Informe de Código de Proyecto de Programación no Lineal (Noviembre de 2025)

Jose M. Benavides,Emmanuel Martinez

Resumen - El Big Data se ha consolidado como una de las tecnologías más influyentes en el paso de la era análoga a la digital, impulsando la transformación en distintos sectores, como la salud, agrícola y finanzas. Este artículo hace una revisión bibliográfica desde sus orígenes, evolución, aplicaciones y desafíos, destacando su papel como herramienta estratégica para la toma de decisiones y optimización de procesos. Se examinan las tendencias actuales y futuras, como blockchain, inteligencia artificial, computación en la nube, seguridad de los datos. Revisa brevemente la implicación ética de la masificación de los datos de personas, exponiendo sus limitaciones y pensando en futuras legislaciones.

Índice de Términos – Big data, análisis de datos, inteligencia artificial, blockchain, privacidad, ética.

I. INTRODUCCIÓN

Este trabajo presenta la implementación de la función de Rastrigin, su gradiente analítico y un algoritmo de descenso por gradiente para resolver el problema de encontrar el mínimo de dicha función. La función se usa ampliamente para evaluar la calidad de algoritmos de optimización debido a su complejidad y alto número de mínimos locales. El objetivo del informe es mostrar la construcción matemática, el procedimiento aplicado y los resultados obtenidos con un método determinista basado en gradientes.

II. MARCO TEÓRICO

A. Optimización no lineal

La optimización no lineal estudia problemas donde la función objetivo o las restricciones presentan términos no lineales. De manera general, un problema de optimización no lineal se formula como la mostrada en la figura 1.

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$g_i(x) \geq 0, \quad i = 1, \ldots, m$$

$$h_j(x) = 0, \quad j = 1, \ldots, p$$

Figura 1. Ejemplo de función no lineal sujeta a restricciones.

Donde f(x) es la función objetivo, g(x) son restricciones de desigualdad y h(x) restricciones de igualdad. Este tipo de problemas puede presentar múltiples soluciones locales, puntos de silla y regiones no convexas. Por ello se emplean métodos analíticos y numéricos complementarios. El software desarrollado implementa estos enfoques.

B. Cálculo diferencial y gradiente

El método del gradiente, también conocido como gradiente descendente, es un algoritmo iterativo usado para resolver problemas de optimización no lineal sin restricciones. El método se fundamenta en la observación de que el gradiente $\nabla f(x)$ señala la dirección de máximo crecimiento de la función. Por lo tanto:

- Para **minimizar**, se avanza en la dirección opuesta al gradiente.
- Para **maximizar**, se avanza en la misma dirección del gradiente.

La figura 2 muestra como se escribe en caso de maximización.

$$x_{k+1} = x_k + \alpha_k \nabla f(x_k)$$

Figura 2. Regla general de maximización en caso de gradiente.

C. Matriz hessiana y puntos críticos

Una vez hallados los puntos críticos, la naturaleza del punto depende de la Matriz Hessiana.

$$H_f(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Figura 3. Matriz hessiana.

La clasificación se realiza según los valores propios de la Hessiana:

Si todos los valores son positivos, todos negativos o diferentes signos será mínimo local, máximo local o punto de silla respectivamente.

D. Método de Lagrange

Para restricciones de igualdad de la forma h(x) = 0 se emplea la técnica de multiplicadores de Lagrange. Se define el Lagrangiano:

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda \left(h(x, y)\right)$$

Figura 4. Lagrangiano definido.

La función que acompaña al Lambda en la imagen sería la restricción dada por el ejercicio en cuestión de la función objetivo, esto para permitir un desarrollo algebráico posterior a este paso.

E. CONDICIONES DE KARUSH-KUHN-TUCKER

Las Condiciones KKT son una generalización del método de Lagrange para manejar problemas de optimización con restricciones de desigualdad **g >= 0**, además de las de igualdad. Son las condiciones necesarias (y suficientes bajo convexidad) para que una solución x sea un óptimo local. Para restricciones de desigualdad no lineal, el estándar es aplicar las condiciones KKT, que generalizan Lagrange e incorporan complementariedad.

