Programación: un par de métodos de Runge–Kutta de cuarto order para resolver EDO's con valores iniciales

Objetivos. Programar un par de métodos de Runge–Kutte de cuarto order 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. Esquema general de los métodos explícitos de Runge–Kutta de cuarto orden.

$$v_{\text{siguiente}} = v + b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4,$$

donde

$$\begin{aligned}
k_1 &= h\, f(t + c_1 h, v),\\
k_2 &= h\, f(t + c_2 h, v + a_{2,1} k_1),\\
k_3 &= h\, f(t + c_3 h, v + a_{3,1} k_1 + a_{3,2} k_2),\\
k_4 &= h\, f(t + c_4 h, v + a_{4,1} k_1 + a_{4,2} k_2 + a_{4,3} k_3).
\end{aligned}$$

Es cómodo escribir los coeficientes en una tabla (tabla de Butcher):

c_1				
c_2	$a_{2,1}$			
c_3	$a_{3,1}$	$a_{3,2}$		
c_4	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	
	b_1	b_2	b_3	b_4

2. El método Runge–Kutta clásico de orden 4 y la regla 3/8 de Kutta, también de orden 4.

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

0				
1/3	1/3			
2/3	−1/3	1		
1	1	−1	1	
	1/8	3/8	3/8	1/8

3. Programar un paso del método RK clásico y un paso de la regla 3/8 de Kutta. Programamos funciones que calculan una aproximación del valor x_{j+1} a partir de t_j, x_j y h, usando la función f:

```
defun rk41step(f, t, v, h):
   k1 <- h * f(???, ???) // lo mismo que en el método de Euler
   k2 <- h * f(t + h / 2, v + k1 / 2)
   k3 <- h * f(???, ???)
   k4 <- h * f(t + h, v + k3)
   v + (k1 + 2 * k2 + 2 * k3 + k4) / 6

defun rk42step(f, t, v, h):
   k1 <- ???
   k2 <- ???
   k3 <- h * f(t + 2 * h /3, v - k1 / 3 + k2)
   k4 <- ???
   v + (k1 + 3 * k2 + ??? * k3 + ???) / 8
```

4. Esquema general de los métodos de un paso. Si no lo han hecho antes, sugiero programar ahora el **esquema general** 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 `rk41step` y `rk42step`. 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. Haga pruebas de los métodos programados, como en las clases anteriores.

6. El número de evaluaciones de la función f. ¿Cuántas evaluaciones de f se necesitan en un paso del método RK41? ¿Y en un paso del método de RK42? ¿Cuántas veces se evalúa f, si el intervalo está dividido en n partes?

7. El comportamiento del error para los métodos RK41 y RK42. Usando esquemas de las clases anteriores, calcule los errores ε_n para $n = 10, 10^2, 10^3, 10^4$ en el método RK41. También calcule y muestre los cocientes $\dfrac{\varepsilon_{10}}{\varepsilon_{100}}$, $\dfrac{\varepsilon_{100}}{\varepsilon_{1000}}$, $\dfrac{\varepsilon_{1000}}{\varepsilon_{10000}}$.
¿Cómo se cambia el error cuando n se multiplica por 10?
Haga pruebas similares para el método RK42.