MA 544 Programming Assignment 2

Bring your questions on the discussion board for Module 3 for helpful hints on these question.

```
import numpy as np
```

Question 1: Iterative Methods for Linear Systems

Consider the following linear system

$$\begin{pmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 25 \\ -11 \\ 15 \end{pmatrix}$$

Jacobi Method

Initialize the iterative solution vector $x^{(0)}$ randomly, or with the zero vector,

for k=0:maxIteration, update every element until convergece

for i=1:n

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right).$$

```
# You can modify this code to answer the following
'''
Jacobi's iteration method for solving the system of equations Ax=b.
p0 is the initialization for the iteration.
'''
def jacobi(A, b, p0, tol, maxIter=100):
    n=len(A)
    p = p0

    for k in range(maxIter):
        p_old = p.copy() # In python assignment is not the same as
copy

        # Update every component of iterant p
        for i in range(n):
            sumi = b[i];
            for j in range(n):
                if i==j: # Diagonal elements are not included in
Jacobi
                    continue;
                sumi = sumi - A[i,j] * p_old[j]
            p[i] = sumi/A[i,i]

        rel_error = np.linalg.norm(p-p_old)/n # Actually 'n' should be
```

```python
replace by norm of p
        # print("Relative error in iteration", k+1,":",rel_error)
        if rel_error<tol:
            print("TOLERANCE MET BEFORE MAX-ITERATION")
            break
    return p;
```

```python
# Example System
A1 = np.array([[10, -1, 2, 0],
              [-1, 11, -1, 3],
              [2, -1, 10, -1],
              [0, 3, -1, 8]],dtype=float)
b = np.array([6, 25, -11, 15],dtype=float)
```

```python
# Solved by using Jacobi Method
x = jacobi(A1,b, np.array([0,0,0,0],dtype=float),0.0000001, 100)
print("The solution is: ",x)
```

```
TOLERANCE MET BEFORE MAX-ITERATION
The solution is:  [ 1.00000003  1.99999996 -0.99999997  0.99999995]
```

(A) **Modify** the code for Jacobi Method to implement the Gauss-Siedel Iteration in Python. Solve the above system by using this method. Exact answer is (1,2,-1,1). Stopping criteria could be a relative error $\delta < 0.000001$.

```python
# Your Code here
```

(B) Successive overrelaxation (SOR) is another iterative method for solving linear systems. It picks up the next iteration from a weighted sum of the current iteration and the next iteration by Gauss-seidel. **Modify** the code for Jacobi Method to implement the SOR method in Python and solve the above system again with $\omega = 1.5$. Display the solution of the above system by this method.

```python
# Your work starts here
```

Question 2: Gaussian Elimination with Pivoting

```python
## Gaussian Elimination: Scaled Row Pivoting
## This function is based on the pseudo-code on page-148 in the Text
by Kincaid and Cheney
def GE_srpp(X, verbose=False):
    '''
    This function returns the P'LU factorization of a square matrix A
    by scaled row partial pivoting.
    In place of returning L and U, elements of modified A are used to
hold values of L and U.
    '''
    A = np.copy(X)
    m,n = A.shape
    swap=0;

    # The initial ordering of rows
```

```
    p = list(range(m))
    if verbose:
        print("permutation vector initialized to: ",p)

    # Scaling vector: absolute maximum elements of each row
    s = np.max(np.abs(A), axis=1)

    # Start the k-1 passes of Guassian Elimination on A
    for k in range(m-1):
        if verbose:
            print("Scaling Vector: ",s)
        # Find the pivot element and interchange the rows
        pivot_index = k + np.argmax(np.abs(A[p[k:], k])/s[p[k:]])


        # Interchange elements in the permutation vector if needed
        if pivot_index !=k:
            temp = p[k]
            p[k]=p[pivot_index]
            p[pivot_index] = temp
            swap+=1;

        if verbose:
            print("\nPivot Element: {0:.4f} \n".format(A[p[k],k]))
        if np.abs(A[p[k],k]) < 10**(-20):
             sys.exit("ERROR!! Provided matrix is singular or there is a zero pivot.")
        # Check the new order of rows
        if verbose:
            print("permutation vector: ",p)
        # For the k-th pivot row Perform the Gaussian elimination on the following rows
        for i in range(k+1, m):
            # Find the multiplier
            z = A[p[i],k]/A[p[k],k]
            #Save the multiplier z in A itself. You can save this in L also
            A[p[i],k] = z
            #Elimination operation: Changes all elements in a row simultaneously
            A[p[i],k+1:] = A[p[i],k+1:] - z*A[p[k],k+1:]

        if verbose:
            print("\n After PASS {}=========: \n".format(k+1), A)
    return A, p, swap
```

