bAUTISTA DE sUTO nAGY

profesor: iSRaEL PAVELEK

PORTFOLIO

LE

Tabla de contenido

[C: 2](#_Toc119968163)

[Tienda C: 2](#_Toc119968164)

[C++: 2](#_Toc119968165)

[Tienda C++: 2](#_Toc119968166)

[Librería Algebra lineal: 2](#_Toc119968167)

[Python: 2](#_Toc119968168)

[Graficador de diagramas de Bode: 2](#_Toc119968169)

[Machine Learning: 2](#_Toc119968170)

[Repositorio](https://github.com/dbbuty/Buty)

# C:

## Tienda C:

# C++:

## Tienda C++:

## Librería Algebra lineal:

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

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 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, con raíz en . 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(lin))

    return(res)

a cada cero se la da una pendiente de 20, y a cada cero 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')

# Machine Learning: