Python – Matrix Calculation

The modules of interest are `numpy` and `numpy.linalg`.

```
import numpy as np
import numpy.linalg as alg
```

1. Matrix creation. To define a matrix, we use the `array` function of the module `numpy`.

```
>>> A = np.array([[1, 2, 3], [4, 5, 6]])
>>> A
array([[1, 2, 3],
       [4, 5, 6]])
```

The `shape` attribute gives the size of the matrix, that is, the numbers of lines and columns. It is possible to resize a matrix, without modifying its entries, with the `reshape` function.

```
>>> A.shape
(2, 3)
>>> A = A.reshape((3, 2))
>>> A
array([[1, 2],
       [3, 4],
       [5, 6]])
```

To access the ijth entry of matrix A, use the operation `A[i,j]` where i denotes the line number and j denoted the column number. Recall that indices start at the value 0. One can also obtain any part of the matrix: a line, a column or submatrix. Also recall that $a : b$ denotes all indices from a to $b - 1$.

```
>>> A[1, 0]   # element of the 2nd line, 1st column
3
>>> A[0, :]      # 1st line as an 1D array
array([1, 2])
>>> A[0, :].shape
(2,)
>>> A[0:1, :]    # 1st line as a line matrix
array([[1, 2]])
>>> A[0:1, :].shape
(1, 2)
>>> A[:, 1]        # 2nd column as 1D array
    array([2, 4, 6])
>>> A[:, 1:2]    # 2nd column as a column matrix
array([[2],
       [4],
       [6]])
>>> A[1:3, 0:2]  # sub-matrix made of lines 2 and 3 and columns 1 and 2
array([[3, 4],
       [5, 6]])
```

It is possible to create special types of matrices. The function `zeros` and `ones` create matrices filled with 0 or 1. The function `eye` creates an identity matrix. The function `diag` creates a diagonal matrix.

```
>>> np.zeros((2,3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.ones((3,2))
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])
>>> np.eye(4)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
>>> np.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

Finally the function `concatenate` allows the user to create matrices by blocks by superposition (`axis=0`) or by side-by-side placement (`axis=1`) of several matrices.

```
>>> A = np.ones((2,3))
>>> B = np.zeros((2,3))
>>> np.concatenate((A,B), axis=0)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.concatenate((A,B), axis=1)
array([[ 1.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.,  0.]])
```

2. A few useful functions.

To copy an array, we can use the `numpy` function `copy`.

```
>>> A = np.array([[1,-2,3], [-4,5,-6]])
>>> B = A.copy()
>>> B[1, 0] = 8
>>> A
array([[ 1, -2,  3],
       [-4,  5, -6]])
>>> B
array([[ 1, -2,  3],
       [ 8,  5, -6]])
```

The functions `amax,amin` and `mean` of the `numpy` library allow the determination of the maximum, minimum and average value of the array's elements.

```
>>> np.amax(A)
5
>>> np.amin(A)
-6
>>> np.mean(A)
-0.5
```

Finally, the command `array_equal` allows the testing of the term-by-term equality of two arrays of identical size.

```
>>> np.array_equal(A, B)
False
```

3. Matrix Calculus

The operations of addition and multiplication by a scalar are done by the operators `+` and `*`.

```
>>> A = np.array([[1,2], [3,4]])
>>> B = np.eye(2)
>>> A + 3*B
array([[ 4.,  2.],
       [ 3.,  7.]])
```

To multiply two matrices (whenever feasible), one must use the function `dot`.

```
>>> A = np.array([[1,2], [3,4]])
>>> B = np.array([[1,1,1], [2,2,2]])
>>> np.dot(A, B)
array([[ 5,  5,  5],
       [11, 11, 11]])
```

It is also possible to use the dot method for the multiplication of several matrices. The function `matrix_power` of the module `numpy.linalg` allows the calculation of powers of matrices.

```
>>> A.dot(B)
array([[ 5,  5,  5],
       [11, 11, 11]])
>>> A.dot(B).dot(np.ones((3,2)))
array([[ 15.,  15.],
       [ 33.,  33.]])
>>> alg.matrix_power(A,3)
array([[ 37,  54],
       [ 81, 118]])
```

The transpose of a matrix is done by the function `transpose`. The expression `A.T` returns the transpose of matrix `A`.

```
>>> np.transpose(B)
array([[1, 2],
       [1, 2],
       [1, 2]])
>>> B.T
array([[1, 2],
       [1, 2],
       [1, 2]])
```

The determinant, rank and trace of a matrix can be obtained by the functions `det`, `matrix_rank` of the module `numpy.linalg` and the function `trace` of module `numpy`. Finally, the function `inv` of the module `numpy.linalg` returns the inverse of a matrix (whenever it exists)

```
>>> alg.det(A)
-2.0000000000000004
>>> alg.matrix_rank(A)
2
>>> np.trace(A)
5
>>> alg.inv(A)
matrix([[-2. ,  1. ],
        [ 1.5, -0.5]])
```

To solve the linear system `Ax = b` (when matrix `A` is invertible), one can use the function `solve` of the module `numpy.linalg`.

```
>>> b = np.array([1,5])
>>> alg.solve(A, b)
array([ 3., -1.])
```

4. Scalar and vector products.

The function `vdot` allows the calculation of the scalar product of two n-dimensional vectors.

```
>>> u = np.array([1,2])
>>> v = np.array([3,4])
>>> np.vdot(u, v)
11
```

The function `cross` allows the calculation of the cross product of two n-dimensional vectors.

```
>>> u = np.array([1,0,0])
>>> v = np.array([0,1,0])
>>> np.cross(u, v)
array([0, 0, 1])
```