This document was written by Spencer Lyon

Lab 3

Problem 1

In [1]:
```
# Given in lab text, converted to funciton
def create_sq_mat(k):
    """create k x k matrix of ascending ints"""
    return [range(i, i+k) for i in range(0, k**2, k)]
```

In [2]:
```
# list-based function
def mat_mul(a, b):
    """
    Multipliy matrices a and b

    Parameters
    ----------
    a, b : lists
        The square matrices of the same size to be multiplied.
        They should be represented as lists of lists.

    Returns
    -------
    res : lists
        The result of the matrix multiplication

    """
    rows_a = len(a)
    cols_a = len(a[0])
    rows_b = len(b)
    cols_b = len(b[0])

    if cols_a != rows_b:
        raise ValueError('Incompatiable dimensions')

    # Make placeholder list for result
    res = [[0 for row in range(cols_b)] for col in range(rows_a)]
    for i in range(rows_a):
        for j in range(cols_b):
            for k in range(rows_b):
                res[i][j] += a[i][k] * b[k][j]

    return res
```

In [3]:
```
a = create_sq_mat(100)
%timeit mat_mul(a, a)
```

```
1 loops, best of 3: 280 ms per loop
```

In [4]:
```
import numpy as np
b = np.asarray(a)
%timeit np.dot(b, b)
```

```
1000 loops, best of 3: 1.13 ms per loop
```

In [5]:
```
from time import time
import pandas as pd

k_vals = [5, 10, 50, 200, 400]
time_np = []
time_list = []

for k in k_vals:
    b = create_sq_mat(k)
    c = np.arange(k ** 2).reshape(k, k)

    # Do numpy time
    t1 = time()
    np.dot(c, c)
    time_np.append(time() - t1)

    # Do list time
    t1 = time()
    mat_mul(b, b)
    time_list.append(time() - t1)

times = pd.DataFrame([time_np, time_list], columns=k_vals, index=['NumPy', 'List
times.index.name = 'k'
times
```

Out[5]:

	NumPy	Lists
k		
5	0.000022	0.000108
10	0.000008	0.000643
50	0.000145	0.040745
200	0.009628	2.325591
400	0.122684	24.958129

Problem 2

In [6]:
```
a = np.random.randn(50, 1, 3, 1)
b = np.random.randn(50, 1, 3, 1)
c = np.random.randn(1, 42, 3, 7)
d = np.random.randn(7)

case_1 = a + b
case_2 = a / c
case_3 = c * d

msg = 'in 1 shape: %s\nin 2 shape: %s\nout shape: %s'
print('Case 1')
print(msg % (a.shape, b.shape, case_1.shape))

print('\n\nCase 2')
print(msg % (a.shape, c.shape, case_2.shape))

print('\n\nCase 3')
print(msg % (c.shape, d.shape, case_3.shape))
```

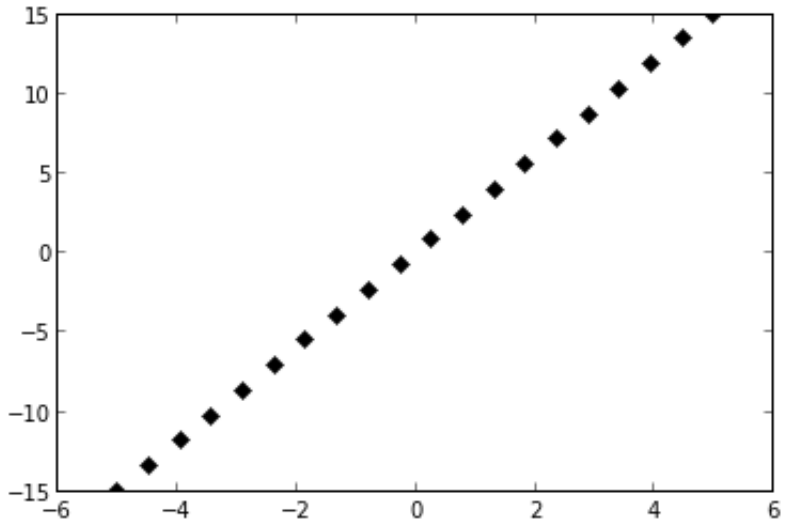
```
Case 1
in 1 shape: (50, 1, 3, 1)
in 2 shape: (50, 1, 3, 1)
out shape: (50, 1, 3, 1)


Case 2
in 1 shape: (50, 1, 3, 1)
in 2 shape: (1, 42, 3, 7)
out shape: (50, 42, 3, 7)


Case 3
in 1 shape: (1, 42, 3, 7)
in 2 shape: (7,)
out shape: (1, 42, 3, 7)
```

