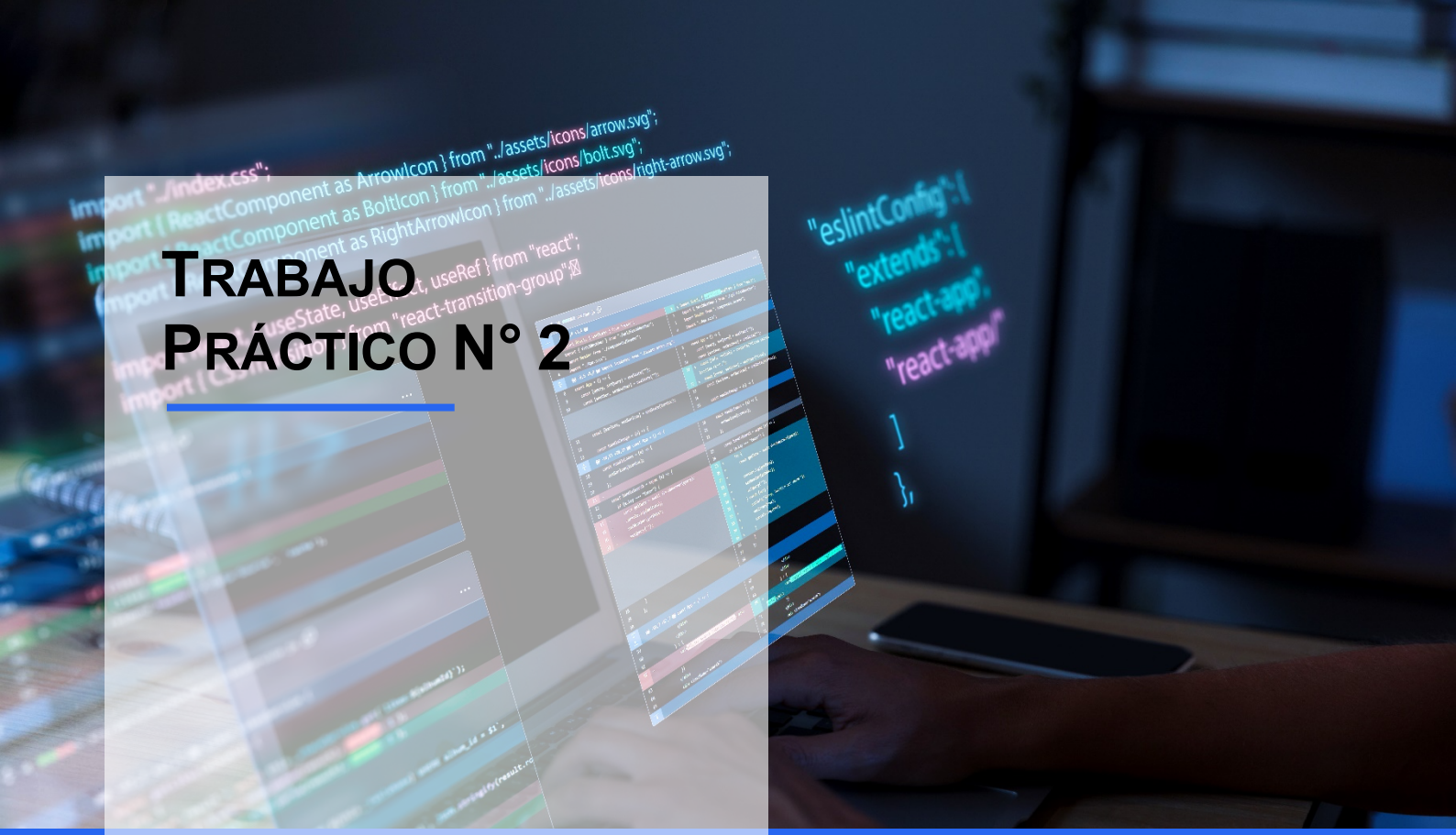


# TRABAJO PRÁCTICO N° 2

**ASIGNATURA:**
Programación Eficiente
**PROFESOR:**
Gómez, Pablo

**INTEGRANTES DEL GRUPO:**
Lo Preiato, Lucas
Oliva, Benjamín
Ventura, Gino

**FECHA**: 28/09/2023

UBP UNIVERSIDAD Blas Pascal

# INTRODUCCIÓN

En el campo de la programación y la informática gráfica, la eficiencia y el rendimiento son aspectos cruciales que determinan la calidad y la experiencia del usuario en aplicaciones y videojuegos. La elección de un algoritmo de renderizado adecuado desempeña un papel fundamental en este contexto. Este informe tiene como objetivo presentar los resultados de un estudio exhaustivo en el que se seleccionó un algoritmo de renderizado y se evaluó su rendimiento en diferentes computadoras.

En este informe, presentaremos el desarrollo de una aplicación grafica utilizando la biblioteca OpenGL. Esta aplicación tiene como objetivo graficar diferentes funciones coseno en ejes de coordenadas x e y.

# HERRAMIENTAS UTILIZADAS

- ***OpenGL:***

Es un estándar para renderizar gráficos en 2D y 3D en aplicaciones de software. Fue en la década de 1990 y desde entonces ha sido ampliamente adoptada en la industria de la informática gráfica debido a su versatilidad y capacidad para trabajar en una variedad de sistemas operativos, incluyendo Windows, macOS y Linux.
En nuestro proyecto, se utiliza OpenGL para crear y gestionar la ventana gráfica, así como para dibujar los ejes de coordenadas y las respectivas funciones.

- ***WSL2:***

WSL2, o Windows Subsystem for Linux 2, es una tecnología desarrollada por Microsoft que permite ejecutar sistemas operativos basados en Linux, como Ubuntu, Debian y otras distribuciones populares, directamente en Windows 10 o versiones posteriores.
En nuestro proyecto lo utilizamos para habilitar un entorno de desarrollo Linux dentro de Windows, lo que permitió compilar y ejecutar programas que dependen de Linux.

- ***GLFW:***

Es una biblioteca de código abierto diseñada para simplificar el proceso de creación y gestión de ventanas y contextos OpenGL en aplicaciones gráficas.

- ***XLaunch:***

Es una herramienta que forma parte del sistema X Window System (también conocido como X11), que es un sistema de ventanas utilizado en sistemas operativos Unix y Linux para la gestión de interfaces gráficas de usuario. La función principal de XLaunch es facilitar la configuración y el inicio de un servidor de visualización X en sistemas Windows.

- ***Visual Studio Code:***

Visual Studio Code, a menudo abreviado como VS Code, es un entorno de desarrollo integrado (IDE) de código abierto desarrollado por Microsoft. Fue utilizado para escribir y depurar el código.

# DESARROLLO DEL PROGRAMA

El programa desarrollado tiene como función principal generar valores de x e y para graficar y visualizar las funciones coseno en una ventana grafica. La cantidad de estas funciones las ingresa el usuario y los puntos se generan con una variación determinada previamente.

- ***Función coseno:***

El coseno es una función matemática fundamental en trigonometría que relaciona un ángulo en un triángulo rectángulo con la relación entre dos de sus lados.
En nuestro programa, realizamos nuestra propia función coseno, en vez de utilizar la función ya conocida de la librería math.h.
La función llamada customCos(x), es una implementación personalizada del cálculo del coseno de un valor x utilizando una serie de Taylor. Esta serie es una aproximación matemática que se utiliza para calcular funciones trigonométricas como el coseno.

1. ***Parámetros:*** La función toma un argumento **x**, que es el valor para el cual deseas calcular el coseno.
2. ***Constante terms:*** Se define una constante llamada **terms** con un valor de 100,000. Esta constante determina cuántos términos de la serie de Taylor se utilizarán en el cálculo. Cuantos más términos se utilicen, mayor será la precisión de la aproximación al coseno.
3. ***Variables locales:***
    - **result**: Inicializada en 1.0, esta variable almacenará el resultado acumulado de la serie de Taylor a medida que se calculan más términos.
    - **term**: Inicializada en 1.0, esta variable almacena el valor del término actual de la serie de Taylor y se multiplica por los términos sucesivos en cada iteración del bucle.
4. ***Bucle for:*** La función utiliza un bucle **for** que itera desde **n = 1** hasta **terms - 1**. En cada iteración, se calcula un nuevo término de la serie de Taylor y se agrega al resultado acumulado **result**.
5. ***Cálculo del Término:*** El cálculo del término se realiza dentro del bucle utilizando la fórmula de la serie de Taylor para el coseno. Esta fórmula utiliza potencias alternas de **x** y números factoriales. El término se multiplica por **-x * x / ((2 * n) * (2 * n - 1))**, y luego se multiplica por el término anterior en la siguiente iteración. Esto agrega sucesivamente más términos de la serie a medida que avanza el bucle.
6. ***Resultado Final:*** Una vez que se han calculado todos los términos dentro del bucle, la función devuelve el valor acumulado en la variable **result**, que es una aproximación al coseno de **x** basada en la serie de Maclaurin.

Es importante mencionar que esta implementación utiliza una cantidad fija de términos (100,000 en este caso) para la aproximación, lo que puede afectar la precisión y el rendimiento dependiendo del valor de x.
Las implementaciones del coseno en bibliotecas matemáticas estándar suelen ser más eficientes y precisas, pero esta función muestra cómo se puede realizar una aproximación del coseno mediante una serie matemática.

# ESTRUCTURA DEL PROGRAMA

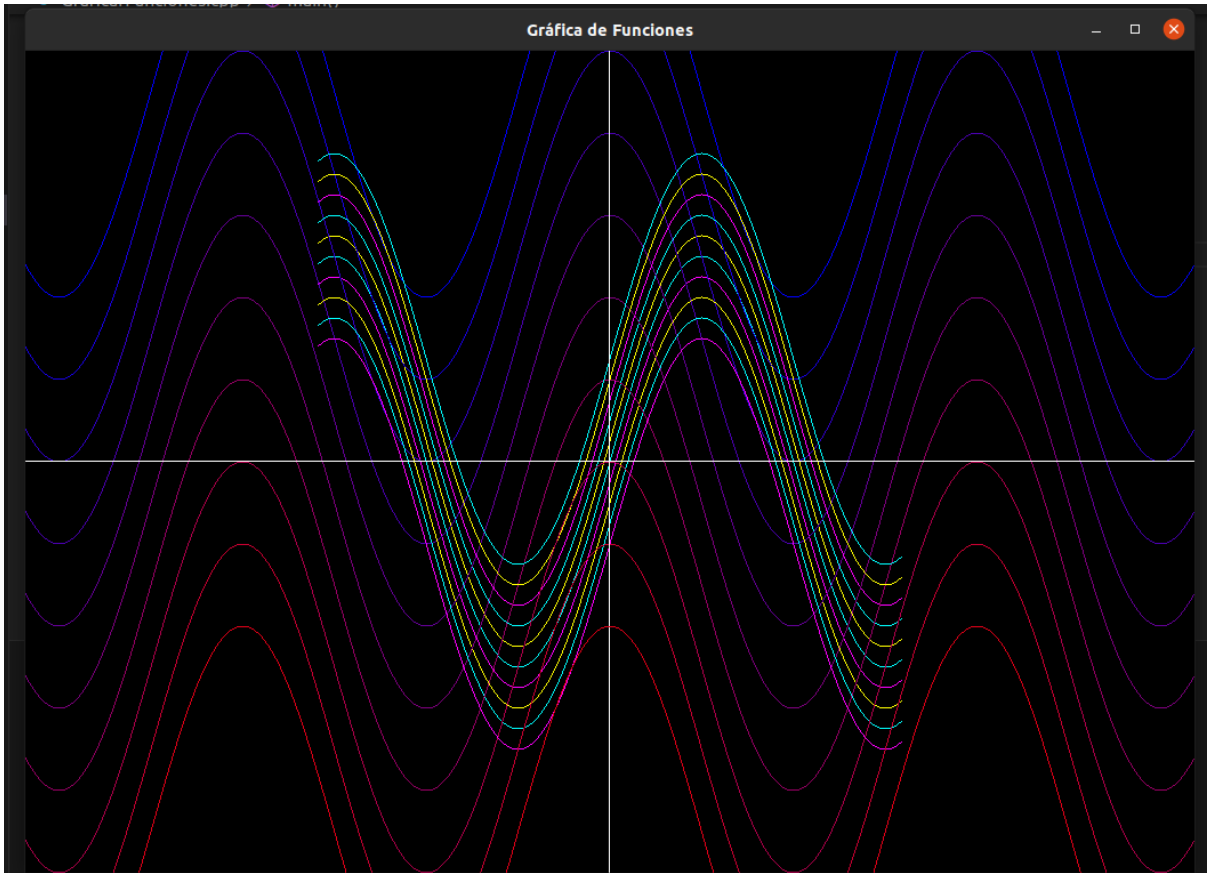
1. ***Inclusión de bibliotecas***: en esta parte, se incluyen las bibliotecas necesarias para el proyecto, estas bibliotecas son utilizadas para trabajar con OpenGL (gráficos), GLFW (manejo de ventanas) y otras para manipular vectores y medir el tiempo.
2. ***Variables globales***: estas variables definen el ancho y alto de la ventana de visualización.
3. ***Funciones trigonométricas personalizadas:*** se definen dos funciones personalizadas para calcular el coseno y el seno (no utilizada) utilizando aproximaciones matemáticas basadas en series de Taylor.
4. ***Dibujo de la función:*** esta función se encarga de dibujar una función en el contexto OpenGL. Toma dos vectores de valores x e y, así como valores de color r, g, y b, para especificar el color de la línea de la función.
5. ***Función principal:*** la función principal es el punto de entrada del programa. Aquí se inicializa GLFW, se crea una ventana de visualización, y luego entra en un bucle principal que se ejecuta hasta que se cierra la ventana.
6. ***Bucle principal:*** Este es el bucle principal del programa que se ejecuta mientras la ventana no se haya cerrado. Aquí se realizan las siguientes tareas:
    a. Se borra el búfer de color actual.
    b. Se generan los valores x para la función.
    c. Se calculan y dibujan las funciones coseno con diferentes desplazamientos verticales para crear múltiples líneas en la gráfica.
    d. Se intercambian los búferes (para mostrar la gráfica en pantalla) y se manejan los eventos de la ventana.
    e. La sección final de este bucle mide el tiempo que tarda en calcular y dibujar las funciones.
    f.

## OPTIMIZACIONES Y CORRECCIONES CON RESPECTO A LA VERSIÓN ANTERIOR

Ambos códigos son iguales en términos de la funcionalidad que realizan, pero existen algunas diferencias clave:

- ***Número de Términos en las Series:*** en la nueva versión, utilizamos 10,000 términos en las series de Taylor para calcular las funciones trigonométricas (seno y coseno), mientras que en la primera versión utilizábamos 100,000 términos. El segundo código utiliza una mayor cantidad de términos, lo que puede resultar en una mayor precisión en los cálculos, pero también en un mayor tiempo de ejecución. Luego de distintas pruebas, descubrimos que, disminuyendo la cantidad de términos, el resultado final no se veía afectado, caso contrario con el tiempo de ejecución, que para graficar 10 funciones disminuyó de 12 segundos a casi 3 segundos.
  Esta disminución en la cantidad de términos de las series de Taylor también nos permitió poder graficar mas funciones sin que el programa se rompa o se quede clavado.
- ***Medición del Tiempo:*** otro de los cambios, es la utilización de una función "measure" para medir el tiempo que lleva ejecutar la función "iterateAndDraw" con un número especificado de líneas. Ya que antes lo realizábamos dentro del bucle principal.

## NUEVOS RESULTADOS

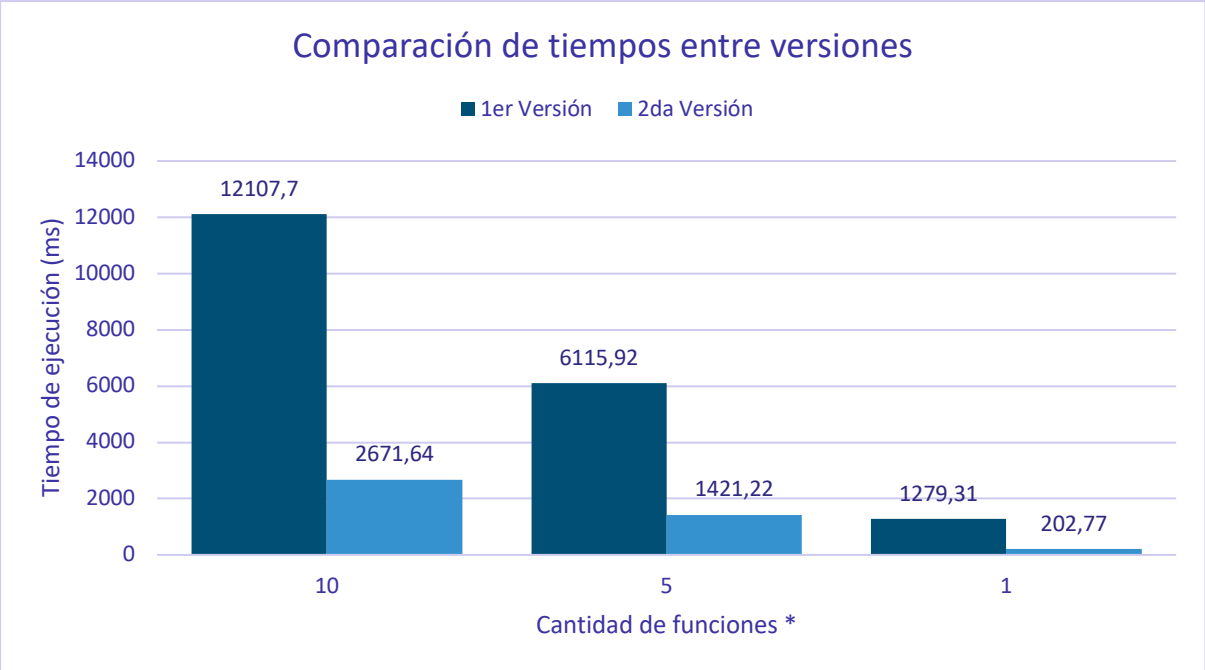


Pruebas de tiempo de ejecución para graficar funciones coseno
Valores de entrada:

1. Arreglo de valores de x (tipo vector y double).
2. Arreglo de valores de y (tipo vector y double).
3. Cantidad de funciones a graficar (entero).

| Versión anterior | |
|---|---|
| **1 función** | 1279.31 ms |
| **5 funciones** | 6115.92 ms |
| **10 funciones** | 12107.7 ms |
| **Version actual** | |
| **2 funciones (1 seno y 1 coseno)** | 202.77 ms |
| **10 funciones (5 seno y 5 coseno)** | 1421.22 ms |
| **20 funciones (10 seno y 10 coseno)** | 2671.64 ms |

# GRÁFICOS COMPARATIVOS DE TIEMPO DE EJECUCIÓN EN DIFERENTES COMPUTADORAS:




* Cabe aclarar que, en la segunda versión, la cantidad de funciones se multiplica por 2, no así el tiempo de ejecución total para graficarlas.

# EJECUCIÓN DE COMANDOS VARIOS

## COMANDO TIME

- ***Comando time con -O0:***

```
Tiempo para graficar las funciones: 1902.35 milisegundos
real    0m12,845s
user    0m12,723s
sys     0m0,065s
```

- ***Comando time con -O1:***

```
Tiempo para graficar las funciones: 950.389 milisegundos
real    0m6,094s
user    0m5,987s
sys     0m0,064s
```

## COMANDO SIZE

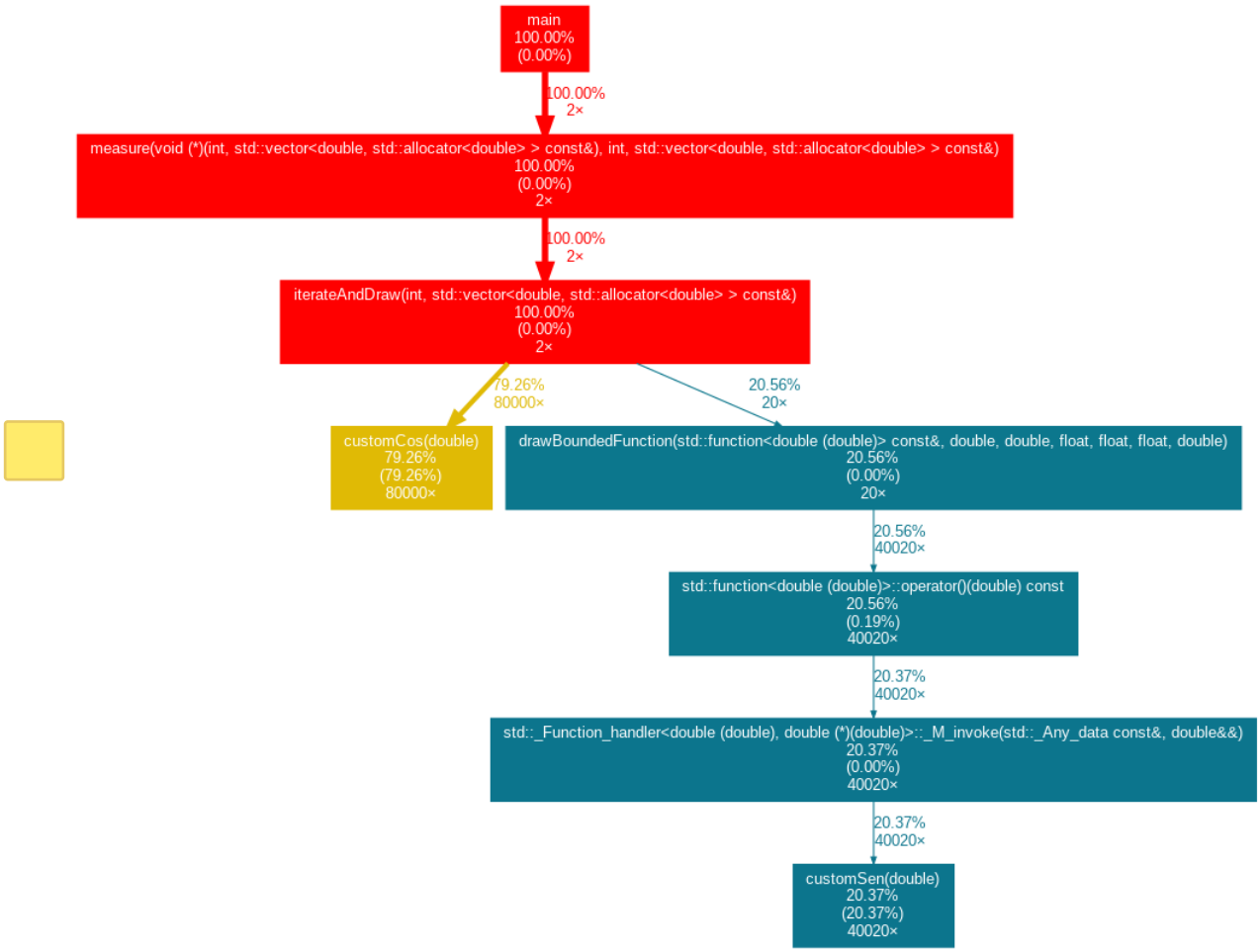
```
Size:
   text    data     bss     dec     hex   filename
  22569    1008     568   24145    5e51   ./GraficarFunciones
```

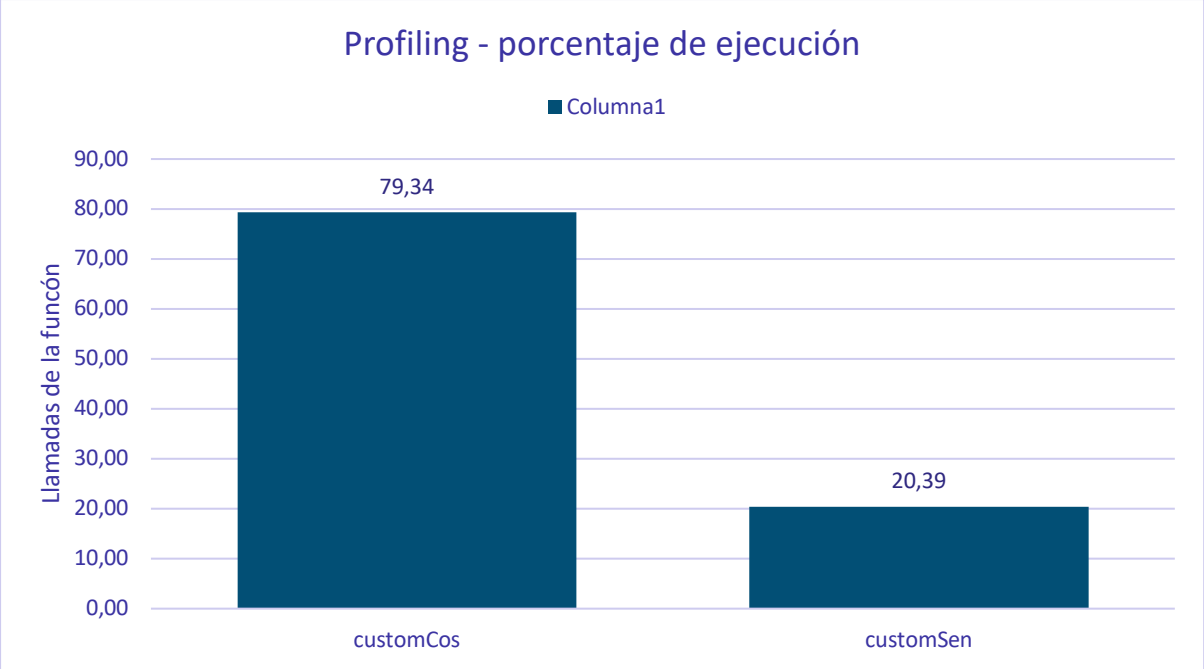
La salida del comando size muestra el tamaño de cada sección de un archivo ejecutable. Las secciones son:

- text: Contiene el código ejecutable del programa.
- data: Contiene los datos inicializados del programa.
- bss: Contiene los datos no inicializados del programa.
- dec: Tamaño total del archivo ejecutable en bytes.
- hex: Tamaño total del archivo ejecutable en hexadecimal.
- filename: Nombre del archivo ejecutable.

## PERFIL DEL PROGRAMA CON GPROF Y GPROF2DOT

También procedimos a visualizar el perfil de ejecución del programa con el uso de gprof y gprof2dot:






Podemos observar que las funciones que más tiempo consumen son “customSen()” y “customCos()”, que son las que calculan los valores para graficar en los ejes de coordenadas.

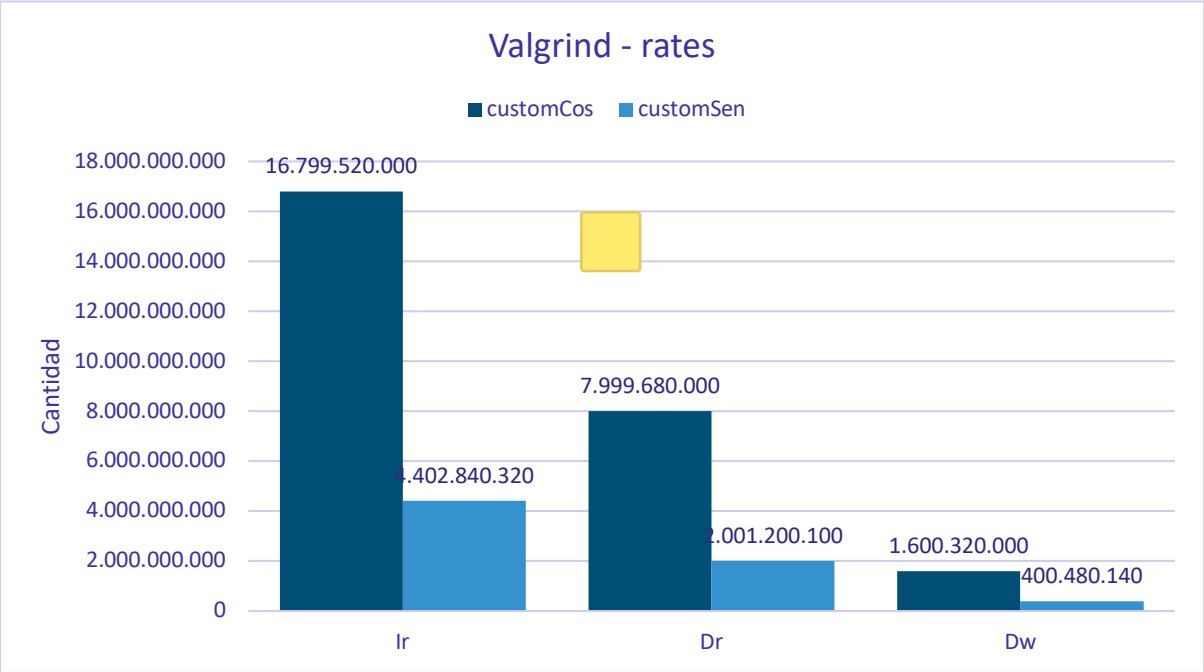
Según la información proporcionada por gprof, la función customCos() es el cuello de botella de rendimiento principal del programa. Esta función toma el 79.34% del tiempo de ejecución total.

Para mejorar el rendimiento de la función customCos(), podemos intentar lo siguiente:

- Optimizar el algoritmo o usar una implementación más eficiente (math.h).
- Reducir el número de llamadas a la función customCos() mediante el almacenamiento en caché de los resultados o mediante el uso de un algoritmo diferente que no requiera llamar a customCos().

Si podemos implementar alguna de estas mejoras, podemos esperar una mejora significativa en el rendimiento general del programa.

## VALGRIND




## ENLACE AL REPOSITORIO

**https://github.com/BenjaOliva/PEF-2023**