1. Estacionariedad

$$\nabla f(\mathbf{x}^*) + \sum_i \mu_i \nabla g_i(\mathbf{x}^*) = \mathbf{0}$$

2. Factibilidad Primal

$$g_i(\mathbf{x}^*) \leq 0 \quad \text{para todo } i$$

3. Factibilidad Dual

$$\mu_i \geq 0 \quad \text{para todo } i$$

4. Holgura Complementaria

$$\mu_i g_i(\mathbf{x}^*) = 0 \quad \text{para todo } i$$

Figura 5. Condiciones de KKT según el caso seleccionado.

F. ALGORITMO DE GRADIENTE

El método iterativo de **descenso por gradiente** es un algoritmo básico para optimización no lineal sin restricciones. Actualiza a la imagen, donde alfa es el tamaño de paso. Si se desea maximizar la función, se invierte el sentido del paso:

$$x_{k+1} = x_k - \alpha \nabla f(x_k)$$

Figura 6. Descenso de gradiente en minimización.

G. PROGRAMACIÓN CUADRÁTICA

La Programación Cuadrática (QP) se refiere a problemas de optimización donde la función objetivo es cuadrática y las restricciones son lineales.

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} + r$$

Figura 7. Función objetivo cuadrática.

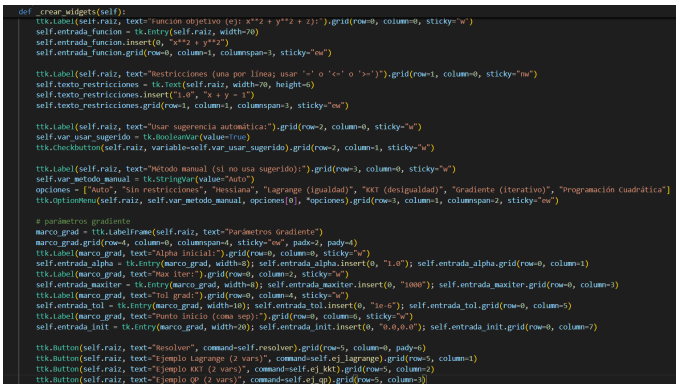
Donde X es el vector de variables, Q es la matriz Hessiana constante de f, c es el vector de coeficientes lineales, y r es una constante.

Convexidad: Un problema QP es convexo si y sólo si la matriz Q es semidefinida positiva. Esto es crucial porque, en problemas convexos, cualquier óptimo local es también un óptimo global.

Sistema KKT Lineal (Restricciones de Igualdad): Para un QP sujeto a restricciones de igualdad lineales Ax = d, las condiciones KKT se convierten en un sistema de ecuaciones lineales que se puede resolver directamente (sin iteraciones).

