Math 246 Unit 8: Numpy matrices and linear algebra

Brenton LeMesurier, October 29, 2015; slightly revised on November 3.

Introduction

The module numpy (and thus also module pylab) has a variable type matrix which is a specialization of the type array that makes some linear algebra operation easier. Matrices are always two dimensional, so to use type matrix for a vector \( v \) and be able to do matrix-vector multiplication with \( A \times v \), the vector needs to be created as a matrix with a single column.

This module shows how to create and use numpy matrices, with an exercise on row reduction.

Aside: Floating point numbers vs integers

Though Python is generally good at understanding when an integer like 7 is to be used as a floating point (real) number, it is sometimes best to make this distinction explicitly when working with module numpy; otherwise sometimes division done within numpy functions returns an integer answer, like \( 7/2 = 3 \).

Thus from now on, when I mean an integer to be used as a floating point number, I give it a decimal point: \( 7./2. \) will reliably be 3.5

First the usual boilerplate, including import of all the numpy functions to be used below.

In [ ]:

```python
'''
Unit 8: Numpy matrices
Author: Brenton LeMesurier, <lemesurierb@cofc.edu>
Date: October 29, 2015
'''

from numpy import array, matrix, zeros

Creating numpy matrices

The methods that work for creating array also work, with an obvious change:
In [ ]:

```python
# Create an array from a list:
Aarray = array([[1., 2.], [3., 4.]])
print(Aarray)
```

In [ ]:

```python
# Create a matrix from a 2D list:
A = matrix([[1., 2.], [3., 4.]])
print(A)
```

One can convert back and forth

In [ ]:

```python
Barray = array([[0., 1.], [1., 0.]])
# Create a matrix from a 2D array:
B = matrix(Barray)
print(B)
```

What's the difference? For one thing, multiplication works differently:

Array multiplication multiplies the corresponding elements:

In [ ]:

```python
print(Aarray * Barray)
```

Matrix multiplication instead does the expected mathematical thing:

In [ ]:

```python
print(A * B)
```

Arrays can be multiplied "mathematically" by using the `.dot()` method, which extends the idea of the dot product of two vectors: it computes the array of all the dot products of each combination of a row of the first array with a column of the second:

In [ ]:

```python
print(Aarray.dot(Barray))
```
More options for matrix creation

Matrices can also be created directly without using lists or arrays, using a string with semicolons separating the rows. (Matlab users should recognize this notation!)

In [ ]:

```python
C = matrix("3., 4., 5.; 6., 7., 8.")
print(C)
```

The commas within a row are optional – but I recommend that you be consistent, not like in this example:

In [ ]:

```python
D = matrix("6. 5. 4.; 1., 5. -2.")
print(D)
```

Row and column matrices are different, and best created with this new notation:

In [ ]:

```python
brow = matrix('2. 4. 10.')
print(brow)
```

In [ ]:

```python
bcol = matrix('2. ; 4. ; 10.')
print(bcol)
```

The difference is important for matrix-vector multiplication:

In [ ]:

```python
# 2x3 matrix times 3x1 (column) matrix gives a 2x1 column matrix:
print(C * bcol)
```

In [ ]:

```python
# 2x3 matrix time 1x3 (row) matrix gives an error!:
print(C * brow)
```
Other matrix operations: transposes, inverses, etc.

In [ ]:

```python
print("C is:")
print(C)
print("Its transpose C.T is:")
print(C.T)
```

In [ ]:

```python
print("A is:")
print(A)
Ainverse = A.I
print("Its inverse is:")
print(Ainverse)
print("Check by multiplying, both ways:")
print(A * Ainverse)
print(Ainverse * A)
```

Slicing: Extracting rows, columns, and other rectangular chunks from matrices

This works with lists, arrays and matrices, and we have seen some of it before; I review it here because it will help with doing row operations on matrices.

Index notation for slicing

For an index with n possible values, from 0 to n-1:

- `a:b` means indices $i$, $a \leq i < b$
- `a:` is short for `a:n`, so indices $a \leq i$, all the way to the maximum index value
- `:` is short for `0:b`, so all indices $i < b$
- `:` combines both of the above, so is short for `0:n`, all possible indices
- index value $-1$ refers the last entry; the same as index $n-1$
- index value $-m$ refers the "m-th last" entry; the same as index $n-m"
Exercise 1

What range of indices do your get with : - 1 ? Be careful!

In [ ]:

Ab = C.copy()  
print("The 'augmented matrix' Ab is:")
print(Ab)

In [ ]:

row1 = Ab[0,:]
print("Its first row (index 0!) is:")
print(row1)

In [ ]:

column2 = Ab[:,1]
print("Its second column (index 1!) is:")
print(column2)

In [ ]:

A = Ab[:,2]
print("Its left-hand 2x2 chunk is the square matrix:")
print(A)

In [ ]:

b = Ab[:, -1]  
print("Its right-hand column is the column matrix:")
print(b)
In [ ]:

```python
print("We can also combine matrices (and arrays) into a larger one:")
A_and_b = zeros((2,3))  # First, create an empty shell of a matrix
'Note well: we always have to specify ranges of index values in the matrix where we are inserting stuff, even when a range contains just a single index value:
print(A_and_b)
A_and_b[0:2,0:2] = A
print(A_and_b)
A_and_b[0:2,2:] = b
print(A_and_b)
```

**Row operations and row reduction**

We can extract a row of a matrix, and so we can use this to do row operations. The above augmented matrix $A_b$ can be row reduced to "row echelon form". Only one operation is needed:

In [ ]:

```python
Ab = C.copy()  # Reset $A_b$ in case it has been changed!
print("Ab is initially")
print(Ab)
m = Ab[1,0]/Ab[0,0]
print("The row multiplier is \$m = \$", m)
Ab[1,:] = Ab[1,:] - m * Ab[0,:]
print("Ab is now")
print(Ab)
```
Exercise 2

This can be done either by modifying this notebook, or copying the relevant code from here into a Python file Unit7.py and adding stuff there. I describe the "notebook" option.

Add code cells below this one, and then in those cells:

1. Create a 3x3 matrix A (You could explore the function rand from module numpy.random or function hilbert from module scipy.linalg)
2. Create a 3 element column matrix b
3. Combine these into the augmented matrix Ab
4. Do the row operations (three in all) needed to put the augmented matrix into row-echelon form. As above, print each multiplier, and print each updated version of the matrix.
5. Compute (and display) the inverse of A, and use this to solve $A x = b$ for x.
6. Read about the numpy function solve, and use this to solve for x the easy way.

It might be best to do each of steps 1 to 4 in successive cells, and with each of the three stages of step 4 in its own cell.