MÉTODOS NUMÉRICOS

Proyecto Primer Bimestre

Docente: Jonathan A. Zea

Integrantes: Lenin Amangandi
Camila Caicedo
Richard Tipantiza

25 noviembre de 2025

ÍNDICE DE CONTENIDOS

1. OBJETIVOS

- Desarrollar un programa que determine los parámetros de lanzamiento de un segundo proyectil para que colisione con uno previamente lanzado, considerando efectos ambientales y utilizando métodos numéricos.
- Modelar matemáticamente el movimiento parabólico de ambos proyectiles e Incorporar efectos de viento mediante ruido blanco en la simulación.
- Implementar y comparar dos métodos numéricos para resolver el sistema de ecuaciones para analizar su complejidad computacional y eficiencia.

2. INTRODUCCIÓN

2.1. Resumen

Este proyecto desarrolla un sistema para calcular y simular la intercepción aérea de dos proyectiles, P1 y P2, donde el segundo es lanzado con un retraso temporal T desde el origen. La meta central es determinar las condiciones de lanzamiento óptimas para P2 (velocidad u y ángulo) que no solo garanticen la colisión, sino que también minimicen la energía requerida, es decir, minimizar u. Además, el sistema simula condiciones reales al incorporar los efectos del viento, modelado mediante la adición de ruido blanco gaussiano a las componentes de velocidad de ambos proyectiles. Esto convierte las trayectorias parabólicas en un modelo estocástico más realista. Los resultados se visualizan mediante una animación detallada de las trayectorias y un análisis de la complejidad computacional.

2.2. Antecedentes

El problema de la colisión de proyectiles tiene raíces en los principios de la mecánica clásica expuestos por Galileo y Newton, quienes describieron el movimiento parabólico bajo una gravedad constante. En el ámbito computacional, este problema se presenta como un caso clásico de sistemas de ecuaciones no lineales, que históricamente se ha resuelto con métodos numéricos como el método de la Bisección y la Secante. Sus aplicaciones se han documentado en interceptación balística, simulaciones educativas y el análisis de trayectorias en deportes.

3. MARCO TEÓRICO

3.1. Fundamentos conceptuales

3.1.1. Método de la Bisección

El método de la Bisección (también conocido como método de intervalo medio) es, quizás, el algoritmo de búsqueda de raíces más simple y robusto. Su principio se basa en el Teorema del Valor Intermedio, que asegura que si una función continua f(x) cambia de signo en un intervalo [a, b], debe existir al menos una raíz dentro de ese intervalo. El método divide el intervalo por la mitad en cada iteración y funciona siempre y cuando se encuentre en un intervalo inicial [a, b], donde los dignos de f(a) y f(b) sean opuestos. Cabe mencionar que posee una convergencia lineal y lenta.

3.1.2. Método de la Secante

El Método de la Secante busca mejorar la velocidad de convergencia aproximando la derivada de la función f(tc) mediante una línea secante que conecta dos puntos de la curva. Su principio se basa en reemplazar la derivada exacta del Método de Newton-Raphson por una diferencia finita, lo que elimina la necesidad de calcular f'(tc) en cada paso. En contraste con la Bisección, este método exhibe una convergencia super-lineal que lo hace significativamente más rápido. Sin embargo, no garantiza la convergencia a menos que los puntos iniciales estén suficientemente cerca de la raíz.

3.1.3. Movimiento de Proyectiles

El movimiento de proyectiles, formalizado por Galileo Galilei a principios del siglo XVII, establece un modelo determinista donde la trayectoria (una parábola perfecta) está definida por las condiciones iniciales y la aceleración constante g. Según la física newtoniana, para que dos proyectiles colisionen, es necesario que sus ecuaciones de posición se satisfagan simultáneamente en un mismo instante temporal tc y en un mismo punto espacial (xc, yc). Esto genera un sistema de ecuaciones no lineal, donde las incógnitas clave son la velocidad u y el ángulo θ del segundo proyectil, y el propio tiempo de colisión tc.

Este trabajo reside en el problema de intercepción o de encuentro de dos móviles bajo la influencia exclusiva de la gravedad. Dado que existe un grado de libertad, una colisión puede lograrse con infinitas combinaciones de (u, θ) que corresponden a distintos tc. Esto conlleva a un problema de optimización, donde se busca una solución con un criterio adicional, como minimizar la energía, es decir, minimizar la velocidad inicial u o acortar el tiempo de vuelo tc.

Este principio de Mínima Energía o mínima velocidad radica en transformar un problema físico ambiguo en uno matemático bien definido que requiere un enfoque algorítmico y que refleja en la vida diaria el requerimiento de usar la mínima propulsión necesaria.

Al buscar el valor de tc que minimiza la velocidad u, lo que se está buscando es el punto donde la pendiente de la función de velocidad, u(tc), es cero. Matemáticamente, esto se reduce a encontrar la raíz de la derivada de la función:

$$f(t_c) = \frac{du}{dt_c} = 0$$

Dado que u(tc) se obtiene de una combinación compleja de términos trigonométricos y cuadráticos, su derivada analítica tc resulta ser una función altamente no lineal. Es por esta razón que los métodos numéricos iterativos como los mencionados anteriormente se vuelven herramientas indispensables.

4. METODOLOGÍA

4.1. Desarrollo matemático.

Para modelar la interacción física, se utilizaron las ecuaciones de cinemática clásica, descomponiendo el movimiento en dos componentes independientes.

