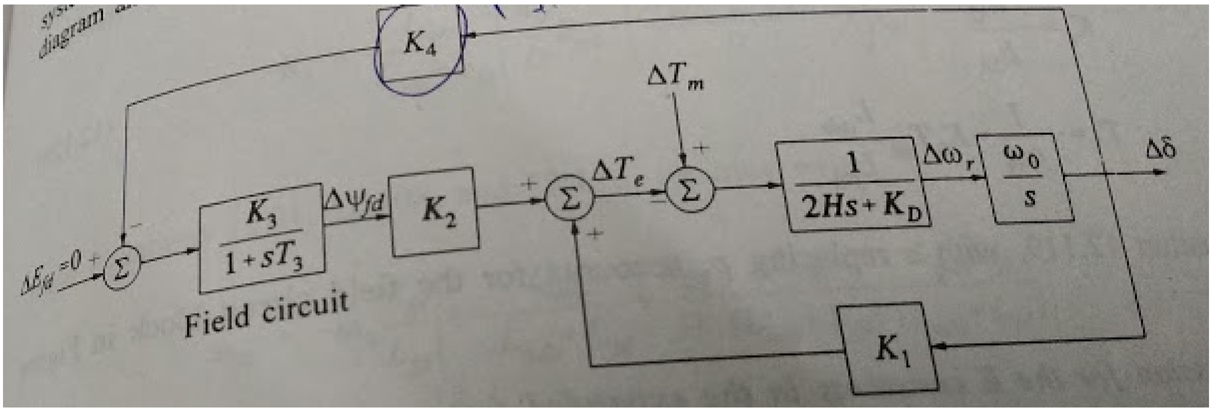
Page no 700 to 800 P Kindur, regarding small signal stability analysis

code credits S.Naresh Ram, Sr.Engineer



744

$$\begin{bmatrix} \Delta\dot{\omega}_r \\ \Delta\dot{\delta} \\ \Delta\dot{\psi}_{fd} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & 0 & 0 \\ 0 & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \Delta\omega_r \\ \Delta\delta \\ \Delta\psi_{fd} \end{bmatrix} + \begin{bmatrix} b_{11} & 0 \\ 0 & 0 \\ 0 & b_{32} \end{bmatrix} \begin{bmatrix} \Delta T_m \\ \Delta E_{fd} \end{bmatrix}$$

where

$$a_{11} = -\frac{K_D}{2H}$$

$$a_{12} = -\frac{K_1}{2H}$$

$$a_{13} = -\frac{K_2}{2H}$$

$$a_{21} = \omega_0 = 2\pi f_0$$

$$a_{32} = -\frac{\omega_0 R_{fd}}{L_{fd}} m_1 L'_{ads} \Rightarrow \frac{K_3 \cdot K_4}{T_3}$$

$$a_{33} = -\frac{\omega_0 R_{fd}}{L_{fd}}\left[1 - \frac{L'_{ads}}{L_{fd}} + m_2 L'_{ads}\right] \Rightarrow -1/T_3$$

