Programación: métodos directos de pasos dobles para resolver EDO's con valores iniciales

Objetivos. Programar un par de métodos de pasos múltiples para resolver ecuaciones diferenciales con condiciones iniciales.

Requisitos. Haber programado varios métodos de Runge–Kutta.

1. Ejemplos y algoritmos de las clases anteriores. En las clases anteriores ya hemos programado un ejemplo `f1` de función del lado derecho de ecuación diferenciales, un ejemplo `x1` de solución correspondiente, y algunos esquemas que realizan un paso usando un valor anterior de la función, incluso el método de Heun o el método del punto promedio.

Idea de métodos de paso múltiple

2. Idea. Se calcula la función `f` (el lado derecho de la ecuación) en varios puntos (t_j, v_j) calculados previamente, y se utiliza una combinación lineal de estos valores de `f` para aproximar la pendiente actual. Por ejemplo, en el método de Adams–Bashforth de dos pasos,

$$v_j = v_{j-1} + \frac{3}{2}hf(t_{j-1}, v_{j-1}) - \frac{1}{2}hf(t_{j-2}, v_{j-2}). \tag{1}$$

Notamos que $f(t_5, v_5)$ se utiliza tanto en el cálculo de v_6 como en el cálculo de v_7, por eso guardaremos $f(t_j, v_j)$ en un arreglo. Si empezamos con un valor dado v_0, entonces para el cálculo de v_1 no podemos utilizar (1), y utilizamos algún metodo de Runge–Kutta de orden dos, por ejemplo, el método de Heun o el método del punto promedio.

3. Una iteración del método de Adams–Bashforth del paso doble. Supongamos que los dos valores $f(t_{j-2}, v_{j-2})$ y $f(t_{j-1}, v_{j-1})$ están dados como `fvalues[j - 2]` y `fvalues[j - 1]`, respectivamente. Entonces la fórmula (1) se puede realizar de la siguiente manera, usando el producto punto (renglón por columna):

```
abcoefs <- [-1/2, 3/2]
v[j] <- v[j - 1] + h * abcoefs * [fvalues[j - 2]; fvalues[j - 1]]
```

4. Ejercicio: entender los primeros pasos. Supongamos que los valores $f(t_j, v_j)$ se guardan en un arreglo `fvalues`. Determinar qué parte de este arreglo se necesaria en cada uno de los primeros pasos del método de Adams–Bashforth con dos puntos anteriores:

```
abcoefs <- [???, ???]
fvalues[0] <- f(t[0], v[0])
v[1] <- heunstep(f, t[???], v[???], h)
fvalues[1] <- f(t[???], v[???])
# empezamos el ciclo principal:
v[2] <- v[???] + h * abcoefs * [fvalues[???]; fvalues[???]]
fvalues[2] <- ???
v[3] <- ???
fvalues[3] <- ???
```

5. Función que realiza el método de paso doble, usando dos valores anteriores. y `adamsbashforthcoefs2` es el arreglo de los coeficientes de algún método de paso doble, digamos `adambashforth2`. En el arreglo `fvalues` guardaremos los valores $f(t_j, v_j)$.

```
defun twostepmethod(f, tmin, tmax, x0, n):
   abcoefs <- [???, ???]
   h <- ???;    t <- tmin + h * range(n + 1)
   v <- zeroarray(n + 1); v[0] <- ???
   fvalues <- zeroarray(n)
   fvalues[0] <- f(t[0], v[0])
   v[1] <- heunstep(f, ???, ???, ???)
   fvalues[1] <- f(???, ???)
   # ahora todo esta preparado, empezamos el ciclo principal:
   for j in range(2, n + 1):
      v[j] <- v[???] + h * ??? * [fvalues[???]; fvalues[???]])
      fvalues[j] <- f(???, ???)
   (t, v)
```

En este pseudocódigo se supone que los índices empiezan desde `0` y que la expresión `range(5, 9)` regresa `[5, 6, 7, 8]`. Recordamos que en el lenguaje Matlab/Octave la expresión `5 : 9` regresa `[5, 6, 7, 8, 9]`, y para construir un vector columna de longitud 5 se escribe `zeros(5, 1)`.

6. Prueba.

```
defun testtwostepmethod(n):
   (tmin, tmax, x0) <- (0, 1, 1)
   starttime <- currenttime()
   (t, xapprox) <- twostepmethod(f1, ???, ???, ???, n)
   elapsedtime <- currenttime() - starttime
   maxerror <- max(abs(x1(t) - xapprox))
   (elapsedtime, maxerror)
```

Ahora podemos ejecutar las funciones programadas:

```
testtwostepmethod(100)
```