Computer Methods (MAE 3403)

Introduction to numpy

Introducing numpy arrays

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

- Numpy array is mostly related to numerical methods

numpy array

```
import numpy as np
x = np.array([1, 4, 3])
y = np.array([[1, 4, 3], [9, 2, 7]])
y.shape
y.size
```

dtype: data type can be complex, np.float32, np.float64, float, int16

structured and predefined arrays

```
z = np.arange(1, 2000, 1)
np.arange(0.5, 3, 0.5)
np.linspace(3, 9, 10)

np.zeros((3, 5))
np.ones((5, 3))
np.eye(3) #identity matrix of dim 3.
np.empty(3) # filled with random very small numbers
```

arange(from,to,increment) function

- Similar to the range function, but returns an array rather than a list.

```
>>> from numpy import *
>>> print(arange(2,10,2))
[2 4 6 8]
>>> print(arange(2.0,10.0,2.0))
[ 2. 4. 6. 8.]
```

```
>>> print(zeros(3))
[ 0. 0. 0.]
>>> print(zeros((3),int))
[0 0 0]
>>> print(ones((2,2)))
[[ 1. 1.]
[ 1. 1.]]
```

Access and change array elements

- For size-2 arrays, a[i,j] accesses the element in row *i* and column *j* . a[i] refers to row *i.*

```
>>> from numpy import *
>>> a = zeros((3,3),int)
>>> print(a)
[[0 0 0]
[0 0 0]
[0 0 0]]
>>> a[0] = [2,3,2] # Change a row
>>> a[1,1] = 5 # Change an element
>>> a[2,0:2] = [8,-3] # Change part of a row
>>> print(a)
```

Indexing

- 1D array x: x[1] # 2nd element, x[1:] #slicing, x[-1]
- 2D array y:
  - y[0,1] # first row and 2nd column
  - y[0,:] # first row of y
  - y[:,-1] # last column of y
  - y[:,[0 2]] # first and third column of y
  - y[1:3,:] # 2nd and third row of y

Assignment

- a = np.arange(1, 7) # array([1, 2, 3, 4, 5, 6])
- a[3] = 7
- a[:3]=1
- a[1:4]=[9,8,7]

Array operations

- Between a scalar (c) and an array (b)
  - b+c, b-c, b*c, b/c, b**c: operates on each element of b with c
- Between two arrays (b and d)
  - b+d, b-d correspond to matrix addition and subtraction
  - b*d, b/d, b**d correspond to element-wise matrix multiplication, division and power
  - Numpy has many arithmetic functions, e.g., cos, sin, sqrt. **They operate on each element of an array**.

Matrices

- Created as 2D or nD arrays
- Matrix multiplication handled by the dot method
  - P*Q in python: np.dot(P,Q)
  - Other methods: P@Q, P.dot(Q), np.matmul(P, Q)
  - Make sure the dimensions of P and Q are compatible for matrix multiplication.

Logic operations

```
x = np.array([1, 2, 4, 5, 9, 3])
y = np.array([0, 2, 3, 1, 2, 3])
x > 3
x > y
y=x[x>3]
x[x>3]=0
```

Review of linear algebra and numpy

- Vectors: column and row vectors
- Norm of a vector measures the magnitude of a vector with respect to the origin.

$$\|v\|_2 = \sqrt{\sum_i v_i^2},\ \|v\|_p = \sqrt[p]{(\sum_i v_i^p)},\ \|v\|_\infty = \max_i |v_i|$$

```
import numpy as np
vector_row = np.array([[1, -5, 3, 2, 4]])
vector_column = np.array([[1], [2], [3], [4]])
print(vector_row.shape)
print(vector_column.shape)
```

Notice we used nested list to define the vectors. Try

```
vector = np.array([1,2,3,4])
print(vector.shape)
```

Operations of vectors

- Transpose: .T
- Computation of norm
- multiplication: scalar multiplication, dot product, cross product

```
from numpy.linalg import norm
new_vector = vector_row.T
print(new_vector)
norm_1 = norm(new_vector, 1)
norm_2 = norm(new_vector, 2)
norm_inf = norm(new_vector, np.inf)
print('L_1 is: %.1f'%norm_1)
print('L_2 is: %.1f'%norm_2)
print('L_inf is: %.1f'%norm_inf)
```

Linear algebra module

- numpy has linalg containing routine tasks, such matrix inversion, etc.

```
from numpy import array
from numpy.linalg import inv
A = array([[ 4.0, -2.0, 1.0], \
[-2.0, 4.0, -2.0], \
[ 1.0, -2.0, 3.0]])
print(inv(A))
```

Square matrix

- Determinant: **from numpy.linalg import** det
- Inverse: **from numpy.linalg import** inv
- Condition number: **from numpy.linalg import** cond
- Rank: **from numpy.linalg import** matrix_rank

Misc

- np.hstack: stack horizontally, np.vstack: stack vertically
- Matrix concatenation: np.concatenate((A,B), axis=)
- for row in b:
  - iterate through each row
- b.flat: flatten the array to a column vector
- np.reshape(# row, # column)
- Shallow copy: b = a # a changes as b changes
- deepcopy: d = a.copy() # a and d are now independent

Python implementations

- Numerous and SIMPLE ways to solve systems of linear equations in Python using the numpy module
- numpy.linalg.solve (LU decomposition)

```python
import numpy as np
A = np.array([[4, 3, -5],
              [-2, -4, 5],
              [8, 8, 0]])
y = np.array([2, 5, -3])
x = np.linalg.solve(A, y)
print(x)
```

- Matrix inverse:

```python
A_inv = np.linalg.inv(A)
x = np.dot(A_inv, y)
```