bAUTISTA DE sUTO nAGY

profesor: iSRaEL PAVELEK

PORTFOLIO

LE

Tabla de contenido

[C: 2](#_Toc120125305)

[Tienda C: 2](#_Toc120125306)

[C++: 2](#_Toc120125307)

[Tienda C++: 2](#_Toc120125308)

[Librería Algebra lineal: 4](#_Toc120125309)

[Python: 5](#_Toc120125310)

[Graficador de diagramas de Bode: 5](#_Toc120125311)

[Machine Learning: 8](#_Toc120125312)

[Repositorio](https://github.com/dbbuty/Portfolio-LE)

# C:

## Tienda C:

Este código es un sistema para una tienda, que permite cargar productos y venderlos llevando cuenta del stock. El código utiliza estructuras y punteros para formar una lista doblemente enlazada, además cuenta con un algoritmo de ordenamiento:

typedef struct productos{

    char nombre[256];

    int stock;

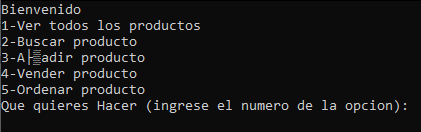
    float precio;

    struct productos \*next;

    struct productos \*prev;

}producto;

La interfaz se ejecuta en la consola y cuenta con las siguientes funcionalidades:



# C++:

## Tienda C++:

Este código es una mejora del código anterior hecho en C. Implementa la librería vector y string, clases y manejo de archivos. EL programa cuenta con 2 clases:

class producto{

    public:

        string nombre;

        int stock;

        float precio;

        producto(string n, int s, float p);

        producto();

        void show();

        void operator = (producto data);

};

class lista{

    private:

    vector<producto> array;

    public:

    producto\* operator [] (int index);

    producto\* operator [] (string name);

    void show();

    void push(producto data);

    void upload(string adress);

    void download(string adress);

};

Los métodos upload y download permiten cargar y descargar la información de los productos en un archivo local de manera que la información no se pierda al cerrar el programa.

La interfaz se ejecuta en la consola y cuenta con las siguientes funcionalidades:

Text

Description automatically generated

Text

Description automatically generated

## 

## Librería Algebra lineal:

Este código implementa un nuevo tipo de clases “matrix”. La clase permite crear matrices matemáticas utilizadas en algebra lineal. Permite, por ejemplo, realizar operaciones entre matrices, cálculo de determinantes de matrices n\*n y cálculo de matriz inversa

class matrix{

    public:

        float\* matriz;

        int row;

        int col;

        matrix(int r, int c); //constuctor

        float & operator () (int r, int c);

        matrix t(void); //matriz traspuesta

        matrix adj(void); //matriz de adjuntos

        float adj(int r, int c); //adjunto

        float det(void); //determinante

        void operator = (matrix data);

        void operator = (float\* data);

        matrix operator + (matrix data);

        matrix operator \* (matrix data);

        matrix operator \* (float data);

        matrix operator ! (void); //matirz inversa

};

El objetivo era combinar lo que aprendí por mi cuenta de matrices con los nuevos conceptos de programación orientada a objetos que aprendimos durante el año en LE.

Un ejemplo de lo que podemos hacer con esta nueva clase es resolver un sistema de ecuaciones lineales como el siguiente:

Para esto debemos multiplicar el inverso de la matriz de coeficientes por la matriz de resultados:

Para hacer esto en el código debemos primero declarar un array con los coeficientes en orden por fila, luego declaramos un objeto de tipo matriz e indicamos que posee 2 columnas y 2 filas en el constructor. Por último, igualamos la matriz al array:

float cofs[4] = {2,1,4,-3};

matrix Cofs(2,2);

Cofs = cofs;

Para declarar la matriz de resultados hacemos lo mismo, pero con una matriz de 1 columna y 2 filas:

float res[2] = {12, 4};

matrix Res(2,1);

Res = res;

Para verificar podemos mostrar las matrices en la consola. Por último, multiplicamos el inverso de la matriz de coeficientes por la matriz de resultados y la mostramos en la consola:

cout << Cofs << endl << Res << endl;

cout << !Cofs \* Res;

Text

Description automatically generated

Al ejecutar el código vemos que las matrices se declaran correctamente y obtenemos un resultado al multiplicar las matrices. Además es solución del sistema de ecuaciones que planteamos, por lo que el programa cumple el objetivo.

# Python:

## Graficador de diagramas de Bode:

Este código utiliza las librerías Matplotlib, Numpy para graficar diagramas aproximados de bode. Además, cuenta con una interfaz grafica echa con la librería Tkinter:

Chart

Description automatically generated

La ventana cuenta con 3 entradas: una para el componente lineal de la transferencia, los polos y los ceros. Además, se puede elegir entre graficar la ganancia o la fase y cuenta con la opción de mostrar las componentes por separado.

En caso de que la transferencia cuente con mas de un cero o un polo, estos deben ser ingresados separados por comas: Chart, line chart

Description automatically generated

Al activar la opción de ver los componentes, podemos ver cada uno de los polos y ceros graficados por separado:

Chart, line chart

Description automatically generated

También se puede graficar la fase, y las componentes de esta:

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Para graficar los diagramas aproximados de bode implemente las siguientes funciones:

def log(x):

    if x: return(np.maximum(0,t-np.log10(np.abs(x))))

    else: return(t)

Esta función es la de un polo o cero individual. Cuando x no es 0, al graficar la función, esta es igual a 0 hasta que (t es el conjunto de valores sobre el que se grafica la función), en ese momento se vuelve una función lineal de pendiente 1. Cuando x es 0 la función es una función lineal de pendiente 1.

Esta función se utiliza para calcular cada polo y cero individual, que luego son agrupados:

def Bode(zeros, poles, lin):

    zeros = [20\*log(zero) for zero in zeros]

    poles = [-20\*log(pole) for pole in poles]

    res = poles+zeros

    res.append(0\*t+20\*np.log10(np.abs(lin)))

    return(res)

a cada cero se la da una pendiente de 20, y a cada polo se le da una pendiente de -20. Por último, se agrega el componente lineal en decibeles.

Para finalizar se suman todos los componentes y se grafican:

s = sum(Bode(Ceros, Polos, Lin))

ax.plot(t, s)

ax.set(xlabel='ω (rad/seg)', ylabel='Av (db)', title='Grafico de Bode')

Para el grafico de la fase utilice un concepto similar:

def tan(x):

    if x: return(np.maximum(0,t+1-np.log10(np.abs(x))) - np.maximum(0,t-1-np.log10(np.abs(x))))

    else: return(0\*t+2)

En este caso También se tiene una función que es 0 hasta cierto punto y luego se vuelve una función lineal. En este caso la función se vuelve lineal una unidad (una década) antes que la función del grafico de ganancia. Luego se le resta una función igual pero que se vuelve lineal una década después que la función del grafico de ganancia, de manera que en ese punto se cancelan.

Para el grafico de la suma de los componentes se usa código casi idéntico al del grafico de ganancia.

# Machine Learning:

Durante las clases que tuvimos de machine learning pude darme cuenta de lo importante que son los datos para el machine learning. Muchas veces conseguir los datos es la parte mas complicada e importante del machine learning. Otra cosa de la que no sabia era la importancia de depurar y entender los datasets para dejar únicamente los datos que nos interesan y arreglar incongruencias. Durante la materia pude utilizar herramientas para visualización como matplotlib, aunque no para visualizar datos de datasets.

Por mi cuenta también mira la siguiente serie de videos que explica las redes neuronales desde un abordaje más matemático: <https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi>

Me gustaría poder aprender más sobre el tema y realizar más código, ya que me parece un tema con mucha utilidad y que se está desarrollando mucho en la actualidad.