Lab 4

Problem 1

In [7]:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-5, 5, 20)
```

```
plt.plot(x, 3 * x, 'kD')
plt.show()
```



Problem 2

In [8]:
```
x = np.arange(1, 7)
print('With numpy.outer')
print(np.outer(x, x))

print('The longer way')
res = np.array([x * i for i in x])
print(res)
```

```
With numpy.outer
[[ 1  2  3  4  5  6]
 [ 2  4  6  8 10 12]
 [ 3  6  9 12 15 18]
 [ 4  8 12 16 20 24]
 [ 5 10 15 20 25 30]
 [ 6 12 18 24 30 36]]
The longer way
[[ 1  2  3  4  5  6]
 [ 2  4  6  8 10 12]
 [ 3  6  9 12 15 18]
 [ 4  8 12 16 20 24]
 [ 5 10 15 20 25 30]
 [ 6 12 18 24 30 36]]
```

Problem 3

In [9]:
```
np.outer(np.arange(1, 6), np.ones(2)).T
```

Out[9]:
```
array([[ 1.,  2.,  3.,  4.,  5.],
       [ 1.,  2.,  3.,  4.,  5.]])
```

Problem 4

In [10]:
```
bucky = np.genfromtxt('../Data/bucky.csv', delimiter=',')
print(bucky.shape)
```

```
(60, 60)
```

Problem 5

In [11]:
```
x = np.random.randn(5, 5)

# x[0] is first row of x
x[0] = [1, 2, 3, 4]
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
ValueError: cannot copy sequence with size 4 to array axis with dimension 5
```

In [12]:
```
y = np.random.randn(5, 6)
np.concatenate((x, y))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-4d383539fa51> in <module>()
      1 y = np.random.randn(5, 6)
----> 2 np.concatenate((x, y))

ValueError: all the input array dimensions except for the concatenation axis
must match exactly
```

Problem 6

In [13]:
```
h = .001
x = np.arange(0, np.pi, h)
approx = np.diff(np.sin(x ** 2)) / h
actual = 2 * np.cos(x ** 2) * x
```

In [14]:
```
np.max(approx - actual)
```

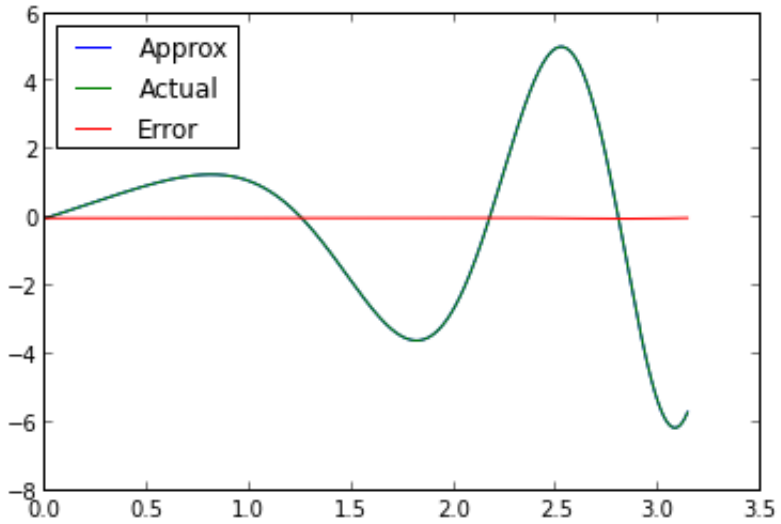
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-14-6d4ff1880ba4> in <module>()
----> 1 np.max(approx - actual)

ValueError: operands could not be broadcast together with shapes (3141) (3142)
```

In [15]:
```
err = approx - actual[:-1]
print(err.max())
```

```
0.00987294960704
```

In [16]:
```
plt.plot(x[:-1], approx, x[:-1], actual[:-1], x[:-1], err)
plt.legend(('Approx', 'Actual', 'Error'), loc=0)
```

Out[16]:
```
<matplotlib.legend.Legend at 0x106c16590>
```



Problem 7

In [17]:
```
t = np.random.rand(10000)
moms = pd.DataFrame([[.5, t.mean()], [1 / np.sqrt(12), t.std(ddof=1)]],
                    columns=['Theory', 'Data'],
                    index=['Mean', 'Std'])
moms
```

Out[17]:

	Theory	Data
Mean	0.500000	0.499425
Std	0.288675	0.287874

In [18]:
```
np.random.rand(50).mean()
```

Out[18]:
```
0.51812069560161922
```

Problem 8

In [19]:
```
from scipy.linalg import inv, lstsq, norm
A = np.random.randn(500, 5)
b = np.random.randn(500)
eq = np.dot(inv(A.T.dot(A)).dot(A.T), b)
least = lstsq(A, b)[0]
diff = norm(eq - least)
print('Error is %.3e' % diff)
```

```
Error is 1.724e-16
```

Lab 5

Problem 1

In [20]:
```
import numpy as np

# First matrix
def flatDiags(n):
    """
    Create n x n upper triangular matrix with values
    on off diagonals equal to diag offset
    """
    x = np.zeros((n, n))
    for i in range(n):
        x +=  np.diagflat(np.ones(n - i) * (i + 1), i)
    return x

flatDiags(5)
```

Out[20]:
```
array([[ 1.,  2.,  3.,  4.,  5.],
       [ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  0.,  1.,  2.,  3.],
       [ 0.,  0.,  0.,  1.,  2.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

In [21]:
```
# Second type
def totalDiags(n):
    x = np.zeros((n,n))
```

```
    for i in range(n):
        x+= np.diagflat(np.ones(n-i)/float((i+1)),i) + np.diagflat(np.ones(n-i)/
    x -= np.diag(np.ones(n),0)
    return x

totalDiags(5)
```

Out[21]:
```
array([[ 1.        ,  0.5       ,  0.33333333,  0.25      ,  0.2       ],
       [ 0.5       ,  1.        ,  0.5       ,  0.33333333,  0.25      ],
       [ 0.33333333,  0.5       ,  1.        ,  0.5       ,  0.33333333],
       [ 0.25      ,  0.33333333,  0.5       ,  1.        ,  0.5       ],
       [ 0.2       ,  0.25      ,  0.33333333,  0.5       ,  1.        ]])
```

Problem 2

In [22]:
```
import scipy.linalg as spla

# First kind with toeplitz and triu
def toep_triu(n):
    return spla.triu(spla.toeplitz(np.arange(1, n + 1, 1)))

toep_triu(5)
```

Out[22]:
```
array([[1, 2, 3, 4, 5],
       [0, 1, 2, 3, 4],
       [0, 0, 1, 2, 3],
       [0, 0, 0, 1, 2],
       [0, 0, 0, 0, 1]])
```

In [23]:
```
# Second kind with toeplitz
def toep_fracs(n):
    return spla.toeplitz(1. / np.arange(1, n + 1))

toep_fracs(5)
```

Out[23]:
```
array([[ 1.        ,  0.5       ,  0.33333333,  0.25      ,  0.2       ],
       [ 0.5       ,  1.        ,  0.5       ,  0.33333333,  0.25      ],
       [ 0.33333333,  0.5       ,  1.        ,  0.5       ,  0.33333333],
       [ 0.25      ,  0.33333333,  0.5       ,  1.        ,  0.5       ],
       [ 0.2       ,  0.25      ,  0.33333333,  0.5       ,  1.        ]])
```

In [24]:
```
def ascending_diag(n):
    return np.diagflat(np.arange(1, n+1, 1))

ascending_diag(5)
```

Out[24]:
```
array([[1, 0, 0, 0, 0],
       [0, 2, 0, 0, 0],
       [0, 0, 3, 0, 0],
       [0, 0, 0, 4, 0]
```

```
[0, 0, 0, 4, 0],
[0, 0, 0, 0, 5]])
```

Problem 3

In [25]:
```
def second_der(n):
    r1 = np.zeros(n)
    r1[0] = 2
    r1[1] = -1
    return spla.toeplitz(r1)

second_der(5)
```

Out[25]:
```
array([[ 2., -1.,  0.,  0.,  0.],
       [-1.,  2., -1.,  0.,  0.],
       [ 0., -1.,  2., -1.,  0.],
       [ 0.,  0., -1.,  2., -1.],
       [ 0.,  0.,  0., -1.,  2.]])
```

Problem 4

In [26]:
```
import scipy.sparse as spar
def second_der_sparse(n, format=None):
    r1 = np.ones(n) * -1
    r2 = np.ones(n) * 2
    r3 = np.ones(n) * -1

    return spar.spdiags([r1, r2, r3], [-1, 0, 1], n, n, format=format)

sparse_2nd = second_der_sparse(5)
sparse_2nd
```

Out[26]:
```
<5x5 sparse matrix of type '<type 'numpy.float64'>'
        with 13 stored elements (3 diagonals) in DIAgonal format>
```

In [27]:
```
sparse_2nd.todense()
```

Out[27]:
```
matrix([[ 2., -1.,  0.,  0.,  0.],
        [-1.,  2., -1.,  0.,  0.],
        [ 0., -1.,  2., -1.,  0.],
        [ 0.,  0., -1.,  2., -1.],
        [ 0.,  0.,  0., -1.,  2.]])
```

Problem 5

In [28]:
```
from scipy.sparse import linalg as sparla
n_list = [50, 100, 300, 500, 1000, 3000]
spar_times = []
den_times = []
for n in n_list:
    A_spar = second_der_sparse(n)
    A = second_der(n)
    b = np.random.randn(n)

    # time sparse
    t1 = time()
    sparla.spsolve(A_spar, b)
    spar_times.append(time() - t1)

    # time sparse
    t1 = time()
    spla.solve(A, b)
    den_times.append(time() - t1)

times = pd.DataFrame([spar_times, den_times],
                     columns=n_list, index=['Sparse', 'Dense']).T
times
```

```
/Users/spencerlyon2/anaconda/lib/python2.7/site-
packages/scipy/sparse/linalg/dsolve/linsolve.py:59: SparseEfficiencyWarning:
spsolve requires CSC or CSR matrix format
  warn('spsolve requires CSC or CSR matrix format', SparseEfficiencyWarning)
```

Out[28]:

	Sparse	Dense
50	0.039803	0.000579
100	0.001207	0.000641
300	0.001133	0.003571
500	0.001297	0.008589
1000	0.001688	0.074200
3000	0.004103	1.710525

Problem 6

In [29]:
```
vals, _ = spla.eig(second_der_sparse(1200).todense())
a = vals.min()
a * 1200 ** 2
```

Out[29]: (9.8531699800200343+0j)

In [30]:
```
# This looks like pi ^ 2
```

```
np.pi ** 2
```

Out[30]: 9.869604401089358

Lab 6

Problem 1

In [31]:
```
import numpy as np
import numpy.linalg as npla

A = np.array([[.75, .5], [.25, .5]])
x = np.array([1, 0])
ans = npla.matrix_power(A, 3).dot(x)

print('The probablility of making the 3rd given he '+
      'made the first is %.3f%%\n' % ans[0])

long_run = npla.matrix_power(A, 100)
print('Long run matrix:\n %s' % long_run)
```

```
The probablility of making the 3rd given he made the first is 0.672%

Long run matrix:
 [[ 0.66666667  0.66666667]
 [ 0.33333333  0.33333333]]
```

We can see that the long run probability of making a free throw is 0.667%

Problem 2

In [32]:
```
# The matrix implied by the transition diagram
A2 = np.array([[.25, 1 / 3., .5],
               [.25, 1 / 3., 1 / 3.],
               [.5, 1 / 3., 1 / 6.]])

x = np.array([1, 0, 0])
ans = npla.matrix_power(A2, 2).dot(x)
print('The probability of being in state B two periods after '+
      'being in state A is %.3f%%\n' % ans[1])
```

```
The probability of being in state B two periods after being in state A is
0.312%
```

In [33]:
```
npla.matrix_power(A2, 50)
```

Out[33]:
```
array([[ 0.35955056,  0.35955056,  0.35955056],
       [ 0.30337079,  0.30337079,  0.30337079],
       [ 0.33707865,  0.33707865,  0.33707865]])
```

Because all the columns of the above matrix are the same, we say that the stable fixed piont is [0.3596, 0.3034, 0.3371]

Problem 3

In [34]:
```
A3 = np.array([[0, 0, 1, 0, 1, 0, 1],
               [1, 0, 0, 0, 0, 1, 0],
               [0, 0, 0, 0, 0, 1 ,0],
               [1, 0, 0, 0, 1, 0, 0],
               [0, 0, 0, 1, 0, 0, 0],
               [0, 0, 1, 0, 0, 0, 1],
               [0, 1, 0, 0, 0, 0, 0]])
```

In [35]:
```
len_5 = npla.matrix_power(A3, 5)
len_7 = npla.matrix_power(A3, 7)
```

In [36]:
```
len_5
```

Out[36]:
```
array([[2, 3, 1, 2, 1, 4, 1],
       [0, 2, 5, 1, 3, 2, 5],
       [0, 1, 2, 0, 1, 1, 2],
       [1, 2, 3, 2, 3, 2, 3],
       [2, 0, 2, 1, 2, 1, 2],
       [1, 2, 1, 1, 0, 3, 1],
       [3, 0, 2, 0, 1, 2, 2]])
```

In [37]:
```
np.where(len_5 == len_5.max())
```

Out[37]:
```
(array([1, 1]), array([2, 6]))
```

There max number of 5 node paths between nodes is 5. They are from node $2 \rightarrow 3$ and from $2 \rightarrow 7$

In [38]:
```
len_7
```

Out[38]:
```
array([[ 2,  6,  9,  4,  6,  7,  9],
       [ 8,  2, 10,  1,  6,  7, 10],
       [ 3,  1,  4,  0,  2,  3,  4],
       [ 6,  3,  9,  3,  7,  6,  9],
       [ 4,  3,  3,  3,  3,  5,  3],
       [ 1,  4,  6,  2,  3,  5,  6],
```

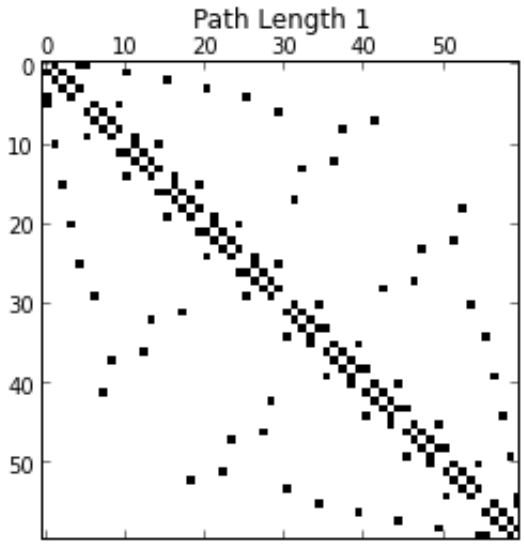
```
       [ 3,  5,  2,  3,  1,  7,  2]])
```

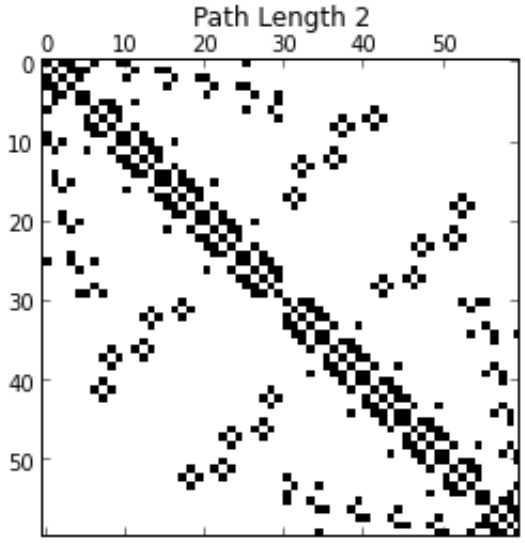
There are no paths from node 3 to node 4

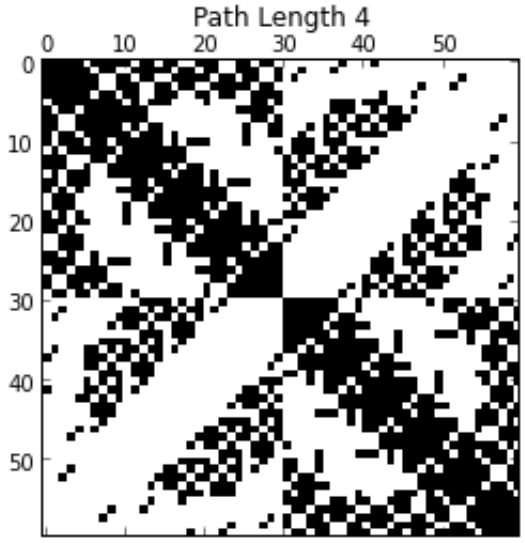
In [39]:
```
bucky = np.genfromtxt('../Data/bucky.csv', delimiter=',')
print('number of connections in bucky: %i' %
      np.count_nonzero(bucky))
msg = 'number %i-node connections in bucky: %i'
print(msg % (2, np.count_nonzero(npla.matrix_power(bucky, 2))))
print(msg % (3, np.count_nonzero(npla.matrix_power(bucky, 3))))
print(msg % (4, np.count_nonzero(npla.matrix_power(bucky, 4))))

done = True
i = 4  # Start where we left off above
while done:
    i += 1
    non_zeros = np.count_nonzero(npla.matrix_power(bucky, i))
    done = non_zeros != bucky.size

print('At path length %i, all atoms are connected' % i)
```

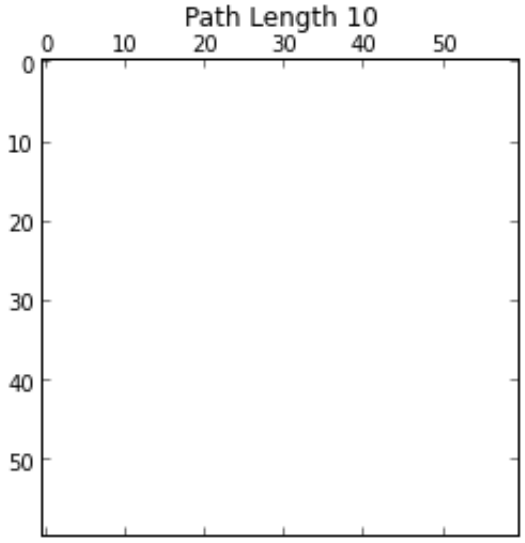
```
number of connections in bucky: 180
number 2-node connections in bucky: 420
number 3-node connections in bucky: 780
number 4-node connections in bucky: 1380
At path length 9, all atoms are connected
```

In [40]:
```
def spy_plot(n):
    plt.figure()
    plt.spy(npla.matrix_power(bucky, n))
    plt.title('Path Length %i' % n)

