APENDICE A

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Código para la obtención de la base de datos para cada código de red

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

```
#librerías
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt


#Se definen los parámetros de entrada al algoritmo de control

Irated = 10
delta = 0.01

#Variables de entrada: Pg (Potencia de referencia), Vp, Vn y phase (Desfase entre secuencias)


vgp1 = np.arange(start=0.046, stop=0.19, step=0.0072) # 20 pasos
vgp2 = np.arange(start=0.19, stop=0.4, step=0.0525)   # 4 pasos
vgp3 = np.arange(start=0.4, stop=0.928, step=0.0264)  # 20 pasos
vgp4 = np.arange(start=0.928, stop=1, step=0.018)     # 4 pasos

vgn1 = np.arange(start=0, stop=0.6, step=0.03)        # 20 pasos
vgn2 = np.arange(start=0.6, stop=1, step=0.04)        # 10 pasos

phase1 = np.arange(start=-178.2, stop=-20.3, step=15.79)   # 10 pasos
phase2 = np.arange(start=-20.3, stop=20.3, step=2.03)      # 20 pasos
phase3 = np.arange(start=20.3, stop=180, step=15.97)       # 10 pasos


vgp=np.zeros(len(vgp1)+len(vgp2)+len(vgp3)+len(vgp4))
vgn=np.zeros(len(vgn1)+len(vgn2))
phase=np.zeros(len(phase1)+len(phase2)+len(phase3))

Pref = np.arange(start=0, stop=1, step=0.1)*1000 #kW       # 10 pasos

vgp = np.concatenate((vgp1,vgp2,vgp3,vgp4))
vgn = np.concatenate((vgn1,vgn2))
phase = np.concatenate((phase1,phase2,phase3))
```

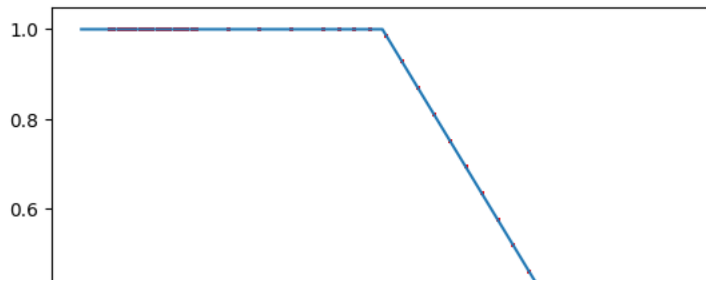
SE DEFINEN LA COORDENADAS DEL CÓDIGO QUE SE DESEA OBTENER

```
x1_vec = [0.5]
y1_vec = [1]
x2_vec = [0.95]
y2_vec = [0]


#Se obtiene la matriz que implica todas las posibles combinaciones

introspace = len(x1_vec)*len(y1_vec)*len(x2_vec)*len(y2_vec)*len(phase)*len(Pref)*len(vgn)*len(vgp)

print('Tamaño del espacio de entrada', introspace,'combinaciones.')

data_intro = np.zeros((introspace,8))

#print(data_intro)

n = 0 # Contador

for aa in range(len(x1_vec)):
    for bb in range(len(y1_vec)):
        for cc in range(len(x2_vec)):
            for dd in range(len(y2_vec)):
                for i in range(len(Pref)):
                    for ii in range(len(vgp)):
```

```
                        for iii in range(len(vgn)):
                            for iv in range(len(phase)):
                                data_intro[n] = [x1_vec[aa],y1_vec[bb],x2_vec[cc],y2_vec[dd],Pref[i],vgp[ii],vgn[iii],phase[iv]]
                                n = n+1
```

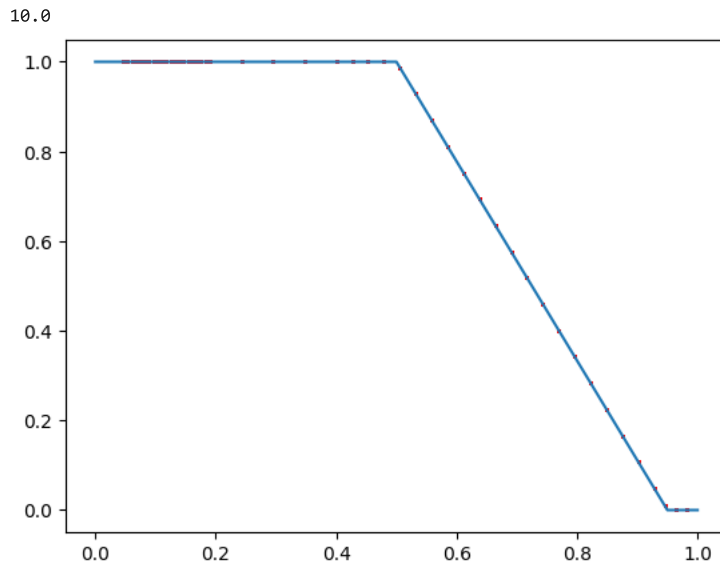
```
Tamaño del espacio de entrada 615000 combinaciones.
```

```
# Pre Algoritmo

V_pos = data_intro[:,5]*110*np.sqrt(2)
V_pos_pu = V_pos/(110*np.sqrt(2))
V_neg = data_intro[:,6]*110*np.sqrt(2)
V_neg_pu= V_neg/(110*np.sqrt(2))
phi = data_intro[:,7]*np.pi/180
#P = (3/2)*V_pos*Irated
m = (data_intro[:,3]-data_intro[:,1])/(data_intro[:,2]-data_intro[:,0])
b = (data_intro[:,2]*data_intro[:,1]-data_intro[:,0]*data_intro[:,3])/(data_intro[:,2]-data_intro[:,0]) # Crea grid code

phi1 = np.cos(phi)
phi2 = np.cos(phi - 2*np.pi/3)
phi3 = np.cos(phi + 2*np.pi/3)

#print(phi)
#print(phi1)
#print(phi2)
#print(phi3)

cos_min_phi = np.zeros(introspace)

for a in range(introspace):
    cos_min_phi[a] = min([phi1[a], phi2[a], phi3[a]])


#Se calcula la corriente Iq+/Irated que se debe inyectar de acuerdo con los códigos de red

y = np.zeros(introspace)

for c in range(introspace):

    if ((V_pos_pu[c] >= 0) and (V_pos_pu[c] <= data_intro[c,0])):

        y[c] = data_intro[c,1]

    elif ((V_pos_pu[c] > data_intro[c,0]) and (V_pos_pu[c] < data_intro[c,2])):

        y[c] = m[c]*V_pos_pu[c]+b[c]

  #  elif (V_pos_pu[c] == data_intro[c,2]):###Activar cuando x2 = 0.2

  #      y[c] = data_intro[c,3]####Activar cuando x2 = 0.2

    else:
 #       y[c] = 0 ##Activar cuando x2 = 0.2
        y[c] = data_intro[c,3] ##Desactivar cuando x2 = 0.2

x = [0,x1_vec[0],x2_vec[0],1] ###Desactivar cuando x2 = 0.2
y1 = [y1_vec[0],y1_vec[0],y2_vec[0],y2_vec[0]] ####Desactivar cuando x2 = 0.2

#x = [0,x1_vec[0],x2_vec[0],x2_vec[0],1] ####Activar cuando x2 = 0.2
#y1 = [y1_vec[0],y1_vec[0],y2_vec[0],0,0] ####Activar cuando x2 = 0.2
plt.plot(x, y1)
plt.scatter(V_pos_pu, y, color='red', s=1)
plt.show()
```




```
 #Obtención de Iq+GC multiplicando y por Irated

Iq_pos_GC = y*10

print(Iq_pos_GC[0])
plt.plot(x, y1)
plt.scatter(V_pos_pu, Iq_pos_GC/10, color='red', s=1)
plt.show()
```

```
10.0
```

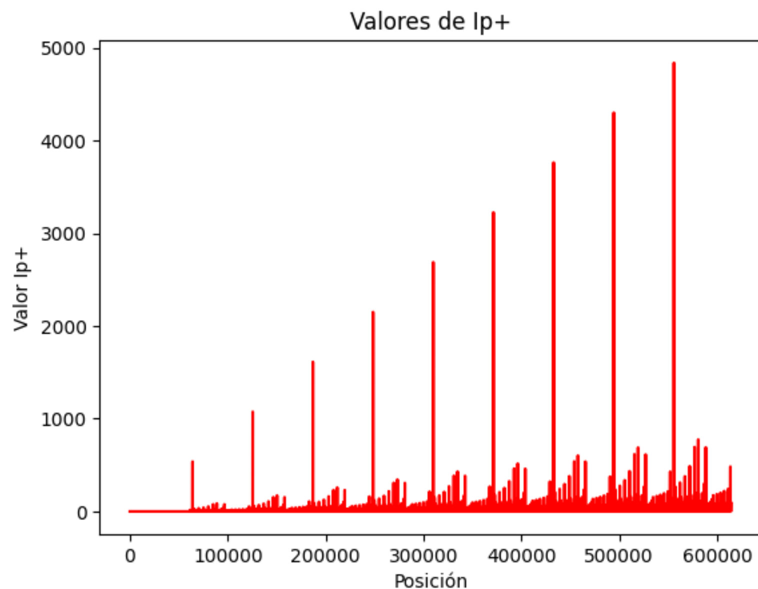


$$I_p^+ = \frac{2}{3}\frac{V^+}{(V^+)^2 - (V^-)^2}P^*$$

```
#Obtención de Ip+ inicial del código
Ip_pos_i = np.zeros(introspace)

for d in range(introspace):

    if V_pos[d]**2-V_neg[d]**2 == 0:

        Ip_pos_i[d] = Irated  #Ip_pos_max? ####PUEDE ESTAR AQUI
    else:

        Ip_pos_i[d] = Ip_pos_i[d] = ((2/3)*V_pos[d]*data_intro[d,4])/((V_pos[d]**2)-1*(V_neg[d]**2)) # Eq. (14) paper

for dd in range(introspace):

    if Ip_pos_i[dd] < 0:

        Ip_pos_i[dd] = 0

plt.xlabel('Posición')
plt.ylabel('Valor Ip+')
plt.title('Valores de Ip+')
```

```
plt.plot(range(0,len(Ip_pos_i)),np.array(Ip_pos_i),color = 'red')
```

```
[<matplotlib.lines.Line2D at 0x7fa5ca754820>]
```

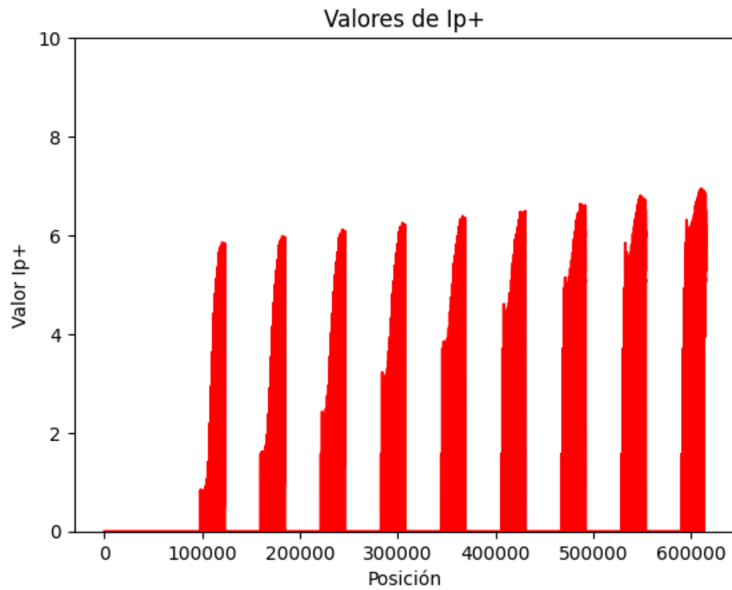


ALGORITMO DE CONTROL

```
Ipeak = np.zeros(introspace)
Iq_pos_min = np.zeros(introspace)
Ip_pos_max = np.zeros(introspace)
Ip_pos = np.zeros(introspace)
Iq_pos = np.zeros(introspace)
Ip_neg = np.zeros(introspace)
Iq_neg = np.zeros(introspace)


for e in range(introspace):

    if V_pos_pu[e] < x2_vec: #Se despliegan los casos 3 4 5 6

        Ipeak[e] = Irated

        Iq_pos_min[e] = Iq_pos_GC[e]

        Iq_pos[e] = Iq_pos_min[e]

        Ip_pos_max[e] = (((V_pos[e]**2)*(Ipeak[e]**2))/((V_pos[e]**2)-(2*V_pos[e]*V_neg[e]*cos_min_phi[e])+(V_neg[e]**2)))-Iq_pos_min[e]**2 #

        if Ip_pos_max[e] < 0:

            Ip_pos_max[e] = 0

        else:
         Ip_pos_max[e] = np.sqrt(Ip_pos_max[e]) #EC 23

        if Ip_pos_max[e] == 0: #Se despliegan los casos 5 Y 6

            Ip_pos[e] = 0

            Ipeak[e] = np.sqrt(((V_pos[e]**2-(2*V_pos[e]*V_neg[e]*cos_min_phi[e])+V_neg[e]**2)*(Ip_pos[e]**2+Iq_pos_min[e]**2))/V_pos[e]**2)

            if Ipeak[e] > Irated:#Caso 6

                Iq_pos_min[e] = Irated

                Iq_pos[e] = Iq_pos_min[e]

                Ip_neg[e] = 0 ; Iq_neg[e] = 0

                Ipeak[e] = Iq_pos[e]
```

```
            else: #Caso 5

                Iq_pos[e] = Iq_pos_min[e]

                Ip_neg[e] = (V_neg[e]/V_pos[e])*Ip_pos[e] #EC. 15

                Iq_neg[e] = (V_neg[e]/V_pos[e])*Iq_pos[e] #EC. 16

        else: #Se despliegan casos 3 y 4

            if Ip_pos_i[e] < Ip_pos_max[e]: #Caso 3

                Ip_pos[e] = Ip_pos_i[e]

                Iq_pos[e] = (((V_pos[e]**2)*Ipeak[e]**2)/((V_pos[e]**2)-(2*V_pos[e]*V_neg[e]*cos_min_phi[e])+(V_neg[e]**2)))-Ip_pos[e]**2 # E

                if Iq_pos[e] < 0:

                    Iq_pos[e] = 0

                else:

                    Iq_pos[e] = np.sqrt(Iq_pos[e]) # EC. 25

            else: #Caso 4

                Ip_pos[e] = Ip_pos_max[e]

                Iq_pos[e] = Iq_pos_min[e]

            Ip_neg[e] = (V_neg[e]/V_pos[e])*Ip_pos[e]

            Iq_neg[e] = (V_neg[e]/V_pos[e])*Iq_pos[e]

    else: #Se despliegan casos 1 y 2

        Iq_pos_min[e] = Iq_pos_GC[e]

        Iq_pos[e] = Iq_pos_min[e] #Recordar que 'y' es un rango y no un valor

        Ip_pos[e] = Ip_pos_i[e]

        Ip_pos_max[e] = (((V_pos[e]**2)*(Irated**2))/(V_pos[e]**2-(2*V_pos[e]*V_neg[e]*cos_min_phi[e])+V_neg[e]**2))-Iq_pos_min[e]**2

        if Ip_pos_max[e] < 0:

            Ip_pos_max[e] = 0

        else:

            Ip_pos_max[e] = np.sqrt(Ip_pos_max[e]) #EC 23

            if Ip_pos[e] == 0: # Corriente que entrega el inversor (desconectado), si es 0, ip_pos_max también

                Ip_pos_max[e] = 0

        if Ip_pos[e] > Ip_pos_max[e]: #Caso 2

            Ip_pos[e] = Ip_pos_max[e]



        Ip_neg[e] = (V_neg[e]/V_pos[e])*Ip_pos[e]

        Iq_neg[e] = (V_neg[e]/V_pos[e])*Iq_pos[e]

        #Caso 1
Ip_pos; Ip_neg; Iq_pos; Iq_neg #Salidas del diagrama de flujo

plt.xlabel('Posición')
plt.ylabel('Valor Ip+')
plt.title('Valores de Ip+')
plt.ylim(0, 10)

plt.plot(range(0,len(Ip_pos)),np.array(Ip_pos),color = 'red')
```

```
[<matplotlib.lines.Line2D at 0x7fa5ca9c5f10>]
```

