In [1]:
```
# Alexander Hebert
# ECE 6390
# Computer Project #2
```

In [2]:
```
# Tested using Python v3.4 and IPython v2
```

In [ ]:
```
# Import libraries
```

In [3]:
```
import numpy as np
```

In [4]:
```
import scipy
```

In [5]:
```
import sympy
```

In [6]:
```
from IPython.display import display
```

In [7]:
```
from sympy.interactive import printing
```

In [8]:
```
np.set_printoptions(precision=6)
```

In [9]:
```
#np.set_printoptions(suppress=True)
```

In [ ]:
```
# Original system:
```

In [10]:
```
A = np.loadtxt('A_ex1.txt')
```

In [11]:
```
A
```

Out[11]:
```
array([[ 1.38  , -0.2077,  6.715 , -5.676 ],
       [-0.5814, -4.29  ,  0.    ,  0.675 ],
       [ 1.067 ,  4.273 , -6.654 ,  5.893 ],
       [ 0.048 ,  4.273 ,  1.343 , -2.104 ]])
```

In [12]:
```
n,nc = A.shape
```

In [13]:
```
B = np.loadtxt('B_ex1.txt')
```

In [14]:
```
B
```

Out[14]:
```
array([[ 0.   ,  0.   ],
       [ 5.679,  0.   ],
       [ 1.136, -3.146],
       [ 1.136,  0.   ]])
```

In [15]:
```
nr,m = B.shape
```

In [ ]:
```
# Compute eigenvalues/poles of A to determine system stability:
```

In [16]:
```
A_eigvals, M = np.linalg.eig(A)
```

In [17]:
```
A_eigvals
```

Out[17]:
```
array([ 1.99096 ,  0.063508, -5.056574, -8.665894])
```

In [18]:
```
# Two poles lie in the RHP and are unstable.
```

In [19]:
```
A_eigvals_desired = np.array([-0.2,-0.5,A_eigvals[2],A_eigvals[3]])
```

In [20]:
```
A_eigvals_desired
```

Out[20]:
```
array([-0.2     , -0.5     , -5.056574, -8.665894])
```

In [21]:
```
Lambda = np.diag(A_eigvals_desired)
```

In [22]:
```
Lambda
```

Out[22]:
```
array([[-0.2     ,  0.      ,  0.      ,  0.      ],
       [ 0.      , -0.5     ,  0.      ,  0.      ],
       [ 0.      ,  0.      , -5.056574,  0.      ],
       [ 0.      ,  0.      ,  0.      , -8.665894]])
```

In []:
```
# Pole Assignment Algorithm from journal paper
```

In [23]:
```
# Step A: Decomposition of B using SVD
# B = U*S*V.H
```

In [24]:
```
U, s, VH = np.linalg.svd(B)
```

In [25]:
```
U
```

Out[25]:
```
array([[  6.016779e-18,  -8.304906e-17,  -9.868038e-01,  -1.619203e-01],
       [ -9.460842e-01,  -2.577794e-01,   3.176055e-02,  -1.935609e-01],
       [ -2.628862e-01,   9.648268e-01,  -2.220446e-16,   0.000000e+00],
       [ -1.892501e-01,  -5.156495e-02,  -1.587748e-01,   9.676342e-01]])
```

In [26]:
```
s
```

Out[26]:
```
array([ 5.944254,  3.065158])
```

In [27]:
```
S = np.zeros((4, 2))
S[:2, :2] = np.diag(s)
```

In [28]:
```
S
```

Out[28]:
```
array([[ 5.944254,  0.      ],
       [ 0.      ,  3.065158],
       [ 0.      ,  0.      ],
       [ 0.      ,  0.      ]])
```

In [29]:
```
VH
```

Out[29]:
```
array([[-0.990274,  0.139133],
       [-0.139133, -0.990274]])
```

In [30]:
```
# Extract U_0 and U_1 from matrix U = [U_0,U_1]
```

In [31]:
```
U_0 = U[:n,:m]
```

In [32]:
```
U_0
```

Out[32]:
```
array([[  6.016779e-18,  -8.304906e-17],
       [ -9.460842e-01,  -2.577794e-01],
       [ -2.628862e-01,   9.648268e-01],
       [ -1.892501e-01,  -5.156495e-02]])
```

In [33]:
```
U_1 = U[:n,m:]
```

In [34]:
```
U_1
```

Out[34]:
```
array([[ -9.868038e-01,  -1.619203e-01],
       [  3.176055e-02,  -1.935609e-01],
       [ -2.220446e-16,   0.000000e+00],
       [ -1.587748e-01,   9.676342e-01]])
```

In [35]:
```
# B = [U_0,U_1][Z,0].T
# Compute Z from SVD of B
```

In [36]:
```
Z = np.diag(s).dot(VH)
```

In [37]:
```
Z
```

Out[37]:
```
array([[-5.886439,  0.82704 ],
       [-0.426464, -3.035345]])
```

In [38]:
```
# Compute the nullspace of U_1.T *(A - lambda_j*I)
# for initial eigenvectors in X

X = np.zeros((n,n))

for j in range(len(A_eigvals_desired)):

    lambda_j = A_eigvals_desired[j]

    # M_j is a temp matrix
    exec("M_%d = np.dot(U_1.T,(A - lambda_j*np.identity(n)))" %(j+1))

    # U_1.T *(A - lambda_j*I) = T_j *[Gamma_j,0]*[S_j_hat,S_j].T
    exec("T_%d, gamma_%d, SH_%d = np.linalg.svd(M_%d)" %(j+1,j+1,j+1,j+1))

    exec("X[:,j] = SH_%d[-2,:]" %(j+1))
    # no transpose in SH_j due to 1-d vector

    exec("S_hat_%d = SH_%d[:m,:].T" %(j+1,j+1))
    exec("S_%d = SH_%d[m:,:].T" %(j+1,j+1))
```

In [39]:
```
# Initial eigenvectors in X
X
```

Out[39]:
```
array([[-0.748637, -0.744617, -0.642351, -0.552079],
       [ 0.049467,  0.015566, -0.281814, -0.241829],
       [ 0.521342,  0.540158,  0.703512,  0.797569],
       [ 0.406569,  0.391833,  0.114178, -0.024705]])
```

In [40]:
```
# Test X for full rank
X_rank = np.linalg.matrix_rank(X)
```

In [41]:
```
all((X_rank,n))
```

Out[41]: True

In [42]:
```
# Step X with Method 0

maxiter = 2
v2current = 0
v2prev = np.linalg.cond(X)
eps = 10e-5
flag = 0
X_j = np.zeros((n,n-1))
cond_num = np.zeros((n,1))

for r in range(maxiter):

    for j in range(n):

        X_j = np.delete(X,j,1)
        Q,R = np.linalg.qr(X_j,mode='complete')
        y_j = Q[:,-1].reshape((4,1))

        exec("S_j = S_%d" %(j+1))
        x_j = (S_j.dot(S_j.T).dot(y_j) / np.linalg.norm(np.dot(S_j.T,y_j))
)
        X[:,j] = x_j[:,0]

        cond_num[j,0] = 1 / np.abs(np.dot(y_j.T,x_j))

    v2current = np.linalg.cond(X)

    if ((v2current - v2prev) < eps):
        print("Tolerance met")
        print("v2 = %.3f" %v2current)
        flag = 1

    else:
        v2prev = v2current

if (flag == 0):
    print("Tolerance not met")
    print("v2 = %.3f" %v2current)
```

```
Tolerance met
v2 = 3.361
Tolerance met
v2 = 3.425
```

In [43]:
```
X
```

Out[43]:
```
array([[ 0.575333, -0.783911,  0.379928,  0.540762],
       [ 0.136018,  0.008725, -0.564158,  0.275325],
       [ 0.440119,  0.514012,  0.211629, -0.794813],
       [ 0.675859,  0.348137,  0.70185 ,  0.00671 ]])
```

In [44]:
```
np.linalg.matrix_rank(X)
```

Out[44]: 4

In [45]:
```
X_inv = np.linalg.inv(X)
```

In [46]:
```
X_inv
```

Out[46]:
```
array([[ 0.626346,  1.008495,  0.77749 ,  0.237151],
       [-1.103892,  0.533861, -0.556035,  1.19435 ],
       [-0.051949, -1.241438, -0.460367,  0.593854],
       [-0.380895,  0.573147, -1.309802,  1.061838]])
```

In [47]:
```
# M defined as A + BF
M = X.dot(Lambda).dot(X_inv)
```

In [48]:
```
M
```

Out[48]:
```
array([[ 1.38    , -0.2077  ,  6.715   , -5.676   ],
       [ 0.748373, -4.938714,  1.793084, -0.851045],
       [-2.339352,  5.050211, -8.454491,  6.350361],
       [ 0.314002,  4.143234,  1.70168 , -2.409263]])
```

In [49]:
```
# Eigenvalues of controlled system
M_eigvals, H = np.linalg.eig(M)
M_eigvals
```

Out[49]:
```
array([-8.665894, -0.2     , -0.5     , -5.056574])
```

In [50]:
```
# Compute feedback matrix F
F = np.dot(np.linalg.inv(Z),np.dot(U_0.T,(M - A)))
```

In [51]:
```
F
```

Out[51]:
```
array([[ 0.234156, -0.11423 ,  0.315739, -0.268717],
       [ 1.167309, -0.288295,  0.686322, -0.242411]])
```

In [52]:
```
np.linalg.norm(F)
```

Out[52]:
```
1.4883896339051255
```

In [53]:
```
# Compute condition number norms
```

In [54]:
```
# Inf norm
np.linalg.norm(cond_num,np.inf)
```

Out[54]:
```
1.8211695614459877
```

In [55]:
```
# 2 norm
np.linalg.norm(cond_num)
```

Out[55]:
```
3.2725658611805772
```

In [55]:
```

```