# Programación: los métodos de Heun y Ralston para resolver EDO's con valores iniciales

**Objetivos.** Programar los métodos de Heun y Ralston para resolver ecuaciones diferenciales ordinarias con condiciones iniciales. Hacer comprobaciones y comparar la eficiencia de los métodos.

**Requisitos.** Haber programado y probado el método de Euler.

**1. Ejemplo.** Se puede usar el ejemplo de la clase anterior. Vamos a resolver el siguiente problema de Cauchy:

$$x'(t) = \frac{2x(t)}{3+t}, \qquad x(0) = 1.$$

Se sabe que la solución exacta es $x(t) = \frac{1}{9}(3+t)^2$. Ya hemos programado una función `f1` de dos argumentos `t`, `x` que regresa el valor del lado derecho de la ecuación, y una función `x1` que calcula los valores de la solución exacta.

**2. Fórmulas de Heun para un paso.** Programamos una función que calcula el valor $x_{j+1}$ a partir de $t_j$, $x_j$ y $h$, usando la función f:

```
defun heunstep(f, t, v, h):
   k1 <- h * f(???, ???) // lo mismo que en el método de Euler
   k2 <- h * f(t + h, v + k1)
   v + (k1 + k2) / 2
```

**3. Fórmulas de Ralston para un paso.**

```
defun ralstonstep(f, t, v, h):
   k1 <- ??? // lo mismo que antes
   k2 <- h * f(t + 2 * h / 3, v + 2 * k1 / 3)
   v + (k1 + 3 * k2) / 4
```

**4. Esquema de los métodos de un paso.**

```
defun onestepmethod(f, tmin, tmax, x0, onestepformula, n):
   v <- zeroarray(n + 1); v[0] <- ???; h <- ???
   t <- tmin + h * range(n + 1)
   for j in range(n):
      v[j + 1] <- onestepformula(???, ???, ???, ???)
   (t, v)
```

Se supone que el argumento `onestepformula` es el apuntador a una función del mismo tipo que las funciones `heunstep` y `ralstonstep`. Se supone que la función interna `range` construye y devuelve el arreglo de los números $0, 1, \ldots, n-1$. En algunos lenguajes hay otras funciones para generar progresiones aritméticas.

**5. Pruebas.**

```
defun testmethod(onestepformula, n):
   (tmin, tmax, x0) <- (0, 1, 1)
   starttime <- currenttime()
   (t, xeuler) <- onestepmethod(f1, ???, ???, ???, onestepformula, n)
   elapsedtime <- currenttime() - starttime
   maxerror <- max(abs(x1(t) - xeuler))
   (elapsedtime, maxerror)
```

Se recomienda ejecutar esta función con varios valores de $n$ ($n = 10, 100, \ldots, 1000000$) y observar el comportamiento del error y del tiempo:

```
testmethod(heunstep, 10)
testmethod(heunstep, 100)
...
```

En algunos lenguajes se utiliza un operador especial para crear el apuntador a una función. Por ejemplo, en Matlab y GNU Octave se puede escribir `testmethod(@heunstep, 10)`. Luego haga pruebas similar para el método de Ralston.

**6. Análisis del tiempo de ejecución.** En este ejercicio elegimos $n$ de tal manera que el tiempo de ejecución sea al menos $0.1$ segundos. ¿Cómo se cambia el tiempo de ejecución cuando $n$ se multiplica por $10$?

**7. El número de evaluaciones de la función f.** ¿Cuántas evaluaciones de $f$ se necesitan en un paso del método de Heun? ¿Y en un paso del método de Ralston? ¿Cuántas veces se evalúa $f$, si el intervalo está dividido en $n$ partes?

**8. El comportamiento del error para los métodos de Heun y de Ralston.** Calcule los errores $\varepsilon_n$ para $n = 10, 10^2, \ldots, 10^6$:

```
defun testerror():
   ns <- [10, 100, 1000, 10000, 100000, 1000000]; eps <- zeroarray(6)
   for j in range(6):
      eps[j] <- testmethod(heunstep, ns[j])[1]
   eps
```

También calcule y muestre los cocientes $\dfrac{\varepsilon_{10}}{\varepsilon_{100}}$, $\dfrac{\varepsilon_{100}}{\varepsilon_{1000}}$, $\ldots$, $\dfrac{\varepsilon_{100000}}{\varepsilon_{1000000}}$.
¿Cómo se cambia el error cuando $n$ se multiplica por $10$?
Haga pruebas similares para el método de Ralston.