Midterm I: Take-Home Programming

PART A

```
import numpy as np
import math as math
from scipy.linalg import lu

#return max of sum of columns
def computeL1norm(A):
    column_sums = [sum([math.fabs(row[i]) for row in A]) for i in range(0,len(A[0]))]
    max = column_sums[0]
    for i in range(len(column_sums)):
        if(i > max):
            max = i
    return max

def solve_lower(U,v):
    m=U.shape[0]
    n=U.shape[1]
    c = [0 for i in range(0,n)]

    if(m!=n):
        print('Matrix is not square')
        return
    for j in range(0,n):
        if(U[j,j] == 0):
            print('Matrix is singular')
            return
        v[j] = max(math.fabs(c[j]-1/U[j,j]),math.fabs(c[j]+1/U[j,j]))
        for i in range(j+1,n):
            c[i] = c[i] - U[i,j]*v[j]

def error(exact, approx):
    return ((exact - approx)/exact) * 100

def main():
    #first matrix
    A1 = np.array([[10.,-7.,0.],[5.,-1.,5.],[-3.,2.,6.]])
    normA1 = computeL1norm(A1)

    P,L,U = lu(A1,permute_l=False)
    U_t = U.transpose()
    L_t = L.transpose()
    A_t = U_t.dot(L_t)
```

```
    n = U_t.shape[1]

    v = np.zeros(n)
    y = np.zeros(n)
    z = np.zeros(n)

    solve_lower(U_t, v)
    y = np.linalg.solve(L_t, v)
    z = np.linalg.solve(A1, y)

    k = (np.linalg.norm(z,1)/np.linalg.norm(y,1)) * normA1

    print('real condition number for A1: ', np.linalg.cond(A1,1))
    print('computed condition number for A1: ', k)
    print('error for A1: ', error(np.linalg.cond(A1,1), k), '%')

    A2 = np.array([[92.,66.,25.],[-73.,78.,24.],[-80.,37.,10.]])
    normA2 = computeL1norm(A2)

    P,L,U = lu(A2,permute_l=False)
    U_t = U.transpose()
    L_t = L.transpose()

    n = U_t.shape[1]

    v = np.zeros(n)
    y = np.zeros(n)
    z = np.zeros(n)

    solve_lower(U_t, v)

    y = np.linalg.solve(L_t, v)
    z = np.linalg.solve(A2, y)

    k = (np.linalg.norm(z,1)/np.linalg.norm(y,1)) * normA2

    print('real condition number for A2: ', np.linalg.cond(A2,1))
    print('computed condition number for A2: ', k)
    print('error for A2: ', error(np.linalg.cond(A2,1), k), '%')

if __name__ == "__main__":
    main()
```

The results of part a show that this method of computing the condition number results in fairly accurate results,with a max error of 36% for A1 an 20% for A2.

When I remove the **math.fabs()** requirement for magnitude from the equality, the results for A1 are even more accurate, with an error of 9.97%.

PART B

```
import numpy as np
import math as math

#return max of sum of columns
def computeL1norm(A):
    column_sums = [sum([math.fabs(row[i]) for row in A]) for i in range(0,len(A[0]))]
    max = column_sums[0]
    for i in range(len(column_sums)):
        if(i > max):
            max = i
    return max

def main():
    A1 = np.array([[10.,-7.,0.],[5.,-1.,5.],[-3.,2.,6.]])
    A2 = np.array([[92.,66.,25.],[-73.,78.,24.],[-80.,37.,10.]])
    normA1 = computeL1norm(A1)
    normA2 = computeL1norm(A2)

    y1A1 = np.random.rand(3,1)
    y2A1 = np.random.rand(3,1)
    y3A1 = np.random.rand(3,1)
    y4A1 = np.random.rand(3,1)
    y5A1 = np.random.rand(3,1)

    z1A1 = np.linalg.solve(A1,y1A1)
    z2A1 = np.linalg.solve(A1,y2A1)
    z3A1 = np.linalg.solve(A1,y3A1)
    z4A1 = np.linalg.solve(A1,y4A1)
    z5A1 = np.linalg.solve(A1,y5A1)

    r1A1 = np.linalg.norm(z1A1,1)/np.linalg.norm(y1A1,1)
    r2A1 = np.linalg.norm(z2A1,1)/np.linalg.norm(y2A1,1)
    r3A1 = np.linalg.norm(z3A1,1)/np.linalg.norm(y2A1,1)
    r4A1 = np.linalg.norm(z4A1,1)/np.linalg.norm(y4A1,1)
    r5A1 = np.linalg.norm(z5A1,1)/np.linalg.norm(y5A1,1)

    y1A2 = np.random.rand(3,1)
    y2A2 = np.random.rand(3,1)
    y3A2 = np.random.rand(3,1)
    y4A2 = np.random.rand(3,1)
```

```
    y5A2 = np.random.rand(3,1)

    z1A2 = np.linalg.solve(A2,y1A2)
    z2A2 = np.linalg.solve(A2,y2A2)
    z3A2 = np.linalg.solve(A2,y3A2)
    z4A2 = np.linalg.solve(A2,y4A2)
    z5A2 = np.linalg.solve(A2,y5A2)

    r1A2 = np.linalg.norm(z1A2,1)/np.linalg.norm(y1A2,1)
    r2A2 = np.linalg.norm(z2A2,1)/np.linalg.norm(y2A2,1)
    r3A2 = np.linalg.norm(z3A2,1)/np.linalg.norm(y2A2,1)
    r4A2 = np.linalg.norm(z4A2,1)/np.linalg.norm(y4A2,1)
    r5A2 = np.linalg.norm(z5A2,1)/np.linalg.norm(y5A2,1)

    kA1 = max(r1A1,r2A1,r3A1,r4A1,r5A1)*normA1
    kA2 = max(r1A2,r2A2,r3A2,r4A2,r5A2)*normA2

    print('condition number for A1: ', kA1)
    print('condition number for A2: ', kA2)

if __name__ == "__main__":
    main()
```

This method of computing the condition number is fairly *inaccurate*, with varied results each time the code is ran. Eventually, the accuracy can be reached, but it may take a while before it does due to the randomness of the arrays.