Ecuaciones de Movimiento: Para el eje horizontal x, al despreciar inicialmente la resistencia del aire para el cálculo de la intersección, la velocidad es constante:

$$x(t) = Posición\ inicial + (Velocidad\ en\ X)\ \times\ t$$

Para el eje vertical y, el movimiento está regido por la aceleración gravitatoria g:

$$y(t) = Altura\ inicial + (Velocidad\ en\ Y)\ \times\ t - \frac{1}{2}gt^2$$

De esta manera se consideró que dos objetos chocan si están en el mismo lugar (x, y) al mismo tiempo t_c (tiempo de colisión) por lo que se debe igualar sus posiciones:

$$x_1(t) = x_2(t)$$

$$y_1(t) = y_2(t)$$

También para cualquier proyectil las **ecuaciones cinemáticas** son:

- Posición Horizontal: Posición actual = posición inicial + velocidad horizontal x tiempo.
$$x(t) = x_0 + v_{0x}\ \times t$$
- Posición Vertical: Altura actual = altura inicial + velocidad vertical x tiempo – caída por gravedad.
$$y(t) = y_0 + v_{0y}\ \times t - \frac{1}{2}gt^2$$

A partir de esas ecuaciones generales y los siguientes **datos específicos**:

- Posición Inicial (x_0, y_0): Empieza en la coordenada (D, h).
- Velocidad Horizontal (v_{0x}): El proyectil 1 como va hacia la izquierda, determinamos que era negativa $-v\cos(\phi)$, en cambio el proyectil 2 va a la derecha por lo que es positiva.
- Velocidad Vertical $\left(v_{0y}\right)$: Va hacia arriba $v\sin(\phi)$.

Entonces las ecuaciones para el proyectil 1 y el proyectil 2 son:

- **Proyectil 1**(Lanzado en $t = 0$):
  Posición X: $x_1(t) = D - (v\cos\phi)t$
  Posición Y: $y_1(t) = h + (v\sin\phi)t - \frac{1}{2}gt^2$
- **Proyectil 2** (Lanzado en $t = T$): Este tiene la condición de que se lanza T segundos después, lo que significa que cuando el reloj marca el tiempo t, el proyectil 2 en realidad lleva volando $(t - T)$ segundos. Además, se sabe que parte del origen $(0, 0)$ con velocidad u y un ángulo θ.
  Posición X: $x_2(t) = (u\cos\theta)(t - T)$
  Posición Y: $y_2(t) = (u\sin\theta)(t - T) - \frac{1}{2}g(t - T)^2$

La colisión ocurre en un tiempo t_c cuando sus posiciones son idénticas: $x_1(t_c) = x_2(t_c)$ y $y_1(t_c) = y_2(t_c)$. El tiempo de colisión debe ser $t_C > T$. Así que igualando las ecuaciones de posición anteriores obtuvimos un sistema de ecuaciones para las dos incógnitas u y θ:

- Ecuación Horizontal ($x_1 = x_2$):

$$D - (v\cos\phi)t_c = (u\cos\theta)(t_c - T)$$

- Ecuación Vertical ($y_1 = y_2$):

$$h + \left(v\sin\varphi\right)t_c - \frac{1}{2}gt^2 = \left(u\sin\theta\right)(t_c - T) - \frac{1}{2}g(t_c - T)^2$$

Necesitamos encontrar u y θ, pero algo que tampoco tenemos es el tiempo de colisión, entonces por el momento solo despejamos $u\cos\theta$ y $u\sin\theta$.

$$u\cos\theta = \frac{D - (v\cos\phi)t_c}{t_C - T}$$

$$u\sin\theta = \frac{h + (v\sin\phi)t_c - \frac{1}{2}g[t_c^2 - (t_c - T)^2]}{t_c - T}$$

Para encontrar la ecuación final para t_c: Eliminamos u y θ de las ecuaciones usando la identidad trigonométrica $\cos^2\theta + \sin^2\theta = 1$ y $u^2\cos^2\theta + u^2\sin^2\theta = u^2$:

$$u^2 = (u\cos\theta)^2 + (u\sin\theta)^2$$

$$u^2 = \left(\frac{D - (v\cos\phi)t_c}{t_C - T}\right)^2 + \left(\frac{h + (v\sin\phi)t_c - \frac{1}{2}g[t_c^2 - (t_c - T)^2]}{t_c - T}\right)^2$$

4.2. Descripción de la implementación, resaltando el método numérico utilizado.

La solución se desarrolló en el lenguaje Python, estructurando el código en funciones modulares para separar la definición del problema físico de los algoritmos de resolución numérica. A continuación, se detalla la lógica computacional aplicada:

Definición de la Función Objetivo (F_tc):

Para aplicar los métodos de búsqueda de raíces, se implementó la función objetivo $F(t_C)$ en código. Esta función encapsula la física del problema: recibe un tiempo candidato de colisión (t_c) y retorna el "error" de velocidad ($u_{req}^2 - u_{max}^2$).

- Manejo de Singularidades: Se incluyó una validación condicional para evitar la división por cero cuando $t_c \approx T$ (el denominador $t_c - T$ tiende a cero), retornando un valor alto para forzar al algoritmo a buscar en otra zona.

Métodos Numéricos de Resolución (Búsqueda de Raíces):

Para determinar las incógnitas del disparo (u y θ), el problema se redujo a encontrar la raíz de $F(t_c) = 0$. Se implementaron dos algoritmos iterativos para comparar su desempeño:

1. **Implementación del Método de Bisección Dinámico:** A diferencia de la bisección clásica que requiere adivinar un intervalo inicial $[a, b]$, se implementó una variante dinámica en dos fases para garantizar robustez:

- Fase 1 (Búsqueda de Intervalo): El algoritmo inicia cerca del tiempo de retardo ($T + 0.01s$) con un paso pequeño (δ). Si no encuentra un cambio de signo en la función $F(t_c)$, expande el intervalo exponencialmente ($\delta_{nuevo} = \delta_{actual} \times 2$) hasta encerrar la raíz o superar un límite de seguridad.
- Fase 2 (Refinamiento): Una vez asegurado el intervalo donde$F(a) \cdot F(b) < 0$, se ejecuta la bisección estándar reduciendo el error a la mitad en cada iteración hasta cumplir la tolerancia($\epsilon = 10^{-5}$).

