\State $EliminacionV \gets find(isnan(Voltajes(:,1)))$

\State $EliminacionO \gets find(isnan(Ángulos(:,1)))$

Se crearon dos vectores, con el fin de obtener las posiciones de las variables no conocidas, esta función es posible, gracias a la función find y isnan, una permite buscar y la otra nos permite definir que buscar, en este caso las posiciones que no son numéricas

Posterior a este paso para futuros usos se necesita conocer los nodos PQ que posee el sistema, para esto se decidió hacer un ciclo for, que aplicando nuevamente la funciones find y isnan podemos identificar cuales son las variables no numéricas

\State $la \gets 1$

\For{$la \leq N$}

\State $do \gets length(find(isnan(PPOO(la,:))))$

\If{$do == 1$}

\State $M \gets M+1$

\Else

\State

\EndIf

\State $la++$

\EndFor

El ciclo for recorre las filas de una matriz que contiene las potencias que conocemos del sistema “PPOO” extraída de la información cargada del \textit{Microsoft Excel}, esta acción en cada iteración, buscando posiciones que almacenes variable tipo no numéricas, si solo encuentra un valor no numérico, indica que es un nodo PQ, por tanto, aplicando un condicional, si ‘do’ que almacena el tamaño de la búsqueda es uno, ‘M’ que es la variable escogida para almacenar cuantos nodos PQ hay, sumara una unidad.

Para definir las variables que conocemos, y remplazarlas en el primer vector todo simbólico, se implementaron dos ciclos for con la misma idea

\State $sol \gets find(~isnan(Voltajes))$

Se crea una variable llamada sol, que, en este caso, negando la función isnan, podemos encontrar las posiciones que almacenan variables de tipo numérica

\State $fa \gets find(~isnan(Angulos))$

La variable fa cumple la misma función de la variable sol, identificar en el vector de ángulos cuales posiciones son numéricas.

Una vez identificado este valor se diseñó un ciclo for que nos permite almacenar en el vector V que en primera instancia es simbólico, las variables que se conocen.

\State $si \gets 1$

\For{$si \leq length(sol}$

\State $ V(sol(si)) \gets Voltajes(sol(si))$

\State $si++$

\EndFor

El segundo for cumple la misma función, pero ahora con el vector de los ángulos correspondientemente

\State $si \gets 1$

\For{$si \leq length(la$}

\State $ O(fa(si)) \gets Angulos(fa(si)) $

\State $si++$

\EndFor

Posterior a la definición de las variables que conocemos se implementó un ciclo for anidado, con el fin de definir las siguientes ecuaciones.

P\_{k}=\sum\_{m=1}^{n} \left |V\_{k}Y\_{km}V\_{m} \right | cos\left ( \theta \_{k}-\theta \_{m}-\varphi\_{km}\right )

Q\_{k}=\sum\_{m=1}^{n} \left |V\_{k}Y\_{km}V\_{m} \right | sen\left ( \theta \_{k}-\theta \_{m}-\varphi\_{km}\right )

El motivo de implementar un for anidado nos permite recorrer filas y columnas, en pro de esto la variable iteradora en el primer for se nombró con la letra m y la variable iteradora del for más profundo se nombró con la variable k, si vemos la ecuación, las ecuaciones se forman con variables ligadas directamente con k y m la misma idea hace posible que el for pueda generar las ecuaciones para cada nodo correspondientemente.

\State $m \gets 1$

\For{$m \leq N$}

\State $k \gets 1$

\For{$k \leq N$}

\State $mi \gets abs(k-m)$

\If{$mi == 0$}

\State $Pn(k,m)\gets(abs(V(k))\*abs(Y(k,m))\*abs(V(m))) \* (cos(O(k)-O(m)-Ya(k,m)))$

\State $Qn(k,m) \gets (abs(V(k))\*abs(Y(k,m))\*abs(V(m))) \* (sin(O(k)-O(m)-Ya(k,m)))

$

\Else

\State $Conectados\gets find( (T(:,1)==k & T(:,2)==m)|(T(:,1)==m & T(:,2)==k))$

\State $Cu\gets ~isempty(Conectados)$

ConectadoConectado

\If{$cu == 0$}

\State $Pn(k,m)\gets cu$

\State $Qn(k,m)\gets cu$

\Else

\State $Pn(k,m)\gets(abs(V(k))\*abs(Y(k,m))\*abs(V(m))) \* (cos(O(k)-O(m)-Ya(k,m)))$

\State $Qn(k,m) \gets (abs(V(k))\*abs(Y(k,m))\*abs(V(m))) \* (sin(O(k)-O(m)-Ya(k,m)))

$

\EndIf

\EndIf

\State $k++$

\EndFor

\State $m++$

\EndFor

Como nos damos cuenta además de los dos for también tenemos dos condicionales if else, el primero nos permite conocer las potencias de la diagonal, y el segundo, en caso de existir nodos no conectados, igualar la posición correspondiente en la matriz a cero.

Por ultimo se implementa un for para hacer la sumatoria de cada nodo (fila) para obtener la ecuación de potencias completa.

\State $aa++$

\For{$aa \leq N$

\State $ Pkk(aa,1)\gets sum(Pn(aa,:))$

\State $ Qkk(aa,1)\gets sum(Qn(aa,:))$

\State $aa++$

\EndFor

Si analizamos las variables Pkk y Qkk almacenan todas las potencias, por tanto, es necesario identificar las potencias que necesitamos para el desarrollo de los métodos numéricos, para esta tarea implementamos nuevamente dos ciclos for, uno para potencias reales y otro para potencias imaginarias

\State $i \gets 1$

\For{$i \leq length(EliminacionO)$}

\State $ pot(i)\gets Pkk(EliminacionO(i))$

\State $ Oe(i)\gets O(EliminacionO(i))$

\State $ Pe(i)\gets DP(EliminacionO(i))$

\State $i++$

\EndFor

De la misma forma para la potencia reactiva

\State $i \gets 1$

\For{$i \leq length(EliminacionV)$}

\State $ poti(i)\gets Qkk(EliminacionO(i))$

\State $ Ve(i)\gets V(EliminacionO(i))$

\State $ Qe(i)\gets DQ(EliminacionO(i))$

\State $i++$

\EndFor

Los dos for aprovechan el vector anteriormente utilizado para encontrar variables no numéricas para eliminar las posiciones que no se necesitan para el procedimiento, además también se creó un vector simbólico de potencias que nos permita identificar que variable se esa evaluando; como sabemos se analizo y se elimino por aparte, por tanto, se crea otra variable que una lo que se encontró en los dos ciclos for propuestos.

\State $Poteliminado \gets [pot;poti]$

\State $Salidaeliminado \gets [Oe;Ve]$

\State $Entradaelminado \gets [Pe;Qe]$

Ya cumpliendo con todo el procesamiento que es necesario ahora se genera el jacobiano gracias a la función jacobian propia de Matlab y los dos vectores hallados en el paso anterior

\State $Jacobiano \gets jacobian(Poteliminado,Salidaeliminado)$

Para usos futuros (Modificaciones del metodo newton raphson) se definieron los componentes de la matriz Jacobiana

\State $Hm \gets Jacobiano(1:(N-1),1:(N-1))$

\State $Nm \gets Jacobiano(1:(N-1),N:(N-1+M))$

\State $Jm \gets Jacobiano(N:(N-1+M),1:(N-1))$

\State $Lm \gets Jacobiano(N:(N-1+M), N:(N-1+M))$

Antes de empezar a iterar se necesitan definir las condiciones iniciales

\State $re\_s \gets length(EliminacionO)$

\State $re\_b \gets length(EliminacionV)$

\State $viv \gets 1$

\For{$viv \leq re\_s}$

\State $ O(EliminacionO(viv)) \gets 0 $

\State $ viv ++$

\EndFor

\State $vivi \gets 1$

\For{$viv \leq re\_b}$

\State $ V(EliminacionV(vivi)) \gets 1 $

\State $ vivi ++$

\EndFor

Los siguientes dos ciclos for nos permiten definir gracias a funciones de matlab las variables simbólicas, en pocas palabras, cada ciclo hace la asignación de variable double a una variable syms

