```
# FORMULACIÓ PRÒPIA


# LLIBRERIES
import numpy as np
from mpmath import mp  # per tenir més decimals
mp.dps = 50
import pandas as pd
import matplotlib.pyplot as plt
from scipy.sparse import csc_matrix, coo_matrix
from scipy.sparse import lil_matrix, diags, hstack, vstack
from scipy.sparse.linalg import spsolve, factorized

np.set_printoptions(linewidth=2000, edgeitems=1000, suppress=True)
pd.set_option('display.max_rows', 5000)
pd.set_option('display.max_columns', 1000)
pd.set_option('display.width', 2000)
pd.set_option("display.precision", 5)
# FI LLIBRERIES


def conv(A, B, c, i, tipus):  # tres tipus de convolucions a executar
    if tipus == 1:
        suma = [np.conj(A[k, i]) * B[c - k, i] for k in range(1, c + 1)]
        return sum(suma)
    elif tipus == 2:
        suma = [A[k, i] * B[c - 1 - k, i] for k in range(1, c)]
        return sum(suma)
    elif tipus == 3:
        suma = [A[k, i] * np.conj(B[c - k, i]) for k in range(1, c)]
        return sum(suma)


# DEFIICIÓ D'OBJECTES INICIALS
df_top = pd.read_excel('IEEE30.xlsx', sheet_name='Topologia')  # dades de topologia
df_bus = pd.read_excel('IEEE30.xlsx', sheet_name='Busos')  # dades dels busos

n = df_bus.shape[0]  # nombre de busos, inclou l'oscil·lant
nl = df_top.shape[0]  # nombre de línies

A = np.zeros((n, nl), dtype=int)  # matriu d'incidència, formada per 1, -1 i 0
L = np.zeros((nl, nl), dtype=complex)  # matriu amb les branques
np.fill_diagonal(L, [1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) for i in range(nl)])
A[df_top.iloc[range(nl), 0], range(nl)] = 1
A[df_top.iloc[range(nl), 1], range(nl)] = -1

Yseries = np.dot(np.dot(A, L), np.transpose(A))  # matriu de les branques sèrie, es reduirà
Yseries_real = np.zeros((n, n), dtype=complex)
Yseries_real[:, :] = Yseries[:, :]  # també contindrà les admitàncies amb el bus oscil·lant

for i in range(nl):  # emplenar matriu quan hi ha trafo de relació variable
    tap = df_top.iloc[i, 5]
    ang_tap = df_top.iloc[i, 6]
    tap = abs(tap) * np.cos(ang_tap) + abs(tap) * np.sin(ang_tap) * 1j
    if tap != 1:
        Yseries[df_top.iloc[i, 0], df_top.iloc[i, 0]] += -1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) \
            + 1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) / (tap * np.conj(tap))
        Yseries[df_top.iloc[i, 1], df_top.iloc[i, 1]] += -1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) \
            + 1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j)
        Yseries[df_top.iloc[i, 0], df_top.iloc[i, 1]] += +1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) \
            + -1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) / np.conj(tap)
        Yseries[df_top.iloc[i, 1], df_top.iloc[i, 0]] += +1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) \
            + -1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) / tap

Yseries = csc_matrix(Yseries)  # a dispersa
Yseries_real = csc_matrix(Yseries_real)
```

```
vec_Pi = np.zeros(n, dtype=float)  # dades de potència activa
vec_Qi = np.zeros(n, dtype=float)  # dades de potència reactiva
vec_Vi = np.zeros(n, dtype=float)  # dades de tensió
vec_Wi = np.zeros(n, dtype=float)  # tensió al quadrat

pq = []  # índexs dels busos PQ
pv = []  # índexs dels busos PV
sl = []  # índexs dels busos slack
vec_Pi[:] = np.nan_to_num(df_bus.iloc[:, 1])  # emplenar el vector de números
vec_Qi[:] = np.nan_to_num(df_bus.iloc[:, 2])
vec_Vi[:] = np.nan_to_num(df_bus.iloc[:, 3])
V_sl = []  # tensions dels oscil·lants

for i in range(n):  # cerca per a guardar els índexs
    if df_bus.iloc[i, 5] == 'PQ':
        pq.append(i)
    elif df_bus.iloc[i, 5] == 'PV':
        pv.append(i)
    elif df_bus.iloc[i, 5] == 'Slack':
        V_sl.append(df_bus.iloc[i, 3]*(np.cos(df_bus.iloc[i, 4])+np.sin(df_bus.iloc[i, 4])*1j))
        sl.append(i)

pq = np.array(pq)  # índexs en forma de vector
pv = np.array(pv)
npq = len(pq)  # nombre de busos PQ
npv = len(pv)  # nombre de busos PV

if npv > 0 and npq > 0:  # ordenar els índexs
    pqpv = np.sort(np.r_[pq, pv])
elif npq > 0:
    pqpv = np.sort(pq)
elif npv > 0:
    pqpv = np.sort(pv)

pq_x = pq  # guardar els índexs originals
pv_x = pv

npqpv = npq + npv  # nombre de busos incògnita
nsl = n - npqpv  # nombre de busos oscil·lants

vec_P = vec_Pi[pqpv]  # agafar la part del vector necessària
vec_Q = vec_Qi[pqpv]
vec_V = vec_Vi[pqpv]

factor_carrega = 1.0  # factor de càrrega de les potències de tots els busos
vec_P = factor_carrega * vec_P
vec_Q = factor_carrega * vec_Q

vecx_shunts = np.zeros((n, 1), dtype=complex)  # vector amb admitàncies shunt, canviat de signe
for i in range(nl):
    if df_top.iloc[i, 5] == 1:  # si la relació de transformació és unitària
        vecx_shunts[df_top.iloc[i, 0], 0] += df_top.iloc[i, 4] * -1j  # es donen en forma d'admitàncies
        vecx_shunts[df_top.iloc[i, 1], 0] += df_top.iloc[i, 4] * -1j
    else:
        vecx_shunts[df_top.iloc[i, 0], 0] += df_top.iloc[i, 4] * -1j / (df_top.iloc[i, 5] ** 2)
        vecx_shunts[df_top.iloc[i, 1], 0] += df_top.iloc[i, 4] * -1j

for i in range(n):  # afegir les càrregues d'admitància constant
    vecx_shunts[df_bus.iloc[i, 0], 0] += df_bus.iloc[i, 6] * -1
    vecx_shunts[df_bus.iloc[i, 0], 0] += df_bus.iloc[i, 7] * -1j

vec_shunts = vecx_shunts[pqpv]  # reduïda

df = pd.DataFrame(data=np.c_[vecx_shunts, vec_Pi, vec_Qi, vec_Vi], columns=['Ysh', 'P0', 'Q0', 'V0'])
print(df)  # imprimir dades inicials

Yslx = np.zeros((n, n), dtype=complex)  # admitàncies que connecten als oscil·lants
```

```
for i in range(nl):
    tap = df_top.iloc[i, 5]
    ang_tap = df_top.iloc[i, 6]
    tap = abs(tap) * np.cos(ang_tap) + abs(tap) * np.sin(ang_tap) * 1j
    if tap == 1:
        if df_top.iloc[i, 0] in sl:
            Yslx[df_top.iloc[i, 1], df_top.iloc[i, 0]] = 1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) + \
                                                         Yslx[df_top.iloc[i, 1], df_top.iloc[i, 0]]
        elif df_top.iloc[i, 1] in sl:
            Yslx[df_top.iloc[i, 0], df_top.iloc[i, 1]] = 1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) + \
