Programación: el método de Euler para resolver numéricamente una EDO con un valor inicial

Objetivos. Programar el método de Euler para resolver ecuaciones diferenciales ordinarias con condiciones iniciales. Hacer comprobaciones.

1. Aclaración sobre el pseudocódigo. Los borradores de programas están escritos en un pseudocódigo que mezcla ideas de varios lenguajes (Python, Matlab, etc.). En el pseudocódigo suponemos que los índices de arreglos empiezan con 0. Es importante elegir algún lenguaje de programación real, escribir programas en este lenguaje, ejecutarlos, depurar los errores, etc.

2. Ejemplo. Desde el inicio elegimos un ejemplo. Vamos a resolver la ecuación diferencial

$$x'(t) = \frac{2x(t)}{3+t} \tag{1}$$

con la condición inicial

$$x(0) = 1.$$

Se sabe que la solución exacta es

$$x(t) = \frac{1}{9}(3+t)^2.$$

3. Programar el lado derecho de la ecuación y la solución exacta. Programamos una función `f1` que calcula el lado derecho de la ecuación (1):

```
defun f1(t, x):
   2 * x / (3 + t)
```

Programamos una función `x1` que calcula la solución exacta de la ecuación en los puntos dados:

```
defun x1(t):
   ((3 + t) .^ 2) / 9
```

Aquí se supone que `t` es un arreglo, y la operación "elevar al cuadrado" se aplica a cada componente del arreglo. En algunos lenguajes de programación se utiliza la operación `.^` para elevar cada componente del arreglo a la potencia dada.

4. Fórmula de Euler 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 eulerstep(f, t, v, h):
   v + ??? * f(???, ???)
```

Aquí se supone que f es un apuntador a una función, t, v, h son algunos números.

5. Esquema de los métodos de un paso para resolver ecuaciones diferenciales ordinarias con ecuaciones iniciales. Programamos el esquema general:

```
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 la función `eulerstep` programada anteriormente. 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. Por ejemplo, se recomienda usar la función `linspace` en Matlab, GNU Octave, Python + numpy, Julia.

6. 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(eulerstep, 10)
testmethod(eulerstep, 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(@eulerstep, 10)`.

7. 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?

8. El número de evaluaciones de la función f. Por lo común la mayor parte del tiempo se gasta en la evaluación de la función f que está en el lado derecho de la ecuación $x'(t) = f(t, x(t))$. ¿Cuántas evaluaciones de f se necesitan en un paso del método de Euler? ¿Cuántas veces se evalúa f, si el intervalo está dividido en n partes?

9. El comportamiento del error. Calcule los errores ε_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(eulerstep, ns[j])[1]
   eps
```

También calcule y muestre los cocientes

$$\frac{\varepsilon_{10}}{\varepsilon_{100}}, \quad \frac{\varepsilon_{100}}{\varepsilon_{1000}}, \quad \ldots, \quad \frac{\varepsilon_{100000}}{\varepsilon_{1000000}}.$$

¿Cómo se cambia el error cuando n se multiplica por 10?