.legend()
    plt.show()

v0, g, angulos = obtener_datos_usuario()
graficar_trayectorias(v0, g, angulos)
```




### 1.8 8

Construya un formulario con deslizador que modifique la potencia de la función $f(x) = x^n$ con $n \in N$ [1, 10]

[59]:
```
def f(x, n):
    return x**n

x = np.linspace(0, 10, 100)

power_slider = widgets.IntSlider(
    value=1,
    min=1,
    max=10,
    step=1,
    description='Potencia',
    continuous_update=True
)

output = widgets.Output()

def update_graph(change):
    n = power_slider.value
    with output:
```

```
        clear_output(wait=True)
        plt.plot(x, f(x, n))
        plt.title(f'Graph of x^{n}')
        plt.xlabel('x')
        plt.ylabel('f(x)')
        plt.grid(True)
        plt.show()

power_slider.observe(update_graph, 'value')

display(power_slider, output)

update_graph(None)
```

```
IntSlider(value=1, description='Potencia', max=10, min=1)
```

```
Output()
```

### 1.9 9 (Opcional)

Punto Extra: Fractales con Python: Escribe un programa que genere el fractal de Sierpinski utilizando ciclos (while), y visualiza el resultado con Matplotlib

$M = (\frac{x_1+x_2}{2} + \frac{y_1+y_2}{2})$

[60]:
```
def plot_triangles(triangles):
    plt.figure()
    for triangle in triangles:
        vertex1, vertex2, vertex3 = triangle
        x1, y1 = vertex1
        x2, y2 = vertex2
        x3, y3 = vertex3

        x_coords = [x1, x2, x3, x1]
        y_coords = [y1, y2, y3, y1]

        plt.plot(x_coords, y_coords, marker='.')

    plt.title('Fractal de Sierpinski')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.grid(True)
    plt.show()


def get_middle_point(point_1, point_2):
    return ((point_1[0] + point_2[0]) / 2, (point_1[1] + point_2[1]) / 2)

get_middle_point((0, 0), (2, 4))
```

```python
def get_inner_triangle_points(points):
    new_points = []
    new_points.append(get_middle_point(points[0], points[1]))
    new_points.append(get_middle_point(points[1], points[2]))
    new_points.append(get_middle_point(points[0], points[2]))
    return tuple(new_points)

initial_triangle = ((0, 0), (2, 4), (4, 0))

triangles = [initial_triangle]

while len(triangles) < 6:
    # Obtner siempre el último triángulo
    last_triangle = triangles[-1]
    inner_triangle = get_inner_triangle_points(last_triangle)
    triangles.append(inner_triangle)

plot_triangles(triangles)
```