                                                         Yslx[df_top.iloc[i, 0], df_top.iloc[i, 1]]

    else:
        if df_top.iloc[i, 0] in sl:
            Yslx[df_top.iloc[i, 1], df_top.iloc[i, 0]] += +1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) / \
                                                          (np.conj(tap))
        elif df_top.iloc[i, 1] in sl:
            Yslx[df_top.iloc[i, 0], df_top.iloc[i, 1]] += +1 / (df_top.iloc[i, 2] + df_top.iloc[i, 3] * 1j) / tap

Ysl1 = Yslx[:, sl]
Ysl = Ysl1[pqpv, :]  # reduïda al que veuen els busos PQ i PV
# FI DEFINICIÓ OBJECTES INICIALS

# PREPARACIÓ DE LA IMPLEMENTACIÓ
prof = 12  # nombre de coeficients de les sèries

U = np.zeros((prof, npqpv), dtype=complex)  # sèries de voltatges
U_re = np.zeros((prof, npqpv), dtype=float)  # part real de voltatges
U_im = np.zeros((prof, npqpv), dtype=float)  # part imaginària de voltatges
X = np.zeros((prof, npqpv), dtype=complex)  # tensió inversa conjugada
X_re = np.zeros((prof, npqpv), dtype=float)  # part real d'X
X_im = np.zeros((prof, npqpv), dtype=float)  # part imaginària d'X
Q = np.zeros((prof, npqpv), dtype=complex)  # sèries de potències reactives

vec_W = vec_V * vec_V  # mòdul de les tensions al quadrat
dimensions = 2 * npq + 3 * npv  # nombre d'incògnites
Yred = Yseries[np.ix_(pqpv, pqpv)]  # reduir per a deixar de banda els oscil·lants
G = np.real(Yred)  # part real de la matriu
B = np.imag(Yred)  # part imaginària de la matriu

nsl_compt = np.zeros(n, dtype=int)  # nombre de busos oscil·lants trobats abans d'un bus
compt = 0
for i in range(n):
    if i in sl:
        compt += 1
    nsl_compt[i] = compt
if npv > 0 and npq > 0:
    pq_ = pq - nsl_compt[pq]
    pv_ = pv - nsl_compt[pv]
    pqpv_ = np.sort(np.r_[pq_, pv_])
elif npq > 0:
    pq_ = pq - nsl_compt[pq]
    pqpv_ = np.sort(pq_)
elif npv > 0:
    pv_ = pv - nsl_compt[pv]
    pqpv_ = np.sort(pv_)
# FI PREPARACIÓ DE LA IMPLEMENTACIÓ

# TERMES [0]
if nsl > 1:
    U[0, :] = spsolve(Yred, Ysl.sum(axis=1))  # solucionar el sistema
else:
    U[0, :] = spsolve(Yred, Ysl)

X[0, :] = 1 / np.conj(U[0, :])
U_re[0, :] = U[0, :].real
```

```
U_im[0, :] = U[0, :].imag
X_re[0, :] = X[0, :].real
X_im[0, :] = X[0, :].imag
# FI TERMES [0]

# TERMES [1]
valor = np.zeros(npqpv, dtype=complex)

prod = np.dot((Ysl[pqpv_, :]), V_sl[:])  # intensitat que injecten els oscil·lants

if npq > 0:
    valor[pq_] = prod[pq_] \
                 - Ysl[pq_].sum(axis=1) + (vec_P[pq_] - vec_Q[pq_] * 1j) * X[0, pq_] \
                 + U[0, pq_] * vec_shunts[pq_, 0]
if npv > 0:
    valor[pv_] = prod[pv_] \
                 - Ysl[pv_].sum(axis=1) \
                 + (vec_P[pv_]) * X[0, pv_] \
                 + U[0, pv_] * vec_shunts[pv_, 0]

    RHS = np.r_[valor.real,
                valor.imag,
                vec_W[pv_] - np.real(U[0, pv_] * np.conj(U[0, pv_]))]  # vector de la dreta del sistema d'equacions

    VRE = coo_matrix((2 * U_re[0, pv_], (np.arange(npv), pv_)), shape=(npv, npqpv)).tocsc() # matriu COO a compr.
