Resources

- M. Scott Shell, `http://www.engr.ucsb.edu/%7eshell/che210d/numpy.pdf`
- `http://wiki.scipy.org/NumPy_for_Matlab_Users`
- `http://mathesaurus.sourceforge.net/matlab-numpy.html`

Numpy arrays

- `import numpy as np`
- Numpy provides class **`ndarray`**, called “array”
- Create array from a list
```
>>> x = np.array([3.0,5,7,5])
>>> x
array([ 3.,  5.,  7.,  5.])
```
- If appear to be integers in list, need “float”
- 2D arrays
```
>>> A = np.array([[8.,1.,6.],[3.,5.,7.],[4.,9.,2.]])
array([[ 8.,  1.,  6.],
       [ 3.,  5.,  7.],
       [ 4.,  9.,  2.]])
```

Subscripts

- Use brackets to denote subscripts
- Start counting at 0

```
>>> x[0]
3.0
>>> A[1,2]
7.0
```

- Colons work, be careful of last value!

```
>>> x[0:1]
array([ 3.0])
```

```
>>> x
array([ 3.,  5.,  7.,  5.])
>>> A
array([[ 8.,  1.,  6.],
       [ 3.,  5.,  7.],
       [ 4.,  9.,  2.]])
```

Subscripts

- Use brackets to denote subscripts
- Start counting at 0

```
>>> x[0]
3.0
>>> A[1,2]
7.0
```

- Colons work, be careful of last value!

```
>>> x[0:1]
array([ 3.0])
>>> x[0:2]
array([ 3.0,  5.])
```

```
>>> x
array([ 3.,  5.,  7.,  5.])
>>> A
array([[ 8.,  1.,  6.],
       [ 3.,  5.,  7.],
       [ 4.,  9.,  2.]])
```

Subscripts

- Use brackets to denote subscripts
- Start counting at 0

```
>>> x[0]
3.0
>>> A[1,2]
7.0
```

- Colons work, be careful of last value!

```
>>> x[0:1]
array([ 3.0])
>>> x[0:2]
array([ 3.0,  5.])
>>> A[:,2]
array([ 6.,  7.,  2.])
```

```
>>> x
array([ 3.,  5.,  7.,  5.])
>>> A
array([[ 8.,  1.,  6.],
       [ 3.,  5.,  7.],
       [ 4.,  9.,  2.]])
```

Subscripts

- Use brackets to denote subscripts
- Start counting at 0

```
>>> x[0]
3.0
>>> A[1,2]
7.0
```

- Colons work, be careful of last value!

```
>>> x[0:1]
array([ 3.0])
>>> x[0:2]
array([ 3.0,  5.])
>>> A[:,2]
array([ 6.,  7.,  2.])
```

- Negative indices count from end

```
>>> x[-1]
5.0
```

```
>>> x
array([ 3.,  5.,  7.,  5.])
>>> A
array([[ 8.,  1.,  6.],
       [ 3.,  5.,  7.],
       [ 4.,  9.,  2.]])
```

Attributes

```
>>> A.shape
(3, 3)
>>> A.flatten()
array([ 8.,  1.,  6.,  3.,  5.,  7.,  4.,  9.,  2.])
>>> B=A.copy()
>>> B[1,1]=-1
>>> A[1,1]
5.0
>>> B[1,1]
-1.0
>>> A.transpose()
array([[ 8.,  3.,  4.],
       [ 1.,  5.,  9.],
       [ 6.,  7.,  2.]])
```

Methods

```
>>> x=np.arange(24)                    # array-range
>>> y=x.reshape([4,6]).copy()          # turn into 4 X 6 matrix
>>> y
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23]])
>>> np.sum(y)                          # sum all of y
276
>>> y.sum()                            # sum all of y
276
>>> y.sum(0)                           # sum columns
array([36, 40, 44, 48, 52, 56])
>>> y.sum(axis=0)                      # sum columns
array([36, 40, 44, 48, 52, 56])
>>> np.sum(y,axis=0)                   # sum columns
array([36, 40, 44, 48, 52, 56])
>>> np.sum(y[:,0])                     # sum only first column
36
>>> np.sum(y[:,1])                     # sum only second column
40
>>> y.sum(1)                           # sum along rows
array([ 15,  51,  87, 123])
>>> y.sum(axis=1)                      # sum along rows
array([ 15,  51,  87, 123])
>>> np.sum(y[0,:])                     # sum first row
15
```

Useful attributes

- $\pi =$**`np.pi`**, $e =$**`np.e`**
- 1d array of zeros **`x=np.zeros(N)`**
- 2d array of zeros **`x=np.zeros([N,M])`**
- Array of zeros same size as: **`y=np.zeros_like(x)`**
- **`np.ones`**, **`np.ones_like`**
- **`np.empty`**, **`np.empty_like`**

Functions like MATLAB

- `np.diag`
- `np.random.rand`
- `np.tril`, `np.triu`

Operations on arrays

Suppose **`x`** is the array $[x_i]$, **`y`** is the array $[y_i]$, **`A`** is the 2d array $[a_{ij}]$ and **`B`** is the 2d array $[b_{ij}]$:

- All operations are elementwise.
- **`np.sin(x)`** is the array $[sin(x_i)]$, same for matrices
- **`3*x`** is the array $[3x_i]$, same for matrices
- **`x+y`** is the array $[x_i + y_i]$, same for matrices
- **`x*y`** is the array $[x_i y_i]$, same for matrices
- **`x/y`** is the array $[x_i / y_i]$, same for matrices
- **`np.dot(x,y)`**$=\sum_i x_i y_i$ (usual dot product)
- **`np.dot(A,B)`**$=[\sum_k a_{ik} b_{kj}]$ (usual matrix multiplication)

Array selection

It is sometimes convenient to select particular elements from arrays.

- **`np.max(x)`** finds the maximum value.
- **`np.argmax(x)`** finds the locaion of the maximum value.
- Can use **`axis=n`**
- Also **`np.min`**, **`np.argmin`**
- Boolean selection

```
>>> x=np.array([[5.,3.],[4.,9.]])
>>> x[x>=5]
array([ 5.,   9.])
>>> x[np.logical_and(x>=5,x<8)]
array([ 5.])
```

Array selection by subscript

- One-dimensional arrays

```
>>> x=2*np.arange(9.)
>>> x
array([  0.,   2.,   4.,   6.,   8.,  10.,  12.,  14.,  16.])
>>> i=np.array([0,0,1,5,2])
>>> i
array([0, 0, 1, 5, 2])
>>> x[i]
array([  0.,    0.,    2.,   10.,    4.])
```

- Two-dimensional arrays: selector for each axis.

```
>>> A=x.reshape([3,3])
>>> A
array([[   0.,    2.,    4.],
       [   6.,    8.,   10.],
       [  12.,   14.,   16.]])
# construct minor associated with A[1,0]
>>> i=[[0,0],[2,2]]   # 2-axis means result will be 2-axis
>>> j=[[1,2],[1,2]]
>>> A[i,j]
array([[  2.,   4.],
       [ 14.,  16.]])
```

Higher-level linear algebra

- **`import scipy.linalg as la`**
- Built on BLAS and Lapack
- **`la.norm`**: vector and matrix norm
- **`la.det`**: determinant
- **`la.solve`**: solve system of equations
- **`la.inv`**: construct inverse matrix
- **`la.eig`**: eigenvalues and eigenvectors
- **`la.eigvals`**: eigenvalues only
- **`fenics`** does not use this package!

Plotting

- `import matplotlib.pyplot as plt`
- Must use `plt.show()` to see your plot!
- Plot like MATLAB

```
>>> x=np.linspace(0,4*np.pi,1000)
>>> plt.plot(x,np.exp(-x),x,np.sin(x))
[<matplotlib.lines.Line2D object at 0x4306a90>, <matp
>>> plt.legend(('exp','sin'))
<matplotlib.legend.Legend object at 0x45df890>
>>> plt.show()
```

- `plt.semilogy` also works