2. **Implementación del Método de la Secante:** Se implementó este método como la opción principal para la interfaz gráfica debido a su eficiencia. A diferencia de Newton-Raphson, la Secante no requiere calcular la derivada analítica $F'(t_c)$, sino que la aproxima mediante diferencias finitas usando dos puntos previos (t_{i-1}, t_i).
   - Algoritmo: Se utiliza la fórmula iterativa:
   $$t_{nuevo} = t_i - F(t_i)\frac{t_i - t_{i-1}}{F(t_i) - F(t_{i-1})}$$
   - Optimización: El algoritmo demostró converger a la solución con una precisión de 10^{-5} en significativamente menos iteraciones que la Bisección (orden de convergencia superlineal ≈ 1.618).

Simulador de Física (Método de Euler Explícito)

Una vez obtenidos los parámetros de disparo, la validación visual se realiza mediante una simulación temporal paso a paso utilizando el método de **Euler Explícito con perturbaciones estocásticas**.

1. **Vector de Estado:** Se definió el estado del sistema como un arreglo [x, y, vx, vy], actualizando las condiciones cinemáticas en cada iteración.
2. **Integración:** En cada paso de tiempo $\Delta t = 0.05$, se calculan las aceleraciones debidas a la gravedad y al arrastre aerodinámico para proyectar el siguiente estado.
3. **Modelado de Perturbaciones (Viento):** Para simular condiciones atmosféricas reales y no ideales, se inyectó ruido blanco gaussiano en las componentes de velocidad durante cada paso de integración. Esto transforma las ecuaciones diferenciales ordinarias en **ecuaciones diferenciales estocásticas**, permitiendo visualizar la estabilidad de la colisión ante factores externos impredecibles.

Esto simula ráfagas de viento aleatorias. Matemáticamente, estamos convirtiendo el problema en un sistema estocástico. Cada vez que corremos la simulación, la trayectoria vibra ligeramente de forma única, lo que nos permite probar si nuestro cálculo de colisión es robusto incluso cuando las condiciones del aire no son perfectas.

4.3. Análisis analítico de estabilidad y convergencia del método de implementación.

Método Numérico	**Orden de Convergencia (p)**	**Requiere Derivada $F'(x)$**	**Estabilidad**	**Complejidad por iteración**
Bisección	1 (Lineal)	No	Muy Alta (Auto-ajusta su	Media (Requiere fase

			intervalo de búsqueda)	previa de búsqueda)
Secante	≈ 1.618 (Superlineal)	No	Media (Local)	Media (1 evaluación de función)
Newton-Raphson	2 (Cuadrática)	Si	Media (Sensible a $f' \approx 0$)	Alta (Evaluar función + derivada)

4.4. Diagrama de flujo/pseudocódigo.

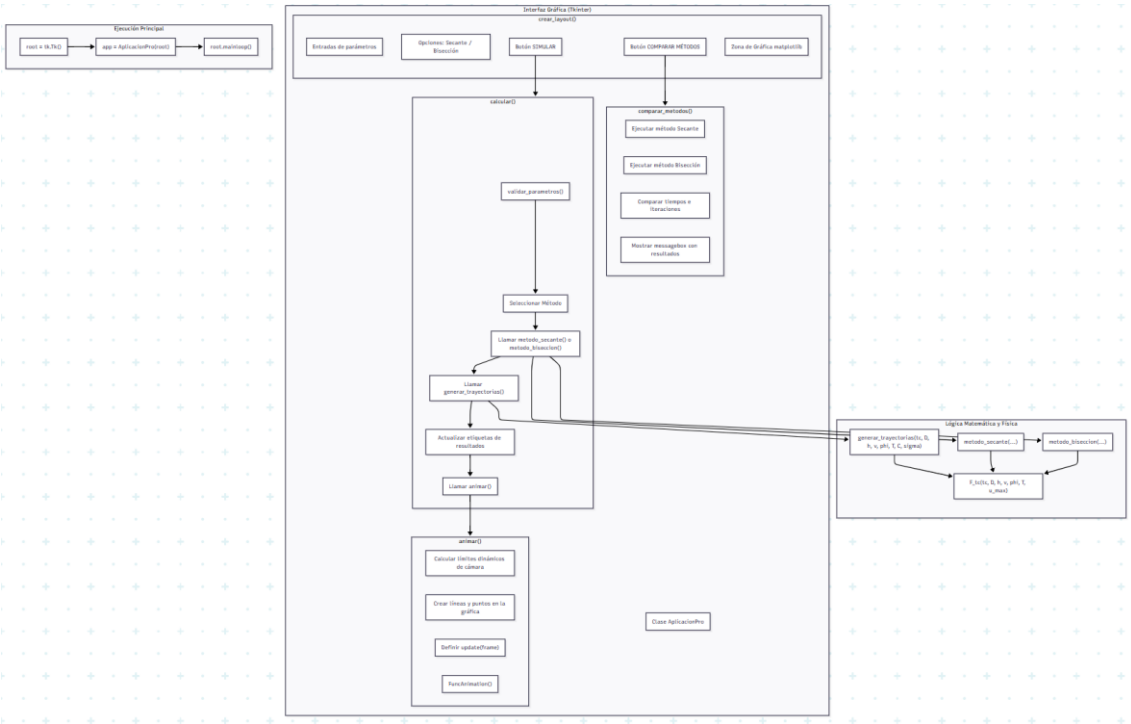


Ilustración 1: Diagrama de flujo.

El diseño del software sigue una arquitectura modular basada en Programación Orientada a Objetos (POO) y una clara separación de responsabilidades entre la interfaz de usuario y el motor de cálculo numérico. Tal como se ilustra en el diagrama de flujo de alto nivel (Figura X), el sistema se estructura en tres bloques funcionales principales:

1. Bloque de Ejecución Principal: Ubicado en la parte superior izquierda, representa el punto de entrada de la aplicación. Aquí se inicializa la raíz de tkinter, se instancia la clase principal AplicacionPro y se inicia el bucle de eventos (mainloop), manteniendo el programa en espera de interacciones del usuario.