$$b_{11} = \frac{1}{2H}$$

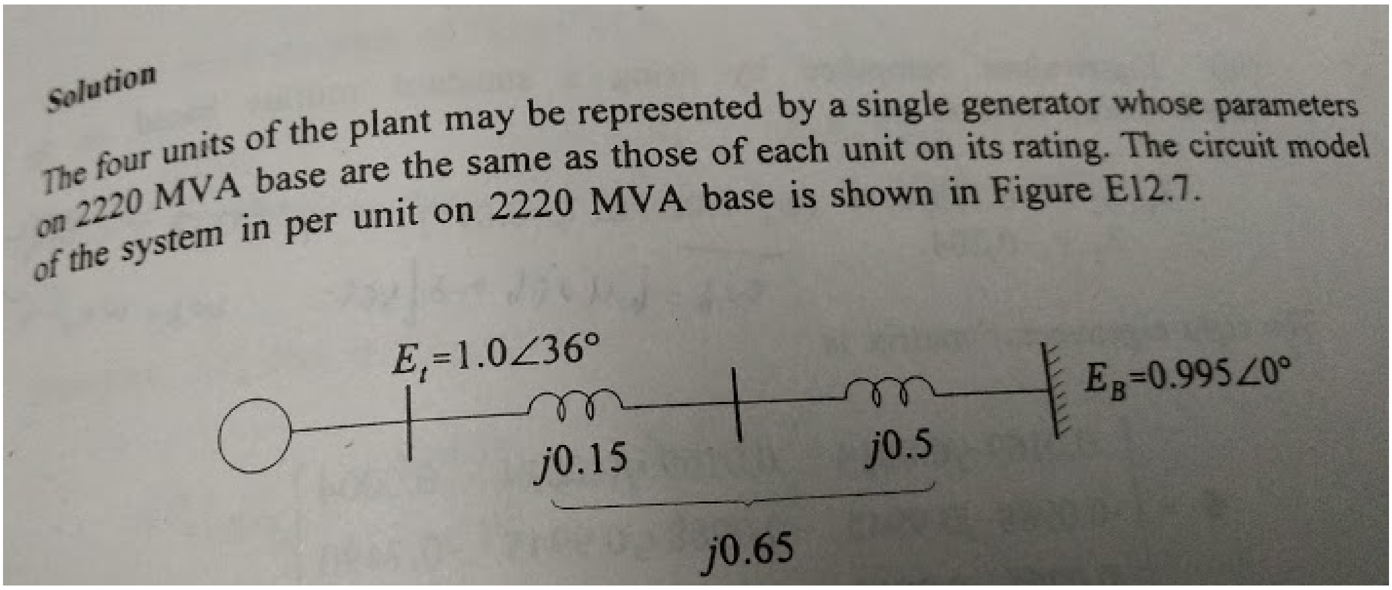
$$b_{32} = \frac{\omega_0 R_{fd}}{L_{adu}} \Rightarrow \frac{K_3}{T_3}$$

K1=Synchronizing Coeffcient , depends on operating load angle

K3 & T3=almost constant, depends on system parameter

K4= demagnetising , depends on operating load angle

K2= relation between voltage and flux, depends on Eqo and Iq0


Solution

The four units of the plant may be represented by a single generator whose parameters on 2220 MVA base are the same as those of each unit on its rating. The circuit model of the system in per unit on 2220 MVA base is shown in Figure E12.7.

In [39]:

```python


import numpy as np
import pandas as pd
from sympy import sin,cos,nsolve,Symbol,tan,cot
from numpy import exp, abs, angle

def polar2z(r,theta):
    return r * exp( 1j * theta )

def z2polar(z):
    return ( abs(z), angle(z) )

#Given values
Xd,X_d,Xq,X_q=complex(0,1.81),complex(0,0.3),complex(0,1.76),0
X_line=complex(0,0.65)
Xl,Ra,T_do,H=complex(0,0.16),0.003,8,3.5
Vref=complex(0.995,0)
Vt=complex(0.809,0.5877)
P,Q=0.9,0.3 #overexcited
L=0.16
Ladu,Laqu=1.65,1.60
#Initial values
Ksd,Ksq=0.8491,0.8491
#deltai=43.13
#ed0=0.6836
#eq0=0.7298
#id0=0.8342
#iq0=0.4518
delta0=79.13
#Efd0=2.395
Ksd_incr,Ksq_incr=0.434,0.434

```

In [40]:

```
Phi=np.angle(complex(P,Q))*180/3.14
print('\n Phi',Phi)
I=np.conjugate(np.divide(complex(P,Q),Vt))
Xds=Ksd*Ladu+L
Xqs=Ksq*Laqu+L
Z=complex(Ra,Xqs)
print('\n compleximpedance\n',Z)
E=Vt+I*Z
print('\n current flowing in the line\n ',I)
print('internal machine voltage',E)
load_angle=np.angle(E)*180/3.14
print('\n loadangle',load_angle)

```

```
 Phi 18.4442992966228

 compleximpedance
 (0.003+1.51856j)

 current flowing in the line
  (0.9045255169537701+0.2862665591022629j)
internal machine voltage (0.377000630560529+1.962135068702624j)

 loadangle 79.16398409330476
```

In [41]:

```
K1,K2,K3,K4=0.7643,0.8649,0.3230,1.4187
T3=2.365
KD=2
```

In [42]:

```
A=np.array([[-KD/(2*H),-K1/(2*H),-K2/(2*H)],[2*3.14*60,0,0],[0,-K3*K4/T3,-1/T3]])
print('\n State_Matrix A\n',A)
```

```
 State_Matrix A
 [[-2.85714286e-01 -1.09185714e-01 -1.23557143e-01]
 [ 3.76800000e+02  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.93759027e-01 -4.22832981e-01]]
```

In [43]:

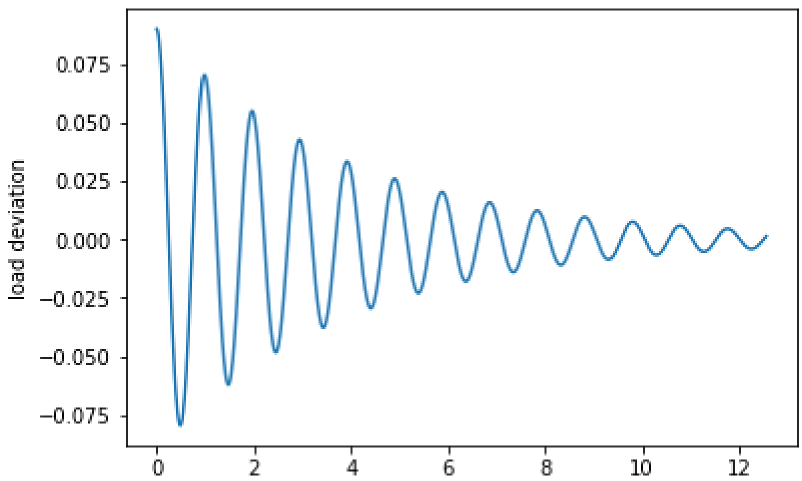
```
w,V=np.linalg.eig(A)
print('\n Eigen values\n',w)
print('\n Right eigen vector\n',V)
L=np.linalg.inv(V)
print('\n Left eigen vector\n',L)
p=np.array([[L[:][0]*V[:][0]],[L[:][1]*V[:][0]],[L[:][2]*V[:][2]]])
print('\n participation factor\n',p)
```

```
 Eigen values
 [-0.2525329 +6.41057269j -0.2525329 -6.41057269j -0.20348146+0.j        ]

 Right eigen vector
 [[-6.69801402e-04+0.01700297j -6.69801402e-04-0.01700297j
   4.04735672e-04+0.j        ]
 [ 9.99399142e-01+0.j          9.99399142e-01-0.j
  -7.49475668e-01+0.j        ]
 [-8.01891701e-04+0.03018545j -8.01891701e-04-0.03018545j
   6.62031766e-01+0.j        ]]

 Left eigen vector
 [[-1.00615177e+00-2.93989204e+01j  5.00081027e-01-1.97067565e-02j
   5.66748918e-01-4.33656333e-03j]
 [-1.00615177e+00+2.93989204e+01j  5.00081027e-01+1.97067565e-02j
   5.66748918e-01+4.33656333e-03j]
 [-2.68333520e+00+1.24905083e-17j -5.85611046e-04+2.72592839e-21j
   1.51147904e+00-9.14023852e-18j]]

 participation factor
 [[[ 5.00543021e-01+2.58386504e-03j -6.70028454e-04-8.48966541e-03j
    2.29383504e-04-1.75516187e-06j]]

 [[-4.99195177e-01-3.67990112e-02j  1.18508840e-07-8.51606463e-03j
    2.29383504e-04+1.75516187e-06j]]

 [[ 2.15174423e-03-8.09976898e-02j  4.69596638e-07+1.76769350e-05j
    1.00064714e+00-6.05112825e-18j]]]
```

In [44]:

```
initial_value=np.matrix([[0.],
               [0.09],
               [0]])
print('\n initial value',initial_value)
C=L*initial_value
print('\n C \n',C)
t = np.arange(0,4*np.pi,0.01)
K=np.array([C[0]*V[1][0],C[1]*V[1][1],C[2]*V[1][2]])
Y=K.flatten()
print('\n K values\n',Y[0])

import matplotlib.pyplot as plt
print('\n time response of load angle deviation\n')
plt.plot(t,Y[0]*np.exp(w[0]*t)+Y[1]*np.exp(w[1]*t)+Y[2]*np.exp(w[2]*t))
plt.ylabel('load deviation')
plt.show()
```

```
 initial value [[0.  ]
 [0.09]
 [0.  ]]

 C
 [[ 4.50072924e-02-1.77360809e-03j]
 [ 4.50072924e-02+1.77360809e-03j]
 [-5.27049941e-05+2.45333555e-22j]]

 K values
 (0.04498024944467663-0.0017725424023342412j)

 time response of load angle deviation


C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:492: Comple
xWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



field dynamics removed

In [45]:

```
# state matrix
```

In [46]:

```
A=np.array([[-KD/(2*H),-K1/(2*H)],[2*3.14*60,0]])
print('\n State_Matrix A\n',A)
```

```
 State_Matrix A
[[-2.85714286e-01 -1.09185714e-01]
[ 3.76800000e+02  0.00000000e+00]]
```

In [47]:

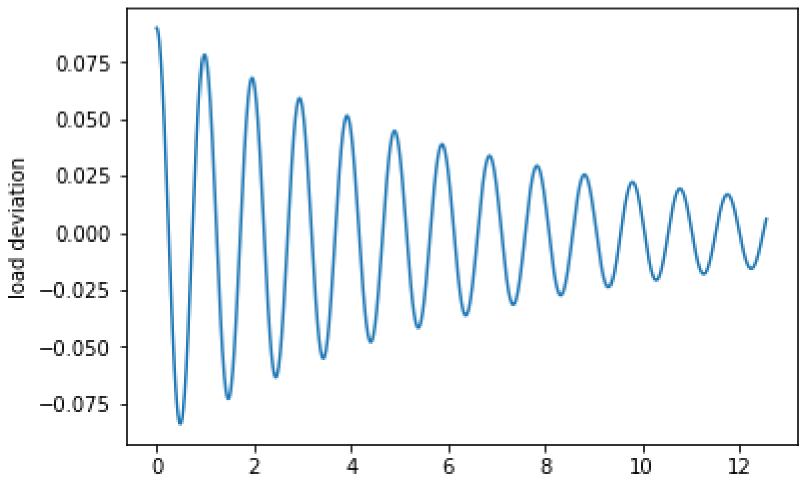
```
w,V=np.linalg.eig(A)
print('\n Eigen values\n',w)
print('\n Right eigen vector\n',V)
L=np.linalg.inv(V)
print('\n Left eigen vector\n',L)
p=np.array([[L[:][0]*V[:][0]],[L[:][1]*V[:][0]]])
print('\n participation factor\n',p)
```

```
 Eigen values
[-0.14285714+6.41254778j -0.14285714-6.41254778j]

 Right eigen vector
[[-3.79077626e-04+0.01701597j -3.79077626e-04-0.01701597j]
[ 9.99855146e-01+0.j          9.99855146e-01-0.j        ]]

 Left eigen vector
[[0.         -2.93841545e+01j 0.50007244-1.11404893e-02j]
[0.        +2.93841545e+01j 0.50007244+1.11404893e-02j]]

 participation factor
[[[ 5.00000000e-01+0.01113888j -3.79132545e-04-0.008505j   ]]

[[-5.00000000e-01-0.01113888j  0.00000000e+00-0.00851344j]]]
```

In [48]:

```
initial_value=np.matrix([[0.],
                [0.09]])
print('\n initial value',initial_value)
C=L*initial_value
print('\n C \n',C)
t = np.arange(0,4*np.pi,0.01)
K=np.array([C[0]*V[1][0],C[1]*V[1][1]])
Y=K.flatten()
print('\n K values\n',Y[0])

import matplotlib.pyplot as plt
print('\n time response of load angle deviation\n')
plt.plot(t,Y[0]*np.exp(w[0]*t)+Y[1]*np.exp(w[1]*t))
plt.ylabel('load deviation')
plt.show()
```

```
 initial value [[0.  ]
 [0.09]]

 C
 [[0.04500652-0.00100264j]
 [0.04500652+0.00100264j]]

 K values
 (0.045-0.0010024987965973292j)

 time response of load angle deviation


C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:492: Comple
xWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



Since $p\Delta\omega_r$ and $p\Delta\delta$ are not directly affected by the exciter,

$$a_{14} = a_{24} = 0$$

The complete state-space model for the power system, including the excitation system of Figure 12.11, has the following form:

$$\begin{bmatrix} \Delta\dot{\omega}_r \\ \Delta\dot{\delta} \\ \Delta\dot{\psi}_{fd} \\ \Delta\dot{v}_1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} \Delta\omega_r \\ \Delta\delta \\ \Delta\psi_{fd} \\ \Delta v_1 \end{bmatrix} + \begin{bmatrix} b_1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \Delta T_m \tag{12.141}$$

With constant mechanical torque input,

$$\Delta T_m = 0$$

Block diagram including the excitation system

Figure 12.12 shows the block diagram obtained by extending the diagram of Figure 12.9 to include the voltage transducer and AVR/exciter blocks. The

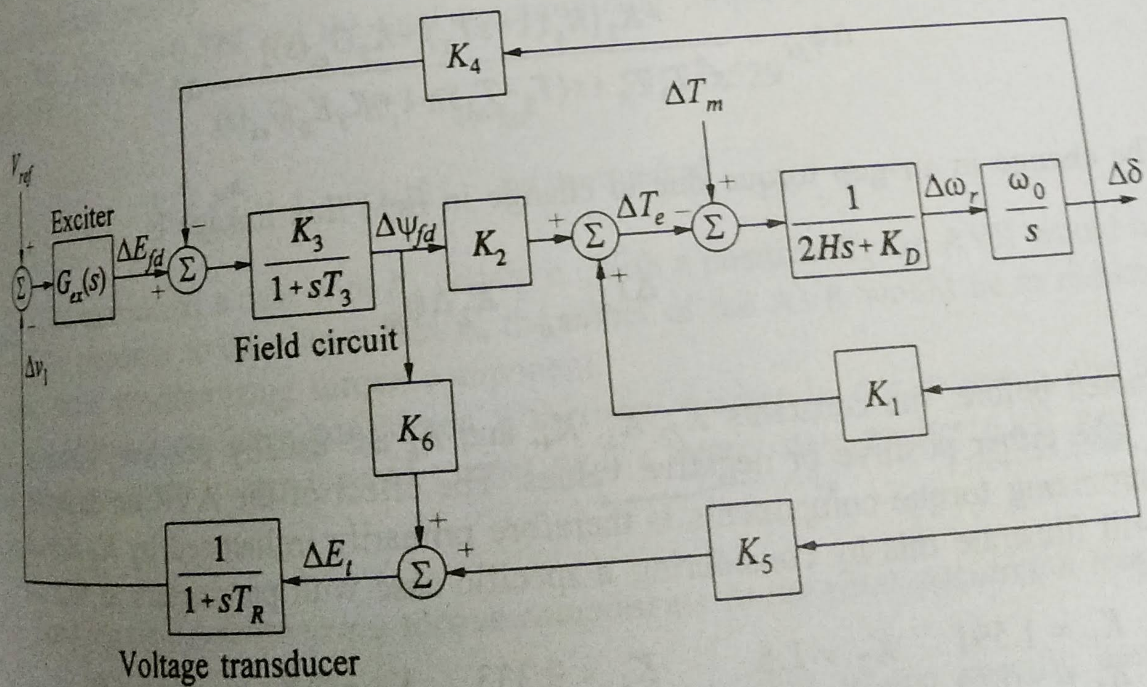


Figure 12.12 Block diagram representation with exciter and AVR

with AVR

In [49]:

```
K1,K2,K3,K4=0.7643,0.8649,0.3230,1.4187
T3=2.365
KD=2
KA=200
TR=0.02
K5,K6=-0.1463,0.4168
```

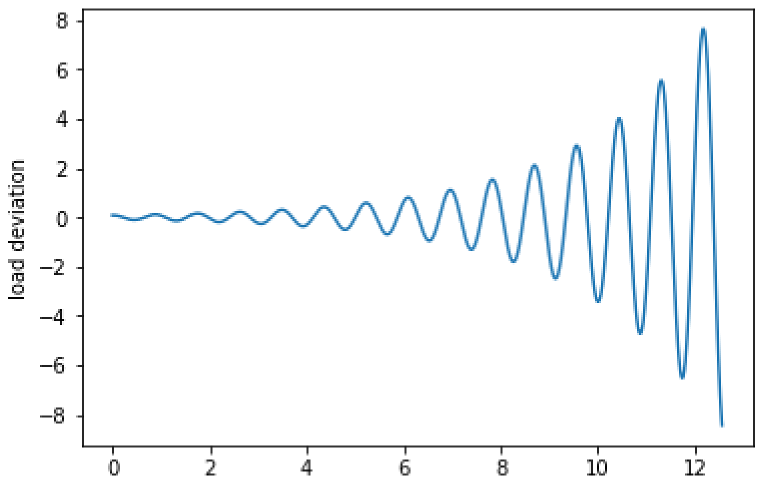
In [50]:

```
A=np.array([[-KD/(2*H),-K1/(2*H),-K2/(2*H),0],[2*3.14*60,0,0,0],[0,-K3*K4/T3,-1/T3,-K3
print('\n State_Matrix A\n',A)
w,V=np.linalg.eig(A)
print('\n Eigen values\n',w)
print('\n Right eigen vector\n',V)
L=np.linalg.inv(V)
print('\n Left eigen vector\n',L)
p=np.array([[L[:][0]*V[:][0]],[L[:][1]*V[:][0]]])
print('\n participation factor\n',p)
initial_value=np.matrix([[0.0],
               [0.09],[0],[0]])
print('\n initial value',initial_value)
C=L*initial_value
print('\n C \n',C)
t = np.arange(0,4*np.pi,0.01)
K=np.array([C[0]*V[1][0],C[1]*V[1][1]])
Y=K.flatten()
print('\n K values\n',Y[0])

import matplotlib.pyplot as plt
print('\n time response of load angle deviation\n')
plt.plot(t,Y[0]*np.exp(w[0]*t)+Y[1]*np.exp(w[1]*t))
plt.ylabel('load deviation')
plt.show()
```

```
 State_Matrix A
[[-2.85714286e-01 -1.09185714e-01 -1.23557143e-01  0.00000000e+00]
 [ 3.76800000e+02  0.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.93759027e-01 -4.22832981e-01 -2.73150106e+01]
 [ 0.00000000e+00 -7.31500000e+00  2.08400000e+01 -5.00000000e+01]]

 Eigen values
 [  0.36907426+7.23527384j   0.36907426-7.23527384j
 -31.22485294+0.j         -20.22184284+0.j        ]

 Right eigen vector
 [[ 9.39198586e-04+0.0184119j  9.39198586e-04-0.0184119j
   2.53982720e-03+0.j        -4.53544931e-03+0.j       ]
 [ 9.58858607e-01+0.j         9.58858607e-01-0.j
  -3.06488838e-02+0.j         8.45104630e-02+0.j       ]
 [ 2.25859165e-01-0.1525712j  2.25859165e-01+0.1525712j
   6.63065560e-01+0.j        -8.06482195e-01+0.j       ]
 [-5.37631648e-02-0.0554029j -5.37631648e-02+0.0554029j
   7.47929314e-01+0.j        -5.85169958e-01+0.j       ]]

 Left eigen vector
 [[  0.60813347-2.56600566e+01j   0.49377849-3.29137444e-02j
    0.09849996+2.20354224e-01j  -0.06915456-1.09563772e-01j]
 [  0.60813347+2.56600566e+01j   0.49377849+3.29137444e-02j
    0.09849996-2.20354224e-01j  -0.06915456+1.09563772e-01j]
 [ -9.64433177+2.37911971e-15j   0.7918984 +7.40102547e-17j
   -2.46734203-2.49078800e-17j   3.58961096+1.64094215e-17j]
 [-17.29745609+2.72118046e-15j   0.9151919 +1.02591826e-16j
   -3.12998339-2.88274943e-17j   2.87108194+1.98913946e-17j]]

 participation factor
 [[[ 4.73021600e-01-0.012903j   -1.42248569e-04-0.00912231j
    2.50172874e-04+0.00055966j  3.13647014e-04+0.00049692j]]
```

```
[[-4.71879284e-01+0.03529678j  1.06976069e-03-0.00906049j
   2.50172874e-04-0.00055966j  3.13647014e-04-0.00049692j]]]

initial value [[0.  ]
[0.09]
[0.  ]
[0.  ]]

C
[[0.04444006-2.96223699e-03j]
[0.04444006+2.96223699e-03j]
[0.07127086+6.66092292e-18j]
[0.08236727+9.23326436e-18j]]

K values
(0.04261173798522495-0.002840366436913836 3j)

time response of load angle deviation
```

```
C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:492: Comple
xWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



result is shocked!, yes AVR damps first few cycles and possible to have negative damping in remaing cycles

its the time to PSS now

In [51]:

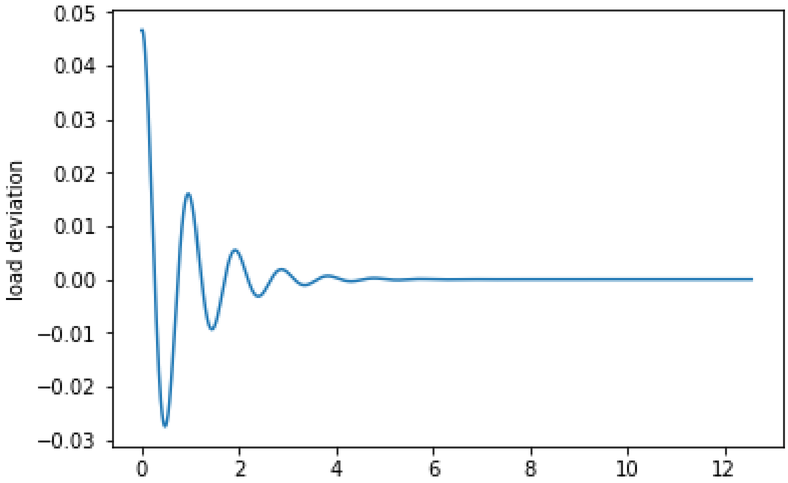
```
K1,K2,K3,K4=0.7643,0.8649,0.3230,1.4187
T3=2.365
KD=2
KA=200
TR=0.02
K5,K6=-0.1463,0.4168
KSTAB=9.5
TW=1.4
T1,T2=0.154,0.033
```

In [52]:

```
A=np.array([[-KD/(2*H),-K1/(2*H),-K2/(2*H),0,0,0],[2*3.14*60,0,0,0,0,0],[0,-K3*K4/T3,-
            [0,K5/TR,K6/TR,-1/TR,0,0],[-(KD*KSTAB)/(2*H),KSTAB*(-K1/(2*H)),KSTAB*(-K2/
[(-T1/T2)*((KD*KSTAB)/(2*H)),(-T1/T2)*(KSTAB*(K1/(2*H))),(T1/T2)*(KSTAB*(-K2/(2*H))),0
print('\n State_Matrix A\n',A)
w,V=np.linalg.eig(A)
print('\n Eigen values\n',w)
print('\n Right eigen vector\n',V)
L=np.linalg.inv(V)
print('\n Left eigen vector\n',L)
p=np.array([[L[:][0]*V[:][0]],[L[:][1]*V[:][0]]])
print('\n participation factor\n',p)
initial_value=np.matrix([[0.0],
               [0.09],[0],[0],[0],[0]])
print('\n initial value',initial_value)
C=L*initial_value
print('\n C \n',C)
t = np.arange(0,4*np.pi,0.01)
K=np.array([C[0]*V[1][0],C[1]*V[1][1]])
Y=K.flatten()
print('\n K values\n',Y[0])

import matplotlib.pyplot as plt
print('\n time response of load angle deviation\n')
plt.plot(t,Y[0]*np.exp(w[0]*t)+Y[1]*np.exp(w[1]*t))
plt.ylabel('load deviation')
plt.show()
```

```
 State_Matrix A
[[-2.85714286e-01 -1.09185714e-01 -1.23557143e-01  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 3.76800000e+02  0.00000000e+00  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00 -1.93759027e-01 -4.22832981e-01 -2.73150106e+01
   0.00000000e+00  2.73150106e+01]
 [ 0.00000000e+00 -7.31500000e+00  2.08400000e+01 -5.00000000e+01
   0.00000000e+00  0.00000000e+00]
 [-2.71428571e+00 -1.03726429e+00 -1.17379286e+00  0.00000000e+00
  -7.14285714e-01  0.00000000e+00]
 [-1.26666667e+01 -4.84056667e+00 -5.47770000e+00  0.00000000e+00
   2.69696970e+01 -3.03030303e+01]]

 Eigen values
 [-39.11517705 +0.j          -1.12325708 +6.56814561j
  -1.12325708 -6.56814561j  -0.73890044 +0.j
 -19.81263582+12.90040806j -19.81263582-12.90040806j]

 Right eigen vector
 [[ 0.00138374+0.j          -0.00273283+0.01597995j -0.00273283-0.01597995j
   -0.0012068 +0.j          -0.00364177-0.00207396j -0.00364177+0.00207396j]
 [-0.01332969+0.j          0.9167345 +0.j          0.9167345 -0.j
   0.61540201+0.j          0.03060322+0.05936926j  0.03060322-0.05936926j]
 [ 0.44663775+0.j          0.02084443+0.25359511j  0.02084443-0.25359511j
  -0.54824844+0.j         -0.81912595+0.j         -0.81912595-0.j        ]
 [ 0.8640873 +0.j         -0.1117642 +0.12314662j -0.1117642 -0.12314662j
  -0.32332131+0.j         -0.489633  +0.19485568j -0.489633  -0.19485568j]
 [ 0.01339004+0.j         -0.04258248+0.15002109j -0.04258248-0.15002109j
  -0.34415054+0.j         -0.03514353-0.02080879j -0.03514353+0.02080879j]
 [ 0.23132005+0.j         -0.16677512+0.12165605j -0.16677512-0.12165605j
```

```
  -0.31261206+0.j          0.09204792-0.19158223j  0.09204792+0.19158223j]]

 Left eigen vector
 [[-2.77369530e+00-2.05576084e-15j  3.22865562e-01-1.60606316e-17j
  -4.18380962e-01+9.63637893e-17j  1.04990963e+00-2.93106526e-16j
  -9.10806934e-01-1.60606316e-16j  1.29685543e+00-3.35265684e-16j]
 [-9.91629413e+00-3.08043510e+01j  5.69474193e-01-1.01027310e-01j
   7.98646982e-02+2.66066804e-01j -6.34682465e-02-1.40163777e-01j
   8.72387788e-01-5.65593236e-01j  1.24514454e-01+2.21036279e-01j]
 [-9.91629413e+00+3.08043510e+01j  5.69474193e-01+1.01027310e-01j
   7.98646982e-02-2.66066804e-01j -6.34682465e-02+1.40163777e-01j
   8.72387788e-01+5.65593236e-01j  1.24514454e-01-2.21036279e-01j]
 [ 2.92031994e+01-4.87453250e-16j -5.74795058e-02+1.24569451e-16j
   3.07884940e-03-1.62177096e-15j -1.70720517e-03+1.62121527e-15j
  -3.11677727e+00+4.50375006e-15j  2.84462301e-03+3.23989481e-15j]
 [-1.24472875e+00-2.09471548e+00j  1.53027621e-01+1.16245842e-01j
  -8.05840493e-01+8.80745630e-01j  3.28587453e-01-9.37362012e-01j
   9.92032870e-01-2.06938263e+00j  2.87348031e-01+1.93993366e+00j]
 [-1.24472875e+00+2.09471548e+00j  1.53027621e-01-1.16245842e-01j
  -8.05840493e-01-8.80745630e-01j  3.28587453e-01+9.37362012e-01j
   9.92032870e-01+2.06938263e+00j  2.87348031e-01-1.93993366e+00j]]

 participation factor
 [[[-3.83807252e-03-2.84463805e-18j -8.82335131e-04+5.15937599e-03j
    1.14336202e-03+6.68570745e-03j -1.26702696e-03+3.53719846e-19j
    3.31694762e-03+1.88897552e-03j -4.72284673e-03+2.68962398e-03j]]

 [[-1.37215707e-02-4.26252058e-02j  5.81381006e-05+9.37625993e-03j
    4.03347834e-03-2.00334805e-03j  7.65932396e-05+1.69149115e-04j
   -4.35005074e-03+2.50463577e-04j -9.11872772e-04-5.46725091e-04j]]]

 initial value [[0.  ]
 [0.09]
 [0.  ]
 [0.  ]
 [0.  ]
 [0.  ]]

 C
 [[ 0.0290579 -1.44545684e-18j]
 [ 0.05125268-9.09245789e-03j]
 [ 0.05125268+9.09245789e-03j]
 [-0.00517316+1.12112506e-17j]
 [ 0.01377249+1.04621258e-02j]
 [ 0.01377249-1.04621258e-02j]]

 K values
 (-0.00038733277988056117+1.9267490272949947e-20j)

 time response of load angle deviation


C:\ProgramData\Anaconda3\lib\site-packages\numpy\core\numeric.py:492: Comple
xWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```