In [1]:

```
import numpy as np
from scipy.integrate import odeint
from scipy import signal
import matplotlib.pyplot as pl
%matplotlib inline
```

Machine parameters for PMSM

We try to create parameter set for 3 types of PMSM

1. IPMSM $l_d < l_q$
2. IPMSM $l_d > l_q$
3. EPMSM $l_d = l_q$

In [2]:

```
#Defining Machine parameters

mach_ramu = {"rs": 0.1729, "lq": 0.6986, "ld": 0.4347, "tmech":50.5}
mach_servo = {"rs": 0.0426, "ld": 2.252+0.078, "lq": 2.252+0.078, "tmech":100.0}
mach_servodq = {"rs": 0.0426, "ld": 2.252+0.078, "lq": 2.252+0.0758, "tmech":100.0}
```

Getting machine parameters

In [3]:

```
def mach_parapm(dict):
    """Takes in the dictionary containing machine parameters (normalized)
       and returns various parameters that are used in space vector equations

       Takes arguement: machine dictionary
       """
    rs = dict["rs"]
    lq = dict['lq']
    ld = dict['ld']
    tsq=lq/rs
    tsd=ld/rs
    tmech = dict['tmech']
    psirm = 0.9
#     print("lh = {0:1.3f}".format(lh))
    return rs, lq, ld, tmech, psirm

def mach_parapmSym(dict):
    """Takes in the dictionary containing machine parameters (normalized)
       and returns various parameters that are used in space vector equations

       Takes arguement: machine dictionary
       """
    rs = dict["rs"]
    lq = dict['lq']
    ld = dict['ld']
    ls = dict['ld']
    tsq=lq/rs
    tsd=ld/rs
    tmech = dict['tmech']
    psirm = 0.95
#     print("lh = {0:1.3f}".format(lh))
    return rs, ls, tmech, psirm

#Test
mach_parapmSym(mach_servo)
```

Out[3]:

```
(0.0426, 2.3299999999999996, 100.0, 0.95)
```

Define PMSM model in stator coordinates with Ld = Lq

This models is for machine is stator coordinates. For simplicity we will assume $l_d = l_q$, which is the case with most surface-mounted permanent magnets on rotor.

$$\vec{v}_s = r_s \vec{i}_s + \frac{d\vec{\psi}_s}{d\tau}$$
$$\vec{\psi}_s = l_s \vec{i}_s + \vec{\psi_{r,m}}$$

Where $\psi_{r,m}$ is the rotor flux coupling with the stator winding. Hence, we get

$$\vec{v}_s = r_s\vec{i}_s + l_s\frac{d\vec{i}_s}{d\tau} + \frac{d\vec{\psi}_{r,m}}{d\tau}$$
$$\vec{\psi}_{r,m} = |\psi_{r,m}|e^{j\delta} = \psi_{r,m\alpha} + j\psi_{r,m\beta}$$
$$\vec{v}_s = r_s\vec{i}_s + l_s\frac{d\vec{i}_s}{d\tau} + j\omega_s\vec{\psi}_{r,m}$$
$$v_{s\alpha} = r_s i_{s\alpha} + l_s\frac{di_{s\alpha}}{d\tau} - \omega_s\psi_{r,m\beta}$$
$$v_{s\beta} = r_s i_{s\beta} + l_s\frac{di_{s\beta}}{d\tau} + \omega_s\psi_{r,m\alpha}$$

In [4]:

```
# Dynamic model in stator coordinates for symmetrical PMSM rotor


#PMSM machine 4x4 dynamic model

def PMSM_dynstepstatorSym(X,t,params):
    """Defines a function to calculate the derivatives for a 4x4 PMS motor dynamice
       1. Uses state variables stator current $\vec{i}_s$ and rotor flux space vector $\vec{\psi}_r$
       2. Is non-linear and uses the state variable $\omega$ rotor angular velocity to describe
          rotor dynamics
       3. calls mach_im function to calculate the system equation coefficients
       4. mach_im will in  turn call mach_para function.
       5. uses normalized time 2*np.pi*50*t
       6. This is to be used in a for loop for the time span
    """
    x0 = X[0]  #isalpha
    x1 = X[1]  #isbeta
    x2 = X[2]  #ws
    x3 = X[3]  #delta

#     wx = X[5]
    u1,u2,mL,rs,ls,tmech,psirm = params


#     rs,ls,tmech,psirm = mach_parapmSym(mach_servo)
#     rs,ls,tmech,psirm = mach_parapmSym(mach_ramu)
    psrma = psirm*np.cos(x3)
    psrmb = psirm*np.sin(x3)
    mdiff = (psrma*x1 - psrmb*x0) - mL
#     print(A[1][0],A[1][1],A[1][2],A[1][3],B[4][0])
    dx0dt = -(rs/ls)*x0  + psrmb*x2 + u1/ls
    dx1dt = -(rs/ls)*x1 - psrma*x2 + u2/ls
    dx2dt = mdiff/tmech
    dx3dt = x2
#     dwxdt = rate
    return [dx0dt,dx1dt,dx2dt,dx3dt]
```

In [5]:

```
def rotorangle(X,t,params):
    del0 = X
    w = params
    ddel0dt = w
    return ddel0dt
```

In [6]:

```
def PIcon(xe,y,dt,Kparams):
    """Pass on xe = [error[k], error[k-1]]
       Pass on y  = [y[k-1]]
       Pass on Parameters = [kp, Ti]
    """
    kp, Ti = Kparams
    xeo = xe[1]
    xen = xe[0]
#     print(xeo,xen,y,kp,Ti)
    y1 = y
    y2 = y1 + kp*(xen - xeo) + (kp/Ti)*(xen)
    return y2

def PIconwithLim(xe,y,dt,Kparams):
    """Pass on xe = [error[k], error[k-1]]
       Pass on y  = [y[k-1]]
       Pass on Parameters = [kp, Ti]
    """
    kp, Ti, uplim, dwnlim = Kparams
    xeo = xe[1]
    xen = xe[0]
#     print(xeo,xen,y,kp,Ti)
#     y2 = y1 + kp*(xen - xeo) + (kp/Ti)*(xen)
#     print(xeo,xen,y,kp,Ti)
    y1 = y
    yx = y1 + kp*(xen - xeo) + (kp/Ti)*(xen)
    if (yx > uplim):
        yx = uplim
    elif (yx < dwnlim):
        yx = dwnlim
    y2 = yx
    return y2
```