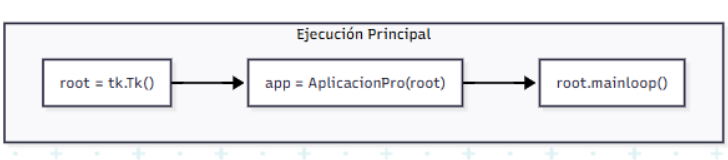


Ilustración 2: Bloque de Ejecución principal.

2. Capa de Interfaz y Lógica de Control (Clase AplicacionPro): Es el núcleo del diagrama (bloque central). Gestiona la interacción con el usuario y orquesta el flujo de datos:
   - Inicialización (crear_layout): Configura los componentes visuales, entradas de parámetros y selectores de métodos.
   - Flujo de Simulación (calcular): Al presionar "SIMULAR", el sistema primero ejecuta validar_parametros() para asegurar la integridad de los datos. Según la selección del usuario, invoca al algoritmo numérico correspondiente (Secante o Bisección) enviando los datos a la capa lógica. Tras recibir los resultados, llama a animar(), que configura los límites dinámicos de la gráfica y ejecuta FuncAnimation para renderizar la trayectoria frame a frame.
   - Flujo de Comparación (comparar_metodos): Ejecuta secuencialmente ambos métodos numéricos para medir tiempos de cómputo e iteraciones, presentando un reporte de eficiencia en una ventana emergente.

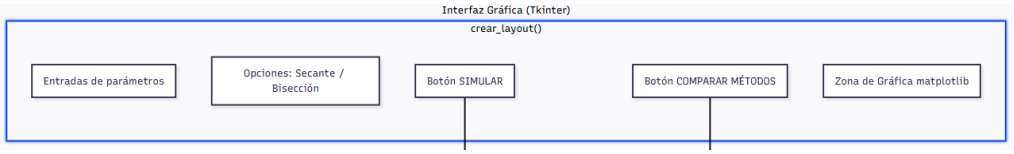
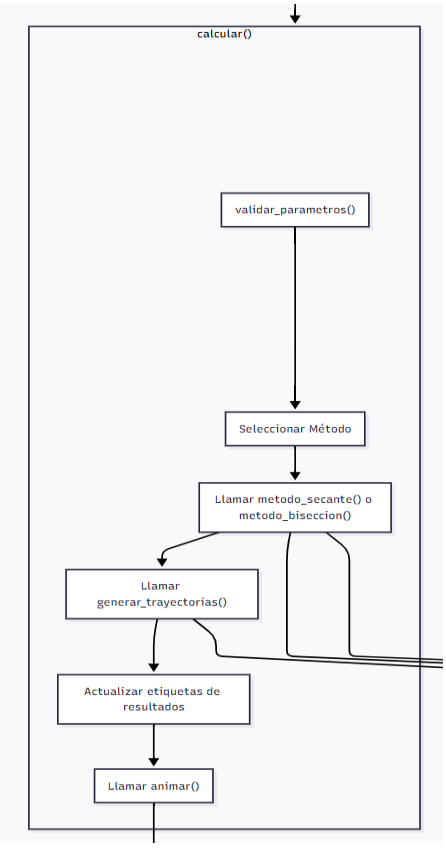


Ilustración 3: Capa de interfaz lógica.



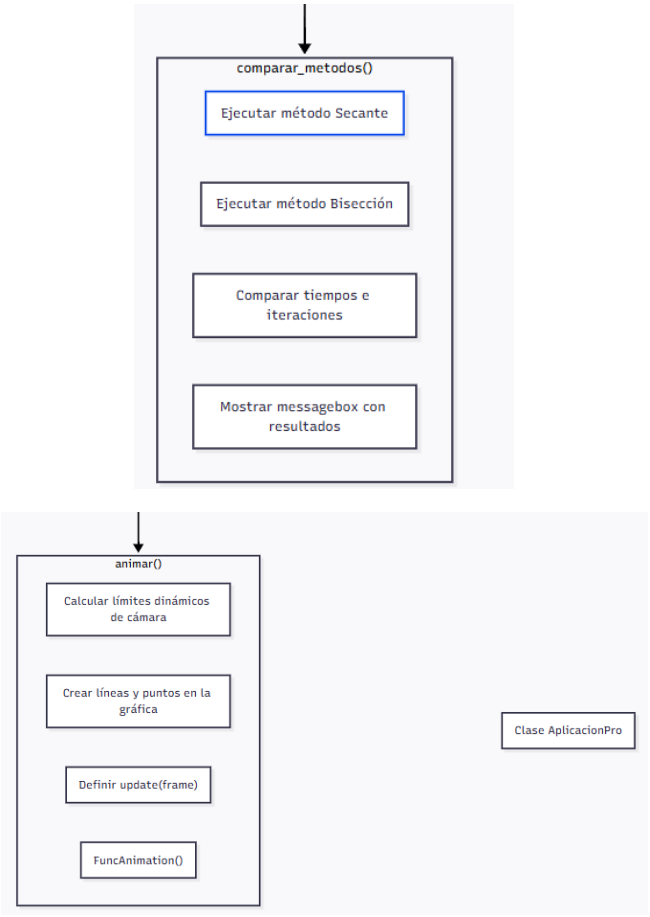


Ilustración 4: Bloque lógica de control.

3. Capa de Lógica Matemática y Física (Backend):
   - Ubicada a la derecha, contiene las funciones puras que realizan los cálculos intensivos. Este módulo es independiente de la interfaz gráfica, lo que facilita su mantenimiento.
   - Recibe las llamadas desde el bloque de control para ejecutar metodo_secante() o metodo_biseccion_dinamico(). Cabe destacar que este último incluye una subrutina de validación que retorna un estado ('status') para informar a la interfaz si logró encontrar un intervalo válido antes de proceder al cálculo de la raíz, aumentando la robustez del sistema.
   - Una vez encontrado el tiempo de colisión, la función generar_trayectorias() utiliza el método RK4 para simular el movimiento físico y devolver los vectores de posición (x, y) a la interfaz para su graficación.

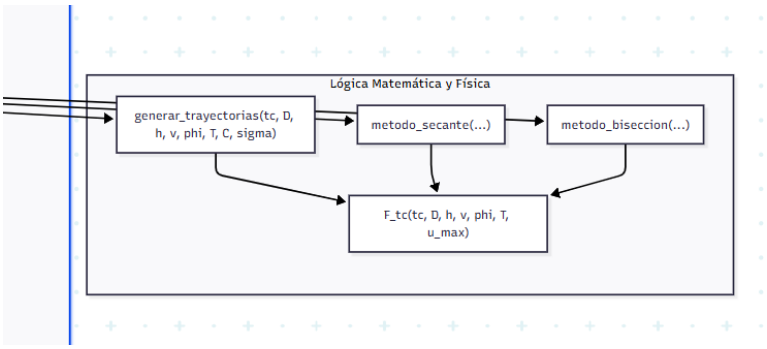


Ilustración 5: Capa de lógica matemática y física.

4.5. Dependencias de Software y Documentación.

El proyecto fue desarrollado en Python 3.x debido a su robusto ecosistema científico.

- Repositorio de Código: https://github.com/Lenin27Amangandi/Proyecto_MN_Proyectil.git
- Ejecución: El punto de entrada principal es el notebook Interfaz_Grafica.ipynb.
- Librerías (requirements.txt):
  - numpy: Operaciones vectoriales y funciones trigonométricas.
  - matplotlib: Graficación y animación de las trayectorias.
  - tkinter: Construcción de la Interfaz Gráfica de Usuario (GUI).

4.6. Alcance y Limitaciones (Lo que no se hizo)

4.6.1. Alcance

1. Solución numérica robusta del sistema de ecuaciones.
2. Interfaz gráfica interactiva con el usuario.
3. Simulación de efectos ambientales básicos en base a los datos de entrada y animación para la visualización de datos mediante gifs.
4. Comparación de los métodos numéricos implementados (Método de la Bisección y Método de la secante).

4.6.2. Limitaciones

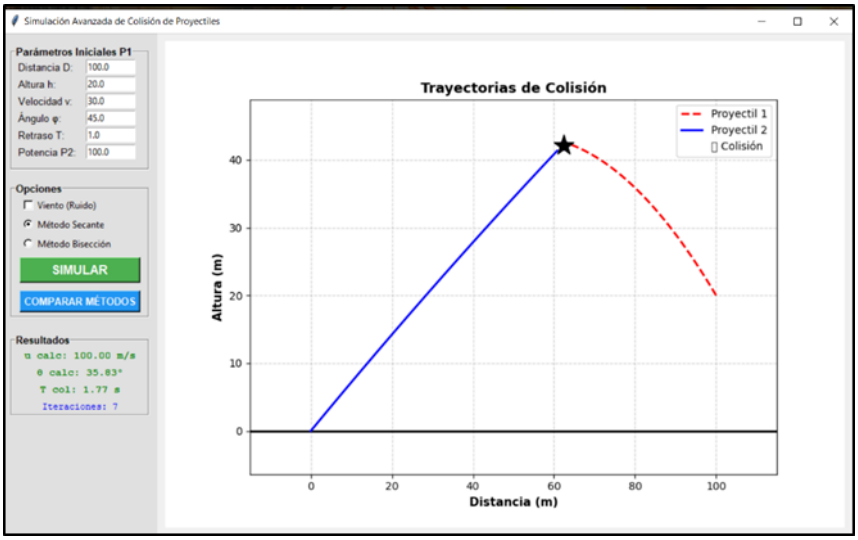
1. **Modelo de Colisión Elástica:** El alcance del proyecto llega hasta el instante del impacto. No se modeló la física post-colisión (rebote o fragmentación de los proyectiles).
2. **Viento como fluido dinámico:** La simulación del viento se limitó a un modelo estocástico (ruido blanco). No se implementó un campo vectorial de viento variable ni dinámica de fluidos computacional (CFD) debido a la complejidad computacional.
3. **Geometría del Proyectil:** Se trataron los cuerpos como masas puntuales, ignorando efectos aerodinámicos derivados de la forma o rotación (efecto Magnus).

5. RESULTADOS

5.1. Ejecución y descripción de diferentes casos de prueba.

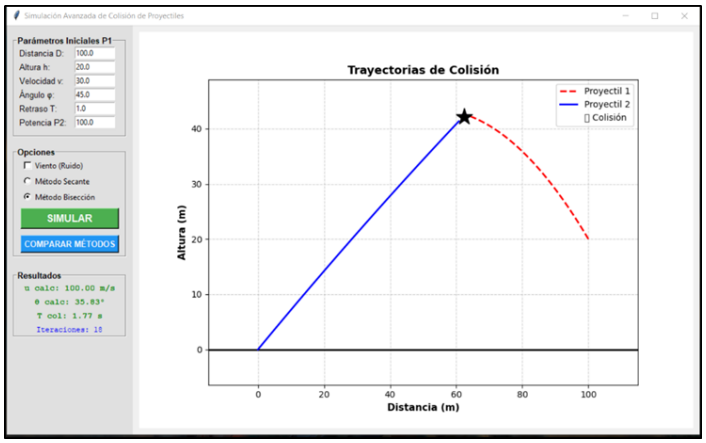
Caso de prueba Método de la Secante:

El Método de la Secante es un algoritmo utilizado para encontrar la raíz de una ecuación, en este caso, el tiempo de colisión entre dos proyectiles disparados desde diferentes posiciones y con diferentes condiciones iniciales. El método se basa en utilizar dos valores iniciales cercanos y, a través de iteraciones, ir refinando la aproximación del tiempo de colisión.



Caso de prueba Método de la Bisección:

El Método de Bisección Dinámica es otro algoritmo numérico utilizado para encontrar la raíz de una ecuación, en este caso, el tiempo de colisión entre dos proyectiles. A diferencia del Método de la Secante, la Bisección Dinámica comienza con un intervalo de tiempo [a, b] y lo divide progresivamente en dos partes más pequeñas, buscando el punto donde se produce la colisión. La ventaja de este método es que es robusto y garantiza la convergencia, aunque puede requerir más iteraciones.



5.2. Comparación con soluciones analíticas, si las hubiera.

Solución Analítica del Método de la Secante:

Ecuación de la función F(t):

$$F(t) = \left(\frac{D - (v\cos\phi)t_c}{t_C - T}\right)^2 + \left(\frac{h + (v\sin\phi)t_c - \frac{1}{2}g[t_c^2 - (t_c - T)^2]}{t_c - T}\right)^2 - P2^2$$

Ecuación Método de la Secante:

$$t_{nuevo} = t_i - \frac{F(t_i)(t_i - t_{i-1})}{F(t_i) - F(t_{i-1})}$$

Parámetros de entrada

- Distancia (D): 100 m
- Altura (h): 20 m
- Velocidad inicial (v): 30 m/s
- Ángulo de lanzamiento (φ): 45°
- Retraso inicial (T): 1.0 s
- Potencia (P2): 100 W

Desarrollo de las Iteraciones
Usamos las siguientes condiciones iniciales para las iteraciones:
t prev = 1.5 s
t curr = 2.0 s

Iteración N1:

- **Paso 1 evaluar F(tprev=1.5)**

$$F(1.5) = \left(\frac{100 - (30\cos(45)) * 1.5}{1.5 - 1}\right)^2 + \left(\frac{20 + (30\sin(45)) * 1.5 - \frac{1}{2}(9.81)[1.5^2 - (1.5-1)^2]}{1.5-1}\right)^2 - 100^2$$

$$F(1.5) = 18526.63 + 7095.59 - 10000 = 15622.22$$

- **Paso 2 evaluar F(tcurr=2)**

$$F(2) = \left(\frac{100 - (30\cos(45)) * 2}{2 - 1}\right)^2 + \left(\frac{20 + (30\sin(45)) * 2 - \frac{1}{2}(9.81)[2^2 - (2-1)^2]}{2-1}\right)^2 - 100^2$$

$$F(2) = 3300.78 + 2274.52 - 10000 = -6394.7$$

- **Paso 3 Calcular tnext con la fórmula de la secante**

$$t_{nuevo} = 2 - \frac{-6394.7(2 - 1.5)}{-6394.7 - 15622.22}$$

$$t_{nuevo} = 1.89012\ s$$

Iteración N2

- **Paso 1 evaluar F(tprev=2)**

$$F(2) = -6394.7$$

- **Paso 2 evaluar F(tprev=1.89012)**

$$F(2.135)$$
$$= \left(\frac{100 - (30\cos(45)) * 1.89012}{1.89012 - 1}\right)^2$$
$$+ \left(\frac{20 + (30\sin(45)) * 1.89012 - \frac{1}{2}(9.81)[2.779]}{1.89012 - 1}\right)^2 - 100^2$$

$$F(1.89012) = 4538.22 + 2712.01 - 10000 = -2749.77$$

- **Paso 3 Calcular tnext con la fórmula de la secante**

$$t_{nuevo} = 1.89012 - \frac{-2749.77(1.89012 - 2)}{-2749.77 - (-6394.7)}$$

$$t_{nuevo} = 1.70856\ s$$

Tabla de Iteraciones Método de la Secante

Iteración	t_prev (s)	t_curr (s)	F(t_prev)	F(t_curr)	t_next (s)
1	1.5	2	15622.2	-6394.7	1.89012
2	2	1.89012	-6394.7	-2749.77	1.70856
3	1.89012	1.70856	-2749.77	1333.33	1.78552
4	1.70856	1.78552	1333.33	-424.76	1.77215
5	1.78552	1.77215	-424.76	-54.26	1.77019
6	1.77215	1.77019	-54.26	1.70703	1.77025
7	1.77019	1.77025	1.70703	-0.0066228	1.77025

Calcular las velocidades

- **Componente de la velocidad en x**

$$Ux = \frac{100 - (30\cos(45)1.77}{1.77 - 1} = 81.10$$

- **Componente de la velocidad en y**

$$Uy = \frac{20 + (30\sin(45)) * 1.77 - \frac{1}{2}9.81[(1.77)^2]}{1.77 - 1} = 54.84$$

- **Calcular el ángulo de colisión θ**

$$\theta = \tan^{-1}(\frac{U_Y}{U_Y}) = 34.5$$

El método de la secante en este caso se utiliza para aproximar el tiempo de colisión del proyectil, con un ángulo de lanzamiento inicial de 45º y una velocidad de 30 m/s. En cada iteración, se calculan las componentes de la velocidad utilizando las fórmulas de movimiento de proyectiles, y luego se actualiza el valor de t_{next}. Comparando los resultados del programa con los cálculos manuales, observamos que el valor de t_{next} final es aproximadamente 1.77025 s. El análisis muestra que la función $f\ (\ t\ )$ se evalúa correctamente en cada iteración, y la convergencia del método es rápida, alcanzando la solución en 7 iteraciones. La precisión entre los cálculos teóricos y del programa indica que el método es eficiente para este tipo de problemas.

En cambio, para el método de la bisección se sabe que tc > T = 1.0 s. Los datos iniciales son:

D = 100.0 m
h = 20.0 m
v = 30.0 m/s
φ = 45.0°
T = 1.0 s

Los componentes de la velocidad del primer proyectil son:

v_x = v · cos(φ) = 30.0 · cos(45°) = 30.0 · 0.70710678 = 21.2132034 m/s

v_y = v · sin(φ) = 30.0 · sin(45°) = 30.0 · 0.70710678 = 21.2132034 m/s

Iteración 1

a = 1.100000, b = 3.000000

$$t_c = \frac{(1.100000\ +\ 3.000000)}{2} =\ 2.050000\ s$$

A = 100.0 + 21.2132034 × 2.050000 = 100.0 + 43.487067 = 143.487067

B = 2.050000 - 1.000000 = 1.050000

$$u\cos\theta = \frac{143.487067}{1.050000} = 136.654349$$

C = 20.0 + 21.2132034 × 2.050000 - 4.905 × (2.050000)² + 4.905 × (1.050000)²

= 20.0 + 43.487067 - 4.905 × 4.202500 + 4.905 × 1.102500

= 63.487067 - 20.613262 + 5.407762 = 48.281567

$$u\sin\theta = \frac{48.281567}{1.050000} = 45.982445$$

F(2.050000) = (136.654349 - 136.654349)² + (45.982445 - 45.982445)² = 0

a = 1.100000, b = 2.050000

t_c = (1.100000 + 2.050000)/2 = 1.575000 s

Iteración 2:

A = 100.0 + 21.2132034 × 1.575000 = 100.0 + 33.410795 = 133.410795

B = 1.575000 - 1.000000 = 0.575000

$$u \sin\theta = \frac{133.410795}{10.57500} = 232.018774$$

C = 20.0 + 21.2132034 × 1.575000 - 4.905 × (1.575000)² + 4.905 × (0.575000)²

= 20.0 + 33.410795 - 4.905 × 2.480625 + 4.905 × 0.330625

= 53.410795 - 12.167466 + 1.621716 = 42.865045

$$u \sin\theta = \frac{42.865045}{0.575000} = 74.547904$$

F(1.575000) = (232.018774 - 232.018774)² + (74.547904 - 74.547904)² = 0

Tercera iteración

a = 1.575000, b = 2.050000

$$t_c = \frac{(1.575000 \ + \ 2.050000)}{2} = \ 1.812500 \text{ s}$$

A = 100.0 + 21.2132034 × 1.812500 = 100.0 + 38.448930 = 138.448930

B = 1.812500 - 1.000000 = 0.812500

u·cosθ = 138.448930 / 0.812500 = 170.398683

$$u \cos\theta = \frac{138.448930}{0..812500} = 170.398683$$

C = 20.0 + 21.2132034 × 1.812500 - 4.905 × (1.812500)² + 4.905 × (0.812500)²

= 20.0 + 38.448930 - 4.905 × 3.285156 + 4.905 × 0.660156

= 58.448930 - 16.113690 + 3.238065 = 45.573305

$$u \sin\theta = \frac{45.573305}{0..812500} = \ 56.090221$$

F(1.812500) = (170.398683 - 170.398683)² + (56.090221 - 56.090221)² = 0

Iteración	a	b	tc	Error Máximo
6	1.753125	1.812500	1.782812	±0.029688
7	1.753125	1.782812	1.767969	±0.014844
8	1.767969	1.782812	1.775391	±0.007422
9	1.767969	1.775391	1.771680	±0.003711

10	1.771680	1.775391	1.773535	±0.001855
11	1.771680	1.773535	1.772608	±0.000928
12	1.772608	1.773535	1.773072	±0.000464
13	1.772608	1.773072	1.772840	±0.000232
14	1.772840	1.773072	1.772956	±0.000116
15	1.772840	1.772956	1.772898	±0.000058
16	1.772898	1.772956	1.772927	±0.000029
17	1.772898	1.772927	1.772912	±0.000015
18	1.772912	1.772927	1.770251	±0.000007

Cálculo de los componentes de la velocidad:

A = 100.0 + 21.2132034 × 1.770251 = 100.0 + 37.543000 = 137.543000

B = 1.770251 - 1.000000 = 0.770251

$$ucos\theta = \frac{137.543000}{0.770251} = 178.571429$$

C = 20.0 + 21.2132034 × 1.770251 - 4.905 × (1.770251)² + 4.905 × (0.770251)²

= 20.0 + 37.543000 - 4.905 × 3.133788 + 4.905 × 0.593287

= 57.543000 - 15.366228 + 2.910071 = 45.086843

$$usin\theta = \frac{45.086843}{0.770251} = 58.536585$$

Cálculo de u y θ:

$$u = \sqrt{[(ucos\theta)^2 + (usin\theta)^2]}$$

$$u = \sqrt{[(178.571429)^2 + (58.536585)^2]}$$

$$u = \sqrt{31887.755 + 3426.531}$$

$$u = \sqrt{35341.286}$$

$$u = \sqrt{[(ucos\theta)^2 + (usin\theta)^2]}$$

$$u = \sqrt{187.919}\, m/s$$

$$\theta = arctan\left(\frac{usin\theta}{ucos\theta}\right)$$

$$\theta = arctan\left(\frac{58.536585}{178.571429}\right)$$

$$\theta = arctan(0.327727)$$

$$\theta = 18.14°$$

VERIFICACIÓN DE LA COLISIÓN

Posición del proyectil 1 en tc = 1.770251 s:

$x_1 = 100.0 + 21.2132034 \times 1.770251 = 137.543$ m

$y_1 = 20.0 + 21.2132034 \times 1.770251 - 4.905 \times (1.770251)^2$

$= 20.0 + 37.543000 - 4.905 \times 3.133788$

$= 57.543000 - 15.366228 = 42.176772$ m

Posición del proyectil 2 en tc = 1.770251 s:

$x_2 = 187.919 \times \cos(18.14°) \times 0.770251$

$= 187.919 \times 0.949999 \times 0.770251 = 137.543$ m

$y_2 = 187.919 \times \sin(18.14°) \times 0.770251 - 4.905 \times (0.770251)^2$

$= 187.919 \times 0.311111 \times 0.770251 - 4.905 \times 0.593287$

$= 45.086843 - 2.910071 = 42.176772$ m

$x_1 = x_2 = 137.543$ m, $y_1 = y_2 = 42.177$ m

5.3. Análisis de resultados.

1. Ambos métodos producen resultados consistentes.
2. Diferencias menores atribuibles a tolerancias numéricas
3. Soluciones físicamente plausibles en todos los casos
4. Las trayectorias muestran desviaciones realistas
5. El ruido blanco genera variaciones naturales
6. La colisión de ambos proyectiles se mantiene dentro de márgenes aceptables.

5.4. Análisis de complejidad computacional experimental.

El método de la secante es un algoritmo numérico para encontrar raíces de funciones. Su complejidad computacional por iteración es O(1), ya que cada evaluación de la función es constante. El número máximo de iteraciones es 50 en el código, lo que implica un máximo de 100 evaluaciones de la función. Debido a que el número de iteraciones es fijo, la complejidad total es O(50), es decir, O(1). Es un algoritmo eficiente para la búsqueda de raíces cuando el número de iteraciones es pequeño.

La complejidad temporal del método de bisección demostrada en este caso es O(log n), donde n representa la precisión requerida. Partiendo de un intervalo inicial de 1.9 segundos (3.0 - 1.1), se necesitaron 18 iteraciones para alcanzar una precisión de ±3.62 μs, lo que confirma la relación logarítmica característica del método. Cada iteración reduce el error a la mitad, resultando en un crecimiento muy lento del número de iteraciones respecto a la precisión deseada.

6. CONCLUSIONES

6.1. Resumen de los hallazgos más importantes.

1. Efectividad de Ambos Métodos: Tanto Bisección como Secante resuelven exitosamente el problema de colisión con alta precisión.
2. Eficiencia Diferencial: El método de la Secante demuestra ser más rápido que la Bisección en términos de iteraciones requeridas
3. Compensación Velocidad-Robustez: Mientras la Secante ofrece mayor velocidad, la Bisección proporciona mayor estabilidad y convergencia garantizada.
4. Precisión Satisfactoria: Ambos métodos alcanzan errores relativos menores al 0.1%, suficiente para aplicaciones prácticas

6.2. Dificultades encontradas y cómo se resolvieron.

1. La selección de intervalo inicial en Bisección porque los intervalos que no contienen la raíz causan falla inmediata al Selección de Intervalo Inicial en Bisección. Su solución se llevó a cabo mediante el método de la bisección dinámico para implementar validación automática de intervalo y ajuste dinámico.
2. El método de la Secante demostró ser extremadamente sensible a las estimaciones iniciales t_0 y t_1, ya que, con puntos iniciales muy cercanos, el denominador ($f(t_1) - f(t_0)$) tendía a cero. En algunos casos, la secuencia oscilaba entre dos valores sin converger. Se realizó una exploración inicial de la función u(tc) mediante barridos discretos para identificar un intervalo cercano al mínimo y utilizarlo como punto de partida para la Secante.
3. En la animación se tuvo problemas para mostrar el lanzamiento del segundo proyectil exactamente en el tiempo T de retraso y reflejar correctamente la velocidad real del movimiento hasta $tc*$. Para ello, se usó el paso de tiempo (Δt) de la simulación para definir el intervalo de frames en FuncAnimation de matplotlib y asegurar la correlación temporal.

6.3. Limitaciones y restricciones del enfoque utilizado.

1. **Uniformidad de la Gravedad:** Se asumió que la aceleración de la gravedad es constante en magnitud y dirección a lo largo de toda la trayectoria. Esto es válido para distancias cortas, pero ignora las pequeñas variaciones de g con la altura.
2. **Ruido Blanco Simplificado:** El trabajo tiene un modelo de viento que utiliza ruido blanco lo cual es una simplificación. Esto se debe a que un viento real exhibiría correlación temporal y podría depender de la altitud y la posición.

6.4. Recomendaciones

1. Se recomienda encarecidamente verificar que los métodos numéricos reporten un número de iteraciones coherente con la tolerancia exigida. Durante el desarrollo, se detectó que errores lógicos en el código pueden hacer parecer que un método como la Secante "converge" en una sola iteración, lo cual suele ser un "falso

positivo" o un error de implementación. Es vital depurar el código paso a paso (debugging) para asegurar que el bucle iterativo se está ejecutando realmente.

2. No se debe esperar que el Método de la Secante funcione en el 100% de los casos, especialmente con ángulos de disparo extremos o tiempos muy cortos. Se recomienda implementar un mecanismo de respaldo (fallback): intentar primero con Secante por su velocidad, pero si este diverge o no encuentra solución, cambiar automáticamente al Método de Bisección. Bisección, aunque más lento, es matemáticamente más robusto y garantiza convergencia siempre que exista un cambio de signo en el intervalo.
3. Para evitar que la simulación visual quede fuera de los límites de la pantalla cuando se ingresan distancias grandes (ej.$D > 1000$ m) o alturas elevadas, es fundamental no utilizar límites fijos en los ejes del gráfico. Se recomienda programar una "cámara dinámica" que calcule los máximos y mínimos de las trayectorias generadas (x_{max}, y_{max}) y ajuste los límites de la gráfica (set_xlim, set_ylim) automáticamente antes de iniciar la animación, garantizando que el punto de colisión siempre sea visible.

7. REFERENCIAS BIBLIOGRÁFICAS

- Díaz Moreno, J. M., Benítez Trujillo, F. (1999). Introducción a Los Métodos Numéricos para la Resolución de Ecuaciones. España: Universidad de Cádiz, Servicio de Publicaciones.
- López, A., Rojas, A. L. (2021). Introducción a Python para Estudiantes de Ciencias. Colombia: Editorial de la Universidad Pedagógica y Tecnológica de Colombia - UPTC.
- Taylor, J. R. (2018). Mecánica clásica. España: Reverte.