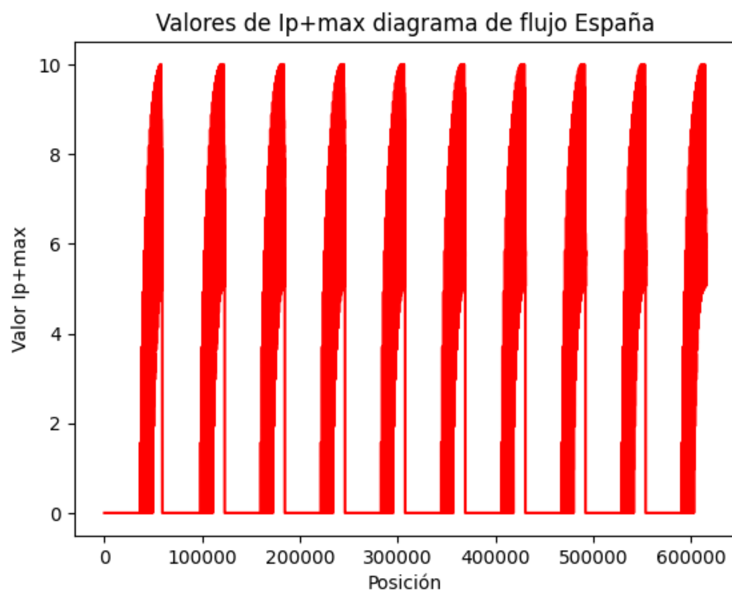



```
np.max(Ip_pos) # Valor máximo de Ip+
```

```
6.942403696161406
```

```
#Gráfica que representa el comportamiento de Ipmax+

plt.xlabel('Posición')
plt.ylabel('Valor Ip+max')
plt.title('Valores de Ip+max diagrama de flujo España')
plt.plot(range(0,len(Ip_pos_max)),np.array(Ip_pos_max),color = 'red')
```

```
[<matplotlib.lines.Line2D at 0x7fa5ca93a160>]
```




```
Ip_pos_max
col = ['Ip_pos_max']
Ip_pos_max_in = pd.DataFrame(np.transpose(Ip_pos_max),columns=col)

Ip_pos_max_in = np.array(Ip_pos_max_in)
```

```
Ipeak
colpeak = ['Imax']
Ipeak_in = pd.DataFrame(np.transpose(Ipeak),columns=colpeak)

Ipeak_in = np.array(Ipeak_in)
```

```
x1cord = x1_vec * len(V_pos_pu)
x2cord = x2_vec * len(V_pos_pu)
y1cord = y1_vec * len(V_pos_pu)
y2cord = y2_vec * len(V_pos_pu)

T_datos=['x1','x2','y1','y2','cos_min_phi_grados','Ip_pos_max','Imax','V+_pu','V-_pu','V+','V-','Desfase','P referencia','Iq+','Iq-','Ip+','I
L_datos=[x1cord,x2cord,y1cord,y2cord,cos_min_phi*180/np.pi,Ip_pos_max,Ipeak,V_pos_pu,V_neg_pu,V_pos,V_neg,data_intro[:,7],data_intro[:,4],Iq_
df_data=pd.DataFrame(np.transpose(L_datos),columns=T_datos)

#BASE DE DATOS

df_data
```

	x1	x2	y1	y2	cos_min_phi_grados	Ip_pos_max	Imax	V+_pu	V-_pu	
0	0.5	0.95	1.0	0.0	-57.267508	0.000000	10.0	0.046	0.00	7.1
1	0.5	0.95	1.0	0.0	-54.616825	0.000000	10.0	0.046	0.00	7.1
2	0.5	0.95	1.0	0.0	-47.844266	0.000000	10.0	0.046	0.00	7.1
3	0.5	0.95	1.0	0.0	-37.460948	0.000000	10.0	0.046	0.00	7.1
4	0.5	0.95	1.0	0.0	-32.830735	0.000000	10.0	0.046	0.00	7.1
...	...	...	...	...	...	...	...	...	...	
614995	0.5	0.95	1.0	0.0	-43.794545	5.383690	0.0	0.982	0.96	152.7
614996	0.5	0.95	1.0	0.0	-31.939839	5.730091	0.0	0.982	0.96	152.7
614997	0.5	0.95	1.0	0.0	-38.405195	5.533181	0.0	0.982	0.96	152.7
614998	0.5	0.95	1.0	0.0	-48.621345	5.259606	0.0	0.982	0.96	152.7
614999	0.5	0.95	1.0	0.0	-55.084500	5.106143	0.0	0.982	0.96	152.7

NOMBRE DEL ARCHIVO QUE CONTIENE LA BASE DE DATOS DESEADA

```
#Se define el nombre del archivo de la siguiente manera:
#BD_GC_1
#Si es un código de red real:
#BD_NombreDelPaís

df_data.to_csv('BD_GC_26', compression='gzip', index=False)
```

APENDICE B

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Código de la red neuronal

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

```
#Se importan las librerías

import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras import metrics
from sklearn.preprocessing import StandardScaler
```

```
#Se leen los 30 archivos correspondientes a las bases de datos

data1 = df = pd.read_csv('BD_ESPAÑA', compression='gzip')
data2 = df = pd.read_csv('BD_ALEMANIA', compression='gzip')
data3 = df = pd.read_csv('BD_DINAMARCA', compression='gzip')
data4 = df = pd.read_csv('BD_EON', compression='gzip')

data5 = df = pd.read_csv('BD_GC_1', compression='gzip')
data6 = df = pd.read_csv('BD_GC_2', compression='gzip')
data7 = df = pd.read_csv('BD_GC_3', compression='gzip')
data8 = df = pd.read_csv('BD_GC_4', compression='gzip')
data9 = df = pd.read_csv('BD_GC_5', compression='gzip')

data10 = df = pd.read_csv('BD_GC_6', compression='gzip')
data11 = df = pd.read_csv('BD_GC_7', compression='gzip')
data12 = df = pd.read_csv('BD_GC_8', compression='gzip')
data13 = df = pd.read_csv('BD_GC_9', compression='gzip')

data14 = df = pd.read_csv('BD_GC_10', compression='gzip')
data15 = df = pd.read_csv('BD_GC_11', compression='gzip')
data16 = df = pd.read_csv('BD_GC_12', compression='gzip')
data17 = df = pd.read_csv('BD_GC_13', compression='gzip')

data18 = df = pd.read_csv('BD_GC_14', compression='gzip')
data19 = df = pd.read_csv('BD_GC_15', compression='gzip')
data20 = df = pd.read_csv('BD_GC_16', compression='gzip')
data21 = df = pd.read_csv('BD_GC_17', compression='gzip')

data22 = df = pd.read_csv('BD_GC_18', compression='gzip')
data23 = df = pd.read_csv('BD_GC_19', compression='gzip')
data24 = df = pd.read_csv('BD_GC_20', compression='gzip')
data25 = df = pd.read_csv('BD_GC_21', compression='gzip')

data26 = df = pd.read_csv('BD_GC_22', compression='gzip')
data27 = df = pd.read_csv('BD_GC_23', compression='gzip')
data28 = df = pd.read_csv('BD_GC_24', compression='gzip')
data29 = df = pd.read_csv('BD_GC_25', compression='gzip')
data30 = df = pd.read_csv('BD_GC_26', compression='gzip')
```

```
#Se concatenan las 30 bases de datos en una sola matriz

data = pd.concat([data1,data2,data3,data4,data5,data6,data7,data8,data9,data10,data11, data12, data13, data14, data15, data16, data17, data18
                  data19, data20, data21, data22, data23, data24, data25, data26, data27, data28, data29, data30])

data
```

	x1	x2	y1	y2	cos_min_phi_grados	Ip_pos_max	Imax	V+_pu	V-_pu	
0	0.5	0.85	0.9	0.0	-57.267508	4.358899	10.0	0.046	0.00	7.1
1	0.5	0.85	0.9	0.0	-54.616825	4.358899	10.0	0.046	0.00	7.1
2	0.5	0.85	0.9	0.0	-47.844266	4.358899	10.0	0.046	0.00	7.1
3	0.5	0.85	0.9	0.0	-37.460948	4.358899	10.0	0.046	0.00	7.1
4	0.5	0.85	0.9	0.0	-32.830735	4.358899	10.0	0.046	0.00	7.1
...	...	...	...	...	...	...	...	...	...	
614995	0.5	0.95	1.0	0.0	-43.794545	5.383690	0.0	0.982	0.96	152.7
614996	0.5	0.95	1.0	0.0	-31.939839	5.730091	0.0	0.982	0.96	152.7
614997	0.5	0.95	1.0	0.0	-38.405195	5.533181	0.0	0.982	0.96	152.7
614998	0.5	0.95	1.0	0.0	-48.621345	5.259606	0.0	0.982	0.96	152.7

```
# Se separan las entradas y las salidas DE LA RED NEURONAL

entradas = data[data.columns[7:13]].copy()
salidas = data[data.columns[13:]].copy()

salidas
```

	Iq+	Iq-	Ip+	Ip-
0	10.0	0.0	0.000000	0.000000
1	10.0	0.0	0.000000	0.000000
2	10.0	0.0	0.000000	0.000000
3	10.0	0.0	0.000000	0.000000
4	10.0	0.0	0.000000	0.000000
...	...	...	...	...
614995	0.0	0.0	5.383690	5.263078
614996	0.0	0.0	5.730091	5.601719
614997	0.0	0.0	5.533181	5.409220
614998	0.0	0.0	5.259606	5.141774
614999	0.0	0.0	5.106143	4.991749

18450000 rows × 4 columns

```
#Se normalizan los datos de entrada y de salida DE LA RED NEURONAL a excepción de las coordenadas

#ENTRADA

scalerE = StandardScaler()
entradas2 = scalerE.fit_transform(entradas)

#SALIDA
scalerS =StandardScaler()
salidas2 = scalerS.fit_transform(salidas)

#Se convierten en un dataframe para observar mejor los datos

columnasE = ['V+_pu','V-_pu','V+', 'V-', 'Desfase', 'P referencia']
entradas3 = pd.DataFrame(entradas2, columns=columnasE)

columnasS = ['Iq+','Iq-','Ip+','Ip-' ]
salidas3 = pd.DataFrame(salidas2, columns=columnasS)

entradas4 = data[data.columns[0:7]].copy()

entradas3 = entradas3.reset_index(drop=True)
entradas4 = entradas4.reset_index(drop=True)

#Se concatenan los datos de las coordenadas y los datos correspondientes a x, Ipmax+ e Imax
entradas5 = pd.concat([entradas4, entradas3], axis=1)
```

```
entradas5
```

```
##Comprobando la desnormalizacion de datos
#salidas4 = scalerS.inverse_transform(salidas3)
#salidas4 = pd.DataFrame(salidas4, columns=columnasS)
```

	x1	x2	y1	y2	cos_min_phi_grados	Ip_pos_max	Imax	V+_pu	V-
0	0.5	0.85	0.9	0.0	-57.267508	4.358899	10.0	-1.201577	-1.603
1	0.5	0.85	0.9	0.0	-54.616825	4.358899	10.0	-1.201577	-1.603
2	0.5	0.85	0.9	0.0	-47.844266	4.358899	10.0	-1.201577	-1.603
3	0.5	0.85	0.9	0.0	-37.460948	4.358899	10.0	-1.201577	-1.603
4	0.5	0.85	0.9	0.0	-32.830735	4.358899	10.0	-1.201577	-1.603
...	...	...	...	...	...	...	...	...	
18449995	0.5	0.95	1.0	0.0	-43.794545	5.383690	0.0	1.839896	1.816
18449996	0.5	0.95	1.0	0.0	-31.939839	5.730091	0.0	1.839896	1.816
18449997	0.5	0.95	1.0	0.0	-38.405195	5.533181	0.0	1.839896	1.816
18449998	0.5	0.95	1.0	0.0	-48.621345	5.259606	0.0	1.839896	1.816
18449999	0.5	0.95	1.0	0.0	-55.084500	5.106143	0.0	1.839896	1.816

```
#División de la base de datos entre datos en entrenamiento, validación y testeo

entradasE1, entradasTesteo, salidasE, salidasTesteo = train_test_split(entradas5, salidas3, test_size=0.3, random_state=42) #70% datos de ent
entradasV1, entradasT1, salidasV, salidasT = train_test_split(entradasTesteo, salidasTesteo, test_size=0.5, random_state=42) # 15% deatos de

#Se eliminan las columnas que no hacen parte de la entrada a la red neuronal

entradasE = entradasE1.drop(['cos_min_phi_grados','Ip_pos_max','Imax'],axis=1) #Entrenamiento

entradasV = entradasV1.drop(['cos_min_phi_grados','Ip_pos_max','Imax'], axis=1) #Validación

entradasT = entradasT1.drop(['cos_min_phi_grados','Ip_pos_max','Imax'], axis=1) #Testeo

entradasT #Datos de Testeo
```

	x1	x2	y1	y2	V+_pu	V-_pu	V+	V-	Desfase	re
13414086	0.4	0.90	1.00	0.2	-1.084597	0.819341	-1.084597	0.819341	0.931588	
17489061	0.5	0.85	1.00	0.0	-0.780450	0.819341	-0.780450	0.819341	0.021048	
3527699	0.6	0.85	1.00	0.0	-0.780450	-1.496188	-0.780450	-1.496188	-0.005268	
2214998	0.5	0.90	1.00	0.2	-1.201577	1.104329	-1.201577	1.104329	-0.110529	
16476984	0.6	0.90	0.95	0.2	1.578641	1.531812	1.578641	1.531812	0.231571	
...	...	...	...	...	...	...	...	...	...	
1858900	0.5	0.90	1.00	0.2	-0.944221	-0.641223	-0.944221	-0.641223	-2.058001	
15164373	0.6	0.90	0.95	0.0	0.120294	0.819341	0.120294	0.819341	0.310518	
15842148	0.6	0.95	0.95	0.0	0.206079	0.961835	0.206079	0.961835	1.138611	
1870286	0.5	0.90	1.00	0.2	-0.733658	0.106871	-0.733658	0.106871	0.310518	
4334737	0.6	0.95	1.00	0.0	-0.221872	-1.068706	-0.221872	-1.068706	-0.163160	

```
Is=10
```

```
# Modelo fully connected
#Modelo de la red neuronal
model = tf.keras.Sequential([

  tf.keras.layers.Dense(1000, activation='relu', input_shape=(Is,)),
  tf.keras.layers.Dense(1000, activation='relu'),
  tf.keras.layers.Dense(1000, activation='relu'),
  tf.keras.layers.Dense(1000, activation='relu'),
  tf.keras.layers.Dense(1000, activation='relu'),
  tf.keras.layers.Dense(1000, activation='relu'),
  tf.keras.layers.Dense(1000, activation='relu'),
  tf.keras.layers.Dense(1000, activation='linear'),
  tf.keras.layers.Dense(1000, activation='linear'),
  tf.keras.layers.Dense(1000, activation='linear'),
  tf.keras.layers.Dense(1000, activation='linear'),
  tf.keras.layers.Dense(4)
  ])


#Compilación del modelo

model.compile(loss = 'mae',
              optimizer = tf.keras.optimizers.Adam(1e-6),
              metrics = ['mae', 'accuracy','mse'])


# Se convierten los datos de entrada y validación en arrays

entradasE = np.array(entradasE)
salidasE = np.array(salidasE)

entradasV = np.array(entradasV)
salidasV = np.array(salidasV)
```

Entrenamiento de la red neuronal

```
def scheduler(epoch, lr):
  if epoch < (10):
    return 1e-3
  elif epoch < (20):
    return 1e-4
  elif epoch < (30):
    return 1e-5
  else:
    return 1e-6

print('Comenzando entrenamiento!')
historial=model.fit(entradasE, salidasE, epochs = 40, batch_size = 14760,
                    validation_data = (entradasV, salidasV), shuffle = True,
                    verbose = True,
                    callbacks=[tf.keras.callbacks.LearningRateScheduler(scheduler)])
print('Entremamiento terminado!!!')
```

```
Comenzando entrenamiento!
Epoch 1/40
875/875 [==============================] - 218s 240ms/step - loss: 0.1269 - mae: 0.1269 - accuracy: 0.9192 - mse: 0.1592 - val_loss: (
Epoch 2/40
875/875 [==============================] - 214s 244ms/step - loss: 0.0437 - mae: 0.0437 - accuracy: 0.9755 - mse: 0.0284 - val_loss: (
Epoch 3/40
875/875 [==============================] - 214s 245ms/step - loss: 0.0359 - mae: 0.0359 - accuracy: 0.9789 - mse: 0.0252 - val_loss: (
Epoch 4/40
875/875 [==============================] - 214s 245ms/step - loss: 0.0304 - mae: 0.0304 - accuracy: 0.9810 - mse: 0.0227 - val_loss: (
Epoch 5/40
875/875 [==============================] - 214s 244ms/step - loss: 0.0265 - mae: 0.0265 - accuracy: 0.9833 - mse: 0.0183 - val_loss: (
Epoch 6/40
875/875 [==============================] - 215s 245ms/step - loss: 0.0250 - mae: 0.0250 - accuracy: 0.9840 - mse: 0.0173 - val_loss: (
Epoch 7/40
875/875 [==============================] - 221s 252ms/step - loss: 0.0242 - mae: 0.0242 - accuracy: 0.9846 - mse: 0.0169 - val_loss: (
Epoch 8/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0232 - mae: 0.0232 - accuracy: 0.9852 - mse: 0.0166 - val_loss: (
Epoch 9/40
875/875 [==============================] - 215s 245ms/step - loss: 0.0218 - mae: 0.0218 - accuracy: 0.9857 - mse: 0.0169 - val_loss: (
Epoch 10/40
875/875 [==============================] - 215s 245ms/step - loss: 0.0222 - mae: 0.0222 - accuracy: 0.9862 - mse: 0.0148 - val_loss: (
Epoch 11/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0037 - mae: 0.0037 - accuracy: 0.9979 - mse: 0.0021 - val_loss: (
Epoch 12/40
```

