Universidad de Costa Rica

Herramientas de Ciencias de Datos II

Modelo de optimización por enjambre de partículas

Proyecto individual

Gustavo Alberto Amador Fonseca - C20451

25 de junio de 2024

1. Objetivo General

Programar y mejorar el modelo de optimización con ejmabre de partículas mediante paralelización en el lenguaje de programación Python.

2. Objetivos específicos

- Analizar y programar el algoritmo PSO estándar planteado en el trabajo de maestría de David Erroz Arroyo.
- Desarrollar un modelo PSO mediante técnicas de paralelización en Python para mejorar la convergencia en el espacio.
- Comparar cuantitativa los modelos PSO estándar, paralelizado y un método de optimización basado en gradiente, evaluando su desempeño en términos de velocidad y precisión.

3. Introducción

El modelo de optimización por enjambre de partículas es una técnica heurística relativamente reciente, creada en 1995 por James Kennedy y Russell Eberhart mientras estudiaban un modelo para describir el comportamiento social de los animales en grupo y su capacidad para solucionar problemas.

Basado en el trabajo del autor David Erroz, la lógica del modelo se basa en liberar aleatoriamente partículas en un espacio. Estas partículas se moverán dependiendo del mejor punto crítico global encontrado por todas las partículas, de la aleatoriedad y de ciertas constantes ya establecidas para el modelo estándar. Las partículas se movilizan por el espacio buscando y reportando los mejores puntos críticos que encuentren, estableciendo como mejor punto crítico el mejor de los puntos críticos individuales de cada partícula. Con las iteraciones y debido a la dependencia del mejor crítico global, las partículas se agruparán en este hasta que, después de ciertas iteraciones, todas las partículas estén en el punto crítico global.(Erroz, 2022)

4. Explicación del código

El codigo se puede encontrar en el repositorio: `https://github.com/Gust4voAmador/Proyecto_Individual_Herramientas_Cienca_de_Datos_II`

1. Bibliotecas utilizadas:

   - `numpy (import numpy as np)`:
     - **Descripción:** `numpy` es una biblioteca fundamental para la computación científica en Python. Proporciona soporte para arrays y matrices multidimensionales, junto con una colección de funciones matemáticas de alto nivel para operar con estos arrays de manera eficiente.
   - `matplotlib.pyplot (import matplotlib.pyplot as plt)`:

- **Descripción:** `matplotlib.pyplot` es una biblioteca utilizada para crear gráficos y visualizaciones en Python. Proporciona una interfaz similar a MATLAB para la creación de gráficos 2D, como líneas, barras, histogramas, dispersión, etc.

- `pandas (import pandas as pd)`:
  - **Descripción:** `pandas` es una biblioteca utilizada para la manipulación y el análisis de datos. Proporciona estructuras de datos rápidas, flexibles y expresivas, como DataFrames, que son ideales para trabajar con datos etiquetados o relacionales.
- `time (import time)`:
  - **Descripción:** `time` es un módulo estándar de Python que proporciona varias funciones relacionadas con el tiempo. Se utiliza para medir la duración de la ejecución de fragmentos de código, pausar la ejecución, obtener la hora actual, entre otros.
- `scipy.optimize (from scipy.optimize import minimize)`:
  - **Descripción:** `scipy.optimize` es un submódulo de `SciPy` que proporciona algoritmos de optimización. `minimize` es una función que se utiliza para encontrar el mínimo de una función escalar, ya sea sin restricciones o con restricciones.

2. Programción de funciónes de prueba: Para probar los distintos modelos en el codigo decidí hacerlo mediante dos funciones, una función cuadrática centrada en 0.6667 de 2 dimensiones y la función de Ackley que se puede establecer para muchas dimensiones y presenta una infinidad de mínimos y máximos locales, no obtante el el mínimo global es en cero.`None`.

