Final Report, Gaussian Elimination by LU Decomposition With Permutation Matrix

Zubair Mukhi
MATH 201, Case Western Reserve University

May 1, 2020

Contents

1 Introduction

3.

A can be written as
$\begin{bmatrix}0 & 1\\ 1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}2 & 2 & 2 & 2\\ 1 & 2 & 1 & 2\end{bmatrix}$
this can further be simplified to
$\begin{bmatrix}1\\ 1\\ 1\end{bmatrix}\begin{bmatrix}2 & 2 & 2 & 2\end{bmatrix} - \begin{bmatrix}1\\ 0\\ 1\end{bmatrix}\begin{bmatrix}1 & 0 & 1 & 1\end{bmatrix}$
B can be written as
$\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}1 & 3 & 3\\ 1 & 2 & 2\end{bmatrix}$
this can be simplfied to
$\begin{bmatrix}1\\ 1\end{bmatrix}\begin{bmatrix}1 & 3 & 3\end{bmatrix} - \begin{bmatrix}1\\ 0\end{bmatrix}\begin{bmatrix}0 & 1 & 1\end{bmatrix}$

4.

$BB^T = \begin{bmatrix}9 & 13\\ 13 & 19\end{bmatrix}$, determinant is 2, trace is 28
$B^TB = \begin{bmatrix}2 & 5 & 5\\ 5 & 13 & 13\\ 5 & 13 & 13\end{bmatrix}$, determinant is 0, trace is 28

B is compressible since the sigma values are fairly small

6.

$(2-\lambda)(2-\lambda)-4=0$
$4-4\lambda+\lambda^2+4=0$
$\lambda=0,4$
Singular values:
$(20-\lambda)(5-\lambda)-100=0$
$25\lambda+\lambda^2=0$
$\lambda=0,25$
$\sigma_1=5,\sigma_2=0$
We know that the eigenvectors of A^TA are orthogonal because it is by definition a symmetric matrix.

A Source Code

```
import numpy as np


def solver():
    print('this is a solver for Ax=b via Gaussian elimination, providing x, L, and U.\n'
          'It currently only accepts square matrices.\n'
          'The solver will move the largest value in the first row to the diagonal, solving
          'it holds the permutations in an internal matrix, p.')
    print('please enter rows as numbers separated by commas, no spaces.')
    n_rows = int(input("How many rows?\n"))
    arr_p = np.identity(n_rows)
    arr_l = np.identity(n_rows)
    arr_a = []
    for i in range(0, n_rows):
        row = (input('please enter row {} of A\n'.format(i+1)))
        row = row.split(',')
        while len(row) != n_rows:
            print('invalid input, please re-enter row.')
            row = (input('please enter row {} of A\n'.format(i + 1)))
            row = row.split(',')
        for r in range(0, len(row)):
            row[r] = float(row[r])
        arr_a.append(row)
    arr_a = np.array(arr_a, dtype=np.double)
    b_string = input('please enter b\n')
    arr_b = b_string.split(',')
    for e in range(0, len(arr_b)):
        arr_b[e] = float(arr_b[e])
```

```
    while len(arr_b) != n_rows:
        print('invalid b, please try again.')
        b_string = input('please enter b\n')
        arr_b = int(b_string.split(','))
    arr_b = np.array(arr_b, dtype=np.double)
    for i in range(0, arr_a.shape[0]):
        reorder(arr_a, arr_b, arr_p, i)
    arr_u, arr_c, arr_l = generate_u(arr_a, arr_b, arr_l, n_rows)
    print('L: {}\nU: {}\nc: {}'.format(arr_l, arr_u, arr_c))
    arr_x = solve(arr_u, arr_c)
    print('x = {}'.format(arr_x))


def reorder(a, b, p, ptr):
    pointer = ptr
    for i in range(ptr, a.shape[0]):
        if a[i][ptr] > a[pointer][ptr]:
            pointer = i
            # print(i)
    a[[ptr, pointer]] = a[[pointer, ptr]]
    b[[ptr, pointer]] = b[[pointer, ptr]]
    p[[ptr, pointer]] = p[[pointer, ptr]]


def generate_u(arr_a, arr_b, arr_l, n):
    u = np.copy(arr_a)
    c = np.copy(arr_b)
    for diagonal in range(0, n):
        pivot = u[diagonal][diagonal]
        for i in range(diagonal+1, u.shape[0]):
            if pivot != 0:
                prod = u[i][diagonal] / pivot
                u[i] -= prod * u[diagonal]
                c[i] -= prod * c[diagonal]
                arr_l[i][diagonal] = prod
            else:
                pass  # avoids division by zero
    return u, c, arr_l


def solve(arr_u, arr_c):
    m = arr_u.shape[0]
    u = arr_u
    arr_x = np.zeros_like(arr_c)
    det = np.linalg.det(arr_u)
    if det != 0:  # this matrix has one solution
```

```
        for ptr in reversed(range(0, m)):
            arr_x[ptr] = arr_c[ptr] / u[ptr][ptr]
            for i in range(0, ptr):
                arr_c[i] = arr_c[i] - arr_x[ptr] * u[i][ptr]
        return arr_x
    else:
        for ptr in reversed(range(0, m)):
            if u[ptr][ptr] != 0:
                arr_x[ptr] = arr_c[ptr] / u[ptr][ptr]
                for i in range(0, ptr):
                    arr_c[i] = arr_c[i] - arr_x[ptr] * u[i][ptr]
            elif arr_c[ptr] != 0:
                return 'No Solution'
            else:
                return infinite_sols(u, arr_x, arr_c, ptr)
        return arr_x


def infinite_sols(arr_u, arr_x, arr_c, p):
    arr_x_neg = np.copy(arr_x)
    arr_c_neg = np.copy(arr_c)
    arr_x[p] = np.double(1)
    arr_x_neg[p] = np.double(-1)
    for i in range(0, p):
        arr_c[i] = arr_c[i] - arr_x[p] * arr_u[i][p]
        arr_c_neg[i] = arr_c_neg[i] - arr_x_neg[p] * arr_u[i][p]
    for ptr in reversed(range(0, p)):
        arr_x[ptr] = arr_c[ptr] / arr_u[ptr][ptr]
        for i in range(0, ptr):
            arr_c[i] = arr_c[i] - arr_x[ptr] * arr_u[i][ptr]
    for ptr in reversed(range(0, p)):
        arr_x_neg[ptr] = arr_c_neg[ptr] / arr_u[ptr][ptr]
        for i in range(0, ptr):
            arr_c_neg[i] = arr_c_neg[i] - arr_x_neg[ptr] * arr_u[i][ptr]
    return 'infinitely many solutions', arr_x, arr_x_neg


if __name__ == '__main__':
    solver()
```