In [7]:

```
#Preparing simulation with current control
# We will use for loop


# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-3

tend = 20*2*np.pi
tstart = 0.0
delta_t = 0.01
n = 100000
n2 = int(100*2*np.pi/0.01)
ws = -0.5
mL = 0.0
Tta = np.arange(tstart, tend,delta_t)
# Tta = np.linspace(tstart, tend, n2)
#delta_t = Tta[1] - Tta[0]
usa = np.zeros(len(Tta))
usb = np.zeros(len(Tta))
sol1 = np.zeros((len(Tta),4))
sol2 = np.zeros(len(Tta))
FCangle = np.zeros(len(Tta))
gamma = np.zeros(len(Tta))
# Field coordinate currents
cosdelta = np.zeros(len(Tta))
sindelta = np.zeros(len(Tta))
isa = np.zeros(len(Tta))
isb = np.zeros(len(Tta))
w = np.zeros(len(Tta))
gamma = np.zeros(len(Tta))
isd = np.zeros(len(Tta))
isq = np.zeros(len(Tta))
usd = np.zeros(len(Tta))
usq = np.zeros(len(Tta))
eid = np.zeros(len(Tta))
eiq = np.zeros(len(Tta))

#Creating reference values for isd and isq
isdrefval = 0.01
isdref = np.zeros(len(Tta))
isqref = np.zeros(len(Tta))

isqref1 = 0.0
isqref2 = 0.9
isqref3 = -0.35
for ii in range(len(Tta)):
    isdref[ii] = isdrefval
    if (Tta[ii]>=18*2*np.pi):
        isqref[ii] = isqref3
    elif(Tta[ii]>=1*2*np.pi):
        isqref[ii]=isqref2
    else:
        isqref[ii]=isqref1



#
rs,ls,tmech, psirm = mach_parapmSym(mach_servo)
rs,ls,tmech,psirm = mach_parapmSym(mach_servo)
```

```
psird = np.zeros(len(Tta))
psirq = np.zeros(len(Tta))

me = np.zeros(len(Tta))
x0 = [0.0,0.0,ws,0]

kp = 5.5
Ti = 5.0e1

Kparams = [kp,Ti]

#isdref = 0.26
#isqref = 0.75
#The intial value error will never get corrected in
#rotor angle
y0 = 0
for ii in range(len(Tta)):
    #Start controller after first step
    if ii>=0:
        eid[ii] = isdref[ii-1] - isd[ii-1]
        PIed = [eid[ii],eid[ii-1]]
        usd[ii] = PIcon(PIed, usd[ii-1], delta_t,Kparams)
        eiq[ii] = isqref[ii-1] - isq[ii-1]
        PIeq = [eiq[ii],eiq[ii-1]]
        usq[ii] = PIcon(PIeq, usq[ii-1], delta_t,Kparams)
#         usd[ii] = 0
#         usq[ii] =.0001
    if ws>=1.0:
        a = 1.0
    else:
        a = ws
#     usa[ii] = a*np.cos(ws*Tta[ii])
#     usb[ii] = a*np.sin(ws*Tta[ii])
    usa[ii] = usd[ii-1]*cosdelta[ii-1] - usq[ii-1]*sindelta[ii-1]
    usb[ii] = usq[ii-1]*cosdelta[ii-1] + usd[ii-1]*sindelta[ii-1]
#     me[ii] = kr*(sol1[ii-1][2]*sol1[ii-1][1] - sol1[ii-1][3]*sol[ii-1][0]) - mL
    params = [usa[ii], usb[ii],mL,rs,ls,tmech,psirm]
    sol1a = odeint(PMSM_dynstepstatorSym,x0,[0,delta_t], args = (params,),atol = abserr, rtol= relei
    sol1[ii]= sol1a[-1]
#     display(sol1a[-1])
    x0 = sol1a[-1]
    params2 = sol1[ii][2] #  w
    sol2a = odeint(rotorangle,y0,[0,delta_t], args = (params2,), atol = abserr, rtol= relerr)
    sol2[ii] = sol2a[-1]
    #Using the internal state of the machine
    #for internal model use sol1[ii][3]
    #Change this to resolver model
    #Resolver model use sol2[ii]
    FCangle[ii] = sol2[ii] #sol1[ii][3]
    cosdelta[ii] = np.cos(FCangle[ii])
    sindelta[ii] = np.sin(FCangle[ii])
    #Coordinate transformation
    isa[ii] = sol1[ii][0]
    isb[ii] = sol1[ii][1]
    w[ii] = sol1[ii][2]
    gamma[ii] = sol1[ii][3]
    #Convert stator coordinate current to field coordinates
    isd[ii] = sol1[ii][0]*cosdelta[ii] + sol1[ii][1]*sindelta[ii]
    isq[ii] = sol1[ii][1]*cosdelta[ii] - sol1[ii][0]*sindelta[ii]
    #Rotor flux in field coordinates Using estimator output as in practice
    #actual flux will not be available for measurement
```

```
    psird[ii] = 1.0
    y0 = sol2a[-1]
    psird[ii] = psirm

#Do not use these assigments
# isa = sol1[:, 0]
# isb = sol1[:, 1]
# w = sol1[:, 2]
# gamma = sol1[:,3]



me = psirm*isq
```

