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

Objetivos. Programar un par de métodos de Runge–Kutte de quinto 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 quinto orden, con cinco evaluaciones. Los números $k_1, \ldots, k_5, v_{\text{siguiente}}$ se calculan de la siguiente manera, y los coeficientes se guardan en una tabla (tabla de Butcher):

$$
\begin{aligned}
k_1 &= h\, f(t + c_1 h, v), \\
k_2 &= h\, f(t + c_2 h, v + a_{2,1} k_1), \\
k_3 &= h\, f(t + c_3 h, v + a_{3,1} k_1 + a_{3,2} k_2), \\
k_4 &= h\, f(t + c_4 h, v + a_{4,1} k_1 + a_{4,2} k_2 + a_{4,3} k_3), \\
k_5 &= h\, f(t + c_5 h, v + a_{5,1} k_1 + a_{5,2} k_2 + a_{5,3} k_3 + a_{5,4} k_4), \\
v_{\text{sig}} &= v + b_1 k_1 + b_2 k_2 + b_3 k_3 + b_4 k_4 + b_5 k_5.
\end{aligned}
$$

c_1					
c_2	$a_{2,1}$				
c_3	$a_{3,1}$	$a_{3,2}$			
c_4	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$		
c_5	$a_{5,1}$	$a_{5,2}$	$a_{5,3}$	$a_{5,4}$	
	b_1	b_2	b_3	b_4	b_5

2. Dos métodos explícitos de Runge–Kutta de orden 5. El primero de estos dos métodos fue propuesto por Butcher, y el segundo por Merson. Nótese que el método de Butcher necesita 6 evaluaciones de la función f (hay que aumentar el esquema escrito arriba), y el método de Merson necesita solamente 5 evaluaciones de la función f (se aplica el esquema escrito arriba).

0						
1/4	1/4					
1/4	1/8	1/8				
1/2	0	−1/2	1			
3/4	3/16	0	0	9/16		
1	−3/7	2/7	12/7	−12/7	8/7	
	$\frac{7}{90}$	0	$\frac{32}{90}$	$\frac{12}{90}$	$\frac{32}{90}$	$\frac{7}{90}$

0					
1/3	1/3				
1/3	1/6	1/6			
1/2	1/8	0	3/8		
1	1/2	0	−3/2	2	
	$\frac{1}{6}$	0	0	$\frac{4}{6}$	$\frac{1}{6}$

3. Programar un paso del método de Butcher y un paso del método de Merson. 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 rk51step(f, t, v, h):
   k1 <- h * f(???, ???) // lo mismo que en el método de Euler
   k2 <- h * f(t + h / 4, v + k1 / 4)
   k3 <- h * f(???, ???)
   k4 <- h * f(???, ???)
   k5 <- h * f(???, ???)
   k6 <- h * f(t + h, v + (-3 * k1 + ???) / 7)
   v + (7 * k1 + ??? * k3 + ??? * k4 + ??? * k5) / 90
```

```
defun rk52step(f, t, v, h):
   k1 <- ???
   k2 <- h * f(t + h / 3, v + k1 / 3)
   ...
   k5 <- h * f(???, ???)
   v + (???) / 6
```

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 `rk51step` y `rk52step`. 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. El número de evaluaciones de la función f. ¿Cuántas evaluaciones de f se necesitan en un paso del método RK51? ¿Y en un paso del método de RK52? ¿Cuántas veces se evalúa f, si el intervalo está dividido en n partes?

6. El comportamiento del error para los métodos RK51 y RK52. Haga pruebas de los métodos programados, como en las clases anteriores. Usando esquemas de las clases anteriores, calcule los errores ε_n para $n = 10, 20, 40, 80$ en el método RK51. También calcule y muestre los cocientes $\dfrac{\varepsilon_{10}}{\varepsilon_{20}}$, $\dfrac{\varepsilon_{20}}{\varepsilon_{40}}, \dfrac{\varepsilon_{40}}{\varepsilon_{80}}$.
¿Cómo se cambia el error cuando n se multiplica por 2?
Haga pruebas similares para el método RK52.