Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Laboratorio de Simulación
Diego Sarceño 201900109
5 de mayo de 2022

Práctica 4

Índice

1. Problema 1

1.1. Enunciado

Debe de ingresar un vector de 10 elementos, llenarlo de números pares del 2 al 20. Al iniciar el programa debe preguntar al usuario como quiere ver los números, el menú debe de ser por medio de caracteres: “a”verlos de forma ascendente, “d”descendente, en caso que el usuario escriba otro valor debe de decir que no es correcto y preguntarle el carácter nuevamente, hasta que este sea el correcto, al ingresar el valor correcto muestra el vector en pantalla y termina el programa.

1.2. Metodología

Por un error de lectura del problema se realizó de modo que la única interacción con el usuario sea la parte de ”ascendente” o ”descendente”. De este modo, se genera una lista con los números pares del 2 al 20 y se realiza un método *shuffle* para randomizar la posición de los elementos de la lista, se utiliza como **seed** el ”time(0)”. Con esto, se utiliza la el método de ordenamiento merge sort. Ya dependiendo de la entrada del usuario se invierte el orden de la lista resultante o no.

1.3. Variables de entrada y salida

→: Caracter ”A” o ”D”.

←: lista de números.

1.4. Pseudocógido o Diagrama de Flujo

1.4.1. Algoritmo Merge Sort

Función MergeSort(vector, incio, final)

Paso 1: Definimos donde queremos ”partir” la lista: half = (inicio + final)/2

Paso 2: Se inicia la recursividad. Llamada a MergeSort(vector, inicio, half)

Paso 3: Llamada a MergeSort(vector, half + 1, final)

Paso 4: Llamada a Merge(vector, inicio, half, final)

Función Merge(vector, inicio, half, final)

Paso 1: Se define una lista auxiliar y los iteradores i = start, j = half + 1, k = 0 y t

Paso 2: Mientras $i \leq$ half && $j \leq$ final, hacer

- **Paso 2.1:** k++. Si vector[i] < vector[j] entonces auxiliar[k] = vector[i], i++.
- **Paso 2.2:** En otro caso auxiliar[k] = vector[j], j++.

Paso 3: Se agrega el elemento sobrante de alguna de las mitades al final de la lista auxiliar. (siempre habrá uno sobrante)

Paso 4: Reescribir todo en el vector original utlizando el contador ”t”.

1.4.2. Algoritmo Shuffle

Función shuffle(vector, length)

Paso 1: Se declaran variables locales "random_ position", "step" y "k = 0"

Paso 2: Mientras int $k <$ length hacer

- **Paso 2.1:** random_ position = numero entero random $\in [0, \text{length} - 1]$
- **Paso 2.2:** step = vector[k]
- **Paso 2.3:** vector[k] = vector[random_ position]
- **Paso 2.4:** vector[random_ position] = step

1.4.3. Main

Paso 1: Generación del vector con elementos pares del 2 al 20

Paso 2: Aplicamos *Shuffle* al vector para aleatorizarlo

Paso 3: Declaramos la variable de inicio del usuario y una variable control = 0

Paso 4: Mientras control == 0 hacer

- **Paso 4.1:** Solicitar el ingreso del caracter "A" o "D"
- **Paso 4.2:** Si es "A", aplicar MergeSort e imprimir el vector resultante. control = 1.
- **Paso 4.3:** Si es "D", aplicar MergeSort e imprimir desde el final del vector hasta el inicio. control = 1.
- **Paso 4.4:** Sino es ninguna, imprimir que no se ingresó ningún valor válido.

1.5. Código

```
1  //      Librerias
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  // 0. Prototipado de funciones
7  void mergeSort(int main_array[], int start, int final);
8  void merge(int array[], int start, int half, int final);
9  void reverse(int array[], int length);
10 void printArrays(int array[], int length);
11 void shuffle(int array[], int length);
12 #define l 10
13
14 // 1. Funcion main
15
16 int main(){
17   // 1.1. Definimos la "seed" como el tiempo, para el generador de
       numeros
```

```
18    srand(time(0));
19
20    // 2. Se crea un arreglo de numeros pares del 2 al 20
21    int numeros[l];
22    // 2.1. se ingresan los datos en la lista
23    for (int i = 0; i < l; i++){ numeros[i] = 2*i + 2; }
24
25    // 3. Se aleatoriza el arreglo
26    printArrays(numeros, l);
27    shuffle(numeros, l);
28    printArrays(numeros, l);
29
30    // 4. Interaccion con el usuario y ordenamiento del arreglo
31    char input;
32    int control = 0;
33    while(control == 0){
34      puts("Ingrese como desea ver los numeros.");
35      puts("'A' para verlos de forma ascendente.");
36      puts("'D' para verlos de forma descendente.");
37      scanf("%s",&input);
38      if (input == 65){
39        control = 1;
40        puts("WUUU FORMA ASCENDENTE");
41        //printArrays(numeros, l);
42        mergeSort(numeros, 0, l);
43        printArrays(numeros, l);
44      } else if (input == 68){
45        control = 1;
46        puts("WUUU FORMA DESCENDENTE");
47        //printArrays(numeros, l);
48        mergeSort(numeros, 0, l);
49        reverse(numeros, l);
50        printArrays(numeros, l);
51      } else {
52        puts("Ingrese un valor valido.");
53      } // END IF
54    } // END WHILE
55
56
57    return 0;
58  } // END MAIN
59
60
61  /*
62                                  FUNCIONES UTILIZADAS
63  */
64
65  void mergeSort(int main_array[], int start, int final){
```

```
66    int half;
67    half = (start + final)/2;
68    if (start < final){
69      mergeSort(main_array, start, half);
70      mergeSort(main_array, half + 1, final);
71      merge(main_array, start, half, final);
72    } // END IF
73  } // END MERGESORT
74
75  void merge(int array[], int start, int half, int final){
76    int aux[final + 1],i,j,k,t;
77
78    k = 0; // movimiento por la lista auxiliar
79    i = start; // movimiento por la sublista izquierda
80    j = half + 1; // movimiento por la sublista derecha
81
82    // ciclo para empezar a unir los arrays
83    while(i <= half && j <= final){
84      k++;
85      if (array[i] < array[j]){
86        aux[k] = array[i];
87        i++;
88      } else {
89        aux[k] = array[j];
90        j++;
91      } // END IF
92    } // END WHILE
93
94    // para los elementos sobrantes de alguna de las sublistas
95    for (t = i; t <= half; t++){
96      k++;
97      aux[k] = array[t];
98    } // END FOR
99
100   for (t = j; t <= final; t++){
101     k++;
102     aux[k] = array[t];
103   } // END FOR
104
105   // regresar todo al vector original
106   for (t = 1; t <= k; t++){
107     array[start + t - 1] = aux[t];
108   } // END FOR
109 } // END MERGE
110
111 void reverse(int array[], int length){
112   int aux[length];
113
```

```
114   // Navegando el array del final hacia el inicio, y aux en forma
          contraria
115   for (int i = length - 1; i >= 0; i--){
116     aux[length - i - 1] = array[i];
117   } // END FOR
118
119   // regresando todo al vector original
120   for (int j = 0; j < length; j++){
121     array[j] = aux[j];
122   } // END FOR
123 } // END INVERSE
124
125 void printArrays(int array[], int length){
126   // impresion
127   printf("(");
128   for (int i = 0; i < length; i++){
129     if (i != length - 1){
130       printf("%d, ",array[i]);
131     } else {
132       printf("%d)\n", array[i]);
133     } // END IF
134   } // END FOR
135 } // END printArrays
136
137 void shuffle(int array[], int length){
138   int random_position, step;
139   for (int k = 0; k < length; k++){
140     // Generamos una posicion random
141     random_position = rand() % length;
142     // intercambiar el actual elemento con el de la poiscion aleatoria
143     step = array[k];
144     array[k] = array[random_position];
145     array[random_position] = step;
146   } // END FOR
147 } // END SHUFFLE
148
149
150 // END PROGRAM
```

2. Problema 2

2.1. Enunciado

Crear un programa que solicite al usuario 5 números enteros, estos se deben de guardar en un vector, al terminar de guardar los valores, el programa debe de ordenarlos de forma ascendente y mostrar el vector ordenado. (utilice un método de ordenación.)

2.2. Metodología

Al igual que el problema anterior se utiliza el algoritmo merge sort, aunque se realizó una implementación de quick-sort y bubble-sort.

2.3. Variables de entrada y salida

→: 5 números **enterios** por parte del usuario.

←: lista ordenada de números.

2.4. Pseudocógido o Diagrama de Flujo

Paso 1: Prototipado de funciones y definición de constantes, en este caso l = 5.

Paso 2: int vector[l]

Paso 3: Ingreso de los números por parte del usuario

Paso 4: Ordenar el vector con el algoritmo Merge Sort mostrado en el problema anterior.

Paso 5: Impresión del vector ordenado.

2.5. Código

```
1  //      Librerias
2  #include <stdio.h>
3
4
5  // 0. Prototipado de funcion y definicion de variables
6  #define l 5
7  void mergeSort(int main_array[], int start, int final);
8  void merge(int array[], int start, int half, int final);
9  void printArrays(int array[], int length);
10
11 // 1. funcion main
12 int main(){
13   // 3. declaracion de variables
14   int vector[5];
15
16   // 4. ingreso de datos
17   puts("Ingrese␣5␣numeros␣enteros.");
18   for (int i = 0; i < l; i++){ scanf("%d", &vector[i]); }
19
20   // 5. Ordenamiento del array
21   puts("Array␣ordenado.");
22   mergeSort(vector, 0, l);
23   printArrays(vector, l);
24
25   return 0;
26 } // END MAIN
```

```
27
28
29 /*
30                                FUNCITON
31 */
32
33
34 void mergeSort(int main_array[], int start, int final){
35   int half;
36   half = (start + final)/2;
37   if (start < final){
38     mergeSort(main_array, start, half);
39     mergeSort(main_array, half + 1, final);
40     merge(main_array, start, half, final);
41   } // END IF
42 } // END MERGESORT
43
44 void merge(int array[], int start, int half, int final){
45   int aux[final + 1],i,j,k,t;
46
47   k = 0; // movimiento por la lista auxiliar
48   i = start; // movimiento por la sublista izquierda
49   j = half + 1; // movimiento por la sublista derecha
50
51   // ciclo para empezar a unir los arrays
52   while(i <= half && j <= final){
53     k++;
54     if (array[i] < array[j]){
55       aux[k] = array[i];
56       i++;
57     } else {
58       aux[k] = array[j];
59       j++;
60     } // END IF
61   } // END WHILE
62
63   // para los elementos sobrantes de alguna de las sublistas
64   for (t = i; t <= half; t++){
65     k++;
66     aux[k] = array[t];
67   } // END FOR
68
69   for (t = j; t <= final; t++){
70     k++;
71     aux[k] = array[t];
72   } // END FOR
73
74   // regresar todo al vector original
```

```
75    for (t = 1; t <= k; t++){
76      array[start + t - 1] = aux[t];
77    } // END FOR
78  } // END MERGE
79
80  void printArrays(int array[], int length){
81    // impresion
82    printf("(");
83    for (int i = 0; i < length; i++){
84      if (i != length - 1){
85        printf("%d, ",array[i]);
86      } else {
87        printf("%d)\n", array[i]);
88      } // END IF
89    } // END FOR
90  } // END printArrays
91
92
93  // END PROGRAM
```

3. Problema 3

3.1. Enunciado

Crear un programa que solicite al usuario dos posiciones en coordenadas (x,y,z) al obtenerlas debe de almacenarlas en dos vectores, el programa automáticamente debe de mostrar los siguientes resultados:

a) magnitud de cada vector

b) suma de los dos vectores

c) producto escalar

d) producto vectorial

3.2. Metodología

a) magnitud de cada vector: cálculo de magntidud de un vector

$$|\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2}.$$

b) suma de los dos vectores: para dos vectores $\vec{A} = (x_1, y_1, z_1)$ y $\vec{B} = (x_2, y_2, z_2)$.

$$\vec{A} + \vec{B} = (x_1 + x_2, y_1 + y_2, z_1 + z_2).$$

c) producto escalar: para los vectores anteriores

$$\langle \vec{A}, \vec{B} \rangle = x_1x_2 + y_1y_2 + z_1z_2.$$

d) producto vectorial, dados los vectores anteriores

$$\begin{aligned} &(y_1z_2 - z_1y_2)\hat{\mathbf{x}} \\ \vec{A} \times \vec{B} = &-(x_1z_2 - z_1x_2)\hat{\mathbf{y}}. \\ &+(x_1y_2 - y_1x_2)\hat{\mathbf{z}} \end{aligned}$$

3.3. Variables de entrada y salida

→: Vectores (x_1, y_1z_1) y (x_2, y_2, z_2)

←: float para la magnitud, array para la suma, float para el producto escalar y array para el producto vectorial.

3.4. Pseudocógido o Diagrama de Flujo

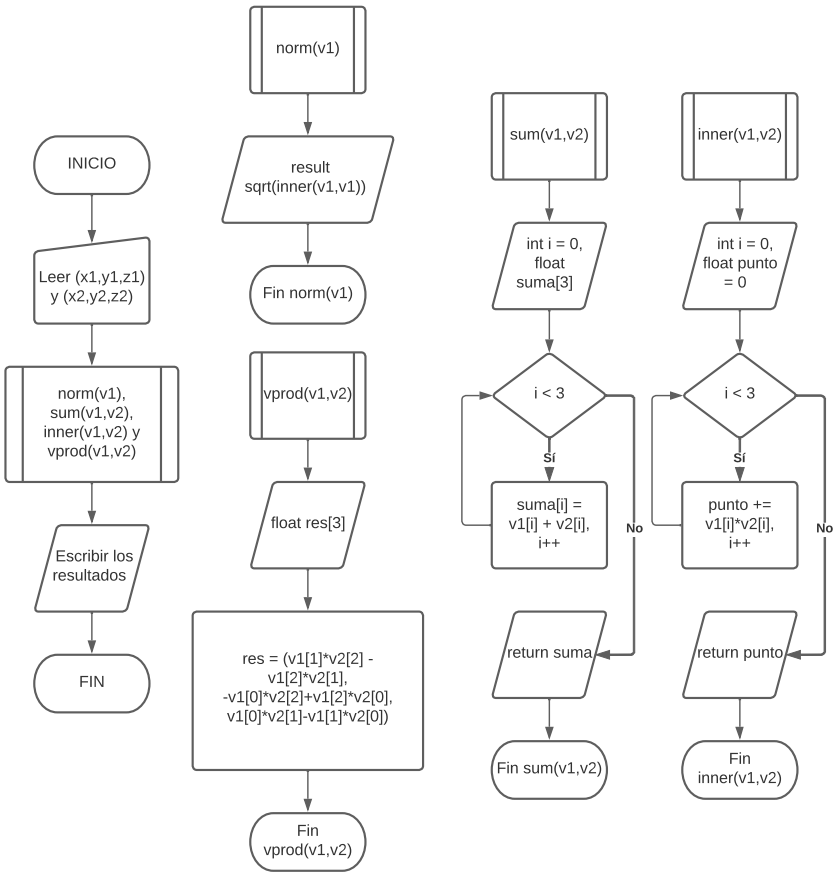


Figura 1: Diagrama de flujo de las diferentes operaciones con vectores.

3.5. Código

```
//      Librerias
#include <stdio.h>
#include <math.h>

// 0. Prototipado de funciones y declaracion de variables
//double sqrt(double x);
void printArrays(float array[], int length);
void suma_vectorial(float array1[], float array2[], float result[]);
float magnitud(float array[]);
float producto_interno(float array1[], float array2[]);
void producto_vectorial(float array1[], float array2[], float result[]);

// 1. Main function
int main(){
  // 2. declaracion de variables generales
```

```
16   float vec1[3], vec2[3];
17   float resvec[3];
18
19   // 3. Ingreso de los vectores por parte del usuario
20   puts("Ingrese␣dos␣vectores␣en␣R3␣en␣la␣forma␣x1,x2,x3.");
21   scanf("%f,%f,%f",&vec1[0],&vec1[1],&vec1[2]);
22   scanf("%f,%f,%f",&vec2[0],&vec2[1],&vec2[2]);
23
24   // 4. Realizando las diferente operaciones
25   // 4.1. Suma de vectores
26   suma_vectorial(vec1,vec2,resvec);
27   puts("Suma␣de␣vectores.");
28   printArrays(resvec, 3);
29   // 4.2. magnitud de cada vector
30   printf("Magnitud␣de␣cada␣vector.␣V1␣=␣%f,␣V2␣=␣%f\n", magnitud(vec1),
        magnitud(vec2));
31   // 4.3. vector product
32   producto_vectorial(vec1,vec2,resvec);
33   puts("Producto␣vectorial␣entre␣ambos␣vectores.");
34   printArrays(resvec, 3);
35   // 4.4. inner product
36   printf("Producto␣interno␣entre␣ambos␣vectores:␣%f\n",producto_interno(
        vec1,vec2));
37
38   return 0;
39 } // END MAIN
40
41
42 /*
43                             FUNCIONES
44 */
45
46 float producto_interno(float array1[], float array2[]){
47   float result = 0;
48
49   for (int i = 0; i < 3; i++){
50     result += array1[i]*array2[i];
51   } // END FOR
52
53   return result;
54 } // END PRODUCTO_INTERNO
55
56
57 void producto_vectorial(float array1[], float array2[], float result[]){
58   // componentes del vector resultante
59   result[0] = (array1[1]*array2[2] - array1[2]*array2[1]);
60   result[1] = -(array1[0]*array2[2] - array1[2]*array2[0]);
61   result[2] = (array1[0]*array2[1] - array1[1]*array2[0]);
```

```
62 } // END PRODUCTO_VECTORIAL
63
64
65 void printArrays(float array[], int length){
66   // impresion
67   printf("(");
68   for (int i = 0; i < length; i++){
69     if (i != length - 1){
70       printf("%f, ",array[i]);
71     } else {
72       printf("%f)\n", array[i]);
73     } // END IF
74   } // END FOR
75 } // END printArrays
76
77
78 void suma_vectorial(float array1[], float array2[], float result[]){
79   for (int i = 0; i < 3; i++){
80     result[i] = array1[i]+array2[i];
81   } // END FOR
82 } // END SUMA_VECTORIAL
83
84
85 float magnitud(float array[]){
86   //double sqrt(double x);
87   float norm = 0;
88
89   for (int i = 0; i < 3; i++){
90     norm += pow(array[i],2);
91   } // END FOR
92   return sqrt(norm);
93 } // END SUMA_VECTORIAL
94
95
96 // END PROGRAM
```

4. Problema 4

4.1. Enunciado

Crear un programa que solicite al usuario dos matrices de 3X3 almacenarlas como (matA, matB) y una constante, el programa automáticamente debe de mostrar las los siguientes resultados:

a) Matriz A por constante
b) Suma de las matrices
c) Resta de las matrices
d) Multiplicación de las matrices
e) Determinante de la matriz A
f) Transpuesta de la matriz B
g) Inversa de la matriz A
h) Reducción de Gauss de la matriz A
i) Reducción de Gauss Jordan de la matriz B

4.2. Metodología

Sean $A = (a_{ij})$ y $B = (b_{ij})$, con ambas matrices 3×3 y $c \in \mathbb{R}$.

a) Matriz A por constante. Cada elemento de la matriz multiplicado por la constante c.

$$cA = (ca_{ij}).$$

b) Suma de las matrices. Suma elemento a elemento, es decir

$$A + B = (a_{ij} + b_{ij}).$$

c) Resta de las matrices. Resta elemento a elemento, i.e.

$$A - B = (a_{ij} - b_{ij}).$$

d) Multiplicación de las matrices. Cada elemento de la matriz resultante $\Gamma = (\gamma_{ij})$ es de la forma

$$\gamma_{ij} = \sum_{k=1}^{n=3} a_{ik} b_{kj}.$$

e) Determinante de la matriz A. Tomando el método de cofactores, se define el menor como el determinante de una matriz M_{ij} que resulta de quitar la $i-$ésima fila y la $j-$ésima columna de la matriz original. Dado esto, se define el cofactor asociacdo a ij, como $\mathcal{A}_{ij} = (-1)^{i+j} M_{ij}$. Y el determinante de la matriz se calcula multiplicando cada entrada de cualquier fila (o columna) por su respectivo cofactor, es decir (tomando la fila 1 como ejemplo)

$$\det\{A\} = \sum_{j=1}^{3} (-1)^{1+j} a_{1j} M_{1j}.$$

Esta fue la implementación que se quizo hacer, dado que no se pudo por problemas en el código, se utilizó la regla de Sarrus

$$\det\{A\} = (a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32})$$
$$-(a_{31}a_{22}a_{13} + a_{32}a_{23}a_{11} + a_{33}a_{12}a_{21}).$$