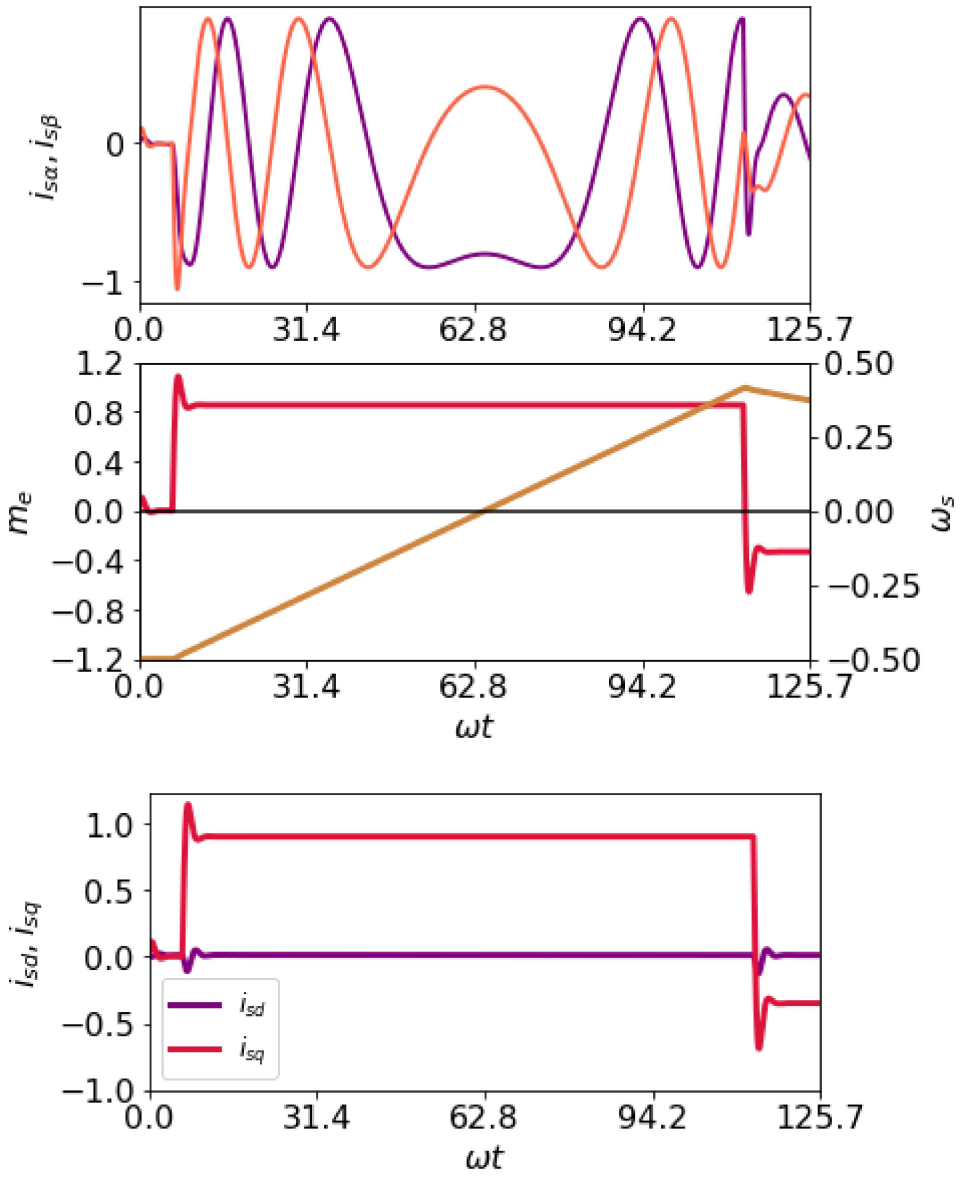
In [ ]:

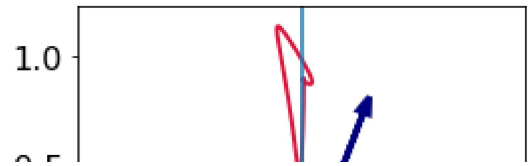
```

```

In [8]:

```
dirfig =  "C:/Users/ashwi/Documents/tex/lec/drives/figures/"
pl.figure(501, figsize = (6,6))
pl.rc('font', size = 16)
pl.subplot(2,1,1)
pl.plot(Tta,  isa, 'purple', Tta, isb,'tomato', lw=2)
# pl.plot(Tta, isd,'tomato', Tta,isq ,'crimson' ,lw =2)
pl.xlim(0,tend)
pl.ylabel(r'$i_{s\alpha}, i_{s\beta}$' )
pl.xticks(np.linspace(0,tend,5))
ax1= pl.subplot(2,1,2)
ax1.plot(Tta, me, 'crimson', lw =3)
# pl.plot(Tta,w, 'peru', lw =2)
# pl.plot(Tta,gamma, 'navy')
ax1.set_ylabel(r'$m_e$' )
ax1.set_xlabel(r'$\omega t$')
ax2 = ax1.twinx()
ax2.plot(Tta,w, 'peru', lw =3)
ax2.set_yticks(np.linspace(-0.5,0.5,5))
ax2.set_ylim(-0.5,0.5)
ax1.set_yticks(np.linspace(-1.2,1.2,7))
ax1.set_ylim(-1.2,1.2)
ax2.set_ylabel(r'$\omega_s$')
ax2.axhline(0, c= 'k')
ax1.set_xlim(0,tend)
ax1.set_xticks(np.linspace(0,tend,5))
# pl.savefig(dirfig + "PMSMSpeedConurrentsvst.pdf", bbox_inches = 'tight', transparent = True)

pl.figure(521, figsize = (6,6))
pl.rc('font', size = 16)
pl.subplot(2,1,1)
pl.plot(Tta,  isd, 'purple', lw =3, label = '$i_{sd}$')
pl.plot(Tta,isq ,'crimson' ,lw =3, label = '$i_{sq}$')
pl.xlim(0,tend)
pl.ylabel(r'$i_{sd}, i_{sq}$' )
pl.xticks(np.linspace(0,tend,5))
pl.yticks(np.linspace(-1.0,1.0,5))
pl.rcParams['legend.fontsize']=12
pl.legend(loc='lower left')
# pl.subplot(2,1,2)
# pl.plot(Tta, psra, 'tomato', lw =2)
# pl.plot(Tta,psrb, 'peru', lw =2)
# pl.ylabel(r'$\psi_{r,\alpha}, \psi_{r,\beta}$' )
pl.xlabel(r'$\omega t$')
# pl.xlim(0,tend)
# pl.xticks(np.linspace(0,tend,5))
# pl.yticks(np.linspace(-1.0,1.0,5))
# pl.savefig(dirfig + "PMSMCurrentconsvst.pdf", bbox_inches = 'tight', transparent = True)

ui = np.where(Tta == 60)
index1 = ui[0][0]
pl.figure(523, figsize = (4,4))
pl.rc('font', size = 16)
pl.plot(isd,isq,'crimson', lw =2, label = 'estimate')
# pl.plot(psra, psrb, 'blue', lw =2 , label = "actual")
pl.arrow(0,0,psird[-1],psirq[-1], fc = 'magenta', ec = 'magenta',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
pl.arrow(0,0,isd[-1],isq[-1], fc = 'crimson', ec = 'crimson',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
pl.arrow(0,0,usd[-1],usq[-1], fc = 'navy', ec = 'navy',  head_width = 0.05\
```

```
        ,length_includes_head = True, lw =3)
pl.arrow(0,0,usd[index1],usq[index1], fc = 'blue', ec = 'blue',  head_width = 0.05\
        ,length_includes_head = True, lw =3)
pl.axhline(0)
pl.axvline(0)
pl.xlim(-1.0,1.0)
pl.xlabel('d-axis')
pl.ylabel('q-axis')
# pl.axis('equal')
# pl.savefig(dirfig + "PMSMXYwCC.pdf", bbox_inches = 'tight', transparent = True)
pl.show()
print(psirm)
```





0.95

Model in stator coordinates, asymmetrical

In [ ]:

