```python
import tkinter as tk
from tkinter import ttk, scrolledtext, messagebox
from math import *
import sys

# Intentamos importar matplotlib para la gráfica
HAS_MPL = True
try:
    from matplotlib.figure import Figure
    from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
except ImportError:
    HAS_MPL = False


class HeunSolver:
    """
    Resuelve y' = f(x, y) con el método de Heun (Euler mejorado).
    """
    def __init__(self, f, verbose=False):
        self.f = f
        self.verbose = verbose

    def heun(self, x0, y0, h, n_pasos):
        """
        x0: valor inicial de x
        y0: condición inicial y(x0)
        h: tamaño de paso
        n_pasos: cantidad de pasos
        Retorna: listas xs, ys con los valores aproximados.
        """
        f = self.f
        xs = [x0]
        ys = [y0]

        x = x0
        y = y0

        for i in range(1, n_pasos + 1):
            k1 = f(x, y)
            y_pred = y + h * k1          # Predicción (Euler)
            k2 = f(x + h, y_pred)        # Pendiente corregida
            y = y + (h / 2) * (k1 + k2) # Fórmula de Heun
            x = x + h
```

```
            xs.append(x)
            ys.append(y)

            if self.verbose:
                print(f"Paso {i}: x = {x}, y = {y}, k1 = {k1}, k2 =
{k2}")

        return xs, ys


# Algunas EDO de ejemplo y' = f(x, y)
def edo1(x, y):
    # y' = x + y
    return x + y

def edo2(x, y):
    # y' = y - x**2 + 1
    return y - x**2 + 1

def edo3(x, y):
    # y' = x * exp(-x) - y
    return x * exp(-x) - y

def edo4(x, y):
    # y' = y * sin(x)
    return y * sin(x)


class InterfazHeun:
    def __init__(self):
        self.ventana = tk.Tk()
        self.ventana.title("Método de Heun para EDOs")
        self.ventana.geometry("1000x650")
        self.ventana.configure(bg="#e8e9ff")  # lavanda suave

        # ===== Estilos ttk =====
        style = ttk.Style()
        try:
            style.theme_use("clam")
        except:
            pass
```

```
        style.configure("TFrame", background="#e8e9ff")
        style.configure("Card.TFrame", background="white",
relief="raised", borderwidth=2)
        style.configure("Header.TLabel", font=("Segoe UI", 16, "bold"),
background="#4b5bdc", foreground="white")
        style.configure("SubHeader.TLabel", font=("Segoe UI", 11,
"bold"), background="white")
        style.configure("TButton", font=("Segoe UI", 10, "bold"))
        style.configure("Treeview.Heading", font=("Segoe UI", 10,
"bold"))

        # Diccionario de funciones: nombre -> función f(x, y)
        self.funciones = {
            "y' = x + y": edo1,
            "y' = y - x² + 1": edo2,
            "y' = x·e^(-x) - y": edo3,
            "y' = y·sin(x)": edo4,
            "Personalizada: f(x,y) =": None
        }

        self.crear_interfaz()

    # ================== UI ==================
    def crear_interfaz(self):
        self.ventana.grid_rowconfigure(1, weight=1)
        self.ventana.grid_columnconfigure(0, weight=0)
        self.ventana.grid_columnconfigure(1, weight=1)

        # ===== Encabezado =====
        frame_header = ttk.Frame(self.ventana, style="TFrame")
        frame_header.grid(row=0, column=0, columnspan=2, sticky="ew")
        frame_header.grid_columnconfigure(0, weight=1)

        header_label = ttk.Label(
            frame_header,
            text="Método de Heun para ecuaciones diferenciales",
            style="Header.TLabel",
            anchor="center"
        )
        header_label.grid(row=0, column=0, sticky="ew", padx=0, pady=5)

        # ===== Panel izquierdo: parámetros =====
        frame_left = ttk.Frame(self.ventana, style="TFrame")
```

```
        frame_left.grid(row=1, column=0, sticky="nsw", padx=10,
pady=10)
        frame_left.grid_rowconfigure(1, weight=1)

        card_param = ttk.Frame(frame_left, style="Card.TFrame")
        card_param.grid(row=0, column=0, sticky="n", padx=5, pady=5)

        ttk.Label(card_param, text="Configuración",
style="SubHeader.TLabel").grid(
            row=0, column=0, columnspan=2, sticky="w", padx=10,
pady=(10, 5)
        )

        # Selección de EDO
        ttk.Label(card_param, text="EDO (y' = f(x,y)):",
background="white").grid(
            row=1, column=0, sticky="w", padx=10, pady=2
        )
        self.dropdown_funcion = ttk.Combobox(
            card_param,
            values=list(self.funciones.keys()),
            state="readonly",
            width=30
        )
        self.dropdown_funcion.grid(row=1, column=1, sticky="w",
padx=10, pady=2)
        self.dropdown_funcion.current(0)
        self.dropdown_funcion.bind("<<ComboboxSelected>>",
self.on_cambio_funcion)

        # Campo para función personalizada
        ttk.Label(card_param, text="f(x,y) personalizada:",
background="white").grid(
            row=2, column=0, sticky="w", padx=10, pady=2
        )
        self.entry_func_personal = ttk.Entry(card_param, width=32)
        self.entry_func_personal.grid(row=2, column=1, sticky="w",
padx=10, pady=2)
        self.entry_func_personal.insert(0, "x + y")  # ejemplo
        self.entry_func_personal.configure(state="disabled")

        # Parámetros numéricos
```

```
        ttk.Label(card_param, text="x₀:",
background="white").grid(row=3, column=0, sticky="e", padx=10, pady=2)
        self.entry_x0 = ttk.Entry(card_param, width=12)
        self.entry_x0.grid(row=3, column=1, sticky="w", padx=10,
pady=2)
        self.entry_x0.insert(0, "0.0")

        ttk.Label(card_param, text="y₀:",
background="white").grid(row=4, column=0, sticky="e", padx=10, pady=2)
        self.entry_y0 = ttk.Entry(card_param, width=12)
        self.entry_y0.grid(row=4, column=1, sticky="w", padx=10,
pady=2)
        self.entry_y0.insert(0, "1.0")

        ttk.Label(card_param, text="h (paso):",
background="white").grid(row=5, column=0, sticky="e", padx=10, pady=2)
        self.entry_h = ttk.Entry(card_param, width=12)
        self.entry_h.grid(row=5, column=1, sticky="w", padx=10, pady=2)
        self.entry_h.insert(0, "0.1")

        ttk.Label(card_param, text="N (n° pasos):",
background="white").grid(row=6, column=0, sticky="e", padx=10, pady=2)
        self.entry_n = ttk.Entry(card_param, width=12)
        self.entry_n.grid(row=6, column=1, sticky="w", padx=10, pady=2)
        self.entry_n.insert(0, "10")

        # Checkbox detalle
        self.var_detalle = tk.BooleanVar(value=False)
        chk_detalle = ttk.Checkbutton(
            card_param,
            text="Mostrar detalles adicionales en texto",
            variable=self.var_detalle
        )
        chk_detalle.grid(row=7, column=0, columnspan=2, sticky="w",
padx=10, pady=(5, 5))

        # Botones
        frame_botones = ttk.Frame(card_param, style="Card.TFrame")
        frame_botones.grid(row=8, column=0, columnspan=2, pady=(10,
10))

        btn_calcular = ttk.Button(frame_botones, text="Ejecutar Heun",
command=self.ejecutar_heun)
```

```
        btn_calcular.grid(row=0, column=0, padx=5, pady=5)

        btn_limpiar = ttk.Button(frame_botones, text="Limpiar salida",
command=self.limpiar_resultados)
        btn_limpiar.grid(row=0, column=1, padx=5, pady=5)

        btn_salir = ttk.Button(frame_botones, text="Salir",
command=self.ventana.destroy)
        btn_salir.grid(row=0, column=2, padx=5, pady=5)

        # Tarjeta de ayuda
        card_info = ttk.Frame(frame_left, style="Card.TFrame")
        card_info.grid(row=1, column=0, sticky="n", padx=5, pady=5)

        ttk.Label(card_info, text="Ayuda rápida",
style="SubHeader.TLabel").grid(
            row=0, column=0, sticky="w", padx=10, pady=(10, 5)
        )

        text_help = (
            "• Elige una EDO de la lista o usa 'Personalizada'.\n"
            "• En función personalizada, escribe f(x,y), por ej:\n"
            "     y - x**2 + 1\n"
            "     x*exp(-x) - y\n"
            "• Puedes usar funciones de math: sin, cos, exp, log,
etc.\n"
            "• Ingresa x₀, y₀, h y número de pasos N.\n"
            "• Se mostrará la tabla y, si tienes matplotlib,\n"
            "  también la gráfica de la solución aproximada."
        )

        txt_help_widget = scrolledtext.ScrolledText(
            card_info, width=28, height=10, wrap=tk.WORD, font=("Segoe
UI", 9)
        )
        txt_help_widget.grid(row=1, column=0, padx=10, pady=(0, 10))
        txt_help_widget.insert("1.0", text_help)
        txt_help_widget.configure(state="disabled")

        # ===== Panel derecho: tabla + gráfica =====
        frame_right = ttk.Frame(self.ventana, style="TFrame")
        frame_right.grid(row=1, column=1, sticky="nsew", padx=10,
pady=10)
```

```
        frame_right.grid_rowconfigure(0, weight=1)
        frame_right.grid_columnconfigure(0, weight=1)

        self.notebook = ttk.Notebook(frame_right)
        self.notebook.grid(row=0, column=0, sticky="nsew")

        # --- Tab de tabla ---
        self.tab_tabla = ttk.Frame(self.notebook, style="TFrame")
        self.notebook.add(self.tab_tabla, text="Tabla de valores")

        self.tree = ttk.Treeview(
            self.tab_tabla,
            columns=("iter", "x", "y"),
            show="headings",
            height=18
        )
        self.tree.heading("iter", text="Iteración")
        self.tree.heading("x", text="x_n")
        self.tree.heading("y", text="y_n (Heun)")

        self.tree.column("iter", width=80, anchor="center")
        self.tree.column("x", width=120, anchor="center")
        self.tree.column("y", width=180, anchor="center")

        self.tree.grid(row=0, column=0, sticky="nsew", padx=5, pady=5)

        scroll_y = ttk.Scrollbar(self.tab_tabla, orient="vertical",
command=self.tree.yview)
        self.tree.configure(yscroll=scroll_y.set)
        scroll_y.grid(row=0, column=1, sticky="ns")

        self.tab_tabla.grid_rowconfigure(0, weight=1)
        self.tab_tabla.grid_columnconfigure(0, weight=1)

        # Zona de texto (detalles)
        self.texto_detalles = scrolledtext.ScrolledText(
            self.tab_tabla, height=8, wrap=tk.WORD, font=("Consolas",
9)
        )
        self.texto_detalles.grid(row=1, column=0, columnspan=2,
sticky="ew", padx=5, pady=(0, 5))

        # --- Tab de gráfica ---
```

```
        self.tab_grafica = ttk.Frame(self.notebook, style="TFrame")
        self.notebook.add(self.tab_grafica, text="Gráfica")

        self.canvas_plot = None
        if HAS_MPL:
            self.fig = Figure(figsize=(5, 4), dpi=100)
            self.ax = self.fig.add_subplot(111)
            self.ax.set_title("Solución aproximada con Heun")
            self.ax.set_xlabel("x")
            self.ax.set_ylabel("y")
            self.ax.grid(True)

            self.canvas_plot = FigureCanvasTkAgg(self.fig,
master=self.tab_grafica)
            self.canvas_plot.get_tk_widget().pack(fill="both",
expand=True, padx=5, pady=5)
        else:
            lbl = ttk.Label(
                self.tab_grafica,
                text="Matplotlib no está instalado.\nNo se puede
mostrar la gráfica.",
                anchor="center"
            )
            lbl.pack(expand=True)

    # ================== Lógica ==================
    def on_cambio_funcion(self, event=None):
        nombre = self.dropdown_funcion.get()
        if "Personalizada" in nombre:
            self.entry_func_personal.configure(state="normal")
        else:
            self.entry_func_personal.configure(state="disabled")

    def construir_funcion(self):
        nombre_funcion = self.dropdown_funcion.get()
        if "Personalizada" not in nombre_funcion:
            return self.funciones[nombre_funcion], nombre_funcion

        expr = self.entry_func_personal.get().strip()
        if not expr:
            raise ValueError("Debe ingresar una expresión para
f(x,y).")
```

```
        # Entorno seguro para eval
        import math
        allowed_names = {k: getattr(math, k) for k in dir(math) if not
k.startswith("_")}
        # puedes usar: sin, cos, exp, log, etc.

        def f(x, y, expr=expr, allowed=allowed_names):
            local_dict = {"x": x, "y": y}
            return eval(expr, {"__builtins__": {}}, {**allowed,
**local_dict})

        return f, f"f(x,y) = {expr}"

    def ejecutar_heun(self):
        try:
            f, nombre_mostrado = self.construir_funcion()

            x0 = float(self.entry_x0.get())
            y0 = float(self.entry_y0.get())
            h = float(self.entry_h.get())
            n = int(self.entry_n.get())

            if h <= 0:
                raise ValueError("El tamaño de paso h debe ser
positivo.")
            if n <= 0:
                raise ValueError("El número de pasos N debe ser mayor
que 0.")

            solver = HeunSolver(f, verbose=False)
            xs, ys = solver.heun(x0, y0, h, n)

            self.llenar_tabla(xs, ys)
            self.mostrar_detalles(nombre_mostrado, x0, y0, h, n, xs,
ys)
            self.actualizar_grafica(xs, ys)

        except Exception as e:
            messagebox.showerror("Error", f"Error en el cálculo:
{str(e)}")

    def llenar_tabla(self, xs, ys):
        # Limpiar tabla
```

```
        for item in self.tree.get_children():
            self.tree.delete(item)

        for i, (x, y) in enumerate(zip(xs, ys)):
            self.tree.insert("", "end", values=(i, f"{x:.6f}",
f"{y:.10f}"))

    def mostrar_detalles(self, nombre_funcion, x0, y0, h, n, xs, ys):
        self.texto_detalles.delete("1.0", tk.END)

        texto = ""
        texto += "=" * 60 + "\n"
        texto += f"EDO: {nombre_funcion}\n"
        texto += f"Condición inicial: y({x0}) = {y0}\n"
        texto += f"h = {h}, N = {n}\n"
        texto += f"Intervalo aproximado: [{x0}, {xs[-1]}]\n"
        texto += "=" * 60 + "\n"
        texto += f"Valor final aproximado: y({xs[-1]:.6f}) ≈
{ys[-1]:.10f}\n"

        if self.var_detalle.get():
            texto += "\nDetalles adicionales:\n"
            texto += "Se usó el método de Heun (Euler mejorado):\n"
            texto += "  k1 = f(x_n, y_n)\n"
            texto += "  y* = y_n + h·k1\n"
            texto += "  k2 = f(x_n + h, y*)\n"
            texto += "  y_{n+1} = y_n + (h/2)·(k1 + k2)\n"

        self.texto_detalles.insert("1.0", texto)

    def actualizar_grafica(self, xs, ys):
        if not HAS_MPL or self.canvas_plot is None:
            return

        self.ax.clear()
        self.ax.grid(True)
        self.ax.set_title("Solución aproximada con Heun")
        self.ax.set_xlabel("x")
        self.ax.set_ylabel("y")
        self.ax.plot(xs, ys, marker="o")
        self.canvas_plot.draw()

    def limpiar_resultados(self):
```

```
        # Tabla
        for item in self.tree.get_children():
            self.tree.delete(item)
        # Texto
        self.texto_detalles.delete("1.0", tk.END)
        # Gráfica
        if HAS_MPL and self.canvas_plot is not None:
            self.ax.clear()
            self.ax.grid(True)
            self.ax.set_title("Solución aproximada con Heun")
            self.ax.set_xlabel("x")
            self.ax.set_ylabel("y")
            self.canvas_plot.draw()

    def ejecutar(self):
        self.ventana.mainloop()


if __name__ == "__main__":
    app = InterfazHeun()
    app.ejecutar()
```