\State $k \gets 1$

\For{$k \leq N}$

\State $eval(sprintf(' O%d = O(k) ', k))$

\State $ k ++$

\EndFor

\State $k \gets 1$

\For{$k \leq N}$

\State $eval(sprintf(' V%d = V(k) ', k))$

\State $ k ++$

\EndFor

Finalmente, por medio de un ‘Switch case’ permitimos que el usuario decida que método usar.

---- Newton Rapshon

Se define iteración numero 0

\State $Pcal0 \gets eval(Poteliminado)$

\State $Jacobiano0 \gets eval(Jacobiano)$

\State $DeltaP0 \gets Pesp-Pcal0$

\State $Delta0 \gets (inv(Jacobiano0))\*DeltaP0$

\State $Variable0 \gets eval(Salidaeliminado) + Delta0$

Ahora definimos la épsilon que será nuestro criterio para poder finalizar el ciclo while

\State $e \gets 0.001$

\State $a \gets 600$

\State $L \gets 0$

L será la variable que nos indica cuantas iteraciones tenemos en cada método, al tener una actualización de nuestras variables es necesario volver a convertir por medio del for antes usado de variable simbólica a numérica.

\State $k \gets 1$

\For{$k \leq N}$

\State $eval(sprintf(' O%d = O(k) ', k))$

\State $ k ++$

\EndFor

\State $k \gets 1$

\For{$k \leq N}$

\State $eval(sprintf(' V%d = V(k) ', k))$

\State $ k ++$

\EndFor

Antes de entrar al ciclo while se definio una variable detecta, igual a cero, esta variable debe ser igual al numero de variables para poder salir del ciclo while.

re\_s = length(EliminacIonO)  
re\_b = length(EliminacIonV)

\State $detecta \gets 0$

\While {$Ni ~= detecta$}

    \State $detecta \gets 0$

\State $f \gets 1$

\For{$f \leq Ni}$

\State $t \gets e <= abs(DeltaP0(f,1))$

\State $t \gets eval(t)$

\State $detecta \gets ~t + detecta$

\EndFor

\State $L \gets L+1$

\State $Pcal0 \gets eval(Poteliminado)$

\State $Jacobiano0 \gets eval(Jacobiano)$

\State $DeltaP0 \gets Pesp-Pcal0$

\State $Delta0 \gets (inv(Jacobiano0))\*DeltaP0$

\State $Variable0 \gets eval(Salidaeliminado) + Delta0$

\State $k \gets 1$

\For{$k \leq N}$

\State $eval(sprintf(' O%d = O(k) ', k))$

\State $ k ++$

\EndFor

\State $k \gets 1$

\For{$k \leq N}$

\State $eval(sprintf(' V%d = V(k) ', k))$

\State $ k ++$

\EndFor

\EndWhile

Como se observa nuevamente en el ciclo while se añadieron los dos for que nos permiten volver a definir nuestras variables, primero simbólicas y luego numéricas.

Para poder salir del ciclo while las tres variables deben ser menores a épsilon, cuanto este sea el caso la variable detecta será igual al numero de incógnitas, y se podrá salir del ciclo while

Los métodos a continuación deben su lógica al anterior ya explicado, con el fin de no ser redundante, se explicará las modificaciones que se le realizaron al jacobiano, luego de esto el algoritmo ingresa al ciclo while y realiza el mismo procedimiento ya explicado, pero por supuesto con las variaciones a la matriz del jacobiano correspondiente.

---- Newton Rapshon Modificado

Para desarrollar este método es necesario multiplicar la componente Nm y Lm por la variable V2, luego se adiciono al jacobiano de la siguiente forma.

\State $Nm \gets Nm\*V2$

\State $Lm \gets Lm\*V2$

Se vuelve a integrar el jacobiano

\State $Jacobianom \gets [Hm,Nm;Jm,Lm]$

Y se define de nuevo el Jacobiano

\State $Jacobiano \gets Jacobianom$

A su vez sabemos que el delta de variables también se debe modificado por tanto es necesario redefinir esta variable además adicionar una variable que nos permita saber que incógnitas están relacionadas con Nm y Lm.

\State $Delta0 \gets (inv(Jacobiano0))\*DeltaP0$

\State $ NRM \gets length(pot)+length(poti)$

Esta variable nos permite saber el tamaño del vector de incógnitas

\State $ Delta0(length(pot)+ 1:NRM) \gets (Delta0(length(pot)+1:NRM))\*V2$

Se modifica entonces la variable a continuación de las incógnitas asociadas a la potencia real.

A continuación, el algoritmo conduce al ciclo while explicado anteriormente

---- Newton Rapshon desacoplado

Al igual que el anterior método, los componentes Nm y Lm son multiplicados por V2

\State $Nm \gets Nm\*V2$

\State $Lm \gets Lm\*V2$

Sin embargo, recordemos que la razón de la característica desacoplado se debe a las componentes que toman el valor de cero, que teóricamente son

\State $Nm \gets zeros(size(Jm))$

\State $Lm \gets zeros(size(Nm))$

Fue posible gracias a la función zeros que genera una matriz cero con las dimensiones que especifiquemos.

Y redefinimos entonces nuevamente el jacobiano

\State $Jacobianom \gets [Hm,Nm;Jm,Lm]$

\State $Jacobiano \gets Jacobianom$

A su vez sabemos que el delta de variables también se debe modificado por tanto es necesario redefinir esta variable además adicionar una variable que nos permita saber que incógnitas están relacionadas con Nm y Lm.

\State $Delta0 \gets (inv(Jacobiano0))\*DeltaP0$

\State $ NRM \gets length(pot)+length(poti)$

Esta variable nos permite saber el tamaño del vector de incógnitas

\State $ Delta0(length(pot)+ 1:NRM) \gets (Delta0(length(pot)+1:NRM))\*V2$

Se modifica entonces la variable a continuación de las incógnitas asociadas a la potencia real.

A continuación, el algoritmo conduce al ciclo while explicado anteriormente

---- Newton Rapshon desacoplado rápido

Nuevamente se plantea el jacobiano de acuerdo con el Newton Rapshon desacoplado, sin embargo, como sabemos para este método se realiza un reemplazo con la parte imaginaria de la admitancia que correspondiente a esa posición.

Se diseñaron dos ciclos for anidados que nos permiten realizar la tarea anteriormente descrita.

\State $z \gets 1$

\For{$z \leq length(EliminacionO)}$

\State $k \gets 1) $

\For{$k \leq length(EliminacionO)}$

\State $ReemplaYa(z,k) \gets -imag(Y(EliminacionO(z),EliminacionO(k)))) $

\EndFor

\EndFor

Se realiza la misma lógica pero ahora para los voltajes

\State $z \gets 1$

\For{$z \leq length(EliminacionV)}$

\State $k \gets 1) $

\For{$k \leq length(EliminacionV)}$

\State $ReemplaYv(z,k) \gets -imag(Y(EliminacionO(z),EliminacionO(k)))) $

\EndFor

\EndFor

Luego se establece que

\State $Nm \gets ReemplaYa$

\State $Lm \gets ReemplaYv$

Redefiniendo el jacobiano

\State $Jacobianom \gets [Hm,Nm;Jm,Lm]$