LAB ACTIVITY 3
CHL711
Linear Algebraic Equations

Karan Singla
2012CH10094

1 Introduction

Linear Algebraic Equations is the fundamental of linear algebra, which is an important part of all engineering, physics, mathematics, computer science as well economic studies. The fact that a non-linear system of equations is often approximated by a linear system makes their study inevitable. An attempt has been made to obtainin quick and accurate solutions with minimum errors through a program written in python.

2 The Gauss Elimation Method and The Gauss Siedel Method

Out of the several methods to solve linear algebraic equations, two ways have been used in the program to solve the linear algebric equations. They are the Gauss elimation method and the Gauss Sidel Iterative method.

In the Gauss elimination method, the Matrix containing the coefficients of the variables is triangularized and subsequently, the variables are obtained by back substitution.

On the other hand, the Gauss Sidel Method is an iterative Scheme, in which an initial guess is made and the solution is refined upon suscessive iterations. It is faster than the elimation method, but it's convergability needs to be checked before applying it.

It has been checked whether the iterative method will converge or diverge using the condition for convergence that the matrix must be diagnolly dominant. If it is found to be converging, the iterative method has been used, else the gauss elimination method has been used. This limitation on the iterative scheme can be removed to a great extent by using L-U decomposition.

3 Checking for consistency

If the Gauss Siedel method is applicable, the matrix would be diagnolly dominant and hence there is no need to check for consistency of the system. Otherwise, a check has been placed on the consistency by calculating the determinant of the matrix after triangularing it. If the determinant is extremely small, the system will give no solution or meaningless results.

4 Errors and their elimination

4.1 Pivoting

It can be seen through simulation of a simple system of equations (even 2 equations are sufficient), that if pivoting is not done, equations with pivot coefficients smaller than around 10^{-6} give a wrong output according to the set tolerance of 10^{-8}, but this analysis varies depending on the number of equations we are looking at.
Pivoting is an extremely useful technique to get accurate solutions. The residual terms reduce significantly when pivoting is used. The output, obtained using either iterative method or has been checked, by calculating the residues and compared against three values of tolerances which are 10^{-8}, 10^{-4}, 10^{-2}.

4.2 Ill-conditioned Equations

If even upon applying complete pivoting, the pivot term is very small, the system of equations is said to be ill-conditioned. The Gauss siedel method, if applicable, provides a much better solution in these cases.

Another kind of ill-conditioned system is possible, when a slight change in coefficients changes the answer by a huge value. The program, in it's current stage does not keep a check for such systems.

4.3 Build up of errors

In the gauss elimination method, errors might build up during triangularizing the matrix, but they are taken care of to a great extent by the pivoting. Another possibility of build-up is during the back substitution steps, in which the errors add up as we move up the matrix calculating the values of x.

The results are reported in a output file, which gives the method used, the values of variables and fractional tolerances of each equation. If the error is greater than 1 percent, a warning is given to the user that the calculated values may be wrong. Through this, the user can know the equation which is producing a greater error, which will help in further analysis of the situation at hand.

5 Further Work

The program can be extended further to the tackling of ill-conditioned systems, which result due to errors in input handling. Also, in case there are errors in solutions(which the program gives precisely), the source code may be extended to determine the variables which have the maximum error in their calculations, since it may be the case, that the values of only certain variables are determined incorrectly, especially those which have small diagnol elements after triangularization.

