Suma de Riemann por la derecha

Castañón Hernández V. J., Castro Elvira D., Sánchez Zanjuampa M. A., Galicia Cocoletzi N.
Instituto Politécnico Nacional, UPIIT
{vcastanonh2100,dcastroe2100,msanchezz2100,ngaliciac2100}@alumno.ipn.mx

1. Problema

La suma de Riemann es un método utilizado para aproximar el área debajo de la curva de una función definida. Esta técnica tiene variantes como hacer la evaluación dependiendo de la posición de donde se tome como referencia el rectángulo debajo de la curva.

La suma por la derecha busca hacer una evaluación de una sumatoria la cual calcula los valores de los rectángulos por la parte derecha de la figura. Cabe mencionar que estos rectángulos deben ser idealmente del mismo ancho dependiendo del número de subintervalos que se quieran representar dentro del intervalo principal (los límites de la función definida).

Formalmente, la suma de Riemann por la derecha se puede definir por la fórmula[1]

$$\sum_{n=1}^{n} f(x_i)\Delta x \tag{1}$$

Donde x_i son los puntos del extremo **derecho** de cada subintervalo, y Δx representa el ancho de cada una de los rectángulos inscritos debajo de la curva de la función, y esta dada por la función[2]

$$\Delta x = \frac{b-a}{n} \tag{2}$$

En donde a y b son los intervalos de la función evaluada.

Dato: Entre mayor sea el número de rectángulos inscritos bajo la curva de la función (subintervalos), el área total de la suma de Riemann se acercará cada vez más al valor real, siempre y cuando la función sea integrable entre un intervalo dado.

El problema a resolver consiste en la paralelización de operaciones, en donde dependiendo del número de ***subintervalos*** *requiere de una operación completa más al número total de operaciones realizadas en la suma de Riemann. Para este proceso será necesario dividir el número de operaciones que realizará cada procesador para reducir tiempo de ejecución y un mejor aprovechamiento de los recursos de la computadora. Siempre y cuando la cantidad de subintervalos sea significativa y represente un reto para el cálculo total del problema.*

2. Pseudocódigo

Algorithm 1 Nivelación de Cargas

function NIVELACION_CARGAS(D, n_p)
  $s \leftarrow$ longitud(D) $\mod n\_p$
  $t \leftarrow$ entero $\left(\frac{\text{longitud}(D)-s}{n\_p}\right)$
  $out \leftarrow []$
  for $i \leftarrow 0$ **to** n_p **do**
    if $i < s$ **then**
      out.añadir($D[i \cdot t + i : i \cdot t + i + t + 1]$)
    else
      out.añadir($D[i \cdot t + s : i \cdot t + s + t]$)
    end if
  end for
  return out
end function

Algorithm 2 Suma de Riemann sin pool

function SUMA_RIEMANN(a, b, funcion, figuras_inscritas)
  $t \leftarrow$ time()
  $ancho \leftarrow \frac{b-a}{\text{figuras\_inscritas}}$
  escribir .Ancho de los rectangulos ->"$ancho$
  $suma\_riemann \leftarrow 0$
  for $i \leftarrow 1$ **to** figuras_inscritas **do**
    $datos \leftarrow a + i \times ancho$
    $suma\_riemann \leftarrow suma\_riemann +$ funcion($datos$) $\times ancho$
  end for
  $tiempo\_total \leftarrow$ time() $- t$
  retornar $suma\_riemann, tiempo\_total$
end function

Algorithm 3 Suma de Riemann con pool

function RIEMANN_POOL(a, b, num_proce, funcion, figuras_inscritas)
  $t \leftarrow$ time()
  $ancho \leftarrow \frac{b-a}{\text{figuras\_inscritas}}$
  escribir .Ancho de los rectangulos ->"$ancho$
  $datos \leftarrow$ dividir equitativamente($a + ancho, b$, figuras_inscritas)
  $nivel\_cargas \leftarrow$ nivelacion_cargas(np.array($datos$), num_proce)
  con Pool(num_proce) **como** pool:
    $resul\_sublistas \leftarrow$ pool.map(funcion, nivel_cargas)
  $resultados\_combinados \leftarrow$ lista(concatenar iterables($resul\_sublistas$))
  $resul \leftarrow$ suma($resultados\_combinados$) $\times ancho$
  $tiempo\_total \leftarrow$ time() $- t$
  retornar $resul, tiempo\_total$
end function

3. Descripción del programa

Nuestra solución propuesta consta de:

1. **Cálculo del Ancho de los Rectángulos**

- En primer lugar Se calcula el ancho de cada rectángulo en la suma de Riemann, siguiendo la formula: $\Delta x = \frac{b-a}{n}$ donde recibimos el límite inferior (a), limite superior (b) y la cantidad de figuras inscritas que deseamos.

2. **Generación de Puntos para la Suma de Riemann**
   - Para la creacion de los puntos, se utilizo la funcion : $np.linspace$, la cual se le pasa el intervalo [a + ancho, b] con un total de las figuras inscritas deseadas. Estos puntos representan el lado derecho de cada subintervalo en la suma de Riemann por la derecha.
3. **División de Cargas**
   - Una vez generados los puntos, se dividen en la cantidad de procesadores dada, utilizando la función $nivelacion_cargas$. Esta función devuelve una lista de listas, donde cada lista interna contiene los puntos asignados a un proceso.
4. **Ejecución en Paralelo**
   - Para realizar la parte de paralelo se utiliza $multiprocessing.Pool$ para ejecutar la función lambda dada en paralelo sobre las sublistas generadas. Cada proceso ejecuta la función en su conjunto de puntos asignados.
5. **Aplanado y Suma de Resultados**
   - Los resultados obtenidos se aplanan utilizando $chain.from_iterable$ esto con la finalidad de unificarlos en una lista para luego sumar los elementos y multiplicarlos por el ancho obtenido para obtener el resultado final