3. Programación del modelo del modelo PSO estándar:

   Primero se crea el método initialize_particles el cual ubica de manera aleatorial la cantidad de partículas seleccionadas en el espacio seleccionado, además se les asigna un vector de velocidad, que es como la ïnersia"de cada partícula que depende de constantes ya establecidas y aleatoriedad.

   Posteriormente se crea otro médodo llamado pso, el cual da como inicio la implementación del la búsqueda. Este hace uso del método initialize_particles para iniciar con las partículas aleatorias, se determina el mínimo personal para cada partícula en la iteración, estos mínimos personales se comparan para obtener el mínimo global y se establece; todo esto para crear las condiciones para implementar el modelo en iteraciones.

   Luego se da inicio a un ciclo for por la cantidad de iteraciones que se establecieron, este posee dentro dos ciclo for más. El primero se dedica a generar nuevas posiciones para las partículas y ubicarlas influenciadas por el mínimo global actual, el segundo se encarga de establecer los nuevos minimos personales y si hay un nuevo minimo global lo establece, sino se queda con el ya encontrado. Para el final de los ciclos for se espera que la mayoría de las partículas se encuentren en la posición y valor de mínimo globla de la función seleccionada. `None`.

4. Algoritmo de obtimización por enjambre de partículas en 6 particiones con paralelización: En un script .py Clase_paralelizacion se creó una clase llamada Paralelizacion. Esta cuenta con los método del modelo estándar, los métodos para las funciones de prueba y los que llevan acabo la paralelización. Primeramente

se crea el método ejecutar_pso_en_subregion el cual lo simplemente aplica el modelo pso un una subregión indicada. También se crea un método que encargado de dividir el espacio global 6 partes.

El método ejecutar_optimizacion se encarga de llamar los metodos ya creados para primeramente dividir el espacio de búsqueda el 6 particiones de tamaño similar, a estos subespacios un núcleo del precesador de la cumputadora se dedica a aplica el modelo pso de forma paralelizada. Una vez terminado el modelo se comparan los mínimos globales de cada subespacio y el menor de todos se establece como el mínimo global de toda el área. Decidí que fuera 6 espácios ya que mi computadora cuenta con 6 núcleos, y así no se queden subespacios en espera.

5. Métodos ejecutadores: Se crean metodos para ejecutar 5 veces un modelo y devolver un data frame con el número de corridas, el tiemo de ejecución de cada ejecución, el mínimo global encontrado y el promedio del tiempo y mejor valor encontrado. Estos métodos se crean para el modelo estándar, el paralelizado y el Algoritmo de Descenso del Gradiente Método BFGS, el cual fue cargado de la librería de python llamada scipy.optimize.

6. Método graficador: Se crea un método encargado de graficar el comportamiento de las partículas del modelo estándar para 3 iteraciones específicas en la funciones cuadrática y Ackley.

5. Comparación de los modelos:

5.1. Comparación Función Cuadrática: valor óptimo = 2/3:

- Intervalo de prueba de [-10, 10] y de 2 dimensiones: Se puede identificar que todos los modelos llegaron al punto de optimización. No obstante, la mayores diferencias radican en el tiempo de ejecución. Pues el Gradiente fue extremadamente rápido, luego sigue el estandar y por último el paralilizado, siendo el algoritmo que más tardó en ejecutarse.

- Intervalo de prueba de [-1000, 1000] y de 2 dimensiones: Se puede evidenciar que el algoritmo de gradiente sigue durando muy poco aunque aumentó su tiempo de ejecución, pero sigue siendo el mas rápido. Es interesante notar que el PSO estádar más bien dueró menos tiempo que cuando el intervalo era mucho más pequeño mostrando que es más efectivo a mayor exigencia. Por otro lado el algoritmo paralelizado a pesar que duró más, fue necesario aumentar el número de particulas a 120 para que llegara al punto mínimo de la función.

- Conclusión: Se pudo evidenciar como en una función "bien portada"de dos variables el algoritmo PSO fue ampliamente vencido en efectividad y tiempo de ejecución por el Descenso Gradiente.

5.2. Comparación Función Ackley: valor óptimo = 0:

- Gráfica de la función Ackley en 2 dimensiones aproximada por el algoritmo PSO Al ejercutar los algoritmos con la función Ackley se pueden observar varias cosas, primeramente el algoritmo gradiente sigue ejecutándose muy rápido pero al valor que llega es sumamente incorrecto, por lo que la velocidad de ejecución no aporta algo necesario si el resultado está mal. No obstante el algoritmo de ejambre estandar

y el paralelizado llegaron a un resultado; el estándar fue más rápido pero es menos preciso, mientras que el paralelizado no fue más rápido pero fue extremadamente preciso el resultado.

- Intervalo de prueba de [-10, 10] y de 10 dimensiones: Desde la prueba anterior el algoritmo de gradiente no sirve. Al aumentar las dimensiones a 10 y manteniendo la misma área se evidencia que el modelo PSO estádar se queda corto para poder optimizar pues en los 5 casos se quedó atrapado en un punto crítico local. No obstante, el PSO paralelizado sigue durando más pero el encontrar el punto crítico global fue impecable y recordando que está haciendo 25 iteraciones por partición.

- Intervalo total [-300, 300] en 10 dimensiones: Con 30 particulas y 100 iteraciones por subintervalo,nótese que para este intervalo, con la cantidad de particulas e iteraciones empleadas el modelo ya no es suficiente para encontrar el punto crítico global y queda atrapado en una local. Por lo que se procede a subir el número de particulas o el número de iteraciones para ver el resultado.

- 8000 particulas y 25 iteraciones por subintervalo: Para poder encontrar el punto crítico global en la mayoría de las pruebas fue necesario subir hasta 8000 partículas por subintervalo, probando aparte se subió hasta 10 000 partículas e igualmente en casos no aproximaba bien. Además que el tiempo de ejecución subía considerablemente al aumentar el número de partículas y los resultados no eran tan evidentes como se esperaba.

- 30 particulas y 10000 iteraciones por subintervalo: Solo aumentando el número de iteracine no se evidenció mejora en lo absoluto, pues no fue capáz de encontrar el punto crítico global y se quedó atrapado el famoso valor de 20. En una prueba se subió a 100 000 iteraciones por subintervalo y quedó atrapdo en el mismo valor y el tiempo de ejecución subió a 74.71 en promedio por ejecución, lo que no en lo absoluto optimizado.

- 100 particulas y 5000 iteraciones por subintervalo: Se puede evidenciar que el aumento de partículas e iteraciones por subintervalo si mejora la precisión del modelo, pero el esfuerzo y recursos invertidos para encontrar algunos puntos globales y que incluso aveces no es demasiado para los resultados que se obtienen.

5.3. Conclusión:

El modélo de optimización por ejambre paralelizado logró lo esperado, que era superar tanto el modelo de gradiente de una librería de python así como el modelo estádar de optimización por ejambre obtenido de una fuente, pero al probar con funciones complejas con muchos puntos críticos, pues en funciones con monotonías mas evidentes, el modelo es ampliamente superado en tiempo de ejecución por otros menos robustos.

Solamente el aumento del número de partículas e iteraciones no muestran un aumento de rendimiento directo, la necesidad de recuersos necesario crece muy rápido comparado con los resultados.

El aumento del intervalo de búsqueda baja considerablemente la efectividad del modelo, por lo que trabaja mejor en intervalos menores a una longitud de 200, es decir con particiones menores o iguales a 33 aproximadamente.

Como recomendación, este modelo se puede hacer mucho más robusto agregando un segundo nivel de búsqueda más acotada al rededor de los puntos locales encontrados y asé aumentar la probabilidades de éxito.

Referencias

del Misterio, E.-T. (2021). *Pso: Optimización por enjambre de partículas.* Descargado de `https://www.youtube.com/watch?v=fPPjF_F8k3U&t=15s`

Erroz.D. (2022). *Sobre el funcionamiento interno del pso y construcción de nuevo método.* Descargado de `https://oa.upm.es/71422/`