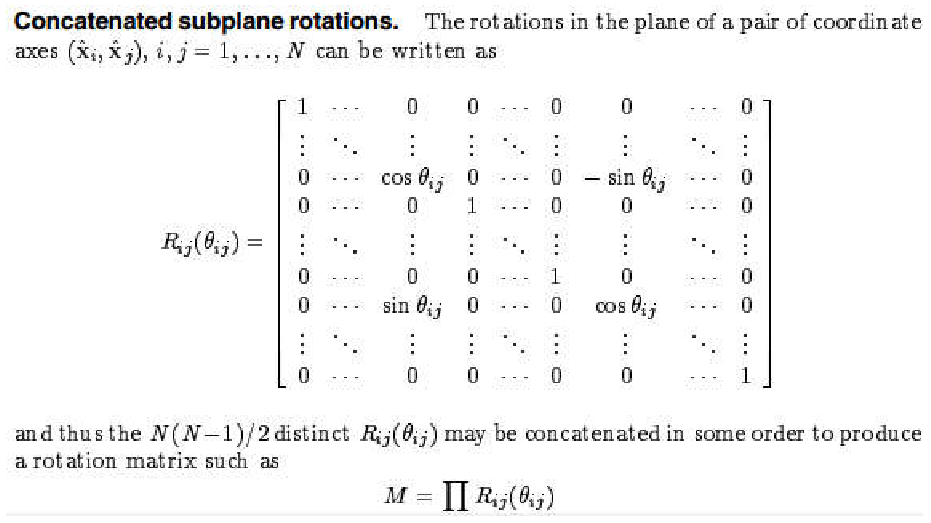
Alexander Hebert
ECE 6390
Computer Project #2

Matlab has a function called *place.m* included in the Controls Toolbox. That function is based off the algorithm(s) in the journal article [1]. The paper describes one main algorithm with three steps: A, X, and F. In step X, the paper describes four methods (0, 1, 2&3). Methods 0 and 1 are similar but separate. Methods 2 and 3 are complementary and produce equivalent results, so they are called method 2/3.

Based on the source code of *place.m* (from edit place in Matlab), that implementation most likely applies method 0 or a variation of it for step X. *Place* may also use some of method 1.

In this project, the journal's algorithm is implemented using method 0 and method 2/3. Some of the results from example 1 in the paper are reproduced to validate the code. The same results for the feedback matrix F were achieved using method 0. Roughly similar results for the feedback matrix F were achieved using method 2/3. All code was written in Python v3.4 using IPython and Numpy. The Python function planeRotation.py was coded for method 2/3 using reference [2]. Python code and results are included in the appendix.

Due to limited time, the complicated algorithm is not explained here. Hopefully, the code is quite readable and clear enough to explain on its own. Two important details are mentioned. In method 0, the nullspace for $P_j = N\{U_1^T(A \quad \lambda_j I)\}$ was computed using SVD. The nullspace vectors correspond to the zero singular values. In method 2/3, the N-dimensional rotation matrix has the following form [2]:



Concatenated subplane rotations. The rotations in the plane of a pair of coordinate axes $(\hat{x}_i, \hat{x}_j)$, $i, j = 1, \ldots, N$ can be written as

$$R_{ij}(\theta_{ij}) = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cos\theta_{ij} & 0 & \cdots & 0 & -\sin\theta_{ij} & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & \sin\theta_{ij} & 0 & \cdots & 0 & \cos\theta_{ij} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}$$

and thus the $N(N-1)/2$ distinct $R_{ij}(\theta_{ij})$ may be concatenated in some order to produce a rotation matrix such as

$$M = \prod R_{ij}(\theta_{ij})$$

Results

Part of the results for example 1 (Chemical reactor) in [1] are listed in Table 1.

Table 1. Results for example 1 (Chemical reactor) in [1]

Number of sweeps	Method	$\|\|c\|\|_{infinity}$	$K_2(X)$	$\|\|c\|\|_2$	$\|\|F\|\|_2$
2	0	1.8212	3.425	3.2726	1.4884
1	2/3	NA	4.936	NA	1.3001

Feedback matrix F for example 1 (Chemical reactor) in [1]:

Method 0 (after 2 sweeps):

F =

```
[[ 0.234156, -0.11423 ,  0.315739, -0.268717],
 [ 1.167309, -0.288295,  0.686322, -0.242411]]
```

Method 2/3 (after 1 sweep):

F =

```
[[ 0.124295, -0.636973, -0.188754,  0.199174],
 [ 0.829187,  0.382232, -0.270522,  0.535619]]
```

References

[1] J. Kautsky, N. K. Nichols, and P. V. Dooren, "Robust Pole Assignment in Linear State Feedback," Int. J. Control, vol. 41, no. 5, pp. 1129-1155, 1985.

[2] A. J. Hanson, "Rotations for N-Dimensional Graphics," TR406.pdf, Indiana University, CS dept., p.7 http://www.cs.indiana.edu/pub/techreports/TR406.pdf

****Extra Matlab results after due date**

```
>> A = load('A_ex1.txt')

A =

    1.3800   -0.2077    6.7150   -5.6760
   -0.5814   -4.2900         0    0.6750
    1.0670    4.2730   -6.6540    5.8930
    0.0480    4.2730    1.3430   -2.1040

>> B = load('B_ex1.txt')

B =

         0         0
    5.6790         0
    1.1360   -3.1460
    1.1360         0

>> A_eigvals_desired = [-0.2   , -0.5   , -5.0566, -8.6659]

A_eigvals_desired =

   -0.2000   -0.5000   -5.0566   -8.6659

>> K =  place(A,B,A_eigvals_desired)

K =

   -0.2441    0.1374   -0.3483    0.2934
   -1.1585    0.3070   -0.6474    0.2050

>> M = A - B*K

M =

    1.3800   -0.2077    6.7150   -5.6760
    0.8049   -5.0701    1.9778   -0.9914
   -2.3002    5.0828   -8.2950    6.2045
    0.3253    4.1169    1.7386   -2.4373

>> [M_evec,M_evals] = eig(M)

M_evec =

   -0.5269   -0.5713   -0.7865    0.4127
   -0.3129   -0.1362    0.0083   -0.5540
    0.7902   -0.4426    0.5121    0.1794
    0.0138   -0.6777    0.3451    0.7004


M_evals =

   -8.6659         0         0         0
         0   -0.2000         0         0
         0         0   -0.5000         0
         0         0         0   -5.0566
```

```
>> cond(M_evec)

ans =

    3.4348

>> norm(K)

ans =

    1.4500

>>
```

Appendix

Python function *PlaneRotationFn.py*

```
import numpy as np

def planeRotation1(size,theta,j,k):
    """Compute the rotation matrix
       in the plane of columns j an k (j < k)
       by angle theta (in radians)."""

    protation = np.identity(size)
    protation[j,j] = np.cos(theta)
    protation[k,j] = np.sin(theta)
    protation[k,k] = np.cos(theta)
    protation[j,k] = -1*np.sin(theta)

    return protation

def planeRotation2(size,cos_theta,sin_theta,j,k):
    """Compute the rotation matrix
       in the plane of columns j an k (j < k)
       by cos_theta and sin_theta."""

    protation = np.identity(size)
    protation[j,j] = cos_theta
    protation[k,j] = sin_theta
    protation[k,k] = cos_theta
    protation[j,k] = -1*sin_theta

    return protation
```