En dicha fórmula, los índices varían según la relación $i = (1 + x) \mod 3$ y $j = (2 + x) \mod 3$, con $x \in [0, 2]$ (esto obviamente esta dedicado a la implementación, por eso los índices son 1 menos de lo que normalmente son.)

$$\det\{A\} = \sum_{x=0}^{2} a_{0x}(a_{1i}a_{2j} - a_{1j}a_{2i}).$$

f) Transpuesta de la matriz B. Se intercambian filas por columnas.

$$B^t = (b_{ji}).$$

g) Inversa de la matriz A. Para calcular la inversa se utiliza la siguiente relación

$$A^{-1} = \frac{1}{\det\{A\}} \text{adj}(A).$$

En donde la matriz adj(A) es la matriz de cofactores transpuesta, en forma visual

$$\text{adj}(A) = \text{cof}(A)^t = \begin{pmatrix} \mathcal{A}_{11} & \mathcal{A}_{12} & \mathcal{A}_{13} \\ \mathcal{A}_{21} & \mathcal{A}_{22} & \mathcal{A}_{23} \\ \mathcal{A}_{31} & \mathcal{A}_{32} & \mathcal{A}_{33} \end{pmatrix}^t.$$

h) Reducción de Gauss de la matriz A. Es un método en el cual se busca reducir la matriz a una matriz escalonada, esta reducción se hace mediante operaciones entre filas y columnas, con el fin de llegar a una matriz triangular.

$$A_{\text{gauss}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix}$$

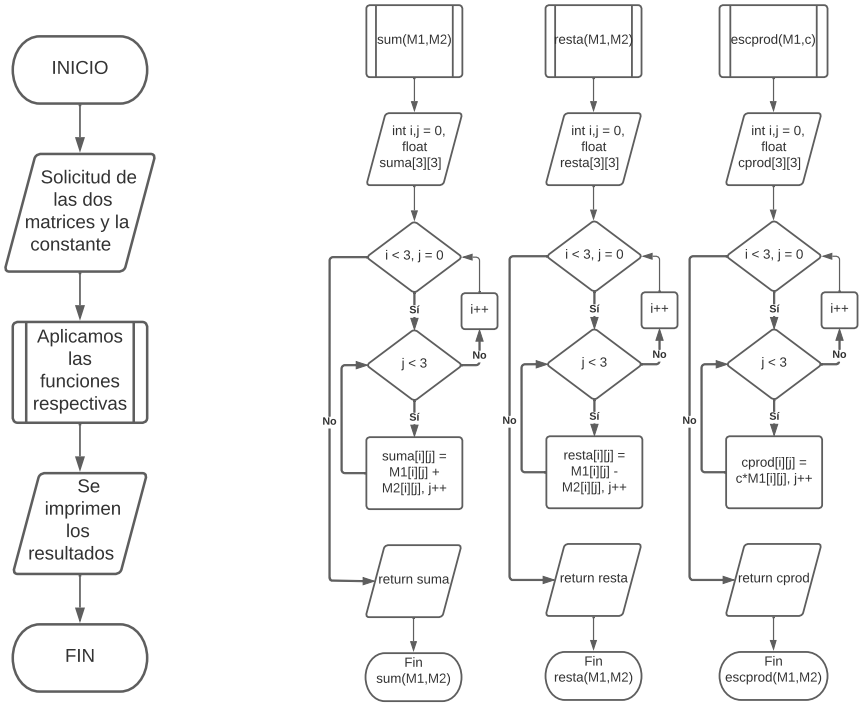
i) Reducción de Gauss Jordan de la matriz B. Para este método es exactamente la misma idea que el anterior, con la savedad de que la matriz resultate es la identidad.

4.3. Variables de entrada y salida

→: Matrices 3×3 A y B, y constante $c \in \mathbb{R}$.

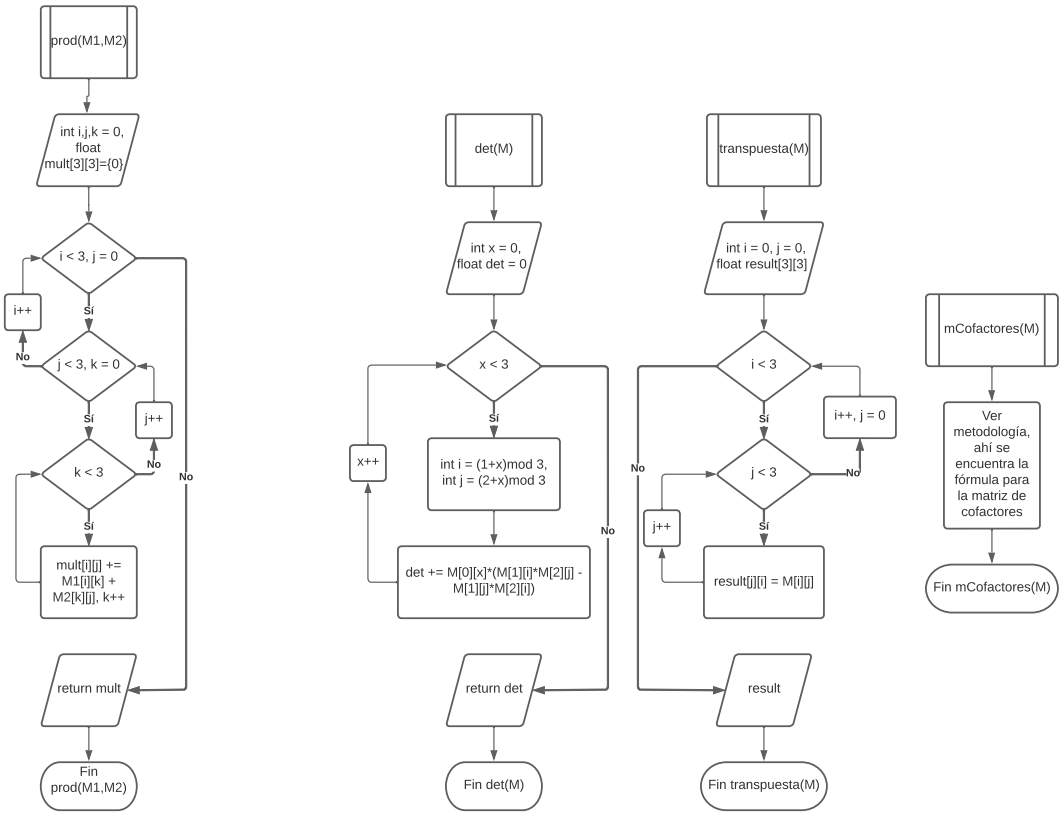
←: Arrays bidimensionales y constantes, cada una en relación a cada inciso.

4.4. Pseudocógido o Diagrama de Flujo



(a) "Main" (b) Suma, resta y producto por escalar

Figura 2: (a) Diagrama de flujo de la función main. (b) Implementación de la suma y resta de matrices así como el producto por escalar.



(a) Producto Matricial.

(b) Determinante, inversa y matriz de cofactores.

Figura 3: (a) Producto de matrices según lo explicado en la metodología. (b) Determinante mediante la regla de Sarrus, cálculo de la matriz transpuesta y cálculo de la matriz de cofactores de manera directa (usando una fórmula).

4.5. Código

```
1  //      Librerias
2  #include <stdio.h>
3  #include <math.h>
4
5  // 0. prototipado de funciones (m filas y n columnas)
6  #define fila 3
7  #define col 3
8  void prod_escalar(float array[][col], int m, int n, float result[][col],
       float c);
9  void suma(float array1[][col], float array2[][col], int m, int n, float
       result[][col]);
10 void resta(float array1[][col], float array2[][col], int m, int n, float
       result[][col]);
11 void prod_matrices(float array1[][col], float array2[][col], int m, int
       n, float result[][col]);
12
```

```
13 float determinante(float array[][col]);
14 //float cofactor(float array[][col], int n, int file, int column);
15 void transpuesta(float array[][col], int m, int n, float result[][col]);
16 void mCofactores(float array[][col], float result[][col]);
17 void redGauss(float array[][col], float result[][col]);
18 void redGaussJordan(float array[][col], float result[][col]);
19
20 // 1. main
21 int main(){
22   // 2. definiendo las variables a utilizar
23   float matA[fila][col], matB[fila][col], result[fila][col], aux[fila][
       col];
24   float c;
25
26   // 3. ingreso de las matrices y la constante
27   puts("Ingrese la primera matriz por filas en el siguiente formato: x,y
       ,z");
28   for (int i = 0; i < fila; i++){ scanf("%f,%f,%f", &matA[i][0], &matA[i
       ][1], &matA[i][2]); }
29   puts("Ingrese la primera matriz por filas en el siguiente formato: x,y
       ,z");
30   for (int j = 0; j < fila; j++){ scanf("%f,%f,%f", &matB[j][0], &matB[j
       ][1], &matB[j][2]); }
31   puts("Ingrese el valor de la constante.");
32   scanf("%f",&c);
33   puts("");
34   // 4. calculo de los respectvos incisos
35   // a)
36   prod_escalar(matA, fila, col, result, c);
37   puts("a) Matriz por una constante.");
38   for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
       result[i][1], result[i][2]); }
39   puts("");
40
41   // b)
42   suma(matA, matB, fila, col, result);
43   puts("b) matA + matB.");
44   for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
       result[i][1], result[i][2]); }
45   puts("");
46
47   // c)
48   resta(matA, matB, fila, col, result);
49   puts("c) matA - matB.");
50   for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
       result[i][1], result[i][2]); }
51   puts("");
52
```

```
53    // d)
54    prod_matrices(matA, matB, fila, col, result);
55    puts("d) matA * matB.");
56    for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
         result[i][1], result[i][2]); }
57    puts("");
58
59    // e)
60    puts("e) det(matA).");
61    printf("%f\n", determinante(matA));
62    puts("");
63
64    // f)
65    transpuesta(matB, fila, col, result);
66    puts("f) transpuesta de matB.");
67    for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
         result[i][1], result[i][2]); }
68    puts("");
69
70    // g)
71    mCofactores(matA, result);
72    transpuesta(result, fila, col, aux);
73    prod_escalar(aux, fila, col, result, 1/determinante(matA));
74    puts("g) inversa de matA.");
75    for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
         result[i][1], result[i][2]); }
76    puts("");
77
78    // h)
79    redGauss(matA, result);
80    puts("h) reduccion gauss de matA.");
81    for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
         result[i][1], result[i][2]); }
82    puts("");
83
84    // i)
85    redGauss(matB, result);
86    puts("i) reduccion gauss-jordan de matB.");
87    for (int i = 0; i < fila; i++){ printf("%f,%f,%f\n", result[i][0],
         result[i][1], result[i][2]); }
88    puts("");
89
90    return 0;
91 } // END MAIN
92
93
94 /*
95                                FUNCIONES
```

```
96 */
97
98
99 void prod_escalar(float array[][col], int m, int n, float result[][col],
       float c){
100   for (int i = 0; i < m; i++){
101     for (int j = 0; j < n; j++){
102       result[i][j] = c*array[i][j];
103     } // END FOR
104   } // END FOR
105 } // END prod_escalar
106
107
108 void suma(float array1[][col], float array2[][col], int m, int n, float
      result[][col]){
109   for (int i = 0; i < m; i++){
110     for (int j = 0; j < n; j++){
111       result[i][j] = array1[i][j] + array2[i][j];
112     } // END FOR
113   } // END FOR
114 } // END suma
115
116
117 void resta(float array1[][col], float array2[][col], int m, int n, float
       result[][col]){
118   for (int i = 0; i < m; i++){
119     for (int j = 0; j < n; j++){
120       result[i][j] = array1[i][j] - array2[i][j];
121     } // END FOR
122   } // END FOR
123 } // END suma
124
125
126 void prod_matrices(float array1[][col], float array2[][col], int m, int
      n, float result[][col]){
127   for (int i = 0; i < m; i++){
128     for (int j = 0; j < n; j++){
129       result[i][j] = 0;
130     } // END FOR
131   } // END FOR
132
133   for (int i = 0; i < m; i++){
134     for (int j = 0; j < n; j++){
135       for (int k = 0; k < n; k++){
136         result[i][j] += array1[i][k] * array2[k][j];
137       } // END FOR
138     } // END FOR
139   } // END FOR
```

```
140 } // END prod_matrices
141
142 // REGLA DE SARRUS
143 float determinante(float array[][col]){
144   float det = 0;
145   int i = 0, j = 0;
146
147   for (int x = 0; x < 3; x++){
148     i = (1 + x) % 3;
149     j = (2 + x) % 3;
150
151     det += array[0][x]*(array[1][i]*array[2][j] - array[1][j]*array[2][i
          ]);
152   } // END FOR
153   return det;
154 } // END determinante
155
156
157 void transpuesta(float array[][col], int m, int n, float result[][col]){
158   for (int i = 0; i < m; i++){
159     for (int j = 0; j < n; j++){
160       result[j][i] = array[i][j];
161     } // END FOR
162   } // END FOR
163 } // END transpuesta
164
165
166 void mCofactores(float array[][col], float result[][col]){
167   // manualmente
168   result[0][0] = array[1][1]*array[2][2] - array[1][2]*array[2][1];
169   result[0][1] = -(array[1][0]*array[2][2] - array[1][2]*array[2][0]);
170   result[0][2] = array[1][0]*array[2][1] - array[1][1]*array[2][0];
171   result[1][0] = -(array[0][1]*array[2][2] - array[0][2]*array[2][1]);
172   result[1][1] = array[0][0]*array[2][2] - array[0][2]*array[2][0];
173   result[1][2] = -(array[0][0]*array[2][1] - array[0][1]*array[2][0]);
174   result[2][0] = array[0][1]*array[1][2] - array[0][2]*array[1][1];
175   result[2][1] = -(array[0][0]*array[1][2] - array[0][2]*array[1][0]);
176   result[2][2] = array[0][0]*array[1][1] - array[0][1]*array[1][0];
177 } // END inversa
178
179
180 void redGauss(float array[][col], float result[][col]){
181   result = array;
182   for(int i=0;i<=2;i++){
183         for(int j=3;j>=0;j--){
184             result[i][j] = result[i][j]/result[i][i];
185         } // END FOR
186         for(int k=i+1;k<=2;k++){
```

```
187             for(int j=3;j>=0;j--){
188                 result[k][j] = result[k][j] - result[k][i]*result[i][j];
189             } // END FOR
190         } // END FOR
191     } // END FOR
192 } // END redGauss
193
194
195
196
197 void redGaussJordan(float array[][col], float result[][col]){
198   result = array;
199   for(int i=0;i<=2;i++){
200         for(int j=3;j>=0;j--){
201             result[i][j] = result[i][j]/result[i][i];
202         }
203         for(int k=i+1;k<=2;k++){
204             for(int j=3;j>=0;j--){
205                 result[k][j] = result[k][j] - result[k][i]*result[i][j];
206             }
207         }
208         for (int k=0; k<=i-1;k++){
209             for(int j=3;j>=0;j--){
210                 result[k][j] = result[k][j] - result[k][i]*result[i][j];
211             }
212         }
213     }
214 } // END redGaussJordan
215
216
217 // END PROGRAM
```

5. Problema 5

5.1. Enunciado

Crear un programa que encuentre el factorial de un numero entero ingresado, debe de utilizar una función recursiva.

5.2. Metodología

La función factorial es una función recursiva definida a tramos

$$n! = \begin{cases} n*(n-1)! & n \geq 2 \\ 1 & n = 0,1 \\ 0 & \text{en otro caso.} \end{cases}$$

Existen muchas otras definiciones, como la función gamma, pero para interés del problema, utilizamos la definición recursiva.

5.3. Variables de entrada y salida

→: Número entero n.

←: Número entero. (Dado el rápido crecimiento de la función factorial, es probable que para valores, aparentemente, normales de n, ni siquiera los `unsigned long long int` logren poder almacenar el número.)

5.4. Pseudocógido o Diagrama de Flujo

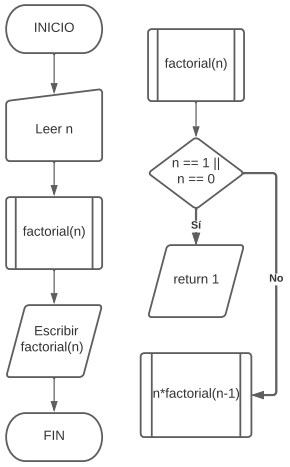


Figura 4: Diagrama de flujo de la implementación recursiva de la función factorial.

5.5. Código

```c
//      Librerias
#include <stdio.h>

// 0. prototipado de funciones y definicion de variables
unsigned long long int factorial(int n);

// 1. funcion main
int main(){
  // 2. definicion de variables
  int n;

  // 3. ingreso y validacion del dato ingresado por el usuario
  puts("Ingrese un numero entero positivo.");
  scanf("%d", &n);

  //4. utilizacion de la funcion recursiva
  printf("El factorial del numero ingresado es: %lld\n", factorial(n));

  return 0;
```

```
} // END MAIN


/*
                              FUNCIONES
*/


unsigned long long int factorial(int n){
  if (n == 1 || n == 0){
    return 1;
  } else {
    return n*factorial(n-1);
  } // END IF
} // FUNCION FACTORIAL


// END PROGRAM
```

6. Problema 6

6.1. Enunciado

Crear un programa que realice la sumatoria desde 1 hasta un numero n que ingrese el usuario de las siguientes funciones.

a)
$$\sum_{k=1}^{n} k^2(k-3)$$

b)
$$\sum_{k=2}^{n} \frac{3}{k-1}$$

c)
$$\sum_{k=1}^{n} \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^k$$

d)
$$\sum_{k=2}^{n} 0.1(3 \cdot 2^{k-2} + 4).$$

6.2. Metodología

No hay mucho que añadir, la sumatoria es un bucle con almacenamiento en una variable que agrega el valor del paso anterior más el de la sucesión valuada en el k correspondiente.

6.3. Variables de entrada y salida

→: Número entero n.

←: Número entero o flotante, según sea el caso de la serie.

6.4. Pseudocógido o Diagrama de Flujo

Paso 1: Prototipado de funciones para cada inciso (por simplicidad fueron nombradas con el inciso correspondiente), cuyas entradas son un número entero.

Paso 2: Ingreso del numero entero n (limite superior de las series).

Paso 3: Para cada función se realizó exactamente el mismo procedimiento, solo que con diferente término de sumatoria, de modo que el ciclo for general es

```
Definir la suma como un float, suma
Mientras un iterador sea menor igual que n, hacer
suma = suma + f_k
aumentar en 1 el iterador
Devolver "suma"
```

Paso 4: Realizar esto para cada función e imprimir los respectivos resultados.

6.5. Código

```c
//      Librerias
#include <stdio.h>
#include <math.h>

// 0. Prototipado de funciones
float a(int n);
float b(int n);
float c(int n);
float d(int n);


// 1. funcion main
int main(){
  // 2. ingreso del limite superior
  int n;
  puts("Ingrese un numero entero positivo.");
  scanf("%d", &n);

  printf("Inciso a) %.2f \n", a(n));
  printf("Inciso b) %.2f \n", b(n));
  printf("Inciso c) %.2f \n", c(n));
  printf("Inciso d) %.2f \n", d(n));
}


/*
                              FUNCIONES
*/

float a(int n){
  float sum1 = 0;
  for (int i = 1; i <= n; i++){
    sum1 += pow(i,2)*(i - 3);
  }
  return sum1;
```

```
36 } //END a
37
38 float b(int n){
39   float sum2 = 0;
40   for (int i = 2; i <= n; i++){
41     sum2 += 3/(i - 1);
42   }
43   return sum2;
44 } // END b
45
46 float c(int n){
47   float sum3 = 0;
48   for (int i = 1; i <= n; i++){
49     sum3 += (1/sqrt(5))*pow(((1 + sqrt(5))/2),i) - (1/sqrt(5))*pow(((1 -
         sqrt(5))/2),i);
50   }
51   return sum3;
52 } // END c
53
54 float d(int n){
55   float sum4 = 0;
56   for (int i = 2; i <= n; i++){
57     sum4 += 0.1*(3*pow(2,i - 2) + 4);
58   }
59   return sum4;
60 } // END d
61
62
63 // END PROGRAM
```