gaussian

September 9, 2020

[1]:
```
# Python code: Gaussian elimination

import numpy as np



# Elementary row operations

def Rswitch(matrix,i,j):
    r = matrix[i-1].copy()
    matrix[i-1] = matrix[j-1]
    matrix[j-1] = r
    return(matrix)

def Rmult(matrix,i,c):
    matrix[i-1]=matrix[i-1]*c
    return(matrix)

def Radd(matrix,i,j,c):
    matrix[j-1]=matrix[j-1] + c*matrix[i-1]
    return(matrix)



# A simple version of Gauss

def Gauss(matrix):
    # check the number of rows
    if matrix.shape[0]<=1:
        return(matrix)
    # get the leftmost column, nonzero positions
    lcol = matrix[:,0]
    nz = np.arange(lcol.size)[lcol != 0]
    # when zero column, move to next column, if any
    if nz.size==0:
        if matrix.shape[1]<=1:
            return(matrix)
```

```
        Gauss(matrix[:,1:])
        return(matrix)
    # find first non-zero entry in column
    p=nz[0]
    if p!=0:
        Rswitch(matrix,1,p+1)
    # get zeros under the pivot
    for r in range(1,lcol.size):
        Radd(matrix,1,r+1,-matrix[r,0]/matrix[0,0])
    if matrix.shape[1]<=1:
        return(matrix)
    # if there is anything left to do after deleting first row/column, call␣
 ↪recursively
    Gauss(matrix[1:,1:])
    return(matrix)



# A bit more advanced version
# More checks, and anything in the interval (-sensititity,sensitivity) is␣
 ↪considered zero

def Gauss2(matrix,sensitivity=0.0):
    if matrix.dtype == np.int32:
        print("Must be a floating point matrix")
        return(-1)
    if matrix.ndim!=2:
        print("Wrong dimensions")
        return(-1)
    if matrix.shape[0]<=1:
        return(matrix)
    lcol = matrix[:,0]
    nz = np.arange(lcol.size)[np.abs(lcol)>sensitivity]
    if nz.size==0:
        if matrix.shape[1]<=1:
            return(matrix)
        Gauss2(matrix[:,1:],sensitivity)
        return(matrix)
    p=nz[0]
    if p!=0:
        Rswitch(matrix,1,p+1)
    for r in range(1,lcol.size):
        Radd(matrix,1,r+1,-matrix[r,0]/matrix[0,0])
    if matrix.shape[1]<=1:
        return(matrix)
    Gauss2(matrix[1:,1:],sensitivity)
    return(matrix)
```

```
# Some tests that you can run

A = np.array([[1,1,1,3,-1],[1,2,4,7,3],[2,3,5,10,2]])
B = np.random.randn(10,5)
```

[2]:
```
# matrix of integers without pivots in all columns
print(A)
print(Gauss(A))
```

```
[[ 1  1  1  3 -1]
 [ 1  2  4  7  3]
 [ 2  3  5 10  2]]
[[ 1  1  1  3 -1]
 [ 0  1  3  4  4]
 [ 0  0  0  0  0]]
```

[3]:
```
# print with 4 decimals, suppress scientific notation ("E notation")
np.set_printoptions(precision=4,suppress=True)
print(B)
print(Gauss(B))
print(Gauss2(B,0.0001))
```

```
[[ 1.9425  1.943  -0.1413  0.2211  0.9861]
 [ 0.9768  2.3272 -0.8968  0.2793 -0.6048]
 [-0.9959  0.5387  0.0484 -1.7085 -0.4591]
 [-0.7925  1.3756 -2.5251  0.5295 -0.603 ]
 [-0.7764 -0.9578  1.5136  1.8578  0.1102]
 [-0.2445 -2.6114 -1.9827  0.1552  1.0587]
 [ 0.5137  0.5413  0.9217 -2.299  -1.4295]
 [-1.3818  1.061   1.6216  0.0361  0.5546]
 [ 0.0675  0.4346  0.1718  1.8314 -0.2577]
 [ 0.2839  0.8513 -0.4633 -1.205   0.3668]]
[[ 1.9425  1.943  -0.1413  0.2211  0.9861]
 [-0.      1.3501 -0.8257  0.1681 -1.1007]
 [-0.     -0.      0.9147 -1.7863  1.2979]
 [ 0.      0.      0.     -2.1043  3.35  ]
 [ 0.      0.      0.     -0.      5.7662]
 [ 0.      0.      0.      0.      0.    ]
 [ 0.     -0.      0.      0.      0.    ]
 [ 0.      0.      0.      0.      0.    ]
 [ 0.      0.      0.      0.     -0.    ]
 [ 0.      0.      0.      0.      0.    ]]
[[ 1.9425  1.943  -0.1413  0.2211  0.9861]
 [ 0.      1.3501 -0.8257  0.1681 -1.1007]
 [ 0.     -0.      0.9147 -1.7863  1.2979]
```

```
[ 0.      0.      0.     -2.1043  3.35  ]
[ 0.      0.      0.      0.      5.7662]
[ 0.      0.      0.      0.      0.    ]
[ 0.      0.     -0.      0.      0.    ]
[ 0.      0.      0.      0.      0.    ]
[ 0.      0.      0.      0.     -0.    ]
[ 0.      0.      0.      0.      0.    ]]
```

[ ]: