# 1 Problem 1 Solution

In order to show that the matrix $L_k$ is the inverse of the matrix $M_k$, we need to show that

$$L_k M_k = I \quad (\text{or} \quad M_k L_k = I).$$

Since

$$L_k = I + m_k e_k^T, \quad M_k = I - m_k e_k^T,$$

we need to show that

$$(L_k = I + m_k e_k^T)(M_k = I - m_k e_k^T) = I.$$

Since

$$L_k M_k = (I + m_k e_k^T)(I - m_k e_k^T) = I - m_k (e_k^T m_k) e_k^T,$$

According to the definition of elimination matrix, the $e_k$ as the $n \times 1$ basis vector can be represented as

$$e_k = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Then $e_k^T = (0 \quad \dots \quad 0 \quad 1 \quad 0 \quad \dots \quad 0)$, where 1 is in the $k^{th}$ column. And $m_k$ can be represented as

$$m_k = \frac{1}{a_{kk}} \cdot \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_{k+1,k} \\ \vdots \\ a_{n,k} \end{pmatrix}$$

Taking the dot product of $e_k^T$ with $m_k$ gives zero, so $L_k M_K = I$. Similar reasoning also shows that $M_k L_k = I$. Thus, matrix $L_k$ is the inverse of the matrix $M_k$.

## 1.1 Common Mistakes

### 1.1.1 No explanation of $e_k^T m_k = 0$

Five points are given for the explanation of deriving $e_k^T m_k = 0$ from the definition of elimination matrix.

#### 1.1.2 Using example only to explain

Only using a specific $A$ to derive the result without using the generalized formula is not sufficient.

# 2 Problem 2 Solution

The solution for the given system is:

$$\bar{x} = \begin{bmatrix} 0.0933 \\ 0.3841 \\ -1.2857 \\ 0.0580 \\ 0.0493 \\ 0.0382 \\ 0.2905 \\ -0.4883 \\ 0.5828 \end{bmatrix}$$

## 2.1 Common Mistakes

#### 2.1.1 Ignored Back Substitution

Given a linear system $A\bar{x} = \bar{y}$, it is definitely possible to obtain $\bar{x}$ from $\bar{x} = A^{-1}\bar{y}$. You can also use the lower and upper decompositions of $A$ to obtain $A^{-1}$ to perform the above operation. Uses of this strategy lost 5 points.

#### 2.1.2 Use of Gaussian Elimination

The problem asked for implementation of elimination matrices. Partial credit was given here.

#### 2.1.3 Use of linalg Library

The purpose of this exercise was to learn matrix manipulations in Python by implementing an interesting algorithm. Forming a matrix and calling a library function requires only a few lines of code and no thinking. Any code that made use of these functions lost all points on this problem.

#### 2.1.4 Elimination Matrices Produced Incorrect Results

At any point of implementing the algorithm, a mistake could compound itself in forming the upper and lower triangular matrices. This would also lead to obtaining an incorrect $\bar{x}$ matrix. In cases where the $\bar{x}$ vector was incorrect, if the intermediate matrices were in upper and lower form, then only 5 points were taken.

# 3 Problem 3 Solution

Suppose that $||\cdot||_a$ and $||\cdot||_b$ are two equivalent vector norms in $\mathbb{R}^n$. Thus, by definition, there exist two constants $c, d > 0$ such that for any vector $x \in \mathbb{R}^n$, we have

$$c||x||_a \leq ||x||_b \leq d||x||_a \tag{1}$$

Prove that their induced matrix norms are also equivalent, i.e., there exist two constants $c', d' > 0$ such that for any matrix $A \in \mathbb{R}^{n\times n}$

$$c'||A||_a \leq ||A||_b \leq d'||A||_a \tag{2}$$

**Proof:**
From (2) we see that we must show two things:

$$||A||_b \geq c'||A||_a \tag{3}$$
$$||A||_b \leq d'||A||_a \tag{4}$$

To start with we manipulate equation (1) in two separate ways to obtain

$$||x||_a \geq \frac{1}{d}\ ||x||_b$$
$$\frac{1}{||x||_a} \geq \frac{c}{||x||_b}$$

The two equations above imply

$$\frac{||Ax||_a}{||x||_a} \geq \frac{1}{d}\frac{||Ax||_b}{||x||_a} \geq \frac{c}{d}\frac{||Ax||_b}{||x||_b}$$

Using the properties of induced matrix norm we see

$$\max_{x\neq 0}\frac{||Ax||_a}{||x||_a} \geq \frac{c}{d}\max_{x\neq 0}\frac{||Ax||_b}{||x||_b}$$
$$||A||_a \geq \frac{c}{d}||A||_b$$
$$\frac{d}{c}||A||_a \geq ||A||_b$$

$$d'||A||_a \geq ||A||_b \tag{5}$$

Where we take $d' = \frac{d}{c}$. This proves equation (4).

Now we must show (3). First we set up similarly to the first part

$$||x||_a \leq \frac{1}{c}\ ||x||_b$$
$$\frac{1}{||x||_a} \leq \frac{d}{||x||_b}$$

The two equations above imply

$$\frac{||Ax||_a}{||x||_a} \leq \frac{1}{c}\frac{||Ax||_b}{||x||_a} \leq \frac{d}{c}\frac{||Ax||_b}{||x||_b}$$

Using the properties of induced matrix norm we see

$$\begin{aligned}\max_{x\neq 0}\frac{||Ax||_a}{||x||_a} &\leq \frac{d}{c}\max_{x\neq 0}\frac{||Ax||_b}{||x||_b}\\ ||A||_a &\leq \frac{d}{c}||A||_b\\ \frac{c}{d}||A||_a &\leq ||A||_b\end{aligned}$$

$$c'||A||_a \leq ||A||_b \tag{6}$$

Where we take $c' = \frac{c}{d}$. This proves equation (3). Now we use (5) and (6) to find

$$c'||A||_a \leq ||A||_b \leq d'||A||_a$$

This equation is precisely equation (2), so we conclude that the induced norms of (1) are also equivalent.

## 3.1 Common Mistakes

### 3.1.1 Incorrect Start

With a proof similar to this it is necessary to show manipulations step by step along the way from the first equation to the second. If you don't start with the first equation, nothing stated can justify the conclusion.

In some proofs the start was at the end and moved backwards to the first equation. If you do this particular type of proof you must make sure that the operation you perform satisfies an if and only if condition. Not only do you need to ensure it satisfies it, but you also need to state it.

# 4 Problem 4 Solution

Python Code

```
import numpy as np
from copy import deepcopy

# Forward substitution for lower triangular system
def Forward_Substitution(A,b,x):
    m=A.shape[0]
    n=A.shape[1]
    if(m!=n):
        print 'Matrix is not square!'
```

```
        return
    for j in range(0,n):
        if A[j,j] == 0:
            print 'Matrix is singular!'
            return          # matrix is singular
        x[j] = np.float32(b[j]) / np.float32(A[j,j])
        for i in range(j+1,n):
            b[i] = np.float32(b[i]) - np.float32(A[i,j]) *
                np.float32(x[j])

# Back substitution for upper triangular system
def Back_Substitution(A,b,x):
    m=A.shape[0]
    n=A.shape[1]
    if(m!=n):
        print 'Matrix is not square!'
        return
    for j in range(n-1,-1,-1):
        if A[j,j] == 0:
            print 'Matrix is singular!'
            return          # matrix is singular
        x[j] = np.float32(b[j]) / np.float32(A[j,j])
        for i in range(0,j):
            b[i] = np.float32(b[i]) - np.float32(A[i,j]) *
                np.float32(x[j])

# LU decomposition of square systems
def Gaussian_Elimination(A, b, x):
    m=A.shape[0]
    n=A.shape[1]

    U = np.matrix.copy(A)

    L = np.identity(n)
    if(m!=n):
        print 'Matrix is not square!'
        return
    for k in range(0,n-1):
        if A[k,k] == 0:
            return
        for i in range(k+1,n):
            L[i,k] = np.float32(A[i,k]) / np.float32(A[k,k])
            A[i,k] = np.float32(A[i,k]) / np.float32(A[k,k])
            U[i,k] = np.float32(0.)
        for j in range(k+1,n):
            for i in range(k+1,n):
                A[i,j] -= np.float32(A[i,k]) * np.float32(A[
                    k,j])
                U[i,j] = np.float32(A[i,j])

```

```
57      y = np.float32(np.zeros(n))
58      Forward_Substitution(L, b, y)
59      Back_Substitution(U, y, x)
60
61  def main():
62      A = np.matrix( [[21.0, 67.0, 88.0, 73.0],[76.0, 63.0,
            7.0, 20.0],
63                      [0.0, 85.0, 56.0, 54.0],[19.3, 43.0,
                           30.2, 29.4]])
64      b = np.array([141.0, 109.0, 218.0, 93.7])
65      x=np.zeros(4)
66      epsilon = 10**-6
67      print "PART A"
68      print "x:"
69      Acopy = np.matrix.copy(A)
70      bcopy = deepcopy(b)
71      Gaussian_Elimination(Acopy,bcopy,x)
72      x_n = np.zeros(4)
73      for i in range(4):
74          x_n[i] = x[i]
75      print x
76
77      print "PART B"
78
79      for iter in range(1000):
80          print "ITERATION:", iter
81
82          Acpy = np.matrix.copy(A)
83          Ax = Acpy.dot(x_n)
84          for j in range(4):
85              r[i] = np.float32(b[i] - Ax[0,i]
86
87          z = np.zeros(4)
88          Gaussian_Elimination(Acpy, r, z)
89
90          for k in range(4):
91              x_improved = x_n[k] + z[k]
92          print "x improved:", x_improved
93
94          for l in range(4):
95              x_n[l] = x_improved[l]
96
97          if(np.linalg.norm(x_improved - x_n, np.inf) <
                epsilon): break
98
99
100
101 if __name__ == "__main__":
```

```
102     main()
```

Listing 1: Part a Python code

This code will result in an original $x$ vector : x = [-0.99991786 2.00037098 -2.99869657 3.99806452]. This is an approximate solution to the problem that will be improved. A residual vector r is found by subtracting the answer vector $b$ by $Ax$ and this is used to compare the accuracy of our answer. The vector $z$ is computed by performing Gaussian Elimination on $Az = r$ and the improved vector is $x = x + z$. Thus, we compute the iterations until the threshold error value is reached.
The points were broken down as follows:
a) 4 points (for performing Gaussian Elimination and showing the $x$ vector)
b) 7 points (for computing r and storing the answer in single precision)
c) 5 points (for computing z and showing the improved vector $x = x + z$
d) 4 points (for performing iterations to show improvements)

## 4.1 Common Mistakes

### 4.1.1 Incorrect computation of $x$

One common area where many people lost points was being unable to compute the original $x$ with their Gaussian Elimination method. This occurred because many people were unable to implement the method correctly as the values were not correctly returned from the function, resulting in incorrect evaluations.

### 4.1.2 Storing $r$ in single precision

Another common error was that many people computed $r$ in single precision, which resulted in a loss of precision. The computation of $r$ was supposed to be in double precision and $r$ was to be stored in single precision.

### 4.1.3 Not performing enough iterations

Another common error was not performing iterations to see the value of $x$ improve. Many people failed to perform this step and reported the original $x$ vector as the final solution.

# 5 Problem 5 Solution

The output of your code [$x1$, $x2$] should be close to one of the following

| | | |
|---|---|---|
| $k = 1$ | [1. 1.] | [0.99999905 1.] |
| $k = 2$ | [1. 1.] | [1.00016594 1.] |
| $k = 3$ | [1. 1.] | [1.01327896 0.99999994] |
| $k = 4$ | [0.99999999 1.] | [0. 1.] |
| $k = 5$ | [1.00000008 1.] | [0. 1.] |
| $k = 6$ | [0.99986686 1.] | [0. 1.] |
| $k = 7$ | [0.99920072 1.] | [0. 1.] |
| $k = 8$ | [2.22044605 1.] | [0. 1.] |
| $k = 9$ | [0. 1.] | [0. 1.] |
| $k = 10$ | [0. 1.] | [0. 1.] |

depends on the float type (64 or 32) you used in code.

## 5.1 Example Code

```
import numpy as np
from copy import copy, deepcopy
import math

# Forward substitution for lower triangular system
def Forward_Substitution(A,b,x):
    m=A.shape[0]
    n=A.shape[1]
    if(m!=n):
        print 'Matrix is not square!'
        return
    for j in range(0,n):
        if A[j,j] == 0:
            print 'Matrix is singular!'
            return          # matrix is singular
        x[j] = b[j] / A[j,j]
        for i in range(j+1,n):
            b[i] = b[i] - A[i,j]*x[j]

# Back substitution for upper triangular system
def Back_Substitution(A,b,x):
    m=A.shape[0]
    n=A.shape[1]
    if(m!=n):
```

```
25          print 'Matrix is not square!'
26          return
27      for j in range(n-1,-1,-1):
28          if A[j,j] == 0:
29              print 'Matrix is singular!'
30              return          # matrix is singular
31          x[j] = b[j] / A[j,j]
32          for i in range(0,j):
33              b[i] = b[i] - A[i,j] * x[j]
34
35  def gaussian_elimination(A, b, x):
36      m=A.shape[0]
37      n=A.shape[1]
38
39      U = deepcopy(A)
40      L = np.identity(n)
41      if(m!=n):
42          print 'Matrix is not square!'
43          return
44      for k in range(0,n-1):
45          if A[k,k] == 0:
46              return #need pivoting for this problem, but not
                    for problem #5
47          for i in range(k+1,n):
48              L[i,k] = A[i,k] / A[k,k]
49              A[i,k] = A[i,k] / A[k,k]
50              U[i,k] = 0.
51          for j in range(k+1,n):
52              for i in range(k+1,n):
53                  A[i,j] -= A[i,k] * A[k,j]
54                  U[i,j] = A[i,j]
55      y=np.zeros(n)
56      Forward_Substitution(L, b, y)
57      Back_Substitution(U, y, x)
58
59  for k in range(1, 11):
60      epsilon = math.pow(10, -2 * k)
61      A = np.matrix([[epsilon, 1.], [1., 1.]])
62      b = np.array([1. + epsilon, 2.])
63      x = np.zeros(2)
64      gaussian_elimination(A,b,x)
65      print(x)
```

## 5.2 Common Mistakes

### 5.2.1 Misunderstanding the question

Some students expanded the matrix by hand and drew the conclusion that $\varepsilon$ does not have influence, but this question is looking for the "computed solution".