Programación: evaluación de un polinomio en un punto con varios algoritmos

Objetivos. Realizar varios algoritmos que calculan el polinomio dado por su lista de coeficientes en el punto dado.

Requisitos. Programación con funciones, ciclos y arreglos.

1. Convenio: guardar polinomios como arreglos de coeficientes, empezando los índices de 1. Por ejemplo, la forma general de polinomio de grado ≤ 3 es

$$f(t) = a_1 + a_2 t + a_3 t^2 + a_4 t^3. \tag{1}$$

Guardamos este polinomio como un arreglo a cuyas componentes son a_1, a_2, a_3, a_4. Estos ejercicios están orientados al lenguaje Matlab/Octave, pero se pueden modificar ligeramente y servir para Wolfram Mathematica, R, Julia y otros lenguajes.

2. Evaluar polinomios con el algoritmo ingenuo. Mostramos la idea con el caso particular cuando el polinomio es de la forma (1), esto es, la longitud del arreglo de coeficientes es 4.

```
v = 0;
v = v + a(1) * (t ^ 0);
v = v + a(2) * (t ^ 1);
v = v + a(3) * (t ^ 2);
v = v + a(4) * (t ^ 3);
```

Generelice esta idea al caso general (cuando el arreglo de coeficientes es de longitud general n), usando un ciclo `for`. Escriba una función `poleval1` y guárdela en el archivo `poleval1.m`:

```
function [v] = poleval1(a, t),
   n = length(a);
   v = ???;
   for j = 1 : n,
      v = v + ???;
   endfor
endfunction
```

En el intérprete de Matlab/Octave haga una comprobación:

```
poleval1([3; -1; 5; 2; 1], -3)
```

La función debe devolver el número 78.

3. Evaluar polinomios acumulando simultáneamente las potencias. Vamos a acumular las potencias en una variable auxiliar `p`:

```
v = 0;
p = 1;
v = v + a(1) * p;
p = p * t; # ahora p = t ^ 1
v = v + a(2) * p;
p = p * t; # ahora p = t ^ 2
v = v + a(3) * p;
p = p * t; # ahora p = t ^ 3
v = v + a(4) * p;
```

Escriba una función `poleval2` que realiza esta idea y haga una comprobación.

4. Evaluar polinomios con el algoritmo de Horner. Representamos el polinomio (1) en la forma

$$f(t) = (((0\,t + a_4)\,t + a_3)\,t + a_2)\,t + a_1.$$

Denotamos por v_4, v_3, v_2, v_1, v_0 a los valores intermedios:

$$\begin{aligned}
v_4 &= 0;\\
v_3 &= v_4 t + a_4 = a_4;\\
v_2 &= v_3 t + a_3 = a_4 t + a_3;\\
v_1 &= v_2 t + a_2 = a_4 t^2 + a_3 t + a_2;\\
v_0 &= v_1 t + a_1 = a_4 t^3 + a_3 t^2 + a_2 t + a_1.
\end{aligned}$$

En el programa guardamos cada uno de los valores v_4, v_3, v_2, v_1, v_0 en una variable `v`, olvidando los valores anteriores:

```
v = 0;
v = v * t + a(4);
v = v * t + a(3);
v = v * t + a(2);
v = v * t + a(1);
```

Escriba una función `poleval3` que realiza esta idea y haga una comprobación.

5. Matriz de Vandermonde asociada a un punto. Consideramos el polinomio (1). El número $f(t)$ se puede calcular como el siguiente producto de matrices; más precisamente, un renglón se multiplica por una columna:

$$f(t) = \begin{bmatrix} t^0 & t^1 & t^2 & t^3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}.$$

La fila $[t^0, t^1, t^2, t^3]$ se puede construir con los siguientes comandos:

```
p = zeros(4, 1);
p(1) = 1;
p(2) = p(1) * t;
p(3) = p(2) * t;
p(4) = p(3) * t;
```

En lenguajes de nivel alto, como Matlab o GNU Octave, resultan ser más rápidas operaciones matriciales:

```
p = t .^ (0 : 3);
```

Para polinomios de longitud n es necesario calcular las potencias de 0 a $n-1$. Escriba la función correspondiente. Utilice un ciclo o un comando matricial:

```
function [p] = vandermonde(t, n),
   ???
endfunction
```

Escriba una función que calcula el valor del polinomio dado en el punto dado usando el arreglo que devuelve la función `vandermonde`:

```
function [v] = poleval4(a, t),
   n = length(a);
   p = vandermonde(???, ???);
   v = p * a;
endfunction
```

Haga una comprobación.

6. Prueba con un polinomio de grado pequeño. Probemos que las funciones programadas devuelven el mismo valor, cuando se aplican a los mismos datos:

```
function [] = test1poleval(),
   a = [3; -1; 5; 2; 1];
   t = -3;
   v1 = poleval1(a, t);
   v2 = poleval2(a, t);
   v3 = poleval3(a, t);
   v4 = poleval4(a, t);
   display([v1, v2, v3, v4]);
endfunction
```

7. Pruebas con polinomios de grado grande. Generamos un polinomio pseudoaleatorio y un punto pseudoaleatorio. Calculamos el valor de este polinomio con cada uno de los cuatro algoritmos. Medimos el tiempo de ejecución y el error relativo. Como el tiempo de ejecución es muy pequeño, repetimos el mismo cálculo muchas veces.

```
function [] = test2poleval(n, nrep),
   a = randn(n, 1); t = randn(1, 1);
   t1 = cputime();
   for r = 1 : nrep,
      v1 = poleval1(a, t);
   endfor
   t1 = cputime() - t1;
   t2 = cputime();
   for r = 1 : nrep,
      v2 = poleval2(a, t);
   endfor
   t2 = cputime() - t2;
   t3 = cputime();
   for r = 1 : nrep,
      v3 = poleval3(a, t);
   endfor
   t3 = cputime() - t3;
   t4 = cputime();
   for r = 1 : nrep,
      v4 = poleval4(a, t);
   endfor
   t4 = cputime() - t4;
   times = [t1, t2, t3, t4];
   abserrors = abs([v2 - v1, v3 - v1, v4 - v1]);
   relerrors = abserrors / abs(v1);
   display(times);
   display(relerrors);
endfunction
```