In [1]:

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

In [2]:

```
#Defining Machine parameters
mach_mc = {"rs": 0.1729, "lq": 0.6986, "ld": 0.4347, "tmech":50.5, "psi_rm":0.9}
# mach_mc = {"rs": 0.009, "ld": 4.14, "lq": 4.21, "tmech":509.6, "psi_rm": 0.5}

# mach_mb = {"rs": 0.0185, "rr": 0.0132, "lh": 3.81, "ls": 3.9, "lr": 3.9, "tmech":397.31}

# mach_ma = {"rs": 0.015, "rr": 0.04, "lh": 2.31, "ls": 2.35, "lr": 2.35, "tmech":596.9}

# mach_BM = {"rs": 0.0426, "rr": 0.02113, "lh": 2.252, "ls": 2.252+0.078, "lr": 2.252+0.1052, "tmech
```

The Machine Parameters

We use machine dictionaries to calculate the various parameters used in space vector based machine model This function is called mach_para(dict)

In [3]:

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

       Takes arguement: machine dictionary
       """
    rs = mach["rs"]
    ld = mach['ld']
    lq = mach['lq']
#     ws = mach['ws']
#     sig= 1-(lh*lh)/(lr*ls)
#     kr=lh/lr
#     sigls=sig*ls
#     tr=lr/rr
#     rk=(rs+(kr)*(kr)*rr)
#     tk=sigls/(rs+(kr)*(kr)*rr)
    tmech = mach['tmech']
    psi_rm = mach['psi_rm']
#     print("lh = {0:1.3f}".format(lh))
#     return rs,rr,lh,ls,lr,sig,kr,tr,rk,tk,tmech
    return rs, ld, lq, tmech, psi_rm
```

In [4]:

```
mach_para(mach_mc)
```

Out[4]:

```
(0.1729, 0.4347, 0.6986, 50.5, 0.9)
```

PMSM Dynamics

Building the Dynamic Model of a PMSM in the d-q Coordinate

To model a PMSM, two state variables are chosen, which are i_s and ω_s. The i_s can be further decomposed to i_{sd}, i_{sq} in the permanant magnetic field since it is a complex space vector. The differential equations of these vectors are given as:

$$
\begin{aligned}
\frac{di_{sd}}{d\tau} &= -\frac{r_s}{l_d} i_{sd} + \omega_s \frac{l_q}{l_d} i_{sq} + \frac{v_{sd}}{l_d} \\
\frac{di_{sq}}{d\tau} &= -\omega_s \frac{l_d}{l_q} i_{sd} - \frac{r_s}{l_q} i_{sq} - \omega_s \frac{\psi_{r,m}}{l_q} + \frac{v_{sq}}{l_q} \\
\frac{d\omega_s}{d\tau} &= \frac{1}{\tau_{mech}} [m_e - m_L], m_e = \psi_{r,m} i_{sq} + (l_d - l_q) i_{sd} i_{sq}
\end{aligned}
$$

In [5]:

```
#Induction machine 5x5 dynamic model

def IM_dynstep(X,t,params):
    """Defines a function to calculate the derivatives for a 5x5 Induction 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] #isd
    x1 = X[1] #isq
    ws = X[2] # w
#     x2 = X[2]  #psiralpha
#     x3 = X[3]  #psirbeta
#     x4 = X[4]  #w
#     wx = X[5]
#     u1,u2,m_l,w_s,psi_rm, l_d, l_q = params
    u1,u2,m_l,w_s = params
    rs,l_d,l_q, tmech, psi_rm = mach_para(mach_mc)
#     print("u1:")
#     print(u1)

    A,B,C,D,tm = mach_imstep(mach_mc, params)
#     print("B[0]:")
#     print(B[0])
    m_diff = (psi_rm*x0+(l_d-l_q)*x0*x1) - m_l
    dx0_dt = (A[0][0]*x0 + A[0][1]*x1 + A[0][2]*ws + B[0][0]*u1)
    dx1_dt = (A[0][0]*x0 + A[0][1]*x1 + A[0][2]*ws + B[1][0]*(-w_s*psi_rm+u2))
    dx2_dt = (A[0][0]*x0 + A[0][1]*x1 + A[0][2]*ws + B[2][0]*m_diff)
#     print(A[1][0],A[1][1],A[1][2],A[1][3],B[4][0])
#     dx0dt = (A[0][0]*x0 + A[0][1]*x1 + A[0][2]*x2 + A[0][3]*x3*x4 + A[0][4]*x4 + B[0][0]*u1)
#     dx1dt = (A[1][0]*x0 + A[1][1]*x1 + A[1][2]*x2*x4 + A[1][3]*x3 + A[1][4]*x4 + B[1][0]*u2)
#     dx4dt = (A[4][0]*x0 + A[4][1]*x1 + A[4][2]*x2 + A[4][3]*x3 + A[4][4]*x4 + B[4][0]*mdiff)
#     dx2dt = (A[2][0]*x0 + A[2][1]*x1 + A[2][2]*x2 + A[2][3]*x3*x4 + A[2][4]*x4 + 0)
#     dx3dt = (A[3][0]*x0 + A[3][1]*x1 + A[3][2]*x2*x4 + A[3][3]*x3 + A[3][4]*x4 + 0)
#     dx4dt = (A[4][0]*x0 + A[4][1]*x1 + A[4][2]*x2 + A[4][3]*x3 + A[4][4]*x4 + B[4][0]*mdiff)
#     dwxdt = rate
    return [dx0_dt,dx1_dt,dx2_dt]


def mach_imstep(mc_dict, params):
    """

    Define the induction model that is a 3x3 matrix.
    1. The current components are $i_s\alpha + ji_{s \beta}$.
    2. The rotor flux components are $\psi_{s\alpha} + j\psi_{s\beta}$ and $\omega$
    3. Needs machine dictionary as input
    4. does not use $\omega_s$ in this function, it will be multipied in the derivative function
    """
    u1,u2,m_l,ws = params
#     rs,rr,ld,lq,sig,kr,tr,rk,tk,tmech = mach_para(mc_dict)
    rs,ld,lq, tmech, psi_rm = mach_para(mc_dict)
    a11 = -(rs/ld)
    a12 = ((ws*lq)/ld)
    a13 = 0
#     a14 = (kr/(sig*ls))   #has to be multiplied by omega
#     a15 = 0
```

```
    a21 = -((ld*ws)/lq)
    a22 = -rs/lq
    a23 = 0
#     a24 = a13
#     a25 = 0

    a31 = 0
    a32 = 0
    a33 = 0

    b11 = 1/(ld)
    b21 = 1/(lq)
    b31 = 1/(tmech)
#     b41 = 0
#     b51 = 1/(tmech)


    c11 = 1.0
    c22 = 1.0
    c33 = 1.0
#     c44 = 1.0
#     c55 = 1.0

    A = ([a11,a12,a13],[a21,a22,a23],[a31,a32,a33])
    B = ([b11], [b21], [b31])
    C = ([c11, c22, c33])
    D= ([0,0], [0,0],[0,0],[0,0],[0,0])
#     C=([c11,0,0],[0,c22,0],[0,0,c33])
#     D= ([0,0], [0,0],[0,0],[0,0],[0,0])
#     A=([a11,a12,a13,a14,a15],[a21,a22,a23,a24,a25],[a31,a32,a33,a34,a35],[a41,a42,a43,a44,a45,[a51
#     B=([b11,0],[b21,0],[b31,0],[b41,0],[b51,0])
#     C=([c11,0,0,0,0],[0,c22,0,0,0],[0,0,c33,0,0],[0,0,0,c44,0],[0,0,0,0,a55])
#     D= ([0,0], [0,0],[0,0],[0,0],[0,0])

    return A, B, C, D, tmech
```

In [6]:

```
#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 [7]:

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

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

In [8]:

```
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 [9]:

```
# # testing block:
# TA,TB,TC,TD,tm = mach_imstep(mach_mc, params)
# print(TA, TB, TC, TD, tm)
# # print('kr = {0:3.3f}.'.format(kr))
```

1. System Dynamics without Control

In [24]:

```
#Setting up the simulaton for rotor model
# We will use for loop for simulation
rs, ld, lq, tmech, psi_rm  = mach_para(mach_mc)
# 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]
usd = np.zeros(len(Tta))
usq = np.zeros(len(Tta))
sol1 = np.zeros((len(Tta),3))
# sol2 = np.zeros((len(Tta),2))
# FCangle = 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))
eid = np.zeros(len(Tta))
eiq = np.zeros(len(Tta))
w = np.zeros(len(Tta))

