# Actividad: métodos numéricos en C++

Jesús María Mora Mur

May 4, 2025

# Contents


**1 Descripción de la actividad** **3**
- 1.1 Método de Muller . . . 3
- 1.2 Método de Ridders . . . 3

**2 Implementación** **3**

**3 Resultados** **3**
- 3.1 Método de Muller . . . 4
- 3.2 Método de Ridders . . . 4
- 3.3 Comparación con métodos conocidos . . . 5
  - 3.3.1 *Regula Falsi* . . . 5
  - 3.3.2 Bisección . . . 6
  - 3.3.3 Método de Newton-Raphson . . . 7

**4 Conclusiones.** **7**

# 1 Descripción de la actividad

En la presente actividad se han trabajado los métodos de Muller y Ridders para la resolución de ecuaciones. Dichos métodos son numéricos y utilizan la interpolación cuadrática y exponencial, respectivamente, para posibilitar la resolución de la ecuación siguiente:

$$f(x) = \mathrm{e}^{0.75 \cdot x} - 3 \cdot \sin(1.25 \cdot x) \tag{1}$$

Se evaluará la rapidez de los métodos en base a las iteraciones que realizan hasta llegar a la solución con una precisión de $10^{-6}$unidades. Se exponen a continuación los parámetros necesarios para que los métodos realicen correctamente la tarea encomendada.

## 1.1 Método de Muller

El método de Muller pretende interpolar la función a una parábola en un entorno localizado de una función $f(x)$. Dados dos puntos extremos y su punto medio, es posible obtener una parábola que se acerque a la función. Encontrando las soluciones a la anulación de la parábola conseguimos una aproximación. En función de en qué subintervalo se encuentre la solución, se escoge para conseguir acotar más la solución. El método converge, pero de manera lenta.

## 1.2 Método de Ridders

El método de Ridders pretende aproximar la función a una exponencial a la que se le aplica el método *regula falsi*. Con cuatro puntos obtenemos una aproximación correcta de la solución a nuestra función.

Así pues, se han creado dos funciones en C++ llamadas `muller` y `ridders` para implementar dichos métodos. Han de recibir como argumentos el *extremo inferior*, el *extremo superior* y el *número de iteraciones* que se realizarán.

# 2 Implementación

Para implementar los métodos se han realizado sendos ficheros de cabecera con formato `.hpp` en los que se da cuenta de la implementación del método. Para acceder a ellos se puede utilizar los enlaces siguientes: muller y ridders. Asimismo, se ha creado un programa principal en el que se compara estos dos métodos con otros conocidos utilizando como medición el número de iteraciones que se deben realizar para obtener el resultado con una precisión de $10^{-6}$ unidades. Se dan cuenta de los resultados en los párrafos venideros.

# 3 Resultados

Los resultados de la implementación son los siguientes:

## 3.1 Método de Muller

El método de Muller obtiene los resultados siguientes por iteración:

```
Muller 1:  0.349278

Muller 2:  0.362391

Muller 3:  0.362119

Muller 4:  0.362119
```

Se obtiene el resultado correcto con 4 iteraciones.

## 3.2 Método de Ridders

El método de Ridders consigue una precisión de $10^{-6}$ unidades con necesidad de pocas iteraciones. En concreto, los resultados son estos:

```
Ridders 1:  0.334200

Ridders 2:  0.350235

Ridders 3:  0.356890

Ridders 4:  0.359785

Ridders 5:  0.361070

Ridders 6:  0.361647

Ridders 7:  0.361906

Ridders 8:  0.362023

Ridders 9:  0.362076

Ridders 10:  0.362100

Ridders 11:  0.362110

Ridders 12:  0.362115

Ridders 13:  0.362118
```

Como se ve, no se llega al número deseado, pero se obtienen unos resultados buenos, con precisión hasta el quinto decimal.

## 3.3 Comparación con métodos conocidos

### 3.3.1 *Regula Falsi*

El método *regula falsi* converge con cierta rapidez como demuestran los siguientes resultados.

```
RF 1:  0.578050

RF 2:  0.401083

RF 3:  0.367668

RF 4:  0.362875

RF 5:  0.362222

RF 6:  0.362133

RF 7:  0.362121

RF 8:  0.362120

RF 9:  0.362119

RF 10:  0.362119

RF 11:  0.362119

RF 12:  0.362119

RF 13:  0.362119

RF 14:  0.362119

RF 15:  0.362119

RF 16:  0.362119

RF 17:  0.362119
```

Pasadas 17 iteraciones se obtiene el número esperado.

### 3.3.2 Bisección

El método de la bisección es bastante lento en comparación con los anteriores, obteniendo los siguientes resultados:

```
Bisección 1:  0.500000

Bisección 2:  0.250000

Bisección 3:  0.375000

Bisección 4:  0.312500

Bisección 5:  0.343750

Bisección 6:  0.359375

Bisección 7:  0.367188

Bisección 8:  0.363281

Bisección 9:  0.361328

Bisección 10:  0.362305

Bisección 11:  0.361816

Bisección 12:  0.362061

Bisección 13:  0.362183

Bisección 14:  0.362122

Bisección 15:  0.362091

Bisección 16:  0.362106

Bisección 17:  0.362114

Bisección 18:  0.362118

Bisección 19:  0.362120

Bisección 20:  0.362119
```

Pasadas 20 iteraciones se consigue el número buscado.

### 3.3.3 Método de Newton-Raphson

El método de Newton-Raphson es muy rápido:

```
Newton 1:  0.333333

Newton 2:  0.361670

Newton 3:  0.362119
```

Se consigue el número con 3 iteraciones.

# 4 Conclusiones.

A la vista de los resultados obtenidos, establecemos la siguiente clasificación en los métodos.

1. Método de Newton-Raphson, con 3 iteraciones.
2. Método de Muller, con 4.
3. Método de Ridders, con 13 iteraciones, aunque con imprecisión.
4. Método *Regula Falsi*, con 17 iteraciones.
5. Método de la Bisección, con 20 iteraciones.

Así, destacan los métodos estudiados por conseguir rápidamente los ceros de la función 1. En otro orden de cuestiones, los códigos han necesitado de 37 líneas para Newton-Raphson, 41 para Muller y 43 para Ridders. Se comparan solo estos por estar hechos con el mismo estilo entre sí, sin comentarios ni líneas en blanco que modifican el recuento, como es evidente. Se confirman así dichos métodos como buenos en lo que a eficiencia respecta, consiguiendo buenos resultados en poco tiempo y con relativamente poca carga computacional.