III. PROCEDIMIENTO

Se realiza un código desarrollado en lenguaje Python y el IDE Visual Studio Code, junto a librerías comunes en la matemática como symbols, sympy, solve, entre otros.



```
def _crear_widgets(self):
    ttk.Label(self.raiz, text="Función objetivo (e): x**2 + y**2 + z):").grid(row=0, column=0, sticky="w")
    self.entrada_funcion = tk.Entry(self.raiz, width=70)
    self.entrada_funcion.insert(0, "x**2 + y**2")
    self.entrada_funcion.grid(row=0, column=1, columnspan=3, sticky="ew")

    ttk.Label(self.raiz, text="Restricciones (una por línea; usar '=' o '<=' o '>='):").grid(row=1, column=0, sticky="nw")
    self.texto_restricciones = tk.Text(self.raiz, width=70, height=6)
    self.texto_restricciones.insert("1.0", "x + y = 1")
    self.texto_restricciones.grid(row=1, column=1, columnspan=3, sticky="ew")

    ttk.Label(self.raiz, text="Usar sugerencia automática:").grid(row=2, column=0, sticky="w")
    self.var_usar_sugerido = tk.BooleanVar(value=True)
    ttk.Checkbutton(self.raiz, variable=self.var_usar_sugerido).grid(row=2, column=1, sticky="w")

    ttk.Label(self.raiz, text="Método manual (si no usa sugerido):").grid(row=3, column=0, sticky="w")
    self.var_metodo_manual = tk.StringVar(value="Auto")
    opciones = ["Auto", "Sin restricciones", "Hessiana", "Lagrange (igualdad)", "KKT (desigualdad)", "Gradiente (iterativo)", "Programación Cuadrática"]
    ttk.OptionMenu(self.raiz, self.var_metodo_manual, opciones[0], *opciones).grid(row=3, column=1, columnspan=2, sticky="ew")

    # parámetros gradiente
    marco_grad = ttk.LabelFrame(self.raiz, text="Parámetros Gradiente")
    marco_grad.grid(row=4, column=0, columnspan=4, sticky="ew", padx=2, pady=4)
    ttk.Label(marco_grad, text="Alpha inicial:").grid(row=0, column=0, sticky="w")
    self.entrada_alpha = tk.Entry(marco_grad, width=8); self.entrada_alpha.insert(0, "1.0"); self.entrada_alpha.grid(row=0, column=1)
    ttk.Label(marco_grad, text="Max iter:").grid(row=0, column=2, sticky="w")
    self.entrada_maxiter = tk.Entry(marco_grad, width=8); self.entrada_maxiter.insert(0, "1000"); self.entrada_maxiter.grid(row=0, column=3)
    ttk.Label(marco_grad, text="Tol grad:").grid(row=0, column=4, sticky="w")
    self.entrada_tol = tk.Entry(marco_grad, width=10); self.entrada_tol.insert(0, "1e-6"); self.entrada_tol.grid(row=0, column=5)
    ttk.Label(marco_grad, text="Punto inicio (coma sep):").grid(row=0, column=6, sticky="w")
    self.entrada_init = tk.Entry(marco_grad, width=20); self.entrada_init.insert(0, "0.0,0.0"); self.entrada_init.grid(row=0, column=7)

    ttk.Button(self.raiz, text="Resolver", command=self.resolver).grid(row=5, column=0, pady=6)
    ttk.Button(self.raiz, text="Ejemplo Lagrange (2 vars)", command=self.ej_lagrange).grid(row=5, column=1)
    ttk.Button(self.raiz, text="Ejemplo KKT (2 vars)", command=self.ej_kkt).grid(row=5, column=2)
    ttk.Button(self.raiz, text="Ejemplo QP (2 vars)", command=self.ej_qp).grid(row=5, column=3)
```

Figura 8. Creación de interfaz gráfica de usuario.

```python
def _sugerir_metodo(self, expr_func, lineas_restr, lista_vars):

    nonpoly_funcs = any(expr_func.has(fn) for fn in (sin, cos, exp, tan, log))

    try:
        pol = expand(expr_func)
        terminos = pol.as_ordered_terms()
        deg = max([t.as_poly(*lista_vars).total_degree() for t in terminos]) if terminos else 0
        es_cuad = deg <= 2
    except Exception:
        es_cuad = False

    func_simple = es_funcion_simple(expr_func, lista_vars)
    any_ineq = any(('>=' in r or '<=' in r) for r in lineas_restr)
    any_eq = any('=' in r for r in lineas_restr)

    if not lineas_restr:
        if nonpoly_funcs:
            return "Hessiana"
        return "Sin restricciones" if es_cuad else "Hessiana"

    if any_eq and not any_ineq:
        if func_simple:
            return "Lagrange (igualdad)"
        if es_cuad:
            return "Programación Cuadrática"
        return "Lagrange (igualdad)"

    if any_ineq:
        return "KKT (desigualdad)"

    return "Programación Cuadrática" if es_cuad else "KKT (desigualdad)"
```

Figura 9. Validación de otras opciones del código, como acción de optimización de la función objetivo entre otros.

```python
def _resolver_sin_restricciones(self, f, lista_vars, log):
    log("Resolviendo ∇f = 0 simbólicamente.")
    gradientes = [diff(f, v) for v in lista_vars]
    for i, g in enumerate(gradientes):
        log(f"∂f/∂{lista_vars[i]} = {texto_ecuacion(g)}")
    soluciones = solve(gradientes, tuple(lista_vars), dict=True)
    if not soluciones:
        log("No se hallaron soluciones simbólicas para ∇f = 0.")
        return
    for s in soluciones:
        val_f = evaluar_numerico_seguro(f, s)
        if val_f is None:
            log(f"Solución: {s} -> f (no numérico): {texto_ecuacion(f.subs(s))}")
        else:
            log(f"Solución: {s} -> f = {val_f:.12g}")
```

Figura 10. Opción de resolución de función objetivo sin restricciones.

```python
def _resolver_hessiana(self, f, lista_vars, log):
    log("Cálculo de Hessiana y clasificación de puntos críticos.")
    grads = [diff(f, v) for v in lista_vars]
    H = Matrix([[diff(f, vi, vj) for vi in lista_vars] for vj in lista_vars])
    for i, g in enumerate(grads):
        log(f"∂f/∂{lista_vars[i]} = {texto_ecuacion(g)}")
    log("Matriz Hessiana simbólica (sin formato multilinea):")
    log(texto_ecuacion(H))
    try:
        soluciones = solve(grads, tuple(lista_vars), dict=True)
    except Exception:
        soluciones = []
    if not soluciones:
        soluciones = intentar_nsolve_vect(grads, lista_vars, log)
    if not soluciones:
        log("No se encontraron puntos críticos simbólicos.")
        return

    log("\n=== RESULTADO FINAL ===")
    for s in soluciones:
        H_evaluada = H.subs(s)
        log(f"\nPunto crítico: {s}")
        val_f = evaluar_numerico_seguro(f, s)
        if val_f is None:
            log(f"f (no numérico): {texto_ecuacion(f.subs(s))}")
        else:
            log(f"f = {val_f:.12g}")

        try:
            eigs = list(H_evaluada.eigenvals().keys())
            eigs_num = [complex(N(ev)) for ev in eigs]
            log(f"Valores propios H: {[float(ev.real) for ev in eigs_num]}")
            if all(ev.real > 1e-9 for ev in eigs_num):
                log("Tipo: MÍNIMO LOCAL (H positiva definida)")
            elif all(ev.real < -1e-9 for ev in eigs_num):
                log("Tipo: MÁXIMO LOCAL (H negativa definida)")
            else:
                log("Tipo: PUNTO DE SILLA o INDEFINIDO")
        except Exception:
            log("No fue posible calcular valores propios numéricos.")
```

Figura 11. Opción de resolución de función objetivo detectada como método hessiano.

```python
def _resolver_lagrange(self, f, lista_vars, igualdades, log):
    if not igualdades:
        raise ValueError("No hay igualdades para Lagrange.")
    log("Construyendo Lagrangiano con igualdades.")
    n_eq = len(igualdades)
    lam = symbols(f'lam0:{n_eq}')
    exprs_constr = [(eq.lhs - eq.rhs) for eq in igualdades]
    L = f - sum(lam[i] * exprs_constr[i] for i in range(n_eq))
    log("Lagrangiano (una sola línea, sin exponentes multilinea):")
    log("L = " + texto_ecuacion(L))
    sistema = [diff(L, v) for v in lista_vars] + [diff(L, lm) for lm in lam]
    log("Ecuaciones del sistema (gradientes y derivadas respecto multiplicadores):")
    for i, e in enumerate(sistema):
        log(f"{i+1}: {texto_ecuacion(e)}")
    incognitas = tuple(lista_vars) + tuple(lam)
    soluciones = solve(sistema, incognitas, dict=True)
    if not soluciones:
        log("No se hallaron soluciones simbólicas para el sistema de Lagrange.")
        return

    H_f = Matrix([[diff(f, vi, vj) for vi in lista_vars] for vj in lista_vars])
    A = Matrix([[diff(expr, v) for v in lista_vars] for expr in exprs_constr])

    log("\n=== RESULTADO FINAL ===")
    for s in soluciones:
        log(f"\nPunto: {s}")
        fval = evaluar_numerico_seguro(f, s)
        if fval is None:
            log(f"f (no numérico): {texto_ecuacion(f.subs(s))}")
        else:
            log(f"f = {fval:.12g}")

        # Clasificación con Hessiana bordeada: [0 A'; A H]
        try:
            H_eval = H_f.subs(s)
            A_eval = A.subs(s)

            n = len(lista_vars)
            m = len(igualdades)
            H_bordeada = Matrix.zeros(n + m, n + m)
            H_bordeada[m:, m:] = H_eval
            H_bordeada[:m, m:] = A_eval
            H_bordeada[m:, :m] = A_eval.T

            eigs_bordeada = list(H_bordeada.eigenvals().keys())
```