PMSM dynamic equations

we will define the parameters in d-q coordinates

$$\frac{di_{sd}}{d\tau} = -\frac{r_s}{l_d} i_{sd} + \omega_s \frac{l_q}{l_d} i_{sq} + \frac{v_{sd}}{l_d}$$

$$\frac{di_{sq}}{d\tau} = -\frac{r_s}{l_q} i_{sq} - \omega_s \frac{l_d}{l_q} i_{sd} - \omega_s \frac{\psi_{r,m}}{l_q} + \frac{v_{sq}}{l_q}$$

$$m_e = \psi_s \times \vec{i}_s = \psi_{r,m} i_{sq} + (l_d - l_q) i_{sd} i_{sq}$$

$$\frac{d\omega_s}{d\tau} = m_e - m_L$$

In [9]:

```
def PMSM_dynstepdq(X,t,params):
    """Defines a function to calculate the derivatives for a 2x2 PMS motor dynamice
       1. Uses state variables stator current $\vec{i}_s$ id d-q coordinates.
       2. Is non-linear and uses the state variable $\omega$ rotor angular velocity to describe
          rotor dynamics. rotor angular velocity will be go from outside as parameter
       3. calls mach_im function to calculate the system equation coefficients
       4. mach_im will in  turn call mach_para function.
       5. uses normalized time 2*np.pi*50*t
       6. This is to be used in a for loop for the time span
    """
    x0 = X[0]  #isd
    x1 = X[1]  #isq
#     x2 = X[2]  #ws
#     x3 = X[3]  #rotor angle
#     wx = X[5]
    u1,u2,w,  rs,ld,lq,tmech,psirm = params #needs state variable ws from outside
    #Convert voltage from stator coordinates into field coordinates
#     ud = u1*np.cos(x3) + u2*np.sin(x3)
#     uq = -u1*np.sin(x3) + u2*np.cos(x3)
    #
    ud = u1
    uq = u2
#     rs,ld,lq,tmech,psirm = mach_parapm(mach_ramu)
#     rs,ld,lq,tmech,psirm = mach_parapm(mach_servodq)
#     mdiff = psirm*x1 + (ld - lq)*x0*x1 - mL
#     print(A[1][0],A[1][1],A[1][2],A[1][3],B[4][0])
    dx0dt = -(rs/ld)*x0 +w*(lq*x1)/ld + ud/ld
    dx1dt = - w*(ld*x0)/lq -(rs/lq)*x1 - w*psirm/lq + uq/lq
#     dx2dt = mdiff/tmech
#     dx3dt = x2
#     dwxdt = rate
    return [dx0dt,dx1dt]
```

In [10]:

```
#Rotor dynamics as mechanics
def rotor_dyndq(X,t,params):
    w = X[0]
    gamma = X[1] #rotor angle

    me,mL,tmech = params
    dwdt = (me-mL)/tmech
    dgammadt = w
    return [dwdt, dgammadt]
```

In [11]:

```
#Function for rotor angle integration
# d\delta/dt = w

def rotorangle(X,t,params):
    del0 = X
    w = params
    ddel0dt = w
    return ddel0dt
```

PMSM for loop dynamic simulation

In [12]:

```
#Preparing simulation for current control
# We will use for loop


# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-3

tend = 20*2*np.pi
tstart = 0.0
delta_t = 0.01
n = 100000
n2 = int(100*2*np.pi/0.01)
ws = -0.5
mL = 0.0
Tta = np.arange(tstart, tend,delta_t)
# Tta = np.linspace(tstart, tend, n2)
#delta_t = Tta[1] - Tta[0]
usa = np.zeros(len(Tta))
usb = np.zeros(len(Tta))
sol1 = np.zeros((len(Tta),2))
sol2 = np.zeros((len(Tta),2))
FCangle = np.zeros(len(Tta))
gamma = np.zeros(len(Tta))
# Field coordinate currents
cosdelta = np.zeros(len(Tta))
sindelta = np.zeros(len(Tta))
isd = np.zeros(len(Tta))
isq = np.zeros(len(Tta))
isa = np.zeros(len(Tta))
isb = np.zeros(len(Tta))
usd = np.zeros(len(Tta))
usq = np.zeros(len(Tta))
eid = np.zeros(len(Tta))
eiq = np.zeros(len(Tta))
w = np.zeros(len(Tta))

rs,ld,lq,tmech,psirm = mach_parapm(mach_ramu)
# rs,ld,lq,tmech,psirm = mach_parapm(mach_servodq)

#Creating reference values for isd and isq
isdrefval = -0.1
isdref = np.zeros(len(Tta))
isqref = np.zeros(len(Tta))

isqref1 = 0.0
isqref2 = 0.8
isqref3 = -0.35
for ii in range(len(Tta)):
    isdref[ii] = 0.0
    if (Tta[ii]>=10*2*np.pi):
        isqref[ii] = isqref3
        isdref[ii] = -isdrefval
    elif(Tta[ii]>=1*2*np.pi):
        isqref[ii]=isqref2
        isdref[ii] = isdrefval
    else:
        isqref[ii]=isqref1
```

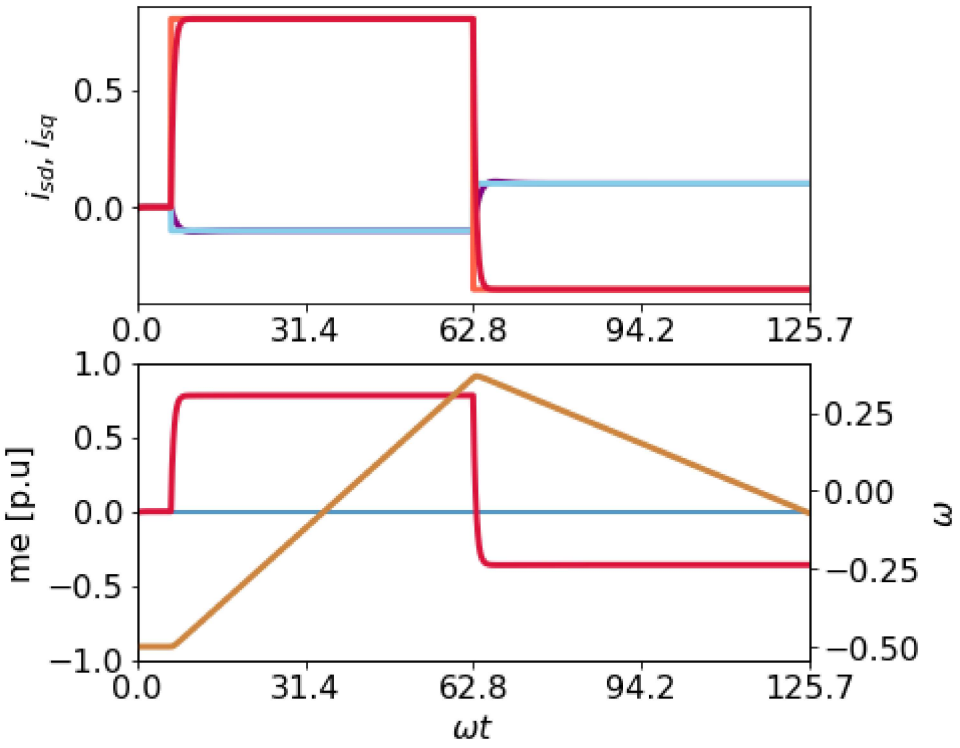
```
psird = np.zeros(len(Tta))
psirq = np.zeros(len(Tta))

me = np.zeros(len(Tta))
x0 = [0.0,0.0]

kpd = ld*1
Tid = 25.0e1

kpq = lq*2
Tiq = 25.0e1

Kparamsd = [kpd,Tid]
Kparamsq = [kpq,Tiq]

#isdref = 0.26
#isqref = 0.75
#The intial value error will never get corrected in
#rotor angle
y0 = [ws,0]

for ii in range(len(Tta)):
    #Start controller after first step
    if ii>=0:
        eid[ii] = isdref[ii-1] - isd[ii-1]
        PIed = [eid[ii],eid[ii-1]]
        usd[ii] = PIcon(PIed, usd[ii-1], delta_t,Kparamsd)
        eiq[ii] = isqref[ii-1] - isq[ii-1]
        PIeq = [eiq[ii],eiq[ii-1]]
        usq[ii] = PIcon(PIeq, usq[ii-1], delta_t,Kparamsq)
#         usd[ii] = 0
#         usq[ii] =.0001
    if ws>=1.0:
        a = 1.0
    else:
        a = ws
#     usa[ii] = a*np.cos(ws*Tta[ii])
#     usb[ii] = a*np.sin(ws*Tta[ii])
#     usa[ii] = usd[ii-1]*cosdelta[ii-1] - usq[ii-1]*sindelta[ii-1]
#     usb[ii] = usq[ii-1]*cosdelta[ii-1] + usd[ii-1]*sindelta[ii-1]

    params = [usd[ii], usq[ii],w[ii], rs,ld,lq,tmech,psirm]
    sol1a = odeint(PMSM_dynstepdq,x0,[0,delta_t], args = (params,),atol = abserr, rtol= relerr)
    sol1[ii]= sol1a[-1]
#     display(sol1a[-1])
    x0 = sol1a[-1]
    me[ii] = 1.0*sol1[ii][1] + (ld -lq)*sol1[ii][0]*sol1[ii][1]
    paramsw = [me[ii],0.0,tmech]
    sol2a = odeint(rotor_dyndq, y0, [0,delta_t], args = (paramsw,), atol = abserr, rtol= relerr)
    y0 = sol2a[-1]
    sol2[ii] = sol2a[-1]
    #Rotor angle using internal angle
    FCangle[ii] = sol2[ii][1]
    gamma[ii] = sol2[ii][1]
    w[ii] = sol2[ii][0]
    #using resolver output
#     FCangle[ii] = sol2[ii]
    cosdelta[ii] = np.cos(FCangle[ii])
    sindelta[ii] = np.sin(FCangle[ii])
    #Coordinate transformation
```

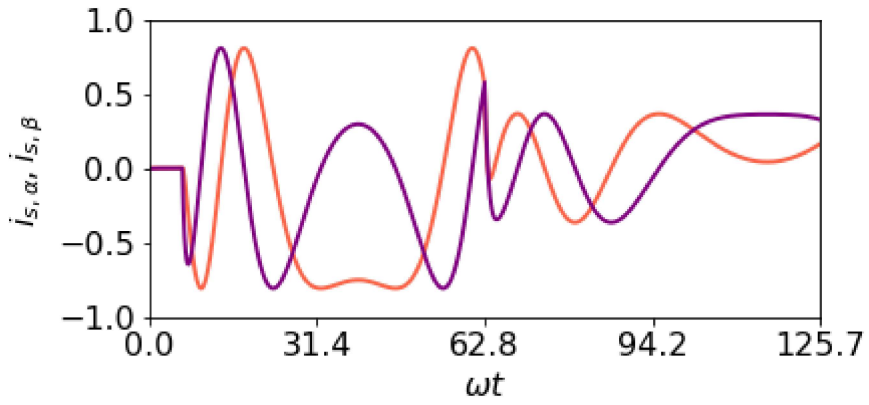
```
    isd[ii] = sol1[ii][0]
    isq[ii] = sol1[ii][1]
    #Convert stator coordinate current to field coordinates
    isa[ii] = sol1[ii][0]*cosdelta[ii] - sol1[ii][1]*sindelta[ii]
    isb[ii] = sol1[ii][1]*cosdelta[ii] + sol1[ii][0]*sindelta[ii]
    #Rotor flux in field coordinates Using estimator output as in practice
    #actual flux will not be available for measurement
    psird[ii] = 1.0
    psirq[ii]= 0.0




# rs,ld,lq,tmech,psirm = mach_parapm(mach_servodq)

# me = 1.0*isq + (ld - lq)*isd*isq
```

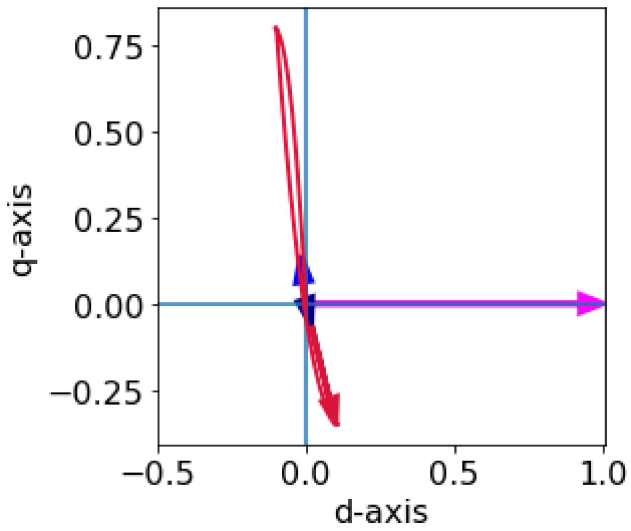
In [13]:

```
dirfig =  "C:/Users/ashwi/Documents/tex/lec/drives/figures/"
pl.figure(501, figsize = (6,6))
pl.rc('font', size = 16)
pl.subplot(2,1,1)
pl.plot(Tta,  isd, 'purple', Tta, isdref,'skyblue', lw=3)
pl.plot(Tta, isqref,'tomato', Tta,isq ,'crimson' ,lw =3)
pl.xlim(0,tend)
pl.ylabel(r'$i_{sd}, i_{sq}$' )
pl.xticks(np.linspace(0,tend,5))
ax1 = pl.subplot(2,1,2)
ax1.axhline(0)
ax1.plot(Tta, me, 'crimson', lw =3)
ax1.set_yticks(np.linspace(-1.0,1.0,5))
ax1.set_ylabel('me [p.u]')
# ax1.plot(Tta,gamma, 'navy', Tta, FCangle)
ax2 = ax1.twinx()
ax2.set_ylabel('$\omega$')
ax2.plot(Tta,w, 'peru', lw =3)

# pl.ylabel(r'$\psi_{r,\alpha}, \psi_{r,\beta}$' )
ax1.set_xlabel(r'$\omega t$')
ax1.set_xlim(0,tend)
ax1.set_xticks(np.linspace(0,tend,5))
# pl.savefig(dirfig + "PMSMdqSpeedConurrentsvst.pdf", bbox_inches = 'tight', transparent = True)


pl.figure(521, figsize = (6,6))
pl.rc('font', size = 16)
# pl.subplot(2,1,1)
# pl.plot(Tta,  psiradash, 'r', Tta,psirbdash ,'purple' ,lw =2)
# pl.xlim(0,tend)
# # pl.ylabel(r'$\psi_{s\alpha}^{\prime}, \psi_{s\beta}^{\prime}$' )
# pl.xticks(np.linspace(0,tend,5))
# pl.yticks(np.linspace(-1.0,1.0,5))
pl.subplot(2,1,2)
pl.plot(Tta, isa, 'tomato', lw =2)
pl.plot(Tta,isb, 'purple', lw =2)
pl.ylabel(r'$i_{s,\alpha}, i_{s,\beta}$' )
pl.xlabel(r'$\omega t$')
pl.xlim(0,tend)
pl.xticks(np.linspace(0,tend,5))
pl.yticks(np.linspace(-1.0,1.0,5))
# pl.savefig(dirfig + "PMSMdqCurrentconsvst.pdf", bbox_inches = 'tight', transparent = True)

pl.figure(523, figsize = (4,4))
pl.rc('font', size = 16)
pl.axhline(0)
pl.axvline(0)
pl.plot(isd,isq,'crimson', lw =2, label = 'estimate')
# pl.plot(psird, psirq, 'blue', lw =2 , label = "actual")
pl.arrow(0,0,psird[-1],psirq[-1], fc = 'magenta', ec = 'magenta',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
pl.arrow(0,0,isd[-1],isq[-1], fc = 'crimson', ec = 'crimson',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
pl.arrow(0,0,usd[-1],usq[-1], fc = 'navy', ec = 'navy',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
pl.arrow(0,0,usd[index1],usq[index1], fc = 'blue', ec = 'blue',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
```

```
# pl.axis('equal')
pl.xlabel('d-axis')
pl.ylabel('q-axis')
pl.xlim(-0.5,1.01)
# pl.savefig(dirfig + "PMSMdqXYwCC.pdf", bbox_inches = 'tight', transparent = True)
pl.show()
print(psirm)
```





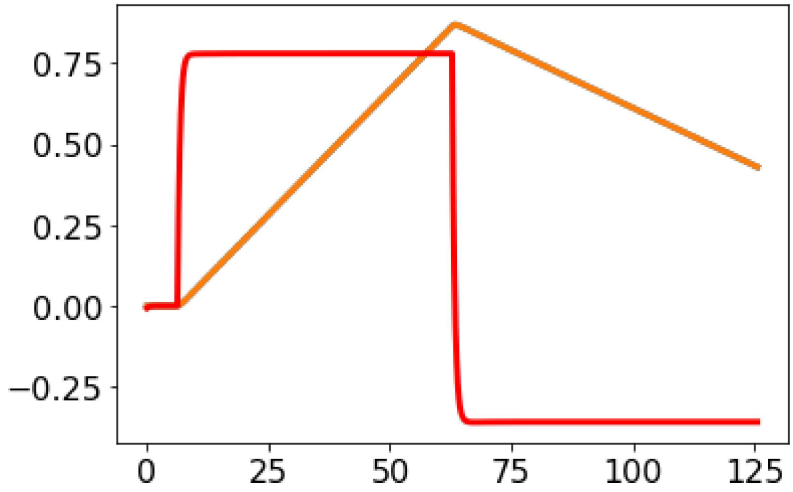


0.9

In [14]:

```
y0 = [0,0]
wxx = np.zeros(len(Tta))
for ii in range(len(Tta)):
    params = [me[ii],0.0,tmech]
    sol2a = odeint(rotor_dyndq, y0, [0,delta_t], args = (params,), atol = abserr, rtol= relerr)
    sol2[ii] = sol2a[-1]
    y0 = sol2a[-1]
    wxx[ii] = sol2[ii][0]

# wxx = sol2[:,0]
pl.plot(Tta,sol2[:,0], Tta, wxx, lw =3)
pl.plot(Tta,me, 'r', lw =3)
# pl.plot(Tta,sol2[:,1], lw =3)

print(ws*lq/ld, ws*ld/lq, 1.0/lq, 1.0/ld, ld, )
```

-0.31112224448897796 -0.8035426731078905 2.3004370830457788 1.43143429716576 0.6986



4x4 PMSM model salient rotor

In [15]:

```
def PMSM_dynstepdq44(X,t,params):
    """Defines a function to calculate the derivatives for a 2x2 PMS motor dynamice
       1. Uses state variables stator current $\vec{i}_s$ id d-q coordinates.
       2. Is non-linear and uses the state variable $\omega$ rotor angular velocity to describe
          rotor dynamics. rotor angular velocity will be go from outside as parameter
       3. calls mach_im function to calculate the system equation coefficients
       4. mach_im will in  turn call mach_para function.
       5. uses normalized time 2*np.pi*50*t
       6. This is to be used in a for loop for the time span
    """
    x0 = X[0]  #isd
    x1 = X[1]  #isq
    x2 = X[2]  #ws
    x3 = X[3]  #rotor angle
#     wx = X[5]
    u1,u2,w,rs,ld,lq,tmech,psirm = params #needs state variable ws from outside
    #Convert voltage from stator coordinates into field coordinates
#     ud = u1*np.cos(x3) + u2*np.sin(x3)
#     uq = -u1*np.sin(x3) + u2*np.cos(x3)
    #
    ud = u1
    uq = u2
#     rs,ld,lq,tmech,psirm = mach_parapm(mach_ramu)
#     rs,ld,lq,tmech,psirm = mach_parapm(mach_servodq)
    mdiff = psirm*x1 + (ld - lq)*x0*x1 - mL
#     print(A[1][0],A[1][1],A[1][2],A[1][3],B[4][0])
    dx0dt = -(rs/ld)*x0 +w*(lq*x1)/ld + ud/ld
    dx1dt = - w*(ld*x0)/lq -(rs/lq)*x1 - w*psirm/lq + uq/lq
    dx2dt = mdiff/tmech
    dx3dt = x2
    return [dx0dt,dx1dt,dx2dt,dx3dt]
```

In [16]:

```
#Preparing simulation for current control
# We will use for loop


# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-3

tend = 20*2*np.pi
tstart = 0.0
delta_t = 0.005
n = 100000
n2 = int(100*2*np.pi/0.01)
ws = -0.5
mL = 0.0
Tta = np.arange(tstart, tend,delta_t)
# Tta = np.linspace(tstart, tend, n2)
#delta_t = Tta[1] - Tta[0]
usa = np.zeros(len(Tta))
usb = np.zeros(len(Tta))
sol1 = np.zeros((len(Tta),4))
sol2 = np.zeros((len(Tta),2))
FCangle = np.zeros(len(Tta))
gamma = np.zeros(len(Tta))
# Field coordinate currents
cosdelta = np.zeros(len(Tta))
sindelta = np.zeros(len(Tta))
isd = np.zeros(len(Tta))
isq = np.zeros(len(Tta))
isa = np.zeros(len(Tta))
isb = np.zeros(len(Tta))
usd = np.zeros(len(Tta))
usq = np.zeros(len(Tta))
eid = np.zeros(len(Tta))
eiq = np.zeros(len(Tta))
w = np.zeros(len(Tta))
w2 = np.zeros(len(Tta))
gamma2 = np.zeros(len(Tta))


rs,ld,lq,tmech,psirm = mach_parapm(mach_ramu)
# rs,ld,lq,tmech,psirm = mach_parapm(mach_servodq)

#Creating reference values for isd and isq
isdrefval = -0.1
isdref = np.zeros(len(Tta))
isqref = np.zeros(len(Tta))

isqref1 = 0.0
isqref2 = 0.8
isqref3 = -0.35
for ii in range(len(Tta)):
    isdref[ii] = 0.0
    if (Tta[ii]>=10*2*np.pi):
        isqref[ii] = isqref3
        isdref[ii] = -isdrefval
    elif(Tta[ii]>=1*2*np.pi):
        isqref[ii]=isqref2
        isdref[ii] = isdrefval
```

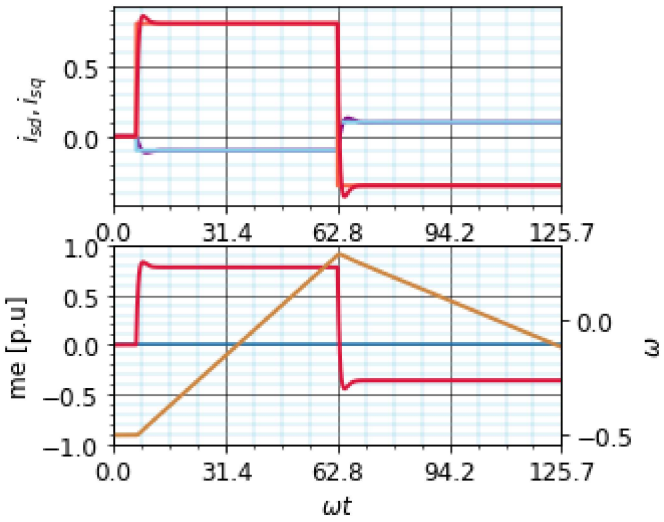
```
    else:
        isqref[ii]=isqref1


psird = np.zeros(len(Tta))
psirq = np.zeros(len(Tta))

me = np.zeros(len(Tta))
x0 = [0.0,0.0,ws,0]

kpd = ld*1
Tid = 25.0e1

kpq = lq*2
Tiq = 25.0e1

Kparamsd = [kpd,Tid]
Kparamsq = [kpq,Tiq]

#isdref = 0.26
#isqref = 0.75
#The intial value error will never get corrected in
#rotor angle
y0 = [ws,0]

for ii in range(len(Tta)):
    #Start controller after first step
    if ii>=0:
        eid[ii] = isdref[ii-1] - isd[ii-1]
        PIed = [eid[ii],eid[ii-1]]
        usd[ii] = PIcon(PIed, usd[ii-1], delta_t,Kparamsd)
        eiq[ii] = isqref[ii-1] - isq[ii-1]
        PIeq = [eiq[ii],eiq[ii-1]]
        usq[ii] = PIcon(PIeq, usq[ii-1], delta_t,Kparamsq)
#         usd[ii] = 0
#         usq[ii] =.0001
    if ws>=1.0:
        a = 1.0
    else:
        a = ws
#     usa[ii] = a*np.cos(ws*Tta[ii])
#     usb[ii] = a*np.sin(ws*Tta[ii])
#     usa[ii] = usd[ii-1]*cosdelta[ii-1] - usq[ii-1]*sindelta[ii-1]
#     usb[ii] = usq[ii-1]*cosdelta[ii-1] + usd[ii-1]*sindelta[ii-1]

    params = [usd[ii], usq[ii],w[ii],rs,ld,lq,tmech,psirm]
    sol1a = odeint(PMSM_dynstepdq44,x0,[0,delta_t], args = (params,),atol = abserr, rtol= relerr)
    sol1[ii]= sol1a[-1]
#     display(sol1a[-1])
    x0 = sol1a[-1]
    me[ii] = 1.0*sol1[ii][1] + (ld -lq)*sol1[ii][0]*sol1[ii][1]
    paramsw = [me[ii],0.0,tmech]
    sol2a = odeint(rotor_dyndq, y0, [0,delta_t], args = (paramsw,), atol = abserr, rtol= relerr)
    y0 = sol2a[-1]
    sol2[ii] = sol2a[-1]
    #Rotor angle using internal angle
    FCangle[ii] = sol1[ii][3]
    gamma[ii] = sol1[ii][3]
    w[ii] = sol1[ii][2]
    #using resolver output
#     FCangle[ii] = sol2[ii]
```

