Memoria de la entrega final de Matemáticas Aplicadas

Introducción

En esta memoria se explica el funcionamiento del programa adjunto para hacer diversos cálculos con matrices, además de su proceso de creación, corrección y mejora. Todo el código se ha desarrollado en Visual Studio 2022, con las herramientas aprendidas durante las asignaturas de Introducción a la Programación y Programación Orientada a Objetos, además de herramientas y funciones aprendidas de forma independiente.

Funcionamiento del programa

El programa carece de interfaz gráfica, por tanto todas sus funciones se harán desde la propia consola de comandos. El funcionamiento es muy sencillo.
Según iniciamos el programa, nos aparece un menú con diversas opciones asignadas cada una a un número diferente.
Deberemos teclear uno de esos números + intro para activar la opción elegida. Una vez hecho esto, el programa ejecutará las funciones necesarias y nos pedirá la información requerida para realizar la tarea pedida.

Desarrollo del programa

El programa en sí se puede dividir en cuatro secciones diferentes.

Funciones recurrentes

Estas funciones son las primeras en aparecer en el programa ya que son funciones que se utilizan varias veces a lo largo del programa. Son cinco:

bool CheckInt(string num)

Recibe un *string* y mediante un *int.TryParse* comprueba si es o no un número entero. Devuelve un valor booleano.

double[,] ConstruirMatrizCuadrada()

Esta función se utilizará en todas las funciones que requieran de una matriz cuadrada de un tamaño específico. Mediante un *Console.ReadLine()* registra en un *int* el tamaño de la matriz, pasándolo por un *int.Parse* para convertir el *string* del *ReadLine* en un *int*.
A continuación crea la matriz con el tamaño registrado previamente y con dos bucles *for* anidados y un *Console.ReadLine()* lee y crea la matriz.
Dicha matriz la almacenamos en un *double[,]* que es lo que devuelve la función.

double[,] ConstruirMatriz()

Esta función se utilizará más adelante en los casos en los que se necesite crear una matriz de un tamaño específico.
Se declaran dos variables de tipo *int* que se asignarán mediante un *Console.ReadLine()* (que pasarán también por la función *CheckInt()*). A continuación se crea una matriz del tamaño asignado a las dos variables de tipo *int* y, al igual que en la función anterior, mediante dos bucles *for* anidados y un *Console.ReadLine()* se lee y se crea la matriz deseada.
Dicha matriz se almacena en un *double[,]* que es lo que devuelve la función.

void ImprimirMatriz(double[,] matriz)

Esta función se llama a lo largo del programa cada vez que se necesita imprimir una matriz por pantalla. No devuelve ningún valor y recoge la matriz que se desea imprimir.
Primero se saca el número de filas y columnas de la matriz mediante *GetLength(0)* y *GetLength(1)* y se almacenan en dos variables de tipo *int*.
La función imprime la matriz mediante dos bucles *for* anidados que van recorriendo las filas y columnas de la matriz.

Funciones de inicio

Son las funciones que se ejecutan nada más iniciar el programa.

void Main(string[] args)

En este programa el Main lo único que hace es llamar a la siguiente función.

void Eleccion()

En un principio el contenido de esta función estaba incluido en el *Main* pero finalmente se hizo una función por separado del *Main* para así poder llamarla desde otras funciones. De esta forma podemos reiniciar el programa cada vez que acabemos de utilizar una de sus funciones. El funcionamiento es simple.
Primero, creamos una variable *int* en la que almacenaremos más adelante el número elegido por el usuario.Después, escribimos en pantalla todas las opciones del menú y con un *ReadLine* almacenamos en un *string* la elección del usuario.
Pasando el *string* por la función *CheckInt()* comprobamos si es un *int*. Si no lo es, mostramos un mensaje de error y relanzamos la función *Eleccion().* Si es un *int,* lo comparamos con las opciones elegidas y según cual haya elegido el usuario, ejecutaremos una de las **funciones principales**.

Funciones principales

Estas son las funciones que se ejecutan al seleccionar una de las opciones del menú principal.

void Determinante()

Se ejecuta con la primera opción del menú y se encarga de calcular el determinante de una matriz de orden 3.
Primero crea una matriz bidimensional de orden 3 y posteriormente, mediante un *Console.ReadLine* con un *int.Parse* en un bucle *for* se va escribiendo y almacenando los valores introducidos en la matriz creada.
A continuación crea un *int* en el que almacena la operación en la que calculamos el determinante mediante la regla de Sarrus.
Para terminar, imprimimos por pantalla el determinante y volvemos a ejecutar la función *Eleccion()* para volver al menú principal.

void Sumar()

Se ejecuta con la segunda opción el menú y se encarga de calcular la suma de dos matrices. Como las matrices tienen que ser del mismo tamaño pero no necesariamente cuadradas, se utiliza la función *ConstruirMatriz()* para construir las matrices que se van a sumar. Además, el programa crea una matriz bidimensional del mismo tamaño que las matrices construidas previamente para almacenar el resultado.
Posteriormente, mediante dos bucles *for* anidados, se recorren las dos matrices de forma paralela y se van sumando los elementos que comparten posición en las distintas matrices. La matriz resultante se guarda en la matriz creada para ello previamente, y se imprime mediante la función *ImprimirMatriz()*.

void Restar()

Se ejecuta con la tercera opción del menú y se encarga de calcular la resta de dos matrices. El funcionamiento es idéntico a la función *Sumar()* pero cambiando el signo de la operación.

void Producto()

Se ejecuta con la cuarta opción del menú y se encarga de calcular el producto de dos matrices. Este caso es particular, ya que es necesario que el tamaño de las matrices sea inverso, es decir, que si la matriz A tiene una forma de **m x n** la matriz B debería tener una forma de **n x m**. Aún así, la función no limita la introducción de valores para las filas y columnas, sino que permite introducir los valores de forma individual, y cuando se construyen las matrices se comprueba si estas son compatibles para hacer la operación, es decir, que se cumpla la equivalencia indicada anteriormente.

Con el condicional *if* comprobamos esta relación de filas y columnas y si es correcta, realizamos la operación. Creamos la matriz resultado y almacenamos en ella los resultados de las operaciones con tres bucles *for* anidados.

void DeterminanteCualquierOrden()

Se ejecuta con la quinta opción del menú y se encarga de calcular el determinante de una matriz cuadrada de cualquier orden.
Para empezar, construimos la matriz con *ConstruirMatrizCuadrada()* y la almacenamos en un *double[,].* A continuación utilizamos un conjunto de bucles *for* anidados para realizar el método de reducción de Gauss. De esa forma, la función sacará el determinante de la matriz dentro del segundo bucle *for*, multiplicando los elementos de la diagonal principal de la matriz resultante de reducir por Gauss.
Por último, la función escribe por pantalla el determinante y ejecuta la función *Eleccion()* para elegir una nueva opción del menú.

void MatrizInversa()

Se ejecuta con la sexta opción del menú y se encarga de calcular la matriz inversa de una matriz invertible. Si la matriz no es invertible (su determinante es igual a 0, devuelve un error y reinicia el menú principal).
Para empezar, construimos la matriz con *ConstruirMatrizCuadrada()* y calculamos su determinante con *DeterminanteGeneral()*. Si el determinante es igual a 0, el programa devuelve un error y reinicia la función para que el usuario introduzca una matriz invertible.
Si la matriz es válida, se utilizan las funciones auxiliares *MatrizAdjunta()* y *MatrizTraspuesta()* para calcular la matriz adjunta y su traspuesta respectivamente. Después, se utiliza un *double* fac, que es el resultado de dividir 1 entre el determinante.
Se utiliza la función auxiliar *MatrizPorEscalar()* para multiplicar la matriz adjunta traspuesta por el fac (1 partido por el determinante) y el resultado se almacena en la matriz, la cual con dos bucles *for* anidados se muestra por pantalla.

Funciones auxiliares

Son funciones que no aparecen en el menú de elección pero que son operaciones auxiliares para las funciones principales.

double[,] MatrizTraspuesta(double[,] matriz)

Se encarga de calcular la matriz traspuesta de una matriz que se pasa por su argumento. Para ello, recibe una matriz y crea dos nuevas, una en la que iremos almacenando el resultado y otra que utilizaremos como copia de la original para no alterar la matriz inicial. Después, en dos bucles *for* anidados se va alterando la posición de cada elemento cambiando su número de fila por su número de columna y viceversa.

double[,] Submatriz(double[,] matriz, int fila, int columna)

Se encarga de calcular una submatriz que es de un orden menos que la matriz que recibe por el argumento. Se utiliza dentro de la función *MatrizAdjunta()* para calcular los adjuntos en cada posición.

double[,] MatrizAdjunta(double[,] matriz)

Se encarga de calcular la matriz adjunta de una matriz pasada por su argumento. Para ello, utiliza la previamente explicada función *Submatriz()* para ir posición a posición calculando su adjunto. Para ello, por cada iteración dentro de los bucles *for* anidados, calcula el signo que le corresponde (comprobando la posición de la matriz en la que se encuentra, si es posición par o impar), calcula la submatriz de ese adjunto y su determinante para luego asignarle el signo adecuado.
Finalmente, devuelve la matriz adjunta al completo en forma de *double[,]*.

double[,] MatrizPorEscalar(double[,] matriz, double escalar)

Esta función recibe una matriz en forma de *double[,]* y un escalar en forma de *double* y se encarga de multiplicarlos, recorriendo la matriz mediante dos bucles *for* anidados y multiplicando cada elemento por el escalar.
Finalmente, devuelve la matriz resultante en forma de *double[,]*.

double DeterminanteGeneral(double[,] matriz)

Es la misma función que la función principal DeterminanteCualquierOrden() sólo que eliminando la parte de escritura por pantalla. El funcionamiento es idéntico.

Conclusión

El programa ha pasado por varias fases. Al principio no existían apenas funciones auxiliares, sino que solamente había una función por cada opción elegida por el usuario y en ella se desarrollaban todos los pasos, pero poco a poco fui asignando a funciones externas algunas tareas como el cálculo de determinantes, imprimir matrices, etc.
El programa no es 100% óptimo y tiene varias estructuras que se podían haber hecho de forma más efectiva, pero funciona correctamente y al leerse se entiende cómo está hecho y estructurado. Además, me ha ayudado a refrescar conocimientos de las asignaturas de programación y a entender mejor el funcionamiento de las matrices.