4. Código

```
def riemann_Pool(a, b, num_proce, funcion, figuras_inscritas):
    t = time()
    ancho = (b - a) / figuras_inscritas
    datos = np.linspace(a + ancho, b, figuras_inscritas)
    nivel_cargas = nivelacion_cargas(D=np.array(datos), n_p=num_proce)

    with Pool(num_proce) as pool:
        resul_sublistas = pool.map(funcion, nivel_cargas)

    resultados_combinados = list(chain.from_iterable(resul_sublistas))
    resul = sum(resultados_combinados) * ancho
    tiempo_total = time() - t
    return resul, tiempo_total
```

5. Resultados

Para la verificación de los resultados obtenidos mediante la técnica de la suma de Riemann, se utilizó la siguiente página web como punto de comparación https://www.emathhelp.net/es/calculators/calculus-2/riemann-sum-calculator/
Se seleccionaron los siguientes datos:

$$\lambda \to x^2 + 10x \tag{3}$$

$$datos \to 200$$

$$rango \to [0 - 200]$$

$$num\_procesadores \to 8$$

Para este caso, la respuesta obtenida, tanto de la implementación realizada como la página web, arrojaron un resultado de aproximadamente, 2845700. También se obtuvieron los resultados de la siguiente ecuación, usando la misma cantidad de daros y el mismo rango, que el caso anterior.

$$\lambda \to x/2 + x^{\frac{1}{2}} + 2 \tag{4}$$

En este caso la respuesta encontrada por ambos métodos fue de 12342.4842.
Por lo tanto, se puede decir que la implementación propuesta en esta práctica es correcta

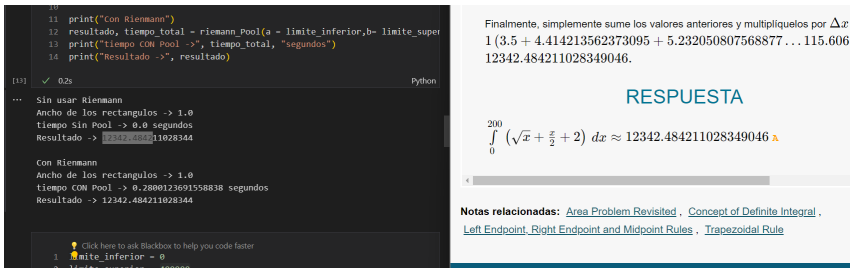

Figura 1: Datos obtenidos por ambas vías de comprobación

6. Pruebas

Se realizó un análisis del tiempo destinado para la resolución de la suma de Riemann mediante dos enfoques, de un lado utilizando el cómputo paralelo y del otro, empleado el enfoque secuencial. Es importante recalcar que para observar la diferencia de tiempo entre estos enfoques se empleó una cantidad considerable de datos

1. Se aplicó la siguiente función lambda, dentro de la suma de Riemann:

$$\lambda \rightarrow x^2 + 10x \tag{5}$$

$$datos \rightarrow 50000000$$

$$rango \rightarrow [0 - 50000000]$$

$$num\_procesadores \rightarrow 8$$

Se comprueba los tiempos de ejecución en segundos usando la técnica de paralelización y sin paralelización obtenemos:

Tabla 1: Comparación para la ecuación 1

Prueba	Paralelo	Sin paralelización	figuras inscritas
1	5.6497	8.5831	50000000
2	10.1285	7.5513	60000000
3	5.1677	6.3147	40000000

2. Se aplicó la siguiente función lambda, dentro de la suma de Riemann:

$$\lambda \rightarrow x/2 + x^{\frac{1}{2}} + 2 \tag{6}$$

$$datos \rightarrow 400000$$

$$rango \rightarrow [0 - 400000]$$

$$num\_procesadores \rightarrow 4$$

Se comprueba los tiempos de ejecución en segundos usando la técnica de paralelización y sin paralelización obtenemos:

Tabla 2: Comparación para la ecuación 2

Prueba	Paralelo	Sin paralelización	figuras inscritas
1	0.6120	0.6957	4000000
2	1.3598	1.0334	8000000
3	2.0781	1.3895	12000000

3. Se aplicó la siguiente función lambda, dentro de la suma de Riemann:

$$\lambda \rightarrow x^2 - 2^x - \frac{x}{2} * x^8 + 2x^3 \tag{7}$$

$$datos \rightarrow 100000000$$

$$rango \rightarrow [0 - 10000000]$$

$$num\_procesadores \rightarrow 7$$

Se comprueba los tiempos de ejecución en segundos usando la técnica de paralelización y sin paralelización obtenemos:

Tabla 3: Comparación para la ecuación 3

Prueba	Paralelo	Sin paralelización	figuras inscritas
1	35.9221	47.8063	100000000
2	0.1531	0.3744	500000
3	0.2700	0.3995	800000

7. Conclusiones

Haciendo una evaluación de 2 algoritmos, en donde uno realiza los cálculos de forma secuencia, y otro de manera paralela, podemos apreciar que para un número reducido de rectángulos inscritos debajo de la curva de una función no representa una mejoría en cuanto al aprovechamiento de recursos y de reducción de tiempo, pero cuando incrementamos **significativamente** el número de rectángulos, la división de las operaciones de manera paralela sí representa una mejoría notable en cuanto al tiempo de ejecución, esto es cada vez más evidente cuando la cantidad de subintervalos supera cantidades de 6 cifras de longitud o más. Además de que el usar más figuras inscritas nos aproxima cada vez más al valor real del área bajo la curva de una función.