Figura 12. Opción de resolución de función objetivo

detectada como método de Lagrange.

```
def _resolver_kkt_conjuntos_activos(self, f, lista_vars, igualdades, desigualdades, log):
    log("Resolviendo condiciones KKT por enumeración de conjuntos activos (active-set).")
    g_exprs = []
    for r in desigualdades:
        g_exprs.append(self._parsear_inecuacion(r))
    log(f"Igualdades: {len(igualdades)}, Desigualdades procesadas: {len(g_exprs)}")

    max_activos = 3
    if len(g_exprs) > 10:
        raise ValueError("Demasiadas desigualdades para enumerar (límite práctico).")

    encontrado = False
    indices = list(range(len(g_exprs)))
    for r in range(0, min(len(g_exprs), max_activos) + 1):
        for subconjunto in itertools.combinations(indices, r):
            activos = [g_exprs[i] for i in subconjunto]
            lam_eq = symbols(f'lamE0:{len(igualdades)}')
            mu_act = symbols(f'muA0:{len(activos)}')
            Ls = f
            for i_eq, eq in enumerate(igualdades):
                Ls = Ls - lam_eq[i_eq] * (eq.lhs - eq.rhs)
            for j, g in enumerate(activos):
                Ls = Ls - mu_act[j] * g
            sistema = [diff(Ls, v) for v in lista_vars]
            for eq in igualdades:
                sistema.append(eq.lhs - eq.rhs)
            for g in activos:
                sistema.append(g)
            desconocidas = tuple(lista_vars) + tuple(lam_eq) + tuple(mu_act)
            try:
                soluciones = solve(sistema, desconocidas, dict=True)
            except Exception:
                soluciones = []
            if not soluciones:
                continue
            for s in soluciones:
                log(f"Solución cruda: {s}")
                # residuos de todas las desigualdades (g_exprs)
                residuos = []
                factible = True
                for g in g_exprs:
                    r = evaluar_numerico_seguro(g, s)
                    residuos.append(r)
                    if r is None or r < -1e-8:
                        factible = False
```

Figura 13. Opción de resolución de función objetivo detectada como método de condiciones de KKT.

```
def _resolver_gradiente(self, f, lista_vars, log):
    alpha0 = a_flotante_seguro(self.entrada_alpha.get(), 1.0)
    max_iter = int(a_flotante_seguro(self.entrada_maxiter.get(), 1000))
    tol = a_flotante_seguro(self.entrada_tol.get(), 1e-6)
    init_text = self.entrada_init.get().strip()
    try:
        init_vals = [float(v.strip()) for v in init_text.split(",") if v.strip()!='']
    except Exception:
        init_vals = []
    if len(init_vals) < len(lista_vars):
        init_vals = init_vals + [0.0] * (len(lista_vars) - len(init_vals))
    xk = np.array(init_vals[:len(lista_vars)], dtype=float)
    log(f"Gradiente numérico (Armijo). Inicio={formatear_vector(xk)}, alpha0={alpha0}, max_iter={max_iter}, tol={tol}")
    grad_sim = self._gradientes_simbolicos(f, lista_vars)
    f_num = crear_funcion_numerica(f, lista_vars)

    historia = []
    c = 1e-4
    rho = 0.5
    for k in range(1, max_iter+1):
        gval = self._evaluar_grad(grad_sim, xk, lista_vars)
        ng = np.linalg.norm(gval)
        fv = f_num(xk)
        historia.append((k, xk.copy(), fv, ng))
        if ng < tol:
            log(f"Convergencia alcanzada: ||grad|| = {ng:.6g} < tol")
            break
        pk = -gval
        alpha = alpha0
        while alpha > 1e-12:
            xprueba = xk + alpha * pk
            fprueba = f_num(xprueba)
            if fprueba <= fv + c * alpha * np.dot(gval, pk):
                break
            alpha *= rho
        xk = xk + alpha * pk
        if k % 50 == 0:
            log(f"Iter {k}: x={formatear_vector(xk)}, f={fv:.6g}, ||g||={ng:.3g}, alpha={alpha:.3g}")
    for it, pt, fv, ng in historia[-20:]:
        log(f"Iter {it}: x={formatear_vector(pt)}, f={fv:.9g}, ||g||={ng:.3g}")
    log(f"Resultado final: x={formatear_vector(xk)}, f={f_num(xk):.9g}")
```

Figura 14. Opción de resolución de función objetivo detectada como método de condiciones de gradiente.

```
def _decidir_sin_restricciones(self, f, lista_vars, log):
    """Decide entre resolución simbólica, Hessiana"""
    log("Caso sin restricciones: intento resolver simbólicamente ∇f = 0 y analizar Hessiana.")
    grads = [diff(f, v) for v in lista_vars]
    for i, g in enumerate(grads):
        log(f"∂f/∂{lista_vars[i]} = {texto_ecuacion(g)}")
    try:
        soluciones = solve(grads, tuple(lista_vars), dict=True)
    except Exception:
        soluciones = []
    if not soluciones:
        soluciones = intentar_nsolve_vect(grads, lista_vars, log)

    if not soluciones:
        log("No se hallaron soluciones simbólicas para ∇f = 0. Uso método numérico (gradiente).")
        return self._resolver_gradiente(f, lista_vars, log)

    # si hay soluciones, evaluar Hessiana en cada una y decidir
    H = Matrix([[diff(f, vi, vj) for vi in lista_vars] for vj in lista_vars])
    log("Hessiana simbólica (una línea): " + texto_ecuacion(H))
```

Figura 15. Función que refiere al usuario un método adecuado para la solución del problema entre Hessiano o por álgebra. Luego comprueba la viabilidad de gradiente.

IV. RESULTADOS

Se ingresa la función objetivo $(X-2)^2 + (Y+1)^2$ sin restricciones, para comprobar qué método selecciona el código.

```
[Paso 1] Variables detectadas: x, y
[Paso 2] Función: (x - 2)**2 + (y + 1)**2
[Paso 3] Igualdades: 0, Desigualdades: 0
Sugerido: Sin restricciones
[Paso 4] Método seleccionado: Sin restricciones
[Paso 5] Caso sin restricciones: intento resolver simbólicamente ∇f = 0 y analizar Hessiana.
[Paso 6] ∂f/∂x = 2*x - 4
[Paso 7] ∂f/∂y = 2*y + 2
[Paso 8] Hessiana simbólica (una línea): Matrix([[2, 0], [0, 2]])
[Paso 9] Punto crítico candidato: {x: 2, y: -1}
[Paso 10] Valores propios (num aproximados): [2.0]
[Paso 11] Clasificación en este punto: mínimo local (H positiva definida).
[Paso 12]
=== RESULTADO FINAL ===
[Paso 13]
Punto: {x: 2, y: -1}
[Paso 14] f = 0
[Paso 15] Tipo: MÍNIMO LOCAL (H positiva definida)
```

Figura 16. Solución de la función objetivo bajo el método sin restricciones.

Se observa la elección de método sin restricciones.

Para el segundo uso, se ingresa la función objetivo $X^2 + Y^2$, y restricciones $X + Y = 1$. Se observa que se selecciona el método de Lagrange para su solución, la que es descrita en un paso a paso, como se espera.

```
[Paso 1] Variables detectadas: x, y
[Paso 2] Función: x**2 + y**2
[Paso 3] Igualdades: 1, Desigualdades: 0
Sugerido: Lagrange (igualdad)
[Paso 4] Método seleccionado: Lagrange (igualdad)
[Paso 5] Construyendo Lagrangiano con igualdades.
[Paso 6] Lagrangiano (una sola línea, sin exponentes multilinea):
[Paso 7] L = -lam0*(x + y - 1) + x**2 + y**2
[Paso 8] Ecuaciones del sistema (gradientes y derivadas respecto multiplicadores):
[Paso 9] 1: -lam0 + 2*x
[Paso 10] 2: -lam0 + 2*y
[Paso 11] 3: -x - y + 1
[Paso 12]
=== RESULTADO FINAL ===
[Paso 13]
Punto: {lam0: 1, x: 1/2, y: 1/2}
[Paso 14] f = 0.5
[Paso 15] Valores propios Hessiana bordeada: [-0.732051, 2.0, 2.732051]
[Paso 16] Tipo: MÍNIMO LOCAL condicionado
```

Figura 17. Solución de la función objetivo bajo el método de Lagrange.

Para el tercer uso, se ingresa la función objetivo X^2, con restricciones $X >= 1$. Se observa que se selecciona el método de KKT para su solución, la que es descrita en un paso a paso, como se espera.

```
[Paso 1] Variables detectadas: x, y
[Paso 2] Función: x**2 + y**2
[Paso 3] Igualdades: 1, Desigualdades: 0
Sugerido: Lagrange (igualdad)
[Paso 4] Método seleccionado: Lagrange (igualdad)
[Paso 5] Construyendo Lagrangiano con igualdades.
[Paso 6] Lagrangiano (una sola línea, sin exponentes multilínea):
[Paso 7] L = -lam0*(x + y - 1) + x**2 + y**2
[Paso 8] Ecuaciones del sistema (gradientes y derivadas respecto multiplicadores):
[Paso 9] 1: -lam0 + 2*x
[Paso 10] 2: -lam0 + 2*y
[Paso 11] 3: -x - y + 1
[Paso 12] Solución candidata: {lam0: 1, x: 1/2, y: 1/2} -> f = 0.5
[Paso 13] Nota: para clasificación avanzada use evaluación de Hessiana restringida (bordeada).
```

Figura 18. Solución de la función objetivo bajo el método de KKT de desigualdades.

Para el cuarto uso, se ingresa la función objetivo $X^2 + Y^2 + sen(x)$, sin restricciones. Se observa que se selecciona el método Hessiano su solución, la que es descrita en un paso a paso, como se espera.

```
[Paso 1] Variables detectadas: x, y
[Paso 2] Función: x**2 + y**2 + sin(x)
[Paso 3] Igualdades: 0, Desigualdades: 0
Sugerido: Hessiana
[Paso 4] Método seleccionado: Hessiana
[Paso 5] Cálculo de Hessiana y clasificación de puntos críticos.
[Paso 6] ∂f/∂x = 2*x + cos(x)
[Paso 7] ∂f/∂y = 2*y
[Paso 8] Matriz Hessiana simbólica (sin formato multilínea):
[Paso 9] Matrix([[2 - sin(x), 0], [0, 2]])
[Paso 10]
=== RESULTADO FINAL ===
[Paso 11]
Punto crítico: {x: -0.450183611294874, y: 0}
[Paso 12] f = -0.232465575158
[Paso 13] Valores propios H: [2.43513085903671, 2.0]
[Paso 14] Tipo: MÍNIMO LOCAL (H positiva definida)
[Paso 15]
Punto crítico: {x: -0.450183611294874, y: 0}
[Paso 16] f = -0.232465575158
[Paso 17] Valores propios H: [2.43513085903671, 2.0]
[Paso 18] Tipo: MÍNIMO LOCAL (H positiva definida)
[Paso 19]
Punto crítico: {x: -0.450183611294874, y: 0}
[Paso 20] f = -0.232465575158
[Paso 21] Valores propios H: [2.4351308590367093, 2.0]
[Paso 22] Tipo: MÍNIMO LOCAL (H positiva definida)
[Paso 23]
Punto crítico: {x: -0.450183611294874, y: 0}
[Paso 24] f = -0.232465575158
[Paso 25] Valores propios H: [2.4351308590367093, 2.0]
[Paso 26] Tipo: MÍNIMO LOCAL (H positiva definida)
```

Figura 19. Solución de la función objetivo bajo el método Hessiano.

Para el quinto uso, se ingresa la función objetivo $3X^2 + 2XY + 4XY^2 + 5X - 6Y$, con restricciones $X + 2Y = 4$. Se observa que se selecciona el método de programación cuadrática para su solución, la que es descrita en un paso a paso, como se espera.

```
[Paso 1] Variables detectadas: x, y
[Paso 2] Función: 3*x**2 + 2*x*y + 5*x + 4*y**2 - 6*y
[Paso 3] Igualdades: 1, Desigualdades: 0
Sugerido: Programación Cuadrática
[Paso 4] Método seleccionado: Programación Cuadrática
[Paso 5] Intentando interpretar función como cuadrática (QP).
[Paso 6] Matriz Q estimada:
[Paso 7] [
 [6, 2]
 [2, 8]
]
[Paso 8] Vector c estimado:
[Paso 9] [5, -6]
[Paso 10] Valores propios Q: [4.76393, 9.23607]
[Paso 11] Convexidad (Q semidef positiva): True
[Paso 12] Hay restricciones: intentando resolver numéricamente con SLSQP.
[Paso 13] Matriz A (igualdades) extraída con linear_eq_to_matrix:
[Paso 14] [
 [1, 2]
]
[Paso 15] Vector b:
[Paso 16] [-4]
[Paso 17] Solución SLSQP: x = [-2, -1], fun = 16.0000000001
```

Figura 20. Solución de la función objetivo bajo el método programación cuadrática.

V. CONCLUSIONES

El código desarrollado automatiza la selección del método de optimización más adecuado (sin restricciones, Hessiana, Lagrange, KKT o programación cuadrática) a partir de la forma de la función objetivo y de las restricciones que ingresa el usuario, lo que facilita el análisis de problemas de programación no lineal en contextos educativos.

Además, las pruebas con funciones clásicas —desde cuadráticas simples hasta funciones con restricciones de igualdad, desigualdad y términos trigonométricos— evidencian que el sistema identifica correctamente el tipo de problema y produce soluciones coherentes con la teoría, incluyendo la clasificación de puntos críticos mediante la matriz Hessiana y el diagnóstico de mínimos locales condicionados.

Finalmente, el uso combinado de bibliotecas simbólicas y numéricas en Python, como SymPy y NumPy, demuestra la viabilidad de integrar en una sola herramienta métodos analíticos y numéricos, abriendo la posibilidad de abordar problemas de mayor dimensión e incorporar algoritmos de optimización más avanzados en futuras extensiones del proyecto.

VI. REFERENCIAS

- Taha, H. A. (2017). Investigación de operaciones (10.ª ed.). Pearson Educación.

- Ghojogh, B., & otros. (2021). KKT Conditions, First-Order and Second-Order Optimization, and Distributed Optimization
- NEOS Guide. (2022). Quadratic Programming.
- Bazaraa, M. S., Sherali, H. D., & Shetty, C. M. (2006). Nonlinear Programming: Theory and Algorithms (3rd ed.). Wiley.