6 Appendix

6.1 Python Code

```python
import copy
def main():
        a,b=input_module()
        a1,b1,n,pivot_zero,tol1,tol2,tol3,converge=setup(a,b)
        consistent=True
        if (converge==True):
                x=work1(n,a1,b1)

        else:
                x=work(n,a,b,pivot_zero)
                if (x[0]=='x'):
                        consistent=False
                        tolerances=[]
                        output(x,tolerances,converge,consistent)
                        return 0
        tolerances=analysis(n,a1,b1,x,tol1,tol2,tol3)
        output(x,tolerances,converge,consistent)

def input_module():
        a=[]
        b=[]

        #first line must have no. of equations
        # each of the next n lines must have rows of matrix a
        # next n lines must have values of marix b
        input1=open('input','r')
        n=int(input1.readline())

        for i in range(0,n):
                a.append([])
                string=input1.readline()
                a[i]=[float(x) for x in string.split()]
        for i in range(0,n):
                string=input1.readline()
                b.append(float(string))
        input1.close()

        return (a,b)


def setup(a,b):
        a1=copy.deepcopy(a)
        b1=copy.deepcopy(b)
        n=len(a)
        pivot_zero=10**(-6)    #for pivot element
        tol1=10**(-8)
        tol2=10**(-4)
        tol3=10**(-2)

        #Check if the iterative method will diverge or converge
        converge=True
        flag=0
        for i in range(0,n):
                flag2=0
                for j in range(0,n):
```

```
            if (a[i][i]<a[i][j]):
                flag=1
                break
            if(a[i][i]>a[i][j]):
                flag2=1

        if (flag==1):
            converge=False
            break
        if (flag2==0):
            converge=False
            break

    return (a1,b1,n,pivot_zero,tol1,tol2,tol3,converge)

def work(n,a,b,pivot_zero):

    #triangulisation of matix

    for i in range(0,n):

        #finding max pivot element
        maximum=a[i][i]
        max_index=i
        for index in range(i,n):
            if (a[index][i]>maximum):
                max_index=index
                maximum=a[index][i]
        #max found

        #reordering equations for a good pivot term
        if (max_index!=i):
            for i1 in range(i,n):
                temp1=a[max_index][i1]
                a[max_index][i1]=a[i][i1]
                a[i][i1]=temp1

            temp2= b[max_index]
            b[max_index]=b[i]
            b[i]=temp2
```

```
                #reordering done

                for j in range(i+1,n):
                        c=float(a[j][i])/a[i][i]
                        for k in range(i,n):
                                a[j][k]=a[j][k]-c*a[i][k]
                        b[j]=b[j]-c*b[i]

        #triangularization done
        determinant=1
        for i in range(0,n):
                determinant=determinant*a[i][i]
        if (determinant<=10**(-16)):
                return ['x']
        #back substitution
        x=[0]*n
        i=n-1
        while(i>=0):
                lhs=0
                for j in range(i,n):
                        lhs=lhs+a[i][j]*x[j]
                x[i]=float(b[i]-lhs)/a[i][i]
                i=i-1
        return x

#guass sidel method

def work1(n,a,b):
        iterations=25
        x1=[0]*n
        for iterations in range(0,iterations):
                big=0
                for i in range(0,n):
                        sum1=0
                        for j in range(0,n):
                                if (j!=i): sum1=sum1+a[i][j]*x1[j]
                        temp=float(b[i]-sum1)/a[i][i]
                        relerror=abs(float((x1[i]-temp))/temp)
                        if (relerror>big): big=relerror
                        x1[i]=temp
        return x1



def analysis(n,a1,b1,x,tol1,tol2,tol3):
        tolerances=[]
        for i in range(0,n):
                sum1=0
                for j in range(0,n):
                        sum1=sum1+a1[i][j]*x[j]
                r=abs(float(sum1-b1[i])/sum1)
                if (r<=tol1): tolerances.append(tol1)
                elif (r<=tol2): tolerances.append(tol2)
                elif (r<=tol3): tolerances.append(tol3)
                else: tolerances.append('Bad result')
        return tolerances


def output(x,tolerances,converge,consistent):
        output=open('output','w')
        if (consistent==True):
                if (converge==True): output.write('Gauss Sidel Iteration
                else: output.write('Gauss Elimination used\n')
                output.write('x\n\n')
                for i in range(0,len(x)):
                        output.write(str(x[i])+'\n')
                output.write('\n\nTolerances\n\n')
                for i in range(0,len(x)):
                        output.write(str(tolerances[i])+'\n')
        output.close()

main()
```

6.2 How to run the code

1. Create an input file named 'input'.
2. The first line must have the numbers of equations, n
3. The next n lines must contain the coefficient matrix, rox by row, and values separated by spaces.

4. The next n lines must contain the the B matrix, that is, values of the contants on the RHS of the equations.
5. The results will be obtained in the file named *'output'*.