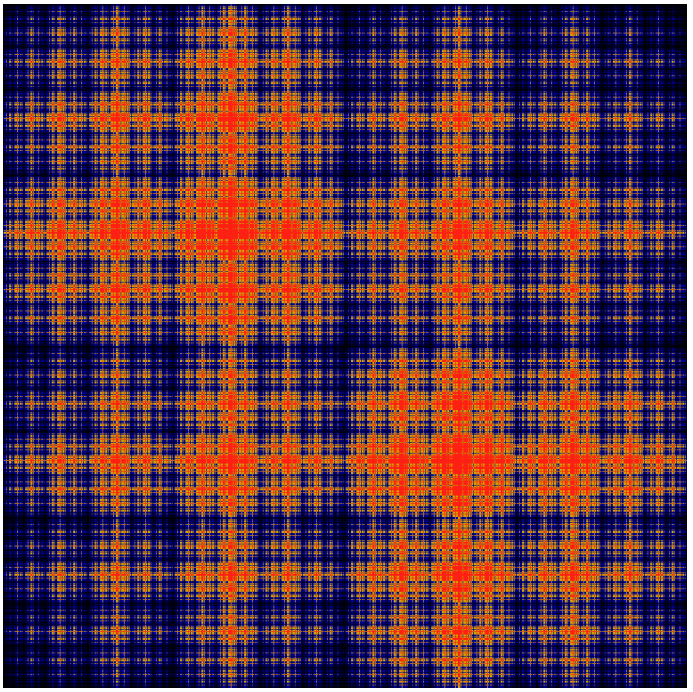
Applied Mathematics
and
Computing

Volume I



List of Contributors

J. Humpherys
Brigham Young University

J. Webb
Brigham Young University

R. Murray
Brigham Young University

J. West
University of Michigan

R. Grout
Brigham Young University

K. Finlinson
Brigham Young University

A. Zaitzeff
Brigham Young University

Preface

This lab manual is designed to accompany the textbook *Foundations of Applied Mathematics* by Dr. J. Humpherys.

Lab 7

Algorithms: RREF/Elementary Matrices

Lesson Objective: *In this section we will use elementary matrices to find the RREF and to find the LU decomposition.*

In Linear algebra there are 3 elementary row operations: switching two rows, multiplying a row by a constant, and adding a multiple of one row to another row. We carry out each of these operations with a corresponding elementary matrix. These matrices are easy to construct. Suppose A is an $m \times n$ matrix and you want to perform one of the three elementary operations on A. You can do this be constructing the $m \times m$ identity matrix, I, performing the elementary row operation on I to obtain E and then multiplying EA. For example, consider the matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}$$

If we want to swap the first two rows, we can left multiply the matrix A by:

$$E = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

then

$$EA = \begin{pmatrix} a_{21} & a_{22} & a_{23} & a_{24} \\ a_{11} & a_{12} & a_{13} & a_{14} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}.$$

E in this case is called a type I matrix.

Now let's examine the next row operation. If we want to multiply, say, the second row of A by the constant b, we can left multiply the matrix A by the following

matrix:

$$\tilde{E} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Then

$$\tilde{E}A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ ba_{21} & ba_{22} & ba_{23} & ba_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}.$$

$\tilde{E}$ is called a type II matrix.

Now let's examine the last row operation. If we want to multiply, say, the first row of A by a constant c and add it to the second row, we can left multiply the matrix A by the following matrix:

$$\widehat{E} = \begin{pmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Then

$$\widehat{E}A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ ca_{11} + a_{21} & ca_{12} + a_{22} & ca_{13} + a_{23} & ca_{14} + a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix}.$$

$\widehat{E}$ is called a type III matrix.

Below, the elementary matrices corresponding to each row operation is implemented in Python.

```
from scipy import eye

def rowswap(n, j, k):
    """Swaps two rows

    INPUTS: n -> matrix size
            j, k -> the two rows to swap"""
    out = 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)
28  out[j,k] = const
    return out
```

row_opers.py

Programming Row Reduction

A fundamental problem in linear algebra is using matrix representations to solve systems of linear equations. In this section, we do this by using elementary matrices to reduce a matrix into "row echelon form" (REF), as opposed to "reduced row echelon form" (RREF). We remark that to solve a linear system, it is actually faster computationally to use REF and then finish with back-substitution, than it is to use RREF. Consider the following matrix:

$$\begin{pmatrix} 4 & 5 & 6 & 3 \\ 2 & 4 & 6 & 4 \\ 7 & 8 & 0 & 5 \end{pmatrix}$$

By iteratively left multiplying by elementary matrices, we can reduce as follows:

Remember that our functions returns the elementary array corresponding to the desired row operation. Also note that setting the type of our initial array is crucial.

```
: import scipy as sp
: import row_opers as op
: A = sp.array([[4, 5, 6, 3],[2, 4, 6, 4],[7, 8, 0, 5]], dtype='
    float32')
array([[ 4.,  5.,  6.,  3.],
       [ 2.,  4.,  6.,  4.],
       [ 7.,  8.,  0.,  5.]], dtype=float32)
: A1 = sp.dot(op.cmultadd(3,1,0,-A[1,0]/A[0,0]), A); A1
array([[ 4. ,  5. ,  6. ,  3. ],
       [ 0. ,  1.5,  3. ,  2.5],
       [ 7. ,  8. ,  0. ,  5. ]])
: A2 = sp.dot(op.cmultadd(3,2,0,-A1[2,0]/A1[0,0]), A1); A2
array([[  4.  ,   5.  ,   6.  ,   3.  ],
       [  0.  ,   1.5 ,   3.  ,   2.5 ],
       [  0.  ,  -0.75, -10.5 ,  -0.25]])
: A3 = sp.dot(op.cmultadd(3,2,1,-A2[2,1]/A2[1,1]), A2); A3
array([[ 4. ,  5. ,  6. ,  3. ],
       [ 0. ,  1.5,  3. ,  2.5],
       [ 0. ,  0. , -9. ,  1. ]])
```

To complete REF we would need to divide each row by its leading coefficient. We can do that using Type II matrices. We leave it to you to carry this out.

Problem 1 Write a Python function, which takes as input an $n \times (n+1)$ matrix (in other words and *augmented* matrix) and performs the above naive row reduction to REF using elementary matrices. (You do not need to worry about underdetermined matrices or getting zeros on the main diagonal)

LU Decomposition

Again, consider the matrix A. By iteratively left multiplying by Type 3 elementary matrices, we reduce as follows:

```
: E1 = op.cmultadd(3,1,0,-A[1,0]/A[0,0]); E1
array([[ 1. ,  0. ,  0. ],
       [-0.5,  1. ,  0. ],
       [ 0. ,  0. ,  1. ]])
: B1 = sp.dot(E1, A)
array([[ 4. ,  5. ,  6. ,  3. ],
       [ 0. ,  1.5,  3. ,  2.5],
       [ 7. ,  8. ,  0. ,  5. ]])
: E2 = op.cmultadd(3,2,0,-B1[2,0]/B1[0,0])
: B2 = sp.dot(E2, B1)
: E3 = op.cmultadd(3,2,1,-B2[2,1]/B2[1,1])
: U = sp.dot(E3, B2); U
array([[ 4. ,  5. ,  6. ,  3. ],
       [ 0. ,  1.5,  3. ,  2.5],
       [ 0. ,  0. , -9. ,  1. ]])
```

Note that we have reduced the above matrix into upper-triangular form, denoted as U. Hence, we have

$$U = E_3 E_2 E_1 A.$$

Since the elementary matrices are invertible, we also have

$$(E_3 E_2 E_1)^{-1} U = A.$$

This can be re-written as

$$E_1^{-1} E_2^{-1} E_3^{-1} U = A.$$

Then we define L to be

$$L = E_1^{-1} E_2^{-1} E_3^{-1},$$

which yields $LU = A$.

```
: from scipy import linalg as la
: I = lambda x: la.inv(x)
: L = sp.dot(sp.dot(I(E1), I(E2)), I(E3)); L
array([[ 1.  ,  0.  ,  0.  ],
       [ 0.5 ,  1.  ,  0.  ],
       [ 1.75, -0.5 ,  1.  ]])
: sp.dot(L, U)
array([[ 4.,  5.,  6.,  3.],
       [ 2.,  4.,  6.,  4.],
       [ 7.,  8.,  0.,  5.]])
```

What makes LU decomposition so easy is that the inverses of elementary matrices are elementary matrices. For example, the inverse of a Type 3 elementary matrix is the same matrix with the opposite sign in the (j, k) entry. In the above problem, we have: Note that the minus signs are gone. Doing it this way, we don't have to actually invert anything to compute L. This makes the computation much faster.

Why Should I Care?

The LU Decomposition isn't very useful when doing matrix computation by hand. It is, however, very important in scientific computation for the following reasons:

- If you want to solve the matrix equation $Ax = b$, for several different $b's$, you can replace A with L and U, giving $LUx = b$. Then solve the equations $Ly = b$ and $Ux = y$, using forward and backward substitution, respectively. This is actually faster than solving them with row reduction.
- The LU decomposition allows quick computation of both inverses and determinants.
- For very large matrices the LU decomposition is crucial. Indeed one can perform the LU decomposition on a given matrix A without needing additional space, that is, the program actually over-writes A with L and U. Note that since the diagonal of L are all ones, they don't need to be stored, and so the upper diagonal (including the diagonal) is U and the lower diagonal (not including the diagonal) is L.

Problem 2 Write a Python function which takes as input a random $n \times n$ matrix, performs the LU decomposition and returns L and U. To verify that it works, multiply L and U together and compare to A. Note: you should not use the `inv` function when you do this. You should only use the elementary matrices that we just created. Additionally, have your function count the number of operations needed to perform the LU decomposition.

Problem 3 Write a Python function which uses the solution to Problem 2 to find the determinant of A.