Implementing Gaussian Elimination in Python

Walter Nam (wkn2)

May 1, 2020

1 Introduction

Gaussian elimination is a method in linear algebra to solve a series of linear equations. The process involves performing a series of operations on the corresponding coefficients matrix (also known as the A matrix). Gaussian elimination can also be used to calculate the rank of a matrix, find the determinant of a matrix, and calculate the inverse of a square matrix. To perform row reduction, a series of elementary row operations must be utilized to transform the given matrix to one whose lower left corner is filled with zeros, also known as an "upper triangular" matrix. Such row operations include: swapping rows, multiplying a row by a scalar, and adding a multiple of one row to another row. The solution vector can be derived by using a process called "back-substitution", where the known variables are substituted into other equations to solve for the rest of the unknowns. A special type of reduced matrix, called reduced row echelon form, can be derived using gaussian. This matrix is an upper triangular where all the leading coefficients of the rows are equal to 1, and each column containing a leading 1 has zeros in all other entries. Transforming an augmented matrix into this form makes back-substitution unnecessary, as the modified b vector becomes equivalent to the solution vector.

2 Algorithm and Row Operations

We can apply gaussian elimination to find the solution vectors of the following example matrices. An explanation of the algorithm and the row reduction operations are also provided below.

a)

$$A = \begin{bmatrix} 1 & -1 & 2 & -1 \\ 2 & -2 & 3 & -3 \\ 1 & 1 & 1 & 0 \\ 1 & -1 & 4 & 3 \end{bmatrix} b = \begin{bmatrix} -8 \\ -20 \\ -2 \\ 4 \end{bmatrix}$$

b)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} b = \begin{bmatrix} 4 \\ 6 \\ 6 \end{bmatrix}$$

c)

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} b = \begin{bmatrix} 4 \\ 4 \\ 6 \end{bmatrix}$$

To solve the system of equations, the number of equations should be equivalent to the number of unknowns. We can express the system in matrix form, with A representing the coefficients of the unknowns, and b representing the constant terms of the equations. In reducing the coefficients matrix, the first row will be fixed, as it acts as a reference row for elimination. The elements of the first column below the first row will be transformed to zero, and the elements of the rows below the reference are multiplied by a factor. The factor is defined as the main diagonal element of the reference row divided by the first element of the corresponding row. Thus, subtracting these rows from the reference row will result in all the elements below the reference element of the first column to be equal to 0. The python algorithm traverses the matrix to perform this operation until reaching the $n - 1$ column. However, in case of 0 values located in the main diagonal, this algorithm implements partial pivoting to swap rows. The pivot element is defined as the first non zero element of the reference row, and determines the factor that will be multiplied to the rows that will be subtracted from the reference row. To avoid division by 0, partial pivoting involves a simple swap of the row containing a 0 pivot, with a another non-zero pivot row below it. The corresponding row in the b vector should also be swapped accordingly. This process is called ”partial pivoting” because the columns are not swapped in the process.

```
for i in range (n-1):
    if np.fabs(A[i,i]) < sys.maxsize:
        for j in range(i+1, n):
            if np.fabs(A[j,i]) > np.fabs(A[i,i]):
                A[[i,j]] = A[[j,i]]
                b[[i,j]] = b[[j,i]]
                break
    for k in range (i+1, n):
        if A[k,i] == 0: continue
        factor = A[i,i] / A[k,i]
        for j in range(i,n):
            A[k,j] = A[i,j] - A[k,j] * factor
        b[k] = b[i] - b[k] * factor
```

Figure 1: Three nested for loops demonstrating partial pivoting. i represents the index of the fixed row, j represents the indices of the rows below the reference, and k represents the indices of the columns. The outermost loop acts as the index between the fixed rows and eliminated columns, whereas the next inner loop applies the elimination to rows below the reference or "fixed" row. This loop calculates the factor for each row and the corresponding b element. The inner most j loop performs the calculations of the element of each row on the corresponding column. The algorithm skips elimination for any rows containing a 0 element under the main diagonal and applies partial pivoting. This process begins by checking the absolute value of the pivot element and examining its convergence to zero. The inner j loop compares the absolute values of the row elements below the reference row and executes a swap accordingly.

```
#Backward substitution
x[n - 1] = b[n - 1] / A[n - 1, n - 1]
for i in range(n - 2, -1, -1):
    sum_ax = 0
    for j in range(i + 1, n):
        sum_ax += A[i, j] * x[j]
    x[i] = (b[i] - sum_ax) / A[i,i]

return A, b, x
```

Figure 2: Back substitution algorithm starts calculating $x_n - 1$ then uses two nested for loops. The outer loop indexes the up to the second to last column of the matrix. The inner loop calculates the sum of the x element multiplied by the corresponding coefficient in the A matrix. The algorithm then plugs the value to solve for the unknowns in the upper equations from $n - 1$. The tables below demonstrate all the row operations performed in reducing the example matrices.

Augmented matrix	Row operations
$\left[\begin{array}{cccc\|c} 1 & -1 & 2 & -1 & -8 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 2 & -1 & 1 & 6 \\ 0 & 0 & 2 & 4 & 12 \end{array}\right]$	$R_2 - 2R_1 \to R_2,\ R_3 - R_1 \to R_3,\ R_4 - R_1 \to R_4$
$\left[\begin{array}{cccc\|c} 1 & -1 & 2 & -1 & -8 \\ 0 & 2 & -1 & 1 & 6 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 0 & 2 & 4 & 12 \end{array}\right]$	$R_2 \leftrightarrow R_3$
$\left[\begin{array}{cccc\|c} 1 & -1 & 2 & -1 & -8 \\ 0 & 1 & -\frac{1}{2} & \frac{1}{2} & 3 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 0 & 2 & 4 & 12 \end{array}\right]$	$\frac{R_2}{2} \to R_2$
$\left[\begin{array}{cccc\|c} 1 & 0 & \frac{3}{2} & -\frac{1}{2} & -5 \\ 0 & 1 & -\frac{1}{2} & \frac{1}{2} & 3 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 0 & 2 & 4 & 12 \end{array}\right]$	$R_1 + R_2 \to R_1$
$\left[\begin{array}{cccc\|c} 1 & 0 & \frac{3}{2} & -\frac{1}{2} & -5 \\ 0 & 1 & -\frac{1}{2} & \frac{1}{2} & 3 \\ 0 & 0 & 1 & 1 & 4 \\ 0 & 0 & 2 & 4 & 12 \end{array}\right]$	$\frac{R_3}{-1} \to R_3$
$\left[\begin{array}{cccc\|c} 1 & 0 & 0 & -2 & -11 \\ 0 & 1 & 0 & 1 & 5 \\ 0 & 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 2 & 4 \end{array}\right]$	$R_1 - \frac{3}{2}R_3 \to R_1,\ R_2 + \frac{1}{2}R_3 \to R_2,\ R_4 - 2R_3 \to R_4$
$\left[\begin{array}{cccc\|c} 1 & 0 & 0 & -2 & -11 \\ 0 & 1 & 0 & 1 & 5 \\ 0 & 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 1 & 2 \end{array}\right]$	$\frac{R_4}{2} \to R_4$
$\left[\begin{array}{cccc\|c} 1 & 0 & 0 & 0 & -7 \\ 0 & 1 & 0 & 0 & 3 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 & 2 \end{array}\right]$	$R_1 + 2R_4 \to R_1,\ R_2 - R_4 \to R_2,\ R_3 - R_4 \to R_3$

Table 1: Stages of the first example augmented matrix and its corresponding elementary row operations.

Augmented matrix	Row operations
$\left[\begin{array}{ccc\|c} 1 & 1 & 1 & 4 \\ 0 & 0 & -1 & -2 \\ 0 & 0 & 1 & 2 \end{array}\right]$	$R_2 - 2R_1 \to R_2,\ R_3 - R_1 \to R_3$
$\left[\begin{array}{ccc\|c} 1 & 1 & 1 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 2 \end{array}\right]$	$\frac{R_2}{-1} \to R_2$
$\left[\begin{array}{ccc\|c} 1 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 \end{array}\right]$	$R_1 - R_2 \to R_1,\ R_3 - R_2 \to R_3$

Table 2: Stages of the second example augmented matrix and its corresponding elementary row operations. Note that x_1 and x_2 are free variables, meaning this system has infinitely many solutions.

Augmented matrix	Row operations
$\left[\begin{array}{ccc\|c} 1 & 1 & 1 & 4 \\ 0 & 0 & -1 & -4 \\ 0 & 0 & 1 & 2 \end{array}\right]$	$R_2 - 2R_1 \to R_2,\ R_3 - R_1 \to R_3$
$\left[\begin{array}{ccc\|c} 1 & 1 & 1 & 4 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 1 & 2 \end{array}\right]$	$\frac{R_2}{-1} \to R_2$
$\left[\begin{array}{ccc\|c} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & -2 \end{array}\right]$	$R_1 - R_2 \to R_1,\ R_3 - R_2 \to R_3$

Table 3: Stages of the third example augmented matrix and its corresponding elementary row operations. This system has no solution, as 0 != -2.

In [14]:
```python
#Author: Walter Nam
import sys
import pprint
import numpy as np
import scipy as sp
import sympy
from scipy import linalg
```

In [15]:
```python
class Data:

    def __init__(self, A, b):
        self.A = A
        self.b = b

    def computed_sol(self, A, b):
        print("Numpy solution")
        try:
            return np.linalg.solve(A,b)
        except np.linalg.LinAlgError:
            print("No solution")
            return None

    def gaussian(self, A, b):
        n = len(b)
        x = np.zeros(n, float)
        for i in range (n-1):
            if np.fabs(A[i,i]) < sys.maxsize:
                for j in range(i+1, n):
                    if np.fabs(A[j,i]) > np.fabs(A[i,i]):
                        A[[i,j]] = A[[j,i]]
                        b[[i,j]] = b[[j,i]]
                        break
            for k in range (i+1, n):
                if A[k,i] == 0: continue
                factor = A[i,i] / A[k,i]
                for j in range(i,n):
                    A[k,j] = A[i,j] - A[k,j] * factor
                b[k] = b[i] - b[k] * factor

        #Backward substitution
        x[n - 1] = b[n - 1] / A[n - 1, n - 1]
        for i in range(n - 2, -1, -1):
            sum_ax = 0
            for j in range(i + 1, n):
                sum_ax += A[i, j] * x[j]
            x[i] = (b[i] - sum_ax) / A[i,i]

        return A, b, x

    def check_valid_solution(self, A, b, gauss, free_variable_solution):
        solution = gauss(A, b)[2]
        b_0 = gauss(A, b)[1]
        A_0 = gauss(A, b)[0]
        if np.isnan(solution[0]):
            print("There are an infinite number of solutions")
            print("Solution with free variable = 1")
            fv = 1
            free_variable_solution(A_0, b_0, fv, solution)
            print("Solution with free variable = -1")
            fv = -1
            free_variable_solution(A_0, b_0, fv, solution)
            return ""
        elif np.isinf(solution[0]):
            print("No Solution")
            return ""
        else:
            print("Solution vector")
            print(solution)
            return ""

    def free_variable_solution(self, A_0, b_0, free_variable, solution_vector):
        x2 = free_variable
        x3 = solution_vector[len(solution_vector) - 1]
        x = sympy.symbols('x')
        expr = b_0[0] - A_0[0][0]*x - A_0[0][1]*x2  - A_0[0][2]*x3
        x1 = sympy.solve(expr)
        print(np.array([x1[0], x2, x3], float))

    def lu_decomp(self, A):
        (P, L, U) = sp.linalg.lu(A)
        print("L factor:")
        pprint.pprint(L)
        print("U factor:")
        pprint.pprint(U)
        return ""
```

In [16]:
```python
A = np.array([
    [1,-1,2,-1],
    [2,-2,3,-3],
    [1,1,1,0],
    [1,-1,4,3]
    ],
    float)

b = np.array([-8,-20,-2,4], float)
data = Data(A, b)
print(data.check_valid_solution(A, b, data.gaussian, data.free_variable_solution))
print(data.lu_decomp(A))
```

```
Solution vector
[-7.  3.  2.  2.]

L factor:
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0., -0.,  1.,  0.],
       [ 0., -0., -0.,  1.]])
U factor:
array([[ 2., -2.,  3., -3.],
       [ 0., -4.,  1., -3.],
       [ 0.,  0., -5., -9.],
       [ 0.,  0.,  0., -4.]])
```

In [17]:
```python
A = np.array([
    [1,1,1],
    [2,2,1],
    [1,1,2]
    ],
    float)

b = np.array([4,6,6], float)
data = Data(A, b)
print(data.check_valid_solution(A, b, data.gaussian, data.free_variable_solution))
print(data.lu_decomp(A))
```

```
There are an infinite number of solutions
Solution with free variable = 1
[1. 1. 2.]
Solution with free variable = -1
[ 3. -1.  2.]

L factor:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
U factor:
array([[ 2.,  2.,  1.],
       [ 0.,  0., -1.],
       [ 0.,  0., -3.]])
```

```
C:\Users\aircr\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: RuntimeWarning: invalid value encountered in dou
ble_scalars
```

In [18]:
```python
A = np.array([
    [1,1,1],
    [2,2,1],
    [1,1,2]
    ],
    float)

b = np.array([4,4,6], float)
data = Data(A, b)
print(data.check_valid_solution(A, b, data.gaussian, data.free_variable_solution))
print(data.lu_decomp(A))
```

```
No Solution

L factor:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
U factor:
array([[ 2.,  2.,  1.],
       [ 0.,  0., -1.],
       [ 0.,  0., -3.]])
```

```
C:\Users\aircr\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: RuntimeWarning: divide by zero encountered in do
uble_scalars
```