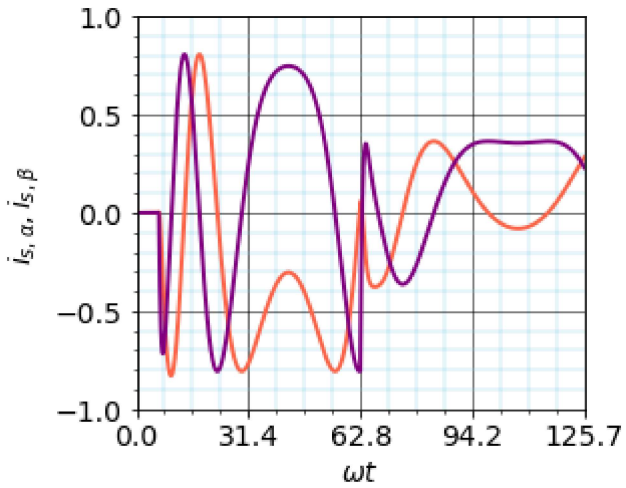
```
    cosdelta[ii] = np.cos(FCangle[ii])
    sindelta[ii] = np.sin(FCangle[ii])
    isd[ii] = sol1[ii][0]
    isq[ii] = sol1[ii][1]
    w2[ii] = sol2[ii][0]
    gamma2[ii] = sol2[ii][1]
    #Coordinate transformation
    #Convert stator coordinate current to field coordinates
    isa[ii] = sol1[ii][0]*cosdelta[ii] - sol1[ii][1]*sindelta[ii]
    isb[ii] = sol1[ii][1]*cosdelta[ii] + sol1[ii][0]*sindelta[ii]
    #Rotor flux in field coordinates Using estimator output as in practice
    #actual flux will not be available for measurement
    psird[ii] = 1.0
    psirq[ii]= 0.0


# isd = sol1[:, 0]
# isq = sol1[:, 1]
# w2 = sol2[:, 0]
# gamma2 = sol2[:,1]



# rs,ld,lq,tmech,psirm = mach_parapm(mach_servodq)

# me = 1.0*isq + (ld - lq)*isd*isq
```

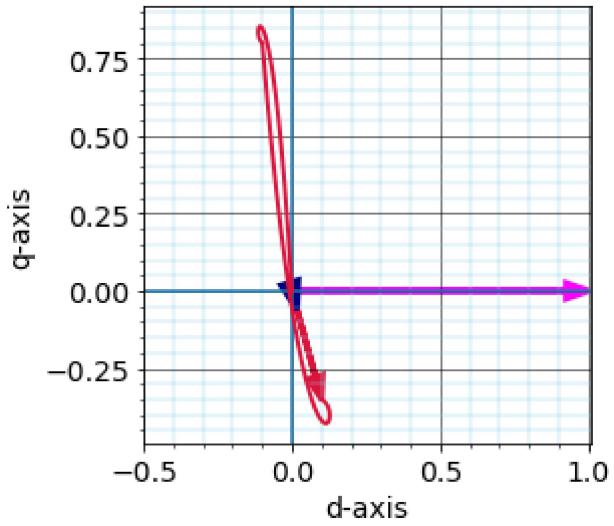
In [17]:

```
dirfig =  "C:/Users/ashwi/Documents/tex/lec/drives/figures/"
pl.figure(501, figsize = (4,4))
pl.rc('font', size = 12)
pl.subplot(2,1,1)
pl.plot(Tta,  isd, 'purple', Tta, isdref,'skyblue', lw=2)
pl.plot(Tta, isqref,'tomato', Tta,isq ,'crimson' ,lw =2)
pl.xlim(0,tend)
pl.ylabel(r'$i_{sd}, i_{sq}$' )
pl.xticks(np.linspace(0,tend,5))
pl.minorticks_on()
pl.grid(which = 'major', lw =0.5, c = 'k')
pl.grid(which = 'minor', lw =0.4, color = 'skyblue')

ax1 = pl.subplot(2,1,2)
ax1.axhline(0)
ax1.plot(Tta, me, 'crimson', lw =2)
ax1.set_yticks(np.linspace(-1.0,1.0,5))
ax1.set_ylabel('me [p.u]')
# ax1.plot(Tta,gamma, 'navy', Tta, FCangle)
ax2 = ax1.twinx()
ax2.set_ylabel('$\omega$')
ax2.plot(Tta,w, 'peru', lw =2)
# ax2.plot(Tta,w2,'brown',lw =2)

# pl.ylabel(r'$\psi_{r,\alpha}, \psi_{r,\beta}$' )
ax1.set_xlabel(r'$\omega t$')
ax1.set_xlim(0,tend)
ax1.set_xticks(np.linspace(0,tend,5))
ax1.minorticks_on()
ax1.grid(which = 'major', lw =0.5, c = 'k')
ax1.grid(which = 'minor', lw =0.4, color = 'skyblue')
# pl.savefig(dirfig + "PMSMdqSpeedConurrentsvst.pdf", bbox_inches = 'tight', transparent = True)


pl.figure(521, figsize = (4,3.6))
pl.rc('font', size = 14)
# pl.subplot(2,1,1)
# pl.plot(Tta,  psiradash, 'r', Tta,psirbdash ,'purple' ,lw =2)
# pl.xlim(0,tend)
# # pl.ylabel(r'$\psi_{s\alpha}^{\prime}, \psi_{s\beta}^{\prime}$' )
# pl.xticks(np.linspace(0,tend,5))
# pl.yticks(np.linspace(-1.0,1.0,5))
pl.subplot(1,1,1)
pl.plot(Tta, isa, 'tomato', lw =2)
pl.plot(Tta,isb, 'purple', lw =2)
pl.ylabel(r'$i_{s,\alpha}, i_{s,\beta}$' )
pl.xlabel(r'$\omega t$')
pl.xlim(0,tend)
pl.xticks(np.linspace(0,tend,5))
pl.yticks(np.linspace(-1.0,1.0,5))
pl.minorticks_on()
pl.grid(which = 'major', lw =0.5, c = 'k')
pl.grid(which = 'minor', lw =0.4, color = 'skyblue')
# pl.savefig(dirfig + "PMSMdqCurrentconsvst.pdf", bbox_inches = 'tight', transparent = True)

pl.figure(523, figsize = (4,4))
pl.rc('font', size = 14)
pl.axhline(0)
pl.axvline(0)
```

```
pl.plot(isd,isq,'crimson', lw =2, label = 'estimate')
# pl.plot(psird, psirq, 'blue', lw =2 , label = "actual")
pl.arrow(0,0,psird[-1],psirq[-1], fc = 'magenta', ec = 'magenta',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
pl.arrow(0,0,isd[-1],isq[-1], fc = 'crimson', ec = 'crimson',  head_width = 0.05\
         ,length_includes_head = True, lw =3)
pl.arrow(0,0,usd[-2],usq[-2], fc = 'navy', ec = 'navy',  head_width = 0.06\
         ,length_includes_head = True, lw =3)
# pl.arrow(0,0,usd[index1],usq[index1], fc = 'blue', ec = 'blue',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)

# pl.axis('equal')
pl.xlabel('d-axis')
pl.ylabel('q-axis')
pl.xlim(-0.5,1.01)
# pl.savefig(dirfig + "PMSMdqXYwCC.pdf", bbox_inches = 'tight', transparent = True)
pl.minorticks_on()
pl.grid(which = 'major', lw =0.5, c = 'k')
pl.grid(which = 'minor', lw =0.4, color = 'skyblue')
pl.show()
print(psirm)
```







0.9