Lista de problemas:
ciclos y funciones en el lenguaje MATLAB

Esta lista de problemas está compuesta para probar si el estudiante tiene ciertas habilidades básicas de programación con arreglos, funciones y ciclos `for` y `while`.

En cada uno de los problemas hay que escribir una función que regrese lo que se pide y que no escriba nada en la pantalla. Se puede usar la sintaxis del lenguaje Matlab o de sus análogos libres (GNU Octave, Scilab, FreeMat). En los ejemplos se usa la sintaxis de GNU Octave.

1. Ejemplo. Escribir una función `log7` de un argumento x que calcule el logaritmo de x en base 7.

ENTRADA: un número x (se supone que $x > 0$).

SALIDA: $\log_7(x)$.

EJEMPLO: `log7(49)` debe regresar `2`.

Solución. La solución es un archivo `log7.m` en el formato "texto simple". El archivo puede tener el siguiente contenido:

```
function y = log7(x)
   y = log(x) / log(7);
endfunction
```

□

2. Ejemplo. Escribir una función `factorials` que construya y regrese el vector que consista de los factoriales de los números $1, \ldots, n$.

ENTRADA: un número n (se supone que n es entero y $n \geq 1$).

SALIDA: un vector que consiste de los factoriales de los números $1, \ldots, n$.

EJEMPLO: `factorials(5)` debe regresar el vector `[1; 2; 6; 24; 120]`.

Solución. Una solución posible se puede escribir en el archivo `factorials.m` de la siguiente manera:

```
function f = factorials(n)
   f = ones(n, 1);
   for j = 2 : n,
      f(j) = f(j - 1) * j;
   endfor
endfunction
```

Otra solución de este mismo problema sería el comando `factorial(1 : n)'`. El propósito de esta lista de problemas es probar si el estudiante sabe programar funciones con ciclos, por eso se pide una solución similar a la primera, y la segunda se puede indicar como algo adicional. □

Ciclo for

3. Problam (sumarray). Escribir una función `sumarray` que calcule la suma de las componentes del vector dado. El sentido de este problema es practicar la programación con ciclos.

ENTRADA: un vector a.

SALIDA: la suma de los elementos de a: $\sum_{k=1}^{n} a_k$, donde n es la longitud de a.

EJEMPLO: `sumarray([5; -1; 7; 2])` debe devolver `13`.

4. Problema (mydotproduct). Escribir una función `mydotproduct` que calcule el producto punto de dos vectores dados. El sentido de este problema es practicar la programación con ciclos.

ENTRADA: dos vectores a y b reales de la misma longitud (no es necesario verificar que tienen la misma longitud).

SALIDA: el producto interno canónico de los vectores a y b.

EJEMPLO: `mydotproduct([3; 4; 5], [11; 12; 13])` debe regresar `146`.

5. Problema (accumulate). Escribir una función `accumulate` que calcule las sumas parciales de las entradas del vector dado.

ENTRADA: un vector a.

SALIDA: un vector b de la misma longitud que a y tal que para cada $k \in \{1, \ldots, n\}$ la k-ésima entrada de b es la suma de las primeras k entradas del vector a.

EJEMPLO: `accumulate([7; -2; 8])` debe regresar el vector `[7; 5; 13]`.

6. Problema (factorial). Escribir una función `fa` que calcule el factorial del número dado.

ENTRADA: un número n; se supone que n es entero y no negativo.

SALIDA: el número $n!$.

EJEMPLO: `fa(4)` debe regresar `24`.

7. Problema (fib). Escribir una función `fib` que calcule los primeros n elementos de la sucesión de Fibonacci, empezando con 0 y 1.

ENTRADA: un número entero positivo n ($n \geq 2$).

SALIDA: un vector v de longitud que n cuyos elementos son los primeros n elementos de la sucesión de Fibonacci, esto es,

$$\begin{aligned} v_1 &= 0; \\ v_2 &= 1; \\ v_k &= v_{k-1} + v_{k-2} \quad (k \geq 3). \end{aligned}$$

EJEMPLO: `fib(7)` debe regresar el vector `[0; 1; 1; 2; 3; 5; 8]`.

8. Problema (mulpolbinom). Calcular los coeficientes $c_1, \ldots, c_{n+1}$ del producto de un polinomio por un binomio mónico:

$$(a_1 + a_2x + \ldots + a_nx^{n-1})(b + x) = c_1 + c_2x + \ldots + c_{n+1}x^n.$$

ENTRADA: un vector a y un número b (suponemos que el binomio es mónico, por eso sólo pasamos a la función el coeficiente de x^0).

SALIDA: el vector c de los coeficientes del producto.

EJEMPLO: `mulpolbinom([3; -5; 1], 2)` debe regresar el vector `[6; -7; -3; 1]` porque $(3 - 5x + x^2)(2 + x) = 6 - 7x - 3x^2 + x^3$.

9. Problema (poleval). Calcular el valor del polinomio

$$f(t) = a_1 + a_2t + \cdots + a_nt^{n-1}$$

en el punto dado b.

ENTRADA: un vector a y un número b.

SALIDA: el número $a_1 + a_2b + \cdots + a_nb^{n-1}$.

EJEMPLO: `poleval([4; -1; -6; 3], 3)` debe regresar el número `28`.

10. Problema (poleval, versión vectorial). Calcular los valores del polinomio

$$f(t) = a_1 + a_2t + \cdots + a_nt^{n-1}$$

en cada elemento del vector dado b.

ENTRADA: dos vectores a y b.

SALIDA: el vector con las componentes $f(b_1), \ldots, f(b_m)$, donde m es la longitud de b.

