EE 523: Power System Stability and Control Report

Aryan Ritwajeet Jha

011807182

Washington State University, Pullman

Prof. Mani V. Venkatsumbramanian

05 May 2023

Summary of Work Completed:

	Type 3	Type 2	Type 1	Remarks
Dynamic Initialization	✅	✅	✅	
Small Signal Stability Analysis	✅	🟨	✅	
Transient Stability Analysis	✅	🟨	🔶	
Small Signal Stability with PSS from chosen Project Paper	NA	NA	🟨	Proposed PSS is a higher order transfer function block. MATLABs fsolve seems to be unable to solve for double derivatives. Of course, suitable modifications could have been made, but due to lack of time, couldn't be implemented. A set of hand-written equations is presented which if implemented, should be sufficient for modelling the PSS in MATLAB.
Remarks		Infeasible Eigenvalues for Type 2		

Legend:

✅	Implemented and Successfully Running
🟨	Implemented but NOT giving expected results
🔶	NOT Implemented

Questions for Assignment 08

Question 01:

Powerflow:

Consider the Kundur system in example 12.6 of Kundur book. Assume there are three lines connecting buses 7 and 8 and two lines connecting buses 8 and 9. Assume the capacities of the shunt capacitors have been increased to be 400 MVar each at buses 7 and 9. Resolve the power-flow using the Newton-Raphson algorithm.

CDF File:

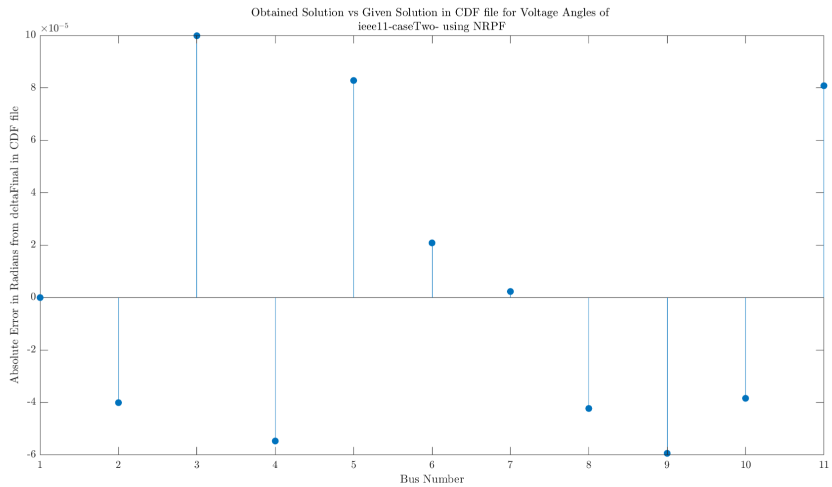
```
??/??/?? UW ARCHIVE          100.0  IEEE 11 Bus Test Case
BUS DATA FOLLOWS             11 ITEMS
 1 Bus 1      HV  1  1  3 1.030  20.20        0.0        0.0        700.0  185.0    0.0  1.030    0.0        0.0  0.0    0.0        0
 2 Bus 2      HV  1  1  2 1.010  10.86        0.0        0.0        700.0  235.0    0.0  1.010    0.0        0.0  0.0    0.0        0
 3 Bus 3      HV  2  1  2 1.030   1.18        0.0        0.0        719.0  176.0    0.0  1.030    0.0        0.0  0.0    0.0        0
 4 Bus 4      HV  2  1  2 1.010  -8.88        0.0        0.0        700.0  202.0    0.0  1.010    0.0        0.0  0.0    0.0        0
 5 Bus 5      HV  1  1  0 1.020  13.90        0.0        0.0        0.0        0.0        0.0  1.000    0.0        0.0  0.0    0.0        0
 6 Bus 6      LV  1  1  0 1.012   4.30        0.0        0.0        0.0        0.0        0.0  1.000    0.0        0.0  0.0    0.0        0
 7 Bus 7      ZV  1  1  0 1.021  -3.45        967.0      100.0      0.0        0.0        0.0  1.000    0.0        0.0  0.0    4.0        0
 8 Bus 8      TV  3  1  0 1.010 -11.46        0.0        0.0        0.0        0.0        0.0  1.000    0.0        0.0  0.0    0.0        0
 9 Bus 9      LV  2  1  0 1.002 -23.60   1767.0   100.0      0.0        0.0        0.0  1.000    0.0        0.0  0.0    4.0        0
 10 Bus 10    LV  2  1  0 1.001 -15.51        0.0        0.0        0.0        0.0        0.0  1.000    0.0        0.0  0.0    0.0        0
 11 Bus 11    LV  2  1  0 1.015  -5.40        0.0        0.0        0.0        0.0        0.0  1.000    0.0        0.0  0.0    0.0        0
-999
BRANCH DATA FOLLOWS          10 ITEMS
 1            5  1  1 1 0  0.00000  0.01667    0.0        0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 2            6  1  1 1 0  0.00000  0.01667    0.0        0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 3  11  2  1 1 0  0.00000  0.01667    0.0        0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 4  10  2  1 1 0  0.00000  0.01667    0.0        0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 5            6  1  1 1 0  0.00250  0.02500    0.04375    0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 6            7  1  1 1 0  0.00100  0.01000    0.01750    0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 7            8  1  1 1 0  0.00367  0.03667    0.57750    0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 8            9  2  1 1 0  0.00550  0.05500    0.38500    0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 9  10  2  1 1 0  0.00100  0.01000    0.01750    0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
 10  11  2  1 1 0  0.00250  0.02500    0.04375    0        0        0        0 0  0.0    0.0 0.0    0.0    0.0    0.0  0.0
-999
LOSS ZONES FOLLOWS           1 ITEMS
 1 IEEE 9 BUS
-99
INTERCHANGE DATA FOLLOWS                1 ITEMS
 1            2 Bus 2      HV           0.0  999.99  IEEE11  IEEE 11 Bus Test Case
-9
```

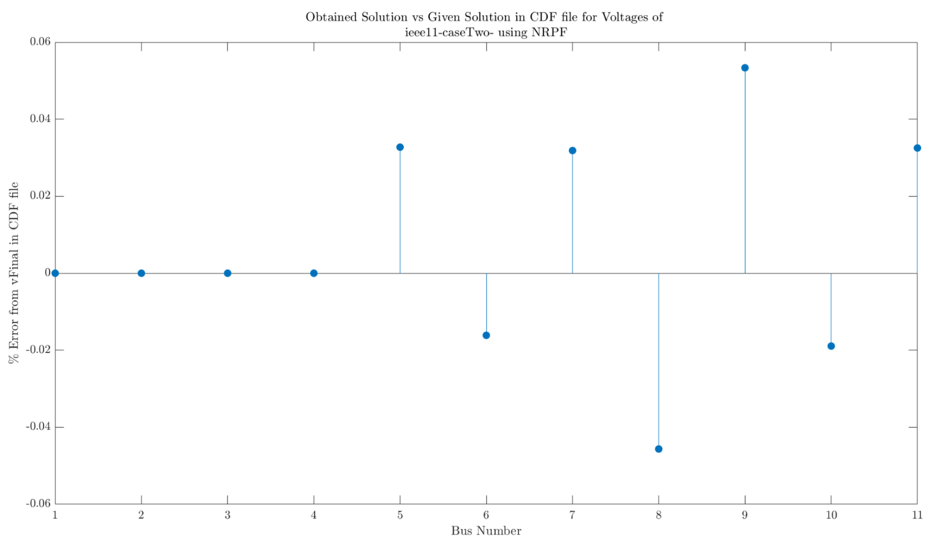
TIE LINES FOLLOWS 0 ITEMS

-999

END OF DATA

Power Flow Results Plots:





Power Flow Results Table:

resultTable = 11×4 table

	P	Q	V	delta
1 Bus 1	6.9125	0.9763	1.0300	0
2 Bus 2	6.9999	0.2874	1.0100	-0.1631
3 Bus 3	7.1900	1.3188	1.0300	-0.3319
4 Bus 4	6.9999	0.9600	1.0100	-0.5076
5 Bus 5	-0.0003	0.0001	1.0203	-0.1099
6 Bus 6	0.0001	0.0004	1.0118	-0.2775
7 Bus 7	-9.6704	-0.9992	1.0213	-0.4128
8 Bus 8	-0.0002	0.0006	1.0095	-0.5526
9 Bus 9	-17.6690	-0.9991	1.0025	-0.7645
10 Bus 10	0.0003	0.0005	1.0008	-0.6233
11 Bus 11	0	0.0001	1.0153	-0.4467

Y_Bus values:

ybusSparse =

(1,1)   0 -   59.988i

(5,1)   0 +   59.988i

(2,2)   0 -   59.988i

(6,2)	0 + 59.988i
(3,3)	0 - 59.988i
(11,3)	0 + 59.988i
(4,4)	0 - 59.988i
(10,4)	0 + 59.988i
(1,5)	0 + 59.988i
(5,5)	3.9604 - 99.57i
(6,5)	-3.9604 + 39.604i
(2,6)	0 + 59.988i
(5,6)	-3.9604 + 39.604i
(6,6)	13.861 - 198.57i
(7,6)	-9.901 + 99.01i
(6,7)	-9.901 + 99.01i
(7,7)	12.603 - 121.71i
(8,7)	-2.7022 + 27i
(7,8)	-2.7022 + 27i
(8,8)	4.5024 - 44.52i
(9,8)	-1.8002 + 18.002i
(8,9)	-1.8002 + 18.002i
(9,9)	11.701 - 112.81i

(10,9)	-9.901 + 99.01i
(4,10)	0 + 59.988i
(9,10)	-9.901 + 99.01i
(10,10)	13.861 - 198.57i
(11,10)	-3.9604 + 39.604i
(3,11)	0 + 59.988i
(10,11)	-3.9604 + 39.604i
(11,11)	3.9604 - 99.57i

Question 02:

Small-signal stability:

For the Kundur system from above, we want to study the small-signal stability properties. Assume KD = 2 pu for all generators. Assume a first order exciter control model with KA=50 and TA = 0.01 sec. Assume Vrmin = -4 and VRmax = +4. Efdmin = 0 and Efdmax = 2.0. For the governor model, assume that Tsg = 100 and Ksg = 1 with Psgmin=0 and Psgmax = 1 pu. R=5%. Then carry out initialization and small-signal analysis for each of Type 1, 2 and 3 models. For Type 1, assume the loads to be 50% P and 50% Z. Represent the equations in the standard DAE form for Type 1 and ODE form for Type 2.

1) Starting from the power-flow solution, initialize the steady-state values of all the dynamic variables.
2) Linearize the equations and find the system Jacobian matrix. You can use numerical differencing to compute the Jacobian entries numerically.
3) Find all eigenvalues and eigenvectors.
4) Compute all the participation factors and analyze each mode.
5) Design Power System Stabilizers (PSSs) as needed to render the damping ratios of all modes to be over 5% for each of Type 1 and Type 2 models.

Type 3:

DynInit:

```
ode_Type3_Init =
```

$$
\begin{pmatrix}
0 = 376.9911\,\omega_2 - 376.9911 \\
0 = 376.9911\,\omega_3 - 376.9911 \\
0 = 376.9911\,\omega_4 - 376.9911 \\
0 = 0.0085\,P_{\mathrm{m2}} - 0.1538\,\omega_2 + 0.0940 \\
0 = 0.0090\,P_{\mathrm{m3}} - 0.1619\,\omega_3 + 0.0973 \\
0 = 0.0090\,P_{\mathrm{m4}} - 0.1619\,\omega_4 + 0.0990
\end{pmatrix}
$$

resultGen = 4×2 table

	E'	theta
Gen 1	1.0300	0
Gen 2	1.0745	0.1465
Gen 3	1.1392	-0.0383
Gen 4	1.1045	-0.2068

yGenTable = 4×4 table

	1	2	3	4
1	3.2168 -11.5866i	1.6434 + 7.9619i	0.9492 + 1.0023i	1.3592 + 1.3569i
2	1.6434 + 7.9619i	1.3830 -10.5241i	0.6581 + 0.6164i	0.9393 + 0.8316i
3	0.9492 + 1.0023i	0.6581 + 0.6164i	1.6473 - 8.1778i	1.9706 + 4.2053i
4	1.3592 + 1.3569i	0.9393 + 0.8316i	1.9706 + 4.2053i	2.9049 - 9.8799i

EE 523: Power System Stability and Control (Spring 2023)

Type 2:

DynInit:

```
unknowns0_Type2_Gens_Vals_Table = 36×1 table
```

	Values
1 **theta2**	0.7510
2 **omega2**	1
3 **E_prime_q2**	0.7998
4 **E_prime_d2**	0.5411
5 **V_R2**	1.7154
6 **P_m2**	6.9999
7 **V_ref2**	1.0443
8 **P_C2**	6.9999
9 **V_q2**	0.6166
10 **V_d2**	0.7999

11 I_q2	4.2349
12 I_d2	5.4936
13 theta3	0.5838
14 omega3	1
15 E_prime_q3	0.8151
16 E_prime_d3	0.5525
17 V_R3	1.7531
18 P_m3	7.1900
19 V_ref3	1.0651
20 P_C3	7.1900
21 V_q3	0.6275
22 V_d3	0.8168
23 I_q3	4.3240

24 I_d3	5.6277
25 theta4	0.4106
26 omega4	1
27 E_prime_q4	0.7986
28 E_prime_d4	0.5428
29 V_R4	1.7249
30 P_m4	6.9999
31 V_ref4	1.0445
32 P_C4	6.9999
33 V_q4	0.6134
34 V_d4	0.8024
35 I_q4	4.2482
36 I_d4	5.5578

```
x0_Type2_Gens_Table = 18×1 table
```

	Values
1 theta2	0.7510
2 omega2	1
3 E_prime_q2	0.7998
4 E_prime_d2	0.5411
5 V_R2	1.7154
6 P_m2	6.9999
7 theta3	0.5838
8 omega3	1
9 E_prime_q3	0.8151
10 E_prime_d3	0.5525
11 V_R3	1.7531
12 P_m3	7.1900

13 theta4	0.4106
14 omega4	1
15 E_prime_q4	0.7986
16 E_prime_d4	0.5428
17 V_R4	1.7249
18 P_m4	6.9999

Small Signal Stability:

Jacobian:

theta2 omega2 E_prime_q2 E_prime_d2 V_R2 P_m2 theta3 omega3 E_prime_q3 E_prime_d3 V_R3 P_m3 theta4 omega4 E_prime_q4 E_prime_d4 V_R4 P_m4

_______ _______ ___________ __________ _____ ______ _______ ______ __________ __________ _____ _____ _______ ______ ___________ __________ _____ _____

theta2 0 376.99 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

omega2 -0.0725 -0.0171 -0.095 0.0263 0

0.0085 0.004 0 -0.0028 -0.0069 0 0 0.0037

0 -0.0058 -0.0086 0 0

E_prime_q2 -0.1855 0 -0.3443 -0.0288 0.125

0 0.0185 0 0.0104 -0.0157 0 0 0.0247 0

0.0098 -0.0242 0 0

E_prime_d2 1.4614 0 0.4418 -5.8619 0 0

0.0029 0 0.2401 0.1592 0 0 0.0817 0

0.3715 0.1502 0 0

V_R2 -4439.9 0 -5316.6 377.54 -100

0 -242.38 0 -136.03 205.15 0 0 -323.24 0

-128.38 317.5 0 0

P_m2 0 -0.2 0 0 0 -0.01 0

0 0 0 0 0 0 0 0 0 0 0

theta3 0 0 0 0 0 0 0

376.99 0 0 0 0 0 0 0 0 0 0

omega3 0.0061 0 -0.00044056 -0.008 0
0 -0.0479 -0.018 -0.0818 0.0033 0 0.009 0.0325
0 -0.00064669 -0.0411 0 0

E_prime_q3 0.0172 0 0.0149 -0.0114 0 0
-0.1334 0 -0.2954 -0.0343 0.125 0 0.0874 0
0.0792 -0.0555 0 0

E_prime_d3 -0.0888 0 0.1745 0.2292 0 0
0.6998 0 0.5262 -5.1123 0 0 -0.5079 0
0.8516 1.2148 0 0

V_R3 -225.72 0 -196.24 149.45 0 0
-5037.2 0 -5969.5 450.63 -100 0 -1147 0
-1040.3 729.24 0 0

P_m3 0 0 0 0 0 0 0 -0.2
0 0 0 -0.01 0 0 0 0 0 0

theta4 0 0 0 0 0 0 0 0
0 0 0 0 0 376.99 0 0 0 0

omega4 0.0092 0 0.00099266 -0.0109 0 0
0.0384 0 0.0131 -0.0381 0 0 -0.0618 -0.018
-0.0992 0.0099 0 0.009

E_prime_q4 0.0225 0 0.0229 -0.0127 0 0

0.0722 0 0.0934 -0.0253 0 0 -0.1331 0

-0.3308 -0.0605 0.125 0

E_prime_d4 -0.1753 0 0.1942 0.3506 0 0

-0.9523 0 0.3886 1.4317 0 0 1.3585 0

0.9279 -5.6561 0 0

V_R4 -296.41 0 -301.22 166.81 0 0

-951.76 0 -1230.1 333.85 0 0 -4977.2 0

-5522.2 797.27 -100 0

P_m4 0 0 0 0 0 0 0 0

0 0 0 0 0 -0.2 0 0 0 -0.01

Je_Type2_Gens_Table = 18×18 table

	heta2	mega2	_prime_q2	_prime_d2	_R2	_m2	heta3	mega3	_prime_q3	_prime_d3	_R3	_m3	heta4	mega4	_prime_q4	_prime_d4	_R4	_m4
theta2		76.9911																

omega2	0.0725	0.0171	0.0950	.0263		.0085	.0040		0.0028	0.0069			.0037		0.0058	0.0086		
E_prime_q2	0.1855		0.3443	0.0288	.1250		.0185		.0104	0.0157			.0247		.0098	0.0242		
E_prime_d2	.4614		.4418	5.8619			.0029		.2401	.1592			.0817		.3715	.1502		
V_R2	4.4399e+03		5.3166e+03	77.5412	100		242.3760		136.0339	05.1494			323.2366		128.3760	17.4983		
P_m2		0.2000				0.0100												
theta3								76.9911										
omega3	.0061		4.4056e-04	0.0080			0.0479	0.0180	0.0818	.0033		.0090	.0325		6.4669e-04	0.0411		

E_prime_q3	.0172		.0149	0.0114			0.1334		0.2954	0.0343	.1250		.0874		.0792	0.0555		
0 E_prime_d3	0.0888		.1745	.2292			.6998		.5262	5.1123			0.5079		.8516	.2148		
1 V_R3	225.7156		196.2388	49.4509			5.0372e+03		5.9695e+03	50.6272	100		1147		1.0403e+03	29.2441		
2 P_m3								0.2000				0.0100						
3 theta4														76.9911				
4 omega4	.0092		.9266e-04	0.0109			.0384		.0131	0.0381			0.0618	0.0180	0.0992	.0099		.0090

5 E_prime_q4	.0225		.0229	0.0127			.0722		.0934	0.0253			0.1331		0.3308	0.0605	.1250	
6 E_prime_d4	0.1753		.1942	.3506			0.9523		.3886	.4317			.3585		.9279	5.6561		
7 V_R4	296.4063		301.2165	66.8143			951.7605		1.2301e+03	33.8473			4.9772e+03		5.5222e+03	97.2723	100	
8 P_m4														0.2000				0.0100

lambdasType2 = 18×1 complex

-90.3864 + 0.0000i

-92.8899 + 0.0000i

-93.8844 + 0.0000i

3.7230 + 0.0000i

-3.3880 + 5.0894i

-3.3880 - 5.0894i

-4.0915 + 4.2569i

-4.0915 - 4.2569i

-5.8262 + 3.0605i

-5.8262 - 3.0605i

So eigenvalues are coming in the RHP, which makes it small signal unstable.

Type 1:

Jacobian:

1.0e+03 *

Columns 1 through 15

0	0	0	0.3770	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0.3770	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0.3770	0	0	0	0	0	0	0	0	0

-0.0002 -0.0000 -0.0000 -0.0000 0 0 -0.0002 -0.0000

-0.0000 0.0001 0.0000 0.0000 0 0 0

-0.0000 -0.0002 -0.0000 0 -0.0000 0 -0.0000 -0.0002

-0.0000 -0.0000 0.0001 -0.0000 0 0 0

-0.0000 0.0000 -0.0002 0 0 -0.0000 -0.0000 -0.0000

-0.0002 -0.0000 -0.0000 0.0001 0 0 0

-0.0005 0.0001 0.0001 0 0 0 -0.0006 0.0001

0.0001 0.0001 -0.0001 -0.0001 0.0001 0 0

0.0000 -0.0006 0.0000 0 0 0 0.0000 -0.0006

0.0001 0.0000 0.0002 0.0001 0 0.0001 0

0.0000 -0.0000 -0.0005 0 0 0 0.0000

0.0001 -0.0005 0.0000 0.0001 0.0001 0 0 0.0001

0.0033 0.0016 0.0015 0 0 0 0.0020

0.0014 0.0013 -0.0064 -0.0007 -0.0007 0 0 0

0.0001 0.0025 0.0004 0 0 0 0.0003

0.0018 0.0015 0.0002 -0.0057 0.0006 0 0 0

0.0001 -0.0002 0.0036 0 0 0 0.0004

0.0009 0.0028 0.0002 0.0008 -0.0062 0 0 0

-0.1090 -1.9626 -1.8677 0 0 0 -2.4383 -1.6978 -1.5995

-1.5720 0.8448 0.8507 -0.1000 0 0

```
  -0.0826    0.8937   -0.5186         0         0         0   -0.3998   -2.2298   -1.7629
-0.1898   -2.4640   -0.6783         0   -0.1000         0
  -0.1110    0.2445   -0.4961         0         0         0   -0.5376   -1.0931   -3.3175
-0.2552   -1.0086   -1.7798         0         0   -0.1000
        0         0         0   -0.0002         0         0         0         0         0         0
0         0         0         0         0
        0         0         0         0   -0.0002         0         0         0         0         0
0         0         0         0         0
        0         0         0         0         0   -0.0002         0         0         0         0
0         0         0         0         0
```

Columns 16 through 18

```
        0         0         0
        0         0         0
        0         0         0
   0.0000         0         0
        0    0.0000         0
        0         0    0.0000
        0         0         0
```

0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
-0.0000	0	0
0	-0.0000	0
0	0	-0.0000

Eigenvalues:

lambdas_Type1 = 18×1 complex

```
-0.6864 - 9.0045i
-6.2148 + 0.0000i
-5.7105 + 0.0000i
-4.4652 + 2.2061i
```

-4.4652 - 2.2061i

-2.4172 + 0.0000i

-1.6255 + 0.0000i

-0.0100 + 0.0000i

-0.0100 + 0.0000i

-0.0100 + 0.0000i

Type 1:

Set of ODEs for DynInit.

$$\left(\begin{array}{c}
0 = 376.9911\,\omega_2 - 376.9911 \\
0 = 376.9911\,\omega_3 - 376.9911 \\
0 = 376.9911\,\omega_4 - 376.9911 \\
0 = 0.0085\,P_{\mathrm{m2}} - 0.0171\,\omega_2 - 0.0427 \\
0 = 0.0090\,P_{\mathrm{m3}} - 0.0180\,\omega_3 - 0.0467 \\
0 = 0.0090\,P_{\mathrm{m4}} - 0.0180\,\omega_4 - 0.0450 \\
0 = 0.1250\,V_{\mathrm{R2}} - 0.0208\,I_{\mathrm{d2}} - 0.1250\,{E'}_{\mathrm{q2}} \\
0 = 0.1250\,V_{\mathrm{R3}} - 0.0208\,I_{\mathrm{d3}} - 0.1250\,{E'}_{\mathrm{q3}} \\
0 = 0.1250\,V_{\mathrm{R4}} - 0.0208\,I_{\mathrm{d4}} - 0.1250\,{E'}_{\mathrm{q4}} \\
0 = 0.3194\,I_{\mathrm{q2}} - 2.5000\,{E'}_{\mathrm{d2}} \\
0 = 0.3194\,I_{\mathrm{q3}} - 2.5000\,{E'}_{\mathrm{d3}} \\
0 = 0.3194\,I_{\mathrm{q4}} - 2.5000\,{E'}_{\mathrm{d4}} \\
0 = 5000\,V_{\mathrm{ref2}} - 100\,V_{\mathrm{R2}} - 5050 \\
0 = 5000\,V_{\mathrm{ref3}} - 100\,V_{\mathrm{R3}} - 5150 \\
0 = 5000\,V_{\mathrm{ref4}} - 100\,V_{\mathrm{R4}} - 5050
\end{array}\right.$$

