Numerical Analysis

2023/1

Fernando Deeke Sasse

CCT - UDESC

Gaussian Elimination - 1

1. Linear Systems

We will define functions in Python that perform elementary operations on rows of a matrix that will be useful for solving systems of linear equations or simply, linear systems. A linear system of equations is a set of linear equations coupled together by means of variables $x_0, \ldots, x_n$, as follows:

$$
\begin{aligned}
a_{00}x_0 + a_{01}x_2 + \cdots + a_{0n}x_n &= b_0 && (1)\\
a_{10}x_0 + a_{11}x_2 + \cdots + a_{1n}x_n &= b_1 && (2)\\
&\vdots && (3)\\
a_{m0}x_0 + a_{m1}x_2 + \cdots + a_{mn}x_n &= b_m && (4)\\
& && (5)
\end{aligned}
$$

Unlike the more common indicial notation, which starts at 1, we will use the convention in use in Python, where the first index is 0. In matrix notation the linear system of $n+1$ and $n+1$ unknowns is represented in the form $AX = B$, where

$$
A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0n} \\ a_{10} & a_{11} & \cdots & a_{1n} \\ \vdots & & & \vdots \\ a_{n0} & a_{n1} & \cdots & a_{nn} \end{bmatrix}, \quad X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad B = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}
$$

What we call the augmented matrix associated with the linear system $AX = B$ is the matrix defined by

$$
[A|B] = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0n} & b_0 \\ a_{10} & a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & & & \vdots & \\ a_{n0} & a_{n1} & \cdots & a_{nn} & b_2 \end{bmatrix}
$$

The Gaussian elimination method for linear systems, which we will deal with below, makes use of the augmented matrix.

2. Gaussian Elimination - Introduction

The so-called Gaussian elimination method is the most widely used direct method for solving linear systems and is the basis for all variants of elimination methods. The method described here is also called *simple Gaussian elimination*, to distinguish it from more improved (but computationally slower) ones. It consists of two steps:

1. Elimination phase: we perform the elimination of the elements below the main diagonal of the enlarged matrix $[A|B]$ by means of elementary transformations on matrices, until we obtain the staggered form.
2. Backsubstitution.

We will describe each of these steps below:

(a) Elimination phase

In this phase we use the so-called *elementary operations on lines* of the enlarged matrix. The idea is that the rows of the matrix can be multiplied by factors, summed together and exchanged in order, without the solution of the corresponding linear system being changed. The idea is to perform elementary operations in order to place the matrix in the so called echelon form:

$$[A|B] = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0n} & b_0 \\ 0 & a_{11} & \cdots & a_{1n} & b_1 \\ \vdots & & & \vdots & \vdots \\ 0 & 0 & \cdots & a_n & b_m \end{bmatrix}$$

The corresponding system now has the general form:

$$a_{00}x_0 + a_{01}x_2 + \cdots + a_{0n}x_n = b_0 \tag{6}$$
$$a_{11}x_2 + \cdots + a_{1n}x_n = b_1 \tag{7}$$
$$\vdots \tag{8}$$
$$a_n x_n = b_n \tag{9}$$
$$\tag{10}$$

In this way the solution can be easily obtained through the back-replacement process, described below:

(b) Back-substitution phase

We start solving the corresponding last equation of the reduced system, i.e.

$$x_n = \frac{b_n}{a_{nn}}, .$$

Let us now consider the stage of back-substitution where $x_n, x_{n-1}, \ldots, x_{k+1}$ have already been computed and we must compute x_k, from the equation k:

$$a_{kk}x_k + a_{kk+1}x_{k+1} + \cdots + a_{kn}x_n = b_k\,.$$

Solving for x_k we get

$$x_k = \frac{1}{a_{kk}}(b_k - a_{kk+1}x_{k+1} - \cdots - a_{kn}x_n) = \frac{1}{a_{kk}}\left(b_k - \sum_{j=k+1}^{n} a_{kj}x_j\right)\,.$$

The implementation of this method in Python will be covered later. Before that we will examine in more detail the process of elementary operations on lines in Python.

3. Elementary row operations

Elementary row operations include:

1. Add λ times to the j line to the i line.
2. Multiply the line i by the scale λ.
3. Exchange the lines i and j with each other.

Each of the elementary line operations can be understood as the result of a matrix multiplication to the left by an elementary matrix.

To add k times the i row to the j row in a A array, we multiply A by the E array by E being equal to the identity array, except for the component $E_{ij} = k$.

For example, if A is the enlarged array given by

$$A = \begin{bmatrix} 6 & 1 & 2 & 3 \\ 5 & 11 & -3 & 5 \\ -3 & 4 & 3 & 7 \end{bmatrix}\,.$$

Now we want to add 2 times the line 2 to line 0, so we must calculate $E_1 A$, being

$$E_1 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\,.$$

In fact,

In [1]:
```
import numpy as np
```

In [10]:
```
A = np.array([[6,1,2,3],[5,11,-3,5],[-3,4,3,7]])
print(A)
```

```
[[ 6  1  2  3]
 [ 5 11 -3  5]
 [-3  4  3  7]]
```

In [11]:
```
E1 = np.array([[1,0,2],[0,1,0],[0,0,1]])
print(E1)
```

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

In [12]:
```
E1@A
```

Out[12]:
```
array([[ 0,  9,  8, 17],
       [ 5, 11, -3,  5],
       [-3,  4,  3,  7]])
```

Suppose we want to exchange rows 2 and 1 of the array A. To do this, simply calculate E_2A, with

$$E_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

In fact,

In [13]:
```
E2 = np.array([[1,0,0],[0,0,1],[0,1,0]])
print(E2)
```

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

In [14]:
```
E2 @ A
```

Out[14]:
```
array([[ 6,  1,  2,  3],
       [-3,  4,  3,  7],
       [ 5, 11, -3,  5]])
```

4. Python functions for elementary row operations

Next, we will define the functions that implement elementary operations on lines by means of matrix multiplication.

4.1 Adding rows

We define a function called soma*linhas that has as inputs the matrix A, indexes k, i and j and as output the matrix that results from the sum of the line j multiplied by k with the line i. If $i = i$, just set $E*{ii}=k+1$.*

In [21]:
```
def sum_rows(A,k,i,j):
    "Add k times row j to row i"
    n=A.shape[0]
    E=np.eye(n)
    if i == j:
        E[i,i] = k+1
    else:
        E[i,j] = k
    return E@A
```

Let us test the function for the reduction of a matrix to the upper triangular shape.

In [22]:
```
M=np.array([[1,2,3,8],[4,5,6,10], [7,8,4,11]])
M
```

Out[22]:
```
array([[ 1,  2,  3,  8],
       [ 4,  5,  6, 10],
       [ 7,  8,  4, 11]])
```

In [23]:
```
M1 = sum_rows(M,-4,1,0)
M2 = sum_rows(M1,-7,2,0)
M3 = sum_rows(M2,-2,2,1)
M3
```

Out[23]:
```
array([[  1.,   2.,   3.,   8.],
       [  0.,  -3.,  -6., -22.],
       [  0.,   0.,  -5.,  -1.]])
```

4.2 Row rescaling

Let's define a function called escala_linha that has as inputs the matrix A and the indices k and i and as output the matrix that results from the multiplication by k of the line i of A.

In [24]:
```
def scale_row(A,k,i):
    "Multiply row i by k"
    n=A.shape[0]
    E=np.eye(n)
    E[i,i] = k
    return E@A
```

Let's consider again the M matrix:

In [25]:
```
M
```

Out[25]:
```
array([[ 1,  2,  3,  8],
       [ 4,  5,  6, 10],
       [ 7,  8,  4, 11]])
```

We now multiply the second line by 1/4:

In [26]:
```
scale_row(M,1/4,1)
```

Out[26]:
```
array([[ 1.  ,  2.  ,  3.  ,  8.  ],
       [ 1.  ,  1.25,  1.5 ,  2.5 ],
       [ 7.  ,  8.  ,  4.  , 11.  ]])
```

4.3 Row swap

Let's define a function that has as inputs the matrix A and the indices i and j and as output the matrix resulting from the exchange of the rows i and j of A.

In [27]:
```
def swap_rows(A,i,j):
    "Swap rows i and j"
    n=A.shape[0]
    E=np.eye(n)
    E[i,i]=0
    E[j,j]=0
    E[i,j]=1
    E[j,i]=1
    return E@A
```

Let's consider again the M matrix:

In [28]:
```
M
```

Out[28]:
```
array([[ 1,  2,  3,  8],
       [ 4,  5,  6, 10],
       [ 7,  8,  4, 11]])
```

Let's swap the first line for the third line:

In [29]:
```
swap_rows(A,0,2)
```

Out[29]:
```
array([[ 2.,  7.,  8.],
       [ 8.,  7., 12.],
       [ 6., 15.,  1.]])
```

5. Solution of a linear system using Gaussian elimination

Let's use Gauss elimination to solve the system $AX = B$, with A and B defined below:

In [59]:
```
A = np.array([[6,15,1],[8,7,12],[2,7,8]])
print(A)
```

```
[[ 6 15  1]
 [ 8  7 12]
 [ 2  7  8]]
```

In [60]:
```
B = np.array([[2],[14],[10]])
print(B)
```

```
[[ 2]
 [14]
 [10]]
```

Let us form the augmented matrix M:

In [61]:
```
M = np.hstack([A,B])
print(M)
```

```
[[ 6 15  1  2]
 [ 8  7 12 14]
 [ 2  7  8 10]]
```

Let us now do the elementary row operations:

In [62]:
```
M1 = sum_rows(M,-M[1,0]/M[0,0],1,0)
print(M1)
```

```
[[  6.          15.           1.           2.        ]
 [  0.         -13.          10.66666667  11.33333333]
 [  2.           7.           8.          10.        ]]
```

In [63]:
```
M2 = sum_rows(M1,-M1[2,0]/M1[0,0],2,0)
print(M2)
```

```
[[  6.          15.           1.           2.        ]
 [  0.         -13.          10.66666667  11.33333333]
 [  0.           2.           7.66666667   9.33333333]]
```

In [64]:
```
M3 = sum_rows(M2,-M2[2,1]/M2[1,1],2,1)
print(M3)
```

```
[[  6.          15.           1.           2.        ]
 [  0.         -13.          10.66666667  11.33333333]
 [  0.           0.           9.30769231  11.07692308]]
```

Therefore, performing the backsubstitution we obtain

In [65]:
```
x2 = M3[2,3]/M3[2,2]
x2
```

Out[65]:
```
1.1900826446280992
```

In [66]:
```
x1 = 1./M3[1,1]*(M3[1,3]-M3[1,2]*x2)
x1
```

Out[66]:
```
0.10468319559228649
```

In [67]:
```
x0 = 1./M3[0,0]*(M3[0,3]-M3[0,1]*x1-M3[0,2]*x2)
x0
```

Out[67]:
```
-0.12672176308539942
```

That is, the solution is given by

In [76]:
```
x = np.transpose(np.array([[x0,x1,x2]]))
x
```

Out[76]:
```
array([[-0.12672176],
       [ 0.1046832 ],
       [ 1.19008264]])
```

Let us check the result by calculating the residue:

In [77]:
```
A@x-B
```

Out[77]:
```
array([[0.],
       [0.],
       [0.]])
```

This result can be obtained directly using numpy's command:

In [45]:
```
x = np.linalg.solve(A,B)
print(x)
```

```
[[-0.12672176]
 [ 0.1046832 ]
 [ 1.19008264]]
```

6. Exercise

Consider the linear system $AX = B$, with

$$A = \begin{bmatrix} 0 & 2 & 3 & 5 \\ 4 & -5 & 6 & -3 \\ 1 & 5 & 2 & 1 \\ 2 & -4 & 1 & 3 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 \\ -4 \\ 2 \\ 3 \end{bmatrix}.$$

(i) Find the echelon form of the augmented matrix, step by step, defining a elementary operation matrix at each step.

(ii) Solve the system using the elementary operation functions defined above and verify that the echelon matrix obtained is the same as in (i).

(iii) Perform backsubstitution to find the solution and check the result by calculating the residue.

(iv) Obtain the solution using numpy solver.

In [ ]: