Computational Linear Algebra Project

Aalekh Roy
2020D001

June 2021

Contents

1 Acknowledgement

This project is aimed to compile the program codes related to matrix algebra covered in the course titled "Computational Linear Algebra" offered in 2020 under the guidance of Dr. Sudhakar Sahoo.

The document has been compiled in RStudio using knitr package which provides an efficient yet beautiful way to make latex style document with code snippets. The language is R, due to the inbuilt libraries and overall functionality of matrix theory and linear algebra provided by R.

The author of the document claims no responsibility for the material provided as all the material/codes has been accumulated via various sources. The only claim he intends to make is the effort to compile the document word by word as he believes in "Learning by Doing". A special thanks to Dr. Sudhakar for providing me this opportunity.

2 Vector Operations

2.1 Define vectors

Vectors are generally created using the c() function.

```
> # Define two vectors
> x <- c(30, 20, 40, 10)
> y <- c(20, 15, 18, 40)
> print(x)

[1] 30 20 40 10

> print(y)

[1] 20 15 18 40
```

If we want to create a vector of consecutive numbers, the : operator is very helpful.

```
> x=1:7 ; x

[1] 1 2 3 4 5 6 7

> y=2:-2 ; y

[1]  2  1  0 -1 -2
```

More complex sequences can be created using the seq() function, like defining number of points in an interval, or the step size.

```
> seq(1, 3, by=0.2)           # specify step size

 [1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0

> seq(1, 5, length.out=4)    # specify length of the vector

[1] 1.000000 2.333333 3.666667 5.000000
```

2.2 Arithemtic of vectors

We can add, subtract, multiply by a scalar and calculate dot or inner product of vectors.

2.2.1 Addition and Subtraction

Let a and b be n -vectors. The sum $a + b$ is the n -vector

$$a + b = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_n + b_n \end{bmatrix} = b + a$$

Only vectors of the same dimension can be added.

Example

$$\begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} + \begin{bmatrix} 2 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 1+2 \\ 3+8 \\ 2+9 \end{bmatrix} = \begin{bmatrix} 3 \\ 11 \\ 11 \end{bmatrix}$$

```
> x=c(1,3,2)
> y=c(2,8,9)
> x + y
[1]  3 11 11
> x - y
[1] -1 -5 -7
```

If the the vectors don't have the same length the elements of the smallest will be recycled:

```
> x
[1] 1 3 2
> z=c(10, 10)
> z
[1] 10 10
> x + z #every element of x gets added by 10 even though the vector z has only 2
[1] 11 13 12
```

2.2.2 Multiplication by scalar

To multiple by a scalar, we use * operator:

If a is a vector and α is a number then αa is the vector

$$\alpha a = \begin{bmatrix} \alpha a_1 \\ \alpha a_2 \\ \vdots \\ \alpha a_n \end{bmatrix}$$

Example

$$7 \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 7 \\ 21 \\ 14 \end{bmatrix}$$

```
> x=c(1,3,2)
> x

[1] 1 3 2

> 7 * x

[1]  7 21 14
```

2.2.3 Dot or Inner product of vectots

Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$. The (inner) product of a and b is

$$a \cdot b = a_1 b_1 + \cdots + a_n b_n$$

Note, that the product is a number - not a vector, but since R takes the vectors as a list and it outputs as a matrix object.

For calculating dot product(Inner product), we use:

```
> x=c(1,3,2)
> y=c(1,2,3)
> x %*% y

     [,1]
[1,]   13
```

Note that this operator returns a matrix object. If we need just the numeric value, use the *as.numeric* function :

```
> as.numeric(x %*% y)

[1] 13
```

2.3 Norm of a vector

The length (or norm) of a vector a is

$$\|a\| = \sqrt{a \cdot a} = \sqrt{\sum_{i=1}^{n} a_i^2}$$

The vector magnitude, or norm, can be obtained as follows:

```
> x

[1] 1 3 2

> sqrt(x %*% x)

         [,1]
[1,] 3.741657
```

2.4 Transpose of a vector

Transposing a vector means turning a column (row) vector into a row (column) vector. The transpose is denoted by T.

Example

$$\begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}^\top = [1,3,2] \text{ og } \quad [1,3,2]^\top = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}$$

Hence transposing twice takes us back to where we started:

$$a = \left(a^\top\right)^\top$$

```
> x=c(1,3,2)
> x

[1] 1 3 2

> t(x)

     [,1] [,2] [,3]
[1,]    1    3    2
```

2.5 The 0-vector and 1-vector

The 0-vector(1-vector) is a vector with 0(1) on all entries. The 0-vector (1-vector) is frequently written simply as 0(1).

```
> rep(0,5) #repeat function

[1] 0 0 0 0 0

> rep(1,5)

[1] 1 1 1 1 1
```

2.6 Orthogonal(perpendicular) vectors

Two vectors v1 and v2 are orthogonal if their inner or dot product is zero, written v1.v2 = 0

```
> v1=c(1,1)
> v1

[1] 1 1

> v2=c(-1,1)
> v2

[1] -1  1

> v1%*%v2

     [,1]
[1,]    0
```

3 Matrix Operations

3.1 Define a matrix

An $r \times c$ matrix A (reads "an r times c matrix") is a table with r rows og c columns

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1c} \\ a_{21} & a_{22} & \dots & a_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \dots & a_{rc} \end{bmatrix}$$

Note that one can regard A as consisting of c columns vectors put after each other:

$$A = [a_1 : a_2 : \cdots : a_c]$$

We use the inbuilt function in R to define matrices:

```
> # Define a matrix
> m <- c(7, -6, 12, 8)
> m <- matrix(m, nrow = 2)
> m

     [,1] [,2]
[1,]    7   12
[2,]   -6    8
```

Note that the numbers 1, 3, 2, 2, 8, 9 are read into the matrix column by column. To get the numbers read in row by row do *byrow=TRUE*.

```
> m1=matrix(c(7,-6,12,8),ncol=2,byrow=T)
> m1

     [,1] [,2]
[1,]    7   -6
[2,]   12    8
```

3.2 Matrix Addition and Subtraction

We can add and subtract two matrices just as we can with vectors i.e using arithmetic addition and subtraction element wise.

Let A and B be 3x2 matrices. The sum A + B is the 3x2 matrix obtained by adding A and B element-wise. Only matrices with the same dimensions can be added/subtracted.

Example

$$\begin{bmatrix} 1 & 2 \\ 3 & 8 \\ 2 & 9 \end{bmatrix} + \begin{bmatrix} 5 & 4 \\ 8 & 2 \\ 3 & 7 \end{bmatrix} = \begin{bmatrix} 6 & 6 \\ 11 & 10 \\ 5 & 16 \end{bmatrix}$$

```
> x=c(1,2,3,8,2,9)
> y=c(5,4,8,2,3,7)
> x=matrix(x,ncol=2,byrow = TRUE)
> y=matrix(y,ncol=2,byrow=TRUE)
> x

     [,1] [,2]
[1,]    1    2
[2,]    3    8
[3,]    2    9

> y

     [,1] [,2]
[1,]    5    4
[2,]    8    2
[3,]    3    7

> x+y

     [,1] [,2]
[1,]    6    6
[2,]   11   10
[3,]    5   16

> x-y

     [,1] [,2]
[1,]   -4   -2
[2,]   -5    6
[3,]   -1    2
```

3.3 Transpose of Matrix

A matrix is transposed by interchanging rows and columns and is denoted by "T". Example

$$\begin{bmatrix} 1 & 2 \\ 3 & 8 \\ 2 & 9 \end{bmatrix}^\top = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 8 & 9 \end{bmatrix}$$

Note that if A is an $r \times c$ matrix then $A^\top$ is a $c \times r$ matrix.

```
> m

     [,1] [,2]
[1,]    7   12
[2,]   -6    8

> t(m)

     [,1] [,2]
[1,]    7   -6
[2,]   12    8
```

3.4 Multiplication of a matrix and a vector

Let A be an $r \times c$ matrix and let b be a c -dimensional column vector. The product Ab is the $r \times 1$ matrix

$$Ab = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1c} \\ a_{21} & a_{22} & \dots & a_{2c} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \dots & a_{rc} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_c \end{bmatrix} = \begin{bmatrix} a_{11}b_1 + a_{12}b_2 + \cdots + a_{1c}b_c \\ a_{21}b_1 + a_{22}b_2 + \cdots + a_{2c}b_c \\ \vdots \\ a_{r1}b_1 + a_{r2}b_2 + \cdots + a_{rc}b_c \end{bmatrix}$$

Example

$$\begin{bmatrix} 1 & 2 \\ 3 & 8 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 5 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 \cdot 5 + 2 \cdot 8 \\ 3 \cdot 5 + 8 \cdot 8 \\ 2 \cdot 5 + 9 \cdot 8 \end{bmatrix} = \begin{bmatrix} 21 \\ 79 \\ 82 \end{bmatrix}$$

```
> A <- matrix(c(1, 3, 2, 2, 8, 9), ncol = 3)
> A
```

```
     [,1] [,2] [,3]
[1,]    1    2    8
[2,]    3    2    9
```

```
> a <- c(1, 3, 2)
> a
```

```
[1] 1 3 2
```

```
> J = A %*% a #Matrix multiplication
> J
```

```
     [,1]
[1,]   23
[2,]   27
```

3.5 Matrix Matrix Multiplication

When we multiply two numbers a, b (scalars or 1×1 matrices) we simply multiply them as $a * b$ which always equals $b * a$. Not so if we are multiplying matrices! This section will show that the order of multiplication does matter for matrices A, B. That is, matrix multiplication AB is not commutative $AB \neq BA$. In fact, it is possible that one or more of the matrix multiplications may not be well defined (conformable).

Recall the curly braces notation and let $A = \{a_{ij}\}$ be $r_a \times c_a$ matrix with $i = 1, 2, \cdots, r_a$ rows and $j = 1, 2, \cdots, c_a$ columns. Similarly let $B = \{b_{k\ell}\}$ matrix have $k = 1, 2, \cdots, r_b$ rows, and $\ell = 1, 2, \cdots, c_b$ columns. Let us understand the notion of matrix conformability before discussing matrix multiplication. If A is $r_a \times c_a$ matrix, and B is is $r_b \times c_b$ matrix, then the row column multiplication of AB is conformable (well defined) only if Number of columns of $A = c_a = r_b =$ Number of rows of B. An easy way to remember this is that in any "row-column" multiplication, we must have equality of the numbers of "column-rows".

```
> A <- matrix(c(1, 3, 2, 2, 8, 9), ncol = 2);A
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    8
[3,]    2    9
```

```
> B <- matrix(c(5, 8, 4, 2), ncol = 2);B
```

```
     [,1] [,2]
[1,]    5    4
[2,]    8    2
```

```
> A%*%B
```

```
     [,1] [,2]
[1,]   21    8
[2,]   79   28
[3,]   82   26
```

3.6 Rank of a Matrix

The rank of a matrix corresponds to the maximal number of linearly independent columns of the matrix.

To obtain this value, use the function qr as follows:

```
> # Rank of a Matrix
> C <- matrix(c(1, 0, 1, -2, -3, 1, 3, 3, 0), nrow = 3, byrow = TRUE)
> C
```

```
     [,1] [,2] [,3]
[1,]    1    0    1
[2,]   -2   -3    1
[3,]    3    3    0
```

```
> qr(C)$rank
```

```
[1] 2
```

```
> D <- matrix(c(1, 1, 0, 2, -1, -1, 0, -2), nrow = 2, byrow = TRUE)
> D
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    1    0    2
[2,]   -1   -1    0   -2
```

```
> qr(D)$rank
```

```
[1] 1
```

3.7 Trace of a matrix

Trace of a matrix is the sum of the values on the main diagonal(upper left to lower right) of the matrix.

tr() function is used to calculate the trace of a matrix.We need to have (psych) package installed.

```
> # R program to calculate
> # trace of a matrix
>
> # Loading library
> library(psych)
> # Creating a matrix
> A = matrix(
+   c(6, 1, 1, 4, -2, 5, 2, 8, 7),
+   nrow = 3,
+   ncol = 3,
+   byrow = TRUE
+ )
> A

     [,1] [,2] [,3]
[1,]    6    1    1
[2,]    4   -2    5
[3,]    2    8    7

> # Calling tr() function
> cat("Trace of A:\n")

Trace of A:

> tr(A)

[1] 11
```

3.8 Null Space, Column Space of a matrix

Null Space

The null space of any matrix A consists of all the vectors B such that AB

= 0 and B is not zero. It can also be thought as the solution obtained from AB = 0 where A is known matrix of size m x n and B is matrix to be found of size n x k. The size of the null space of the matrix provides us with the number of linear relations among attributes.

Column Space
A column space (or range) of matrix X is the space that is spanned by X's columns.

Row Space
A row space is spanned by rows of a matrix.

In R, there is an inbuilt function "nullspace()" to find the nullspace of a matrix and orth() function for column space

```
> library(pracma)
> M <- matrix(1:12, 3, 4)
> Rank(M)
[1] 2
> nullspace(M)#nullspace of M
           [,1]        [,2]
[1,] -0.3920453 -0.38249241
[2,]  0.2376341  0.80220323
[3,]  0.7008678 -0.45692923
[4,] -0.5464566  0.03721841
> orth(M) #column space of M
           [,1]        [,2]
[1,] -0.5045331 -0.76077568
[2,] -0.5745157 -0.05714052
[3,] -0.6444983  0.64649464
> #Column space of a transpose of a matrix is row space of the original matrix
> orth(t(M)) #row space of M
           [,1]        [,2]
[1,] -0.1408767 -0.82471435
[2,] -0.3439463 -0.42626394
[3,] -0.5470159 -0.02781353
[4,] -0.7500855  0.37063688
```

3.9 Diagonal Matrix

A square matrix in which every element except the principal diagonal elements is zero is called a Diagonal Matrix. A square matrix D = [dij]n x n will be called a diagonal matrix if dij = 0, whenever i is not equal to j.

```
> # R program to illustrate
> # diag function
>
> # Calling the diag() function with
> # some number which will act as
> # rows as well as columns number
> diag(3)

     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1

> diag(5)

     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1

> # R program to illustrate
> # diag function
>
> # Calling the diag() function
> diag(5, 2, 3)

     [,1] [,2] [,3]
[1,]    5    0    0
[2,]    0    5    0

> diag(10, 3, 3)
```

```
     [,1] [,2] [,3]
[1,]   10    0    0
[2,]    0   10    0
[3,]    0    0   10
```

```
> #Extract the diagonal of a matrix
> m=matrix(c(1,2,3,4,5,6,7,8,9),nrow=3)
> m
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
> diag(m)
```

```
[1] 1 5 9
```

3.10 Triangular Matrix

A triangular matrix is a special kind of square matrix.
A square matrix is called lower triangular if all the entries above the main diagonal are zero. Similarly, a square matrix is called upper triangular if all the entries below the main diagonal are zero.
An upper triangular matrix U is defined by

$$U_{ij} = \begin{cases} a_{ij} & \text{for } i \leq j \\ 0 & \text{for } i > j \end{cases}$$

Written explicitly,

$$\mathrm{U} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

A lower triangular matrix L is defined by

$$L_{ij} = \begin{cases} a_{ij} & \text{for } i \geq j \\ 0 & \text{for } i < j \end{cases}$$

Written explicitly,

$$\mathrm{L} = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

```
> library(fBasics)
> A=matrix(1:9,3) #create simple matrix 3 by 3
> A #display the matrix

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> diag(A)#display diagonals

[1] 1 5 9

> cumprod(diag(A)) #product of diagonals

[1]  1  5 45

> triang(A)#display lower triangular matrix

     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    2    5    0
[3,]    3    6    9

> det(triang(A)) #det of lower triangular matrix

[1] 45

> Triang(A)#display upper triangular matrix

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    0    5    8
[3,]    0    0    9

> det(Triang(A))#det of upper triangular matrix

[1] 45
```

3.11 Inverse of matrix

In general, the inverse of an $n \times n$ matrix A is the matrix B (which is also $n \times n$) which when multiplied with A gives the identity matrix I. That is,

$$AB = BA = I$$

One says that B is A 's inverse and writes $B = A^{-1}$. Likewise, A is B s inverse. Example 9 Let

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad B = \begin{bmatrix} -2 & 1.5 \\ 1 & -0.5 \end{bmatrix}$$

Now $AB = BA = I$ so $B = A^{-1}$.
Example 10 If A is a 1×1 matrix, i.e. a number, for example $A = 4$, then $A^{-1} = 1/4$
Some facts about inverse matrices are:
− Only square matrices can have an inverse, but not all square matrices have an inverse.
− When the inverse exists, it is unique.
− Finding the inverse of a large matrix A is numerically complicated (but computers do it for us).

Finding the inverse of a matrix in R is done using the solve() function:

```
> A=matrix(c(1,3,2,4),ncol=2,byrow=T)
> A

     [,1] [,2]
[1,]    1    3
[2,]    2    4

> B=solve(A)
> B

     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5

> A%*%B

     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

3.12 Determinant of matrix

Let the matrix A be a square matrix $A = (a_{ij})$ with i and $j = 1, 2, \cdots, n$. The determinant is a scalar number. When $n = 2$ we have a simple calculation of the determinant as follows:

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{22}$$

where one multiplies the elements along the diagonal (going from top left corner to bottom right corner) and subtracts a similar multiplication of diagonals going from bottom to top.

In R the function function 'matrix' creates a square matrix if the argument list neglects to specify a second dimension. For example 'A = matrix(10 : 13, 2) ' creates a 2×2 square matrix from the four numbers: $(10, 11, 12, 13)$. In R the function 'det' gives the determinant as shown below. For the following example, the det(A) is -2 and is also obtained by cross multiplication and subtraction by $10^*13 - 11^*12$.

```
> A=matrix(10:13,2) #creates a square matrix from 10,11,12,13
> A

     [,1] [,2]
[1,]   10   12
[2,]   11   13

> det(A)

[1] -2
```

4 Sparse Matrix

If most of the elements of the matrix have 0 value, then it is called a sparse matrix.
Why to use Sparse Matrix instead of simple matrix ?
Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements.
We can create a sparse matrix from a dense matrix:

```
> library(Matrix)
> data=rnorm(1e6) #rnorm function generates random
> #number with argument as the number of elements
> zero_index=sample(1e6)[1:9e5] #create a vector with million elements,
>                                  #but 90% of the elements are zeros
> data[zero_index]=0
> mat = matrix(data, ncol=1000) #create matrix using above data
> print(mat[1:5,1:5]) #display 5*5 sample of matrix
```

```
           [,1]      [,2] [,3]       [,4]      [,5]
[1,]  0.000000 0.0000000    0  0.0000000  0.000000
[2,] -0.201013 0.0000000    0  0.0000000 -1.193795
[3,]  0.000000 0.0000000    0  0.5988704  0.000000
[4,]  0.000000 0.7125958    0 -0.8873753  0.000000
[5,]  0.000000 0.0000000    0  0.0000000  0.000000
```

```
> print(object.size(mat),units="auto") #check the size of the matrix
```

```
7.6 Mb
```

```
> mat1 = Matrix(mat,sparse=TRUE) #using sparse matrix library to convert the mat
> print(mat1[1:5,1:5]) # see the sparse 5*5 matrix
```

```
5 x 5 sparse Matrix of class "dgCMatrix"

[1,]  .        .         .  .          .
[2,] -0.201013 .         .  .         -1.193795
[3,]  .        .         .  0.5988704  .
[4,]  .        0.7125958 . -0.8873753  .
[5,]  .        .         .  .          .
```

```
> print(object.size(mat1),units="auto") # print size of sparse matrix
1.1 Mb
> image(mat1[1:10,1:10]) #using image function to visualize
```

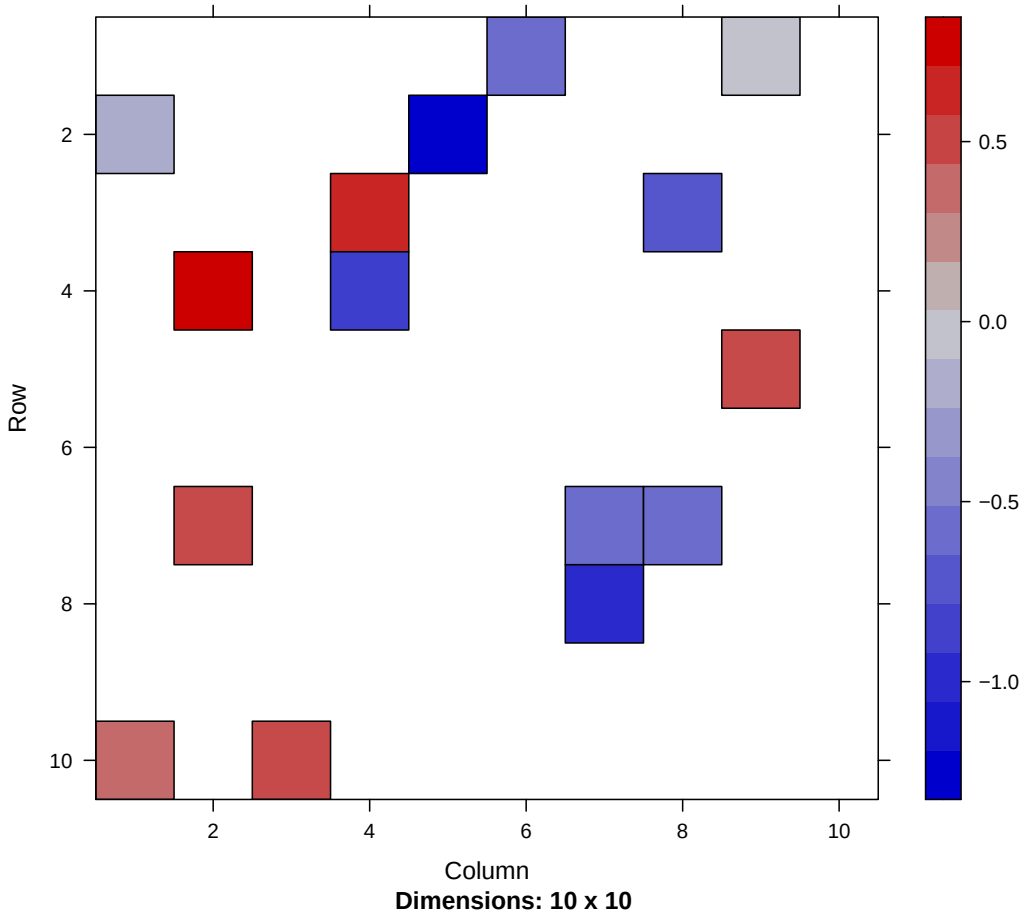

Figure 1: The sparse matrix 10x10

5 Block Matrices

It is often convenient to partition a matrix M into smaller matrices called blocks, like so:

$$M = \left(\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 0 \\ \hline \frac{7}{0} & 8 & 9 & \frac{1}{0} \end{array}\right) = \left(\begin{array}{cc} A & B \\ \hline C & D \end{array}\right)$$

$$\text{Here } A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, B = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, C = \begin{pmatrix} 0 & 1 & 2 \end{pmatrix}, D = (0).$$

- The blocks of a block matrix must fit together to form a rectangle. So There are many ways to cut up an $n \times n$ matrix into blocks. Often context or the entries of the matrix will suggest a useful way to divide the matrix into blocks. For example, if there are large blocks of zeros in a matrix, or blocks that look like an identity matrix, it can be useful to partition the matrix accordingly.

```
> library(blockmatrix)
> # create toy matrices (block matrix elements)
> # with values which makes it easier to track them in the block matrix in the e
> A <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
> B <- matrix(c(5,6,7,8), nrow = 2, ncol = 2)
> # function for creating the block matrix
> # n: number of repeating blocks in each dimension
> # (I use n instead of T, to avoid confusion with T as in TRUE)
> # m_list: the two matrices in a list
>
> block <- function(n, m_list){
+   # create a 'layout matrix' of the block matrix elements
+   m <- matrix("B", nrow = n, ncol = n)
+   diag(m) <- "A"
+
+   # build block matrix
+   as.matrix(blockmatrix(dim = dim(m_list[[1]]), value = m, list = m_list))
+ }
> # try with different n
> block(n = 2, m_list = list(A = A, B = B))
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
[3,]    5    7    1    3
[4,]    6    8    2    4

> #block(n = 3, m_list = list(A = A, B = B))
> #block(n = 5, m_list = list(A = A, B = B))
```

6 Strassen's Matrix Multiplication Algorithm

Illustration of method

The Strassen's algorithm is an algorithm used for matrix multiplication. It is faster than the standard matrix multiplication algorithm, but would be slower than the fastest known algorithm (Coppersmith-Winograd algorithm) for extremely large matrices.

Let $\mathbf{A}, \mathbf{B}$ two square matrix, $\in R^{2^n \times 2^n}$, with $n = 2, 3, \ldots$. We want to calculate the matrix $\mathbf{C}$, defined by $\mathbf{C} = \mathbf{AB}$.

First, we divide the two matrix $\mathbf{A}, \mathbf{B}$, into equally size block-matrices of dimensions $2^{n-1} \times 2^{n-1}$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}$$

Now define new matrices: $\mathbf{M}_1 = (\mathbf{A}_{11} + \mathbf{A}_{22})(\mathbf{B}_{11} + \mathbf{B}_{22})$
$\mathbf{M}_2 = (\mathbf{A}_{21} + \mathbf{A}_{22})\mathbf{B}_{11}$
$\mathbf{M}_3 = \mathbf{A}_{11}(\mathbf{B}_{12} - \mathbf{B}_{22})$
$\mathbf{M}_4 = \mathbf{A}_{22}(\mathbf{B}_{21} - \mathbf{B}_{11})$
$\mathbf{M}_5 = (\mathbf{A}_{11} + \mathbf{A}_{12})\mathbf{B}_{22}$
$\mathbf{M}_6 = (\mathbf{A}_{21} - \mathbf{A}_{11})(\mathbf{B}_{11} + \mathbf{B}_{12})$
$\mathbf{M}_7 = (\mathbf{A}_{12} - \mathbf{A}_{22})(\mathbf{B}_{21} + \mathbf{B}_{22})$
The block-matrices of the product matrix $\mathbf{C}$ are: $\mathbf{C}_{11} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$

$$\mathbf{C}_{12} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{21} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{22} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

So the matrix $\mathbf{C}$ is:

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{C}_{12} \\ \mathbf{C}_{21} & \mathbf{C}_{22} \end{bmatrix}$$

Example We have these two matrices:

$$\mathbf{A} = \begin{bmatrix} 7 & 31 & 13 & 106 \\ 24 & 19 & 51 & 68 \\ 139 & 127 & 121 & 117 \\ 13 & 105 & 53 & 59 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 22 & 111 & 93 & 181 \\ 155 & 42 & 120 & 17 \\ 171 & 115 & 26 & 26 \\ 167 & 203 & 6 & 31 \end{bmatrix}$$

Split the matrices:

$$\mathbf{A} = \left[\begin{array}{cc|cc} 7 & 31 & 13 & 106 \\ 24 & 19 & 51 & 68 \\ \hline 139 & 127 & 121 & 117 \\ 13 & 105 & 53 & 59 \end{array}\right] \quad \mathbf{B} = \left[\begin{array}{cc|cc} 22 & 111 & 93 & 181 \\ 155 & 42 & 120 & 17 \\ \hline 171 & 115 & 26 & 26 \\ 167 & 203 & 6 & 31 \end{array}\right]$$

Now we have, for example, $\mathbf{A}_{11} = \begin{bmatrix} 7 & 31 \\ 24 & 19 \end{bmatrix}$, and $\mathbf{B}_{21} = \begin{bmatrix} 171 & 115 \\ 167 & 203 \end{bmatrix}$. Now calculate the matrices $\mathbf{M}_{1:7}$

$$\mathbf{M}_1 = \begin{bmatrix} 29972 & 16254 \\ 28340 & 16243 \end{bmatrix}$$

$$\mathbf{M}_2 = \begin{bmatrix} 43540 & 26872 \\ 39108 & 14214 \end{bmatrix}$$

$$\mathbf{M}_3 = \begin{bmatrix} 4003 & 3774 \\ 651 & 3454 \end{bmatrix}$$

$$\mathbf{M}_4 = \begin{bmatrix} 19433 & 8605 \\ 19321 & 9711 \end{bmatrix}$$

$$\mathbf{M}_5 = \begin{bmatrix} 1342 & 2472 \\ 4767 & 4647 \end{bmatrix}$$

$$\mathbf{M}_6 = \begin{bmatrix} 41580 & 22385 \\ 44208 & 1862 \end{bmatrix}$$

$$\mathbf{M}_7 = \begin{bmatrix} -23179 & 1163 \\ -17802 & 1824 \end{bmatrix}$$

Now calculate:

$$\mathbf{C}_{11} = \begin{bmatrix} 24884 & 23550 \\ 25092 & 23131 \end{bmatrix}$$

$$\mathbf{C}_{12} = \begin{bmatrix} 5345 & 6246 \\ 5418 & 8101 \end{bmatrix}$$

$$\mathbf{C}_{21} = \begin{bmatrix} 62973 & 35477 \\ 58429 & 23925 \end{bmatrix}$$

$$\mathbf{C}_{22} = \begin{bmatrix} 32015 & 15541 \\ 34091 & 7345 \end{bmatrix}$$

The final result is:

$$\mathbf{C} = \begin{bmatrix} 24884 & 23550 & 62973 & 35477 \\ 25092 & 23131 & 58429 & 23925 \\ 5345 & 6246 & 32015 & 15541 \\ 5418 & 8101 & 34091 & 7345 \end{bmatrix}$$

Solution using R

We saw that there are 4 steps to be solved:

1. Split matrices
2. Calculate $\mathbf{M}_{1:7}$
3. Calculate block-matrices of the product $\mathbf{C}$
4. Recompose the matrix $\mathbf{C}$

```
> A = matrix ( c
+ (7 ,31 ,13 ,106 ,24 ,19 ,51 ,68 ,139 ,127 ,121 ,117 ,13 ,105 ,53 ,59) ,
+ byrow =T , nrow =4)
> B = matrix ( c
+ (22 ,111 ,93 ,181 ,155 ,42 ,120 ,17 ,171 ,115 ,26 ,26 ,167 ,203 ,6 ,31) ,
+ byrow =T , nrow =4)
> # Step -1 -
> A11 = A [1:2 ,1:2]
> A12 = A [1:2 ,3:4]
> A21 = A [3:4 ,1:2]
> A22 = A [3:4 ,3:4]
> B11 = B [1:2 ,1:2]
> B12 = B [1:2 ,3:4]
> B21 = B [3:4 ,1:2]
> B22 = B [3:4 ,3:4]
> # Step -2 -
> M1 = ( A11 + A22 ) %*% ( B11 + B22 )
> M2 = ( A21 + A22 ) %*% B11
> M3 = A11 %*% ( B12 - B22 )
> M4 = A22 %*% ( B21 - B11 )
> M5 = ( A11 + A12 ) %*% B22
> M6 = ( A21 - A11 ) %*% ( B11 + B12 )
> M7 = ( A12 - A22 ) %*% ( B21 + B22 )
> # Step -3 -
```

```
> C11 = M1 + M4 - M5 + M7
> C12 = M3 + M5
> C21 = M2 + M4
> C22 = M1 - M2 + M3 + M6
> # Step -4 -
> C = rbind ( cbind ( C11 , C12 ) , cbind ( C21 , C22 ) )
> C

      [,1]  [,2]  [,3]  [,4]
[1,] 24884 25092  5345  5418
[2,] 23550 23131  6246  8101
[3,] 62973 58429 32015 34091
[4,] 35477 23925 15541  7345

> A%*%B

      [,1]  [,2]  [,3]  [,4]
[1,] 24884 25092  5345  5418
[2,] 23550 23131  6246  8101
[3,] 62973 58429 32015 34091
[4,] 35477 23925 15541  7345

> all(C==A%*%B)

[1] TRUE
```

7 Solving system of linear equations

Example 11 Matrices are closely related to systems of linear equations. Consider the two equations

$$x_1 + 3x_2 = 7$$
$$2x_1 + 4x_2 = 10$$

The system can be written in matrix form

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 7 \\ 10 \end{bmatrix} \text{ i.e. } Ax = b$$

Since $A^{-1}A = I$ and since $Ix = x$ we have

$$x = A^{-1}b = \begin{bmatrix} -2 & 1.5 \\ 1 & -0.5 \end{bmatrix} \begin{bmatrix} 7 \\ 10 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

A geometrical approach to solving these equations is as follows: Isolate x_2 in the equations:

$$x_2 = \frac{7}{3} - \frac{1}{3}x_1 \quad x_2 = \frac{1}{0}4 - \frac{2}{4}x_1$$

These two lines intersect at a point and gives the solution as $x_1 = 1, x_2 = 2$.

```
> A=matrix(c(1,2,3,4),ncol=2)
> b=c(7,10)
> x=solve(A) %*% b
> x

     [,1]
[1,]    1
[2,]    2
```

8 Eigenvalues and eigenvectors

8.1 Characteristic equation

Consider a square matrix A of dimension n. The scalar eigenvalue λ and an $n \times 1$ eigenvectors x of A are defined from a fundamental defining equation, sometimes called the characteristic equation.

$$Ax = \lambda x$$

where the eigenvector x must be further assumed to be nonzero. (why?) The Eq. (9.1) obviously holds true when $x = 0$ with all n zero elements. The defining equation for eigenvalues and eigenvectors rules out the trivial solution $x = 0$. Since $x \neq 0$, something else must be zero for Eq. to hold. It is not intuitively obvious, but true that the equation $Ax-$
$\lambda x = 0$ can hold when $(Ax - \lambda I_n)\, x = 0$, where I_n denotes the identity matrix, which must be inserted so that matrix subtraction of λ from A is conformable (makes sense). Since $x \neq 0$, we must make the preceding matrix $(Ax - \lambda I_n)$ somehow zero. We cannot make it a null matrix with all elements zero, because A is given to contain nonzero elements. However, we can make a scalar associated with the matrix zero. The determinant is a scalar associated with a matrix. Thus we require that the following determinantal equation holds.

$$\det\left(A - \lambda I_n\right) = 0$$

Let us use R to better understand the defining equation Eq. (9.1) by computing Eq. (9.2) for a simple example of a 2×2 matrix with elements 1 to 4. R computes the (first) eigenvalue to be $\lambda = 5.3722813$.

```
> A=matrix(1:4,2);A #view A

     [,1] [,2]
[1,]    1    3
[2,]    2    4

> ea=eigen(A) #object ea contains eigenvalues-vectors of A
> ea

eigen() decomposition
$values
```

```
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736

> lam=ea$val[1];lam #view lambda

[1] 5.372281

> x=ea$vec[,1];x # view x

[1] -0.5657675 -0.8245648

> det(A-lam*diag(2)) #should be zero

[1] 0

> lam^2-5*lam #solving quadratic OK if this is 2

[1] 2
```

Thus in the example $\det(A - \lambda I_n) = 0$. More explicitly we have:

$$\text{If } A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \text{ then } (\mathrm{A} - \lambda \mathrm{I_n}) = \begin{bmatrix} 1-\lambda & 3 \\ 2 & 4-\lambda \end{bmatrix}$$

The determinant can be written as $[(1-\lambda)(4-\lambda) - 2*3] = 0$, leading to a quadratic polynomial in λ as $[4 + \lambda^2 - \lambda - 4\lambda - 6] = 0$, with two roots, With two roots it is obvious that we have two eigenvalues (characteristic roots).

Clearly, R computations claim that 5.372281 is one of the roots of $[\lambda^2 - 5\lambda - 2] = 0$. A direct check on a root is obtained by simply substituting $\lambda = 5.372281$ in $[\lambda^2 - 5\lambda - 2]$ and seeing that it equals zero. The last line of the R snippet accomplishes this task of checking.

8.2 Eigenvectors

The R function 'eigen' provides all eigenvectors. For the 2×2 example of the earlier snippet, the two eigenvectors are available by issuing the R command: 'ea$vec.' as seen in the following output by R

```
> G=ea$vec; G #view matrix of eigenvectors

           [,1]       [,2]
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

8.3 Eigenvalues

Eigenvalues are a special set of scalars associated with a linear system of equations (i.e., a matrix equation) that are sometimes also known as characteristic roots, characteristic values.

```
> ea$values

[1]  5.3722813 -0.3722813
```

9 Matrix Decomposition

9.1 Gaussian Elimination Method

Gaussian elimination is a method for solving matrix equations of the form

$$Ax = b.$$

To perform Gaussian elimination starting with the system of equations

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}$$

compose the "augmented matrix equation"

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1k} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2k} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} & b_k \end{array}\right] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix}.$$

Here, the column vector in the variables $\mathbf{x}$ is carried along for labeling the matrix rows. Now, perform elementary row operations to put the augmented matrix into the upper triangular form

$$\left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1k} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2k} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{kk} & b'_k \end{array}\right].$$

A matrix that has undergone Gaussian elimination is said to be in echelon form.The echelon form of a matrix isn't unique, which means there are infinite answers possible when you perform row reduction. Reduced row echelon form is at the other end of the spectrum; it is unique, which means row-reduction on a matrix will produce the same answer no matter how you perform the same row operations.

Row Echelon Form

A matrix is in row echelon form if it meets the following requirements: - The first non-zero number from the left (the "leading coefficient") is always to the right of the first non-zero number in the row above. - Rows consisting of all zeros are at the bottom of the matrix.

$$\begin{bmatrix} 1 & a_0 & a_1 & a_2 & a_3 \\ 0 & 0 & 2 & a_4 & a_5 \\ 0 & 0 & 0 & 1 & a_6 \end{bmatrix}$$

Row Reduced Echelon Form

Reduced row echelon form is a type of matrix used to solve systems of linear equations. Reduced row echelon form has four requirements: - The first non-zero number in the first row (the leading entry) is the number 1. - The second row also starts with the number 1 , which is further to the right than the leading entry in the first row. For every subsequent row, the number 1 must be further to the right. - The leading entry in each row must be the only non-zero number in its column. - Any non-zero rows are placed at the bottom of the matrix.

$$\begin{bmatrix} 1 & 0 & a_1 & 0 & b_1 \\ 0 & 1 & a_2 & 0 & b_2 \\ 0 & 0 & 0 & 1 & b_3 \end{bmatrix}$$

How to perform Gaussian Elimination Method

Gaussian elimination is a way to find a solution to a system of linear equations. The basic idea is that you perform a mathematical operation on a row and continue until only one variable is left. For example, some possible row operations are:

- Interchange any two rows - Add two rows together. - Multiply one row by a non-zero constant (i.e. $1/3, -1, 5$) You can also perform more than one row operation at a time. For example, multiply one row by a constant and then add the result to the other row.

Following this, the goal is to end up with a matrix in reduced row echelon form where the leading coefficient, a 1 , in each row is to the right of the leading coefficient in the row above it. In other words, you need to get a 1 in the upper left corner of the matrix. The next row should have a 0 in position 1 and a 1 in position 2 . This gives you the solution to the system of linear

equations.

Gaussian Elimination Example
Solve the following system of linear equations using Gaussian elimination:

$$\begin{aligned} -x + 5y &= 7 \\ -2x - 7y &= -5 \end{aligned}$$

Step 1: Convert the equation into coefficient matrix form. In other words, just take the coefficient for the numbers and forget the variables for now:

$$\begin{bmatrix} 1 & 5 & 7 \\ -2 & -7 & -5 \end{bmatrix}$$

Step 2: Turn the numbers in the bottom row into positive by adding 2 times the first row:

$$\begin{bmatrix} 1 & 5 & 7 \\ 0 & 3 & 9 \end{bmatrix}$$

Step 3: Multiply the second row by 1/3. This gives you your second leading 1 :

$$\begin{bmatrix} 1 & 5 & 7 \\ 0 & 1 & 3 \end{bmatrix}$$

Step 4: Multiply row 2 by -5, and then add this to row 1 :

$$\begin{bmatrix} 1 & 0 & -8 \\ 0 & 1 & 3 \end{bmatrix}$$

That's it! In the first row, you have $x = -8$ and in the second row, $y = 3$. Note that x and y are in the same positions as when you converted the equation in step 1,50 all you have to do is read the solution:

$$\begin{bmatrix} 1 & 0 & -8 \\ 0 & 1 & (3) \end{bmatrix}$$

R code
In R, we can use *pracma* library to call the function ***rref*** which transforms the augmented matrix $A : b$ into row reduced echelon form and thus solves the system of linear equations using Gaussian Elimination Method.

```
> library(pracma)
> A = matrix(c(1,5,-2,-7),byrow=T,nrow=2,ncol=2)
> b = matrix(c(7,-5),nrow=2,ncol=1)
> rref(cbind(A, b))

     [,1] [,2] [,3]
[1,]    1    0   -8
[2,]    0    1    3
```

As we can see, we get the same result, so our method works correctly as it should be.

9.2 LU Decomposition

LU decomposition Every square matrix A can be decomposed into a product of a lower triangular matrix L and a upper triangular matrix U, as described in LU decomposition.

$$A = LU$$

It is a modified form of Gaussian elimination. While the Cholesky decomposition only works for symmetric, positive definite matrices, the more general LU decomposition works for any square matrix. There are several algorithms for calculating L and U. To derive Crout's algorithm for a 3×3 example, we have to solve the following system:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} = LU$$

We now would have to solve 9 equations with 12 unknowns. To make the system uniquely solvable, usually the diagonal elements of L are set to 1

$$\begin{aligned} l_{11} &= 1 \\ l_{22} &= 1 \\ l_{33} &= 1 \end{aligned}$$

so we get a solvable system of 9 unknowns and 9 equations.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} =$$

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ u_{11}l_{21} & u_{12}l_{21} + u_{22} & u_{13}l_{21} + u_{23} \\ u_{11}l_{31} & u_{12}l_{31} + u_{22}l_{32} & u_{13}l_{31} + u_{23}l_{32} + u_{33} \end{pmatrix} = LU$$

Solving for the other l and u, we get the following equations:

$$\begin{aligned} u_{11} &= a_{11} \\ u_{12} &= a_{12} \\ u_{13} &= a_{13} \\ u_{22} &= a_{22} - u_{12}l_{21} \\ u_{23} &= a_{23} - u_{13}l_{21} \\ u_{33} &= a_{33} - (u_{13}l_{31} + u_{23}l_{32}) \end{aligned}$$

and for l :

$$\begin{aligned} l_{21} &= \frac{1}{u_{11}} a_{21} \\ l_{31} &= \frac{1}{u_{11}} a_{31} \\ l_{32} &= \frac{1}{u_{22}} (a_{32} - u_{12}l_{31}) \end{aligned}$$

We see that there is a calculation pattern, which can be expressed as the following formulas, first for U

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} u_{kj}l_{ik}$$

and then for L

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} u_{kj}l_{ik} \right)$$

We see in the second formula that to get the l_{ij} below the diagonal, we have to divide by the diagonal element (pivot) u_{jj}, so we get problems when u_{jj} is either 0 or very small, which leads to numerical instability. The solution to this problem is pivoting A, which means rearranging the rows of A, prior to the LU decomposition, in a way that the largest element of each column gets onto the diagonal of A. Rearranging the rows means to multiply A by a permutation matrix P :

$$PA \Rightarrow A'$$

Example:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 4 \\ 2 & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix}$$

The decomposition algorithm is then applied on the rearranged matrix so that

$$PA = LU$$

In R, The easiest way to do this is to use the *lu* function in the ***Matrix package***. This will give you a slight variant on the A=LU decomposition. You'll get A=PLU instead, where P is a permutation matrix. Note LU decomposition can only be performed on invertible matrices.

```
> library(Matrix)
> mat = matrix(c(1,2,3,4,5,6,5,4,3,2,1,2,-3,2,5,20), ncol = 4)
> det(mat) # -144 => matrix is not singular

[1] -144

> mat_lu = lu(mat)
> expand(mat_lu) #contains P, L, U

$L
4 x 4 Matrix of class "dtrMatrix" (unitriangular)
     [,1]       [,2]       [,3]       [,4]
[1,]  1.0000000          .          .          .
[2,]  0.5000000  1.0000000          .          .
[3,]  0.2500000  1.0000000  1.0000000          .
[4,]  0.7500000  0.5000000 -0.6666667  1.0000000

$U
4 x 4 Matrix of class "dtrMatrix"
     [,1] [,2] [,3] [,4]
[1,]  4.0  4.0  2.0 20.0
[2,]    .  4.0  1.0 -8.0
[3,]    .    .  1.5  0.0
[4,]    .    .    . -6.0

$P
4 x 4 sparse Matrix of class "pMatrix"
```

```
[1,] . . | .
[2,] . | . .
[3,] . . . |
[4,] | . . .
```

9.3 Cholesky Decomposition

Cholesky decompositions $\left(A = U^T U\right)$ only work with matrices that are - real - symmetric - positive definite (i.e. symmetric and only has positive eigenvalues, or symmetric and only has positive pivots) - square If you're getting a strange error when calculating the Cholesky decomposition of a matrix, it's probably because it has negative eigenvalues.

In R, the ***Matrix*** package as *chol()* function which is used for cholesky decomposition of a real, symmetric, positive definite square matrix.

```
> library(Matrix)
> x = matrix(c(8,5,5,4), nrow = 2)
> y = matrix(c(8,6,6,4), nrow = 2)  #y marix has negative eignevalues, so chol(y
> eigen(x) # positive eigenvalues

eigen() decomposition
$values
[1] 11.3851648  0.6148352

$vectors
           [,1]       [,2]
[1,] -0.8280672  0.5606288
[2,] -0.5606288 -0.8280672

> eigen(y) # negative eigenvalues

eigen() decomposition
$values
[1] 12.3245553 -0.3245553

$vectors
           [,1]       [,2]
[1,] -0.8112422  0.5847103
[2,] -0.5847103 -0.8112422
```

```
> chol(x) # returns stuff

         [,1]      [,2]
[1,] 2.828427 1.7677670
[2,] 0.000000 0.9354143
```

You can also find the inverse of a matrix using the Cholesky decomposition:

```
> #Two ways to find the inverse
> x = matrix(c(8,5,5,4), nrow = 2)
> x_chol = chol(x)
> #usual way of finding an inverse
> solve(x)

           [,1]       [,2]
[1,]  0.5714286 -0.7142857
[2,] -0.7142857  1.1428571

> #using the Cholesky decomposition
> chol2inv(x_chol)

           [,1]       [,2]
[1,]  0.5714286 -0.7142857
[2,] -0.7142857  1.1428571

> #check the two are the same
> solve(x) - chol2inv(x_chol) #pretty much 0

              [,1]          [,2]
[1,]  3.330669e-16 -3.330669e-16
[2,] -4.440892e-16  2.220446e-16
```

9.4 QR Decomposition

The QR decomposition (also called the QR factorization) of a matrix is a decomposition of the matrix into an orthogonal matrix and a triangular matrix. A QR decomposition of a real square matrix A is a decomposition of A as

$$A = QR,$$

where Q is an orthogonal matrix (i.e. $Q^TQ = I$) and R is an upper triangular matrix. If A is nonsingular, then this factorization is unique.

There are several methods for actually computing the QR decomposition. One of such method is the Gram-Schmidt process.

Gram-Schmidt process

Consider the GramSchmidt procedure, with the vectors to be considered in the process as columns of the matrix A. That is,

$$A = \left[\begin{array}{c|c|c|c} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{array} \right]$$

Then,

$$\begin{aligned} \mathbf{u}_1 &= \mathbf{a}_1, \quad \mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\ \mathbf{u}_2 &= \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{e}_1)\,\mathbf{e}_1, \quad \mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\ \mathbf{u}_{k+1} &= \mathbf{a}_{k+1} - (\mathbf{a}_{k+1} \cdot \mathbf{e}_1)\,\mathbf{e}_1 - \cdots - (\mathbf{a}_{k+1} \cdot \mathbf{e}_k)\,\mathbf{e}_k, \qquad \mathbf{e}_{k+1} = \frac{\mathbf{u}_{k+1}}{\|\mathbf{u}_{k+1}\|} \end{aligned}$$

Note that $\|\cdot\|$ is the L_2 norm.

QR Factorization

The resulting QR factorization is

$$A = \left[\begin{array}{c} \\ \mathbf{a}_1 \,|\mathbf{a}_2|\cdots|\, \mathbf{a}_n \\ \\ \end{array} \right] = \left[\begin{array}{c|c|c|c} \mathbf{e}_1 & |\mathbf{e}_2| & \cdots & |\, \mathbf{e}_n \end{array} \right] \left[\begin{array}{cccc} \mathbf{a}_1 \cdot \mathbf{e}_1 & \mathbf{a}_2 \cdot \mathbf{e}_1 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_1 \\ 0 & \mathbf{a}_2 \cdot \mathbf{e}_2 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{a}_n \cdot \mathbf{e}_n \end{array} \right] = QR$$

Note that once we find $\mathbf{e}_1, \ldots, \mathbf{e}_n$, it is not hard to write the QR factorization.

2 Example Consider the matrix

$$A = \left[\begin{array}{ccc} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{array} \right]$$

with the vectors $\mathbf{a}_1 = (1,1,0)^T, \mathbf{a}_2 = (1,0,1)^T, \mathbf{a}_3 = (0,1,1)^T$. Note that all the vectors considered above and below are column vectors. From now on, I will drop ^{T} notation for simplicity, but we have to remember that all the

vectors are column vectors. Performing the Gram-Schmidt procedure, we obtain:

$$\mathbf{u}_1 = \mathbf{a}_1 = (1,1,0),$$
$$\mathbf{e}_1 = \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} = \frac{1}{\sqrt{2}}(1,1,0) = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right),$$
$$\mathbf{u}_2 = \mathbf{a}_2 - (\mathbf{a}_2 \cdot \mathbf{e}_1)\,\mathbf{e}_1 = (1,0,1) - \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right) = \left(\frac{1}{2}, -\frac{1}{2}, 1\right),$$
$$\mathbf{e}_2 = \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} = \frac{1}{\sqrt{3/2}}\left(\frac{1}{2}, -\frac{1}{2}, 1\right) = \left(\frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}\right),$$
$$\mathbf{u}_3 = \mathbf{a}_3 - (\mathbf{a}_3 \cdot \mathbf{e}_1)\,\mathbf{e}_1 - (\mathbf{a}_3 \cdot \mathbf{e}_2)\,\mathbf{e}_2$$
$$= (0,1,1) - \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right) - \frac{1}{\sqrt{6}}\left(\frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}, \frac{2}{\sqrt{6}}\right) = \left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right),$$
$$\mathbf{e}_3 = \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} = \left(-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right).$$

Thus,

$$Q = [\mathbf{e}_1 \,|\mathbf{e}_2| \cdots |\, \mathbf{e}_n] = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{bmatrix}$$

$$R = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{e}_1 & \mathbf{a}_2 \cdot \mathbf{e}_1 & \mathbf{a}_3 \cdot \mathbf{e}_1 \\ 0 & \mathbf{a}_2 \cdot \mathbf{e}_2 & \mathbf{a}_3 \cdot \mathbf{e}_2 \\ 0 & 0 & \mathbf{a}_3 \cdot \mathbf{e}_3 \end{bmatrix} = \begin{bmatrix} \frac{2}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & \frac{3}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ 0 & 0 & \frac{2}{\sqrt{3}} \end{bmatrix}$$

In R, we can use qr() function to decompose a matrix A into a product A=QR where Q is a matrix with unit norm orthogonal vectors and R is an upper triangular matrix.

```
> A = matrix(c(1,1,0,1,0,1,0,1,1),byrow =TRUE, nrow=3, ncol=3)  #take the matrix
> QR = qr(A) #created a variable QR for ease
> QR$rank  #find the rank

[1] 3

> Q = qr.Q(QR); Q  #find the Q factor
```

```
           [,1]       [,2]       [,3]
[1,] -0.7071068  0.4082483 -0.5773503
[2,] -0.7071068 -0.4082483  0.5773503
[3,]  0.0000000  0.8164966  0.5773503

> R <- qr.R(QR); R  #find the R factor

          [,1]       [,2]       [,3]
[1,] -1.414214 -0.7071068 -0.7071068
[2,]  0.000000  1.2247449  0.4082483
[3,]  0.000000  0.0000000  1.1547005

> qr.X(QR)  #reconstruct A

     [,1] [,2] [,3]
[1,]    1    1    0
[2,]    1    0    1
[3,]    0    1    1
```

9.5 Singular Value Decomposition

SVD (Singular Value Decomposition) stands for splitting a matrix A into a product $A = USV^H$ where U and V are unitary matrices and S is a diagonal matrix consisting of singular values on its main diagonal arranged in non-increasing order where all the singular values are non-negative. Computing the singular value decomposition of a matrix:

```
> A = matrix(c(1, 2, -1, 3, -6, 9, -1, 2, 1), nrow=3)
> svd(A)  #using the inbuilt function svd()

$d
[1] 11.355933  2.475195  1.707690

$u
           [,1]       [,2]       [,3]
[1,]  0.2526715 -0.1073216 -0.9615816
[2,] -0.5565826  0.7968092 -0.2351827
[3,]  0.7914373  0.5946235  0.1415976
```

```
$v
            [,1]      [,2]       [,3]
[1,] -0.14546854 0.3602437 -0.9214464
[2,]  0.98806904 0.1005140 -0.1166899
[3,] -0.05058143 0.9274273  0.3705672

> svd(A)$d # find the diagonal entries of S matrix

[1] 11.355933  2.475195  1.707690

> svd(A)$u  #u matrix

           [,1]       [,2]       [,3]
[1,]  0.2526715 -0.1073216 -0.9615816
[2,] -0.5565826  0.7968092 -0.2351827
[3,]  0.7914373  0.5946235  0.1415976

> svd(A)$v   #v matrix

            [,1]      [,2]       [,3]
[1,] -0.14546854 0.3602437 -0.9214464
[2,]  0.98806904 0.1005140 -0.1166899
[3,] -0.05058143 0.9274273  0.3705672

> svd(A)$d[1]     #largest singular value

[1] 11.35593

> tail(svd(A)$d, n=1)    #smallest singular value

[1] 1.70769

>
```