EJEMPLO: `poleval([4; -1; -6; 3], [3; -2; 7]` debe devolver `[28; -42; 96]`.

11. Problema (myproductmatvec). Escribir una función `myproductmatvec` que calcule el producto de una matriz por un vector. Se puede suponer (y no comprobar) que la longitud del vector dado coincide con el número de columnas de la matriz dada. Hay que escribir su propia realización de la operación $a * b$.

ENTRADA: una matriz a y un vector b; se supone que la longitud de b coincide con el número de columnas de a.

SALIDA: el producto de la matriz a por el vector b.

EJEMPLO: `myproductmatvec([3, 5, -1; 2, 0, 6], [-3; 7; 4])` debe devolver el vector `[22; 18]`.

12. Problema (vandermatrix). Escribir una función `vandermatrix` que construya y regrese la matriz de Vandermonde de tamaño $n \times n$ asociada a los números dados $x_1, \ldots, x_n$.

ENTRADA: un vector x.

SALIDA: la matriz de Vandermonde cuadrada asociada al vector x.

EJEMPLO: `vandermatrix([3, -2, 0, 5])` debe regresar la matriz

$$\begin{bmatrix} 1 & 3 & 9 & 27 \\ 1 & -2 & 4 & -8 \\ 1 & 0 & 0 & 0 \\ 1 & 5 & 25 & 125 \end{bmatrix}.$$

13. Problema (solvelt1). Escribir una función `solvelt1` que resuelva el sistema $Lx = b$ con una matriz L unitriangular inferior. Una matriz cuadrada se llama *unitriangular inferior* si es triangular inferior y todas sus entradas diagonales son iguales a uno.

ENTRADA: una matriz unitriangular inferior L y un vector b cuya longitud coincide con el orden de la matriz L.

SALIDA: un vector x tal que $Lx = b$.

EJEMPLO: `solvelt1([1, 0, 0; -2, 1, 0; 3, -1, 1], [4; -7; 8])` debe regresar el vector `[4; 1; -3]` porque

$$\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \\ 8 \end{bmatrix}.$$

14. Problema (solveut). Escribir una función `solveut` que resuelva el sistema $Ux = b$, donde U es una matriz cuadrada triangular superior con entradas diagonales no nulas.

ENTRADA: una matriz cuadrada triangular superior U y un vector b cuya longitud coincide con el orden de la matriz U.

SALIDA: un vector x tal que $Ux = b$.

EJEMPLO: `solveut([3, -1, 5; 0, -2, 5; 0, 0, 4], [3; -8; -8])` debe regresar el vector `[4; -1; -2]` porque

$$\begin{bmatrix} 3 & -1 & 5 \\ 0 & -2 & 5 \\ 0 & 0 & 4 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \\ -2 \end{bmatrix} = \begin{bmatrix} 3 \\ -8 \\ -8 \end{bmatrix}.$$

Ciclo while

Se recomienda resolver los siguientes problemas usando el ciclo `while`.

15. Problema (sumdigits). Escribir una función `sumdigits` que calcule la suma de los dígitos decimales del número dado.

ENTRADA: un número entero x.

SALIDA: la suma de los dígitos decimales de x.

EJEMPLO: `sumdigits(365)` debe regresar `14`.
Antes de escribir esta función, se recomienda probar los comandos `idivide(20, 3)` y `rem(20, 3)`.

16. Problema (gcd). Escribir una función `mygcd` que calcule el máximo común divisor de dos números enteros dados. Si ambos números son 0, hay que devolver 0.

ENTRADA: dos números enteros a y b.

SALIDA: el máximo común divisor de a y b; se puede suponer que $a \geq 0$ y $b \geq 0$.

EJEMPLO: `mygcd(30, 80)` debe devolver `10`.

EJEMPLO: `mygcd(0, 12)` debe devolver `12`.

EJEMPLO: `mygcd(0, 0)` debe devolver `0`.

17. Problema (countfirstnegatives). Escribir una función `countfirstnegatives` que calcule el número de las entradas negativas (seguidas) en el inicio del vector dado.

ENTRADA: un vector a.

SALIDA: $\max\{j \in \{1, \ldots, n\}: \quad \forall k \in \{1, \ldots, j\} \quad a_k < 0\}$, donde n es la longitud de a.

EJEMPLO: `countfirstnegatives([-7; -5; -5; 4; -6])` debe regresar `3`.

EJEMPLO: `countfirstnegatives([4; -1; -6; -8; -2])` debe regresar `0`.

EJEMPLO: `countfirstnegatives([-1; -1; -3; -6])` debe regresar `4`.

18. Problema (myfixedpoint). Escribir una función `myfixedpoint` que resuelva con el método del punto fijo (llamado también el *método de iteración simple*) la ecuación $x = a\exp(-x)$, donde a es un parámetro positivo.

ENTRADA: un número positivo a.

SALIDA: un número positivo x tal que $x = a\exp(-x)$.

EJEMPLO: `myfixedpoint(2.0)` debe regresar `0.85261`.

19. Problema (solvemyeq). Escribir una función `solvemyeq` que resuelva con el método de la bisección (o de la regla falsa) la ecuación $\cos(x) - ax = 0$ en el intervalo de 0 a $\pi/2$.

ENTRADA: un número positivo a.

SALIDA: un número $x \in (0, \pi/2)$ tal que $\cos(x) - ax = 0$.

EJEMPLO: `solvemyeq(2.0)` debe regresar `0.45018`.

EJEMPLO: `solvemyeq(0.5)` debe regresar `1.0299`.