Matrix Operations in Python

The goal of this Python Notebook is to explore basic array and matrix operations as provided by the NumPy package in Python3. The first statement imports all numpy (and matplotlib) functions so that they can be used without the numpy (or np) prefix.

```
In [1]: from pylab import *
```

Note that in some cases this can lead to conflicts if the same name is used for more than one function. Here is an example where we want to produce powers of 2.0 with exponents from 0 to 8:

```
In [103]: p2 = power(2.0, arange(9))
```

```
In [104]: print(p2)

          []
```

This was interpreted as a call to the power function distribution which generates random samples with a specific probability density function. To generate an array with the powers of 2 the np (for numpy) prefix needs to be used as shown below.

```
In [107]: p2 = np.power(2.0, arange(9))
```

```
In [108]: print(p2)

          [   1.    2.    4.    8.   16.   32.   64.  128.  256.]
```

Now lets start by generating two vectors, one with 3 and one with 4 components.

```
In [78]: x = [1.,2.,3.]
         y = [4.,5.,6.,7.]
```

Two vectors with different lengths cannot be added or multiplied, but we should be able to scale them. Let's try this:

```
In [112]: print(4*x)

          [1.0, 2.0, 3.0, 1.0, 2.0, 3.0, 1.0, 2.0, 3.0, 1.0, 2.0, 3.0]
```

This is not quite the result we expected. The reason is that Python interprets x as a list and the multiplication means replication in this context, thus list x is replicated 4 times. To declare a vector we should thus use the array command from numpy to define vectors x and y:

```
In [113]: x = array([1.,2.,3.])
          y = array([4.,5.,6.,7.])
```

```
In [114]: print(4*x)

          [  4.   8.  12.]
```

What happens if we add 1.5?

```
In [115]: print(4*x + 1.5)
```

```
[  5.5   9.5  13.5]
```

Thus, the addition of a scalar is broadcast over a elements of the array. We can of course also add two vectors together if they have the same lengths:

```
In [116]: print(x+y[1:])
```

```
[  6.   8.  10.]
```

Here we used all three components of x and the last three components of y. Note that indexing starts at 0 and thus the first component in y is y[0]. Next, let's create a 3x4 matrix A1 from the outer product of x (as a column vector) and y (as a row vector). In NumPy we do not need to specify whether a vector is a row or column vector, this is implicit in the specification of the 'outer' function and the order in which x and y are entered.

```
In [117]: A1 = outer(x,y)
```

```
In [118]: print(A1)
```

```
[[  4.   5.   6.   7.]
 [  8.  10.  12.  14.]
 [ 12.  15.  18.  21.]]
```

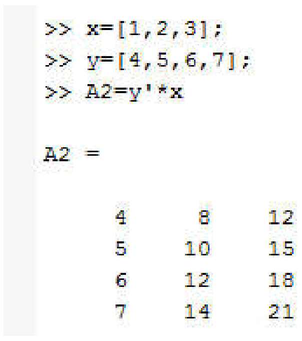
If we reverse the order of x and y we obtain a 4x3 matrix A2:

```
In [119]: A2 = outer(y,x)
```

```
In [120]: print(A2)
```

```
[[  4.   8.  12.]
 [  5.  10.  15.]
 [  6.  12.  18.]
 [  7.  14.  21.]]
```

Here is what the computation of A2 would look like in Matlab: included using caption



```
>> x=[1,2,3];
>> y=[4,5,6,7];
>> A2=y'*x

A2 =

     4     8    12
     5    10    15
     6    12    18
     7    14    21
```

Now let's try matrix multiplication:

```
In [30]: B12 = A1*A2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-30-8c305f905317> in <module>()
----> 1 B12 = A1*A2

ValueError: operands could not be broadcast together with shapes (3,4) (4,3)
```

```
In [29]: B12 = A1*A2.T
```

```
In [31]: print(B12)
```

```
[[  16.   25.   36.   49.]
 [  64.  100.  144.  196.]
 [ 144.  225.  324.  441.]]
```

```
In [41]: C12 = dot(A1,A2)
```

```
In [42]: print(C12)
```

```
[[  126.   252.   378.]
 [  252.   504.   756.]
 [  378.   756.  1134.]]
```

```
In [43]: C21 = dot(A2,A1)
```

```
In [44]: print(C21)
```

```
[[ 224.  280.  336.  392.]
 [ 280.  350.  420.  490.]
 [ 336.  420.  504.  588.]
 [ 392.  490.  588.  686.]]
```

```
In [60]: C12m = matrix(C12)
```

```
In [61]: print(C12m)
```

```
[[  126.   252.   378.]
 [  252.   504.   756.]
 [  378.   756.  1134.]]
```

```
In [62]: print(det(C12m))
```

```
0.0
```

```
In [63]: print(matrix_rank(C12m))
```

```
1
```

```
In [66]: C12mr = C12m + np.random.randint(-50,50,9).reshape((3,3))
```

```
In [67]: print(C12mr)
```

```
[[  101.   254.   373.]
 [  233.   492.   802.]
 [  391.   787.  1107.]]
```

```
In [68]: print(det(C12mr))

         2038451.0

In [69]: C12mri = C12mr.I

In [70]: print(C12mri)

         [[-0.0424489   0.0060698   0.00990556]
          [ 0.02730063 -0.01669699  0.00289779]
          [-0.00441561  0.0097265  -0.0046555 ]]

In [73]: CC = dot(C12mr,C12mri)

In [74]: print(CC)

         [[  1.00000000e+00   0.00000000e+00   0.00000000e+00]
          [ -1.33226763e-15   1.00000000e+00   0.00000000e+00]
          [ -8.88178420e-16   0.00000000e+00   1.00000000e+00]]

In [91]: print('x =', x, ', y =', y)

         x = [1.0, 2.0, 3.0] , y = [4.0, 5.0, 6.0, 7.0]

In [94]: print(A1)

         [[  4.   5.   6.   7.]
          [  8.  10.  12.  14.]
          [ 12.  15.  18.  21.]]

In [95]: print(A2)

         [[  4.   8.  12.]
          [  5.  10.  15.]
          [  6.  12.  18.]
          [  7.  14.  21.]]

In [97]: u1 = dot(x,A1)

In [98]: u2 = dot(A1,y)

In [99]: print('u1 =', u1, ', u2 =', u2)

         u1 = [ 56.  70.  84.  98.] , u2 = [ 126.  252.  378.]

In [100]: v1 = dot(A2,x)

In [101]: v2 = dot(y,A2)

In [102]: print('v1 =', v1, ', v2 =', v2)

         v1 = [ 56.  70.  84.  98.] , v2 = [ 126.  252.  378.]

In [ ]:
```