    VIM = coo_matrix((2 * U_im[0, pv_], (np.arange(npv), pv_)), shape=(npv, npqpv)).tocsc()
    XIM = coo_matrix((-X_im[0, pv_], (pv_, np.arange(npv))), shape=(npqpv, npv)).tocsc()
    XRE = coo_matrix((X_re[0, pv_], (pv_, np.arange(npv))), shape=(npqpv, npv)).tocsc()
    EMPTY = csc_matrix((npv, npv))  # matriu dispera comprimida

    MAT = vstack((hstack((G,   -B,   XIM)),
                  hstack((B,    G,   XRE)),
                  hstack((VRE,  VIM, EMPTY))), format='csc')

else:
    RHS = np.r_[valor.real,
                valor.imag]
    MAT = vstack((hstack((G, -B)),
                  hstack((B, G))), format='csc')


MAT_LU = factorized(MAT.tocsc())  # matriu factoritzada, només cal fer-ho una vegada
LHS = MAT_LU(RHS)  # vector amb les solucions

U_re[1, :] = LHS[:npqpv]  # actualització de les incògnites
U_im[1, :] = LHS[npqpv:2 * npqpv]
if npv > 0:
    Q[0, pv_] = LHS[2 * npqpv:]

U[1, :] = U_re[1, :] + U_im[1, :] * 1j
X[1, :] = (-X[0, :] * np.conj(U[1, :])) / np.conj(U[0, :])
X_re[1, :] = X[1, :].real
X_im[1, :] = X[1, :].imag
# FI TERMES [1]

# TERMES [c>=2]
range_pqpv = np.arange(npqpv)  # tots els busos ordenats

for c in range(2, prof):  # c és la profunditat actual
    if npq > 0:
        valor[pq_] = (vec_P[pq_] - vec_Q[pq_] * 1j) * X[c - 1, pq_] + U[c - 1, pq_] * vec_shunts[pq_, 0]
    if npv > 0:
        valor[pv_] = conv(X, Q, c, pv_, 2) * -1j + U[c - 1, pv_] * vec_shunts[pv_, 0] + X[c - 1, pv_] * vec_P[pv_]
        RHS = np.r_[valor.real,
                    valor.imag,
                    -conv(U, U, c, pv_, 3).real]  # vector de la dreta del sistema d'equacions
    else:
```

```
        RHS = np.r_[valor.real,
                    valor.imag]

    LHS = MAT_LU(RHS)  # vector amb les solucions

    U_re[c, :] = LHS[:npqpv]  # actualització de les incògnites
    U_im[c, :] = LHS[npqpv:2 * npqpv]
    if npv > 0:
        Q[c - 1, pv_] = LHS[2 * npqpv:]

    U[c, :] = U_re[c, :] + 1j * U_im[c, :]
    X[c, range_pqpv] = -conv(U, X, c, range_pqpv, 1) / np.conj(U[0, range_pqpv])
    X_re[c, :] = np.real(X[c, :])
    X_im[c, :] = np.imag(X[c, :])
# FI TERMES [c>=2]

# RESULTATS
Pfi = np.zeros(n, dtype=complex)  # potència activa final
Qfi = np.zeros(n, dtype=complex)  # potència reactiva final
U_sum = np.zeros(n, dtype=complex)  # tensió a partir la suma de coeficients
U_pa = np.zeros(n, dtype=complex)  # tensió amb aproximants de Padé
U_th = np.zeros(n, dtype=complex)  # tensió amb aproximants de Thévenin
U_ait = np.zeros(n, dtype=complex)  # tensió amb Delta d'Aitken
U_shanks = np.zeros(n, dtype=complex)  # tensió amb transformacions de Shanks
U_rho = np.zeros(n, dtype=complex)  # tensió amb Rho de Wynn
U_eps = np.zeros(n, dtype=complex)  # tensió amb Èpsilon de Wynn
U_theta = np.zeros(n, dtype=complex)  # tensió amb Theta de Brezinski
U_eta = np.zeros(n, dtype=complex)  # tensió amb Eta de Bauer
Q_eps = np.zeros(n, dtype=complex)
Q_ait = np.zeros(n, dtype=complex)
Q_rho = np.zeros(n, dtype=complex)
Q_theta = np.zeros(n, dtype=complex)
