01-HOMEWORK-BasicLinearAlgebra

August 26, 2021

1 Homework 1

In this notebook, you will use the `numpy` library to do some basic tasks in linear algebra.

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

1.1 Matrix Algebra

Some `numpy` arrays (i.e., matrices) are defined in the next cell.

[32]:
```
A = np.array([[0,1,2],
              [3,4,5]])

B = np.array([[-1,-1,-1,-1],
              [0,1,2,3],
              [-3,-2,-1,0]])
```

To show the matrices in a somewhat readable way, we can use the `print` function.

[33]:
```
print(A)
print(B)
```

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

To determine the shape of a matrix, use the `shape` attribute as follows.

[34]:
```
A.shape
```

[34]:
```
(2, 3)
```

Note: The shapes of these particular matrices are obvious, by inspection, but the shape attribute of a `numpy` array is something that you will use frequently when coding!

Problem 1 Figure out how to multiply the `numpy` array `A` by the number 3. Call your answer `C`. Figure out how to use `numpy` to multiply the matrices `A` and `B`. Call the result `D`. Print out your answers `C` and `D`.

General Remark: When doing coding exercies, Google is your friend! Feel free to use any resources you like when completing these assignments.

[35]:
```
C = 3*A
D = A@B
print(C,'\n',D)
```

```
[[ 0  3  6]
 [ 9 12 15]]
 [[ -6  -3   0   3]
 [-18  -9   0   9]]
```

There is a matrix operation called *transpose* which we haven't yet defined in class (although you may have seen it before). It is done in `numpy` by appending `.T` to a `numpy` array. For example:

[36]:
```
A.T
```

[36]:
```
array([[0, 3],
       [1, 4],
       [2, 5]])
```

Problem 2 Define a new `numpy` array called `X`. This array can be anything you want. Print out your array `X` and then print its transpose.

[10]:
```
X = np.array([[1,0,1],[0,1,0],[-1,0,-1]])
print(X)
print(X.T)
```

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

Change the cell below to a 'Markdown' cell and type a description (in your own words) of what the transpose operation does to a matrix, in general.

The transpose operation takes each row and column in a matrix and swaps them. In other words, an n x m matrix becomes an m x n matrix.

1.2 Solving Systems of Linear Equations

There is a simple `numpy` function for solving *certain types* of systems of linear equations. For example:

[37]:
```
# Define a coefficient array
A = np.array([[0,1,-2],
              [-1,1,3],
              [5,4,3]])
```

```
# Define the right hand side of the system of equations
b = np.array([-1,-2,-3])

# Solve the equation Ax = b for x
x = np.linalg.solve(A, b)
print('The solution is', x)
```

```
The solution is [ 0.44444444 -1.22222222 -0.11111111]
```

Problem 3 Check that the solution above is correct. (I.e., verify that it satisfies the system of equations. This should only take about a line of code.)

[49]:
```
# Function from the example given by numpy https://numpy.org/doc/stable/
 ↪reference/generated/numpy.linalg.solve.html
print(np.allclose(np.dot(A, x),b))
```

```
True
```

Problem 4 Make up a new system of linear equations and find its solution. Make sure to print the solution.

Note: Depending on the system you pick, this function may give you an error! This has to do with the fact that this function can only solve *certain types* of systems. Think about what special systems this function might be designed to solve. Keep trying to come up with a new system until you are successful.

[51]:
```
F = np.array([[1,-1,1,-1],
              [1,1,-1,-1],
              [1,1,1,1],
              [1,-1,-1,1]])
f = np.array([2,2,2,-2])
x_f = np.linalg.solve(F,f)
print('The solution to Problem 4\'s system is', x_f)
```

```
The solution to Problem 4's system is [ 1.  1.  1. -1.]
```

1.2.1 Underdetermined Systems

Hopefully you spent some time thinking about which types of systems the function from Problem 3 is able to solve, because I'm about to give it away…

The `np.linalg.solve` function is only designed to solve systems of linear equations for which the number of equations is equal to the number of variables and for which there is exactly one solution. To get a solution to an *underdetermined* system (more variables than equations), we will have to be a bit trickier.

Consider the following system.

[52]:
```
A = np.array([[1,1,2,3],
              [0,0,2,4]])
```

```
b = np.array([2,3])
```

The `np.linalg.solve` function will give us an error if we try to solve this:

[53]:
```
np.linalg.solve(A,b)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-53-36718354ad3a> in <module>
----> 1 np.linalg.solve(A,b)

<__array_function__ internals> in solve(*args, **kwargs)

F:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in solve(a, b
    378     a, _ = _makearray(a)
    379     _assert_stacked_2d(a)
--> 380     _assert_stacked_square(a)
    381     b, wrap = _makearray(b)
    382     t, result_t = _commonType(a, b)

F:\ProgramData\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in␣
 ↪_assert_stacked_square(*arrays)
    201         m, n = a.shape[-2:]
    202         if m != n:
--> 203             raise LinAlgError('Last 2 dimensions of the array must be␣
 ↪square')
    204
    205 def _assert_finite(*arrays):

LinAlgError: Last 2 dimensions of the array must be square
```

We expect infinitely many solutions to such a system, so what would a 'solution' even look like, computationally? First, we can try to find a *particular solution*. We can use the 'least squares' function `np.linalg.lstsq` to find a particular solution which is optimal in some sense (if you've seen the concept of a norm before, it is a solution with the smallest possible norm — not so important to know this detail for now).

[54]:
```
results = np.linalg.lstsq(A, b)
```

```
<ipython-input-54-a9279e6a6360>:1: FutureWarning: `rcond` parameter will change
to the default of machine precision times ``max(M, N)`` where M and N are the
input matrix dimensions.
To use the future default and silence this warning we advise to pass
`rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  results = np.linalg.lstsq(A, b)
```

Note: You may get a deprecation warning when running the above. You can ignore it!

The `results` variable contains a few things, and we can get the solution by picking out the first

thing using the following code.

[55]:
```
x=results[0]
print('A solution is',x)
```

```
A solution is [-0.18181818 -0.18181818  0.22727273  0.63636364]
```

We can check that this is really a solution (note that matrix multiplication is being used below, so this shows a possible answer to part of Problem 1…):

[56]:
```
print(A@x)
print(b)
```

```
[2. 3.]
[2 3]
```

Recall from class that we can describe the general equation by solving the homogeneous equation $A\vec{x} = \vec{0}$. This can be done by using another function called `null_space` from the package `scipy`.

[57]:
```
from scipy.linalg import null_space
```

[58]:
```
N = null_space(A)
print(N)
```

```
[[-0.47304472 -0.56717126]
 [ 0.08068822  0.73412802]
 [ 0.784713   -0.33391353]
 [-0.3923565   0.16695676]]
```

Each column of N gives a solution to the homogeneous equation. The next code block creates a variable for each of these columns.

[59]:
```
u = N[:,0]
v = N[:,1]

print(u)
print(v)
```

```
[-0.47304472  0.08068822  0.784713   -0.3923565 ]
[-0.56717126  0.73412802 -0.33391353  0.16695676]
```

In class we showed that any solution of our original system of equations $A\vec{x} = \vec{b}$ can be written as $\vec{x} + x_2\vec{u} + x_4\vec{v}$, where $\vec{x}$ is our particular solution, $\vec{u}, \vec{v}$ are solutions to the homogeneous equation and x_2 and x_4 are arbitrary real numbers.

Problem 5 Create three different vectors of this form and show that each of them solves the system of equations.

[61]:
```
y1 = x + u + v
y2 = x + 2*u - 3*v
y3 = x - u - v
print(y1, y2, y3)
```

```
[-1.22203416  0.63299806  0.6780722   0.4109639 ] [ 0.57360617 -2.22282582
2.79843929 -0.64921965] [ 0.8583978  -0.99663443 -0.22352674  0.86176337]
```

[63]:
```
print(A@y1)
print(A@y2)
print(A@y3)
print(b)
```

```
[2. 3.]
[2. 3.]
[2. 3.]
[2 3]
```