spy_plot(1)
spy_plot(2)
spy_plot(4)
spy_plot(10)
```




Path Length 2
0
10
20
30
40
50


Path Length 4
0
10
20
30
40
50


Path Length 10
0
10
20
30
40
50

Lab 7

In [41]:
```
import numpy as np


def rowswap(n, j, k):
    """ Swaps two
    INPUTS: n -> matrix size
            j, k -> the two rows to swap
    """
    out = np.eye(n)
    out[j, j] = 0
    out[k, k] = 0
    out[j, k] = 1
    out[k, j] = 1
    return out


def cmult(n, j, const):
    """ Multiplies a row by a constant
    INPUTS: n -> array size
            j -> row
        const -> constant
    """
    out = eye(n)
    out[j, j] = const
    return out


def cmultadd(n, j, k, const):
    """Multiplies a row (k) by a constant and adds the result to
    another row (j)"""
    out = eye(n)
    out[j, k] = const
    return out
```

In [42]:
```
def ref(matrix):
    """
    Put an n x (n+1) matrix in REF.

    Parameters
    ----------
    matrix : array_like, dtype=float, shape=(n, n+1)
        The matrix representing an augmented linear system.

    Returns
    -------
    ref : array_like, dtype=float, shape=(n, n+1)
        The REF of the input matrix

    """
    n = matrix.shape[0]
```

```
    for i in range(0, n):
        for j in range(i, n):
            matrix = np.dot(cmultadd(n, j, i, -matrix[j,i] / matrix[i,i]), matri
    for i in range(0, n):
        matrix[i,:] /= matrix[i,i]
    return matrix
```

In [43]:
```
A = np.array([[4, 5, 6, 3], [2, 4, 6, 4], [7, 8, 0, 5]])
```

In [44]:
```
ref(A)
```

Out[44]:
```
array([[ 1.        ,  1.25      ,  1.5       ,  0.75      ],
       [-0.        ,  1.        ,  2.        ,  1.66666667],
       [ 0.        ,  0.        ,  1.        , -0.11111111]])
```

Problem 2

In [45]:
```
def LUDecomp(mat):
    """
    Perform LU decomposition of a matrix using only elementary matrices
    """
    n = mat.shape[0]
    EL = []
    L = np.eye(n)
    U = mat
    # Construct all type 3 matricies
    for col in range(0, n):
        for row in range(col + 1, n):
            E = cmultadd(n, row, col, (-U[row, col] / U[col, col]))
            E1 = cmultadd(n, row, col,  U[row, col] / U[col, col])
            U = np.dot(E, U)
            EL.append(E1)

    # Construct all type 1 matrcies.
    for j in range(0, n):
        E = cmult(n, j, 1 / U[j, j])
        E1 = cmult(n, j, U[j, j])
        U = np.dot(E, U)
        EL.append(E1)

    for i in EL:
        L = np.dot(L, i)

    return [L, U]
```

In [46]:
```
A2 = np.random.randn(5, 5)
L, U = LUDecomp(A2)
```

```
np.allclose(A2, L.dot(U))
```

Out[46]: True

Problem 3

In [47]:
```
def detLU(mat):
    """
    Use the LUDecomp function above to find the determinant of a matrix
    """
    L, U = LUDecomp(mat)
    sumL = 1
    sumU = 1
    for i in range(0, L.shape[0]):
        sumL *= L[i, i]
        sumU *= U[i ,i]
    return sumL * sumU
```

In [48]:
```
np.allclose(detLU(A2), npla.det(A2))
```

Out[48]: True

In [ ]: