UECM1703 Introduction to Scientific Computing: Topic 2: Array Programming

Dr Liew How Hui

Oct 2020

Outline

Array "Syntax"

To make the *n*-D array processing easy, we need some kind of syntax. Let `A` and `B` be "numeric" arrays of the same (or similar) shape.

- Subarray: Numpy array slicing
- Point-wise/vector operations: `A+B`, `A-B`, `A*B`, `A/B`, `A+=B`, ..., `A<B`, `A<=B`, `A>B`, `A>=B`, `A==B`, `A~=B`, ..., `f(A)`, ... defined below.
- Boolean operations and masking.
- Linear algebra with vectors `x` and matrices `A`: E.g. `np.dot(A, x)`

Numpy Array

The major problems with Python's "collections" data types such as list are:

- List is slow, i.e. processing list can be 10 times or sometimes even 100 times slower than Numpy arrays.
- List does not have nice "array operations" (It is mentioned in Practical 1). Numpy array has nice syntax similar (but different) to MATLAB, which makes it nice for machine learning prototyping.

Definition of an Array

An *array* is a “multi-index” **homogeneous** data structure with elements of the same type.

We can have:

- Array of Booleans
- Array of Integers
- Array of Floating Point Numbers (Key in Scientific Computing)
- Array of Strings (not use in Scientific Computing)

Numpy Array

The Python module `Numpy` resolves this problem by introducing an *n*-D array object, `numpy.ndarray`. To use the Numpy module, we need to import it into Python:

```
>>> import numpy as np
```

After the "name space" for the Numpy module is imported, we can assess all the functions in Numpy by prefixing them with "`np.`". To list functions that are available in Numpy, we can use the command:

```
>>> dir(np)
```

Numpy Array (cont)

Ways of creating 1-D arrays:

- `np.array([1,2,3,4],dtype=np.double)`
- `np.arange(1,5)`
- `np.r_[1:5]`, `np.r_[1:50:2]`
- `np.linspace(start,stop,num=50, endpoint=True,retstep=False,dtype=None)`

Ways of creating 2-D arrays:

- `np.array([[1,3],[4,5]],dtype=np.double).`
- `np.array([1,3,4,5]).reshape((2,2))`
- Special arrays (next slide)

Numpy Array (cont)

Ways of creating "special" 2-D arrays:

- `np.zeros((2,4))`
- `np.eye(n)` creates an $n \times n$ array. E.g.

$$\texttt{np.eye(4)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \texttt{np.eye(3,2)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

- `np.ones` creates an array of ones
- `np.full((m1,···,mk),v)` creates a full array of `v`.

Numpy Array (cont)

- `np.diag(D)` is used to construct an $n \times n$ matrix with diagonal elements from `D`. E.g.

$$\texttt{np.diag(np.arange(5,8))} = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

- `np.vander(`$[x_1, \cdots, x_n]$`)` is used to construct the Vandermonde matrix arises in interpolation.

Numpy Array (cont)

- Other special matrices are available from `scipy.linalg` module: `block_diag(*arrs)`, `circulant(c)`, `companion(a)`, `convolution_matrix(a, n[, mode])`, `dft(n[, scale])`, `fiedler(a)`, `hadamard(n[, dtype])`, `hankel(c[, r])`, `helmert(n[, full])`, `hilbert(n)`, `invhilbert(n[, exact])`, `leslie(f, s)`, `pascal(n[, kind, exact])`, `invpascal(n[, kind, exact])`, `toeplitz(c[, r])`, `tri(N[, M, k, dtype])`

Numpy Array (cont)

Special matrices with random numbers are essential in computer simulation.

- An array of uniform random numbers between 0 and 1:

```
np.random.rand(m1,...,mk)
np.random.random((m1,...,mk))
```

- An array of normally distributed random numbers:

```
np.random.randn(d0,d1,...,dn)
```

- To prevent the random numbers to be different every time we use them, the seed should be set:

```
np.random.seed(some_integer)
```

Numpy Array (cont)

Ways of creating “special” 3-D arrays:

- `np.array([[[6,3],[9,8]], [[1,3],[4,5]]],dtype=np.double).`
- `np.r_[1:(2*3*4+1)].reshape((2,3,4))`
- Load from an “array” file: `np.loadtxt("array2d.txt").reshape(...)`

Ways of creating general $n \geq 4$-D arrays: similar to 3-D cases but rarely used.

Outline

Indexing / Slicing

Index: Integer(s) to assess “elements” of an array.

Consider a 2-D array `A`.

- Indexing an element of `A` at `(m,n)`: `A[m-1,n-1]`
- Indexing `A` at `m`-row: `A[m-1,:]`
- Indexing `A` at `n`-column: `A[:,n-1]`
- Indexing using rows & columns:
  `A[m1-1:m2,n1-1:n2]`,
  `A[m1-1:m2:ms,n1-1:n2:ns]`.
- Indexing a sub-array of `A` using rows `r1`, $\cdots$, `rk`, columns `c1`, $\cdots$, `cl`:
  `A[[r1-1,...,rk-1],:][:,[c1-1,...,cl-1]]`

Indexing: 2D Example

```
A=np.arange(1,55,dtype=np.double).reshape(6,9)
```

$A =$

	0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8	9
1	10	11	12	13	14	15	16	17	18
2	19	20	21	22	23	24	25	26	27
3	28	29	30	31	32	33	34	35	36
4	37	38	39	40	41	42	43	44	45
5	46	47	48	49	50	51	52	53	54

Study the following "indexing"/"slicing" commands:

```
(a) A[3,4]          (b) A[3,:]          (c) A[:,4]
(d) A[1:3,3:7]      (e) A[1:6:2,:][:,3:8:2]
```

Indexing: 2D Example (cont)

Numpy provides an alternative indexing function `take` (or `put`) to take (or assign values to) a "slice" of elements from an array `A`. To get an element from an item, the function `item` is used instead:

```
(a) A.item(3,4)
(b) A.take(indices=[3],axis=0)
(c) A.take(indices=[4],axis=1)
(d) A.take(range(3,7),axis=1).take(range(1,3),axis=0)
(d) A.take([3,5,7],axis=1).take([1,3,5],axis=0)
```

Indexing: 3D Example

A 3-D array is used in the representation of a coloured image. Python's PIL.Image.open or imageio.imread could be used to load images to 3-D arrays.

```
1  import numpy as np
2  from PIL import Image
3  import matplotlib.pylab as plt
4  im = Image.open("emoji1.jpg")   # https://getemoji.com/
5  imarr = np.array(im)
6  #imarr = np.full((5,5,3),255)
7  fig=plt.figure(figsize=(8, 8))
8  mycmaps = [plt.cm.Reds_r,plt.cm.Greens_r,plt.cm.Blues_r]
9  nrows = 1
0  ncols = 4
1  for i in range(3):
2      fig.add_subplot(nrows, ncols, i+1)
3      plt.imshow(imarr[:,:,i],interpolation='None',cmap=mycmaps[i])
4  fig.add_subplot(nrows, ncols, 4)
5  plt.imshow(imarr)
6  plt.show()
```

Indexing: 3D Example (cont)

`imarr.shape` $\Rightarrow$

Red: `imarr[:,:,0]`)

$\Rightarrow$

14	4	0	0	6	19	29	27	29	14	15	7	8	19	11
10	255	255	255	255	247	243	241	224	234	231	249	255	255	15
14	255	255	255	241	247	246	249	252	236	230	251	248	255	3
7	255	255	233	240	253	214	232	251	240	240	245	230	227	0
3	255	238	236	227	156	116	138	237	209	141	123	218	221	0
19	255	231	247	170	132	136	117	225	135	114	110	151	203	24
53	252	252	255	228	180	176	207	243	195	161	158	228	255	16
84	255	255	255	255	255	255	255	253	255	255	255	255	255	20
85	255	247	255	255	255	255	255	251	255	255	255	255	225	33
52	255	245	249	240	243	254	229	255	245	244	240	243	220	30
16	255	241	243	241	241	243	203	145	177	227	247	246	224	20
6	255	249	244	247	251	252	226	235	238	246	246	245	233	5
7	255	255	251	245	249	252	255	250	250	253	248	247	244	0
6	255	255	255	255	253	244	253	252	240	235	246	255	255	12
1	254	249	252	255	255	252	242	227	238	252	255	255	255	8
0	251	240	245	254	253	255	255	255	255	251	255	255	255	7
0	0	0	0	0	3	6	6	4	3	1	1	1	1	3

Green: `imarr[:,:,1]`)

Blue: `imarr[:,:,2]`)

Ref: `https://en.wikipedia.org/wiki/RGB_color_model`

Indexing / Slicing (cont)

Applications of Indexing:

- Obtain sub-array(s): E.g. cutting a “rectangular” piece of large image.
- Arranging matrix to special form: E.g.

$$A = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 2 & 2 \\ 3 & 8 & 3 \end{bmatrix} \Rightarrow \texttt{A[[1,2,0],:]} = \begin{bmatrix} 5 & 2 & 2 \\ 3 & 8 & 3 \\ 1 & 1 & 3 \end{bmatrix}$$

The right-hand matrix is “diagonally dominant”.

Indexing: Row vs Column Major Order

The “C index” and “Fortran index” is formally called the row- and column-major order (see `https://en.wikipedia.org/wiki/Row-_and_column-major_order`). It works as follows:

Array	Starts	1	2	3	4
C index	0	`a[0,0]`	`a[0,1]`	`a[1,0]`	`a[1,1]`
Fortran index	1	`a(1,1)`	`a(2,1)`	`a(1,2)`	`a(2,2)`

Translating from MATLAB

The three things we need to beware when porting MATLAB to Python:

1. MATLAB index start from 1 and Python index start from 0;
2. When porting MATLAB code to Python/Numpy code, we need to know that MATLAB is initially developed as an interface to Fortran and the array are stored in Fortran index while Python is developed based on C index.
3. Index slicing in Numpy may just create **views** instead of **copies**.

Translating from MATLAB

Study the following MATLAB and Python/Numpy instructions and explain what is each output:

`A = [1:3; 4:6];`
`B = A(1:2,1:2);`
`B(1,1) = 8;`
`disp("A=")`
`disp(A)`
`disp("B=")`
`disp(B)`
	`import numpy as np`
	`A = np.array([[1,2,3],[4,5,6]])`
	`B = A[0:2,0:2]`
	`B[0,0] = 8`
	`print("A=",A)`
	`print("B=",B)`

$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, $B = \begin{bmatrix} 8 & 2 \\ 4 & 5 \end{bmatrix}$	$A = \begin{bmatrix} 8 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, $B = \begin{bmatrix} 8 & 2 \\ 4 & 5 \end{bmatrix}$

Example: Sept 2012 FE, Q3(b)

Given the array `A = [2 1 0 0;0 0 -2 6;1 1 3 4]`, explain the results of the following commands.

(i) `A([2 1],[3 2])` (2 marks)

(ii) `reshape(A,2,6)` (2 marks)

(iii) `A; A(1:2,:)` (2 marks)

(iv) Translate all the MATLAB commands in part (i) to (iii) to Python/Numpy.

Example: Sept 2015 FE, Q1(a)

Consider an array in MATLAB:

$$M = \begin{bmatrix} 6 & 9 & 12 & 4 & 3 & 0 \\ 4 & 4 & 15 & 2 & 1 & 1 \\ 2 & 1 & 18 & -5 & 8 & 2 \\ -6 & -4 & 21 & 1 & -5 & 2 \end{bmatrix}.$$

What will display if the following commands in (i) to (iii) are executed by MATLAB?

(i) `A = M([1,3],[2,4])` (2 marks)

(ii) `B = M(:,[1,4:6])` (2 marks)

(iii) `C = M([2:3],:)` (2 marks)

Translate the MATLAB instructions to Python/Numpy instructions.

Example: Sept 2015, Q1(c)

Given that an array `A = [1:10;11:20;21:30]`. Reshape `A` into a two-row array. What is the top and bottom number in each column? (Write the MATLAB command used to reshape array `A` and display the new array `A`). (4 marks)

Example: Sept 2015, Q3(a)

The following vector is defined in MATLAB:

```
V = [2 7 -3 5 0 14 -1 10 -6 8]
```

What will be displayed if the following commands in (i) to (iii) are executed by MATLAB?

1. `>> B = V([2,4:7,10])` (2 marks)
2. `>> C = V([9,3,2,10])` (2 marks)
3. `>> D = [V([1,3,5]); V([2,4,6]); V([3,6,9])]` (2 marks)
4. Translate the commands in part (i) to (iii) to Python/Numpy.

Example: Sept 2014 FE, Q2(b)

Consider the following MATLAB code.

```
clear;
for i=1:4
    for j=i:4
        M(i, j) = i+j;
        M(j, i) = M(i,j)-2;
    end
end
```

1. Write down the value for `M` and `M(4,3)` after the code is executed.
2. Rewrite the MATLAB code into Python code.

Example: Numerical Methods

In applying finite difference approximation to the type 1 ODE-BVP:

$$y''(x)+p(x)y'(x)+q(x)y(x) = r(x),\ y(a) = y_a,\ y(b) = y_b$$

we obtain matrix

$$C = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1-\frac{h}{2}p(x_1) & (h^2q(x_1)-2) & 1+\frac{h}{2}p(x_1) & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 1-\frac{h}{2}p(x_{n-1}) & (h^2q(x_{n-1})-2) & 1+\frac{h}{2}p(x_{n-1}) \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

where $a < x < b$, $h = \frac{b-a}{n}$. Write a Python script to generate the matrix C.

Outline

Array Operations

The basic methods (operations) associated with Numpy array `A`:

- Information about an array:
  - `A.ndim`: the dimension of `A`;
  - `A.shape`: $(m_1, m_2, \cdots, m_k)$, k=dim `A`
  - `A.size`: number of elements $m_1 \times m_2 \times \cdots \times m_k$
  - `A.nbytes`: the total number of bytes used, i.e. `A.size*A.itemsize`. E.g. if there are n elements in `A` and all the elements are 64-bit floating numbers, then the total number of bytes used in `A` to store array data is $8n$.
- "Negative" indexing: Count backwards. Not very nice for scientific computing.

Array Operations (cont)

1. Getting a "view" of the array `A`.
   - Indexing usually returns a "view" of `A`.
   - Transpose view: `A.T`, `np.transpose(A)`.
2. Creating a "copy" of an array `A`.
   - Return a duplicate of `A`: `A.copy()`.
   - Return a "copy" of `A` with a specific type: `A.astype(sometype)`, here `sometype` can be `'double'`, `'bool'`, `'int8'`, etc.
   - Return a new array by stacking existing array(s): `np.hstack` and `np.vstack`.
3. Converting an array to string (for us to see its content): `A.tostring(order='C')`.

Array Operations (cont)

Operations for compatible numeric arrays `A`, ...

- Scalar multiplication: `5.34*A`
- abs()
- +, -, *, /, **
- >, >=, ==, != , <=, <

Remark: MATLAB uses 0 and 1 for True and False.

Example:

```
A = np.r_[1:10].reshape((3,3))
A + np.r_[-1:-4:-1]
A > np.r_[2:5]
```

Array Operations (cont)

Operations for compatible “Boolean” arrays `B`, ...

- | or +, & or *, ~ or -
- np.any(), np.all()

Example:

```
np.random.seed(2020)
A = np.random.randint(-5,5,3*4).reshape((3,4))
B = A > 0
C = np.random.randint(-5,5,3*4).reshape((3,4)) < 0
B | C, B & C, -B
```

Example

In your Python environment, after you have executed the following instructions:

```
>>> p = np.array([True,True,False,False])
>>> q = np.array([True,False,True,False])
```

Command	Output
-p	array([False, False, True, True], dtype=bool)
~p	array([False, False, True, True], dtype=bool)
p&q	array([True, False, False, False], dtype=bool)
p*q	array([True, False, False, False], dtype=bool)
p|q	array([True, True, True, False], dtype=bool)
p+q	array([True, True, True, False], dtype=bool)

Example: Sept 2015 FE, Q2(a)(iv)

- What will display if the following commands are executed by MATLAB?

  ```
  >> [~0 0 ~1] & [1 1 1]
  ```

  (2 marks)
- What is the corresponding instruction in Python/Numpy?

Example: Sept 2012 FE, Q1(c)

Given that `x=[1 5 2 8 9 0 1]` and `y=[5 2 2 6 0 0 2]`, execute and explain the results of the following commands (i) to (iii):

1. `x|y` (2 marks)
2. `x&(~y)` (2 marks)
3. `(x>y)&(y<x)` (2 marks)
4. Translate the commands in part (i) to (iii) to Python/Numpy commands.

Example: Sept 2013 FE, Q1(c)

Given that `x=[1 3 4 2 5 0 -3]` and `y=[6 3 2 4 1 0 6]`, list the results of the following commands (i) to (iii):

1. `x-2*(y>3)` (2 marks)
2. `x&(~y)` (2 marks)
3. `(x==y)|(y<x)` (2 marks)
4. Translate all the commands (i) to (iii) to Python/Numpy commands.

Example: Sept 2014 FE, Q1(a)

Given that `x = [-1, 2, 3, -2]`, `y = [0.2, 3.1, 0, -3]` and `z = [3, 1, 1, 1]`, write down the outputs to the following MATLAB operations. Explain your answer.

(i) `x < y > z` (2 marks)

(ii) `x + ~y > z` (2 marks)

(iii) `x == y ~= z` (3 marks)

(iv) `x(z) > y` (3 marks)

Translate all the instructions to Python/Numpy instructions.

Example: Sept 2014 FE, Q1(a) cont

In (i), `x < y` returns true or false values, comparing true or false values with `z` is not really meaningful. In MATLAB, the direct conversion from Boolean to Numbers can be misused but in Python/Numpy, we need can compare types which are sensible:

```
x=np.array([-1.,2,3,-2])
y=np.array([0.2,3.1,0,-3])
z=np.array([3.,1,1,1])
>>> x < y > z  # Not allowed: precedence ambiguity
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: The truth value of an array with more than
one element is ambiguous. Use a.any() or a.all()
>>> (x < y) > z # This is OK.
```

Example: Oct 2018 FE, Q1(a), CO1

The output of the Python commands below

```
>>> import numpy as np
>>> A = np.arange(1,36).astype('float').reshape(5,7)
>>> print(A)
```

is

```
[[  1.   2.   3.   4.   5.   6.   7.]
 [  8.   9.  10.  11.  12.  13.  14.]
 [ 15.  16.  17.  18.  19.  20.  21.]
 [ 22.  23.  24.  25.  26.  27.  28.]
 [ 29.  30.  31.  32.  33.  34.  35.]]
```

Example: Oct 2018 FE, Q1(a), CO1

Use the above information to write down the output to the following Python commands for item (i) to item (iv).

(i) `print(A[2,:])` (2 marks)

(ii) `print(A[1:4,3:5])` (3 marks)

(iii) `print(A[2:4,4:6].mean())` (5 marks)

(iv) `print(A[:,2]>10)` (3 marks)

(v) Write down the Python command to **count** the number of elements in `A` who are larger than 20. (3 marks)

Array Functions

Mathematical Fn	Python/Numpy	Inverse/Pair Function	Python/Numpy
Exponential, e^x	`np.exp`	Natural log, ln x	`np.log`
10^x	`np.power (10,x)`	Base 10 log, $\log_{10} x$	`np.log10`
Hyperbolic sine, $\sinh(x) = \frac{e^x - e^{-x}}{2}$	`np.sinh`	Inverse hyperbolic sine	`np.arcsinh`
Hyperbolic cosine, $\cosh(x) = \frac{e^x + e^{-x}}{2}$	`np.cosh`	Inverse hyperbolic cosine	`np.arccosh`
Hyperbolic tangent, $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	`np.cosh`	Inverse hyperbolic tangent	`np.arctanh`
Sine	`np.sin`	Inverse sine	`np.arcsin`
Cosine	`np.cos`	Inverse cosine	`np.arccos`
Tangent	`np.tan`	Inverse tangent	`np.arctan`
Floor	`np.floor`	Ceiling	`np.ceil`
Rounding	`np.round`	-	-

Note: angle(s) in **radian**.

Array Functions (cont)

Array Functions are called *universal functions* ("ufunc"s for short).

Example:

Let

$$A = [x1, x2, x3], \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}.$$

The array function `np.sin` works as follows:

$$\texttt{np.sin(A)} = [\sin(x1), \sin(x2), \sin(x3)]$$

$$\texttt{np.sin(B)} = \begin{bmatrix} \sin(1) & \sin(2) & \sin(3) \\ \sin(4) & \sin(5) & \sin(6) \end{bmatrix}.$$

Array Functions (cont)

Example:

Use the array mathematical functions to generate the following arrays in Python:

1. `A1`=[$\sin 10°$, $\sin 30°$, $\sin 50°$, $\sin 70°$]
2. `A2`=[$\tan 6°$, $\tan 42°$, $\tan 66°$, $\tan 78°$]

Answer:

- `A1 = np.sin(np.radians([10,30,50,70]))`
- Exercise: Try to write down the answer for `A2`.

Something to Beware of Rounding

According to `https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.around.html`, for values exactly halfway between rounded decimal values, NumPy rounds to the nearest even value. Thus 1.5 and 2.5 round to 2.0, –0.5 and 0.5 round to 0.0, etc. Results may also be surprising due to the inexact representation of decimal fractions in the IEEE floating point standard.

Array Functions Example

Use two plotting functions `plt.plot` and `plt.show` to draw (a) Sine function; (b) Cosine function; and (c) floor function for the domain $[-2\pi, 2\pi]$.

Answer:

```
import matplotlib.pylab as plt, numpy as np
x = np.linspace(-2*np.pi,2*np.pi,100)
y1, y2, y3 = np.sin(x), np.cos(x), np.floor(x)
plt.plot(x,y1,x,y2,x,y3)
plt.show()  # Not needed in Spyder
```

Example: Sept 2011 FE, Q3(a)

Consider the following M-file

```
1  % FILE: myfun.m
2  function ans = myfunc(x,y)
3  if x < y              % True only when all values in vector are 1
4    ans = (y-x) * y'; % Adding a transpose will resolve the error
5  else
6    ans = (x-y) * x';
7  end
```

Translate the MATLAB program to a Python program.

Vector Representation of Loop

Modern “data analysis” programming emphasises “array” and “array” functions. To support the definition of an “array” function, we can use

- `np.vectorize`
- `np.frompyfunc`.

Array Functions (cont)

A friendly way of defining "ufunc" with only one input is using the basic method `np.frompyfunc`.

Example: Given a function

$$\text{sinhc} : \mathbb{R}^+ \to \mathbb{R}^+, \quad x \mapsto \frac{\sinh x}{x}.$$

1. Define a Python function implementing the sinhc function.
2. Calculate `sinhc(1)`. What is the output? Why?
3. Calculate `sinhc([1,2,3,4,5,6])`. What is the output? Why?
4. Define a ufunc `usinhc` to calculate `usinhc([1,2,3,4,5,6])`.

Array Operations: Stacking

If we have two compatible matrices

$$\mathtt{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad \mathtt{B} = \begin{bmatrix} 9 & 8 \\ 7 & 6 \end{bmatrix}$$

we can stack them:

- 1D horizontally: `np.stack([1,2,3],[4,5,6])`
- 2D horizontally: `np.hstack((A,B))`
- 2D vertically: `np.vstack((A.T,B))`

Array Operations: Stacking (cont)

```
import numpy as np, matplotlib.pylab as plt
A = np.r_[1:7].reshape((2,3))
B = np.array([[9,8],[7,6]])
#print(np.stack((A,B),axis=0))
print(np.hstack((A,B)))
print(np.vstack((A.T,B)))
x = np.linspace(-2*np.pi,2*np.pi,100)
plt.plot(x,np.stack((np.sin(x),np.cos(x)),axis=1))
plt.show()
```

Array Functions Example

Write a **Python script** to plot the functions $y_1 = \sin x$, $y_2 = 5 \sin 2x$ and $y_3 = 3 \sin 4x$ in one diagram for $-\pi \leq x \leq x$ using

- CSV + Excel method
- Matplotlib method

Array Operations: Stat Functions

In addition to the arithmetic operations above, there are stat operations which reduces a **1-D array** to a number.

Array Ops	Definition	Python	MATLAB
sum	Total sum over a specified dimension	`A.sum()`	`sum(A)`
product	Total product over a specified dimension	`A.prod()`	`prod(A)`
average	Average, possibly weighted, over axis or array	`A.mean()`	`mean(A)`
population "var"	Finds the "biased" variance of the array elements along given axis	`A.var()`	`var(A,1)`
population "stdev"	"biased" standard deviation	`A.std()`	`std(A,1)`
sample "var"	Finds the "unbiased" variance	`A.var(ddof=1)`	`var(A)`
sample "stdev"	"unbiased" standard deviation	`A.std(ddof=1)`	`std(A)`

Example: Sept 2015 FE, Q1(b)

(i) Write down and explain the values of `C` for the following MATLAB command

```
>> A = [4 6 8];
>> B = [2 0 4];
>> C = sum(A./B)
```

(3 marks)

(ii) Write down the equivalent instructions for Python/Numpy.

Array Operations: Stat Functions (cont)

When we have a **2-D array** (or higher dimension array) A, the stat functions can reduce A along the "given dimension/axis" (1 or 2, by default the dimension is 1) in MATLAB. The dimension 1 and 2 in MATLAB corresponds to axis 0 and 1 in Python respectively.

UEEP2083 Sept 2014 FE, Q1(c)

- For commands typed in MATLAB, the corresponding return are as follows:

```
>> x=4;
>> A=[8, 6, 7, 9; 3, 8, 5, x+2];
>> sum(1:2,2::);
Error: Unexpected MATLAB operator.
```

  Identify and comment on the error in the above commands. Suggest a correction to fix the error.

  (2 marks)

- Translate the corrected MATLAB command to Python command.

Example: Sept 2013 FE, Q2(b)

Write a Python script that perform the following actions:

- Load an array of numbers from a file named `data.txt` and,
- Save the output of ab/n in `result.txt` where
  - a is the largest value in the array,
  - b is the smallest value in the array, and
  - n is the total number of elements in the array. (9 marks)

Outline

Boolean Indexing

The “Boolean” array for an array A generated with the use of relational operations can be used as a kind of *fancy indexing* called *Boolean indexing* for A.

This kind of indexing is widely used in statistics, image processing, signal processing, etc. because it allows us to “filter” out wanted or unwanted data in an array. In the following, we show examples of the applications of Boolean indexing.

Boolean Indexing Example

The test 2 results of UECM3033 Numerical Methods for the Jan 2018 semester are

11.5, 15.1, 10.8, 14.1, 5.8, 4.1, 15.7, 13.3, 14.6, 5.2, 13.1, 8.6, 8.8, 16.3, 11.7, 13.9, 13.6, 14.5, 11, 14, 13.7, 16.1, 12.1, 9.7, 14.9, 12, 10.5, 12.8, 15.3, 4.8, 13.1, 0, 12.5, 8.8, 14.4, 12.7, 8.8, 11.9, 13.1, 14.4, 7.3, 17.1, 9.3, 11, 13.5, 9, 7.9, 4.7, 13.8, 15.5, 13.2, 8.8, 10.1, 13.3, 9.5, 10.5, 12.6, 14.6, 12.8, 11, 2.7, 6.2, 10.6, 14.5, 13.4, 10.5, 11.3, 14.8, 9.9, 8.8, 14.2, 9.7, 9.4, 9, 13.5, 10.3

The full mark for test 2 is 20 marks and the passing mark is 10. Find all those marks which is below 10. How many students fail?

Answer: `M=np.array([11.5,...])`, `fails = M[M<10]` and `sum(fails)`.

Remark

One can identify the indices of those that satisfies a predicate `p(x)` with `np.nonzero(p(anarray))`. In the previous Example, `p(x)=(x<10)`, the indices of those who fail the test can be found by `np.nonzero(M<10)`.

Example: Sept 2013 FE, Q1(b)

Write a Python script to perform the following actions:

- Generate a 2-by-3 array of random numbers using `rand` command and,
- Move through the array, element by element, and set any value that is less than 0.2 to 0 and any value that is greater than or equal to 0.2 to 1.

Solution

The intention of the lecturer who set the question is to ask you to express this in for loop but in reality, everyone use the array functions to achieve the stated requirements.
Are you able to write both solutions (with for loop and with array)?

Example

Suppose you have keyed in an array of the following exam data (out of 30 marks):

10 24 NaN 22 25 17 23

The “NaN” indicates that a student is absent. Find the average and standard deviation by filtering “NaN”.

Example From https://www.python-course.eu/numpy_masking.php

Extract from the array `B=np.array([3,4,6,10,24,89,45,43,46,99,100])` those numbers

- which are not divisible by 3;
- which are divisible by 5;
- which are divisible by 3 and 5;
- which are divisible by 3 and set them to 42.

Outline

Vectors & Matrices

There is no “vector” type in Numpy.

A 1-D array can be regarded as a *vector*. What makes a “vector” special in mathematics (not computer program) is the “length” `np.linalg.norm(x[, ord, axis, keepdims])` and the “angle” `np.arccos(np.vdot(a, b))` associated with it.

There is a “matrix” type (not recommended) in Numpy. It is a 2-D array with extra feature:

```
M = np.mat('1 2 3;4 5 6')  # or np.matrix('1 2 3;4 5 6')
```

The “multiplication” and “power” for arrays now become “matrix multiplication and power”.

Vectors & Matrices (cont)

According to the documentation `https://docs.scipy.org/doc/numpy/reference/routines.linalg.html`, the matrix and vector products (also work for "n-D array" in which I am not familiar with) the functions associated with matrices are:

- `np.matmul(a, b[, out])`: Matrix product of two arrays.
- `np.linalg.matrix_power(A, n)`: Raise a square matrix to the (integer) power `n`.

Vectors & Matrices (cont)

- `np.dot(A, B[, out])`: Dot product of two arrays, giving $z[I, J, j] = \sum_k A[I, k]B[J, k, j]$.
- `np.linalg.multi_dot(arrays)`: Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order.
- `np.inner(a, b)`: Inner product of two arrays, i.e. $\sum_{i=0}^{K} x_i y_i$ if $K = M$.
- `np.outer(a, b[, out])`: Compute the outer product of two vectors, i.e. $[x_i y_j]$.

Vectors & Matrices (cont)

- `np.tensordot(a, b[, axes])`: Compute tensor dot product along specified axes for arrays $\geq$ 1-D.
- `np.einsum(subscripts, *operands[, out, dtype, ...])`: Evaluates the Einstein summation convention on the operands.
- `np.einsum_path(subscripts, *operands[, optimize])`: Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays.
- `np.kron(A,B)`: Kronecker product of two arrays, giving $[a_{ij\cdots k}B]$.

Vectors & Matrices (cont)

From the documentation, it is difficult to see how to do matrix multiplication in Python. There are many ways to do matrix multiplication. Consider $\mathtt{A}=\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$ and $\mathtt{B}=\begin{bmatrix}7 & 8\\ 8 & 7\end{bmatrix}$, all the following commands give us the matrix multiplication:

```
A@B;     np.matmul(A, B);      np.dot(A, B),
np.linalg.multi_dot((A, B)); np.inner(A,B)
```

Vectors & Matrices (cont)

- `A@B`: Only works in Python version ≥ 3.5
- `np.matmul(A, B)` and `np.dot(A, B)`: Works for Python version < 3.5.

Example: Sept 2011 FE, Q1(b)

Consider the following MATLAB code:

```
v = [1,2,3,4]
V = reshape(v, 2,2)
U = [4,3; 2,1; 0,-1]
X = U * V
x = X(1:2, 1)
y = sum( X > 0 )
z = x .* y'
```

List down the values of the variables `V`, `X`, `x`, `y`, and `z`. Explain how to obtain the values.

Translate the MATLAB code above to Python/Numpy code.

Example: Sept 2012 FE, Q1(a)

Consider the following MATLAB code:

```
P=[2 1 3 5; 4 1 0 6; 1 1 4 7];
A=P([2 3],[1 4])
B=reshape(P,6,2)
C=A*B'
D=B(5:end,:)
E=sum(A~=D)
```

List down and explain the values of the variables `A`, `B`, `C`, `D` and `E`. (10 marks)

Translate the above MATLAB code into Python/Numpy commands.

Example: Sept 2013 FE, 1(a)

Consider the following MATLAB code:

```
1 p=[1,2,3,3]; q=[4,5,1,0];
2 R=complex(p,q)
3 S=reshape(q,2,2)
4 A=R'
5 B=diag(p,1)
6 C=p*q'
```

List down and explain the values of the variables `R`, `S`, `A`, `B`, and `C`. (10 marks)

Example: Sept 2013 FE, Q2(c)

Given the array `A = [1 1 2 3; 4 0 2 3; 2 4 -2 1]`, explain the results of the following commands.

1. `A([1 3],1:3)` (2 marks)
2. `reshape(A,2,6)` (2 marks)
3. `sum(A')` (2 marks)
4. `sum(sum(A))` (2 marks)
5. Translate the commands in (i) to (iv) to Python.

Example: Kronecker Product

Explain which of the Python/Numpy instruction is most appropriate achieved the following results.

(a) Generate the results of multiplication tables for 2 to 9.

(b) Let $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$. Generate a matrix like this

$$\begin{bmatrix} 6 & 8 & 10 & 3 & 4 & 5 \\ 10 & 8 & 6 & 5 & 4 & 3 \\ 3 & 4 & 5 & 6 & 8 & 10 \\ 5 & 4 & 3 & 10 & 8 & 6 \end{bmatrix}.$$

Outline

More Numpy Operations

According to Numpy User Guide (`https://docs.scipy.org/doc/numpy/user/quickstart.html`), there are many useful array functions available in Numpy which can be categorised into:

- **Array Creation**: We have introduced `np.arange`, `np.array`, `copy`, `np.eye`, `np.ones`, `np.zeros`, `np.linspace`. The more advanced functions are `np.empty`, `np.fromfile`, `np.fromfunction` (construct array from universal function), `np.empty_like`, `np.ones_like`, `np.zeros_like`, `np.logspace`, `np.mgrid`, `np.ogrid`.

More Numpy Operations (cont)

- **Manipulations**: We have introduced `reshape`, `transpose`, `hstack` (stack a sequence of arrays along the second axis, i.e. column-wise), `vstack` (stack a sequence of arrays along the first axis, i.e. row-wise).

  More complicated functions such as `indices`, `take` and `put`, `array_split`, `clip` (clip array between two values), `column_stack`, `concatenate` (join arrays together), `diagonal` (return diagonal array), `dsplit`, `dstack`, `hsplit`, `vsplit`, `newaxis`, `ravel` (return array as 1-D), `repeat`, `resize` (return array with arbitrary new shape), `squeeze`, `swapaxes`, are only used in more complicated array construction.

More Numpy Operations (cont)

- **Query**: The functions `all`, `any`, `nonzero` (indices of nonzero elements for 1-D array) and `where` (construct array from binary result) can be very convenient when doing programming.
- **Ordering**: `argmax` (index of largest value), `argmin` (index of smallest value), `argsort` (return indices of sorted array), `min`, `max`, `ptp` (range of values (maximum – minimum) along an axis), `searchsorted`, `sort`.

More Numpy Operations (cont)

- **Operations**: `sum`, `inner` were introduced. There are other "operations" such as `real`, `imag`, `choose`, `compress`, `cumprod` (equivalent function `cumproduct`, cumulative product over a specified dimension), `cumsum` (cumulative sum over a specified dimension), `prod`, `putmask` (used in Markov chains?).
  Let x and y be two 1-D arrays, $K = x.\texttt{size} - 1$ and $M = y.\texttt{size} - 1$, signal processing function `correlate(x,y)` gives $z_j = \sum_{i=\max(j-M,0)}^{\min(j,K)} x_i y_{j+i}$, $j = 0, \cdots, K + M$) and `convolve(x,y)` gives convolution $z_j = \sum_{i=\max(j-M,0)}^{\min(j,K)} x_i y_{j-i}$, $j = 0, \cdots, \max(K, M)$.

More Numpy Operations (cont)

- **Basic Statistics**: The functions `cov`, `mean`, `median`, `std`, `var`:
  - `mean(X)`: When X is simulated by data $x_1, \cdots, x_n$, then $\mathbb{E}[X] \approx \frac{x_1 + \cdots + x_n}{n} = \overline{X}$.
  - `median(X)`: Find the median of data X.
  - `var(X)` (and `std(X)` = $\sqrt{\texttt{var}(X)}$): By default, it is the population variance (and standard deviation)

$$\text{Var}[X] = \mathbb{E}[|X - \mathbb{E}[X]|^2] \approx \frac{(x_1 - \overline{X})^2 + \cdots + (x_n - \overline{X})^2}{n}.$$

    Note that for sample variance (and sample population), the n needs to be changed to $n - 1$ (set `ddof=1`).
  - `cov(X)`: Compute the covariance matrix of data in X based on the mathematical formulation:

$$Cov[X] = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^H].$$

Example: Column/Row Subtraction

Write down the Python command to subtract the each column of a matrix `A` by the mean of the data of each column vector.

Example: Column/Row Subtraction

Consider the 2-D array `M`:

```
M=np.array([[6,9,12,4,3,0],[4,4,15,2,1,1],
            [2,1,18,-5,8,2],[-6,-4,21,1,-5,2]])
```

With the help of Python/Numpy array functions,

(a) find the sum, mean, min, max, var, std of `M`;

(b) find the sum, mean, min, max, var, std of `M` column-wise;

(c) find the sum, mean, min, max, var, std of `M` row-wise;

(d) write-down the instruction to determine if all of the elements in `M` are positive.

(e) write-down the instruction to find if `M` has the value 2.

Outline

Linear Algebra Functions

I am classifying those **functions** which **related** to the **solution of the linear system** $A\boldsymbol{x} = \boldsymbol{b}$ below as "linear algebra functions". It may not be a standard terminology but reflects the purpose of the functions mentioned in this subsection.

$$AX = B \tag{1}$$

where A is an $m \times n$ matrix, X is an $n \times k$ matrix and B is an $m \times k$ matrix. X is unknown whereas A and B need to be given.

Linear Algebra Functions (cont)

According `https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html`, Scipy is normally built using the optimised LAPACK and BLAS libraries and it has very fast linear algebra capabilities and contains all the functions in `numpy.linalg`. Therefore, we will be using Scipy instead of Numpy if we want to solve linear algebra problems such as $A\boldsymbol{x} = \boldsymbol{b}$:

```
from scipy import linalg
```

Linear Algebra

Many of the linear algebra routines in `np.linalg` are able to compute results for several matrices at once, if they are **stacked** into the same array. This is indicated in the documentation via input parameter specifications such as `A: (..., M, M) array_like`. This means that if for instance given an input array `A.shape == (N, M, M)`, it is interpreted as a "stack" of `N` matrices, each of size `M`-by-`M`.

Linear Algebra (cont)

Similar specification applies to return values, for instance the determinant has `det : (...)` and will in this case return an array of shape `det(a).shape == (N,)`. This generalises to linear algebra operations on higher-dimensional arrays: the last 1 or 2 dimensions of a multidimensional array are interpreted as vectors or matrices, as appropriate for each operation.

Linear Algebra (cont)

The following is a list of linear algebra functions mostly common to Numpy.linalg and Scipy.linalg (so not everything from `https://docs.scipy.org/doc/scipy/reference/linalg.html` are included)

- Matrix operations and “metrics”
- Decomposition functions
- Matrix eigenvalues
- Linear algebra solvers and inverses
- Matrix functions

Linear Algebra (cont)

Matrix operations and “metrics”:

- `np.linalg.cond(x[, p])`: Compute the condition number of a matrix.
- `np.linalg.det(a)`: Compute the determinant of an array.
- `np.linalg.matrix_rank(M[, tol, hermitian])`: Return matrix rank of array using SVD method.
- `np.linalg.slogdet(a)`: Compute the sign and (natural) logarithm of the determinant of an array.
- `np.trace(a[, offset, axis1, axis2, dtype, out])`: Return the sum along diagonals of the array.

Linear Algebra (cont)

Matrix operations related to “metrics” (cont):

Scipy.linalg: inv(a), solve(a, b), solve_banded, solveh_banded, solve_circulant, solve_triangular, solve_toeplitz, det, norm, lstsq, pinv, pinv2, pinvh, kron(a, b), khatri_rao(a, b), tril(m[, k]), triu(m[, k]), orthogonal_procrustes(A, B), matrix_balance, subspace_angles(A, B)

Linear Algebra (cont)

Decomposition functions:

- `np.linalg.cholesky(a)`: Cholesky decomposition.
- `np.linalg.qr(a[, mode])`: Compute the qr factorization of a matrix.
- `np.linalg.svd(a[, full_matrices, compute_uv])`: Singular value decomposition (SVD).

Linear Algebra (cont)

Scipy.linalg: lu (PLU decomposition), lu_factor, lu_solve, svd, svdvals, diagsvd,
orth, null_space, ldl (Bunch-Kaufman factorization of a symmetric/ hermitian matrix),
cholesky, cholesky_banded, cho_factor, cho_solve, cho_solve_banded,
polar, qr, qr_multiply, qr_update, qr_delete, qr_insert, rq, qz, ordqz, schur (the Schur form is especially useful in calculating **functions of matrices**), rsf2csf (converts from a real Schur form to a complex Schur form), hessenberg, cdf2rdf, cossin,
interpolative

Linear Algebra (cont)

Matrix Eigenvalues:

- `np.linalg.eig(a)`: Compute the eigenvalues and right eigenvectors of a square array.
- `np.linalg.eigh(a[, UPLO])`: Return the eigenvalues and eigenvectors of a Hermitian or symmetric matrix.
- `np.linalg.eigvals(a)`: Compute the eigenvalues of a general matrix.
- `np.linalg.eigvalsh(a[, UPLO])`: Compute the eigenvalues of a Hermitian or real symmetric matrix.

Scipy.linalg: eig, eigvals, eigh (complex Hermitian), eigvalsh, eig_banded, eigvals_banded, eigh_tridiagonal. eigvalsh_tridiagonal.

Matrix Functions

If a function f has a Taylor series of the form

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!} x^k,$$

the corresponding **matrix function** is defined as

$$f(A) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!} A^k.$$

The usual exponential, logarithm, trigonometric and hyperbolic functions can be generalised to become matrix functions.

Matrix Functions (cont)

- expm(A) (Pade approximation), logm(A[, disp])
- cosm(A), sinm(A), tanm(A)
- coshm(A), sinhm(A), tanhm(A)
- signm(A[, disp]): Matrix sign function.
- sqrtm(A[, disp, blocksize])
- funm(A, func[, disp]): Evaluate a matrix function specified by a callable.
- expm_frechet(A, E[, method, compute_expm, ...]): Frechet derivative of the matrix exponential of A in the direction E.
- expm_cond(A[, check_finite])
- fractional_matrix_power(A, t)

Linear Algebra (cont)

Linear algebra solvers and inverses:

- `np.linalg.solve(A, B)`: Solve $AX = B$.
- `np.linalg.lstsq(A, B[, rcond])`: Return the least-squares solution X to $\min_X \|AX = B\|_2$.
- `np.linalg.inv(A)`: Compute the (multiplicative) inverse of a matrix A. It is the same as solving $AX = B$ with B being the identity matrix.
- `np.linalg.pinv(a[, rcond])`: Compute the (Moore-Penrose) pseudo-inverse of a matrix.
- `np.linalg.tensorsolve(a, b[, axes])`: Solve the tensor equation.
- `np.linalg.tensorinv(a[, ind])`: Compute the "inverse" of an N-dimensional array.

Linear Algebra (cont)

Linear algebra solvers and inverses in `scipy.linalg`:

- solve_sylvester(a, b, q): solves for X to the Sylvester equation $AX + XB = Q$.
- solve_continuous_are(a, b, q, r[, e, s, ...]): Solves the the continuous-time algebraic Riccati equation.
- solve_discrete_are(a, b, q, r[, e, s, balanced]) Solves the discrete-time algebraic Riccati equation.
- solve_continuous_lyapunov(a, q): Solves the continuous Lyapunov equation $AX + XA^H = Q$.
- solve_discrete_lyapunov(a, q[, method]) Solves the discrete Lyapunov equation $AXA^H - X + Q = 0$

Linear Algebra (cont)

Special matrices (mentioned in Array Construction) in science and engineering's numerical problems:

- `linalg.block_diag(*arrs)`: Create a block diagonal matrix from the provided arrays.
- `linalg.circulant(c)`: Construct a circulant matrix.
- `linalg.companion(a)`: Create a companion matrix.
- `linalg.dft(n[, scale])`: Discrete Fourier transform matrix.

Linear Algebra (cont)

Special matrices (cont):

- `linalg.hadamard(n[, dtype])`: Construct a Hadamard matrix.
- `linalg.hankel(c[, r])`: Construct a Hankel matrix.
- `linalg.helmert(n[, full])`: Create a Helmert matrix of order `n`.
- `linalg.hilbert(n)`: Create a Hilbert matrix of order `n`.
- `linalg.invhilbert(n[, exact])`: Compute the inverse of the Hilbert matrix of order `n`.

Linear Algebra (cont)

Special matrices (cont):

- `linalg.leslie(f, s)`: Create a Leslie matrix.
- `linalg.pascal(n[, kind, exact])`: Returns the `n`×`n` Pascal matrix.
- `linalg.invpascal(n[, kind, exact])`: Returns the inverse of the `n`×`n` Pascal matrix.
- `linalg.toeplitz(c[, r])`: Construct a Toeplitz matrix.
- `linalg.tri(N[, M, k, dtype])`: Construct (N, M) matrix filled with ones at and below the `k`-th diagonal.
- `numpy.vander`: Generate a Van der Monde matrix.

Example: Sept 2014 FE, Q4(a)

Given the linear system

$$\begin{aligned} 3x_1 + 7x_2 - 2x_3 + 3x_4 - x_5 &= 37 \\ 4x_1 + 3x_5 &= 40 \\ 5x_3 - 4x_4 + x_5 &= 12 \\ 2x_1 + 9x_3 + 4x_4 + 3x_5 &= 14 \\ 5x_4 + 8x_5 &= 20 \end{aligned}$$

Write a MATLAB command and a Python command to solve the linear system. (8 marks)

Example: SPM Forecast Question

It is given that matrix M is a 2×2 matrix such that

$$M\begin{pmatrix} -2 & 1 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Use Python to

1. find matrix M;
2. calculate the value of x and of y for the following simultaneous linear equations

$$-2x + y = 10,$$
$$x + 3y = 9.$$

UEEP2083 Computational Physics Final Exam Sept 2015, Q1(c)(ii)

- Compute the following system of linear equations using MATLAB. Develop the commands in MATLAB.

$$y = 3x - 2$$
$$y = -x - 6$$

(3 marks)

- Compute the system of linear equations using Python/Numpy

Example: Oct 2018 FE, Q2(c)

Given a Python function `myst` as follows:

```
1  def myst(a):
2      b = a.copy()
3      n = a.size
4      for i in range(int(n/2)):
5          b[i],b[n-i-1] = b[n-i-1],b[i]
6      return b
```

After you have imported the function `myst` and run `myst(np.array([1,2,3, 5,7,9]))`, what return value will you obtain? Explain in *one sentence* what the function `myst` does? (5 marks)

Example: Sept 2012 FE, Q5(b)

Given the system of linear equations,

$$\begin{aligned} 2x_1 - 3x_2 + 2x_3 + 5x_4 &= 11 \\ x_1 + 2x_2 + x_3 - 3x_4 &= 3.5 \\ 3x_1 + x_3 - x_4 &= 7.5 \\ 2x_1 + 2x_2 + x_3 - 4x_4 &= 4.5 \end{aligned}$$

Write a Python program to perform the following tasks.

- Check if the system has a solution.
- If so, find the values of x_1, x_2, x_3, and x_4 using Gaussian Elimination Method.
- Otherwise, return an error message “No solution”.

Example: Sept 2013 FE, Q3(b)

Given the system of linear equations,

$$
\begin{aligned}
x_1 - 3x_2 + 2x_3 + 2x_4 &= 8 \\
2x_1 - x_2 + 3x_3 - 3x_4 &= -2 \\
3x_1 - 2x_4 &= -3 \\
2x_1 + x_2 - 5x_3 + 4x_4 &= 5
\end{aligned}
$$

Write a Python program to perform the following tasks.

- Check if the system has a solution.
- If so, find the values of x_1, x_2, x_3, and x_4 using Gaussian Elimination Method.
- Otherwise, return an error message “No solution”.

Outline

Application of Arrays

- Data Processing & Machine Learning: DataFrame (a 2-D table) can be regarded as a list of "1-D" arrays of integers, numbers, strings, etc. ✓
- Financial Time-Series: Moving-average ✓
- Financial Option Pricing: Parabolic PDE
- Structural Mechanics: Array is used to represent force (vector) and stress (tensor)
- Wave Modelling (Hyperbolic PDE) and DSP: `https://pysdr.org/index.html`
- Chemistry: Eigenvalues correspond to "stable" molecular structures in a molecule.
  - Material design ⇒ Green materials
  - Medicine design

DataFrame Example

DataFrame = Labelled 1-D Arrays

In a university environment, the simplest numerical data is the results of each student. Let look at the data from `https://sites.google.com/site/liewhowhui/scicomp/notes/result2018.csv`

```
PROGRAMME,Name,T1Q1a,T1Q1b,T1Q1c1,T1Q1c2,T1Q1d,T1Q2a,T1Q2b1,T1Q2b2,T2Q1a1,T2Q1a2,T2Q1a3,T2Q1b,T2Q
AM,Student1,0,0,0.3,0,1,6.2,0.6,2.5,0.5,0,0.5,2.5,0,4,1,1.4,0.6,12,19,9,20,,5
AM,Student2,0,2,0.9,1,1.6,6.3,0,2.4,0.5,1.7,2.9,1.4,0,3,1,1.4,0.6,13,14,11,17,9,
AM,Student3,1.3,1.5,0.3,0,1.6,6.5,0.6,2.5,0.5,0.5,0.5,4.6,0,1.5,1,1.4,0.6,16,14.5,17,15,6,
AM,Student4,1.8,0.6,0.9,0,2,5.5,0.1,2.4,0.5,2,3,5.4,0.6,3.5,1,1.4,0.6,16,11.5,16,20,,7
AM,Student5,1.8,2,1,2.2,2,6.1,0.5,1.7,0.5,2,3,4.8,0,4,1,1.4,0.6,20,19.5,16,20,13,
AM,Student6,1,2,0.6,0.8,1,3,0.5,1,0.5,2,3,1.3,1,1,1,1.4,0.6,15,7,11,17,7,
AM,Student7,1.7,0.5,1,2.2,1.8,5.5,1,2.5,0,1.2,1.7,5.7,0,3,1,1.4,0.6,20,8.5,6,15,,13
AM,Student8,2,2,1,2.5,2,5.7,1,2.5,0.5,2,3,6.5,0,4,1,1.4,0.6,20,20,19.5,20,,20
AM,Student9,2,2,0.9,2.1,1,4.2,1,2.1,0.5,2,3,6.4,0.7,4,1,1.4,0.6,18,19.5,9,18,,14.5
AM,StudentA,1.4,2,0.9,3,2,6.5,1,2,0.5,2,3,4.2,0.8,3,1,1.4,0.6,18,18,19,15,12,
AM,StudentB,1.3,0.9,1,0,1.1,6.5,1,1.5,0.5,2,3,5.8,0.3,1.6,1,1.4,0.6,14,13.5,9,9.5,,13
AM,StudentC,1.7,2,0.9,0.4,1.4,6.3,0.6,2.1,0.5,2,3,6.3,0.2,4,1,1.4,0.6,20,10.5,7,18,19,
AM,StudentD,2,1.5,0.9,2.4,2,6.5,1,0.5,0.5,1.9,3,3,0,3,1,1.4,0.6,18,14.5,15,20,1,
AM,StudentE,1,2,0.9,2,2,6.5,1,1.5,0.5,2,3,6.5,0.6,4,1,1.4,0.6,16,19.5,19.5,20,,18
AM,StudentF,1.4,2,0.4,1,1,5.9,1,2,0.5,1.7,3,3.3,0,4,1,1.4,0.5,17,15.5,16,15,13,
```

DataFrame (cont)

Python's pandas module which handles DataFrame. the education ministry. Let us see how it works.

```
import pandas as pd
data = pd.read_csv("result2018.csv")
print(data.dtypes)   # OK similar to Numpy
dv = data.as_matrix()  # Convert from dataframe to matrix
dv = dv[:,2:].astype('float')  # Take only the numbers
data['T1Q1'] = data[data.columns[2:7]].sum(axis=1)
data['T1Q2'] = data[data.columns[7:10]].sum(axis=1)
data['T2Q1'] = data[data.columns[10:14]].sum(axis=1)
data['T2Q2'] = data[data.columns[14:16]].sum(axis=1)
data['T2Q3'] = data[data.columns[16:19]].sum(axis=1)
idx = data[['T1Q1','T1Q2','T2Q1','T2Q2',
        'T2Q3']].sum(axis=1)<20.0
data[idx]['Name']
```

DataFrame (cont)

```
# Weights for T1Q1, T1Q2, T2Q1, T2Q2, T2Q3 =10,10,12,5,3
# True Weights for Final Q1, ..., Q6 = 20 marks x 0.6
# T1Q1=10%, FEQ1=12%, T1Q2=10%, FEQ2=12%,
# T2Q1=12%, FEQ3=12%, T2Q2=5%, FEQ3=12%
CO1 = (data['T1Q1'] + 0.6*data['FEQ1'])/22*100
CO2 = (data['T1Q2'] + 0.6*data['FEQ2'])/22*100
CO3 = (data['T2Q1'] + 0.6*data['FEQ3'])/24*100
CO4 = (data['T2Q2'] + 0.6*data['FEQ4'])/17*100
CO5 = (data['T2Q3'] + 0.6*data[['FEQ5',
        'FEQ6']].sum(axis=1))/15*100
```

Financial Time-Series

Stock price data of Telekom Malaysia from Yahoo!Finance

```
Date,Open,High,Low,Close,Volume,Adj Close
2016-12-30,6.06,6.09,5.81,5.95,5842300,5.95
2016-12-29,6.05,6.12,5.98,6.06,6777900,6.06
2016-12-28,5.96,6.06,5.96,6.03,2503700,6.03
2016-12-27,5.93,5.99,5.92,5.99,922400,5.99
2016-12-26,5.95,5.95,5.95,5.95,000,5.95
2016-12-23,5.91,5.97,5.91,5.95,838100,5.95
...
2016-12-09,6.11,6.11,6.01,6.03,1573000,6.03
2016-12-08,6.16,6.20,6.11,6.11,3189800,6.11
2016-12-07,6.13,6.15,6.09,6.11,4564800,6.11
2016-12-06,6.12,6.15,6.09,6.12,2976600,6.12
2016-12-05,6.15,6.19,6.13,6.14,3303800,6.14
2016-12-02,6.15,6.22,6.09,6.13,2134000,6.13
2016-12-01,6.17,6.24,6.14,6.15,6188400,6.15
```

Financial Time-Series (cont)

By using the closing price, write a Python program to calculate

- the price difference between the next day and today for December 2016;
- the three-day (moving) average for December 2016. [Useful reference: `https://rosettacode.org/wiki/Averages/Simple_moving_average`]

Financial Time-Series (cont)

```
import numpy as np
dclose = np.array([6.15, 6.13, 6.14, 6.12, 6.11, 6.11, 6.03,
                   6.03, 5.96, 5.95, 5.9, 5.95, 5.95, 5.98,
                   5.95, 5.95, 5.95, 5.95, 5.99, 6.03, 6.06, 5.95])
price_diff = np.zeros(dclose.size-1)
moving3 = np.zeros(dclose.size-2)
for today in range(price_diff.size):
    next_day = today + 1
    price_diff[today] = dclose[next_day]-dclose[today]
for today in range(moving3.size):
    next_day = today + 1
    next_2day = today + 2
    moving3[today] = (dclose[today]+dclose[next_day]+\
                      dclose[next_2day])/3
print("Price difference between next day and today for December 2016: ")
print(price_diff)
print("3-D moving average for December 2016:", moving3)
```

Example: Oct 2018 FE, Q2(b)

The following function from a program script is used to generate a moving sequence for a 1-D array

```
1  def rolling(a, window=4):
2      n = a.size
3      newarray = np.zeros((n-window+1,window))
4      for i in range(n-window+1):
5          newarray[i,:] = a[i:i+window]
6      return newarray
```

When the one-dimensional array is $\boldsymbol{x} = [x_1, x_2, x_3, x_4, \cdots, x_n]$, the return moving sequence of `rolling(`$\boldsymbol{x}$`)` is

$$[[x_1, x_2, x_3, x_4],\ [x_2, x_3, x_4, x_5],\ \cdots, [x_{n-3}, x_{n-2}, x_{n-1}, x_n]].$$

Example: Oct 2018 FE, Q2(b) cont

1. If `a = np.array([0.95, 0.87, 0.87, 0.98, 1.04, 1.08])`, write down the output of `rolling(a)`. (3 marks)
2. Define a Python function `moving_average` to calculate moving average of $\boldsymbol{x}$ with a window of 4 which returns the following array:

$$[\frac{x_1 + x_2 + x_3 + x_4}{4}, \frac{x_2 + x_3 + x_4 + x_5}{4}, \cdots, \frac{x_{n-3} + x_{n-2} + x_{n-1} + x_n}{4}]$$

   based on the moving sequence `rolling(`$\boldsymbol{x}$`)`. Write down the output of the Python command `print(moving_average(a))` where `a` is given in part (i). (5 marks)
3. Explain how to calculate moving variance of `a` in part (i) with a window of 4. (2 marks)

Physics: 1-D Wave Equation

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} - \alpha^2 \dfrac{\partial^2 u}{\partial x^2} = 0, & 0 < x < L,\ 0 < t < T, \\ u(0,t) = u(1,t) = 0, & 0 < t < T,\ \text{(BC)} \\ u(x,0) = f(x),\ u_t(x,0) = g(x), & 0 \le x \le 1.\ \text{(IC)} \end{cases}$$

The Numerical Method "CTCS Explicit Scheme":

1. GENERATE A GRID: Let $u_{i,j} = u(ih, jk) = u(x_i, t_j)$, $i = 0, \cdots, M$, $j = 0, \cdots, N$.
2. NUMERICAL DIFFERENCE APPROXIMATION:

$$u_{i,j+1} = ru_{i+1,j} + 2(1-r)u_{i,j} + ru_{i-1,j} - u_{i,j-1},\ r = (\frac{\alpha k}{h})^2. \quad (2)$$

Physics: 1-D Wave Equation (cont)

3. The homogeneous boundary conditions (BC):

$$u_{0,j} = u_{M,j} = 0, \quad j = 1, \cdots, N.$$

4. The initial conditions (IC):

$$u_{i,0} = u(x_i, 0) = f(x_i), \quad i = 0, 1, \cdots, M, \tag{3}$$

$$u_{i,1} = (1 - r)f(x_i) + \frac{r}{2}(f(x_{i+1}) + f(x_{i-1})) + kg(x_i). \tag{4}$$

5. SOLVE THE ITERATION (2) FOR A GIVEN INITIAL CONDITION.

Example: 1-D Wave Equation

Consider $L = 1$, $\alpha = 1$, $T = 1$, $f(x) = 0$, $g(x) = \sin \pi x$, $h = k = 0.2$. Calculate $u_{i,j}$ for $i = 1, ..., 4$, $j = 0, ..., 5$.

- Using Excel
- Using Python

Note that

$$r = (\frac{\alpha k}{h})^2 = (\frac{1 \cdot 0.2}{0.2})^2 = 1,$$

- $x_i = ih = i \times 0.2$, $i = 0, 1, ..., 5$.
- BC: $u_{0,j} = u_{5,j} = 0$
- IC: $u_{i,0} = f(x_i) = 0$

Example: 1-D Wave Equation (cont)

$$
\begin{aligned}
u_{i,1} &= (1-r)f(x_i) + \frac{r}{2}(f(x_{i+1}) + f(x_{i+1})) + kg(x_i) \\
&= (1-1) \times 0 + \frac{1}{2}(0+0) + 0.2 \times \sin(\pi \times 0.2 \times i) \\
&= 0.2 \sin(0.2\pi \times i)
\end{aligned}
$$

For $j = 2, 3, 4, 5$:

$$
\begin{aligned}
u_{i,j+1} &= ru_{i+1,j} + 2(1-r)u_{i,j} + ru_{i-1,j} - u_{i,j-1} \\
&= u_{i+1,j} + u_{i-1,j} - u_{i,j-1}
\end{aligned}
$$

Example: 1-D Wave Equation (cont)

Using Excel:

	x_i	0	0.2	0.4	0.6	0.8	1
	Index “i”	0	1	2	3	4	5
	Index “j”						
“t”=0	0	0	0	0	0	0	0
“t”=0.2	1	0	0.11755705	0.190211303	0.190211303	0.11755705	0
“t”=0.4	2	0	0.190211303	0.307768354	0.307768354	0.190211303	0
“t”=0.6	3	0	0.190211303	0.307768354	0.307768354	0.190211303	0
“t”=0.8	4	0	0.11755705	0.190211303	0.190211303	0.11755705	0
“t”=1	5	0	0	0	0	0	0

Using Python: Exercise / Practical Lab

Example: Fourier Series and Wave

Fourier series for periodic signals of period T:

$$signal(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty}(a_i \cos\frac{2\pi t}{T} + b_i \sin\frac{2\pi t}{T})$$

```
1  # https://stackoverflow.com/questions/8299303/generating-sine-wave-sound-
2  # https://stackoverflow.com/questions/31603555/unknown-pcm-cards-pcm-rear
3  # Unknown PCM cards are removed by commenting out relevant lines in /usr/
4  # https://github.com/Uberi/speech_recognition/issues/100
5  import pyaudio
6  import numpy as np
7
8  p = pyaudio.PyAudio()
9
0  volume = 0.5     # range [0.0, 1.0]
1  fs = 44100       # sampling rate, Hz, must be integer
2  duration = 10.0  # in seconds, may be float
3  f = 440.0        # sine frequency, Hz, may be float
```

Example: Fourier Series and Wave

```
1
2  # generate samples, note conversion to float32 array
3  samples = (np.sin(2*np.pi*np.arange(fs*duration)*f/fs))
4  samples = samples.astype(np.float32)  # matching the audio format
5
6  # for paFloat32 sample values must be in range [-1.0, 1.0]
7  stream = p.open(format=pyaudio.paFloat32,
8                  channels=1,
9                  rate=fs,
0                  output=True)
1
2  # play. May repeat with different volume values (if done interactively)
3  stream.write(volume*samples)
4
5  stream.stop_stream()
6  stream.close()
7
8  p.terminate()
```

- https://dspillustrations.com/pages/posts/misc/the-sound-of-harmonics-approximating-instrument-sounds-with-fourier-series.html
- https://sandipanweb.wordpress.com/2020/06/21/fourier-series-and-differential-equations-with-some-applications/