IdentifyingSpecialMatrices

June 12, 2019

1 Identifying special matrices

1.1 Instructions

In this assignment, you shall write a function that will test if a 4Œ4 matrix is singular, i.e. to determine if an inverse exists, before calculating it.

You shall use the method of converting a matrix to echelon form, and testing if this fails by leaving zeros that can't be removed on the leading diagonal.

Don't worry if you've not coded before, a framework for the function has already been written. Look through the code, and you'll be instructed where to make changes. We'll do the first two rows, and you can use this as a guide to do the last two.

1.1.1 Matrices in Python

In the *numpy* package in Python, matrices are indexed using zero for the top-most column and left-most row. I.e., the matrix structure looks like this:

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

You can access the value of each element individually using,

```
A[n, m]
```

which will give the n'th row and m'th column (starting with zero). You can also access a whole row at a time using,

```
A[n]
```

Which you will see will be useful when calculating linear combinations of rows.

A final note - Python is sensitive to indentation. All the code you should complete will be at the same level of indentation as the instruction comment.

1.1.2 How to submit

Edit the code in the cell below to complete the assignment. Once you are finished and happy with it, press the *Submit Assignment* button at the top of this notebook.

Please don't change any of the function names, as these will be checked by the grading script.

If you have further questions about submissions or programming assignments, here is a list of Q&A. You can also raise an issue on the discussion forum. Good luck!

In [2]:
```
# GRADED FUNCTION
import numpy as np

# Our function will go through the matrix replacing each row in order turning it into ec
# If at any point it fails because it can't put a 1 in the leading diagonal,
# we will return the value True, otherwise, we will return False.
# There is no need to edit this function.
def isSingular(A) :
    B = np.array(A, dtype=np.float_) # Make B as a copy of A, since we're going to alter
    try:
        fixRowZero(B)
        fixRowOne(B)
        fixRowTwo(B)
        fixRowThree(B)
    except MatrixIsSingular:
        return True
    return False

# This next line defines our error flag. For when things go wrong if the matrix is singu
# There is no need to edit this line.
class MatrixIsSingular(Exception): pass

# For Row Zero, all we require is the first element is equal to 1.
# We'll divide the row by the value of A[0, 0].
# This will get us in trouble though if A[0, 0] equals 0, so first we'll test for that,
# and if this is true, we'll add one of the lower rows to the first one before the divis
# We'll repeat the test going down each lower row until we can do the division.
# There is no need to edit this function.
def fixRowZero(A) :
    if A[0,0] == 0 :
        A[0] = A[0] + A[1]
    if A[0,0] == 0 :
        A[0] = A[0] + A[2]
    if A[0,0] == 0 :
        A[0] = A[0] + A[3]
    if A[0,0] == 0 :
        raise MatrixIsSingular()
    A[0] = A[0] / A[0,0]
    return A
```

```
# First we'll set the sub-diagonal elements to zero, i.e. A[1,0].
# Next we want the diagonal element to be equal to one.
# We'll divide the row by the value of A[1, 1].
# Again, we need to test if this is zero.
# If so, we'll add a lower row and repeat setting the sub-diagonal elements to zero.
# There is no need to edit this function.
def fixRowOne(A) :
    A[1] = A[1] - A[1,0] * A[0]
    if A[1,1] == 0 :
        A[1] = A[1] + A[2]
        A[1] = A[1] - A[1,0] * A[0]
    if A[1,1] == 0 :
        A[1] = A[1] + A[3]
        A[1] = A[1] - A[1,0] * A[0]
    if A[1,1] == 0 :
        raise MatrixIsSingular()
    A[1] = A[1] / A[1,1]
    return A

# This is the first function that you should complete.
# Follow the instructions inside the function at each comment.
def fixRowTwo(A) :
    # Insert code below to set the sub-diagonal elements of row two to zero (there are t
    A[2] = A[2] - A[2,0] * A[0]
    A[2] = A[2] - A[2,1] * A[1]

    # Next we'll test that the diagonal element is not zero.
    if A[2,2] == 0 :
        # Insert code below that adds a lower row to row 2.
        A[2] = A[2] + A[3]
        A[2] = A[2] - A[2,0] * A[0]
        A[2] = A[2] - A[2,1] * A[1]

        # Now repeat your code which sets the sub-diagonal elements to zero.

    if A[2,2] == 0 :
        raise MatrixIsSingular()
    # Finally set the diagonal element to one by dividing the whole row by that element.
    A[2] = A[2] / A[2,2]

    return A

# You should also complete this function
# Follow the instructions inside the function at each comment.
def fixRowThree(A) :
    # Insert code below to set the sub-diagonal elements of row three to zero.
    A[3] = A[3] - A[3,0] * A[0]
    A[3] = A[3] - A[3,1] * A[1]
```

```
            A[3] = A[3] - A[3,2] * A[2]

            # Complete the if statement to test if the diagonal element is zero.
            if A[3][3] == 0:
                raise MatrixIsSingular()
            # Transform the row to set the diagonal element to one.
            A[3] = A[3] / A[3,3]

            return A
```

1.2 Test your code before submission

To test the code you've written above, run the cell (select the cell above, then press the play button [|] or press shift-enter). You can then use the code below to test out your function. You don't need to submit this cell; you can edit and run it as much as you like.

Try out your code on tricky test cases!

```
In [3]: A = np.array([
                [2, 0, 0, 0],
                [0, 3, 0, 0],
                [0, 0, 4, 4],
                [0, 0, 5, 5]
            ], dtype=np.float_)
        isSingular(A)

Out[3]: True

In [5]: A = np.array([
                [0, 7, -5, 3],
                [2, 8, 0, 4],
                [3, 12, 0, 5],
                [1, 3, 1, 3]
            ], dtype=np.float_)
        fixRowZero(A)

Out[5]: array([[ 1. ,  7.5, -2.5,  3.5],
               [ 2. ,  8. ,  0. ,  4. ],
               [ 3. , 12. ,  0. ,  5. ],
               [ 1. ,  3. ,  1. ,  3. ]])

In [6]: fixRowOne(A)

Out[6]: array([[ 1.        ,  7.5       , -2.5       ,  3.5       ],
               [-0.        ,  1.        , -0.71428571,  0.42857143],
               [ 3.        , 12.        ,  0.        ,  5.        ],
               [ 1.        ,  3.        ,  1.        ,  3.        ]])

In [7]: fixRowTwo(A)
```

```
Out[7]: array([[ 1.        ,  7.5       , -2.5       ,  3.5       ],
               [-0.        ,  1.        , -0.71428571,  0.42857143],
               [ 0.        ,  0.        ,  1.        ,  1.5       ],
               [ 1.        ,  3.        ,  1.        ,  3.        ]])

In [8]: fixRowThree(A)

Out[8]: array([[ 1.        ,  7.5       , -2.5       ,  3.5       ],
               [-0.        ,  1.        , -0.71428571,  0.42857143],
               [ 0.        ,  0.        ,  1.        ,  1.5       ],
               [ 0.        ,  0.        ,  0.        ,  1.        ]])

In [ ]:
```