Q_eta = np.zeros(n, dtype=complex)
Q_sum = np.zeros(n, dtype=complex)
Q_shanks = np.zeros(n, dtype=complex)
Sig_re = np.zeros(n, dtype=complex)  # part real de sigma
Sig_im = np.zeros(n, dtype=complex)  # part imaginària de sigma

Ybus = Yseries - diags(vecx_shunts[:, 0])  # matriu d'admitàncies total

from Funcions import pade4all, thevenin, Sigma, aitken, shanks, rho, epsilon, theta, eta  # importar funcions

# SUMA
U_sum[pqpv] = np.sum(U[:, pqpv_], axis=0)
U_sum[sl] = V_sl
if npq > 0:
    Q_sum[pq] = vec_Q[pq_]
if npv > 0:
    Q_sum[pv] = np.sum(Q[:, pv_], axis=0)
Q_sum[sl] = np.nan
# FI SUMA

# PADÉ
Upa = pade4all(prof, U[:, :], 1)
if npv > 0:
    Qpa = pade4all(prof-1, Q[:, pv_], 1)  # trobar reactiva amb Padé
U_pa[sl] = V_sl
U_pa[pqpv] = Upa
Pfi[pqpv] = vec_P[pqpv_]
if npq > 0:
    Qfi[pq] = vec_Q[pq_]
if npv > 0:
    Qfi[pv] = Qpa
Pfi[sl] = np.nan
Qfi[sl] = np.nan
# FI PADÉ
```

```
limit = 8  # límit per a no utilitzar tots els coeficients
if limit > prof:
    limit = prof - 1

# SIGMA
Ux1 = np.copy(U)
Sig_re[pqpv] = np.real(Sigma(Ux1, X, prof - 1, V_sl))
Sig_im[pqpv] = np.imag(Sigma(Ux1, X, prof - 1, V_sl))
Sig_re[sl] = np.nan
Sig_im[sl] = np.nan

arrel = np.zeros(n, dtype=float)  # discriminant
arrel[sl] = np.nan
arrel[pqpv] = 0.25 + np.abs(Sig_re[pqpv]) - np.abs(Sig_im[pqpv]) ** 2
# FI SIGMA

# THÉVENIN
Ux2 = np.copy(U)
for i in pq:
    U_th[i] = thevenin(Ux2[:limit, i - nsl_compt[i]], X[:limit, i - nsl_compt[i]])
# FI THÉVENIN

# RECURRENTS
Ux = np.copy(U)
Qx = np.copy(Q)

for i in range(npqpv):
    U_ait[i] = aitken(Ux[:, i], limit)
    U_shanks[i] = shanks(Ux[:, i], limit)
    U_rho[i] = rho(Ux[:, i], limit)
    U_eps[i] = epsilon(Ux[:, i], limit)
    U_theta[i] = theta(Ux[:, i], limit)
    U_eta[i] = eta(Ux[:, i], limit)

    if i in pq_:
        Q_ait[i + nsl_compt[i]] = vec_Q[i]
        Q_shanks[i + nsl_compt[i]] = vec_Q[i]
        Q_rho[i + nsl_compt[i]] = vec_Q[i]
        Q_eps[i + nsl_compt[i]] = vec_Q[i]
        Q_theta[i + nsl_compt[i]] = vec_Q[i]
        Q_eta[i + nsl_compt[i]] = vec_Q[i]

    elif i in pv_:
        Q_ait[i + nsl_compt[i]] = aitken(Qx[:, i], limit)
        Q_shanks[i + nsl_compt[i]] = shanks(Qx[:, i], limit)
        Q_rho[i + nsl_compt[i]] = rho(Qx[:, i], limit)
        Q_eps[i + nsl_compt[i]] = epsilon(Qx[:, i], limit)
        Q_theta[i + nsl_compt[i]] = theta(Qx[:, i], limit)
        Q_eta[i + nsl_compt[i]] = eta(Qx[:, i], limit)

U_ait[pqpv] = U_ait[pqpv_]
U_ait[sl] = V_sl
Q_ait[sl] = np.nan
U_shanks[pqpv] = U_shanks[pqpv_]
U_shanks[sl] = V_sl
Q_shanks[sl] = np.nan
U_rho[pqpv] = U_rho[pqpv_]
U_rho[sl] = V_sl
Q_rho[sl] = np.nan
U_eps[pqpv] = U_eps[pqpv_]
U_eps[sl] = V_sl
Q_eps[sl] = np.nan
U_theta[pqpv] = U_theta[pqpv_]
U_theta[sl] = V_sl
Q_theta[sl] = np.nan
U_eta[pqpv] = U_eta[pqpv_]
U_eta[sl] = V_sl
```

```
Q_eta[sl] = np.nan
# FI RECURRENTS

Ybus = Ybus.todense()

# ERRORS
S_out_sum = np.asarray(U_sum) * np.conj(np.asarray(np.dot(Ybus, U_sum)))
S_in_sum = (Pfi[:] + 1j * Q_sum[:])
error_sum = S_in_sum - S_out_sum

S_out = np.asarray(U_pa) * np.conj(np.asarray(np.dot(Ybus, U_pa)))
S_in = (Pfi[:] + 1j * Qfi[:])
error = S_in - S_out  # error final de potències amb Padé

S_out_ait = np.asarray(U_ait) * np.conj(np.asarray(np.dot(Ybus, U_ait)))
S_in_ait = (Pfi[:] + 1j * Q_ait[:])
error_ait = S_in_ait - S_out_ait

S_out_shanks = np.asarray(U_shanks) * np.conj(np.asarray(np.dot(Ybus, U_shanks)))
S_in_shanks = (Pfi[:] + 1j * Q_shanks[:])
error_shanks = S_in_shanks - S_out_shanks

S_out_rho = np.asarray(U_rho) * np.conj(np.asarray(np.dot(Ybus, U_rho)))
S_in_rho = (Pfi[:] + 1j * Q_rho[:])
error_rho = S_in_rho - S_out_rho

S_out_eps = np.asarray(U_eps) * np.conj(np.asarray(np.dot(Ybus, U_eps)))
S_in_eps = (Pfi[:] + 1j * Q_eps[:])
error_eps = S_in_eps - S_out_eps

S_out_theta = np.asarray(U_theta) * np.conj(np.asarray(np.dot(Ybus, U_theta)))
S_in_theta = (Pfi[:] + 1j * Q_theta[:])
error_theta = S_in_theta - S_out_theta

S_out_eta = np.asarray(U_eta) * np.conj(np.asarray(np.dot(Ybus, U_eta)))
S_in_eta = (Pfi[:] + 1j * Q_eta[:])
error_eta = S_in_eta - S_out_eta
# FI ERRORS

df = pd.DataFrame(
    np.c_[np.abs(U_sum), np.angle(U_sum), np.abs(U_pa), np.angle(U_pa), np.real(Sig_re), np.real(Sig_im),
          np.abs(error[0, :])], columns=['|V| sum', 'A. sum', '|V| Padé', 'A. Padé', 'Sigma re', 'Sigma im', 'S error'])
print(df)

print('Error màxim amb suma: ', max(abs(np.r_[error_sum[0, pqpv]])))
print('Error màxim amb Padé: ', max(abs(np.r_[error[0, pqpv]])))
print('Error màxim amb Delta d\'Aitken: ', max(abs(np.r_[error_ait[0, pqpv]])))
print('Error màxim amb transformacions de Shanks: ', max(abs(np.r_[error_shanks[0, pqpv]])))
print('Error màxim amb Rho de Wynn: ', max(abs(np.r_[error_rho[0, pqpv]])))
print('Error màxim amb Èpsilon de Wynn: ', max(abs(np.r_[error_eps[0, pqpv]])))
print('Error màxim amb Theta de Brezinski: ', max(abs(np.r_[error_theta[0, pqpv]])))
print('Error màxim amb Eta de Bauer: ', max(abs(np.r_[error_eta[0, pqpv]])))



"""
# DELTES D'AITKEN
Ux3 = np.copy(U)
Qx3 = np.copy(Q)
for i in range(npqpv):
    U_ait[i] = aitken(Ux3[:, i], limit)
    if i in pq_:
        Q_ait[i + nsl_counted[i]] = vec_Q[i]
    elif i in pv_:
        Q_ait[i + nsl_counted[i]] = aitken(Qx3[:, i], limit)
U_ait[pqpv] = U_ait[pqpv_]
U_ait[sl] = V_sl
```

```
Q_ait[sl] = np.nan
# FI DELTES D'AITKEN

# EPSILONS DE WYNN
Ux4 = np.copy(U)
Qx4 = np.copy(Q)
for i in range(npqpv):
    U_eps[i] = epsilon(Ux4[:, i], limit)
    if i in pq_:
        Q_eps[i + nsl_counted[i]] = vec_Q[i]
    elif i in pv_:
        Q_eps[i + nsl_counted[i]] = epsilon(Qx4[:, i], limit)
U_eps[pqpv] = U_eps[pqpv_]
U_eps[sl] = V_sl
Q_eps[sl] = np.nan
# FI EPSILONS DE WYNN

# RHO
Ux5 = np.copy(U)
Qx5 = np.copy(Q)
for i in range(npqpv):
    U_rho[i] = rho(Ux5[:, i], limit)
    if i in pq_:
        Q_rho[i + nsl_counted[i]] = vec_Q[i]
    elif i in pv_:
        Q_rho[i + nsl_counted[i]] = rho(Qx5[:, i], limit)
U_rho[pqpv] = U_rho[pqpv_]
U_rho[sl] = V_sl
Q_rho[sl] = np.nan
# FI RHO

# THETA
Ux6 = np.copy(U)
Qx6 = np.copy(Q)
for i in range(npqpv):
    U_theta[i] = theta(Ux6[:, i], limit)
    if i in pq_:
        Q_theta[i + nsl_counted[i]] = vec_Q[i]
    elif i in pv_:
        Q_theta[i + nsl_counted[i]] = theta(Qx6[:, i], limit)
U_theta[pqpv] = U_theta[pqpv_]
U_theta[sl] = V_sl
Q_theta[sl] = np.nan
# FI THETA

# ETA
Ux7 = np.copy(U)
Qx7 = np.copy(Q)
for i in range(npqpv):
    U_eta[i] = eta(Ux7[:, i], limit)
    if i in pq_:
        Q_eta[i + nsl_counted[i]] = vec_Q[i]
    elif i in pv_:
        Q_eta[i + nsl_counted[i]] = eta(Qx7[:, i], limit)
U_eta[pqpv] = U_eta[pqpv_]
U_eta[sl] = V_sl
Q_eta[sl] = np.nan
# FI ETA

# SHANKS
Ux8 = np.copy(U)
Qx8 = np.copy(Q)
for i in range(npqpv):
    U_shanks[i] = shanks(Ux8[:, i], limit)
    if i in pq_:
        Q_shanks[i + nsl_counted[i]] = vec_Q[i]
    elif i in pv_:
```

```
        Q_shanks[i + nsl_counted[i]] = shanks(Qx8[:, i], limit)
U_shanks[pqpv] = U_shanks[pqpv_]
U_shanks[sl] = V_sl
Q_shanks[sl] = np.nan
# FI SHANKS



# ERRORS
S_out = np.asarray(U_pa) * np.conj(np.asarray(np.dot(Ybus.todense(), U_pa)))  # computat amb tensions de Padé
S_in = (Pfi[:] + 1j * Qfi[:])
error = S_in - S_out  # error final de potències


S_out_eps = np.asarray(U_eps) * np.conj(np.asarray(np.dot(Ybus.todense(), U_eps)))
S_in_eps = (Pfi[:] + 1j * Q_eps[:])
error_eps = S_in_eps - S_out_eps

S_out_ait = np.asarray(U_ait) * np.conj(np.asarray(np.dot(Ybus.todense(), U_ait)))
S_in_ait = (Pfi[:] + 1j * Q_ait[:])
error_ait = S_in_ait - S_out_ait

S_out_rho = np.asarray(U_rho) * np.conj(np.asarray(np.dot(Ybus.todense(), U_rho)))
S_in_rho = (Pfi[:] + 1j * Q_rho[:])
error_rho = S_in_rho - S_out_rho

S_out_theta = np.asarray(U_theta) * np.conj(np.asarray(np.dot(Ybus.todense(), U_theta)))
S_in_theta = (Pfi[:] + 1j * Q_theta[:])
error_theta = S_in_theta - S_out_theta

S_out_eta = np.asarray(U_eta) * np.conj(np.asarray(np.dot(Ybus.todense(), U_eta)))
S_in_eta = (Pfi[:] + 1j * Q_eta[:])
error_eta = S_in_eta - S_out_eta

S_out_shanks = np.asarray(U_shanks) * np.conj(np.asarray(np.dot(Ybus.todense(), U_shanks)))
S_in_shanks = (Pfi[:] + 1j * Q_shanks[:])
error_shanks = S_in_shanks - S_out_shanks

S_out_sum = np.asarray(U_sum) * np.conj(np.asarray(np.dot(Ybus.todense(), U_sum)))
S_in_sum = (Pfi[:] + 1j * Q_sum[:])
error_sum = S_in_sum - S_out_sum
# FI ERRORS


df = pd.DataFrame(np.c_[np.abs(U_sum), np.angle(U_sum), np.abs(U_pa), np.angle(U_pa), np.abs(U_th),
                        np.abs(U_eps), np.angle(U_eps), np.abs(U_ait), np.angle(U_ait), np.abs(U_rho),
                        np.angle(U_rho), np.abs(U_theta), np.angle(U_theta), np.abs(U_eta), np.angle(U_eta),
                        np.abs(U_shanks), np.angle(U_shanks), np.real(Pfi), np.real(Qfi), np.abs(error[0, :]),
                        np.real(Sig_re), np.real(Sig_im), s_p, s_n],
                        columns=['|V| sum', 'A. sum', '|V| Padé', 'A. Padé', '|V| Thévenin', '|V| Epsilon',
                                 'A. Epsilon', '|V| Aitken', 'A. Aitken', '|V| Rho', 'A. Rho', '|V| Theta', 'A. Theta',
                                 '|V| Eta', 'A. Eta', '|V| Shanks', 'A. Shanks','P', 'Q', 'S error', 'Sigma re',
                                 'Sigma im', 's+', 's-'])

df = pd.DataFrame(np.c_[np.abs(U_sum), np.angle(U_sum), np.abs(U_pa), np.angle(U_pa), np.abs(U_th),
                        np.abs(U_eps), np.angle(U_eps), np.abs(U_ait), np.angle(U_ait), np.abs(U_rho),
                        np.angle(U_rho), np.abs(U_theta), np.angle(U_theta), np.abs(U_eta), np.angle(U_eta),
                        np.abs(U_shanks), np.angle(U_shanks), np.real(Pfi), np.real(Qfi), np.abs(error_sum[0, :]),
                        np.real(Sig_re), np.real(Sig_im)],
                        columns=['|V| sum', 'A. sum', '|V| Padé', 'A. Padé', '|V| Thévenin', '|V| Epsilon',
                                 'A. Epsilon', '|V| Aitken', 'A. Aitken', '|V| Rho', 'A. Rho', '|V| Theta', 'A. Theta',
                                 '|V| Eta', 'A. Eta', '|V| Shanks', 'A. Shanks','P', 'Q', 'S error', 'Sigma re',
                                 'Sigma im'])

"""
#(df)
#err = max(abs(np.r_[error[0, pqpv]]))  # màxim error de potències
```

```
#print('Error màxim amb Padé: ' + str(err))
#print(err)

#print(max(abs(np.r_[error[0, pqpv]])))
#print(max(abs(np.r_[error_ait[0, pqpv]])))
#print(max(abs(np.r_[error_shanks[0, pqpv]])))
#print(max(abs(np.r_[error_rho[0, pqpv]])))
#print(max(abs(np.r_[error_eps[0, pqpv]])))
#print(max(abs(np.r_[error_theta[0, pqpv]])))
#print(max(abs(np.r_[error_eta[0, pqpv]])))
#print(max(abs(np.r_[error_sum[0, pqpv]])))



# ALTRES:
# .....VISUALITZACIÓ DE LA MATRIU .....
from pylab import *
Amm = abs(MAT.todense())  # passar a densa
figure(1)
f = plt.figure()
imshow(Amm, interpolation='nearest', cmap=plt.get_cmap('gist_heat'))
plt.gray()  # en escala de grisos
plt.show()
plt.spy(Amm)  # en blanc i negre
plt.show()

#f.savefig("matriu_imatge.pdf", bbox_inches='tight')

Bmm = coo_matrix(MAT)  # passar a dispersa
density = Bmm.getnnz() / np.prod(Bmm.shape) * 100  # convertir a percentual
#print('Densitat: ' + str(density) + ' %')

# .....DOMB-SYKES .....
bb = np. zeros((prof, npqpv), dtype=complex)
for j in range(npqpv):
    for i in range(3, len(U) - 1):
        bb[i, j] = (U[i, j]) / (U[i-1, j])  # el Domb-Sykes més bàsic
        #bb[i, j] = np.abs(np.sqrt((U[i + 1, j] * U[i - 1, j] - U[i, j] ** 2) / (U[i, j] * U[i - 2, j] - U[i - 1, j] ** 2)))

vec_1n = np. zeros(prof)
for i in range(3, prof):
    vec_1n[i] = i
    #vec_1n[i] = 1 / i

bus = 5

plt.plot(vec_1n[3:len(U)-1], abs(bb[3:len(U)-1, bus]), 'ro ', markersize=2)
plt.show()

#print(abs(bb[-2, :]))
print(1/max(abs(bb[-2, :])))
print(1/min(abs(bb[-2, :])))


# .....GRÀFIC SIGMA .....
a=[]
b=[]
c=[]

x = np.linspace(-0.25, 1, 1000)
y = np.sqrt(0.25+x)
a.append(x)
b.append(y)
c.append(-y)

plt.plot(np.real(Sig_re), np.real(Sig_im), 'ro', markersize=2)
plt.plot(x, y)
```

```
plt.plot(x, -y)
plt.ylabel('Sigma im')
plt.xlabel('Sigma re')
plt.title('Gràfic Sigma')
plt.show()


# .....................EXTRES.............................
#(Ybus)
#print(Pfi)
#print(Qfi)
```