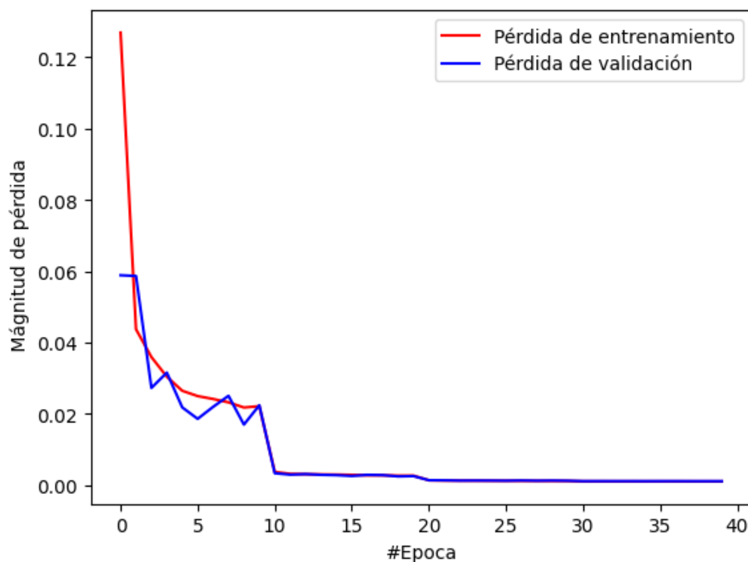
```
875/875 [==============================] - 215s 245ms/step - loss: 0.0032 - mae: 0.0032 - accuracy: 0.9983 - mse: 0.0016 - val_loss: (
Epoch 13/40
875/875 [==============================] - 215s 245ms/step - loss: 0.0031 - mae: 0.0031 - accuracy: 0.9984 - mse: 0.0015 - val_loss: (
Epoch 14/40
875/875 [==============================] - 215s 245ms/step - loss: 0.0030 - mae: 0.0030 - accuracy: 0.9984 - mse: 0.0015 - val_loss: (
Epoch 15/40
875/875 [==============================] - 215s 245ms/step - loss: 0.0030 - mae: 0.0030 - accuracy: 0.9984 - mse: 0.0015 - val_loss: (
Epoch 16/40
875/875 [==============================] - 215s 245ms/step - loss: 0.0029 - mae: 0.0029 - accuracy: 0.9985 - mse: 0.0014 - val_loss: (
Epoch 17/40
875/875 [==============================] - 214s 245ms/step - loss: 0.0028 - mae: 0.0028 - accuracy: 0.9986 - mse: 0.0014 - val_loss: (
Epoch 18/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0028 - mae: 0.0028 - accuracy: 0.9986 - mse: 0.0014 - val_loss: (
Epoch 19/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0027 - mae: 0.0027 - accuracy: 0.9986 - mse: 0.0013 - val_loss: (
Epoch 20/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0027 - mae: 0.0027 - accuracy: 0.9986 - mse: 0.0014 - val_loss: (
Epoch 21/40
875/875 [==============================] - 216s 246ms/step - loss: 0.0014 - mae: 0.0014 - accuracy: 0.9992 - mse: 4.5766e-04 - val_los
Epoch 22/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0013 - mae: 0.0013 - accuracy: 0.9992 - mse: 3.5337e-04 - val_los
Epoch 23/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0013 - mae: 0.0013 - accuracy: 0.9992 - mse: 3.1743e-04 - val_los
Epoch 24/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0013 - mae: 0.0013 - accuracy: 0.9993 - mse: 3.0171e-04 - val_los
Epoch 25/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0013 - mae: 0.0013 - accuracy: 0.9993 - mse: 2.8034e-04 - val_los
Epoch 26/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0013 - mae: 0.0013 - accuracy: 0.9993 - mse: 2.7894e-04 - val_los
Epoch 27/40
875/875 [==============================] - 215s 246ms/step - loss: 0.0012 - mae: 0.0012 - accuracy: 0.9993 - mse: 2.7292e-04 - val_los
Epoch 28/40
```

```
#Se guarda el modelo para a hacer predicciones sin necesidad de entrenar nuevamente

model.save('Neural_Network_PG_LVRT.h5')
```

```
#Se llama el modelo
NNFinal = tf.keras.models.load_model('Neural_Network_PG_LVRT.h5')
```

EVALUACIÓN DEL ENTRENAMIENTO DEL MODELO

```
#Gráfica que representa la función de pérdida a lo largo del entrenamiento por cada epoch
plt.xlabel("#Epoca")
plt.ylabel(" Mágnitud de pérdida")

plt.plot(historial.history["loss"], 'r-', label='Pérdida de entrenamiento')
plt.plot(historial.history["val_loss"], 'b-', label='Pérdida de validación')
plt.legend()
plt.show()
```



EVALUACIÓN DEL NODELO

```
#Evaluación del modelo

loss = model.evaluate(entradasV, salidasV)
```

```
86485/86485 [==============================] - 244s 3ms/step - loss: 0.0011 - mae: 0.0011 - accuracy: 0.9993 - mse: 2.6308e-04
```

```
#Se realizan las predicciones con los datos NORMALIZADOS

print('Prueba de testeo')
resultados_norm = NNFinal.predict(entradasT)
print(resultados_norm)
print('fin test')
```

```
Prueba de testeo
86485/86485 [==============================] - 134s 2ms/step
[[ 0.5361605  -0.4062316  -0.50892574 -0.3848311 ]
 [ 0.5361605  -0.4062316  -0.50892574 -0.3848311 ]
 [ 0.5361605  -0.4062316  -0.50892574 -0.3848311 ]
 ...
 [ 0.5361605  -0.4062316  -0.50892574 -0.3848311 ]
 [ 0.5361605  -0.4062316  -0.50892574 -0.3848311 ]
 [ 0.5361605  -0.4062316  -0.50892574 -0.3848311 ]]
fin test
```

```
#Se desnormalizan las predicciones
resultados = scalerS.inverse_transform(resultados_norm)

columnasS = ['Iq+','Iq-','Ip+','Ip-' ]
resultados = pd.DataFrame(resultados, columns=columnasS)

print(resultados)

salidasT = scalerS.inverse_transform(salidasT)

salidasT = pd.DataFrame(salidasT, columns=columnasS)

print(salidasT)
```

```
               Iq+       Iq-       Ip+       Ip-
0        10.000041  0.000015  0.000025 -0.000005
1        10.000041  0.000015  0.000025 -0.000005
2        10.000041  0.000015  0.000025 -0.000005
3        10.000041  0.000015  0.000025 -0.000005
4         0.002008  0.000552  5.502898  5.363133
...            ...       ...       ...       ...
2767495  10.000041  0.000015  0.000025 -0.000005
2767496  10.000041  0.000015  0.000025 -0.000005
2767497  10.000041  0.000015  0.000025 -0.000005
2767498  10.000041  0.000015  0.000025 -0.000005
2767499  10.000041  0.000015  0.000025 -0.000005

[2767500 rows x 4 columns]
          Iq+  Iq-           Ip+       Ip-
0        10.0  0.0 -1.110223e-16  0.000000
1        10.0  0.0 -1.110223e-16  0.000000
2        10.0  0.0 -1.110223e-16  0.000000
3        10.0  0.0 -1.110223e-16  0.000000
4         0.0  0.0  5.493321e+00  5.361715
...       ...  ...           ...       ...
2767495  10.0  0.0 -1.110223e-16  0.000000
2767496  10.0  0.0 -1.110223e-16  0.000000
2767497  10.0  0.0 -1.110223e-16  0.000000
2767498  10.0  0.0 -1.110223e-16  0.000000
2767499  10.0  0.0 -1.110223e-16  0.000000

[2767500 rows x 4 columns]
```

```
salidasT #Valores reales de las salidas de la base de datos
```

	Iq+	Iq-	Ip+	Ip-
0	10.0	0.0	-1.110223e-16	0.000000
1	10.0	0.0	-1.110223e-16	0.000000
2	10.0	0.0	-1.110223e-16	0.000000
3	10.0	0.0	-1.110223e-16	0.000000
4	0.0	0.0	5.493321e+00	5.361715
...	...	...	...	...
2767495	10.0	0.0	-1.110223e-16	0.000000
2767496	10.0	0.0	-1.110223e-16	0.000000
2767497	10.0	0.0	-1.110223e-16	0.000000

```
resultados #Valores predichos por la red neuronal
```

	Iq+	Iq-	Ip+	Ip-
0	10.000041	0.000015	0.000025	-0.000005
1	10.000041	0.000015	0.000025	-0.000005
2	10.000041	0.000015	0.000025	-0.000005
3	10.000041	0.000015	0.000025	-0.000005
4	0.002008	0.000552	5.502898	5.363133
...	...	...	...	...
2767495	10.000041	0.000015	0.000025	-0.000005
2767496	10.000041	0.000015	0.000025	-0.000005
2767497	10.000041	0.000015	0.000025	-0.000005
2767498	10.000041	0.000015	0.000025	-0.000005
2767499	10.000041	0.000015	0.000025	-0.000005

2767500 rows × 4 columns

EVALUACIÓN DE LOS PARÁMETROS INDIVIDUALES DEL MODELO

Errores obtenidos de los 4 parámetros de salida de la red neuronal, teniendo en cuenta los 30 códigos de red.

```
Vreales = np.array(salidasT)
Vobtenidos = np.array(resultados)


error_absoluto_1 = np.abs(Vobtenidos[:,0] - Vreales[:,0])
mean_error_1 = np.mean(error_absoluto_1)

error_absoluto_2 = np.abs(Vobtenidos[:,1] - Vreales[:,1])
mean_error_2 = np.mean(error_absoluto_2)

error_absoluto_3 = np.abs(Vobtenidos[:,2] - Vreales[:,2])
mean_error_3 = np.mean(error_absoluto_3)

error_absoluto_4 = np.abs(Vobtenidos[:,3] - Vreales[:,3])
mean_error_4 = np.mean(error_absoluto_4)

print(mean_error_1) #Error absoluto medio Iq+
print(mean_error_2) #Error absoluto medio Iq-
print(mean_error_3) #Error absoluto medio Ip+
print(mean_error_4) #Error absoluto medio Ip-
```

```
0.0027395919918993725
0.0017728349898846732
0.0021544383122144677
0.0011678766573570971
```

```
error_cuadratico_1 = (Vobtenidos[:,0] - Vreales[:,0]) ** 2
error_cuadratico_medio_1 = np.mean(error_cuadratico_1)
```

```
error_cuadratico_2 = (Vobtenidos[:,1] - Vreales[:,1]) ** 2
error_cuadratico_medio_2 = np.mean(error_cuadratico_2)

error_cuadratico_3 = (Vobtenidos[:,2] - Vreales[:,2]) ** 2
error_cuadratico_medio_3 = np.mean(error_cuadratico_3)

error_cuadratico_4 = (Vobtenidos[:,3] - Vreales[:,3]) ** 2
error_cuadratico_medio_4 = np.mean(error_cuadratico_4)

print(error_cuadratico_medio_1) #Error cuadrático medio Iq+
print(error_cuadratico_medio_2) #Error cuadrático medio Iq-
print(error_cuadratico_medio_3) #Error cuadrático medio Ip+
print(error_cuadratico_medio_4) #Error cuadrático medio Iq-
```

```
0.0010419525612792846
0.0014136534287684905
8.297339385268207e-05
2.321039826977194e-05
```

```
media_reales_1 = np.mean(Vreales[:,0])
ss_tot_1 = np.sum((Vreales[:,0] - media_reales_1) ** 2)
ss_res_1 = np.sum((Vobtenidos[:,0] - Vreales[:,0]) ** 2)
r_cuadrado_1 = 1 - (ss_res_1 / ss_tot_1)

media_reales_2 = np.mean(Vreales[:,1])
ss_tot_2 = np.sum((Vreales[:,1] - media_reales_2) ** 2)
ss_res_2 = np.sum((Vobtenidos[:,1] - Vreales[:,1]) ** 2)
r_cuadrado_2 = 1 - (ss_res_2 / ss_tot_2)

media_reales_3 = np.mean(Vreales[:,2])
ss_tot_3 = np.sum((Vreales[:,2] - media_reales_3) ** 2)
ss_res_3 = np.sum((Vobtenidos[:,2] - Vreales[:,2]) ** 2)
r_cuadrado_3 = 1 - (ss_res_3 / ss_tot_3)

media_reales_4 = np.mean(Vreales[:,3])
ss_tot_4 = np.sum((Vreales[:,3] - media_reales_4) ** 2)
ss_res_4 = np.sum((Vobtenidos[:,3] - Vreales[:,3]) ** 2)
r_cuadrado_4 = 1 - (ss_res_4 / ss_tot_4)

print(r_cuadrado_1) # Coeficiente de determinación Iq+
print(r_cuadrado_2) # Coeficiente de determinación Iq-
print(r_cuadrado_3) # Coeficiente de determinación Ip+
print(r_cuadrado_4) # Coeficiente de determinación Ip-
```

```
0.9999036543758001
0.9991441725004239
0.9999709948269848
0.9999790576159702
```

Media, mediana y desviación estándar de los datos, teniendo en cuenta los 30 códigos de red

```
media_1O = np.mean(Vobtenidos[:,0])
media_1R = np.mean(Vreales[:,0])

media_2O = np.mean(Vobtenidos[:,1])
media_2R = np.mean(Vreales[:,1])

media_3O = np.mean(Vobtenidos[:,2])
media_3R = np.mean(Vreales[:,2])

media_4O = np.mean(Vobtenidos[:,3])
media_4R = np.mean(Vreales[:,3])

print(media_1O) #media Iq+ obtenidos
print(media_1R) #media Iq+ reales
print(media_2O) #media Iq- obtenidos
print(media_2R) #media Iq- reales
print(media_3O) #media Ip+ obtenidos
print(media_3R) #media Ip+ reales
print(media_4O) #media Ip- obtenidos
print(media_4R) #media Ip- reales
```

```
8.235153
8.23497314369751
```

```
0.52271056
0.5227942355194143
0.8617514
0.8617897221947052
0.40560308
0.4056472673824486
```

```
median_1O = np.median(Vobtenidos[:,0])
median_1R = np.median(Vreales[:,0])

median_2O = np.median(Vobtenidos[:,1])
median_2R = np.median(Vreales[:,1])

median_3O = np.median(Vobtenidos[:,2])
median_3R = np.median(Vreales[:,2])

median_4O = np.median(Vobtenidos[:,3])
median_4R = np.median(Vreales[:,3])

print(median_1O) #mediana Iq+ obtenidos
print(median_1R) #mediana Iq+ reales
print(median_2O) #mediana Iq- obtenidos
print(median_2R) #mediana Iq- reales
print(median_3O) #mediana Ip+ obtenidos
print(median_3R) #mediana Ip+ reales
print(median_4O) #mediana Ip- obtenidos
print(median_4R) #mediana Ip- reales
```

```
10.000041
10.0
1.51423865e-05
0.0
2.4997107e-05
-1.1102230246251565e-16
-4.6683167e-06
0.0
```

```
std_1O = np.std(Vobtenidos[:,0])
std_1R = np.std(Vreales[:,0])

std_2O = np.std(Vobtenidos[:,1])
std_2R = np.std(Vreales[:,1])

std_3O = np.std(Vobtenidos[:,2])
std_3R = np.std(Vreales[:,2])

std_4O = np.std(Vobtenidos[:,3])
std_4R = np.std(Vreales[:,3])

print(std_1O) #desviación estándar Iq+ obtenidos
print(std_1R) #desviación estándar Iq+ reales
print(std_2O) #desviación estándar Iq- obtenidos
print(std_2R) #desviación estándar Iq- reales
print(std_3O) #desviación estándar Ip+ obtenidos
print(std_3R) #desviación estándar Ip+ reales
print(std_4O) #desviación estándar Ip- obtenidos
print(std_4R) #desviación estándar Ip- reales
```

```
3.2883961
3.288576703510465
1.2847905
1.2852226090307808
1.6912463
1.6913430259558817
1.052695
1.052757239757672
```

Se concatenan las columnas que contienen las coordenadas a las salidas para identificar cada código de red

```
cordenadas = entradasT[entradasT.columns[0:4]].copy()
cordenadasT = cordenadas.reset_index(drop=True)
salidasT = salidasT.reset_index(drop=True)
resultados = resultados.reset_index(drop=True)

salidasTcord = pd.concat([cordenadasT, salidasT], axis=1)
resultadoscord = pd.concat([cordenadasT, resultados], axis=1)
```

```
salidasTcord # salidas de la base de datos incluyendo las coordenadas
```

	x1	x2	y1	y2	Iq+	Iq-	Ip+	Ip-
0	0.4	0.90	1.00	0.2	10.0	0.0	-1.110223e-16	0.000000
1	0.5	0.85	1.00	0.0	10.0	0.0	-1.110223e-16	0.000000
2	0.6	0.85	1.00	0.0	10.0	0.0	-1.110223e-16	0.000000
3	0.5	0.90	1.00	0.2	10.0	0.0	-1.110223e-16	0.000000
4	0.6	0.90	0.95	0.2	0.0	0.0	5.493321e+00	5.361715
...	...	...	...	...	...	...	...	...
2767495	0.5	0.90	1.00	0.2	10.0	0.0	-1.110223e-16	0.000000
2767496	0.6	0.90	0.95	0.0	10.0	0.0	-1.110223e-16	0.000000
2767497	0.6	0.95	0.95	0.0	10.0	0.0	-1.110223e-16	0.000000
2767498	0.5	0.90	1.00	0.2	10.0	0.0	-1.110223e-16	0.000000
2767499	0.6	0.95	1.00	0.0	10.0	0.0	-1.110223e-16	0.000000

2767500 rows × 8 columns

```
resultadoscord # salidas de la red neuronal incluyendo las coordenadas
```

	x1	x2	y1	y2	Iq+	Iq-	Ip+	Ip-
0	0.4	0.90	1.00	0.2	10.000041	0.000015	0.000025	-0.000005
1	0.5	0.85	1.00	0.0	10.000041	0.000015	0.000025	-0.000005
2	0.6	0.85	1.00	0.0	10.000041	0.000015	0.000025	-0.000005
3	0.5	0.90	1.00	0.2	10.000041	0.000015	0.000025	-0.000005
4	0.6	0.90	0.95	0.2	0.002008	0.000552	5.502898	5.363133
...	...	...	...	...	...	...	...	...
2767495	0.5	0.90	1.00	0.2	10.000041	0.000015	0.000025	-0.000005
2767496	0.6	0.90	0.95	0.0	10.000041	0.000015	0.000025	-0.000005
2767497	0.6	0.95	0.95	0.0	10.000041	0.000015	0.000025	-0.000005
2767498	0.5	0.90	1.00	0.2	10.000041	0.000015	0.000025	-0.000005
2767499	0.6	0.95	1.00	0.0	10.000041	0.000015	0.000025	-0.000005

2767500 rows × 8 columns

```
entradasT # entradas de la red neuronal incluyendo las coordenadas
```

	x1	x2	y1	y2	V+_pu	V-_pu	V+	V-	Desfase	re
13414086	0.4	0.90	1.00	0.2	-1.084597	0.819341	-1.084597	0.819341	0.931588	
17489061	0.5	0.85	1.00	0.0	-0.780450	0.819341	-0.780450	0.819341	0.021048	

A continuación se separan los datos de cada código de red

```
# En la siguiente línea de código se utilizan las coordenadas de cada código de red
# Para obtener por separado los datos de cada código y poder evaluarlos individualmente
# Por esta razón anteriormente se agregaron las columnas correspondientes a las coordenadas
# En los datos de entrada, salidas reales y salidas obtenidas.

# Por ejemplo, en este caso la función np.where separa en una matriz diferente las filas que tienen
# los valores 0.5, 0.85, 0.9 y 0 (coordenadas x1, x2, y1 y y2 de España)
# en sus primeras cuatro columnas, respectivamente.

# Este procedimiento se realiza en las entradas, las salidas reales y las salidas obtenidas.

mascaraESPsalidastcord = np.where(np.all(np.array(salidasTcord)[:,:4] == [0.5,0.95,1,0], axis=1))
salidasTESP = np.array(salidasTcord)[mascaraESPsalidastcord]

columnasESPA = ['x1','x2','y1','y2','Iq+','Iq-','Ip+','Ip-' ]
salidasTESP = pd.DataFrame(salidasTESP, columns=columnasESPA)

salidasTESP
```

	x1	x2	y1	y2	Iq+	Iq-	Ip+	Ip-
0	0.5	0.95	1.0	0.0	8.410987	1.333666	3.252869e+00	0.515783
1	0.5	0.95	1.0	0.0	10.000000	0.000000	-1.110223e-16	0.000000
2	0.5	0.95	1.0	0.0	7.295014	2.427356	3.206996e+00	1.067102
3	0.5	0.95	1.0	0.0	10.000000	0.000000	-1.110223e-16	0.000000
4	0.5	0.95	1.0	0.0	3.422222	2.751535	4.361500e+00	3.506734
...	...	...	...	...	...	...	...	...
91789	0.5	0.95	1.0	0.0	8.102980	1.017394	3.644647e+00	0.457615
91790	0.5	0.95	1.0	0.0	10.000000	0.000000	-1.110223e-16	0.000000
91791	0.5	0.95	1.0	0.0	4.858786	5.753138	-1.110223e-16	0.000000
91792	0.5	0.95	1.0	0.0	8.290263	2.161424	1.997991e+00	0.520913
91793	0.5	0.95	1.0	0.0	10.000000	0.000000	-1.110223e-16	0.000000

91794 rows × 8 columns

```
mascaraESPresultadoscord = np.where(np.all(np.array(resultadoscord)[:,:4] == [0.5,0.95,1,0], axis=1))
resultadosESP = np.array(resultadoscord)[mascaraESPresultadoscord]

columnasESPA = ['x1','x2','y1','y2','Iq+','Iq-','Ip+','Ip-' ]
resultadosESP = pd.DataFrame(resultadosESP, columns=columnasESPA)

resultadosESP
```

	x1	x2	y1	y2	Iq+	Iq-	Ip+	Ip-
0	0.5	0.95	1.0	0.0	8.397631	1.341849	3.251210	0.521567

```
mascaraESPentradasT = np.where(np.all(np.array(entradasT1)[:,:4] == [0.5,0.95,1,0], axis=1))
entradasTESP = np.array(entradasT1)[mascaraESPentradasT]

columnasESPAET = ['x1','x2','y1','y2','cos_min_phi_grados','Ip_pos_max','Imax','V+_pu','V-_pu','V+', 'V-', 'Desfase', 'P referencia' ]
entradasTESP = pd.DataFrame(entradasTESP, columns=columnasESPAET)

entradasTESP
```

	x1	x2	y1	y2	cos_min_phi_grados	Ip_pos_max	Imax	V+_pu	V-_pu
0	0.5	0.95	1.0	0.0	-36.943797	9.017647	10.0	1.722916	-1.068706
1	0.5	0.95	1.0	0.0	-38.405195	0.000000	10.0	-1.178181	-0.961835
2	0.5	0.95	1.0	0.0	-39.951787	7.895900	10.0	1.578641	-0.534353
3	0.5	0.95	1.0	0.0	-42.759254	0.000000	10.0	-1.131389	-0.854965
4	0.5	0.95	1.0	0.0	-57.267508	4.361500	10.0	1.235501	0.676847
...	...	...	...	...	...	...	...	...	...
91789	0.5	0.95	1.0	0.0	-57.267508	7.217085	10.0	0.978145	-1.282447
91790	0.5	0.95	1.0	0.0	-42.759254	0.000000	10.0	-0.051276	-0.961835
91791	0.5	0.95	1.0	0.0	-44.369430	1.577553	10.0	1.063931	1.531812
91792	0.5	0.95	1.0	0.0	-33.750384	6.280156	10.0	0.892360	-0.961835
91793	0.5	0.95	1.0	0.0	-28.647890	0.000000	10.0	-0.221872	1.389317

```
# Se desnormalizan los datos de entradas que se normalizaron antes del entrenamiento
# Esto con el fin de poder identificar a cual entrada corresponde cada salida obtenida
# Y realizar correctamente la evaluación de los resultados

entradasTESPdesnorm = scalerE.inverse_transform(entradasTESP[entradasTESP.columns[7:13]].copy())
columnasTESPdesnorm = ['V+_pu','V-_pu','V+', 'V-', 'Desfase', 'P referencia' ]

entradasTESPdesnorm = pd.DataFrame(entradasTESPdesnorm, columns=columnasTESPdesnorm)

entradasTESPdesnorm
```

	V+_pu	V-_pu	V+	V-	Desfase	P referencia
0	0.9460	0.15	147.163063	23.334524	1.015000e+01	700.0
1	0.0532	0.18	8.275978	28.001429	1.320900e+02	900.0
2	0.9016	0.30	140.256044	46.669048	-1.421000e+01	600.0
3	0.0676	0.21	10.516092	32.668333	1.827000e+01	500.0
4	0.7960	0.64	123.828540	99.560635	-1.782000e+02	700.0
...	...	...	...	...	...	...
91789	0.7168	0.09	111.507911	14.000714	-1.782000e+02	600.0
91790	0.4000	0.18	62.225397	28.001429	1.827000e+01	200.0
91791	0.7432	0.88	115.614787	136.895873	-9.925000e+01	800.0
91792	0.6904	0.18	107.401035	28.001429	-6.090000e+00	300.0
91793	0.3475	0.84	54.058313	130.673333	1.065814e-14	200.0

91794 rows × 6 columns

```
# Se concatenan las columnas que contienen los valores de x, Ipmax+ e Imax
# Nuevamente, esto se realiza con el fin de evaluar los resultados obtenidos
# teniendo en cuenta las entradas y los valores de x, Ipmax+ e Imax que corresponden
# a cada salida.
```

```
ENTRADAScosminphi = entradasTESP[entradasTESP.columns[4:7]]
ENTRADAScosminphiF = ENTRADAScosminphi.reset_index(drop=True)


ENTRADAScosminphi1 = pd.concat([entradasTESPdesnorm, ENTRADAScosminphiF], axis=1)
ENTRADAScosminphi1
```

	V+_pu	V-_pu	V+	V-	Desfase	P referencia	cos_min_phi_
0	0.9460	0.15	147.163063	23.334524	1.015000e+01	700.0	-36.9
1	0.0532	0.18	8.275978	28.001429	1.320900e+02	900.0	-38.4
2	0.9016	0.30	140.256044	46.669048	-1.421000e+01	600.0	-39.9
3	0.0676	0.21	10.516092	32.668333	1.827000e+01	500.0	-42.7
4	0.7960	0.64	123.828540	99.560635	-1.782000e+02	700.0	-57.2
...	...	...	...	...	...	...	
91789	0.7168	0.09	111.507911	14.000714	-1.782000e+02	600.0	-57.2
91790	0.4000	0.18	62.225397	28.001429	1.827000e+01	200.0	-42.7
91791	0.7432	0.88	115.614787	136.895873	-9.925000e+01	800.0	-44.3
91792	0.6904	0.18	107.401035	28.001429	-6.090000e+00	300.0	-33.7
91793	0.3475	0.84	54.058313	130.673333	1.065814e-14	200.0	-28.6

```
# Se exporta a excel los archivos de valores reales, valores obtenidos y entradas
# de cada combinación predicha para luego evaluar los resultados.
# En este caso se exportan los resultados del código de red de españa.
# Si se desean los resultados de otro código de red se debe cambiar las coordenadas
# En la función np.where mencionada anteriormente y cambiar el nombre de los archivos en las
# Siguientes líneas de código.

salidasTESP.to_csv('BD_GC26_Resultados_Reales', index=False)
resultadosESP.to_csv('BD_GC26_Resultados_Obtenidos', index=False)
ENTRADAScosminphi1.to_csv('BD_GC26_Entradas_Testeo', index=False)
```

APENDICE C

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Código para la evaluación de los parámetros de salida (Iq+, Iq-, Ip+ e Ip-) y verificación del tercer objetivo de control

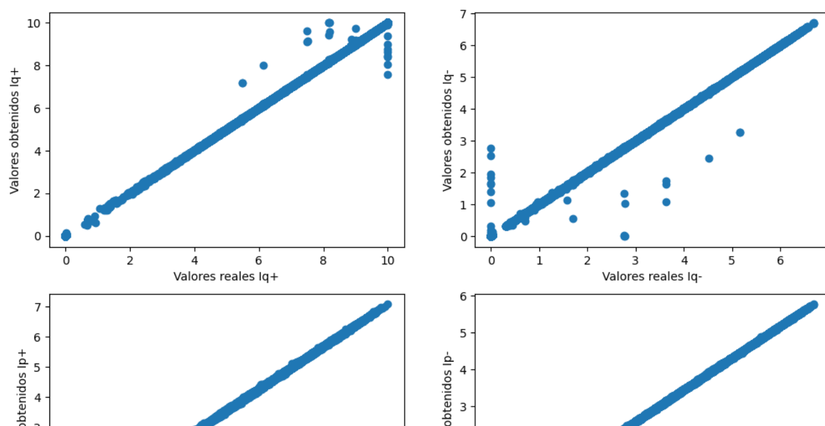
Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

```
#Se importan las librerías

import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras import metrics
from sklearn.preprocessing import StandardScaler
```

```
#Se leen las bases de datos de los resultados de salida reales y obtenidos, al igual que sus respectivas entradas, x, Ipmax+ e Imax

Ro = pd.read_csv('BD_España_Resultados_Obtenidos') #Resultados Obtenidos
Rr = pd.read_csv('BD_España_Resultados_Reales') #Resultados Reales
datain = pd.read_csv('BD_España_Entradas_Testeo')
```

```
#Diagrama de dispersión que compara las corrientes obtenidas con respecto a las corrientes reales

Vobtenidos = np.array(Ro)
Vreales = np.array(Rr)
categoria = list(range(0,len(Vobtenidos)))

fig, ax = plt.subplots(2, 2, figsize=(12, 8))

ax[0,0].scatter(Vreales[:,4], Vobtenidos[:,4])
ax[0,0].set_xlabel('Valores reales Iq+')
ax[0,0].set_ylabel('Valores obtenidos Iq+')

ax[0,1].scatter(Vreales[:,5], Vobtenidos[:,5])
ax[0,1].set_xlabel('Valores reales Iq-')
ax[0,1].set_ylabel('Valores obtenidos Iq-')

ax[1,0].scatter(Vreales[:,6], Vobtenidos[:,6])
ax[1,0].set_xlabel('Valores reales Ip+')
ax[1,0].set_ylabel('Valores obtenidos Ip+')

ax[1,1].scatter(Vreales[:,7], Vobtenidos[:,7])
ax[1,1].set_xlabel('Valores reales Ip-')
ax[1,1].set_ylabel('Valores obtenidos Ip-')




fig.suptitle('Valores obtenidos Vs valores reales')

#plt.scatter(categoria, resultados2[:,0], color='blue', label='resultados obtenidos')
#plt.scatter(categoria, salidasT20[:,0], color='red', label='resultados esperados')
plt.show()
```




```
datainarray = np.array(datain)
```

Error absoluto medio, error cuadrático medio y coeficiente de determinación para los parámetro de salida del codigo de red seelccionado


```
# Calculo del error absoluto para cada una de las corrientes obtenidas para el códido de red seleccionado
# En este caso, el código de red de España

error_absoluto_1 = np.abs(Vobtenidos[:,4] - Vreales[:,4])
mean_error_1 = np.mean(error_absoluto_1)

error_absoluto_2 = np.abs(Vobtenidos[:,5] - Vreales[:,5])
mean_error_2 = np.mean(error_absoluto_2)

error_absoluto_3 = np.abs(Vobtenidos[:,6] - Vreales[:,6])
mean_error_3 = np.mean(error_absoluto_3)

error_absoluto_4 = np.abs(Vobtenidos[:,7] - Vreales[:,7])
mean_error_4 = np.mean(error_absoluto_4)

print(mean_error_1) #Error medio absoluto Iq+
print(mean_error_2) #Error medio absoluto Iq-
print(mean_error_3) #Error medio absoluto Ip+
print(mean_error_4) #Error medio absoluto Ip-
```

```
0.002677345335746742
0.0017215997816560928
0.002160123303657292
0.0011815325334850453
```

```
# Calculo del error cuadrático medio para cada una de las corrientes obtenidas

error_cuadratico_1 = (Vobtenidos[:,4] - Vreales[:,4]) ** 2
error_cuadratico_medio_1 = np.mean(error_cuadratico_1)

error_cuadratico_2 = (Vobtenidos[:,5] - Vreales[:,5]) ** 2
error_cuadratico_medio_2 = np.mean(error_cuadratico_2)

error_cuadratico_3 = (Vobtenidos[:,6] - Vreales[:,6]) ** 2
error_cuadratico_medio_3 = np.mean(error_cuadratico_3)


error_cuadratico_4 = (Vobtenidos[:,7] - Vreales[:,7]) ** 2
error_cuadratico_medio_4 = np.mean(error_cuadratico_4)

print(error_cuadratico_medio_1) #Error cuadrático medio Iq+
print(error_cuadratico_medio_2) #Error cuadrático medio Iq-
print(error_cuadratico_medio_3) #Error cuadrático medio Ip+
print(error_cuadratico_medio_4) #Error cuadrático medio Ip-
```

```
0.0006127453634348273
0.0009342121014639736
7.73246301233084e-05
1.8953975273598696e-05
```

```
#Calculo del coeficiente de determinación (R2)

media_reales_1 = np.mean(Vreales[:,4])
ss_tot_1 = np.sum((Vreales - media_reales_1) ** 2)
ss_res_1 = np.sum((Vobtenidos[:,4] - Vreales[:,4]) ** 2)
r_cuadrado_1 = 1 - (ss_res_1 / ss_tot_1)

media_reales_2 = np.mean(Vreales[:,5])
ss_tot_2 = np.sum((Vreales - media_reales_2) ** 2)
ss_res_2 = np.sum((Vobtenidos[:,5] - Vreales[:,5]) ** 2)
r_cuadrado_2 = 1 - (ss_res_2 / ss_tot_2)

media_reales_3 = np.mean(Vreales[:,6])
ss_tot_3 = np.sum((Vreales - media_reales_3) ** 2)
ss_res_3 = np.sum((Vobtenidos[:,6] - Vreales[:,6]) ** 2)
r_cuadrado_3 = 1 - (ss_res_3 / ss_tot_3)

media_reales_4 = np.mean(Vreales[:,7])
ss_tot_4 = np.sum((Vreales - media_reales_3) ** 2)
ss_res_4 = np.sum((Vobtenidos[:,7] - Vreales[:,7]) ** 2)
r_cuadrado_4 = 1 - (ss_res_4 / ss_tot_4)

print(r_cuadrado_1) #Coeficiente de determinación Iq+
print(r_cuadrado_2) #Coeficiente de determinación Iq-
print(r_cuadrado_3) #Coeficiente de determinación Ip+
print(r_cuadrado_4) #Coeficiente de determinación Ip-
```

```
0.9999984857241058
0.9999874886377645
0.9999988874123812
0.9999997272801928
```

Media, mediana y desviación estándar de los parámetros de salida del código de red seleccionado

```
#Media
media_1O = np.mean(Vobtenidos[:,4])
media_1R = np.mean(Vreales[:,4])

media_2O = np.mean(Vobtenidos[:,5])
media_2R = np.mean(Vreales[:,5])

media_3O = np.mean(Vobtenidos[:,6])
media_3R = np.mean(Vreales[:,6])

media_4O = np.mean(Vobtenidos[:,7])
media_4R = np.mean(Vreales[:,7])

print(media_1O) #media Iq+ obtenidos
print(media_1R) #media Iq+ reales
print(media_2O) #media Iq- obtenidos
print(media_2R) #media Iq- reales
print(media_3O) #media Ip+ obtenidos
print(media_3R) #media Ip+ reales
print(media_4O) #media Ip- obtenidos
print(media_4R) #media Ip- reales
```

```
8.025975613927216
8.025955015646758
0.5489265558890706
0.5490908876868638
0.9683919742002937
0.9684322817413181
0.4401126062452476
0.44018880370143687
```

```
#Mediana
median_1O = np.median(Vobtenidos[:,4])
median_1R = np.median(Vreales[:,4])

median_2O = np.median(Vobtenidos[:,5])
median_2R = np.median(Vreales[:,5])

median_3O = np.median(Vobtenidos[:,6])
median_3R = np.median(Vreales[:,6])

median_4O = np.median(Vobtenidos[:,7])
```

```
median_4R = np.median(Vreales[:,7])

print(median_1O) #mediana Iq+ obtenidos
print(median_1R) #mediana Iq+ reales
print(median_2O) #mediana Iq- obtenidos
print(median_2R) #mediana Iq- reales
print(median_3O) #mediana Ip+ obtenidos
print(median_3R) #mediana Ip+ reales
print(median_4O) #mediana Ip- obtenidos
print(median_4R) #mediana Ip- reales
```

```
10.000032424926758
10.0
-1.3050610505160876e-05
0.0
-3.970749730797252e-06
-1.1102230246251563e-16
-6.426653726521181e-06
0.0
```

```
#Desviación estándar
std_1O = np.std(Vobtenidos[:,4])
std_1R = np.std(Vreales[:,4])

std_2O = np.std(Vobtenidos[:,5])
std_2R = np.std(Vreales[:,5])

std_3O = np.std(Vobtenidos[:,6])
std_3R = np.std(Vreales[:,6])

std_4O = np.std(Vobtenidos[:,7])
std_4R = np.std(Vreales[:,7])

print(std_1O) #desviación estándar Iq+ obtenidos
print(std_1R) #desviación estándar Iq+ reales
print(std_2O) #desviación estándar Iq- obtenidos
print(std_2R) #desviación estándar Iq- reales
print(std_3O) #desviación estándar Ip+ obtenidos
print(std_3R) #desviación estándar Ip+ reales
print(std_4O) #desviación estándar Ip- obtenidos
print(std_4R) #desviación estándar Ip- reales
```

```
3.4484133242237327
3.448445819362191
1.3290053840774954
1.3294055623947172
1.7839592427434747
1.7838440066314092
1.1040044303030032
1.1041286415720557
```

OBJETIVO 3

```
Ip_pos_real =  Vreales[:,6]
Ip_pos_real_r = [round(valor, 1) for valor in Ip_pos_real]



Ip_pos_obt =  Vobtenidos[:,6]
Ip_pos_obt_r = [round(valor, 1) for valor in Ip_pos_obt]


coincidencias = 0
for i in range(len(Ip_pos_obt)):
    if Ip_pos_obt_r[i] == Ip_pos_real_r[i]:
        coincidencias += 1

porcentaje = coincidencias*100/len(Ip_pos_obt)
print("Los vectores coinciden en", coincidencias, "valores. Correspondientes al", porcentaje, "% de los datos")
```

```
Los vectores coinciden en 90733 valores. Correspondientes al 97.92775193463784 % de los datos
```

```
#Valor máximo de Ip+ real y obtenida
print(np.max(Ip_pos_real_r))
print(np.max(Ip_pos_obt_r))
```

```
7.1
7.1
```

APENDICE D

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Código para la obtención del primer Objetivo de control

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

```
import pandas as pd                                  # Librería para leer bases de datos
import numpy as np                                   # Librería para realizar operaciones, matrices, etc.
import matplotlib.pyplot as plt                      # Librería para graficar
```

```
Entrada=pd.read_csv('BD_España_Entradas_Testeo')           # Leer base de datos de Entradas
SalidaR=pd.read_csv('BD_España_Resultados_Reales')         # Leer base de datos de Salidas reales
SalidaO=pd.read_csv('BD_España_Resultados_Obtenidos')      # Leer base de datos de Salidas Obtenidas por la NN
```

```
# Se ejecuta la BD de entrada con los primeros 8 valores para identificar las etiquetas y la naturaleza de las variables
# Se adicionaron otras variables para el cálculo de Imax y el valor de V+ y V- como referencia.

Entrada[0:8]
```

	V+_pu	V-_pu	V+	V-	Desfase	P referencia	cos_min_phi_grados	Ip_pos_m
0	0.5848	0.76	90.973530	118.228254	-146.62	800.0	-47.844266	0.0000
1	0.0676	0.60	10.516092	93.338095	20.30	500.0	-44.083347	0.0000
2	0.0964	0.60	14.996321	93.338095	-14.21	500.0	-39.951787	0.0000
3	0.9460	0.64	147.163063	99.560635	-12.18	900.0	-38.471936	6.5008
4	0.2950	0.48	45.891230	74.670476	-2.03	400.0	-30.387575	0.0000
5	0.1540	0.45	23.956778	70.003571	148.06	300.0	-48.621345	0.0000
6	0.0676	0.09	10.516092	14.000714	-10.15	500.0	-36.943797	0.0000

```
# Se asigna a las variables x1, x2, y1 y y2 las coordenadas del código de red extraidas en la BD en entradas

x1_vec = (SalidaO[SalidaO.columns[0]].copy().values)              # Coordenada x1 del código de red
x2_vec = (SalidaO[SalidaO.columns[1]].copy().values)              # Coordenada x1 del código de red
y1_vec = (SalidaO[SalidaO.columns[2]].copy().values)              # Coordenada y1 del código de red
y2_vec = (SalidaO[SalidaO.columns[3]].copy().values)              # Coordenada y2 del código de red

# Cálculo de la pendiente de la recta que rige el límite entre x1 y x2
# Se utiliza la ecuación de la pendiente m=(Y2-Y1)/(X2-X1)

mm=(y2_vec[0]-y1_vec[0])/(x2_vec[0]-x1_vec[0])                    # Cálculo de la pendiente para la recta
```

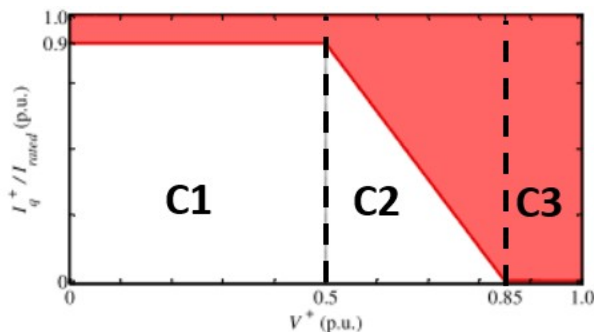
```
# Se ejecuta la BD de salida de la NN con los primeros 8 valores para identificar las etiquetas y la naturaleza de las variables
# Al crear la NN se mezclan todas las BD de códigos de red, separando testeo, validación y comprobación por separado,
# por lo tanto, guardamos las coordenadas asociadas a cada salida

SalidaO[0:8]
```

	x1	x2	y1	y2	Iq+	Iq-	Ip+	Ip-
0	0.5	0.85	0.9	0	10.000011	-0.000002	-0.000007	-0.000006
1	0.5	0.85	0.9	0	10.000011	-0.000002	-0.000007	-0.000006
2	0.5	0.85	0.9	0	10.000011	-0.000002	-0.000007	-0.000006
3	0.5	0.85	0.9	0	-0.000096	0.000494	6.503558	4.398599
4	0.5	0.85	0.9	0	10.000011	-0.000002	-0.000007	-0.000006
5	0.5	0.85	0.9	0	10.000011	-0.000002	-0.000007	-0.000006
6	0.5	0.85	0.9	0	10.000011	-0.000002	-0.000007	-0.000006
7	0.5	0.85	0.9	0	10.000011	-0.000002	-0.000007	-0.000006

```
# Ejecutamos la BD de salida que nos da el diagrama de control para comparar las variables

SalidaR[0:8]
```

	x1	x2	y1	y2	Iq+	Iq-	Ip+	Ip-
0	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
1	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
2	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
3	0.5	0.85	0.9	0	0.0	0.0	6.500872e+00	4.398053
4	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
5	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
6	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
7	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000

```
# Se asignan a su respectivo vector, los valores de V+ (pu), Iq+ real y Iq+ NN

V_posE=(Entrada[Entrada.columns[0]].copy().values)                      #V+
Iq_posR=np.round((SalidaR[SalidaR.columns[4]].copy().values)*0.1,2)     #Iq+R, Se redondean a 2 cifras para truncar los datos y disminuir rui
Iq_posO=np.round((SalidaO[SalidaO.columns[4]].copy().values)*0.1,2)     #Iq+O, Se redondean a 2 cifras para truncar los datos y disminuir rui
# También, se multiplica por 0.1, es decir 1/Irated

n=0                               # Contador que acumula los vectores que no cumplen
Balance=np.zeros(len(Iq_posR))    # Cantidad de datos que cumplen o no, almacenados en 1 y 0
NCIq_posO=np.zeros(len(Iq_posR))  # Valor de Iq+O que no cumple
NCIq_posR=np.zeros(len(Iq_posR))  # Iq+R en la misma posición de Iq+O (Para comparar)
NCV_posE=np.zeros(len(Iq_posR))   # El valor del V+ en donde no cumple
posicion=np.zeros(len(Iq_posR))   # Es la posición en donde se detecta que no cumple en la BD
Diferencia=np.zeros(len(Iq_posR)) # Es la Diferencia entre el valor límite del código de red y el valor de Iq+NN que no cumple
```



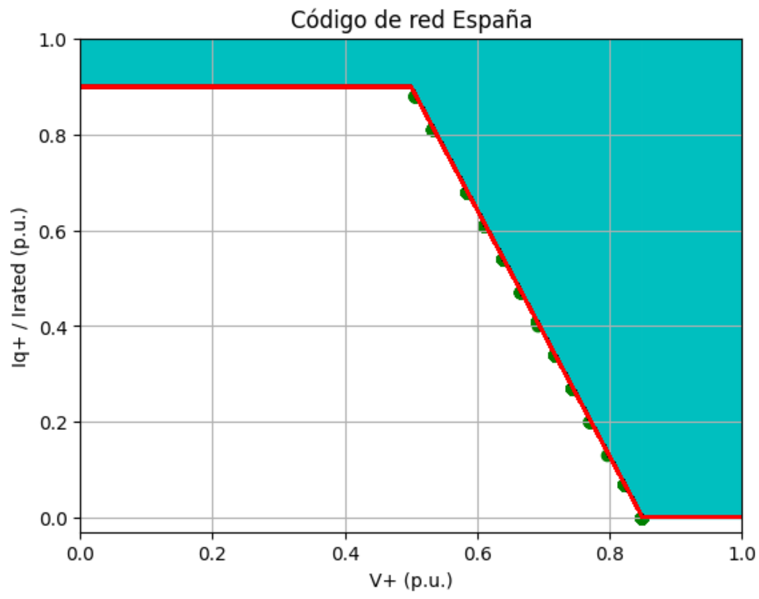

```
for i in range(0,len(Iq_posR)):   # ciclo for, de 0 a 92652 (Siempre en range va de 0 a n-1)

  if V_posE[i]<=x1_vec[0] and Iq_posO[i]>=y1_vec[0]:    # Condicional para saber si está en C1 Y si la Iq+NN >= al límite del código de red

    Balance[i]=True                                     # Guarda el valor de 1

  elif V_posE[i]>x1_vec[0] and V_posE[i]<=x2_vec[0] and Iq_posO[i]>=(mm*(V_posE[i]-x1_vec[0])+y1_vec[0]): # Cond. en C2 y si Iq+NN >= a la re

    Balance[i]=True                                     # Guarda el valor de 1

  elif V_posE[i]>x2_vec[0] and Iq_posO[i]>=y2_vec[0]:   # Cond. en C3 y si Iq+NN >= a y2

    Balance[i]=True                                     # Guarda el valor de 1

  else: # Recordar que en este punto no se cumplieron las condiciones al mismo tiempo, de C1, C2 o C3 y que la corriente Iq+NN >= al límite
        # A continuación, se recolecta y almacena los valores de Iq+ que no cumplieron y se tiene en cuenta el valor de la base de datos
        # para así compararlo, también, se almacena el valor de V+ en donde lo cumplió y la posición en donde se encontró el fallo en el
        # vector Iq_posO.

    Balance[i]=False                # Recolecta las corrientes que no cumplen con un valor de 0
    NCIq_posO[n]=Iq_posO[i]         # i es la posición de la corriente cuando no cumple y NCIq_posO almacena las corrientes que no cumplen
    NCIq_posR[n]=Iq_posR[i]         # Almacenar la corriente real en los puntos en donde la corriente Iq+O no cumple
    NCV_posE[n]=V_posE[i]           # Almacenar el voltaje en los puntos en donde la corriente Iq+O no cumple
```

```
    posicion[n]=i                   # Posición en donde se haya la condición.

        # Lo que se escribió anteriormente se encuentra dentro del else ^^^^ (Claridad para guiarse en la forma de usar cond. en Python)

        # En este punto se evaluará en qué cuadrantes se encuentran las fallas
        # ya que, si entró a este punto, se comprobó que no cumplió en al menos 1 de las 2 condiciones
        # por lo tanto, al condicionar nuevamente si se encuentran en C1, C2 o C3, encontraremos la ubicación de la falla
    #-----------------------------------------------------------------
    if V_posE[i]<=x1_vec[0]:                                                  # Si cumple, la falla está en C1

      Diferencia[n]= y1_vec[0]-Iq_posO[i]                                     # Se calcula la diferencia entre Iq+NN y el límite

    elif V_posE[i]>x1_vec[0] and V_posE[i]<=x2_vec[0]:                        # Si cumple, la falla está en C2 (la recta)

      Diferencia[n]=(mm*(V_posE[i]-x1_vec[0])+y1_vec[0])-Iq_posO[i]           # Se calcula la diferencia entre Iq+NN y el punto de la recta

    else:                                                                     # En este punto se da por hecho que si no pertence a C1 o C2, debe

      Diferencia[n]= y2_vec[0]-Iq_posO[i]                                     # Calcula la diferencia entre Iq+NN y el límite

    #-----------------------------------------------------------------

    if Diferencia[n]==max(Diferencia):                                        # Busca el valor con la mayor diferencia y lo almacena
      VlimCDR=Iq_posR[i]                                                      # Guarda el valor real en donde se encontró mayor diferencia
      VlimCDO=Iq_posO[i]                                                      # Guarda el valor de Iq+NN en donde se encontró mayor diferencia

    #-----------------------------------------------------------------
    n=n+1                                                                     # Acumulador de la condicion de no cumple NC.

    # Lo que se escribió anteriormente se encuentra dentro del primer else ^^^^


# Se crean nuevas varias para tomar solo la cantidad de datos que no cumplieron, debido a que en los anteriores vectores,
# al no saber la cantidad de valores en falla, se creó vectores de ceros con la capacidad total

NCIqpo=np.zeros(n)            # Iq+NN que no cumple
NCIqpr=np.zeros(n)            # Iq+ de BD en la misma posición para comparar
NCVpe=np.zeros(n)             # V+ en donde ocurrió la falla
p=np.zeros(n)                 # Posición
d=np.zeros(n)                 # Diferencia

for l in range(0,n):          # Se trasladan los valores hacia los nuevos vectores con el tamaño adecuado
  NCIqpo[l]=NCIq_posO[l]      # Iq+NN
  NCIqpr[l]=NCIq_posR[l]      # Iq+ Real
  NCVpe[l]=NCV_posE[l]        # V+
  p[l]=posicion[l]            # Posición
  d[l]=Diferencia[l]          # Diferencia


print('A continuación se muestran algunos datos para visualizar y analizar los resultados')
print('')
print('------------------------------------------------------------')
print('Valor límite Real ,',VlimCDR)          # Valor de Iq+ real en donde se encontró mayor diferencia
print('Valor límite Obtenido ,',VlimCDO)      # Valor de Iq+ NN en donde se encontró mayor diferencia
                                              # Diferencia entre el límite y el valor de Iq+ generado por NN
print(' ')
print('------------------------------------------------------------')
                                              # Vectores con celdas vacías
print('NCIqpo   -->',NCIq_posO)               # Iq+ NN
print('NCIqpr   -->',NCIq_posR)               # Iq+ Real
print('NCVpe    -->',NCV_posE)                # V+
print('Posicion -->',posicion)                # Posición

print(' ')
print('------------------------------------------------------------')
                                              # Vectores con tamaño corregido
print('NCIqpo   -->',NCIqpo)                  # Iq+ NN
print('NCIqpr   -->',NCIqpr)                  # Iq+ Real
print('NCVpe    -->',NCVpe)                   # V+
print('Posicion -->',p)                       # Posición
print(' ')
print('------------------------------------------------------------')
print('Iq_posR  -->',Iq_posR[92647])          # Se evalua en alguna posición en donde ocurrió la falla
print('Iq_posO  -->',Iq_posO[92647])          # para confirmar que Iq+NN no cumple y el valor real si
print('V_posE   -->',V_posE[92647])
print(' ')
print('------------------------------------------------------------')
```

```
# A continuación se hace regla de 3 con ayuda de Balance para saber cuántas fallas hubo en porcentaje

print('Cantidad que NO cumple: ',np.sum(Balance==0), ' de un total de ', len(Balance), ', es decir, el ', np.round(np.sum(Balance==0)*100/len
print('')
print('El máximo valor que está más alejado del límite del código de Red es de ', np.round(max(d),5), 'pu')
print('')
```

```
A continuación se muestran algunos datos para visualizar y analizar los resultados

----------------------------------------------------------
Valor límite Real , 0.41
Valor límite Obtenido , 0.4

----------------------------------------------------------
NCIqpo   --> [0.41 0.61 0.27 ... 0.   0.   0.  ]
NCIqpr   --> [0.41 0.61 0.27 ... 0.   0.   0.  ]
NCVpe    --> [0.6904 0.6112 0.7432 ... 0.     0.     0.    ]
Posicion --> [ 99. 115. 162. ...   0.   0.   0.]

----------------------------------------------------------
NCIqpo   --> [0.41 0.61 0.27 ... 0.61 0.61 0.54]
NCIqpr   --> [0.41 0.61 0.27 ... 0.61 0.61 0.55]
NCVpe    --> [0.6904 0.6112 0.7432 ... 0.6112 0.6112 0.6376]
Posicion --> [   99.   115.   162. ... 92554. 92587. 92588.]

----------------------------------------------------------
Iq_posR  --> 0.9
Iq_posO  --> 0.9
V_posE   --> 0.4

----------------------------------------------------------
Cantidad que NO cumple:  2452  de un total de  92653 , es decir, el  2.646 %

El máximo valor que está más alejado del límite del código de Red es de  0.0104 pu
```

```
# En el siguiente código se grafica lo siguiente:
#   - Código de red
#   - Puntos que no cumplen el código
#   - El área del código en el cual Iq+/Irated puede tomar

#Gráfica de Código de Red

plt.title('Código de red España')                             # Título


p1=[0,x1_vec[0],x2_vec[0],1]                                  # Puntos eje X
p2=[y1_vec,y1_vec,y2_vec,y2_vec]                              # Puntos eje Y

t1=np.arange(start=0,stop=x1_vec[0]+0.01, step=0.01)          # Primer rango
t2=np.arange(start=x1_vec[0],stop=x2_vec[0]+0.01, step=0.01)  # Segundo rango
t3=np.arange(start=x2_vec[0],stop=1+0.01, step=0.01)          # Tercer rango

f=mm*(t2-x1_vec[0])+y1_vec[0]                                 # Cálculo de la recta

plt.ylim(-0.03,1)                                             # Límite Y
plt.xlim(0,1)                                                 # Límite X
plt.xlabel('V+ (p.u.)')
plt.ylabel('Iq+ / Irated (p.u.)')
plt.grid()

plt.plot(p1,p2,"-",color='r')                                 # Gráfica de recta límite
plt.scatter(NCVpe, NCIqpo,color='g')                          # Grafica los puntos que no cumplen de Iq+NN

plt.fill_between(t1, y1_vec[0], 1.3,color='c')                # Rellena primera zona
plt.fill_between(t2, f, 1.3,color='c')                        # Rellena segunda zona
plt.fill_between(t3, y2_vec[0], 1.3,color='c')                # Rellena tercera zona
plt.show()
```



Productos pagados de Colab - Cancela los contratos aquí

APENDICE E

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Código para la obtención del segundo Objetivo de control

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

```
import pandas as pd                                                 # Librería para leer bases de datos
import numpy as np                                                  # Librería para realizar operaciones, matrices, etc.
import matplotlib.pyplot as plt                                     # Librería para graficar
```

```
Entrada=pd.read_csv('BD_España_Entradas_Testeo')           # Leer base de datos de Entradas
SalidaR=pd.read_csv('BD_España_Resultados_Reales')         # Leer base de datos de Salidas reales
SalidaO=pd.read_csv('BD_España_Resultados_Obtenidos')      # Leer base de datos de Salidas Obtenidas por la NN
```

```
# Se ejecuta la BD de entrada con los primeros 4 valores para identificar las etiquetas y la
# naturaleza de las variables. Se adicionaron otras variables para el cálculo de Imax, el valor
# real, Ip_pos_max como condicional y el valor de V+ y V- como referencia.

Entrada[0:4]
```

	V+_pu	V-_pu	V+	V-	Desfase	P referencia	cos_min_phi_grados	Ip
0	0.5848	0.76	90.973530	118.228254	-146.62	800.0	-47.844266	
1	0.0676	0.60	10.516092	93.338095	20.30	500.0	-44.083347	
2	0.0964	0.60	14.996321	93.338095	-14.21	500.0	-39.951787	

```
Ip_posO = np.round((SalidaO[SalidaO.columns[6]].copy().values),3)        # Ip+NN, redondeado con 3 valores
Iq_posO = np.round((SalidaO[SalidaO.columns[4]].copy().values),3)        # Iq+NN, redondeado con 3 valores
Vpos_puO = (Entrada[Entrada.columns[0]].copy().values)                   # V+ (pu)
Vneg_puO = (Entrada[Entrada.columns[1]].copy().values)                   # V- (pu)
Ip_posR = (SalidaR[SalidaR.columns[6]].copy().values)                    # Ip+ de BD
Iq_posR = (SalidaR[SalidaR.columns[4]].copy().values)                    # Iq+ de BD
CosMinPhi = (Entrada[Entrada.columns[6]].copy().values)                  # CosMinPhi de BD
Ip_pos_max=(Entrada[Entrada.columns[7]].copy().values)                   # Ip+_max de BD
Imax=np.zeros(len(Ip_posO))                                              # Imax (Vector de ceros)
Irated=10                                                                # Irated
```

$$I_{\max} = \sqrt{\frac{(V^+)^2 - 2V^+V^-x + (V^-)^2}{(V^+)^2}\left((I_p^+)^2 + (I_q^+)^2\right)}. \quad (21)$$

```
SalidaR[0:4]
```

	x1	x2	y1	y2	Iq+	Iq-	Ip+	Ip-
0	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
1	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
2	0.5	0.85	0.9	0	10.0	0.0	-1.110223e-16	0.000000
3	0.5	0.85	0.9	0	0.0	0.0	6.500872e+00	4.398053

```
SalidaO[0:4]
```

```
# A continuación se agregan las condiciones que ejerce el diagrama de control
# con su respectiva modificación sobre el valor de Imax

for i in range (0,len(Ip_pos_max)):     # Ciclo for, de 0 a 92652 (Siempre en range va de 0 a n-1)
  if Vpos_puO[i] < 0.85:                # Condición que abarca Caso 3, 4, 5 y 6
    Imax[i]=Irated                      # Condición especificada por el diagrama de control
    if Ip_pos_max[i] == 0:              # Condición que abarca Caso 5 y 6
     Imax[i]=np.sqrt(((Vpos_puO[i]**2-2*Vpos_puO[i]*Vneg_puO[i]*CosMinPhi[i]+Vneg_puO[i]**2)/(Vpos_puO[i]**2))*(Ip_posO[i]**2+Iq_posO[i]**2))
     if Imax[i]>Irated:                 # Condición que abarca Caso 6
       Imax[i]=Irated                   # Si cumple, se restringe la corriente a Irated (Según el diagrama de control)
  else:                                 # Si no cumple, se va a Caso 1 y 2
    Imax[i]=np.sqrt(((Vpos_puO[i]**2-2*Vpos_puO[i]*Vneg_puO[i]*CosMinPhi[i]+Vneg_puO[i]**2)/(Vpos_puO[i]**2))*(Ip_posO[i]**2+Iq_posO[i]**2))
    if Imax[i]>Irated:                  # Restricción propuesta por el paper (Se restringe el valor de Imax a Irated para evitar pedir más co
     Imax[i]=Irated

# Se tomó la decisión que Imax sea el valor calculado y si es mayor a Irated, tome el valor
# de Irated, pues en las fórmulas, Irated se fija para calcular Ip+max cuando la corriente Imax sobrepasa el límite
# adicional, en el paper de Garnica, este quiere asegurar el Objetivo 2, que Imax<=Irated, por lo tanto
# cae bien dicha reestricción



print('Resultados ----------------------------------------')
print('')

print('Ip+NN -->',np.round(Ip_posO[0:5],3))
print('Iq+NN -->',np.round(Iq_posO[0:5],3))
print('')
print('----------------------------------------------------')
print('')
print('V+    -->',np.round(Vpos_puO[0:5],3))
print('V-    -->',np.round(Vneg_puO[0:5],3))
print('')
print('----------------------------------------------------')
print('')
print('Cos   -->',np.round(CosMinPhi[0:5],3))     # Variable  Cos_Min_Phi utilizada en la fórmula de Imax
print('Imax  -->',np.round(Imax[0:10],3))         # Se muestra los 10 valores de Imax para observar comportamiento
```

```
Resultados ----------------------------------------

Ip+NN --> [-0.    -0.    -0.     6.504 -0.   ]
Iq+NN --> [10. 10. 10. -0. 10.]

---------------------------------------------------

V+    --> [0.585 0.068 0.096 0.946 0.295]
V-    --> [0.76 0.6  0.6  0.64 0.48]

---------------------------------------------------

Cos   --> [-47.844 -44.083 -39.952 -38.472 -30.388]
Imax  --> [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
```

```
# A continuación se contarán cuántos valores no cumplieron

Balance=np.zeros(len(Ip_posO))            # Vector de datos que cumplen o no, almacenados en 1 y 0
for i in range(0,len(Ip_posO)):
  if Imax[i]<=Irated:                     # Imax <= 10, si lo es, cuenta como  valor válido
    Balance[i]=True
  else:
    Balance[i]=False                      # Imax > 10, cuenta como valor de falla o que no cumple el obj


print('Métricas porcentual de fallas')

print('')
print('Imax      --->',np.round(Imax[0:10],3))
print('')
print('Imax > 10 --->  ',np.sum(Imax>10), 'de ',len(Imax))
print('')
print('Cantidad que no cumple:',np.sum(Balance==0), 'de un total de',len(Ip_posO), ', representando un error del', np.round(np.sum(Balance==0
print('')
```

```
Métricas porcentual de fallas

Imax      ---> [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
```

```
Imax > 10 --->   0 de  92653

Cantidad que no cumple: 0 de un total de 92653 , representando un error del 0.0 %
```

Productos pagados de Colab - Cancela los contratos aquí

APENDICE F

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

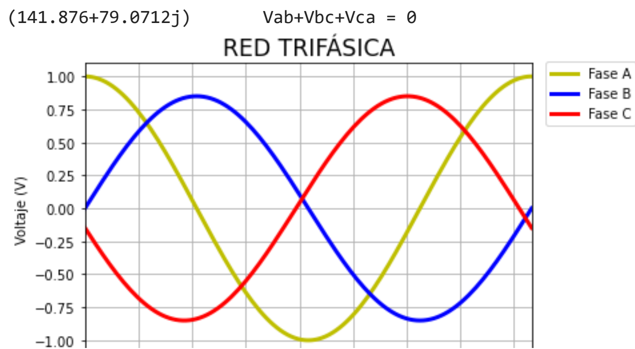
Componentes de secuencia de tensión y desfase de la red trifásica

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

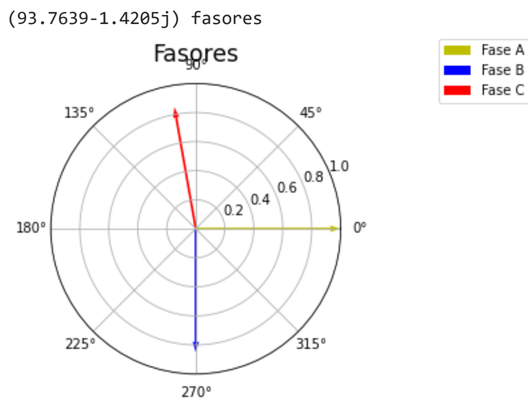
```
# Librerías
import numpy as np
import pylab as pl
import cmath
import matplotlib.pyplot as plt
import pandas as pd
from numpy.linalg import inv
from numpy.polynomial.polynomial import Polynomial

#Señales

f=60        #Frecuencia
T=1/f       #Periodo
t=np.arange(start=0, stop=(T/128)+T, step=T/128)    #Rango de tiempo
v_nom=110   #Tensión nominal

#Magnitudes

k_va=1
k_vb=0.85
k_vc=0.85

#agg Vab Vbc Vca con grafica y corrobar que la suma de 0

#Ángulos

ang_va=0
ang_vb=-90
ang_vc=100

#Senoidales

va=np.round(k_va*v_nom*np.cos(2*np.pi*f*t+np.deg2rad(ang_va)),4)
vb=np.round(k_vb*v_nom*np.cos(2*np.pi*f*t+np.deg2rad(ang_vb)),4)
vc=np.round(k_vc*v_nom*np.cos(2*np.pi*f*t+np.deg2rad(ang_vc)),4)

Vab=np.round(np.sqrt(3)*k_va*v_nom*np.e**(np.deg2rad(ang_va+30)*1j),10)
Vbc=np.round(np.sqrt(3)*k_vb*v_nom*np.e**(np.deg2rad(ang_vb+30)*1j),4)
Vca=np.round(np.sqrt(3)*k_vc*v_nom*np.e**(np.deg2rad(ang_vc+30)*1j),4)

z=np.round(Vab+Vbc+Vca,4)

print(z , '      Vab+Vbc+Vca = 0')

#Gráficas

fig, ax=plt.subplots()
ax.plot(t,va/110,label='Fase A',lw='3',color='y')
ax.plot(t,vb/110,label='Fase B',lw='3',color='b')
ax.plot(t,vc/110,label='Fase C',lw='3',color='r')

#Características de la gráfica

ax.legend(loc='upper right')
ax.axis([0,T,-1.1,1.1])
ax.set_title('RED TRIFÁSICA', fontsize='xx-large')
ax.set_xlabel('Tiempo (t)')
ax.set_ylabel('Voltaje (V)')
ax.legend(bbox_to_anchor=(1,1.05), borderaxespad=1)

plt.grid()
plt.show()
```

```
(141.876+79.0712j)       Vab+Vbc+Vca = 0
```



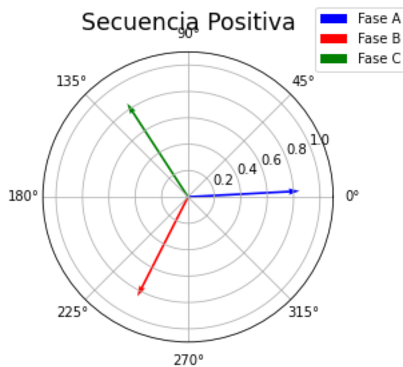

```
#Fasores

a=np.e**((2*(np.pi/3))*1j)    #Desfase de 120°

#Conversión a dominio fasorial

vaa=k_va*v_nom*np.e**((np.deg2rad(ang_va))*1j)
vbb=k_vb*v_nom*np.e**((np.deg2rad(ang_vb))*1j)
vcc=k_vc*v_nom*np.e**((np.deg2rad(ang_vc))*1j)

print(np.round(vaa+vbb+vcc,4),'fasores')

fas=[vaa,vbb,vcc]

fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,cmath.phase(fas[0]),np.abs(fas[0])/110, color=['y'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,orig
ax.quiver(0,0,cmath.phase(fas[1]),np.abs(fas[1])/110,color=['b'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,cmath.phase(fas[2]),np.abs(fas[2])/110,color=['r'],label='Fase C',angles='xy', scale_units='xy', scale=1)
# ax.quiver(0,0,cmath.phase(Vab),np.abs(Vab)/110,color=['k'],label='Vab',angles='xy', scale_units='xy', scale=1)
# ax.quiver(0,0,cmath.phase(Vbc),np.abs(Vbc)/110,color=['k'],label='Vbc',angles='xy', scale_units='xy', scale=1)
# ax.quiver(0,0,cmath.phase(Vca),np.abs(Vca)/110,color=['k'],label='Vca',angles='xy', scale_units='xy', scale=1)
# ax.quiver(0,0,cmath.phase(z),np.abs(z)/110,color=['g'],label='Vab+Vbc+Vc',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,1])
ax.legend(bbox_to_anchor=(1.7, 1.2), borderaxespad=1)

ax.set_title('Fasores',fontsize='xx-large')

plt.show()
```

```
(93.7639-1.4205j) fasores
```

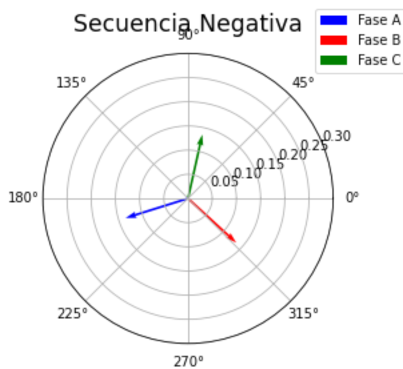



```
#Matrices
A=[[1, 1, 1], [a**2, a, 1], [a, a**2, 1]]

sec=np.round(np.matmul(inv(A),fas),6) #Despeje de la ecuación vaa=A*vaaa



#Secuencia positiva

fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,cmath.phase(sec[0]),np.abs(sec[0])/110, color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,orig
ax.quiver(0,0,cmath.phase(sec[0])-np.deg2rad(120),np.abs(sec[0])/110,color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,cmath.phase(sec[0])+np.deg2rad(120),np.abs(sec[0])/110,color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,1.1])
```

```
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Positiva',fontsize='xx-large')

plt.show()
```

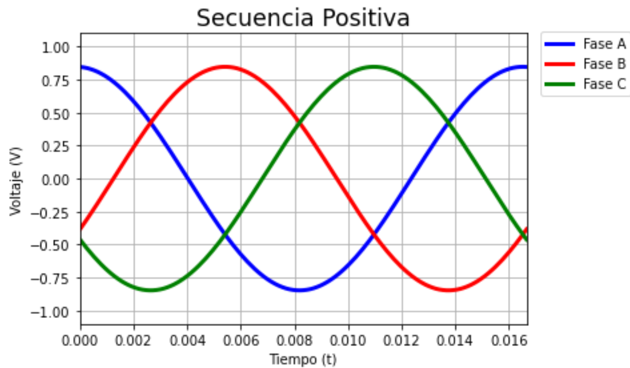



```
#Secuencia Negativa

fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,cmath.phase(sec[1]),np.abs(sec[1])/110, color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,orig
ax.quiver(0,0,cmath.phase(sec[1])+np.deg2rad(120),np.abs(sec[1])/110,color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,cmath.phase(sec[1])-np.deg2rad(120),np.abs(sec[1])/110,color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,0.3])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Negativa',fontsize='xx-large')

plt.show()
```

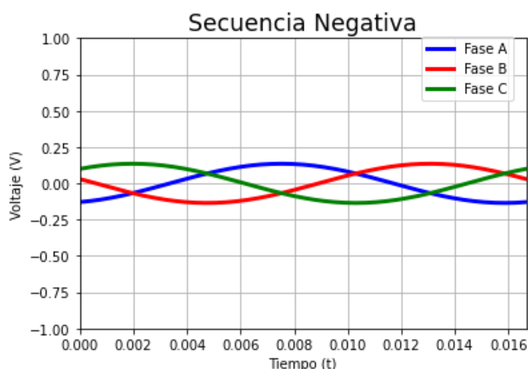



```
#Secuencia positiva senoidal

#Gráfica de las fases en el tiempo

mp=abs(sec[0])                          #Extracción de la magnitud
angap=cmath.phase(sec[0])                #Extracción de la fase
angbp=cmath.phase(sec[0])-np.deg2rad(120)
angcp=cmath.phase(sec[0])+np.deg2rad(120)

#Gráficas

fig, ax=plt.subplots()
ax.plot(t,mp/110*np.cos(2*np.pi*f*t+angap),label='Fase A',lw='3',color='b')
ax.plot(t,mp/110*np.cos(2*np.pi*f*t+angbp),label='Fase B',lw='3',color='r')
ax.plot(t,mp/110*np.cos(2*np.pi*f*t+angcp),label='Fase C',lw='3',color='g')

#Características de la gráfica

ax.legend(loc='upper right')
ax.axis([0,T,-1.1,1.1])
ax.set_title('Secuencia Positiva', fontsize='xx-large')
```

```
ax.set_xlabel('Tiempo (t)')
ax.set_ylabel('Voltaje (V)')
ax.legend(bbox_to_anchor=(1,1.05), borderaxespad=1)

plt.grid()
plt.show()
```

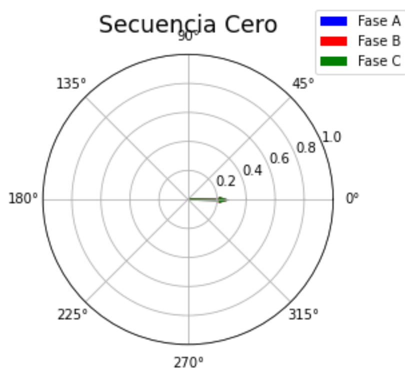



```
#Secuencia negativa senoidal

#Gráfica de las fases en el tiempo

mn=np.abs(sec[1])                             #Extracción de la magnitud
angan=cmath.phase(sec[1])                  #Extracción de la fase
angbn=cmath.phase(sec[1])-np.deg2rad(120)
angcn=cmath.phase(sec[1])+np.deg2rad(120)

#Gráficas

fig, ax=plt.subplots()
ax.plot(t,mn/110*np.cos(2*np.pi*f*t+angan),label='Fase A',lw='3',color='b')
ax.plot(t,mn/110*np.cos(2*np.pi*f*t+angbn),label='Fase B',lw='3',color='r')
ax.plot(t,mn/110*np.cos(2*np.pi*f*t+angcn),label='Fase C',lw='3',color='g')

#Características de la gráfica

ax.legend(loc='upper right')
ax.axis([0,T,-1,1])
ax.set_title('Secuencia Negativa', fontsize='xx-large')
ax.set_xlabel('Tiempo (t)')
ax.set_ylabel('Voltaje (V)')
ax.legend(bbox_to_anchor=(1,1.05), borderaxespad=1)

plt.grid()
plt.show()
```

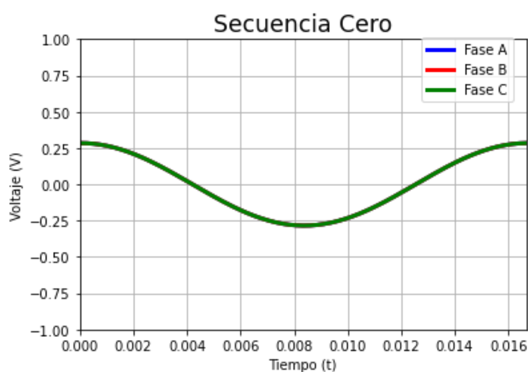



```
#Secuencia Cero

fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,cmath.phase(sec[2]),np.abs(sec[2]/110), color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,orig
ax.quiver(0,0,cmath.phase(sec[2]),np.abs(sec[2]/110),color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,cmath.phase(sec[2]),np.abs(sec[2]/110),color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,1])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)
```

```
ax.set_title('Secuencia Cero',fontsize='xx-large')

plt.show()
```




```
#Secuencia cero senoidal

#Gráfica de las fases en el tiempo

mo=np.abs(sec[2])                              #Extracción de la magnitud
ango=cmath.phase(sec[2])                   #Extracción de la fase

#Gráficas

fig, ax=plt.subplots()
ax.plot(t,mo/110*np.cos(2*np.pi*f*t+ango),label='Fase A',lw='3',color='b')
ax.plot(t,mo/110*np.cos(2*np.pi*f*t+ango),label='Fase B',lw='3',color='r')
ax.plot(t,mo/110*np.cos(2*np.pi*f*t+ango),label='Fase C',lw='3',color='g')

#Características de la gráfica

ax.legend(loc='upper right')
ax.axis([0,T,-1,1])
ax.set_title('Secuencia Cero', fontsize='xx-large')
ax.set_xlabel('Tiempo (t)')
ax.set_ylabel('Voltaje (V)')
ax.legend(bbox_to_anchor=(1,1.05), borderaxespad=1)

plt.grid()
plt.show()
```




```
if (np.round(mp/110,3)) < 0.02 or (np.round(mn/110,3)) < 0.02:


  desfase='Sin desfase'
else:

  desfase=np.round(np.rad2deg(cmath.phase(sec[0])-cmath.phase(sec[1])),1)


dataframe=pd.DataFrame({
                        'Fase A':
                            {0: k_va},
                        'Fase B':
```

```
                                {0: k_vb},
                            'Fase C':
                                {0: k_vc},
                            'Angulo A':
                                {0: ang_va},
                            'Angulo B':
                                {0: ang_vb},
                            'Angulo C':
                                {0: ang_vc},
                            'V+':
                                {0: np.round(mp/110,3)},
                            'V-':
                                {0: np.round(mn/110,3)},
                            'Desfase':
                                {0: desfase }
                            })

print(dataframe)
```

```
   Fase A  Fase B  Fase C  Angulo A  Angulo B  Angulo C     V+     V-  Desfase
0       1    0.85    0.85         0       -90       100  0.846  0.135    165.6
```

APENDICE G

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Rango de hundimientos tipo A

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

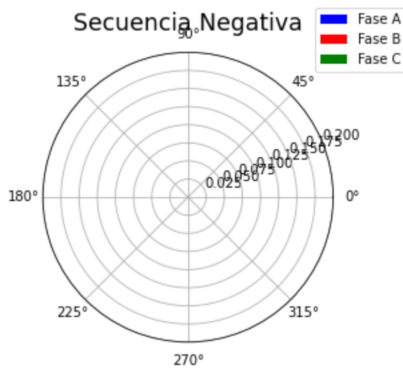
```
# Librerías

import numpy as np
import cmath
import·pandas·as·pd
from·numpy.linalg·import·inv
import·matplotlib.pyplot·as·plt


# Señales

f=60      #Frecuencia Hz
T=1/f     #Periodo
v_nom=1


#Ángulos

ang_va=0
ang_vb=-120
ang_vc=120

#Fasores

a=np.e**((2*(np.pi/3))*1j)    #Desfase de 120°

#Vectores

vsecp=np.zeros(8,float)
vsecn=np.zeros(8,float)
desfase=np.zeros(8,float)
desfase1=np.zeros(8,float)
desfase2=np.zeros(8,float)
factor=np.zeros(8)
l=0.05

for n in range(1,9):

  # Magnitudes
  k_va=l
  k_vb=k_va
  k_vc=k_va

  #Conversión a dominio fasorial

  vaa=k_va*v_nom*np.e**((np.deg2rad(ang_va))*1j)
  vbb=k_vb*v_nom*np.e**((np.deg2rad(ang_vb))*1j)
  vcc=k_vc*v_nom*np.e**((np.deg2rad(ang_vc))*1j)

  fas=[vaa,vbb,vcc]

  #Matrices
  A=[[1, 1, 1], [a**2, a, 1], [a, a**2, 1]]

  sec=np.round(np.matmul(inv(A),fas),6) #Despeje de la ecuación vaa=A*vaaa

  vsecp[n-1]=np.round(np.abs(sec[0]),3)
  vsecn[n-1]=np.round(np.abs(sec[1]),3)

  if (np.round(vsecp[n-1],3)) < 0.02 or (np.round(vsecn[n-1],3)) < 0.02:

    desfase[n-1]=0

  else:

    desfase[n-1]=np.round(np.rad2deg(cmath.phase(sec[0])+cmath.phase(sec[1])),3)
```

```
        desfase1[n-1]=np.round(np.rad2deg(cmath.phase(sec[0])),3)
        desfase2[n-1]=np.round(np.rad2deg(cmath.phase(sec[1])),3)

    factor[n-1]=np.round(l,3)

    l=l+0.02



d = {'Fase':factor,'V+':vsecp,'V-': vsecn,'Desfase': desfase}
pd.DataFrame(d)
```

	Fase	V+	V-	Desfase
0	0.05	0.05	0.0	0.0
1	0.07	0.07	0.0	0.0
2	0.09	0.09	0.0	0.0
3	0.11	0.11	0.0	0.0
4	0.13	0.13	0.0	0.0
5	0.15	0.15	0.0	0.0
6	0.17	0.17	0.0	0.0
7	0.19	0.19	0.0	0.0

```
x=6
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase1[4]),vsecp[4], color=['r'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[4])-np.deg2rad(120),vsecp[4],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[4])+np.deg2rad(120),vsecp[4],color=['r'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.quiver(0,0,np.deg2rad(desfase1[6]),vsecp[6], color=['b'],angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,angulo,magnitud)
ax.quiver(0,0,np.deg2rad(desfase1[6])-np.deg2rad(120),vsecp[6],color=['b'],angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[6])+np.deg2rad(120),vsecp[6],color=['b'],angles='xy', scale_units='xy', scale=1)

ax.axis([0,2*np.pi, 0,0.4])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Positiva',fontsize='xx-large')

plt.show()
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase2[x]),vsecn[x], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase2[x])+np.deg2rad(120),vsecn[x],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase2[x])-np.deg2rad(120),vsecn[x],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,0.2])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Negativa',fontsize='xx-large')

plt.show()
```



APENDICE G

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Rango de hundimientos tipo B

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

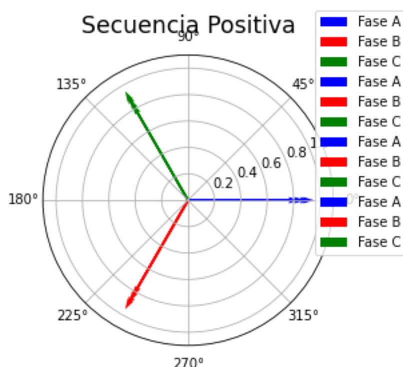
Director: Juan Manuel Rey López

```
# Librerías

import numpy as np
import cmath
import·pandas·as·pd
from·numpy.linalg·import·inv
import·matplotlib.pyplot·as·plt


#·Señales

f=60      #Frecuencia Hz
T=1/f     #Periodo
v_nom=1

k_vb=1
k_vc=1

#Ángulos

ang_va=0
ang_vb=-120
ang_vc=120

#Fasores

a=np.e**((2*(np.pi/3))*1j)    #Desfase de 120°

#Vectores

vsecp=np.zeros(10,float)
vsecn=np.zeros(10,float)
desfase=np.zeros(10,float)
desfase1=np.zeros(10,float)
desfase2=np.zeros(10,float)
factor=np.zeros(10)
l=0.4

for n in range(1,11):

  # Magnitudes
  k_va=l

  #Conversión a dominio fasorial

  vaa=k_va*v_nom*np.e**((np.deg2rad(ang_va))*1j)
  vbb=k_vb*v_nom*np.e**((np.deg2rad(ang_vb))*1j)
  vcc=k_vc*v_nom*np.e**((np.deg2rad(ang_vc))*1j)

  fas=[vaa,vbb,vcc]

  #Matrices
  A=[[1, 1, 1], [a**2, a, 1], [a, a**2, 1]]

  sec=np.round(np.matmul(inv(A),fas),6) #Despeje de la ecuación vaa=A*vaaa

  vsecp[n-1]=np.round(np.abs(sec[0]),3)
  vsecn[n-1]=np.round(np.abs(sec[1]),3)

  if (np.round(vsecp[n-1],3)) < 0.02 or (np.round(vsecn[n-1],3)) < 0.02:


    desfase[n-1]=0

  else:
```

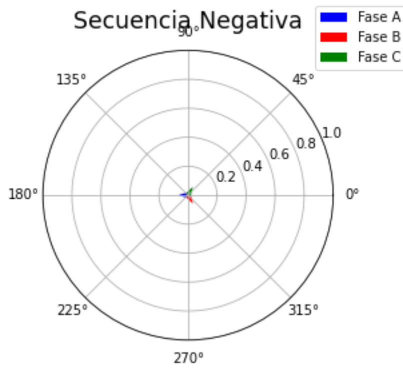
```
      desfase[n-1]=np.round(np.rad2deg(cmath.phase(sec[0])+cmath.phase(sec[1])),3)
      desfase1[n-1]=np.round(np.rad2deg(cmath.phase(sec[0])),3)
      desfase2[n-1]=np.round(np.rad2deg(cmath.phase(sec[1])),3)

    factor[n-1]=np.round(l,3)

    l=l+0.066

d = {'Fase':factor,'V+':vsecp,'V-': vsecn,'Desfase': desfase}
pd.DataFrame(d)
```

	Fase	V+	V-	Desfase
0	0.400	0.800	0.200	-180.0
1	0.466	0.822	0.178	-180.0
2	0.532	0.844	0.156	-180.0
3	0.598	0.866	0.134	-180.0
4	0.664	0.888	0.112	-180.0
5	0.730	0.910	0.090	-180.0
6	0.796	0.932	0.068	-180.0
7	0.862	0.954	0.046	-180.0
8	0.928	0.976	0.024	-180.0
9	0.994	0.998	0.002	0.0

```
x=6
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase1[2]),vsecp[2], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[2])-np.deg2rad(120),vsecp[2],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[2])+np.deg2rad(120),vsecp[2],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.quiver(0,0,np.deg2rad(desfase1[4]),vsecp[4], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[4])-np.deg2rad(120),vsecp[4],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[4])+np.deg2rad(120),vsecp[4],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.quiver(0,0,np.deg2rad(desfase1[6]),vsecp[6], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[6])-np.deg2rad(120),vsecp[6],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[6])+np.deg2rad(120),vsecp[6],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.quiver(0,0,np.deg2rad(desfase1[7]),vsecp[7], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[7])-np.deg2rad(120),vsecp[7],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[7])+np.deg2rad(120),vsecp[7],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.axis([0,2*np.pi, 0,1.1])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Positiva',fontsize='xx-large')

plt.show()
```



```
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase2[x]),vsecn[x], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase2[x])+np.deg2rad(120),vsecn[x],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase2[x])-np.deg2rad(120),vsecn[x],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,1])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Negativa',fontsize='xx-large')

plt.show()
```



APENDICE G

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Rango de hundimientos tipo C

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

```
# Librerías

import·numpy·as·np
import·cmath
import·pandas·as·pd
from·numpy.linalg·import·inv
import·matplotlib.pyplot·as·plt


# Señales

f=60      #Frecuencia Hz
T=1/f     #Periodo
v_nom=1

k_va=1

#Ángulos

ang_va=0

#Fasores

a=np.e**((2*(np.pi/3))*1j)    #Desfase de 120°

#Vectores

vsecp=np.zeros(64,float)
vsecn=np.zeros(64,float)
desfase=np.zeros(64,float)
desfase1=np.zeros(64,float)
desfase2=np.zeros(64,float)
factor=np.zeros(64)
recorridoB=np.zeros(64)
recorridoC=np.zeros(64)

h=0
angb=0

for n in range(1,5):        #Ángulo

  ang_vb=-120-angb
  angc=0
  for c in range(1,5):

    ang_vc=120+angc
    l=0.3

    for m in range(1,5):     #magnitud         #Va de 1 hasta 4

      # Magnitudes
      k_vb=l
      k_vc=k_vb

      #Conversión a dominio fasorial
      vaa=k_va*v_nom*np.e**((np.deg2rad(ang_va))*1j)
      vbb=k_vb*v_nom*np.e**((np.deg2rad(ang_vb))*1j)
      vcc=k_vc*v_nom*np.e**((np.deg2rad(ang_vc))*1j)

      fas=[vaa,vbb,vcc]

      #Matrices
      A=[[1, 1, 1], [a**2, a, 1], [a, a**2, 1]]

      sec=np.round(np.matmul(inv(A),fas),6) #Despeje de la ecuación vaa=A*vaaa
```

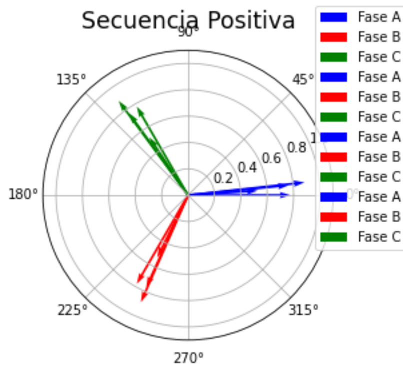
```
        vsecp[h]=np.round(np.abs(sec[0]),3)         #desde 0 hasta Maximo -1
        vsecn[h]=np.round(np.abs(sec[1]),3)

        if (np.round(vsecp[h],3)) < 0.02 or (np.round(vsecn[h],3)) < 0.02:

          desfase[h]=0

        else:

          desfase[h]=np.round(np.rad2deg(cmath.phase(sec[0])-cmath.phase(sec[1])),1)
          desfase1[h]=np.round(np.rad2deg(cmath.phase(sec[0])),3)
          desfase2[h]=np.round(np.rad2deg(cmath.phase(sec[1])),3)

        factor[h]=np.round(l,2)
        recorridoB[h]=-angb-120
        recorridoC[h]=angc+120
        h=h+1
        l=l+0.183

      angc=angc+20
    angb=angb+20

anga=np.ones(64)*ang_va
vmag=np.ones(64)*k_va

d = {'Fase A':vmag,'Fase B':factor,'Fase C':factor,'Ang A':(anga),'Ang B':(recorridoB),'Ang C':(recorridoC),'V+':vsecp,'V-': vsecn,'Desfase':
pd.DataFrame(d)
```

	Fase A	Fase B	Fase C	Ang A	Ang B	Ang C	V+	V-	Desfase
0	1.0	0.30	0.30	0.0	-120.0	120.0	0.533	0.233	0.0
1	1.0	0.48	0.48	0.0	-120.0	120.0	0.655	0.172	0.0
2	1.0	0.67	0.67	0.0	-120.0	120.0	0.777	0.111	0.0
3	1.0	0.85	0.85	0.0	-120.0	120.0	0.899	0.050	0.0
4	1.0	0.30	0.30	0.0	-120.0	140.0	0.528	0.266	6.3
...	...	...	...	...	...	...	...	...	...
59	1.0	0.85	0.85	0.0	-180.0	160.0	0.695	0.525	-1.5
60	1.0	0.30	0.30	0.0	-180.0	180.0	0.433	0.433	0.0
61	1.0	0.48	0.48	0.0	-180.0	180.0	0.494	0.494	0.0
62	1.0	0.67	0.67	0.0	-180.0	180.0	0.555	0.555	-0.0
63	1.0	0.85	0.85	0.0	-180.0	180.0	0.616	0.616	-0.0

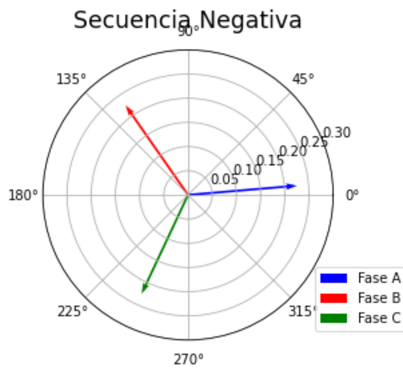
64 rows × 9 columns

```
x=17
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase1[2]),vsecp[2], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[2])-np.deg2rad(120),vsecp[2],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[2])+np.deg2rad(120),vsecp[2],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.quiver(0,0,np.deg2rad(desfase1[4]),vsecp[4], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[4])-np.deg2rad(120),vsecp[4],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[4])+np.deg2rad(120),vsecp[4],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.quiver(0,0,np.deg2rad(desfase1[6]),vsecp[6], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[6])-np.deg2rad(120),vsecp[6],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
```

```
ax.quiver(0,0,np.deg2rad(desfase1[6])+np.deg2rad(120),vsecp[6],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)

ax.quiver(0,0,np.deg2rad(desfase1[7]),vsecp[7], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[7])-np.deg2rad(120),vsecp[7],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[7])+np.deg2rad(120),vsecp[7],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,1.1])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Positiva',fontsize='xx-large')

plt.show()
```




```
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase2[x]),vsecn[x], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase2[x])+np.deg2rad(120),vsecn[x],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase2[x])-np.deg2rad(120),vsecn[x],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,0.3])
ax.legend(bbox_to_anchor=(1.3, 0.3), borderaxespad=1)

ax.set_title('Secuencia Negativa',fontsize='xx-large')

plt.show()
```



APENDICE G

Estrategia Basada en Redes Neuronales para el Control de Inversores Fotovoltaicos Ante Hundimientos de Tensión de la Red

Rango de hundimientos tipo D

Autores: Gabriela Alvarado Agudelo, Laura Sofía Mendoza Ramírez y Sebastián Felipe Rincón García

Director: Juan Manuel Rey López

```
# Librerías

import numpy as np
import cmath
import pandas as pd
from numpy.linalg import inv
import matplotlib.pyplot as plt


# Señales

f=60      #Frecuencia Hz
T=1/f     #Periodo
v_nom=1

#Ángulos

ang_va=0

#Fasores

a=np.e**((2*(np.pi/3))*1j)    #Desfase de 120°

#Vectores

vsecp=np.zeros(256,float)
vsecn=np.zeros(256,float)
desfase=np.zeros(256,float)
desfase1=np.zeros(256,float)
desfase2=np.zeros(256,float)
factor=np.zeros(256)
recorridoB=np.zeros(256)
recorridoC=np.zeros(256)
maga=np.zeros(256)

anga=np.ones(256)*ang_va


h=0
angb=0

for n in range(1,5):        #Ángulo B

  ang_vb=-120+angb
  angc=0
  for c in range(1,5):

    ang_vc=120-angc
    l=0.05

    for m in range(1,5):     #Ágnulo C

      # Magnitudes
      k_vb=l
      k_vc=k_vb
      p=0.05

      for ma in range(1,5):

        k_va=p

        #Conversión a dominio fasorial
        vaa=k_va*v_nom*np.e**((np.deg2rad(ang_va))*1j)
        vbb=k_vb*v_nom*np.e**((np.deg2rad(ang_vb))*1j)
        vcc=k_vc*v_nom*np.e**((np.deg2rad(ang_vc))*1j)
```

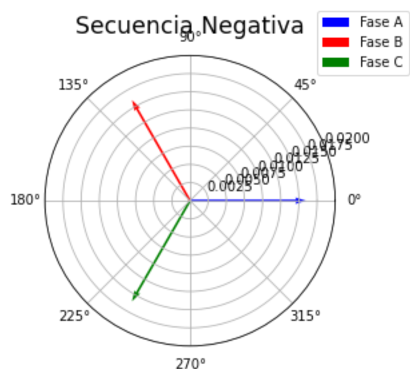
```
        fas=[vaa,vbb,vcc]

        #Matrices
        A=[[1, 1, 1], [a**2, a, 1], [a, a**2, 1]]

        sec=np.round(np.matmul(inv(A),fas),6) #Despeje de la ecuación vaa=A*vaaa

        vsecp[h]=np.round(np.abs(sec[0]),3)         #desde 0 hasta Maximo -1
        vsecn[h]=np.round(np.abs(sec[1]),3)

        if (np.round(vsecp[h],3)) < 0.02 or (np.round(vsecn[h],3)) < 0.02:

          desfase[h]=0

        else:

          desfase[h]=np.round(np.rad2deg(cmath.phase(sec[0])-cmath.phase(sec[1])),1)
          desfase1[h]=np.round(np.rad2deg(cmath.phase(sec[0])),3)
          desfase2[h]=np.round(np.rad2deg(cmath.phase(sec[1])),3)

        factor[h]=np.round(l,2)
        recorridoB[h]=-120+angb
        recorridoC[h]=120-angc
        maga[h]=np.round(p,3)
        h=h+1
        p=p+0.0466

      l=l+0.0466

    angc=angc+10
  angb=angb+10


d = {'Fase A':maga,'Fase B':factor,'Fase C':factor,'Ang A':(anga),'Ang B':(recorridoB),'Ang C':(recorridoC),'V+':vsecp,'V-': vsecn,'Desfase':
pd.DataFrame(d)

#pd.DataFrame(d).to_excel('datosD.xlsx')
```

	Fase A	Fase B	Fase C	Ang A	Ang B	Ang C	V+	V-	Desfase
0	0.050	0.05	0.05	0.0	-120.0	120.0	0.050	0.000	0.0
1	0.097	0.05	0.05	0.0	-120.0	120.0	0.066	0.016	0.0
2	0.143	0.05	0.05	0.0	-120.0	120.0	0.081	0.031	0.0
3	0.190	0.05	0.05	0.0	-120.0	120.0	0.097	0.047	0.0
4	0.050	0.10	0.10	0.0	-120.0	120.0	0.081	0.016	0.0
...	...	...	...	...	...	...	...	...	...
251	0.190	0.14	0.14	0.0	-90.0	90.0	0.146	0.019	0.0
252	0.050	0.19	0.19	0.0	-90.0	90.0	0.126	0.093	180.0
253	0.097	0.19	0.19	0.0	-90.0	90.0	0.142	0.077	180.0
254	0.143	0.19	0.19	0.0	-90.0	90.0	0.157	0.062	180.0
255	0.190	0.19	0.19	0.0	-90.0	90.0	0.173	0.046	180.0

256 rows × 9 columns

```
x=1
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase1[x]),vsecp[x], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase1[x])-np.deg2rad(120),vsecp[x],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase1[x])+np.deg2rad(120),vsecp[x],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,0.2])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Positiva',fontsize='xx-large')

plt.show()
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection="polar")
ax.quiver(0,0,np.deg2rad(desfase2[x]),vsecn[x], color=['b'],label='Fase A',angles='xy', scale_units='xy', scale=1) #quiver(origenx,origeny,an
ax.quiver(0,0,np.deg2rad(desfase2[x])+np.deg2rad(120),vsecn[x],color=['r'],label='Fase B',angles='xy', scale_units='xy', scale=1)
ax.quiver(0,0,np.deg2rad(desfase2[x])-np.deg2rad(120),vsecn[x],color=['g'],label='Fase C',angles='xy', scale_units='xy', scale=1)
ax.axis([0,2*np.pi, 0,0.02])
ax.legend(bbox_to_anchor=(1.3, 1.2), borderaxespad=1)

ax.set_title('Secuencia Negativa',fontsize='xx-large')

plt.show()
```



Productos pagados de Colab - Cancela los contratos aquí