Optimización Cuántica para la Trayectoria de Ascenso de Aeronaves

Marco A. Erazo. Dir: Leonardo A. Pachon

10 de noviembre de 2024

1. Introducción

Este documento describe un enfoque aproximado haciendo solo dos particiones (N=2) en el asenso para la optimización de la trayectoria de la aeronave, combinando técnicas clásicas y de computación cuántica. El problema, derivado del Airbus Quantum Computing Challenge, busca minimizar una función de costo que representa el consumo de combustible y el tiempo de vuelo.

La estrategia se basa en:

1. **Exploración Clásica Inicial:** Se utiliza un método de búsqueda clásico para encontrar una región prometedora en el espacio de soluciones.
2. **Aproximación Polinomial:** Se utiliza una aproximación polinomial de Chebyshev para representar la función de costo en la región identificada.
3. **Optimización Cuántica:** Se emplea un algoritmo de recocido cuántico simulado (Simulated Annealing) sobre la representación polinomial para encontrar la solución.
4. **Comparación con Optimización Clásica:** Finalmente, se compara el resultado con una optimización clásica utilizando el método Nelder-Mead.

2. Descripción del Código

2.1. Función de Costo y Adaptación

El código comienza definiendo la función de costo `Fun(x)` (omitida aquí por brevedad, pero presente en el código original), que encapsula el modelo completo del problema de optimización de la trayectoria de ascenso. Se adapta para su uso con algoritmos de optimización a través de la función `FF(x)`, que maneja los valores no numéricos (NaN) y retorna un valor alto en caso de que la solución sea infactible. Además, se define 'fun(x, y)' para trabajar con arrays, lo cual es necesario para la vectorización y paralelización de la evaluación de la función.

```python
from math import sin, tan, sqrt, pow, asin, atanh, log, exp, pi, nan
import numpy as np

def Fun(x): #Funcion principal de costo
    return   (np.concatenate(([v_0], x1)), np.concatenate(([ _0 ], x2*  /180)))

def FF(x):
    a = -Fun(x)
    if a > 0:
        return a
    else:
        return 58626 # Valor mas alto en la funcion

def fun(x, y):  # Adaptaci n para arrays
    if np.isscalar(x) and np.isscalar(y):
        return FF([x, y])
    else:
        coords = np.stack((x, y), axis=-1)
        vectorized_funn = np.vectorize(FF, signature='(n)->()')
        return vectorized_funn(coords)
```

2.2. Exploración del Espacio de Soluciones

Se implementa la función `buscar_puntos` para explorar el espacio de soluciones alrededor de un punto inicial. Esta función genera puntos aleatorios siguiendo una distribución normal, filtra aquellos que caen fuera de los límites definidos y selecciona los puntos con los valores de la función objetivo más bajos, favoreciendo la convergencia hacia regiones prometedoras.

Esta código que se puede ver completo en el archivo .pynb en la referencia es simple pero altamente funcional para los requerimientos, a partir de una adecuada configuración se expande orgánicamente sobre la superficie multinacional alejándose de saltos los cuales bajo alguna configuración pueden ser los limites impuestos por las restricciones, sobre esto se aclarará y se explicara detalladamente (estoy aun trabajando en esto)

```python
def buscar_puntos(f, punto_inicial, intervalo, n, nb, c):
    return puntos_encontrados

puntos = buscar_puntos(fun, [188, 1.7], [[110,250],[0,20]],100, 10, 0.1)
```

buscar_puntos() recibe como argumentos una función, un punto inicial de referencia desde el cual buscar, un intervalo de búsqueda, un entero "n"que es el numero de puntos generados normalmente al rededor del punto inicial, nb es el numero de iteraciones repitiendo la búsqueda pero al rededor ya no del punto inicial sino del punto medio sobre los puntos encontrados esto genera una búsqueda movil que se aleja de los limites restringidos sobre los que se ha puesto un valor alto (este punto es importante y sigo trabajando en el, hay múltiples formas de trabajar estos limites, se puede hacer directamente en la función final que se optimiza en el simulador cuantico, estoy trabajando en esto)

2.3. Aproximación con Polinomios de Chebyshev

La función `polinomio_de_cheb` genera un polinomio de Chebyshev que aproxima la función de costo en la región de interés. Esto permite una representación manejable para la optimización cuántica.

```
25
26 def polinomio_de_cheb(f, p, g):
27     #... (C digo completo el el repositorio
28     return expr
29
30 polinomio_chebyshev = polinomio_de_cheb(fun, puntos, 5)  # Grado 5
```

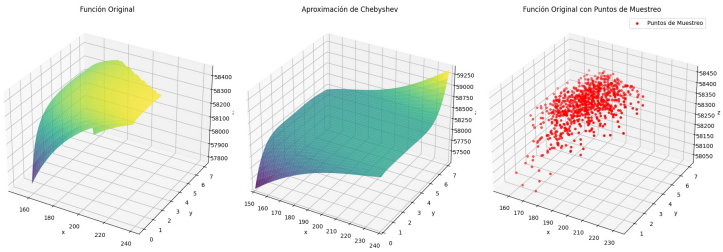

Figura 1: Ejemplo de visualización de la función original y la aproximación de Chebyshev.

2.4. Optimización Cuántica

Se utiliza la biblioteca D-Wave Ocean para realizar una optimización cuántica. La función `crear_qubo` construye una representación QUBO del polinomio de Chebyshev, y la función `resolver_y_mostrar_resultados` ejecuta el recocido simulado y muestra los resultados.

```
31 import dimod
32 from pyqubo import Binary, Array
33
34 def crear_qubo(num_bits=4):
35     return bqm
36
37 def resolver_y_mostrar_resultados(num_bits=4, num_reads=1000):
38     visualizar_funcion(x_valor, y_valor)
39
40 resolver_y_mostrar_resultados(num_bits=10, num_reads=10)
```

2.5. Optimización Clásica y Comparación

Se implementa la función `optimizar_no_suave` para optimizar la función original `fun` utilizando el método Nelder-Mead de SciPy. Esto permite comparar el rendimiento del enfoque híbrido con un método de optimización clásico.

```
from scipy.optimize import minimize
def optimizar_no_suave(fun, x0, y0, bounds_x, bounds_y, method='
    Nelder-Mead', reinicios=5):
    return x_optimo, y_optimo


x_inicial = 180
y_inicial = 1.7
limites_x = (150, 240)
limites_y = (0, 7)

x_opt, y_opt = optimizar_no_suave(fun, x_inicial, y_inicial,
    limites_x, limites_y)
```

3. Resultados y Visualización

Esta es una gráfica de contorno 2D que representa la función objetivo simplificada que se utiliza en la optimización cuántica. Las líneas de contorno conectan puntos con el mismo valor de la función. La estrella roja indica la mejor solución encontrada por el optimizador cuántico.

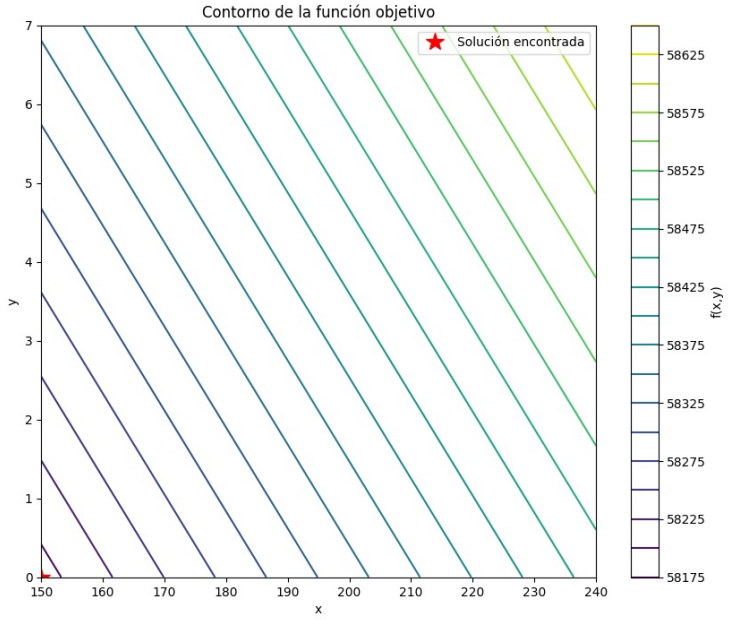

Figura 2: Función objetivo simplificada.

4. Nota

Este enfoque aprovecha las fortalezas de las técnicas clásicas y cuánticas para abordar el problema de optimización de la trayectoria de ascenso. La exploración clásica inicial y la aproximación polinomial reducen la complejidad del problema para la optimización cuántica, mientras que el recocido cuántico permite explorar el espacio de soluciones de manera eficiente. La comparación con un método clásico proporciona una evaluación del rendimiento del enfoque híbrido.

Este docu muestra una simplificación para N=2, una optimización para 10 bits en el simulador, estoy trabajando para escalar estos valores para explorar su viabilidad, ademas de incorporar las restricciones directamente en el optimizador cuántico.

Código fuente en GitHub