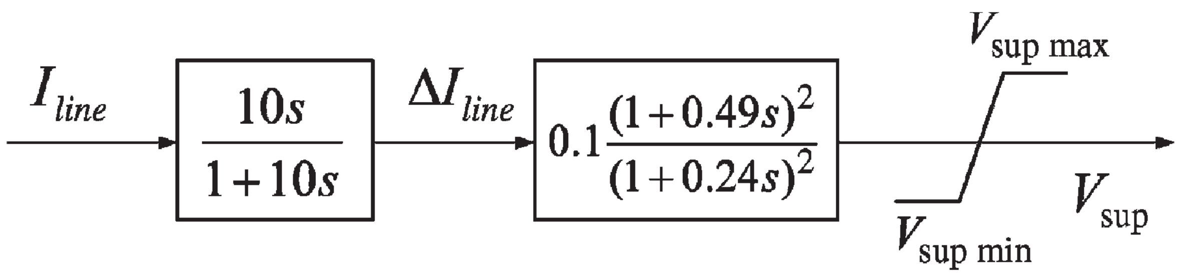
$$
\begin{aligned}
0 &= 0.0100\, P_{\mathrm{C2}} - 0.0100\, P_{\mathrm{m2}} - 0.2000\, \omega_2 + 0.2000 \\
0 &= 0.0100\, P_{\mathrm{C3}} - 0.0100\, P_{\mathrm{m3}} - 0.2000\, \omega_3 + 0.2000 \\
0 &= 0.0100\, P_{\mathrm{C4}} - 0.0100\, P_{\mathrm{m4}} - 0.2000\, \omega_4 + 0.2000 \\
0 &= I_{\mathrm{d2}} - 6.9364\, \sin(\theta_2 + 0.1631) \\
0 &= I_{\mathrm{d3}} - 7.0970\, \sin(\theta_3 + 0.3319) \\
0 &= I_{\mathrm{d4}} - 6.9954\, \sin(\theta_4 + 0.5076) \\
0 &= I_{\mathrm{q2}} - 6.9364\, \cos(\theta_2 + 0.1631) \\
0 &= I_{\mathrm{q3}} - 7.0970\, \cos(\theta_3 + 0.3319) \\
0 &= I_{\mathrm{q4}} - 6.9954\, \cos(\theta_4 + 0.5076) \\
0 &= V_{\mathrm{q2}} - 1.0100\, \cos(\theta_2 + 0.1631) \\
0 &= V_{\mathrm{q3}} - 1.0300\, \cos(\theta_3 + 0.3319) \\
0 &= V_{\mathrm{q4}} - 1.0100\, \cos(\theta_4 + 0.5076) \\
0 &= V_{\mathrm{d2}} - 1.0100\, \sin(\theta_2 + 0.1631) \\
0 &= V_{\mathrm{d3}} - 1.0300\, \sin(\theta_3 + 0.3319) \\
0 &= V_{\mathrm{d4}} - 1.0100\, \sin(\theta_4 + 0.5076)
\end{aligned}
$$

$$
\begin{aligned}
0 &= I_{\mathrm{d2}} - 30\, {E'}_{\mathrm{q2}} + 30\, V_{\mathrm{q2}} \\
0 &= I_{\mathrm{d3}} - 30\, {E'}_{\mathrm{q3}} + 30\, V_{\mathrm{q3}} \\
0 &= I_{\mathrm{d4}} - 30\, {E'}_{\mathrm{q4}} + 30\, V_{\mathrm{q4}} \\
0 &= 16.3636\, {E'}_{\mathrm{d2}} + I_{\mathrm{q2}} - 16.3636\, V_{\mathrm{d2}} \\
0 &= 16.3636\, {E'}_{\mathrm{d3}} + I_{\mathrm{q3}} - 16.3636\, V_{\mathrm{d3}} \\
0 &= 16.3636\, {E'}_{\mathrm{d4}} + I_{\mathrm{q4}} - 16.3636\, V_{\mathrm{d4}}
\end{aligned}
$$

Participation Factors and remarks on inter-area oscillation:

Mode	Type of Oscillation	Coupling State Variables
1.062 Hz	Local oscillation	Theta 2 vs Omega 2
0.892 Hz	Local oscillation	Theta 3 vs Omega 3
0.452 Hz	Inter-area oscillation	Theta 4 vs Theta 2

Project: Implement a PSS in the Type 1 System



Inputs of the PSS: Current flowing between the two areas 1 and 2. Here, the current between buses 7 and 8 is chosen as the input to the PSS.

The first block is a washout filter which negates any steady state values (DC Offset) in the line current, and only allows high frequency signal noise to pass through.

The second block can actually be thought of as two separate lead compensator blocks.

Both Type 1 and Type 2 do not require PSS.

Question 03:

<u>Transient Stability:</u>

We want to study a fault on one of the transmission lines between buses 7 and 8. Assume a solid fault in the middle of the line. Assume Euler integration method with a step size of 1 msec.
1) For tc = 3 cycles, check if the system is stable.
2) Find the critical clearing time.

Repeat for each of Type 1, Type 2 and Type 3 models from your small-signal stability homework solutions.

Type 3:

yGen during Fault:

yGenTable = 4×4 table

	1	2	3	4
1	1.8186 -14.5133i	0.6173 + 6.1083i	0.2689 + 0.3831i	0.3889 + 0.5223i
2	0.6173 + 6.1083i	0.6414 -11.6926i	0.1903 + 0.2393i	0.2740 + 0.3253i
3	0.2689 + 0.3831i	0.1903 + 0.2393i	1.4033 - 8.2672i	1.6277 + 4.0905i
4	0.3889 + 0.5223i	0.2740 + 0.3253i	1.6277 + 4.0905i	2.4236 -10.0261i

ode_Type3_TransientFaultOn =

$$\begin{pmatrix} 0 = 376.9911\,\omega_2 - 376.9911 \\ 0 = 376.9911\,\omega_3 - 376.9911 \\ 0 = 376.9911\,\omega_4 - 376.9911 \\ 0 = 0.0706 - 0.0043\cos(\theta_4 - \theta_2 + 0.8707) - 0.0032\cos(\theta_3 - \theta_2 + 0.8988) - 0.0581\cos(\theta_2 - 1.4701) - 0.0171\,\omega_2 \\ 0 = 0.0663 - 0.0498\cos(\theta_4 - \theta_3 + 1.1921) - 0.0034\cos(\theta_2 - \theta_3 + 0.8988) - 0.0049\cos(\theta_3 - 0.9587) - 0.0180\,\omega_3 \\ 0 = 0.0544 - 0.0498\cos(\theta_3 - \theta_4 + 1.1921) - 0.0045\cos(\theta_2 - \theta_4 + 0.8707) - 0.0067\cos(\theta_4 - 0.9307) - 0.0180\,\omega_4 \end{pmatrix}$$

ode_Type3_TransientPostFault =

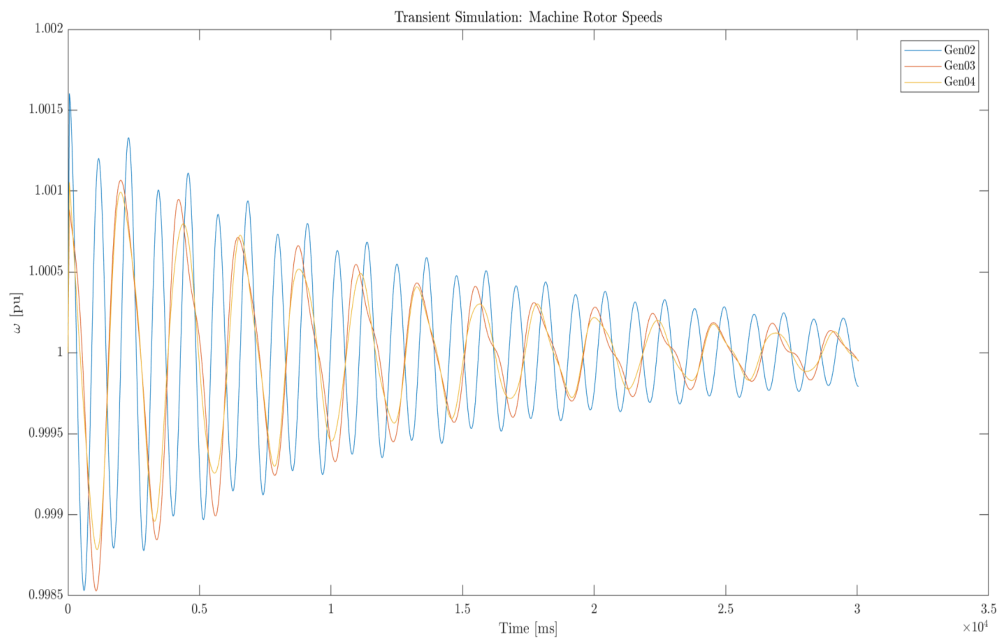
$$\begin{pmatrix} 0 = 376.9911\,\omega_2 - 376.9911 \\ 0 = 376.9911\,\omega_3 - 376.9911 \\ 0 = 376.9911\,\omega_4 - 376.9911 \\ 0 = 0.0629 - 0.0116\cos(\theta_4 - \theta_2 + 0.6973) - 0.0086\cos(\theta_3 - \theta_2 + 0.7254) - 0.0787\cos(\theta_2 - 1.3658) - 0.0171\,\omega_2 \\ 0 = 0.0626 - 0.0537\cos(\theta_4 - \theta_3 + 1.1187) - 0.0090\cos(\theta_2 - \theta_3 + 0.7254) - 0.0133\cos(\theta_3 - 0.7853) - 0.0180\,\omega_3 \\ 0 = 0.0475 - 0.0537\cos(\theta_3 - \theta_4 + 1.1187) - 0.0122\cos(\theta_2 - \theta_4 + 0.6973) - 0.0179\cos(\theta_4 - 0.7572) - 0.0180\,\omega_4 \end{pmatrix}$$

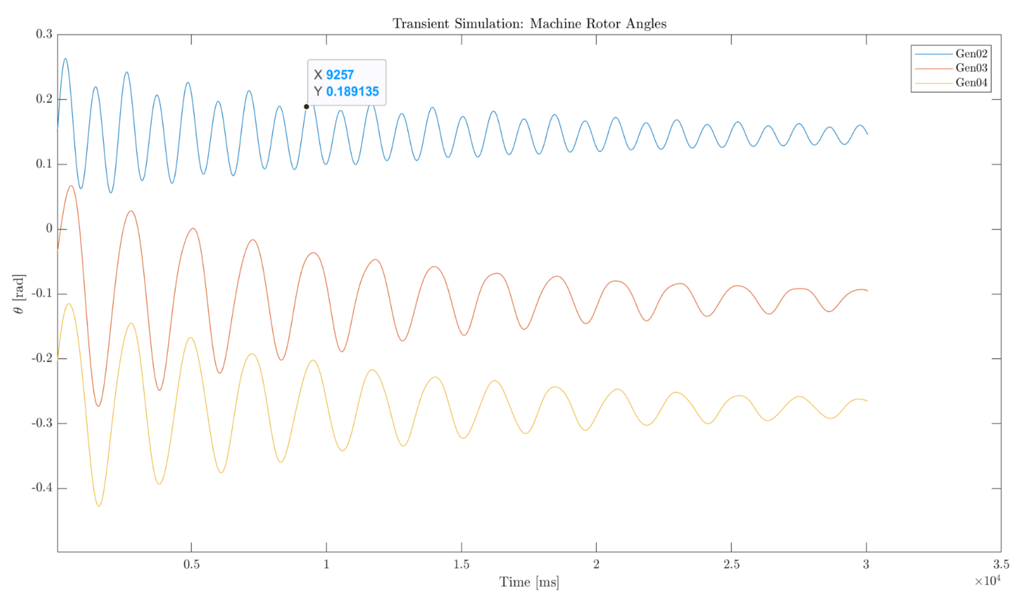
yGen after clearing the fault:

yGenTable = 4×4 table

	1	2	3	4
1	3.2766 -11.3047i	1.6935 + 8.1434i	0.8894 + 0.8893i	1.2717 + 1.2020i
2	1.6935 + 8.1434i	1.4228 -10.4077i	0.6147 + 0.5451i	0.8762 + 0.7340i
3	0.8894 + 0.8893i	0.6147 + 0.5451i	1.7186 - 8.1324i	2.0714 + 4.2656i
4	1.2717 + 1.2020i	0.8762 + 0.7340i	2.0714 + 4.2656i	3.0475 - 9.8000i

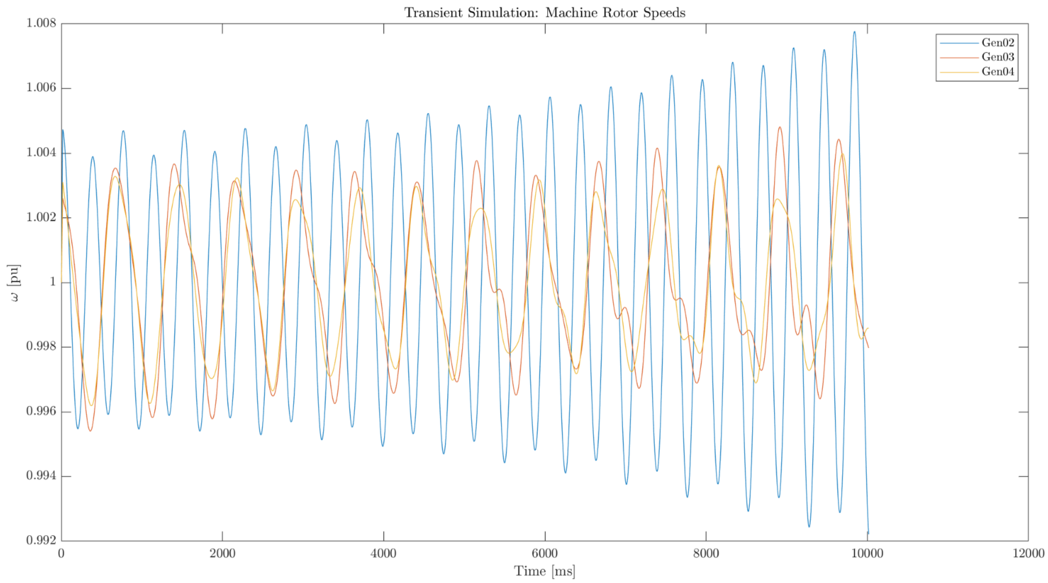
Transient Simulation

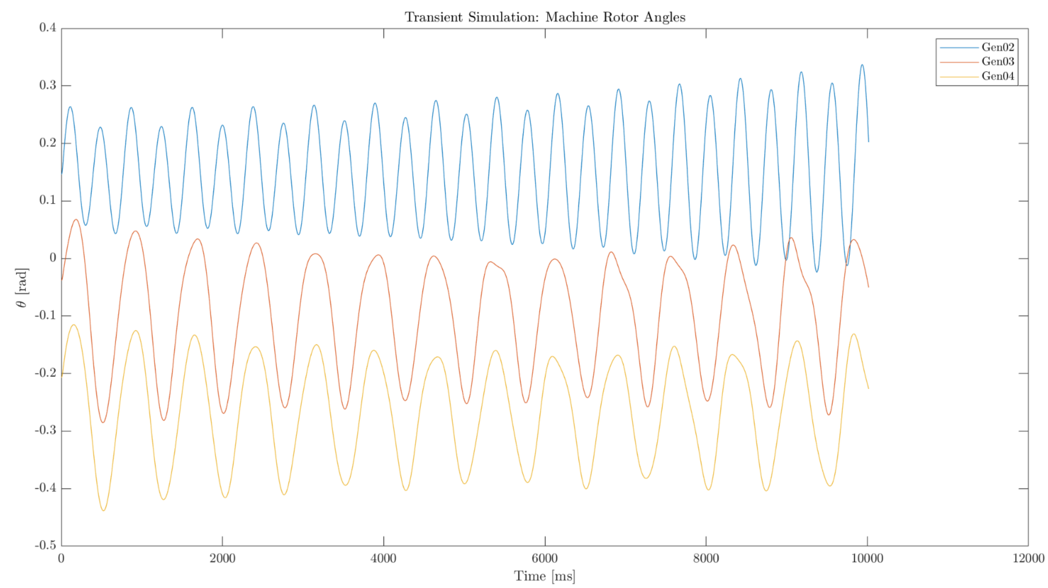




Critical Clearing Time was found to be 7 cycles.

Transient simulation for Type 2 blew up.



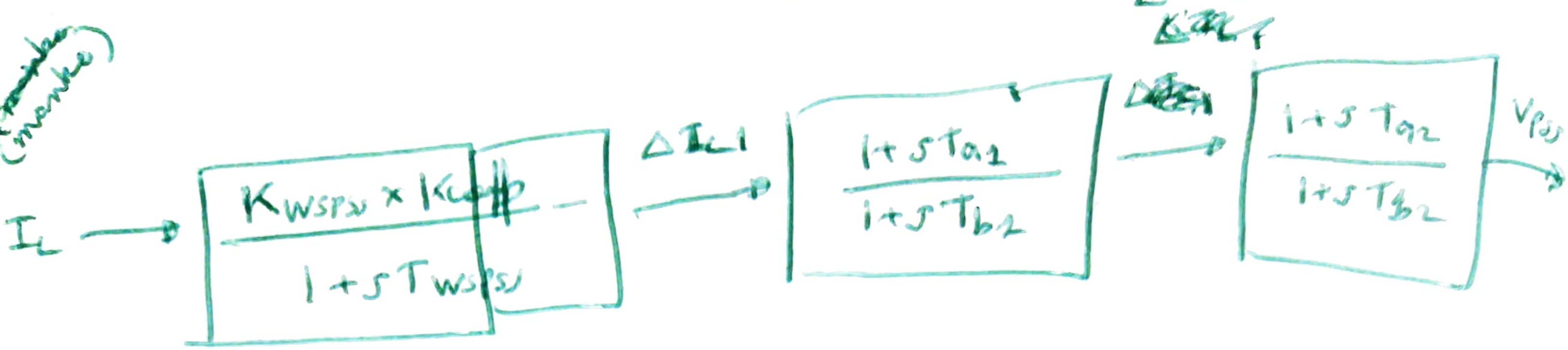


Transient simulation for Type 1 was NOT implemented for lack of time.

References

Kundur, P. S., & Malik, O. P. (2022). Power System Stability and Control. McGraw-Hill Education. Retrieved from https://www.accessengineeringlibrary.com/content/book/9781260473544

Appendix:



$$\underset{\text{delta\_IL2}}{(\dot{\Delta I}_{L_1})} = \frac{1}{T_{wpss}}\left\{ -\Delta I_{L_1} + K_{wpss} \times K_{comp} \times I_L \right\}$$

$I_L = y_{78}(V_7 - V_8)$

$\downarrow$

$-Y_{bus}(7,8)$

$$\Delta I_{L_1} \cdot (1+sT_{a1}) = (1+sT_{b1}) \cdot \Delta I_{L2}$$

$$\Delta I_{L_1} + T_{a1}(\dot{\Delta I}_{L_1}) = \Delta I_{L2} + T_{b1} \cdot (\dot{\Delta I}_{L2})$$

$$\underset{\text{delta\_IL2}}{(\dot{\Delta I}_{L2})} = \frac{1}{T_{b1}}\left\{ -\Delta I_{L2} + \Delta I_{L1} + T_{a1} \cdot (\dot{\Delta I}_{L1}) \right\}$$

$$\underset{\text{v\_pss}}{\dot{V}_{pss}} = \frac{1}{T_{b2}}\left\{ -V_{pss} + \Delta I_{L2} + T_{a2}(\dot{\Delta I}_{L2}) \right\}$$

EE 523 Power System Stability and Control Algorithms

Preamble and Control Inputs

```
tic;
if strcmp(getenv('USERNAME'), 'aryan')
    cd C:\Users\aryan\Documents\documents_general\dablab_files\ee521_and_ee523
end
```

you may continue as usual

```
addpath functions\
systemName = "ieee11-caseTwo-"
```

```
systemName = 
"ieee11-caseTwo-"
```

```
powerFlowMethod =  "NRPF"
```

```
powerFlowMethod = 
"NRPF"
```

```
modelType = "Type3" %Synchronous Machine Model Type
```

```
modelType = 
"Type3"
```

```
useReducedLoadsForPowerFlow = false; %only needed if Type 2 or 3
displayYGen = true; %Show YGen
displayYNet = false; %Show YNet
displayYBus = false;
saveYBus = true;
saveYGenMatrices = true; %Save YNet, Ygr, Yrg, Yrr, YGen
runTransientSimulation = false;
saveTransientRunValues = true;
saveTransientRunPlots = true;
showInternalMatrices = false; %Show Ygg, Ygr, Yrg, Yrr
numIterations = 20; %I don't wait for the system to converge,
printPowerFlowConvergenceMessages = false;
% neither do I care if the system converges earlier.
toleranceLimit = 1e-3; %mean of absolute values of
% corrections should be less than this for convergence to be achieved.
displayRawData = false;
displayRawDataForYNet = false;
displayTables = true; %show busData, branchData ybus,
% basically data structures which are not the final output.
saveBusDataAndBranchData = false;
saveBusDataAndBranchDataNet = false;
printJacobians = false  ; %Print Jacobians during NRPF iterations? Does not work if
displayTables is off.
printMismatches = false; %Print Mismatches during NRPF iterations? Does not work if
displayTables is off.
printCorrections = false;
```

```
disableTaps = false; %Disable Tap-changers when commputing YBus?
showPlots = false;
displayResults = false;
reducedBranchColumnsCDFReading = true;
verboseCDFReading = false; %Will give a verbose output when reading CDF files.
MVAb = 100; %Currently the same for all systems in database.
```

Transient Run Parameters

```
bus1 = 7;
bus2 = 8;
numLinesBeforeFault = 3;
relativeDistance = 0.5;

h = 1e-3;
clearingCycles = 3;
f0 = 60;
clearingTime = clearingCycles/f0;
numStepsFaultOn = ceil(clearingTime/h);
postFaultTime = 30;
numStepsPostFault = ceil(postFaultTime/h);
```

Housekeeping.

```
folder_rawData = "rawData/"; %location of CDF .txt file for the system
file_rawData = strcat(folder_rawData, systemName, "cdf.txt"); %Exact location of
CDF .txt file for the system
folder_processedData = "processedData/";
% Should configure it to be read from the CDF file later.
latex_interpreter %for LaTeX typesetting in plots

if contains(systemName, 'ieee11')
    systemName1 = 'ieee11';
else
    systemName1 = systemName;
end
```

Read CDF file and store the data in neat MATLAB `tables`: `busData` and `branchData`.

```
[busData, branchData, N, numBranchNet] = ...
    readCDF(file_rawData, reducedBranchColumnsCDFReading, verboseCDFReading);
```

Optionally display the system data.

```
if displayRawData
    displayRawDataAsTables(busData, branchData, N, numBranchNet);
end
if saveBusDataAndBranchData
    fileType = '.csv';
```

```
    filenameBusData = strcat(folder_processedData, systemName1, "/busData",
fileType);
    writetable(busData, filenameBusData);
    filenameBranchData = strcat(folder_processedData, systemName1, "/branchData",
fileType);
    writetable(branchData, filenameBranchData);
end
```

Synchronous Machine Parameters

```
[~, ~, ~, ~, ...
        lPQ, lPV, ~, nPV, ...
        ~] = initializeVectors(busData, MVAb);

kVB = 230;
kVB_Gen = 20;
kVB_Gen_side = 20;
MVAb_Gen = 900;

H = [6.5; 6.5; 6.175; 6.175] * (MVAb_Gen/MVAb);
Xd_prime = 0.3* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Xq_prime = 0.55* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Xq = 1.7* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Xd = 1.8* ones(nPV+1, 1) * (MVAb/MVAb_Gen) * (kVB_Gen/kVB_Gen_side)^2;
Tq0_prime = 0.4 * ones(nPV+1, 1);
Td0_prime = 8.0 * ones(nPV+1, 1);
w_s = 2*pi*60;
% K_D = 2*ones(nPV+1, 1) * (MVAb_Gen/MVAb);
K_D = 2*ones(nPV+1, 1);
k_A = 50*ones(nPV+1, 1);
T_A = 0.01*ones(nPV+1, 1);
EfdMax = 2.0*ones(nPV+1, 1);
Efdmin = 0.0*ones(nPV+1, 1);
VRmin = -4*ones(nPV+1, 1);
VRMax = 4*ones(nPV+1, 1);
Tsg = 100*ones(nPV+1, 1);
Ksg = 1*ones(nPV+1, 1);
PsgMax = 1*ones(nPV+1, 1);
Psgmin = 0*ones(nPV+1, 1);
R = 5*1/100*ones(nPV+1, 1);
```

PSS Parameters

```
Kwpss = 10;
Twpss = 10;
Kcomp = 0.1;
Tacomp1 = 0.49;
Tbcomp1 = 0.24;
Tacomp2 = Tacomp1;
Tbcomp2 = Tbcomp1;
```

Powerflow

Extract Y_{Bus}, Adjacency List E from the `branchData table` for Type I OR Make Y_{Bus} (reduced loads only), Y_{Net}, Y_{Gen} for Type II and Type III models.

Then Run Newton Raphson Power Flow and obtain a steady state snapshot of the system variables P_i, Q_i, V_i, δ_i $\forall$ buses $i \in [1, N], i \in \mathbb{N}$

```
if ~strcmp(modelType, "Type1")
    X_prime = 0.5* (Xd_prime + Xq_prime);

    [busDataNet, busDataReducedLoads, busDataGen, branchDataNet, NNet,
numBranchNet] = ...
        modifyForYNet(busData, branchData, N, ...
        numBranchNet, lPV, nPV, ...
        X_prime, MVAb, displayRawDataForYNet);

    if saveBusDataAndBranchDataNet
        fileType = '.csv';
        filenameBusDataNet = strcat(folder_processedData, systemName1, "/
busDataNet", fileType);
        writetable(busData, filenameBusData);
        filenameBranchDataNet = strcat(folder_processedData, systemName1, "/
branchDataNet", fileType);
        writetable(branchDataNet, filenameBranchDataNet);
    end

    [ybus, BMatrix, E, PSpecified, QSpecified, V, delta, listOfPQBuses, lPV,
nPQ, nPV, listOfNonSlackBuses] = generateYBusForType2(useReducedLoadsForPowerFlow,
busData, busDataReducedLoads, branchData, MVAb);

    [yNet, ~, ~, ~, ~, EdgesNet] = ybusGenerator(busDataNet, branchDataNet);

    [yGen, BMatrixGen, EdgesGen, listOfResidualBuses] = constructYGen(yNet, lPV,
NNet, displayYNet, showInternalMatrices, displayYGen, saveYGenMatrices);

    [P, Q, V, delta] = solveForPowerFlow(PSpecified, QSpecified, V, delta,
ybus, BMatrix, E, nPQ, nPV, listOfPQBuses, listOfNonSlackBuses, numIterations,
toleranceLimit, powerFlowMethod, displayTables, printJacobians, printMismatches,
printPowerFlowConvergenceMessages);

    [busDataNetFaultOn, branchDataNetFaultOn, busDataNetPostFault,
branchDataNetPostFault] = modifySystemDuringAndPostFault(busDataNet, branchDataNet,
bus1, bus2, numLinesBeforeFault, relativeDistance);
    yNetFaultOn = ybusGenerator(busDataNetFaultOn, branchDataNetFaultOn);
    % saveAndDisplayYBus(yNetFaultOn, displayYBus, saveYGenMatrices,
folder_processedData, systemName1, 'yNet_FaultOn');
    yGenFaultOn = constructYGen(yNetFaultOn, lPV, NNet, displayYNet,
showInternalMatrices, displayYGen, saveYGenMatrices, 'FaultOn');
```

```
    yNetPostFault = ybusGenerator(busDataNetPostFault, branchDataNetPostFault);
    % saveAndDisplayYBus(yNetPostFault, displayYBus, saveYGenMatrices,
folder_processedData, systemName1, 'yNet_FaultOn');

    % PG_computedUsingYGen = array2table(double(subs(PG_Type3, theta,
theta0_Type3)), 'VariableNames', {'P_G'}, 'RowNames', namesGenBuses)
    yGenPostFault = constructYGen(yNetPostFault, lPV, NNet, displayYNet,
showInternalMatrices, displayYGen, saveYGenMatrices, 'PostFault');

else
    [ybus, BMatrix, ~, ~, ~, E] = ybusGenerator(busData, branchData);

    [PSpecified, QSpecified, V, delta, ...
        listOfPQBuses, lPV, nPQ, nPV, ...
        listOfNonSlackBuses] = initializeVectors(busData, MVAb);

    [P, Q, V, delta] = solveForPowerFlow(PSpecified, QSpecified, V, delta,
ybus, BMatrix, E, nPQ, nPV, listOfPQBuses, listOfNonSlackBuses, numIterations,
toleranceLimit, powerFlowMethod, displayTables, printJacobians, printMismatches,
printPowerFlowConvergenceMessages);

    [busDataFaultOn, branchDataFaultOn, busDataPostFault, branchDataPostFault]
= modifySystemDuringAndPostFault(busData, branchData, bus1, bus2,
numLinesBeforeFault, relativeDistance);
    ybusFaultOn = ybusGenerator(busDataFaultOn, branchDataFaultOn);
    saveAndDisplayYBus(ybusFaultOn, displayYBus, saveYBus, folder_processedData,
systemName1, 'ybus_FaultOn');
    ybusPostFault = ybusGenerator(busDataPostFault, branchDataPostFault);
    saveAndDisplayYBus(ybusPostFault, displayYBus, saveYBus, folder_processedData,
systemName1, 'ybus_PostFault');
end
```

yGenTable = 4×4 table

	1	2	3	4
1 1	3.2168 -11.5866i	1.6434 + 7.9619i	0.9492 + 1.0023i	1.3592 + 1.3569i
2 2	1.6434 + 7.9619i	1.3830 -10.5241i	0.6581 + 0.6164i	0.9393 + 0.8316i
3 3	0.9492 + 1.0023i	0.6581 + 0.6164i	1.6473 - 8.1778i	1.9706 + 4.2053i
4 4	1.3592 + 1.3569i	0.9393 + 0.8316i	1.9706 + 4.2053i	2.9049 - 9.8799i

Convergence using NRPF achieved in 4 iterations.

yGenTable = 4×4 table

	1	2	3	4
1 1	1.8186 -14.5133i	0.6173 + 6.1083i	0.2689 + 0.3831i	0.3889 + 0.5223i
2 2	0.6173 + 6.1083i	0.6414 -11.6926i	0.1903 + 0.2393i	0.2740 + 0.3253i
3 3	0.2689 + 0.3831i	0.1903 + 0.2393i	1.4033 - 8.2672i	1.6277 + 4.0905i
4 4	0.3889 + 0.5223i	0.2740 + 0.3253i	1.6277 + 4.0905i	2.4236 -10.0261i

yGenTable = 4×4 table

	1	2	3	4
1 1	3.2766 -11.3047i	1.6935 + 8.1434i	0.8894 + 0.8893i	1.2717 + 1.2020i
2 2	1.6935 + 8.1434i	1.4228 -10.4077i	0.6147 + 0.5451i	0.8762 + 0.7340i
3 3	0.8894 + 0.8893i	0.6147 + 0.5451i	1.7186 - 8.1324i	2.0714 + 4.2656i
4 4	1.2717 + 1.2020i	0.8762 + 0.7340i	2.0714 + 4.2656i	3.0475 - 9.8000i

```
listOfGenBuses = [1; lPV];

saveAndDisplayYBus(ybus, displayYBus, saveYBus, folder_processedData, systemName1);
```

Compare obtained snapshot values of V_i and δ_i against the ones given in the CDF file.

```
resultTable = displayPowerFlowResults(P, Q, V, delta, displayResults);
plotPowerFlowResults(showPlots, V, busData, systemName, powerFlowMethod, delta);
```

Post-Powerflow

```
P_i_Gen_Vals = P(listOfGenBuses);
Q_i_Gen_Vals = Q(listOfGenBuses);
S_i_Gen_Vals = P_i_Gen_Vals + 1i*Q_i_Gen_Vals;

PGVals = P_i_Gen_Vals + busData.PL(listOfGenBuses)/MVAb;
QGVals = Q_i_Gen_Vals + busData.QL(listOfGenBuses)/MVAb;
SGVals = PGVals + 1i*QGVals;

ViVals = V(listOfGenBuses);
delta_i_Vals = delta(listOfGenBuses);

[Vi_CartesiansReal, Vi_Cartesian_Imag] = pol2cart(delta_i_Vals, ViVals);
V_Gen_Cartesian = Vi_CartesiansReal + 1i*Vi_Cartesian_Imag;
IGenVals = conj(SGVals./V_Gen_Cartesian);

theta = sym('theta', [nPV+1, 1]);
theta(1) = delta_i_Vals(1);

omega = sym('omega', [nPV+1, 1]);

PG = sym('P_G', [nPV+1, 1]);
QG = sym('Q_G', [nPV+1, 1]);

Eq_primeSymbols = arrayfun(@(i) sprintf('E_prime_q%d', i), 1:nPV+1,
'UniformOutput', false);
Eq_prime = str2sym(Eq_primeSymbols');
Eq_prime(1) = ViVals(1);

Ed_primeSymbols = arrayfun(@(i) sprintf('E_prime_d%d', i), 1:nPV+1,
'UniformOutput', false);
```

```
Ed_prime = str2sym(Ed_primeSymbols');
Ed_prime(1) = 0;

Pm = sym('P_m', [nPV+1, 1]);

Id = sym('I_d', [nPV+1, 1]);
Iq = sym('I_q', [nPV+1, 1]);

V_i = sym('V_i', [N, 1]);
V_i(1) = V(1);

delta_i = sym('delta_i', [N, 1]);
delta_i(1) = delta(1);

P_i = sym('P_i', [N, 1]);
Q_i = sym('Q_i', [N, 1]);

P_L = sym('P_L', [N, 1]);
Q_L = sym('Q_L', [N, 1]);

Vq = sym('V_q', [nPV+1, 1]);
Vd = sym('V_d', [nPV+1, 1]);

P_C = sym('P_C', [nPV+1, 1]);
V_R = sym('V_R', [nPV+1, 1]);
Vref = sym('V_ref', [nPV+1, 1]);
```

Dynamic Initialization

```
ode_theta = diff(theta) == (omega-1)*w_s;
ode_thetaEquilibrium = 0 == rhs(ode_theta(lPV));

namesGenBuses = arrayfun(@(i) sprintf('Gen %d', i), 1:nPV+1, 'UniformOutput',
false);

if strcmp(modelType, 'Type3')
    EprimeCartesian = V_Gen_Cartesian + 1i*X_prime.*IGenVals;
    [thetaVals, E_primeVals] = cart2pol(real(EprimeCartesian),
imag(EprimeCartesian));
    thetaVals(1) = delta_i_Vals(1);
    E_primeVals(1) = ViVals(1);

    resultGen = array2table([E_primeVals, thetaVals], 'VariableNames', {'E''',
'theta'}, 'RowNames', namesGenBuses)

    ode_omegaType3 = diff(omega) == (1./(2*H)).*(Pm - PG - K_D.*(omega - 1));
    ode_omegaType3_Equilibrium = 0 == rhs(ode_omegaType3(lPV));
    ode_omegaType3_Init = subs(ode_omegaType3_Equilibrium, PG, PGVals);

    ode_Type3_Init = [ode_thetaEquilibrium; ode_omegaType3_Init];
```

```
    display(ode_Type3_Init);
    unknowns_Type3 = [omega(lPV); Pm(lPV)];

    init_params_Type3 = [ones(nPV, 1); PGVals(lPV)];

    unknowns0_Type3_Vals = solveAndExtract(ode_Type3_Init, unknowns_Type3,
init_params_Type3);
    omega0_Type3 = unknowns0_Type3_Vals(genSVIndices(nPV, 1));
    Pm0_Type3 = unknowns0_Type3_Vals(genSVIndices(nPV, 2));
    theta0_Type3 = thetaVals;
    x0_Type3 = [theta0_Type3; omega0_Type3];


elseif strcmp(modelType, 'Type2') || strcmp(modelType, 'Type1')

    ode_omegaType2 = diff(omega) == ( 1./(2*H) ) .* (Pm - PG - K_D.*(omega - 1));
    ode_omegaType2_Equilibrium = 0 == rhs(ode_omegaType2(lPV));
    ode_omegaType2_Init = subs(ode_omegaType2_Equilibrium, PG, PGVals);
    ode_omegaType1_Init = subs(ode_omegaType2_Equilibrium, PG, PGVals.*(0.5 +
0.5*ViVals.^2));

    eqn_Id_PowerFlow = Id == abs(IGenVals).*sin(theta - delta_i_Vals);
    eqn_Id_PowerFlow_Init = 0 == lhs(eqn_Id_PowerFlow(lPV)) -
rhs(eqn_Id_PowerFlow(lPV));
    eqn_Iq_PowerFlow = Iq == abs(IGenVals).*cos(theta - delta_i_Vals);
    eqn_Iq_PowerFlow_Init = 0 == lhs(eqn_Iq_PowerFlow(lPV)) -
rhs(eqn_Iq_PowerFlow(lPV));

    eqn_Vq = Vq == ViVals.*cos(theta - delta_i_Vals);
    eqn_Vq_Init = 0 == lhs(eqn_Vq(lPV)) - rhs(eqn_Vq(lPV));

    eqn_Vd = Vd == ViVals.*sin(theta - delta_i_Vals);
    eqn_Vd_Init = 0 == lhs(eqn_Vd(lPV)) - rhs(eqn_Vd(lPV));

    eqn_Id_KVL = Id == (Eq_prime - Vq)./Xd_prime;
    eqn_Id_KVL_Init = 0 == lhs(eqn_Id_KVL(lPV)) - rhs(eqn_Id_KVL(lPV));

    eqn_Iq_KVL = Iq == (Ed_prime - Vd)./(-Xq_prime);
    eqn_Iq_KVL_Init = 0 == lhs(eqn_Iq_KVL(lPV)) - rhs(eqn_Iq_KVL(lPV));

    V_R_dot = (1./T_A).*( -V_R + k_A.*(Vref - V_i(listOfGenBuses)) );
    ode_VRType2 = diff(V_R) == V_R_dot;
    ode_VRType2_Equilibrium = 0 == rhs(ode_VRType2(lPV));
    ode_VRType2_Init = subs( ode_VRType2_Equilibrium, V_i(lPV), ViVals(lPV) );

    ode_EqprimeType2 = diff(Eq_prime) == (1./Td0_prime) .* ( -Eq_prime - (Xd -
Xd_prime).*Id + V_R );
    ode_EqprimeType2_Equilibrium = 0 == rhs(ode_EqprimeType2(lPV));
    ode_EqprimeType2_Init = ode_EqprimeType2_Equilibrium;
```

```
    ode_EdprimeType2 = diff(Ed_prime) == (1./Tq0_prime) .* ( -Ed_prime + (Xq -
Xq_prime).*Iq );
    ode_EdprimeType2_Equilibrium = 0 == rhs(ode_EdprimeType2(lPV));
    ode_EdprimeType2_Init = ode_EdprimeType2_Equilibrium;

    P_m_dotType2 = (1./Tsg) .* ( -Pm + Ksg.*( P_C - (1./R).*(omega - 1) ) );
    ode_PmType2 = diff(Pm) == P_m_dotType2;
    ode_PmType2_Equilibrium = 0 == rhs(ode_PmType2(lPV));
    ode_PmType2_Init = ode_PmType2_Equilibrium;

    ode_Type2_Init = [ode_thetaEquilibrium; ode_omegaType2_Init; ...
        ode_EqprimeType2_Init; ode_EdprimeType2_Init; ...
        ode_VRType2_Init; ode_PmType2_Init; ...
        eqn_Id_PowerFlow_Init; eqn_Iq_PowerFlow_Init; ...
        % eqn_Eq_prime_Init; eqn_Ed_prime_Init; ...
        eqn_Vq_Init; eqn_Vd_Init; ...
        eqn_Id_KVL_Init; eqn_Iq_KVL_Init];

    display(ode_Type2_Init)

    unknowns_Type2 = ...
        [theta(lPV); omega(lPV); ...
        Eq_prime(lPV); Ed_prime(lPV); ...
        V_R(lPV); Pm(lPV); ...
        Vref(lPV); P_C(lPV); ...
        Vq(lPV); Vd(lPV); ...
        Iq(lPV); Id(lPV)];

    init_params_Type2 = ...
        [delta_i_Vals(lPV); ones(nPV, 1); ...
        1.1*ViVals(lPV); 0.1*ViVals(lPV); ...
        1.8*ones(nPV, 1); PGVals(lPV); ...
        ViVals(lPV);  PGVals(lPV)./Ksg(lPV); ...
        1.0*ones(nPV, 1); 1.0*ones(nPV, 1); ...
        real(IGenVals(lPV)); imag(IGenVals(lPV))];

    unknowns0_Type2_Vals = solveAndExtract(ode_Type2_Init, unknowns_Type2,
init_params_Type2);
    unknowns0_Type2_Gens_Vals = sortByGenerators(unknowns0_Type2_Vals, nPV);
    namesUnknowns_Type2 = string([theta(lPV); omega(lPV); Eq_prime(lPV);
Ed_prime(lPV); V_R(lPV); Pm(lPV); Vref(lPV); P_C(lPV); Vq(lPV); Vd(lPV); Iq(lPV);
Id(lPV)]);
    namesUnknowns_Type2_Gens = sortByGenerators(namesUnknowns_Type2, nPV);
    unknowns0_Type2_Vals_Table = array2table(unknowns0_Type2_Vals, 'RowNames',
namesUnknowns_Type2, 'VariableNames', {'Values'});
    unknowns0_Type2_Gens_Vals_Table = array2table(unknowns0_Type2_Gens_Vals,
'RowNames', namesUnknowns_Type2_Gens, 'VariableNames', {'Values'});
    % display(unknowns0_Type2_Vals_Table);
    display(unknowns0_Type2_Gens_Vals_Table);
```

```
    theta0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 1));
    omega0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 2));
    Eq_prime0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 3));
    Ed_prime0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 4));
    V_R0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 5));
    Pm0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 6));

    Vref0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 7));
    P_C0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 8));

    Vq0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 9));
    Vd0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 10));
    Iq0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 11));
    Id0_Type2 = unknowns0_Type2_Vals(genSVIndices(nPV, 12));

    x0_Type2 = [theta0_Type2; omega0_Type2; Eq_prime0_Type2; Ed_prime0_Type2;
V_R0_Type2; Pm0_Type2];
    x0_Type2_Gens = sortByGenerators(x0_Type2, nPV);
    namesSVs = string([theta(lPV); omega(lPV); Eq_prime(lPV); Ed_prime(lPV);
V_R(lPV); Pm(lPV)]);
    namesSVs_Gens = sortByGenerators(namesSVs, nPV);
    x0_Type2Table = array2table(x0_Type2, 'RowNames', namesSVs, 'VariableNames',
{'Values'});
    x0_Type2_Gens_Table = array2table(x0_Type2_Gens, 'RowNames', namesSVs_Gens,
'VariableNames', {'Values'});
    % display(x0_Type2Table);
    display(x0_Type2_Gens_Table);
else
    error('This statement shouldn''t be reached.');
end
```

resultGen = 4×2 table

	E'	theta
1 Gen 1	1.0300	0
2 Gen 2	1.0745	0.1465
3 Gen 3	1.1392	-0.0383
4 Gen 4	1.1045	-0.2068

ode_Type3_Init =

$$\begin{pmatrix} 0 = 376.9911\,\omega_2 - 376.9911 \\ 0 = 376.9911\,\omega_3 - 376.9911 \\ 0 = 376.9911\,\omega_4 - 376.9911 \\ 0 = 0.0085\,P_{m2} - 0.0171\,\omega_2 - 0.0427 \\ 0 = 0.0090\,P_{m3} - 0.0180\,\omega_3 - 0.0467 \\ 0 = 0.0090\,P_{m4} - 0.0180\,\omega_4 - 0.0450 \end{pmatrix}$$

Small Signal Stability Analysis

```
if strcmp(modelType, 'Type3')
    PG_Type3 = generateSymbolicPowerFlowEquations(nPV, PG, QG, yGen, E_primeVals,
EdgesGen, theta);
    ode_omegaType3_SS = subs(ode_omegaType3(lPV), PG, PG_Type3);
    ode_Type3_SS = [ode_thetaEquilibrium; ode_omegaType3_SS];
    display(ode_Type3_SS);
    x_Type3 = [theta(lPV); omega(lPV)];

    PG_computedUsingYGen = array2table(double(subs(PG_Type3, theta, theta0_Type3)),
'VariableNames', {'P_G'}, 'RowNames', namesGenBuses);
    display(PG_computedUsingYGen);

    J_Type3 = jacobian(rhs(ode_Type3_SS), x_Type3);
    Je_Type3 = double(subs(J_Type3, [theta; omega(lPV)], x0_Type3));

    [VType3, DType3, WType3] = eig(Je_Type3);
    lambdas_Type3 = diag(DType3);
    display(lambdas_Type3);
    W_T_Type3 = WType3';

    for i = 1:size(VType3, 1)
        v_Type3_i = VType3(:, i);
        w_T_Type3_i = W_T_Type3(i, :);
        pMatrix_Type3 = v_Type3_i*w_T_Type3_i;
        pfactors_Type3 = abs(diag(pMatrix_Type3));
        pfactors_Type3_Normalized = pfactors_Type3./max(pfactors_Type3);
        fprintf('For mode λ = %f + 1i* %f', real(lambdas_Type3(i)),
imag(lambdas_Type3(i)));
        display(pfactors_Type3_Normalized);
    end

elseif strcmp(modelType, 'Type2')

    E_prime_Type2 = sqrt(Eq_prime.^2 + Ed_prime.^2);
    E_prime_Type2(1) = ViVals(1);

    gamma_Type2 = atan(Eq_prime./Ed_prime) + theta - pi/2;
    gamma_Type2(1) = delta_i_Vals(1);

    PG_Type2 = generateSymbolicPowerFlowEquations(nPV, PG, QG, yGen, E_prime_Type2,
EdgesGen, gamma_Type2);
    % display(PG_Type2)
    % PG_computedUsingYGen = array2table(double(subs(PG_Type2, [theta;
Eq_prime; Ed_prime], [ [0; theta0_Type2]; [ViVals(1); Eq_prime0_Type2]; [0;
Ed_prime0_Type2] ])), 'VariableNames', {'P_G'}, 'RowNames', namesGenBuses);
    % display(PG_computedUsingYGen)
```

```
    [Id_Type2, Iq_Type2] = generate_dq_CurrentsFrom_dqVoltages(nPV, Id, Iq,
EdgesGen, yGen, theta, Eq_prime, Ed_prime);

    Id_computedUsingYGen = array2table(double(subs(Id_Type2, [theta(lPV);
Eq_prime(lPV); Ed_prime(lPV)], [ theta0_Type2; Eq_prime0_Type2;
Ed_prime0_Type2] )), 'VariableNames', {'I_d'}, 'RowNames', namesGenBuses);
    Iq_computedUsingYGen = array2table(double(subs(Iq_Type2, [theta(lPV);
Eq_prime(lPV); Ed_prime(lPV)], [ theta0_Type2; Eq_prime0_Type2;
Ed_prime0_Type2] )), 'VariableNames', {'I_q'}, 'RowNames', namesGenBuses);
    display(Iq_computedUsingYGen(lPV, :));
    display(Id_computedUsingYGen(lPV, :));
    display(Iq0_Type2);
    display(Id0_Type2);

    Vq_Type2 = V_i(listOfGenBuses) .* cos( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );
    Vd_Type2 = V_i(listOfGenBuses) .* sin( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );

    % V_i_Type2 = subs( (Ed_prime + Iq.*Xq_prime)./sin(theta - delta_i_Vals), Iq,
Iq_Type2);
    V_i_Type2 = subs( (Eq_prime - Id.*Xd_prime)./cos(theta - delta_i_Vals), Id,
Id_Type2);
    V_i_ComputedUsingFormula = array2table(double(subs(V_i_Type2, [theta(lPV);
Eq_prime(lPV); Ed_prime(lPV)], [theta0_Type2; Eq_prime0_Type2; Ed_prime0_Type2] )),
'VariableNames', {'V_i'}, 'RowNames', namesGenBuses);
    display(V_i_ComputedUsingFormula(lPV, :));
    display(ViVals(lPV));

    % PG_Type2 = subs(Vd.*Id + Vq.*Iq, [Vd, Vq, Id, Iq], [Vd_Type2, Vq_Type2,
Id_Type2, Iq_Type2]);
    % PG_Type2 = subs(PG_Type2, V_i(listOfGenBuses), V_i_Type2);
    % PG_computedUsingFormula = array2table(double(subs(PG_Type2, [theta;
Eq_prime; Ed_prime], [ [0; theta0_Type2]; [ViVals(1); Eq_prime0_Type2]; [0;
Ed_prime0_Type2] ])), 'VariableNames', {'P_G'}, 'RowNames', namesGenBuses);
    % display(PG_computedUsingFormula);

    % ode_thetaType2_SS = ode_thetaEquilibrium;
    ode_omegaType2_SS = subs(ode_omegaType2_Equilibrium, PG, PG_Type2);
    ode_EqprimeType2_SS = subs(ode_EqprimeType2_Equilibrium, Id, Id_Type2);
    % display(ode_EqprimeType2_SS);
    ode_EdprimeType2_SS = subs(ode_EdprimeType2_Equilibrium, Iq, Iq_Type2);
    % display(ode_EdprimeType2_SS);
    ode_VRType2_SS = subs(ode_VRType2_Equilibrium, [Vref(lPV); V_i(listOfGenBuses);
Iq], [Vref0_Type2; V_i_Type2; Iq_Type2]);
    % display(ode_VRType2_SS);
    ode_PmType2_SS = subs(ode_PmType2_Equilibrium, P_C(lPV), P_C0_Type2);
    % display(ode_PmType2_SS);
```

```
    ode_Type2_SS = [ode_thetaEquilibrium; ode_omegaType2_SS; ode_EqprimeType2_SS;
ode_EdprimeType2_SS; ode_VRType2_SS; ode_PmType2_SS];
    % display(ode_Type2_SS)

    x_Type2 = [theta(lPV); omega(lPV); Eq_prime(lPV); Ed_prime(lPV); V_R(lPV);
Pm(lPV)];
    x_Type2_Gens = sortByGenerators(x_Type2, nPV);
    % display(x_Type2_Gens);
    ode_Type2_SS_Gens = sortByGenerators(ode_Type2_SS, nPV);
    display(ode_Type2_SS_Gens)
    J_Type2 = jacobian(rhs(ode_Type2_SS), x_Type2);
    x0_Type2_Gens = sortByGenerators(x0_Type2, nPV);
    J_Type2_Gens = jacobian(rhs(ode_Type2_SS_Gens), x_Type2_Gens);
    % display(J_Type2)
    format shortG
    Je_Type2 = double(subs(J_Type2, x_Type2, x0_Type2));
    Je_Type2_Gens = double(subs(J_Type2_Gens, x_Type2_Gens, x0_Type2_Gens));
    % display(Je_Type2);
    Je_Type2_Table = array2table(Je_Type2, 'VariableNames', namesSVs, 'RowNames',
namesSVs);
    % display(Je_Type2_Table)
    format default
    % display(Je_Type2_Gens)
    Je_Type2_Gens_Table = array2table(Je_Type2_Gens, 'VariableNames',
namesSVs_Gens, 'RowNames', namesSVs_Gens);
    display(Je_Type2_Gens_Table)
    [VType2, DType2, WType2] = eig(Je_Type2);
    lambdasType2 = diag(DType2);
    [VType2_Gens, DType2_Gens, WType2_Gens] = eig(Je_Type2_Gens);
    lambdasType2_Gens = diag(DType2_Gens);
    format shortG
    display(lambdasType2);
    display(lambdasType2_Gens)
    format default
    W_T_Type2 = WType2';
else

    Vq_Type1 = V_i(listOfGenBuses) .* cos( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );
    Vd_Type1 = V_i(listOfGenBuses) .* sin( theta(listOfGenBuses) -
delta_i_Vals(listOfGenBuses) );

    Id_Type1 = subs( (Eq_prime - Vq)./Xd_prime, Vq, Vq_Type1);
    Iq_Type1 = subs( (Ed_prime - Vd)./(-Xq_prime), Vd, Vd_Type1);

    PG_Type1 = subs(Vd.*Id + Vq.*Iq, [Vd, Vq, Id, Iq], [Vd_Type1, Vq_Type1,
Id_Type1, Iq_Type1]);
    QG_Type1 = subs(Vq.*Id - Vd.*Iq, [Vd, Vq, Id, Iq], [Vd_Type1, Vq_Type1,
Id_Type1, Iq_Type1]);
```

```
    ode_omegaType1_SS = subs(ode_omegaType2_Equilibrium, PG, PG_Type1);
    ode_EqprimeType1_SS = subs(ode_EqprimeType2_Equilibrium, Id, Id_Type1);
    ode_EdprimeType1_SS = subs(ode_EdprimeType2_Equilibrium, Iq, Iq_Type1);
    ode_VRType1_SS = subs(ode_VRType2_Equilibrium, Vref(lPV), Vref0_Type2);
    ode_PmType1_SS = subs(ode_PmType2_Equilibrium, P_C(lPV), P_C0_Type2);

    ode_Type1_SS = [ode_thetaEquilibrium; ode_omegaType1_SS; ode_EqprimeType1_SS;
ode_EdprimeType1_SS; ode_VRType1_SS; ode_PmType1_SS];
    % display(ode_Type1_SS)

    IL = sym('I_L', [1, 1]);
    delta_IL1 = sym('I_L1', [1, 1]);
    delta_IL2 = sym('I_L2', [1, 1]);
    V_PSS = sym('V_PSS', [4, 1]);

    ode_deltaIL1_Equilibrium = 0 == (1./Twpss) .* ( -delta_IL1 + Kwpss*Kcomp.*IL );
    ode_deltaIL2_SS = subs( ode_deltaIL1_Equilibrium, IL, -ybus(7, 8).*( V_i(7) -
V_i(8) ) );
    % ode_deltaIL2 = 0 == (1./Tbcomp1) .* ( -delta_IL2 + delta_IL1 +
Tacomp1*diff(delta_IL1) );
    ode_deltaIL2_Equilibrium = 0 == (1./Tbcomp1) .* ( -delta_IL2 + delta_IL1  +
Tacomp1 .* rhs(ode_deltaIL1_Equilibrium) );
    % ode_deltaIL2_SS = subs(ode_deltaIL2_Equilibrium, )
    % ode_VPSS = 0 == (1./Tbcomp2) .* ( -V_PSS + delta_IL2 +
Tacomp2.*diff(delta_IL2) );
    ode_VPSS_Equilibrium = 0 == (1./Tbcomp2) .* ( -V_PSS + delta_IL2 + Tacomp2 .*
rhs(ode_deltaIL2_Equilibrium) );


    x_Type1 = x_Type2;
    x0_Type1 = x0_Type2;
    % x_Type1_Gens = sortByGenerators(x_Type1, nPV);
    % x0_Type1_Gens = sortByGenerators(x0_Type1, nPV);

    y_Type1 = [delta_i(2:N); V_i(2:N)];
    % y_Type1_Gens = sortByGenerators(y_Type1, N-1);
    y0_Type1 = [delta(2:N); V(2:N)];
    % y0_Type1_Gens = sortByGenerators(y0_Type1, N-1);

    % ode_Type1_SS_Gens = sortByGenerators(ode_Type1_SS, nPV);

    A_Type1 = jacobian(rhs(ode_Type1_SS), x_Type1);
    % display(A_Type1)
    format shortG
    Ae_Type1 = double( subs(A_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]) );
    % display(Ae_Type1);
    % Ae_Type1_Table = array2table(Ae_Type1, 'VariableNames', namesSVs, 'RowNames',
namesSVs);
    % display(Ae_Type1_Table)
    format default
```

```
    B_Type1 = jacobian(rhs(ode_Type1_SS), y_Type1);
    Be_Type1 = double(subs(B_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]));
    % display(Be_Type1);
    % Be_Type1_Table = array2table(Be_Type1, 'VariableNames', namesSVs, 'RowNames',
namesSVs);

    [P_i_Type1, Q_i_Type1] = generateSymbolicPowerFlowEquations(N-1, P_i, Q_i,
ybus, V_i, E, delta_i);

    P_L_Type1 = busData.PL/MVAb .* (0.5 + 0.5*V_i.^2);
    Q_L_Type1 = busData.QL/MVAb .* (0.5 + 0.5*V_i.^2);

    g = sym('g', [2*(N-1), 1]);
    % g(lPV-1) = subs( PG(lPV) - busData.PL(lPV)/MVAb - P_i(lPV), [PG; P_i],
[PG_Type1; P_i_Type1]);
    g(lPV-1) = subs( PG(lPV) - P_L(lPV) - P_i(lPV), [PG; P_L; P_i], [PG_Type1;
P_L_Type1; P_i_Type1]);

    % g(lPQ-1) = subs( -busData.PL(lPQ)/MVAb - P_i(lPQ), P_i, P_i_Type1);
    g(lPQ-1) = subs( -P_L(lPQ) - P_i(lPQ), [P_L; P_i], [P_L_Type1; P_i_Type1]);

    % g(lPV-1+N-1) = subs(QG(lPV) - busData.QL(lPV)/MVAb - Q_i(lPV), [QG; Q_i],
[QG_Type1; Q_i_Type1]);
    g(lPV-1+N-1) = subs( QG(lPV) - Q_L(lPV) - Q_i(lPV), [QG; Q_L; Q_i], [QG_Type1;
Q_L_Type1; Q_i_Type1]);

    % g(lPQ-1+N-1) = subs( -busData.QL(lPQ)/MVAb - Q_i(lPQ), Q_i, Q_i_Type1);
    g(lPQ-1+N-1) = subs( -Q_L(lPQ) - Q_i(lPQ), [Q_L; Q_i], [Q_L_Type1; Q_i_Type1]);

    C_Type1 = jacobian(g, x_Type1);
    Ce_Type1 = double(subs(C_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]));

    D_Type1 = jacobian(g, y_Type1);
    De_Type1 = double(subs(D_Type1, [x_Type1; y_Type1], [x0_Type1; y0_Type1]));

    Je_Type1 = Ae_Type1 - Be_Type1/De_Type1 * Ce_Type1;
    % display(Je_Type1);

    [V_Type1, D_Type1, W_Type1] = eig(Je_Type1);
    lambdas_Type1 = diag(D_Type1);
    display(lambdas_Type1);

    W_T_Type1 = W_Type1';

    for i = 1:size(V_Type1, 1)
        v_Type1_i = V_Type1(:, i);
        w_T_Type1_i = W_T_Type1(i, :);
        pMatrix_Type1 = v_Type1_i*w_T_Type1_i;
        pfactors_Type1 = diag(pMatrix_Type1);
```

```
        pfactors_Type1_Normalized = pfactors_Type1./max(abs(pfactors_Type1));
        fprintf('For mode λ = %f + 1i* %f', real(lambdas_Type1(i)),
imag(lambdas_Type1(i)));
        display(pfactors_Type1_Normalized);
    end

end
```

ode_Type3_SS =

$$\left(\begin{array}{c} 0 = 376.9911\ \omega_2 - 376.9911 \\ 0 = 376.9911\ \omega_3 - 376.9911 \\ 0 = 376.9911\ \omega_4 - 376.9911 \\ 0 = 0.0085\ P_{m2} - 0.0171\ \omega_2 - 0.0127\cos(\theta_4 - \theta_2 + 0.7246) - 0.0094\cos(\theta_3 - \theta_2 + 0.7527) - 0.0769\cos \\ 0 = 0.0090\ P_{m3} - 0.0180\ \omega_3 - 0.0526\cos(\theta_4 - \theta_3 + 1.1326) - 0.0099\cos(\theta_2 - \theta_3 + 0.7527) - 0.0146\cos \\ 0 = 0.0090\ P_{m4} - 0.0180\ \omega_4 - 0.0526\cos(\theta_3 - \theta_4 + 1.1326) - 0.0134\cos(\theta_2 - \theta_4 + 0.7246) - 0.0197\cos \end{array}\right.$$

PG_computedUsingYGen = 4×1 table

	P_G
1 Gen 1	6.9143
2 Gen 2	6.9998
3 Gen 3	7.1904
4 Gen 4	7.0018

```
lambdas_Type3 = 6×1 complex
  -0.0090 + 6.6736i
  -0.0090 - 6.6736i
  -0.0086 + 5.6071i
  -0.0086 - 5.6071i
  -0.0090 + 2.8447i
  -0.0090 - 2.8447i
For mode λ = -0.008999 + 1i* 6.673622
pfactors_Type3_Normalized = 6×1
    0.0024
    0.6885
    1.0000
    0.0024
    0.6885
    1.0000
For mode λ = -0.008999 + 1i* -6.673622
pfactors_Type3_Normalized = 6×1
    0.0024
    0.6885
    1.0000
    0.0024
    0.6885
    1.0000
For mode λ = -0.008562 + 1i* 5.607053
pfactors_Type3_Normalized = 6×1
    1.0000
    0.0227
    0.0044
    1.0000
    0.0227
    0.0044
For mode λ = -0.008562 + 1i* -5.607053
pfactors_Type3_Normalized = 6×1
```

```
    1.0000
    0.0227
    0.0044
    1.0000
    0.0227
    0.0044
For mode λ = -0.008989 + 1i* 2.844733
pfactors_Type3_Normalized = 6×1
    0.0437
    1.0000
    0.7086
    0.0437
    1.0000
    0.7086
For mode λ = -0.008989 + 1i* -2.844733
pfactors_Type3_Normalized = 6×1
    0.0437
    1.0000
    0.7086
    0.0437
    1.0000
    0.7086
```

Transient Stability Analysis

Need to fix it to take modelType as argument (currently assumes that only Type 3 is running).

```
if runTransientSimulation
    transientSimulationScript(clearingCycles, saveTransientRunValues,
saveTransientRunPlots, modelType, nPV, PG, QG, yGenFaultOn, E_primeVals, EdgesGen,
theta, yGenPostFault, ode_omegaType3, lPV, Pm, PGVals, ode_thetaEquilibrium,
numStepsFaultOn, numStepsPostFault, x0_Type3, omega, h, folder_processedData,
systemName1);
end
```

Have a nice day!

In case you encounter a Java Heap Memory error, delete the above gif, or go to `Preferences->General->Java Heap Memory` and increase the allocated size.

```
function [P, Q, V, delta, J11] = solveForPowerFlow(PSpecified, QSpecified, ...
    V, delta, ...
    ybus, BMatrix, E, nPQ, nPV, ...
    listOfPQBuses, listOfNonSlackBuses, ...
    numIterations, toleranceLimit, powerFlowMethod, ...
    displayTables, printJacobians, printMismatches, ...
    printPowerFlowConvergenceMessages)

    N = size(ybus, 1);
    if strcmp(powerFlowMethod, 'NRPF')
        for itr = 1:numIterations
            desiredOutput = 'both';
            [PMismatch, QMismatch, P, Q] = ...
                computeMismatches(PSpecified, QSpecified, ...
                V, delta, ybus, BMatrix, E, ...
                listOfPQBuses, listOfNonSlackBuses, powerFlowMethod);

            mismatch = [PMismatch; QMismatch];
            if displayTables && printMismatches
                fprintf("Iteration Number %i Mismatches:\n", itr);
                disp(mismatch)
            end
            [J, JTable] = constructJacobian(P, Q, ...
                V, delta, N, ybus, BMatrix, E, ...
                nPQ, nPV, listOfPQBuses, listOfNonSlackBuses, powerFlowMethod,
desiredOutput);

            if displayTables && printJacobians
                fprintf("Iteration Number %i Jacobian:\n", itr);
                JTable %#ok<NOPRT>
            end

            correction = solveUsingLU(J, mismatch, 2*nPQ+nPV);

            delta = [delta(1); delta(listOfNonSlackBuses) + correction(1:nPQ+nPV)];
            V(listOfPQBuses) = V(listOfPQBuses).*( ones(nPQ, 1) +
correction(nPQ+nPV+1:end) );

            if mean(abs(correction)) < toleranceLimit
                fprintf("Convergence using %s achieved in %i iterations.\n",
powerFlowMethod, itr);
                break;
            else
                if printPowerFlowConvergenceMessages
                    fprintf("Convergence still not achieved in %i iterations as %f
is greater than %f\n", itr, mean(abs(correction)), toleranceLimit);
                end
            end
        end
```

```
    elseif strcmp(powerFlowMethod, 'Decoupled NRPF') || strcmp(powerFlowMethod,
'Fast Decoupled NRPF')
        for itr = 1:numIterations
            desiredOutput = 'P';
            [PMismatch, ~, P, Q] = ...
                computeMismatches(PSpecified, QSpecified, ...
                V, delta, ybus, BMatrix, E, ...
                listOfPQBuses, listOfNonSlackBuses, powerFlowMethod);

            if displayTables && printMismatches
                fprintf("Iteration Number %i P Mismatches:\n", itr);
                disp(PMismatch)
            end
            [J11, J11Table] = constructJacobian(P, Q, ...
                V, delta, N, ybus, BMatrix, E, ...
                nPQ, nPV, listOfPQBuses, listOfNonSlackBuses, powerFlowMethod,
desiredOutput);

            if displayTables && printJacobians
                fprintf("Iteration Number %i Jacobian J11:\n", itr);
                J11Table %#ok<NOPRT>
            end

            correctionDelta = solveUsingLU(J11, PMismatch, nPQ+nPV);

            delta = [delta(1); delta(listOfNonSlackBuses) + correctionDelta];

            desiredOutput = 'Q';
            [~, QMismatch, P, Q] = ...
                computeMismatches(PSpecified, QSpecified, ...
                V, delta, ybus, BMatrix, E, ...
                listOfPQBuses, listOfNonSlackBuses, powerFlowMethod);

            if displayTables && printMismatches
                fprintf("Iteration Number %i Q Mismatches:\n", itr);
                disp(QMismatch)
            end
            [J22, J22Table] = constructJacobian(P, Q, ...
                V, delta, N, ybus, BMatrix, E, ...
                nPQ, nPV, listOfPQBuses, listOfNonSlackBuses, powerFlowMethod,
desiredOutput);

            if displayTables && printJacobians
                fprintf("Iteration Number %i Jacobians:\n", itr);
                J22Table %#ok<NOPRT>
            end

            correctionDeltaVByV = solveUsingLU(J22, QMismatch, nPQ);
```

```
            V(listOfPQBuses) = V(listOfPQBuses).*( ones(nPQ, 1) +
correctionDeltaVByV );

            correction = [correctionDelta; correctionDeltaVByV];
            if mean(abs(correction)) < toleranceLimit
                fprintf("Convergence using %s achieved in %i iterations.\n",
powerFlowMethod, itr);
                break;
            else
                if printPowerFlowConvergenceMessages
                    fprintf("Convergence still not achieved in %i iterations as %f
is greater than %f\n", itr, mean(abs(correction)), toleranceLimit);
                end
            end
        end
    end
end
```

```matlab
function [ybus, BMatrix, E, PSpecified, QSpecified, V, delta,
listOfPQBuses, listOfPVBuses, nPQ, nPV, listOfNonSlackBuses] =
generateYBusForType2(useReducedLoadsForPowerFlow, busData, busDataReducedLoads,
branchData, MVAb)
    if useReducedLoadsForPowerFlow
        [ybus, BMatrix, ~, ~, ~, E] = ybusGenerator(busDataReducedLoads,
branchData); %#ok<*UNRCH>


        [PSpecified, QSpecified, V, delta, ...
            listOfPQBuses, listOfPVBuses, nPQ, nPV, ...
            listOfNonSlackBuses] = initializeVectors(busDataReducedLoads, MVAb);
    else

        [ybus, BMatrix, ~, ~, ~, E] = ybusGenerator(busData, branchData);

        [PSpecified, QSpecified, V, delta, ...
            listOfPQBuses, listOfPVBuses, nPQ, nPV, ...
            listOfNonSlackBuses] = initializeVectors(busData, MVAb);
    end
end
```

```
function [yGen, BMatrixGen, EGen, listOfResidualBuses] = constructYGen(yNet,
listOfPVBuses, NNet, displayYNet, showInternalMatrices, displayYGen,
saveYGenMatrices, varargin)

    if ~isempty(varargin)
        nametag = strcat("_", varargin{1});
    else
        nametag = "";
    end

    listOfGenBuses = [1; listOfPVBuses];
    yNetTable = array2table(yNet, VariableNames=[string(1:NNet)],
RowNames=[string(1:NNet)]);

    if displayYNet
        format shortG
        disp(sparse(yNet));
        format default
    end

    if saveYGenMatrices
        fileType = ".csv";
        filenameYNet = strcat("processedData\ieee11\yNet", nametag, fileType);
        writetable(yNetTable, filenameYNet);
    end

    listOfAllBuses = ones(NNet, 1);
    listOfAllBuses(listOfGenBuses) = 0;
    listOfResidualBuses = find(listOfAllBuses);

    ygg = yNet(listOfGenBuses, listOfGenBuses);
    ygr = yNet(listOfGenBuses, listOfResidualBuses);
    yrg = yNet(listOfResidualBuses, listOfGenBuses);
    yrr = yNet(listOfResidualBuses, listOfResidualBuses);

    yggTable = array2table(ygg, 'VariableNames', [string(listOfGenBuses)],
'RowNames', [string(listOfGenBuses)]);
    ygrTable = array2table(ygr, 'VariableNames', [string(listOfResidualBuses)],
'RowNames', [string(listOfGenBuses)]);
    yrgTable = array2table(yrg, 'VariableNames', [string(listOfGenBuses)],
'RowNames', [string(listOfResidualBuses)]);
    yrrTable = array2table(yrr, 'VariableNames', [string(listOfResidualBuses)],
'RowNames', [string(listOfResidualBuses)]);

    if showInternalMatrices
        display(yggTable);
        display(ygrTable);
        display(yrgTable);
        display(yrrTable);
    end
```

```
    if saveYGenMatrices
        fileType = ".csv";
        filenameYgg = strcat("processedData\ieee11\Ygg", nametag, fileType);
        writetable(yggTable, filenameYgg);
        filenameYgr = strcat("processedData\ieee11\Ygr", nametag, fileType);
        writetable(ygrTable, filenameYgr);
        filenameYrg = strcat("processedData\ieee11\Yrg", nametag ,fileType);
        writetable(yrgTable, filenameYrg);
        filenameYrr = strcat("processedData\ieee11\Yrr", nametag, fileType);
        writetable(yrrTable, filenameYrr);
    end

    yGen = ygg - ygr/yrr * yrg;
    BMatrixGen = -imag(yGen);
    nPV = length(listOfPVBuses);
    EGen = cell(nPV + 1, 1);
    for i = 1: nPV+1
        EGen{i} = [1:i-1 i+1:nPV+1];
    end

    yGenTable = array2table(yGen, VariableNames=[string(listOfGenBuses)],
RowNames=[string(listOfGenBuses)]);

    if displayYGen
        display(yGenTable);
    end

    if saveYGenMatrices
        fileType = ".csv";
        filenameYGen = strcat("processedData\ieee11\yGen", nametag, fileType);
        writetable(yGenTable, filenameYGen);
    end

end
```

```
function [PG, QG] = generateSymbolicPowerFlowEquations(nPV, PG, QG, yGen,
E_primeVals, EdgesGen, theta)
    for i = 1:nPV+1
        PG(i) = real( yGen(i, i) ) * E_primeVals(i)^2;
        QG(i) = -imag( yGen(i, i) ) * E_primeVals(i)^2;
        for k = EdgesGen{i}
            PG(i) = PG(i) + abs(yGen(i, k)) * E_primeVals(i) * E_primeVals(k) *
cos( angle(yGen(i, k)) + theta(k) - theta(i) );
            QG(i) = QG(i) - abs(yGen(i, k)) * E_primeVals(i) * E_primeVals(k) *
sin( angle(yGen(i, k)) + theta(k) - theta(i) );
        end
    end
end
```

```
function [Id, Iq] = generate_dq_CurrentsFrom_dqVoltages(nPV, Id, Iq, EdgesGen,
yGen, theta, Eq_prime, Ed_prime)
    for i = 1:nPV+1
        Id(i) = 0;
        Iq(i) = 0;
        for k = [i, EdgesGen{i}]
            Id(i) = Id(i) + abs(yGen(i, k)) * sqrt( Eq_prime(k)^2 +
Ed_prime(k)^2 ) .* cos( angle(yGen(i, k)) + atan(Eq_prime(k)/Ed_prime(k)) +
theta(k) - theta(i) );
            Iq(i) = Iq(i) + abs(yGen(i, k)) * sqrt( Eq_prime(k)^2 +
Ed_prime(k)^2 ) .* sin( angle(yGen(i, k)) + atan(Eq_prime(k)/Ed_prime(k)) +
theta(k) - theta(i) );
        end
    end
end
```

```
function transientSimulationScript(clearingCycles, saveTransientRunValues,
saveTransientRunPlots, modelType, nPV, PG, QG, yGenFaultOn, E_primeVals, EdgesGen,
theta, yGenPostFault, ode_omegaType3, lPV, Pm, PGVals, ode_thetaEquilibrium,
numStepsFaultOn, numStepsPostFault, x0_Type3, omega, h, folder_processedData,
systemName1)
    if clearingCycles ~= 3
        saveTransientRunValues = false;
        saveTransientRunPlots = false;
    end

    transientRunStart = tic;

    if strcmp(modelType, 'Type3')
        % error('Make sure to save results first! Only if clearing time is 3
cycles.')
        PG_FaultOn_Type3 = generateSymbolicPowerFlowEquations(nPV, PG, QG,
yGenFaultOn, E_primeVals, EdgesGen, theta);
        PG_PostFault_Type3 = generateSymbolicPowerFlowEquations(nPV, PG, QG,
yGenPostFault, E_primeVals, EdgesGen, theta);

        ode_omegaType3_TransientFaultOn = subs(ode_omegaType3(lPV), [PG, Pm],
[PG_FaultOn_Type3, PGVals]);
        ode_Type3_TransientFaultOn = [ode_thetaEquilibrium;
ode_omegaType3_TransientFaultOn];
        ode_Type3_TransientFaultOn = subs(ode_Type3_TransientFaultOn, theta(1), 0);
        display(ode_Type3_TransientFaultOn);

        ode_omegaType3_TransientPostFault = subs(ode_omegaType3(lPV), [PG, Pm],
[PG_PostFault_Type3, PGVals]);
        ode_Type3_TransientPostFault = [ode_thetaEquilibrium;
ode_omegaType3_TransientPostFault];
        ode_Type3_TransientPostFault = subs(ode_Type3_TransientPostFault, theta(1),
0);
        display(ode_Type3_TransientPostFault);

        thetaTransientVals = zeros(nPV, numStepsFaultOn + numStepsPostFault);
        omegaTransientVals = zeros(nPV, numStepsFaultOn + numStepsPostFault);
        xTransientVals = [thetaTransientVals; omegaTransientVals];

        xTransientVals(:, 1) = [x0_Type3(lPV); x0_Type3(nPV+lPV)];
        x_Type3 = [theta(lPV); omega(lPV)];

        fxTransientVals = zeros(size(xTransientVals));

        for t = 2:numStepsFaultOn
            fxTransientVals(:, t-1) = double( subs(rhs(ode_Type3_TransientFaultOn),
x_Type3, xTransientVals(:, t-1) ) ) ;
            xTransientVals(:, t) = xTransientVals(:, t-1) + h*fxTransientVals(:,
t-1);
        end
```

```
        for t = numStepsFaultOn+1:numStepsFaultOn+numStepsPostFault
            fxTransientVals(:, t-1) =
double( subs(rhs(ode_Type3_TransientPostFault), x_Type3, xTransientVals(:, t-1) ) );
            xTransientVals(:, t) = xTransientVals(:, t-1) + h*fxTransientVals(:,
t-1);
        end

    elseif strcmp(modelType, 'Type2')
        error('Check Transient Stability for Type 2 again.\n')
        PG_FaultOn_Type2 = generateSymbolicPowerFlowEquations(nPV, PG, QG,
yGenFaultOn, E_prime_Type2, EdgesGen, theta);
        PG_PostFault_Type2 = generateSymbolicPowerFlowEquations(nPV, PG, QG,
yGenPostFault, E_prime_Type2, EdgesGen, theta);

        ode_omegaType2_TransientFaultOn = subs(ode_omegaType2(lPV), [PG, Pm],
[PG_FaultOn_Type2, PGVals]);

        ode_Type2_TransientFaultOn = [ode_thetaEquilibrium;
ode_omegaType2_TransientFaultOn; ode_EqprimeType2_SS; ode_EdprimeType2_SS;
ode_VRType2_SS; ode_PmType2_SS];
        ode_Type2_TransientFaultOn = subs(ode_Type2_TransientFaultOn, [theta(1),
Eq_prime(1), Ed_prime(1)],  [0, 1, 0]);
        display(ode_Type2_TransientFaultOn);

        ode_omegaType2_TransientPostFault = subs(ode_omegaType2(lPV), [PG, Pm],
[PG_PostFault_Type2, PGVals]);
        ode_Type2_TransientPostFault = [ode_thetaEquilibrium;
ode_omegaType2_TransientPostFault; ode_EqprimeType2_SS; ode_EdprimeType2_SS;
ode_VRType2_SS; ode_PmType2_SS];

        ode_Type2_TransientPostFault = subs(ode_Type2_TransientPostFault,
[theta(1), Eq_prime(1), Ed_prime(1)],  [0, 1, 0]);
        display(ode_Type2_TransientPostFault);

        xTransientVals = zeros(nPV*6, numStepsFaultOn + numStepsPostFault);

        xTransientVals(:, 1) = x0_Type2;

        fxTransientVals = zeros(size(xTransientVals));

        for t = 2:numStepsFaultOn
            fxTransientVals(:, t-1) = double( subs(rhs(ode_Type2_TransientFaultOn),
x_Type2, xTransientVals(:, t-1) ) ) ;
            xTransientVals(:, t) = xTransientVals(:, t-1) + h*fxTransientVals(:,
t-1);
        end

        for t = numStepsFaultOn+1:numStepsFaultOn+numStepsPostFault
```

```
            fxTransientVals(:, t-1) =
double( subs(rhs(ode_Type2_TransientPostFault), x_Type2, xTransientVals(:, t-1) ) );
            xTransientVals(:, t) = xTransientVals(:, t-1) + h*fxTransientVals(:,
t-1);
        end

    else
        error('Type 1 NOT modeled for TS!');
    end

    if saveTransientRunValues
        fileType = '.csv';
        filenameTransientRun = strcat(folder_processedData, systemName1, "/
xTransientRun_", modelType, fileType);
        save(filenameTransientRun, 'xTransientVals');
    end

    thetaValsTransient = xTransientVals(1:nPV, 1:t);

    figTheta = figure('Name', 'Transient Simmulation: Machine Rotor Angles');
    % figure('Name', 'Transient Simulation: Machine Rotor Angles');
    plot(1:t, thetaValsTransient);
    xlabel('Time [ms]');
    ylabel('$\theta$ [rad]');
    legend({'Gen02', 'Gen03', 'Gen04'});
    title('Transient Simulation: Machine Rotor Angles')

    omegaValsTransient = xTransientVals(nPV+1:2*nPV, 1:t);
    figOmega = figure('Name', 'Transient Simmulation: Machine Rotor Speeds');
    % figure('Name', 'Transient Simmulation: Machine Rotor Speeds');
    plot(1:t, omegaValsTransient);
    xlabel('Time [ms]');
    ylabel('$\omega$ [pu]');
    legend({'Gen02', 'Gen03', 'Gen04'});
    title('Transient Simulation: Machine Rotor Speeds')

    if saveTransientRunPlots
        filenameFigTheta = strcat(folder_processedData, systemName1, "/
transientRun_theta_", modelType, '.png');
        save(filenameFigTheta, 'figTheta');
        filenameFigOmega = strcat(folder_processedData, systemName1, "/
transientRun_omega_", modelType, '.png');
        save(filenameFigOmega, 'figOmega');
    end

    transientRunStop = toc(transientRunStart);
    display(transientRunStop);
end
```