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

# # Initializing 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

me = np.zeros(len(Tta))
x0 = [0,0, ws] # isd, isq, ws
y0 = [1.0,0]

kpd = ld*1
Tid = 25.0e1
```

```
kpq = lq*2
Tiq = 25.0e1

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

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], ws]
    sol1a = odeint(IM_dynstep,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

# for ii in range(len(Tta)):
#     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])
# #     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,ws]
#     sol1a = odeint(IM_dynstep,x0,[0,delta_t], args = (params,),atol = abserr, rtol= relerr)
#     sol1[ii]= sol1a[-1]
# #     display(sol1a[-1])
#     x0 = sol1a[-1]
# #     params2 = [sol1[ii][0], sol1[ii][1],sol1[ii][4]] # isa, isb, w
# #     sol2a = odeint(psirest_vr,y0,[0,delta_t], args = (params2,), atol = abserr, rtol= relerr)
# #     sol2[ii] = sol2a[-1]
# #     psircomp = np.complex(sol2[ii][0], sol2[ii][1])
# #     FCangle[ii] = np.arccos(sol2[ii][0]/np.abs(psircomp))
# #     cosdelta[ii] = sol2[ii][0]/np.abs(psircomp)
# #     sindelta[ii] = sol2[ii][1]/np.abs(psircomp)
# #     #Coordinate transformation
# #     #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
# #     psird[ii] = sol2[ii][0]*cosdelta[ii] + sol2[ii][1]*sindelta[ii]
# #     psirq[ii] = sol2[ii][1]*cosdelta[ii] - sol2[ii][0]*sindelta[ii]
# #     y0 = sol2a[-1]

# isd = sol1[:, 0]
# isq = sol1[:, 1]
# w = sol1[:, 2]
# psra = sol1[:, 2]
# psrb = sol1[:, 3]
# w = sol1[:,4]
# psiradash = sol2[:,0]
# psirbdash = sol2[:,1]

#rs,rr,lh,ls,lr,sig,kr,tr,rk,tk,tmech = mach_para(mach_mb)
rs,l_d,l_q, tmech, psi_rm = mach_para(mach_mc)
me = psi_rm*isq + (l_d-l_q)*isd*isq
```

In [25]:

```
pl.figure(531, 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]')
# t1w = np.pi*2*45
# t2w = np.pi*2*60
# pl.xlim(t1w,t2w)
# # pl.xlim(0,tend)
# pl.axhline(0.75)
# pl.axhline(-0.75)
# pl.ylabel(r'$i_{s\alpha}, \i_{s\beta}$' )
# pl.xticks(np.linspace(t1w,t2w,5))
# pl.yticks(np.linspace(-1.0,1.0,5))
# pl.ylim(-1.0,1.0)
# ax1 = pl.subplot(2,1,2)
# ax1.plot(Tta, isd, 'tomato', lw =3, label = r'$i_{sd}$')
# ax1.plot(t1w+1,0.26,'o', c = 'tomato')
# pl.text(t1w+2,0.28, 'i = {0:1.3f}'.format(isd[-1]), fontsize = 12)
# ax1.plot(Tta,isq, 'purple', lw =3, label = r'$i_{sq}$')
# ax1.plot(Tta,psird, 'magenta', lw=2)
# ax1.set_ylabel(r'$i_{sd}, \i_{sq}$' )
# ax1.set_xlabel(r'$\omega t$')
# ax2 = ax1.twinx()
# ax2.plot(Tta,FCangle, 'olive', lw =2)
# pl.rcParams['legend.fontsize']=12
# ax1.legend(loc = 'upper left')

#pl.axhline(1.0)
#pl.axhline(-1.0)

# t1w = np.pi*2*45
# t2w = np.pi*2*60
# ax1.set_xlim(t1w,t2w)
# ax1.set_xticks(np.linspace(t1w,t2w,5))
# ax1.set_yticks(np.linspace(0,1.0,6))
# ax1.set_ylim(-0.1,1.0)
# ax2.set_ylim(0,3.14)
# pl.savefig(dirfig + "FOCurrentsvst.pdf", bbox_inches = 'tight', transparent = True)

# pl.figure(533, figsize = (4,4))
# pl.rc('font', size = 16)
# pl.rcParams['axes.titlesize']=14
# pl.plot(isd,isq,'crimson', lw =2, label = 'Field currents')
# # pl.plot(psird, psirq, 'blue', lw =4 , label = "actual")
# # pl.axhline(0)
# pl.axvline(0)
# 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)
```

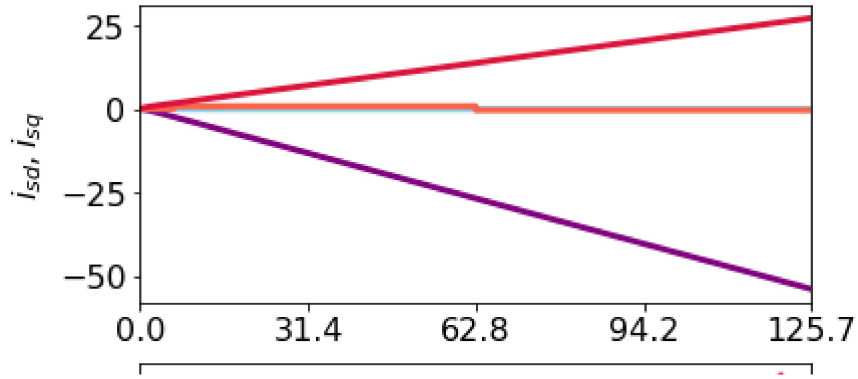
```
# pl.arrow(0,0,isd[-1],0, fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)
# pl.arrow(isd[-1],0,0,isq[-1], fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)
# pl.text(psird[-1]-0.05,psirq[-1]+0.08, r'$\vec{\psi}_r$', fontsize=16 )
# pl.xlabel(r'$d}$')
# pl.ylabel(r'$q$')
# pl.xlim(-0.01,1.0)
# pl.ylim(-0.02,1.0)
# # pl.xticks(np.linspace(-1.0,1.0,5))
# # pl.axis('equal')
# pl.rcParams['legend.fontsize']=9
# # pl.legend(loc = 'lower left', bbox_to_anchor = (0.8,0))
# pl.title('Field Coordinates')
# pl.savefig(dirfig + "FOXYvectors.pdf", bbox_inches = 'tight', transparent = True)


# pl.figure(536, figsize = (4,4))
# pl.rc('font', size = 16)
# pl.rcParams['axes.titlesize']=14
# pl.plot(isd,isq,'crimson', lw =2, label = 'Field currents')
# # pl.plot(psird, psirq, 'blue', lw =4 , label = "actual")
# # pl.axhline(0)
# pl.axvline(0)
# 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,isd[-1],0, fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)
# pl.arrow(isd[-1],0,0,isq[-1], fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)

# pl.xlabel(r'$d}$')
# pl.ylabel(r'$q$')
# # pl.xlim(0,1.0)
# # pl.ylim(-0.02,1.0)
# # pl.xticks(np.linspace(-1.0,1.0,5))
# pl.axis('equal')
# pl.rcParams['legend.fontsize']=9
# # pl.legend(loc = 'lower left', bbox_to_anchor = (0.8,0))
# pl.title('Field Coordinates')
# # pl.savefig(dirfig + "FOXYvectorsFull.pdf", bbox_inches = 'tight', transparent = True)
# pl.show()
# print(isd[-1], isq[-1])
```

Out[25]:

```
Text(0, 0.5, 'me [p.u]')
```



2. System Dynamics with Control

In [19]:

```
#Setting up the simulaton for rotor model
# We will use for loop for simulation
rs, ld, lq, tmech, psi_rm  = mach_para(mach_mc)
# 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]
usd = np.zeros(len(Tta))
usq = np.zeros(len(Tta))
sol1 = np.zeros((len(Tta),3))
# sol2 = np.zeros((len(Tta),2))
# FCangle = 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))
eid = np.zeros(len(Tta))
eiq = np.zeros(len(Tta))
w = np.zeros(len(Tta))

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

# # Initializing 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

me = np.zeros(len(Tta))
x0 = [0,0, ws] # isd, isq, ws
y0 = [1.0,0]

kpd = ld*1
Tid = 25.0e1
```

```
kpq = lq*2
Tiq = 25.0e1

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

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], ws]
    sol1a = odeint(IM_dynstep,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

# for ii in range(len(Tta)):
#     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])
# #     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,ws]
#     sol1a = odeint(IM_dynstep,x0,[0,delta_t], args = (params,),atol = abserr, rtol= relerr)
#     sol1[ii]= sol1a[-1]
# #     display(sol1a[-1])
#     x0 = sol1a[-1]
# #     params2 = [sol1[ii][0], sol1[ii][1],sol1[ii][4]] # isa, isb, w
# #     sol2a = odeint(psirest_vr,y0,[0,delta_t], args = (params2,), atol = abserr, rtol= relerr)
# #     sol2[ii] = sol2a[-1]
# #     psircomp = np.complex(sol2[ii][0], sol2[ii][1])
# #     FCangle[ii] = np.arccos(sol2[ii][0]/np.abs(psircomp))
# #     cosdelta[ii] = sol2[ii][0]/np.abs(psircomp)
# #     sindelta[ii] = sol2[ii][1]/np.abs(psircomp)
# #     #Coordinate transformation
# #     #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
# #     psird[ii] = sol2[ii][0]*cosdelta[ii] + sol2[ii][1]*sindelta[ii]
# #     psirq[ii] = sol2[ii][1]*cosdelta[ii] - sol2[ii][0]*sindelta[ii]
# #     y0 = sol2a[-1]

# isd = sol1[:, 0]
# isq = sol1[:, 1]
# w = sol1[:, 2]
# psra = sol1[:, 2]
# psrb = sol1[:, 3]
# w = sol1[:,4]
# psiradash = sol2[:,0]
# psirbdash = sol2[:,1]

#rs,rr,lh,ls,lr,sig,kr,tr,rk,tk,tmech = mach_para(mach_mb)
rs,l_d,l_q, tmech, psi_rm = mach_para(mach_mc)
me = psi_rm*isq + (l_d-l_q)*isd*isq
```

In [20]:

```
print(isd)
print(isq)
print(me)
```

```
[0.00100436 0.00097342 0.00089229 ... 0.10000011 0.10000011 0.10000011]
[-0.00058619  0.00581419  0.01203562 ... -0.34999994 -0.34999994
 -0.34999994]
[-0.00052742  0.00523128  0.01082922 ... -0.30576344 -0.30576344
 -0.30576344]
```

In [21]:

```
pl.figure(531, 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]')
# t1w = np.pi*2*45
# t2w = np.pi*2*60
# pl.xlim(t1w,t2w)
# # pl.xlim(0,tend)
# pl.axhline(0.75)
# pl.axhline(-0.75)
# pl.ylabel(r'$i_{s\alpha}, \i_{s\beta}$' )
# pl.xticks(np.linspace(t1w,t2w,5))
# pl.yticks(np.linspace(-1.0,1.0,5))
# pl.ylim(-1.0,1.0)
# ax1 = pl.subplot(2,1,2)
# ax1.plot(Tta, isd, 'tomato', lw =3, label = r'$i_{sd}$')
# ax1.plot(t1w+1,0.26,'o', c = 'tomato')
# pl.text(t1w+2,0.28, 'i = {0:1.3f}'.format(isd[-1]), fontsize = 12)
# ax1.plot(Tta,isq, 'purple', lw =3, label = r'$i_{sq}$')
# ax1.plot(Tta,psird, 'magenta', lw=2)
# ax1.set_ylabel(r'$i_{sd}, \i_{sq}$' )
# ax1.set_xlabel(r'$\omega t$')
# ax2 = ax1.twinx()
# ax2.plot(Tta,FCangle, 'olive', lw =2)
# pl.rcParams['legend.fontsize']=12
# ax1.legend(loc = 'upper left')

#pl.axhline(1.0)
#pl.axhline(-1.0)

# t1w = np.pi*2*45
# t2w = np.pi*2*60
# ax1.set_xlim(t1w,t2w)
# ax1.set_xticks(np.linspace(t1w,t2w,5))
# ax1.set_yticks(np.linspace(0,1.0,6))
# ax1.set_ylim(-0.1,1.0)
# ax2.set_ylim(0,3.14)
# pl.savefig(dirfig + "FOCurrentsvst.pdf", bbox_inches = 'tight', transparent = True)

# pl.figure(533, figsize = (4,4))
# pl.rc('font', size = 16)
# pl.rcParams['axes.titlesize']=14
# pl.plot(isd,isq,'crimson', lw =2, label = 'Field currents')
# # pl.plot(psird, psirq, 'blue', lw =4 , label = "actual")
# # pl.axhline(0)
# pl.axvline(0)
# 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)
```

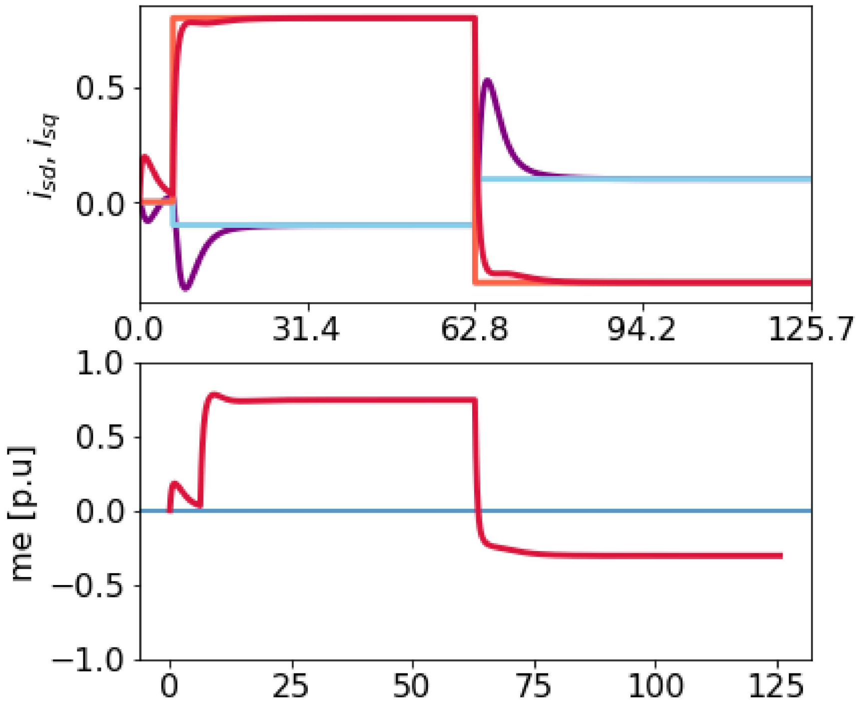
```
# pl.arrow(0,0,isd[-1],0, fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)
# pl.arrow(isd[-1],0,0,isq[-1], fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)
# pl.text(psird[-1]-0.05,psirq[-1]+0.08, r'$\vec{\psi}_r$', fontsize=16 )
# pl.xlabel(r'$d}$')
# pl.ylabel(r'$q$')
# pl.xlim(-0.01,1.0)
# pl.ylim(-0.02,1.0)
# # pl.xticks(np.linspace(-1.0,1.0,5))
# # pl.axis('equal')
# pl.rcParams['legend.fontsize']=9
# # pl.legend(loc = 'lower left', bbox_to_anchor = (0.8,0))
# pl.title('Field Coordinates')
# pl.savefig(dirfig + "FOXYvectors.pdf", bbox_inches = 'tight', transparent = True)


# pl.figure(536, figsize = (4,4))
# pl.rc('font', size = 16)
# pl.rcParams['axes.titlesize']=14
# pl.plot(isd,isq,'crimson', lw =2, label = 'Field currents')
# # pl.plot(psird, psirq, 'blue', lw =4 , label = "actual")
# # pl.axhline(0)
# pl.axvline(0)
# 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,isd[-1],0, fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)
# pl.arrow(isd[-1],0,0,isq[-1], fc = 'coral', ec = 'coral',  head_width = 0.05\
#          ,length_includes_head = True, lw =3)

# pl.xlabel(r'$d}$')
# pl.ylabel(r'$q$')
# # pl.xlim(0,1.0)
# # pl.ylim(-0.02,1.0)
# # pl.xticks(np.linspace(-1.0,1.0,5))
# pl.axis('equal')
# pl.rcParams['legend.fontsize']=9
# # pl.legend(loc = 'lower left', bbox_to_anchor = (0.8,0))
# pl.title('Field Coordinates')
# # pl.savefig(dirfig + "FOXYvectorsFull.pdf", bbox_inches = 'tight', transparent = True)
# pl.show()
# print(isd[-1], isq[-1])
```

Out[21]:

```
Text(0, 0.5, 'me [p.u]')
```



i_{sd}, i_{sq}
0.5
0.0
0.0
31.4
62.8
94.2
125.7
me [p.u]
1.0
0.5
0.0
−0.5
−1.0
0
25
50
75
100
125

In [ ]:

In [ ]: