Escuela Politécnica Nacional

Facultad de Ingeniería de Sistemas

ESCUELA
POLITÉCNICA
NACIONAL

Métodos Numéricos

Integrantes:

Francis Alian Bravo Alemán

Matthew Israel Cedeño Diaz

Michael Javier Yánez Vela

Proyecto Telescopio James Webb

Ing. Jonathan A. Zea

1. Introducción

El telescopio espacial James Webb representa uno de los avances tecnológicos más significativos en la historia de la exploración del universo, marcando un hito en nuestra capacidad para comprender los orígenes del cosmos y los procesos que lo rigen. Este instrumento, diseñado para observar las profundidades del universo en longitudes de onda infrarrojas, combina ingeniería de vanguardia con una arquitectura óptica altamente sofisticada. Su sistema óptico incluye un espejo primario segmentado de 6.5 metros de diámetro y un espejo secundario ajustable, cuya precisión milimétrica es fundamental para garantizar imágenes claras y detalladas de galaxias, exoplanetas y otros fenómenos celestes.

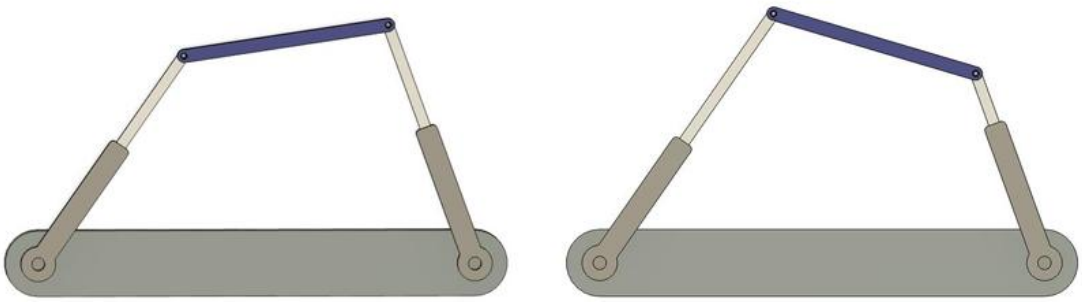


Figura 1: Modelo del Telescopio.

El espejo secundario, pieza clave en el sistema, emplea pistones lineales controlados con extrema exactitud para orientarse y mantener el alineamiento necesario en condiciones extremas del espacio profundo. Inspirados en este fascinante mecanismo, el presente proyecto propone la implementación de un sistema simplificado que modela el comportamiento de un espejo único controlado mediante pistones. Este sistema busca replicar, a pequeña escala, las complejas operaciones de orientación y ajuste, destacando la importancia de la precisión y la lógica computacional en la exploración espacial. A través de simulaciones y cálculos geométricos avanzados, este proyecto no solo rinde homenaje a la tecnología detrás del James Webb, sino que también introduce principios 4 fundamentales aplicables a sistemas ópticos modernos.

2. Objetivos

Formular un modelo matemático que represente el comportamiento de un mecanismo de orientación basado en pistones lineales, inspirado en el sistema del espejo secundario del telescopio James Webb. Este modelo permitirá controlar con exactitud la posición e inclinación del espejo respecto a un punto P dado, garantizando que el eje óptico quede perfectamente alineado con el blanco deseado.

Desarrollar una simulación gráfica del mecanismo para contrastar y validar el modelo propuesto en distintas configuraciones. En ella se analizará cómo los cambios en la longitud de los pistones influyen en la orientación del espejo, aportando datos sobre las tolerancias del sistema y sus márgenes de operación.

3. Metodologia

El desarrollo de la aplicación se basó en un enfoque modular y iterativo que consta de los siguientes pasos:

1. **Definición de requerimientos**: Identificar las variables clave (L, B, D, d_max, F, Px, Py) que describen la geometría del espejo y la posición del punto de incidencia.

2. **Diseño modular**: Separar la lógica en cuatro componentes:
    - **Interfaz gráfica** (Tkinter): Entradas de usuario y presentación de resultados.
    - **Cálculo geométrico**: Determinación de longitudes de pistones y orientación del espejo.
    - **Dibujo del sistema** (Matplotlib): Renderizado visual de pistones, espejo y proyección.
    - **Flujo principal** (`main.py`): Punto de arranque, configuración de fuente e icono.
3. **Iteración y validación**: Probar distintos valores de Px, Py y verificar que el código encuentre soluciones dentro de los límites de los pistones o genere advertencias y recalcule con valores extremos.
4. **Ajustes de presentación**: Incorporar estilos globales (fuente Times New Roman, colores de fondo) y separación de responsabilidades en archivos independientes.

3.1 Descripción de la solución

La aplicación recibe parámetros de entrada que definen tanto la geometría del espejo secundario como la posición de un punto P en el espacio. A través de un proceso de barrido:

1. Se recorre un rango de posibles posiciones verticales `My` del espejo, centrado en Py.
2. Para cada `My`, se calcula el vector normal `N` desde P hasta un punto medio en el espejo, y de él se deriva el vector director `D` (rotación de 90°).
3. Se evalúan las longitudes de pistones `x1` y `x2` para ver si caen dentro de los límites `[D, d_max]`. Si alguna combinación es válida, se detiene el barrido.
4. Si el barrido no arroja soluciones, se asignan valores extremos (`d_max`, `D`) y se reorienta el espejo en función del ángulo al punto P.
5. Finalmente, se calcula el ángulo del espejo respecto a la horizontal, se muestra en pantalla y se dibuja el sistema (pistones, espejo, punto y proyección).

3.2 Desarrollo matemático

Dado:

- Punto P = (Px, Py)
- Eje espejo de largo L centrado en un punto variable M = (0, My)
- Vectores básicos:
    - Normal al espejo: N=M−P=(−Px,My−Py)N = M - P = (-Px, My - Py)
    - Vector director del espejo (perpendicular a N):

$$D = \frac{1}{\|N\|}(-N_y, N_x)$$

Para cada extremo del espejo:

- Extremo izquierdo: $E_1 = M - \frac{L}{2}D$
- Extremo derecho: $E_2 = M + \frac{L}{2}D$

Longitudes de pistón:

$$x_1 = \|E_1 + (\tfrac{B}{2}, 0)\|, \quad x_2 = \|E_2 - (\tfrac{B}{2}, 0)\|$$

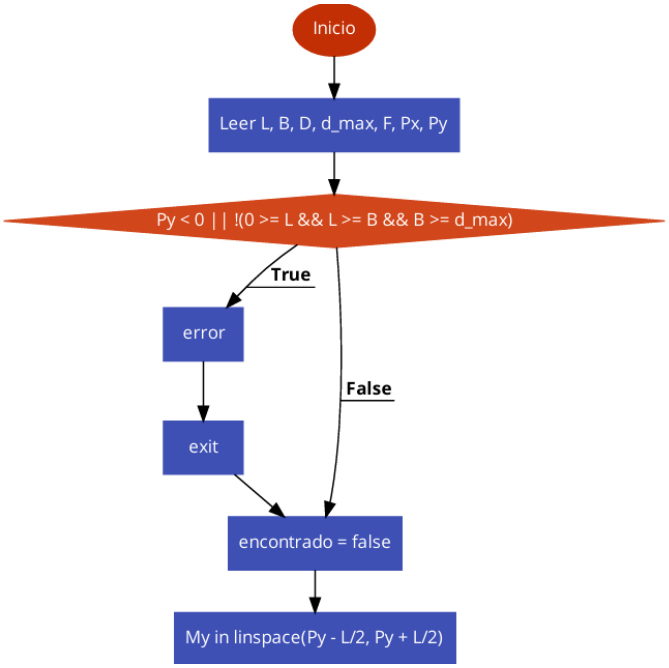
Condición de validez: D≤xi≤dmax Si no se cumple, usar valores extremos y recalcular con ángulo directo:

$$\theta = \arctan 2(Py, Px), \quad D = (\cos\theta, \sin\theta)$$

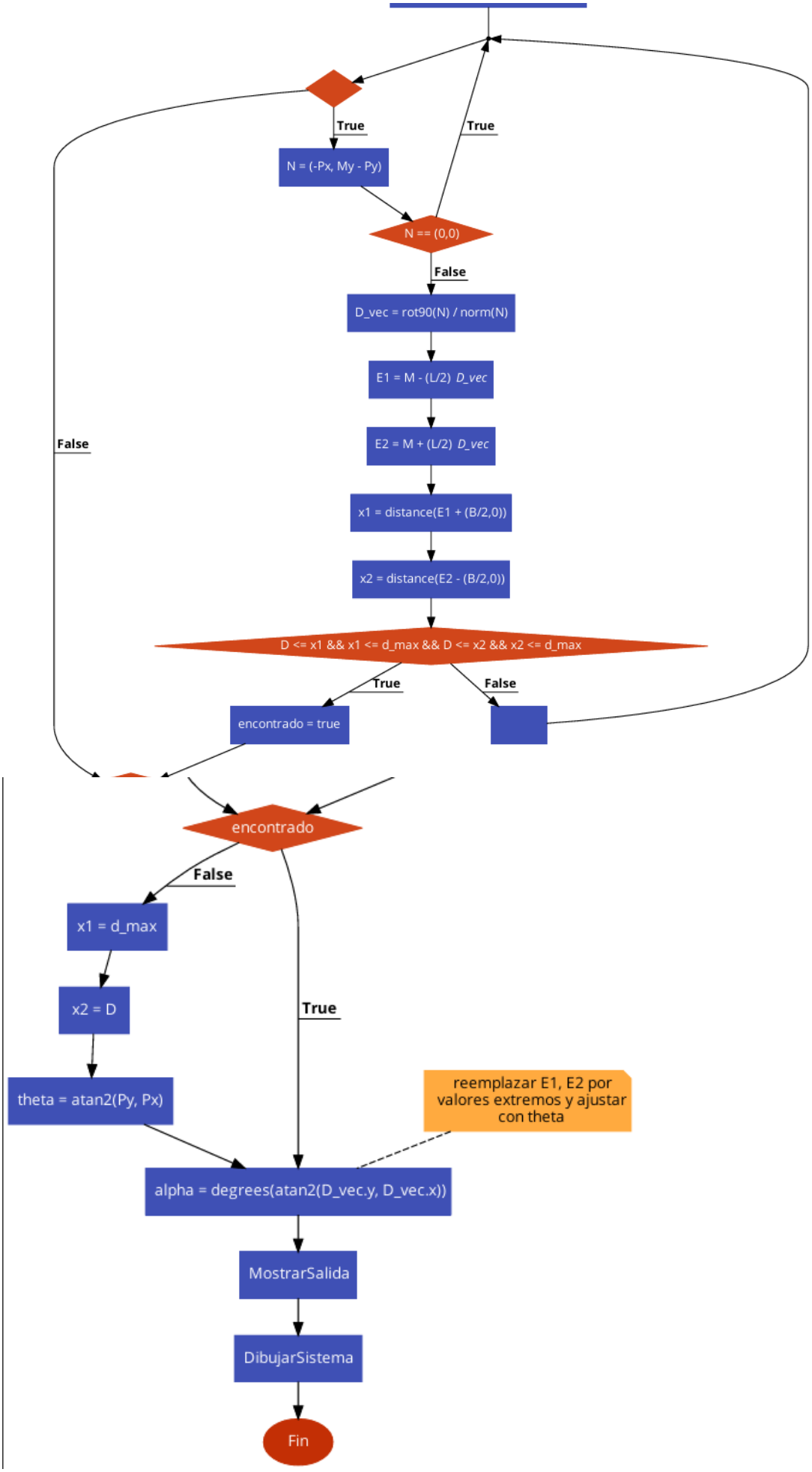
Ángulo final del espejo: α=θ+90 (o directamente α=atan2(Dy,Dx)).

3.3 Diagrama de flujo / Pseudocódigo

```
Inicio
  Leer L, B, D, d_max, F, Px, Py
  Si Py < 0 o 0 ≥ L ≥ B ≥ d_max → error y salir

  encontrado ← falso
  Para My en linspace(Py - L/2, Py + L/2):
    N ← (-Px, My - Py)
    si N == (0,0) → continuar
    D ← rot90(N) / ||N||
    E1 ← M - (L/2)*D; E2 ← M + (L/2)*D
    x1 ← distancia(E1 + (B/2,0)); x2 ← distancia(E2 - (B/2,0))
    si  D ≤ x1 ≤ d_max Y D ≤ x2 ≤ d_max:
      encontrado ← verdadero; romper
  FinPara

  Si ¬encontrado:
    x1 ← d_max; x2 ← D
    θ ← atan2(Py, Px)
    reemplazar E1, E2 con valores extremos y ajustar con θ
  FinSi

  α ← grados(atan2(D_y, D_x))
  Mostrar texto de salida
  Dibujar sistema
```




```
Fin
```


True
True
N = (-Px, My - Py)
N == (0,0)
False
D_vec = rot90(N) / norm(N)
E1 = M - (L/2) D_vec
E2 = M + (L/2) D_vec
False
x1 = distance(E1 + (B/2,0))
x2 = distance(E2 - (B/2,0))
D <= x1 && x1 <= d_max && D <= x2 && x2 <= d_max
True
False
encontrado = true
encontrado
False
x1 = d_max
x2 = D
True
theta = atan2(Py, Px)
reemplazar E1, E2 por valores extremos y ajustar con theta
alpha = degrees(atan2(D_vec.y, D_vec.x))
MostrarSalida
DibujarSistema
Fin

3.4 Detalles importantes de la implementación

Modularidad: Separación en `widgets.py`, `calculo.py`, `dibujo.py` y `ventana.py` para aislar responsabilidad.

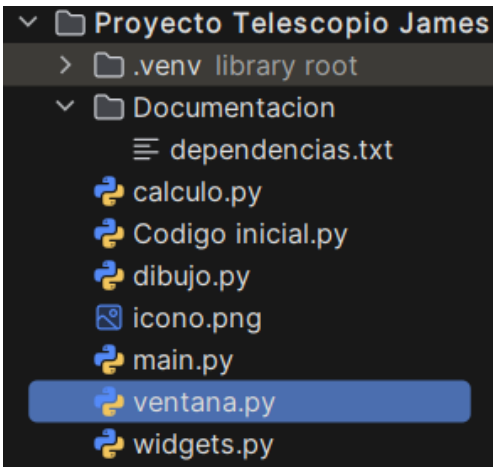


Figura 3. Modularidad

3.4.1 Main

La función main() inicializa la ventana principal de Tkinter, le asigna icono y fuente global, crea la instancia de AplicacionTelescopio(root) y arranca el bucle de eventos (mainloop).

```
import tkinter as tk
from tkinter import PhotoImage
from ventana import AplicacionTelescopio

def main():  1 usage
    root = tk.Tk()
    root.iconphoto( default: False, PhotoImage(file='icono.png'))
    ico = PhotoImage(file='icono.png')
    root.iconphoto( default: False, ico)
    root.option_add( pattern: "*Font",  value: "{Times New Roman} 12")
    app = AplicacionTelescopio(root)
    root.mainloop()

if __name__ == '__main__':
    main()
```

3.4.2 Calculo

El módulo **calculo.py** contiene la función calcular_y_dibujar(self), que se encarga de:

Leer y validar los parámetros de entrada (L, B, D, d_max, F, Px, Py) desde los DoubleVar de la interfaz, mostrando errores si no son numéricos o no cumplen las restricciones geométricas.

Determinar la posición óptima del espejo secundario mediante un barrido de su coordenada vertical My, calculando para cada paso los vectores normales y directores, y evaluando si las longitudes de pistones x_1 y x_2 quedan dentro de los límites permitidos.

Manejar el caso en que no se encuentra solución válida, asignando valores extremos a los pistones y recalculando la orientación respecto al punto P.

Calcular el ángulo final del espejo y componer un texto resumen con esos resultados.

Actualizar la etiqueta de salida en la GUI con los valores calculados y llamar a dibujar_sistema(...) para refrescar el gráfico con la nueva configuración.

```python
import math
import numpy as np
from tkinter import messagebox
import tkinter as tk


def calcular_y_dibujar(self):  3 usages (1 dynamic)
    try:
        L = self.L.get()
        B = self.B.get()
        D = self.D.get()
        d_max = self.d_max.get()
        F = self.F.get()
        Px = self.Px_var.get()
        Py = self.Py_var.get()
    except tk.TclError:
        messagebox.showerror( title: "Error",  message: "Por favor, ingrese valores numéricos válidos.")
        return

    if Py < 0:
        messagebox.showerror( title: "Error",  message: "La coordenada Py debe ser mayor o igual a 0.")
        return

    if not (0 < L < B < d_max):
        messagebox.showerror( title: "Error",  message: "Los parámetros deben cumplir 0 < L < B < d_max.")
        return

    if Px == 0 and Py == 0:
        messagebox.showwarning( title: "Advertencia",  message: "Punto P(0, 0) no es válido. Se usará (1, 1) como referencia.")
        Px = 1
        Py = 1
```

```python
    encontrado = False
    for My in np.linspace(Py - L / 2, Py + L / 2,  num: 500):
        N_x = -Px
        N_y = My - Py
        if N_x == 0 and N_y == 0:
            continue

        D_x, D_y = -N_y, N_x
        norma = math.hypot( *coordinates: D_x, D_y)
        D_x /= norma
        D_y /= norma

        espejo_izq_x = (-L / 2) * D_x
        espejo_izq_y = My + (-L / 2) * D_y
        espejo_der_x = (L / 2) * D_x
        espejo_der_y = My + (L / 2) * D_y

        x1 = math.hypot( *coordinates: espejo_izq_x + B / 2, espejo_izq_y)
        x2 = math.hypot( *coordinates: espejo_der_x - B / 2, espejo_der_y)

        if espejo_izq_y < 0 or espejo_der_y < 0:
            continue

        if D <= x1 <= d_max and D <= x2 <= d_max:
            encontrado = True
            break
```

```
if not encontrado:
    messagebox.showwarning( title: "Advertencia",  message: "Punto fuera de alcance. Se graficará con pistones en sus límites.")

    x1 = d_max
    x2 = D

    theta = math.atan2(Py, Px)
    D_x = math.cos(theta)
    D_y = math.sin(theta)

    espejo_izq_x = -B / 2 + x2 * D_x
    espejo_izq_y = max(0, x2 * D_y)
    espejo_der_x = B / 2 + x1 * D_x
    espejo_der_y = max(0, x1 * D_y)

    centro_x = (espejo_izq_x + espejo_der_x) / 2
    centro_y = (espejo_izq_y + espejo_der_y) / 2

    espejo_izq_x = centro_x - (L / 2) * D_y
    espejo_izq_y = max(0, centro_y + (L / 2) * D_x)
    espejo_der_x = centro_x + (L / 2) * D_y
    espejo_der_y = max(0, centro_y - (L / 2) * D_x)

    angulo_espejo = math.degrees(theta)
else:
    angulo_espejo = math.degrees(math.atan2(D_y, D_x))

texto_salida = (
    f"\u00c1ngulo del espejo respecto a la horizontal: {angulo_espejo:.2f}°\n"
    f"Longitud del pistón izquierdo (x1): {x1:.2f}\n"
    f"Longitud del pistón derecho (x2): {x2:.2f}"
)
self.etiqueta_salida.config(text=texto_salida)

self.dibujar_sistema(x1, x2, Px, Py, L, B, angulo_espejo, espejo_izq_x, espejo_izq_y, espejo_der_x, espejo_der_y)
```

3.4.3 Ventana

El módulo ventana.py define la clase AplicacionTelescopio, que:
Inicializa la ventana de Tkinter (fondo, título, icono, fuente).
Crea y almacena las variables DoubleVar que representan los parámetros del telescopio.
“Inyecta” (ligándolas con __get__) las funciones externas de creación de widgets, cálculo y dibujo como métodos de la clase.
Llama a crear_widgets() para montar la interfaz y a calcular_y_dibujar() para mostrar el primer resultado y gráfico cuando la aplicación arranca.

```python
import tkinter as tk
from widgets import crear_widgets
from calculo import calcular_y_dibujar
from dibujo import dibujar_sistema
class AplicacionTelescopio:  5 usages
    def __init__(self, raiz):
        self.raiz = raiz
        self.raiz.configure(bg='lightgray')
        self.raiz.title("Telescopio James Webb")

        self.raiz = raiz
        # color de fondo da ventana thinker
        self.raiz.configure(bg='lightgray')
        self.raiz.title("Telescopio James Webb")

        # Definir constantes del sistema
        self.L = tk.DoubleVar(value=12.0)
        self.B = tk.DoubleVar(value=14.0)
        self.D = tk.DoubleVar(value=4.0)
        self.d_max = tk.DoubleVar(value=20.0)
        self.F = tk.DoubleVar(value=6.0)
        self.Px_var = tk.DoubleVar(value=8.0)
        self.Py_var = tk.DoubleVar(value=12.0)
        #  ↓↓↓ Aquí haces el "bind" de las funciones como métodos ↓↓↓
        self.crear_widgets      = crear_widgets.__get__(self, AplicacionTelescopio)
        self.calcular_y_dibujar = calcular_y_dibujar.__get__(self, AplicacionTelescopio)
        self.dibujar_sistema    = dibujar_sistema.__get__(self, AplicacionTelescopio)

        # Y **luego** sí llamas a crear_widgets y calcular_y_dibujar
        self.crear_widgets()
        self.calcular_y_dibujar()
```

3.4.4 Dibujo

El módulo dibujo.py define la función `dibujar_sistema(self, x1, x2, Px, Py, L, B, ángulo, espejo_izq_x, espejo_izq_y, espejo_der_x, espejo_der_y)`, que se encarga de:

1. Limpiar el eje (`ax.clear()`) y fijar el color de fondo.
2. Dibujar los elementos geométricos:

   Líneas de los pistones izquierdo y derecho.
   Línea gruesa del espejo.
   Punto P como un marcador.
   Línea punteada de la proyección perpendicular.
3. Etiquetar ejes (`Eje X`, `Eje Y`), título y leyenda.
4. Añadir la cuadrícula y mantener proporciones iguales (`axis('equal')`).
5. Ajustar los límites de los ejes para encuadrar todos los elementos.
6. Redibujar el canvas de Matplotlib en la ventana con `canvas.draw()`.

```python
def dibujar_sistema(self, x1, x2, Px, Py, L, B, angulo_espejo, espejo_izq_x, espejo_izq_y, espejo_der_x, espejo_der_y):
    self.ax.clear()
    # color de fondo del grafico
    self.ax.set_facecolor("lightyellow")
    A = (-B / 2, 0)
    B_punto = (B / 2, 0)
    # para cambiar el color de los pistones, se puede usar 'r-' para rojo, 'b-' para azul, etc.
    self.ax.plot([A[0], espejo_izq_x], [A[1], espejo_izq_y], 'g-', linewidth=3, label='Pistón Izquierdo')
    self.ax.plot([B_punto[0], espejo_der_x], [B_punto[1], espejo_der_y], 'g-', linewidth=3, label='Pistón Derecho')
    self.ax.plot([espejo_izq_x, espejo_der_x], [espejo_izq_y, espejo_der_y], 'c-', linewidth=4, label='Espejo')
    self.ax.plot(Px, Py, 'go', label='Punto P')

    mx = (espejo_izq_x + espejo_der_x) / 2
    my = (espejo_izq_y + espejo_der_y) / 2
    self.ax.plot([Px, mx], [Py, my], 'g--', linewidth=1, label='Proyección Perpendicular')

    self.ax.set_xlabel('Eje X')
    self.ax.set_ylabel('Eje Y')
    self.ax.set_title('Orientación del Espejo Secundario')
    self.ax.legend()
    self.ax.grid(True, which='both', linestyle='--', linewidth=0.5)
    self.ax.axis('equal')

    x_min = min(-B / 2, espejo_izq_x, Px) - 2
    x_max = max(B / 2, espejo_der_x, Px) + 2
    y_min = 0
    y_max = max(espejo_izq_y, espejo_der_y, Py) + 2

    self.ax.set_xlim(x_min, x_max)
    self.ax.set_ylim(y_min, y_max)
    self.canvas.draw()
```

3.4.5 Witgets.py

El módulo widgets.py define la función `crear_widgets(self)`, que arma toda la interfaz gráfica:

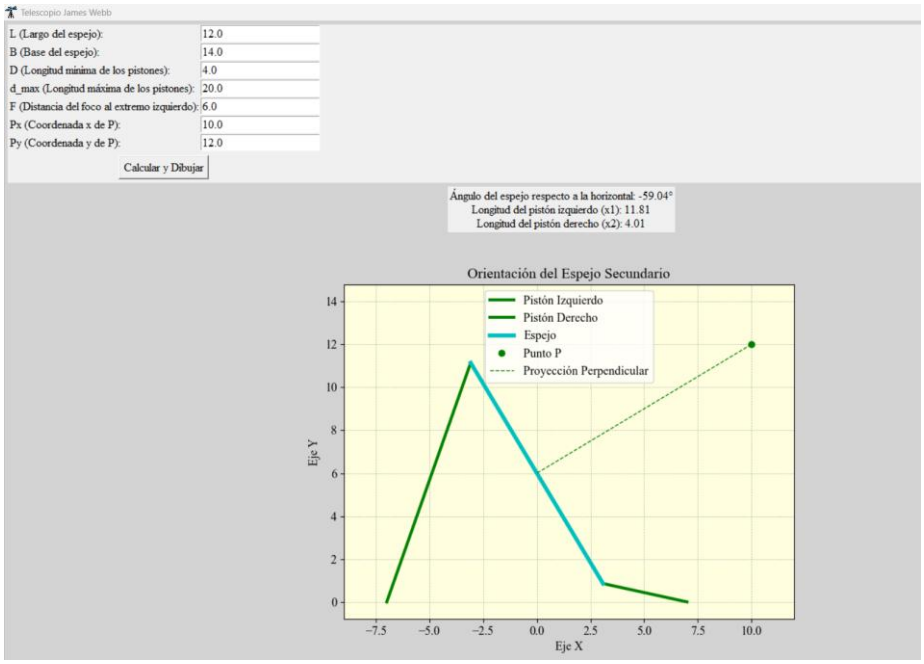
1. Crea un frame de entrada y coloca en él etiquetas (`Label`) y campos de texto (`Entry`) para cada parámetro (L, B, D, d_max, F, Px, Py).
2. Añade un botón “Calcular y Dibujar” ligado al método de cálculo.
3. Genera la **etiqueta de salida** (`self.etiqueta_salida`) donde se mostrarán los resultados numéricos.
4. Crea la figura de Matplotlib (con fondo y fuente ya configurados), el subplot y el canvas que se incrusta en la ventana para renderizar el gráfico.

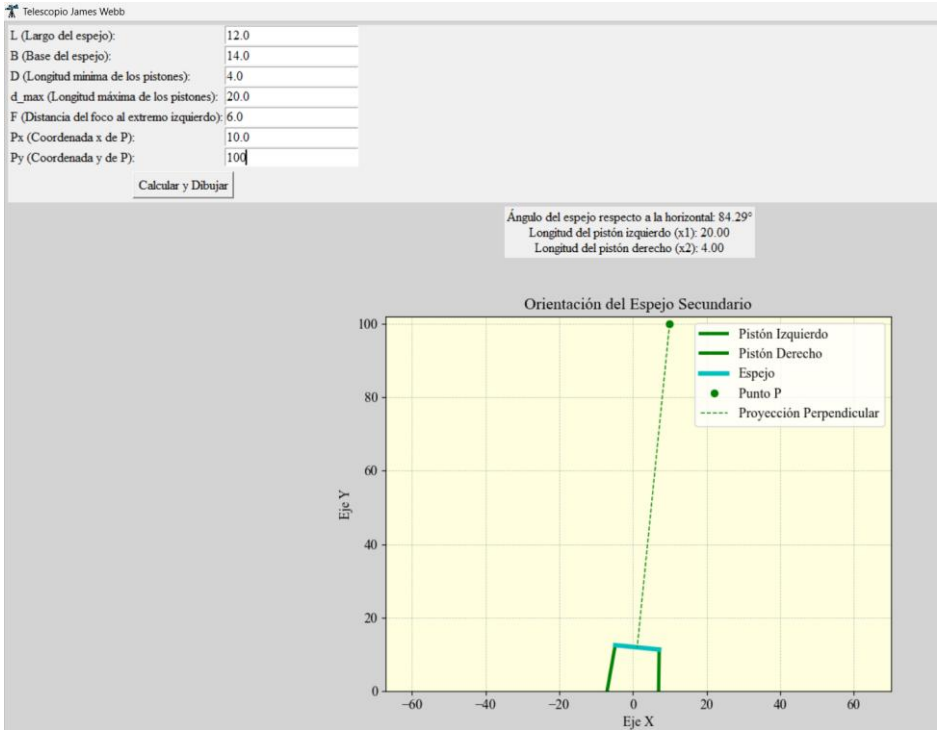
```python
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt

plt.rcParams['font.family'] = 'Times New Roman'
plt.rcParams['font.size']   = 12
def crear_widgets(self):  2 usages
    marco_entrada = tk.Frame(self.raiz)
    marco_entrada.pack(side=tk.TOP, fill=tk.X, padx=5, pady=5)

    etiquetas = [
        ("L (Largo del espejo):", self.L),
        ("B (Base del espejo):", self.B),
        ("D (Longitud mínima de los pistones):", self.D),
        ("d_max (Longitud máxima de los pistones):", self.d_max),
        ("F (Distancia del foco al extremo izquierdo):", self.F),
        ("Px (Coordenada x de P):", self.Px_var),
        ("Py (Coordenada y de P):", self.Py_var)
    ]

    for i, (label, var) in enumerate(etiquetas):
        tk.Label(marco_entrada, text=label).grid(row=i, column=0, sticky=tk.W)
        tk.Entry(marco_entrada, textvariable=var).grid(row=i, column=1)

    tk.Button(marco_entrada, text="Calcular y Dibujar", command=self.calcular_y_dibujar).grid(row=7, column=0,
                                                                                              columnspan=2, pady=5)

    self.etiqueta_salida = tk.Label(self.raiz, text="", font=('Times New Roman', 12))
    self.etiqueta_salida.pack()
    # cambio de color de fondo a 'x' para mejor contraste
    self.figura = plt.Figure(figsize=(8, 8), facecolor='lightgray')
    self.ax = self.figura.add_subplot(111)
    self.canvas = FigureCanvasTkAgg(self.figura, master=self.raiz)
    self.canvas.get_tk_widget().pack()
```

```python
import tkinter as tk
from widgets import crear_widgets
from calculo import calcular_y_dibujar
from dibujo import dibujar_sistema
class AplicacionTelescopio:  5 usages
    def __init__(self, raiz):
        self.raiz = raiz
        self.raiz.configure(bg='lightgray')
        self.raiz.title("Telescopio James Webb")

        self.raiz = raiz
        # color de fondo da ventana thinker
        self.raiz.configure(bg='lightgray')
        self.raiz.title("Telescopio James Webb")

        # Definir constantes del sistema
        self.L = tk.DoubleVar(value=12.0)
        self.B = tk.DoubleVar(value=14.0)
        self.D = tk.DoubleVar(value=4.0)
        self.d_max = tk.DoubleVar(value=20.0)
        self.F = tk.DoubleVar(value=6.0)
        self.Px_var = tk.DoubleVar(value=8.0)
        self.Py_var = tk.DoubleVar(value=12.0)
        #  ↓↓↓ Aquí haces el "bind" de las funciones como métodos ↓↓↓
        self.crear_widgets      = crear_widgets.__get__(self, AplicacionTelescopio)
        self.calcular_y_dibujar = calcular_y_dibujar.__get__(self, AplicacionTelescopio)
        self.dibujar_sistema    = dibujar_sistema.__get__(self, AplicacionTelescopio)

        # Y **luego** sí llamas a crear_widgets y calcular_y_dibujar
        self.crear_widgets()
        self.calcular_y_dibujar()
```

4. Resultados y conclusiones

4.1 Resultados





4.2 Conclusiones

Se realizo un modelo matemático o que describe el funcionamiento del sistema de orientación basado en los pistones lineales permitiendo calcular la orientación y posición del espejo en función del punto de interés teniendo una alineación precisa del eje óptico, implementado restricciones geométricas y físicas permitiendo tener un correcto funcionamiento.

La simulación grafica facilita de manera intuitiva la comprensión del sistema del telescopio indicando las posiciones de los pistones y la posición del espejo