LU Decomposition Example

The above code could be used or modified for a umber of purposes. Here is how it coudl be sed for $PA=LU$ decomposition.

```python
A2 = np.array([[5, 4, 7, 6, 9],
               [7, 8, 9, 9, 8],
               [2, 3, 5, 9, 8],
               [3, 1, 7, 5, 6],
               [9, 1, 3, 7, 3]], dtype=float)

newA,p,swaps = GE_srpp(A2)
print("Modified A after Gaussian elimination:\n",newA)
U=np.triu(newA[p,:])
L=np.tril(newA[p,:], -1)+np.eye(5)
P=np.eye(5)[p,:]
print("\n Upper triangular, U:\n ", U)
print("\n Lower triangular, L:\n", L)
print("\n The Permutation Matrix, P:\n",P)
print("Sanity check: Norm of LU-PA (must be close to
zero)=",np.linalg.norm(P@A2-L@U))
```

```
Modified A after Gaussian elimination:
 [[ 0.55555556  0.47692308  0.4        -0.09795479  3.2007535 ]
 [ 0.77777778  7.22222222  6.66666667  3.55555556  5.66666667]
 [ 0.22222222  0.38461538  0.32857143  5.30857143  3.68285714]
 [ 0.33333333  0.09230769  5.38461538  2.33846154  4.47692308]
 [ 9.          1.          3.          7.          3.        ]]

 Upper triangular, U:
  [[9.         1.         3.         7.         3.        ]
 [0.         7.22222222 6.66666667 3.55555556 5.66666667]
 [0.         0.         5.38461538 2.33846154 4.47692308]
 [0.         0.         0.         5.30857143 3.68285714]
 [0.         0.         0.         0.         3.2007535 ]]

 Lower triangular, L:
 [[ 1.          0.          0.          0.          0.        ]
 [ 0.77777778  1.          0.          0.          0.        ]
 [ 0.33333333  0.09230769  1.          0.          0.        ]
 [ 0.22222222  0.38461538  0.32857143  1.          0.        ]
 [ 0.55555556  0.47692308  0.4        -0.09795479  1.        ]]

 The Permutation Matrix, P:
 [[0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0.]]
Sanity check: Norm of LU-PA (must be close to zero)= 0.0
```

(A) Modify the code for Gaussian elimination to write a function that solves a linear system $Ax = b$. Test this on the following system. Display the verbose output and the solution.

$$
\begin{aligned}
3x-5y+z & \text{¿}0\\
x+2y+3z & \text{¿}1\\
-2x+3y-4z & \text{¿}3
\end{aligned}
$$

```
# Your work starts here
```

(B) Modify this code to find the determinant of any square matrix A. Note that

$$PA=LU \Rightarrow \det A=\pm \det U.$$

The sign depends of the number of row-swaps in the elimination process. Use this code to find the determinant of any 10×10 matrix that you randomly generate. Compare your result with the built-in NumPy method.

```
# Your work starts here
```

(C) Modify your system-solver to find the inverse of a square matrix. Use this code to display the inverse of the matrix

$$A=\begin{pmatrix} 3 & -5 & 1\\ 1 & 2 & 3\\ -2 & 3 & -4\end{pmatrix}.$$

```
# Your work starts here
```

Questions 3: Gradient Descent

Modify the code provided for gradient descent to find the minimum for a function in two variables. Show the output for the function

$$f(x_1,x_2)=x_1^2+x_2^2-2x_1+4x_2+8$$

```
# Your work starts here
```