<Linear Algebra 2023 Fall HW3>

Cosine Transform and its Application

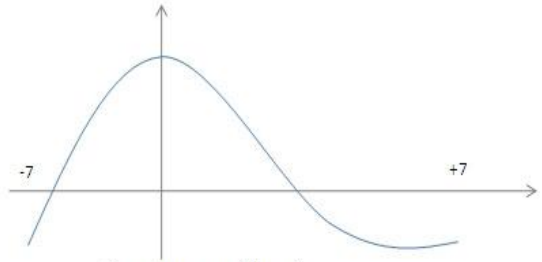
TA 林冠廷
linear-algebra-lee-2023@googlegroups.com

Outline

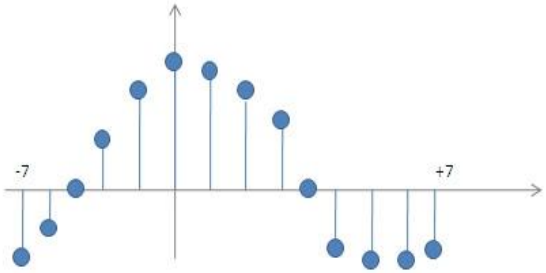
- What is signal
- Fourier Transform
- Cosine Transform
- Homework:
  - Problem 1 - Filtering
  - Problem 2 - JPEG compression
- Rules

What is signal

- 傳遞有關一些現象的行為或屬性的資訊的函數
- Example
  - f(t) : 音訊
  - f(x, y) : 圖片
- Type
  - Continuous
  - Discrete
- In this homework, we use
  - discrete signal



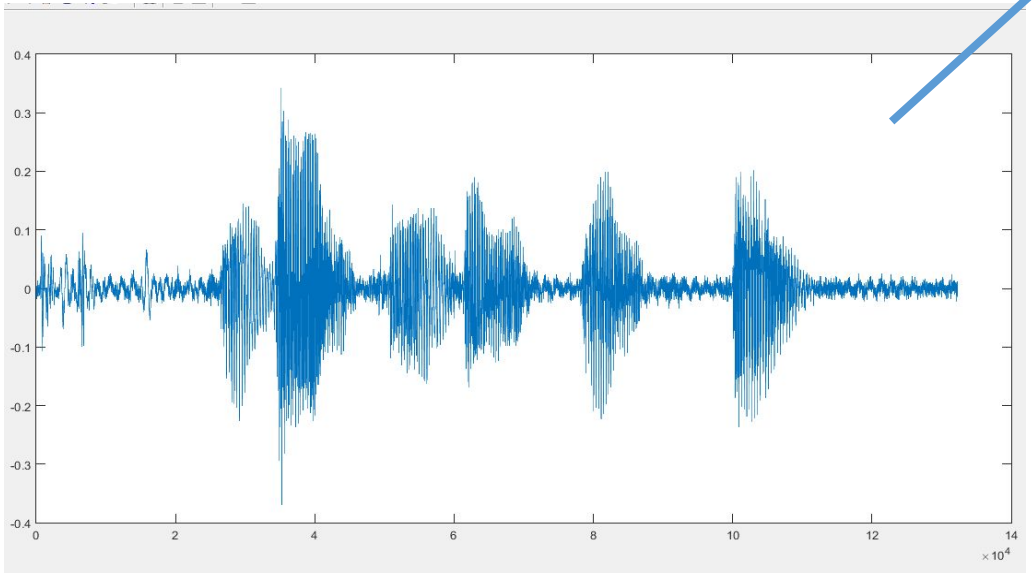
Continuous Signal
(takes values *in the set* [-7,7])



Discrete Signal
(takes values *at the integers* {-7,-6...0...6,7})

Basis

- How to use basis in signal analysis
- Given a speech signal
  - Can we find basis to describe this signal ?



$$\mathbf{x} = [x_0, x_1, \dots, x_{N-1}]$$

$$\mathbf{B} = \{\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_{N-1}\}$$

$$[\mathbf{x}]_{\mathbf{B}} = [a_0, a_1, \dots, a_{N-1}] = \mathbf{a}$$

$$\mathbf{x} = \sum_{k=0}^{N-1} a_k \mathbf{b}_k$$

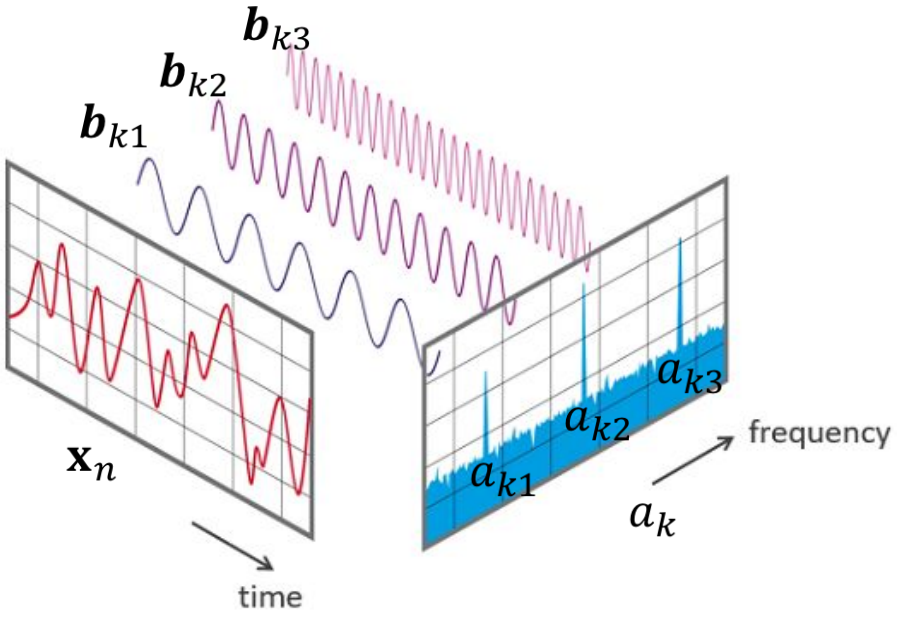
Joseph Fourier

Any periodic signal can be represented as a sum of sinusoids.

Fourier Transform

假設x主要由三個basis vector組成
→三個頻率的cosine signal組成

$$\mathbf{x} = a_{k1}\mathbf{b}_{k1} + a_{k2}\mathbf{b}_{k2} + a_{k3}\mathbf{b}_{k3}$$



x_t 時間上的強度

Fourier Transform

a_k 頻率上的強度

Cosine Transform

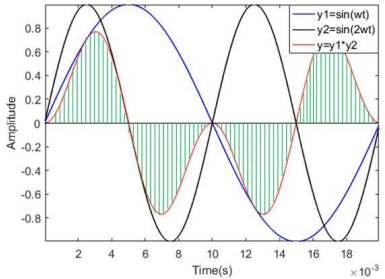
- Fourier Transform includes complex number computation
  - We use cosine transform instead
- Cosine Transform Formula
  - Given a discrete signal $x = [\, x_0, x_1, \dots, x_n, \dots, x_{N-1}]$ with N length
  - Basis Matrix:

- $\mathbf{B} = \{\mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_{N-1}\}$ 用N種basis來分解x，也就是有N種頻率，頻率用下標k表示

$$b_{n,k} = \begin{cases} \frac{1}{\sqrt{N}}, if\ k = 0 \\ \frac{\sqrt{2}}{\sqrt{N}} cos \frac{(n+0.5)k\pi}{N}, else \end{cases}$$

n: 時間上的index
k: 頻率上的index

How to get **a**



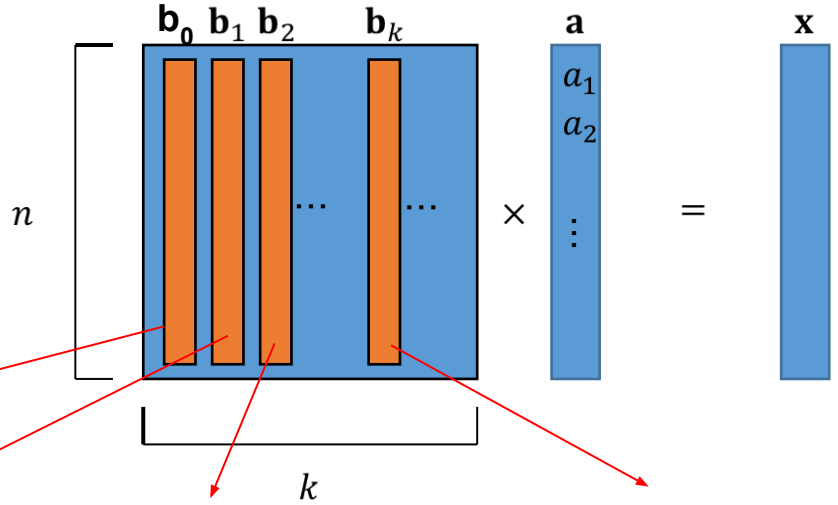
$$\mathbf{x} = \mathbf{Ba}$$ Inverse Cosine transform

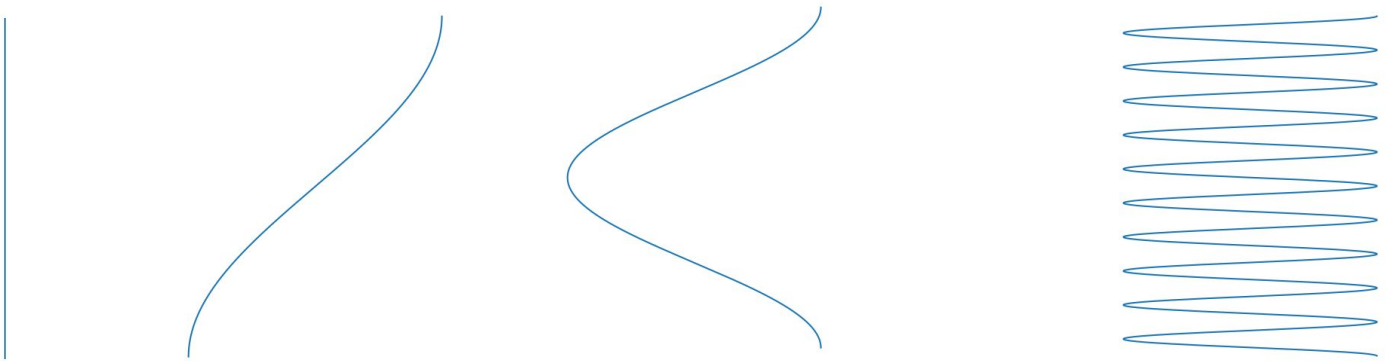
$$\mathbf{a} = \mathbf{B^{-1}x}$$ Cosine transform

Why sine/cosine waves are basis?

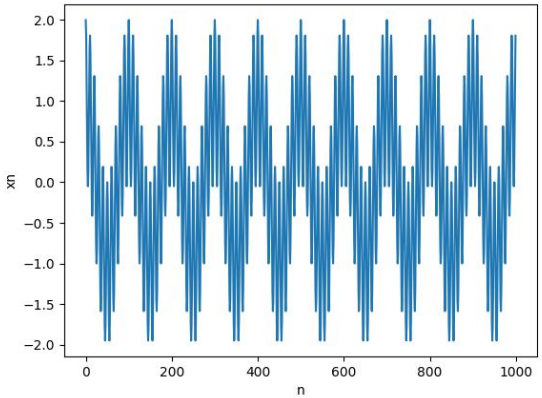
$$\langle \varphi_j, \varphi_k \rangle = \int_{-\pi}^{\pi} \cos jx \cos kx \, dx$$

$$b_{n,k} = \begin{cases} \frac{1}{\sqrt{N}}, if\ k = 0 \\ \frac{\sqrt{2}}{\sqrt{N}} cos \frac{(n+0.5)k\pi}{N}, else \end{cases}$$



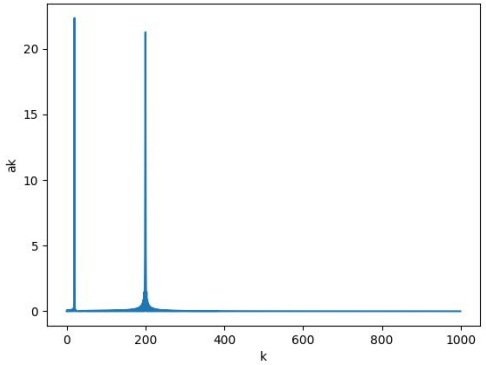


If we want to get the low frequency signal of a mixed signal…



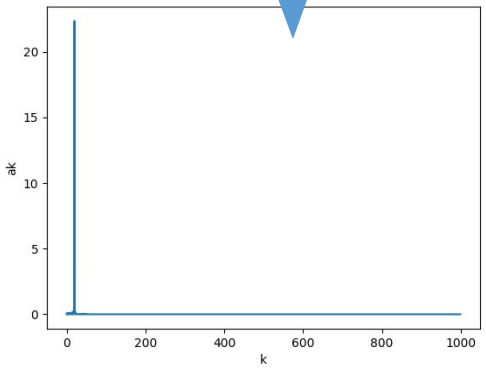
Cosine transform

$$\mathbf{a} = \mathbf{B}^{-1}\mathbf{x}$$



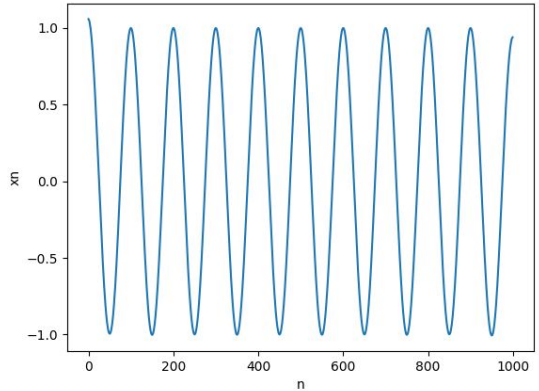
x

mask [**1** 0 0 0 0 0 0 0 0 0 … 0 0 0 0 0]



Inverse cosine transform

$$\mathbf{x}' = \mathbf{Ba}$$



Application

- **Filtering**
  - Human voice
    - Man: 85-180Hz
    - Woman: 165-255Hz
  - Remove high frequency noise from speech signal

- **Compression**
  - Mel-frequency cepstral coefficients (MFCCs)
  - JPEG

strong "energy compaction" property. In typical applications, most of the signal information tends to be concentrated in a few low-frequency components of the DCT

Homework

- **Filtering (problem 1) 3%**
  - p1.py
- **JPEG Compression (problem 2) 3%**
  - p2.py

Code & Data link Colab link

Github repository contains below files:

- p1.py
- p2.py
- example_data
  - test.txt
  - test.png
- example_output
  - freq.png
  - f1.png
  - f3.png
  - reconstructed.png

Problem 1 - Filtering

Input data

- example_data/test.txt
- Total 1000 lines, one value per line

$$x = \sum_{i=1}^{5} Cosine(2\pi f_i)$$

- $f_1 < f_2 < f_3 < f_4 < f_5$

```
1  2.0000000000000000e+00
2  1.807043722803219010e+00
3  1.301131695689425438e+00
4  6.732702563537411589e-01
5  1.595661667536837358e-01
6  -4.894348370484646882e-02
7  1.207594915133041180e-01
8  5.958100580910720145e-01
9  1.185323674418810924e+00
10 1.653344919876962527e+00
11 1.809016994374947451e+00
12 1.579530237150736927e+00
13 1.037985621796358338e+00
14 3.755301115537416079e-01
15 -1.715930046262574837e-01
16 -4.122147477075268629e-01
17 -2.731901993959511277e-01
18 1.727366797267690379e-01
19 7.347962859400198887e-01
20 1.771415470596251480e+00
21 1.309016994374947451e+00
22 1.057706881539801413e+00
23 4.963983089606727184e-01
24 -1.836837608106426378e-01
25 -7.462264748456348684e-01
26 -9.999999999999998890e-01
27 -8.718075139042609223e-01
28 -4.343502279392522092e-01
29 1.216356797892221842e-01
30 5.603271072100921568e-01
31 6.909830056250526598e-01
32 4.408924416902699206e-01
33 -1.676229719012454210e-01
34 -7.907706684766655503e-01
```

TODO

```python
if __name__ == '__main__':
    # Do not modify these 2 lines
    signal_path = sys.argv[1]
    out_directory_path = sys.argv[2]

    # TODO
    # filter original waveform to f1-only and f3-only time-domain waveform
    # f1 = ...
    # f3 = ...
```

- Goal:

Original waveform contain f1 < f2 < f3 < f4 < f5

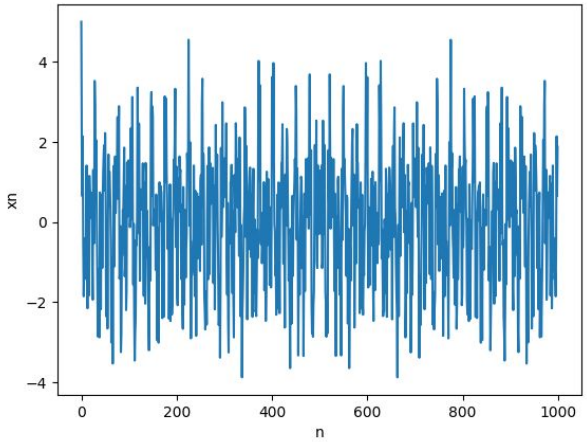
Filtering to extract only f1 and f3 component in time-domain

- You should finish all the **#TODO block** in p1.py

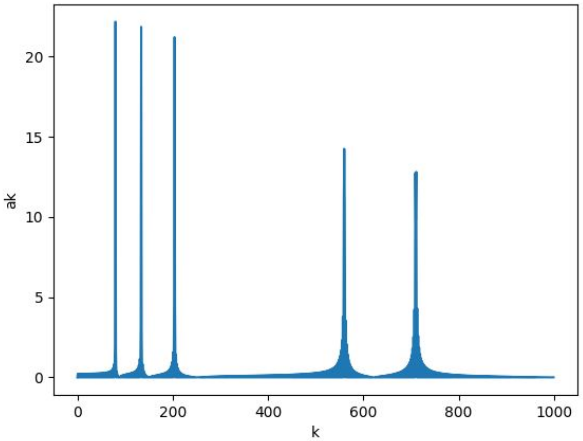
```python
def CosineTrans(x, B):
    # TODO
    # implement cosine transform
    return

def InvCosineTrans(a, B):
    # TODO
    # implement inverse cosine transform
    return
```

```python
def gen_basis(N):
    # TODO
    return
```

Example

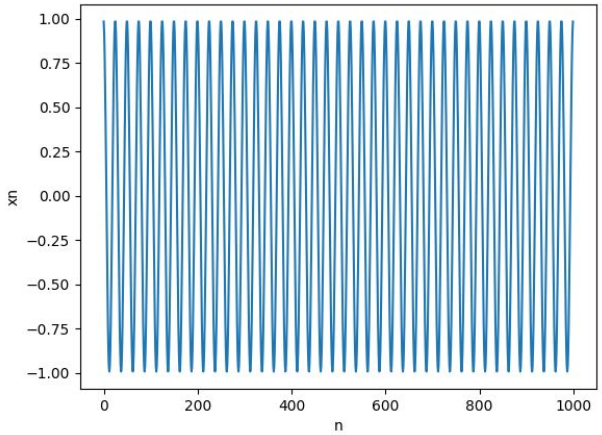


wave

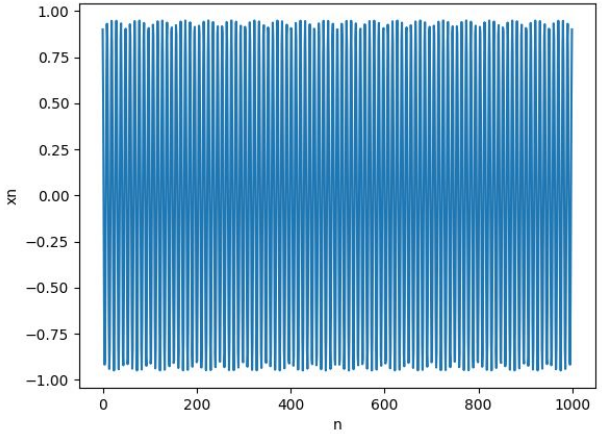


a_k

f1 and f3 in time domain



f1



f3

Run the code

- *python p1.py <input_signal_txt> <output_dir_path>*
- e.g.
  python p1.py example_data/test.txt ./

- Your code should generate 3 .png files in <output_dir_path> (e.g. ./).
  - freq.png
  - f1.png
  - f3.png

Scoring

1. (1%) Plot the figure of a (freq.png)
2. (1%) Plot the figure of f_1 (f1.png)
3. (1%) Plot the figure of f_3 (f3.png)

Problem 2 - JPEG

Extend 1D DCT to 2D DCT

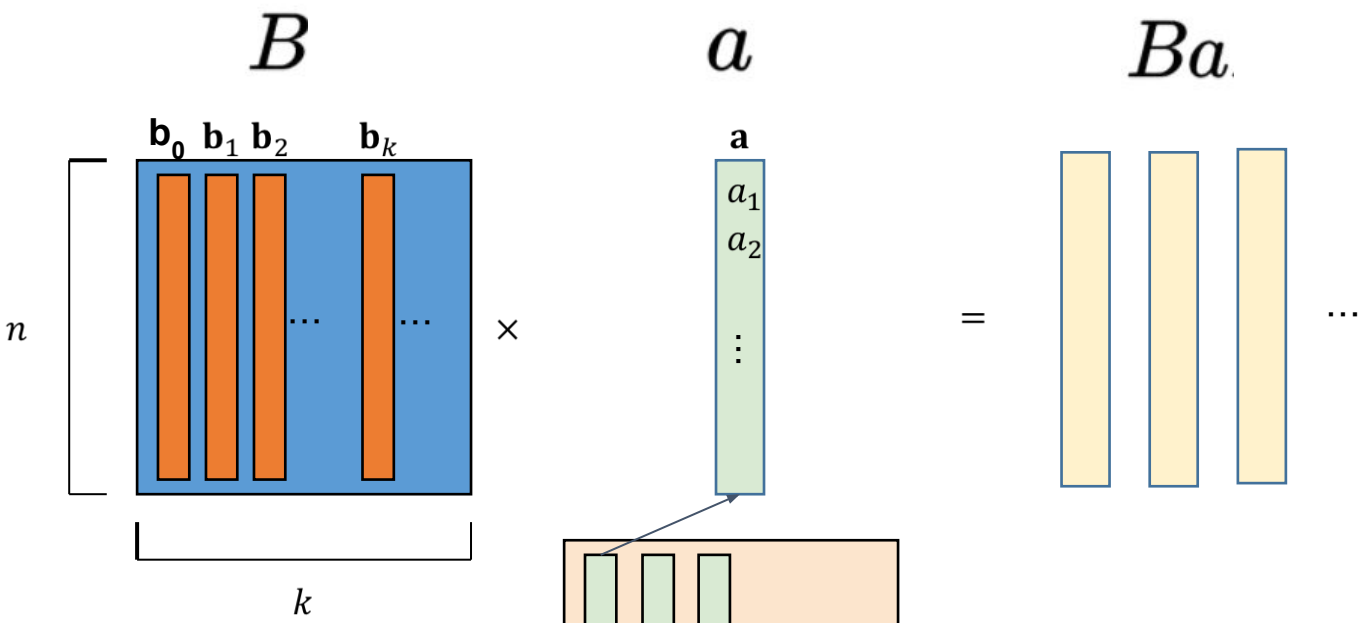
The trick is to **apply the 1D DCT to every column, *and* then also apply it to every row**, i.e.

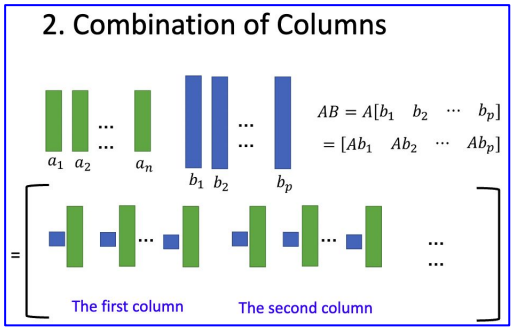
$$F(m,n) = \frac{2}{\sqrt{MN}} C(m)C(n) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)m\pi}{2M} \cos\frac{(2y+1)n\pi}{2N}$$

where $C(m) = C(n) = 1/\sqrt{2}$ for $m,n = 0$ and $C(m), C(n) = 1$ otherwise.

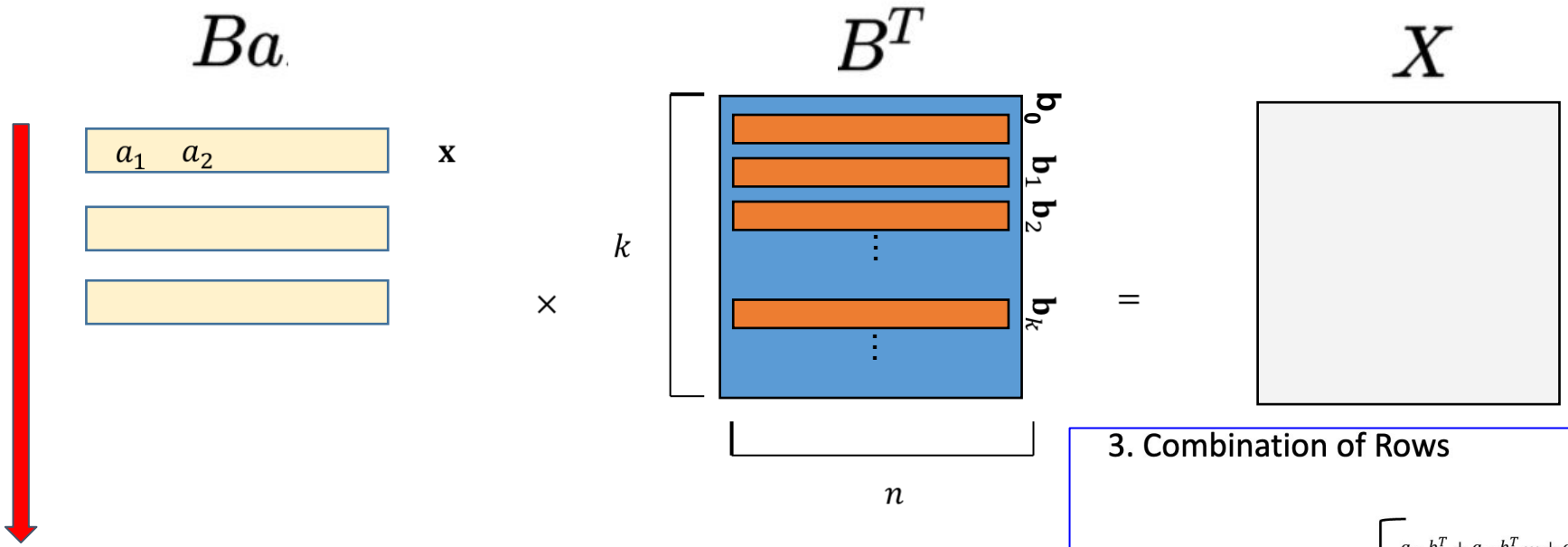
We focus on the square image here, that is M = N,

$$X = BaB^T \qquad a = ?$$



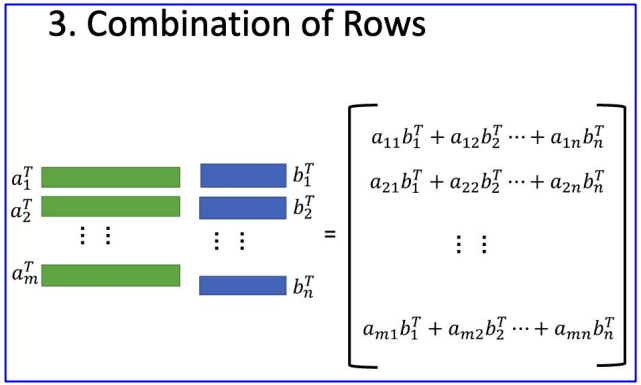


1D iDCT for every columns
each column vector in Ba is the weighted sum of column vector in B



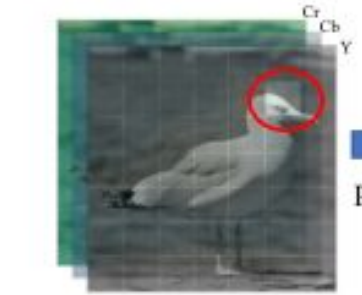
1D iDCT for every rows

each row vector of X is the weighted sum of row vector in the transpose of B



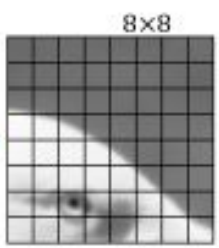
JPEG compression

Strongly recommend you to see this tutorial for detail
https://www.youtube.com/watch?v=Q2aEzeMDHMA&ab_channel=Computerphile



Image

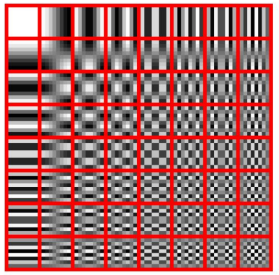
Partitioning



DCT

409.2	17.2	-13.0	1.9	6.9	4.5	7.2	6.5
4.3	-8.9	-7.7	2.3	10.0	2.7	-0.5	-0.7
2.8	17.3	-19.5	0.3	8.0	-2.5	-1.0	-0.2
5.0	25.2	17.3	-14.3	4.8	-1.8	5.0	0.3
-10.1	-25.3	2.7	0.0	-6.2	-6.1	-6.1	-2.7
5.5	-10.3	-8.1	1.4	-4.2	2.4	-0.7	1.1
4.9	-3.6	3.8	-1.9	0.1	-3.0	-0.5	-0.5
1.7	-2.2	-1.5	0.6	-0.9	0.0	0.0	0.0

Low



Division

High

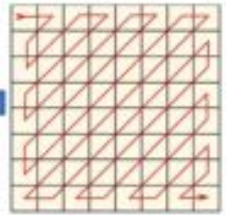
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Quantization

Reduce the influence at high frequency

-10	0	-1	0	-1	0	1	-2
0	-1	-1	0	1	0	-1	-1
0	1	-3	0	1	-1	-1	-1
0	2	2	-3	1	-1	0	0
-1	-5	0	-1	-2	-2	-2	-1
-	-2	-2	0	-1	0	-1	0
-	-2	0	-1	0	-1	-1	-1
0	-1	-1	0	-1	0	0	0

Zig-Zag Reordering



Entropy Coding
+
Huffman Coding

JPEG file

TODO

記得先寫好p1.py再寫p2，因為p2.py
會call p1.py的***gen_basis*** function

- Goal:

  Implement 2D DCT in ***CosineTrans2d*** and 2D iDCT in ***InvCosineTrans2d***

- You should finish all the **#TODO block** in p2.py

```
# input: Basis (B), 2D image array (X)
# output: coefficient in 2D (a)
def CosineTrans2d(B, X):
    # TODO
    # implement 2D DCT
    return
```

```
# input: Basis (B), coefficient in 2D (a)
# output: reconstructed image (X')
def InvCosineTrans2d(B, a):
    # TODO
    # implement 2D DCT
    return
```

Important

- If you want to know more about the implementation of JPEG compression, we encourage you to trace other code in p2.py and play with it.

- **but when submission your final code, DON’T modify other code in p2.py,** or your reconstructed image might look significantly different.

JPEG result

only use 0.15 original size of information to reconstruct

original

JPEG reconstructed

Run the code

- *python p2.py <input_image_file> <output_dir_path>*
- e.g.
  python p2.py example_data/test.png ./

- Your code should generate 1 .png files in <output_dir_path> (e.g. ./), which is your JPEG reconstructed image.
  - reconstructed.png

Scoring

1. 2D DCT (1%)
2. 2D iDCT (1%)
3. reconstructed.png (1%)

只要2D DCT, iDCT實作正確，重建後的圖片後圖片沒有明顯artifact就給過 (除非你改了#TODO以外的code上傳時又沒有改回去)，不需要每個pixel都跟測資一樣，因為矩陣運算些微誤差會導致某些pixel改變

Code Rules

- 不可以import除了numpy以外的library來實作(i.e.上傳的版本中不能import任何其他的library，最多用numpy)

Submission Rule

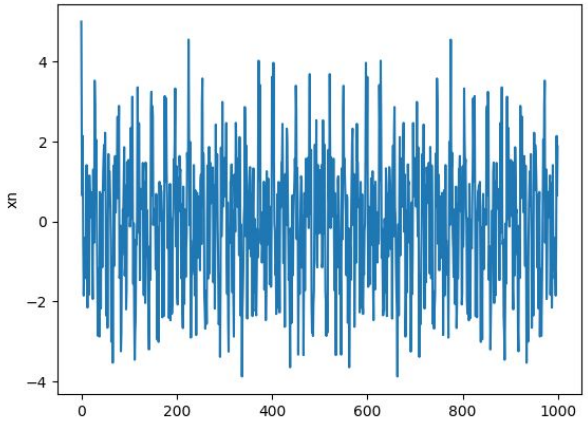
- 不要抄作業，不要交別人的答案，第一次抓到作弊該作業0分計算，第二次抓到作弊直接當掉。本作業會用moss抓抄襲。
- 上傳**p1.py**與**p2.py**檔案到 NTU COOL
- Deadline: **2023/11/17(五) 23:59** (GMT+8:00)
- 遲交每過一天：分數×0.8 (per day) 超過三天拒收
- 格式、檔案、各種奇怪的錯誤讓助教無法改作業：分數×0.7
- **請務必注不要更改input & output path成自己的電腦路徑！保持使用sys.argv的方式彈性讀取檔案**

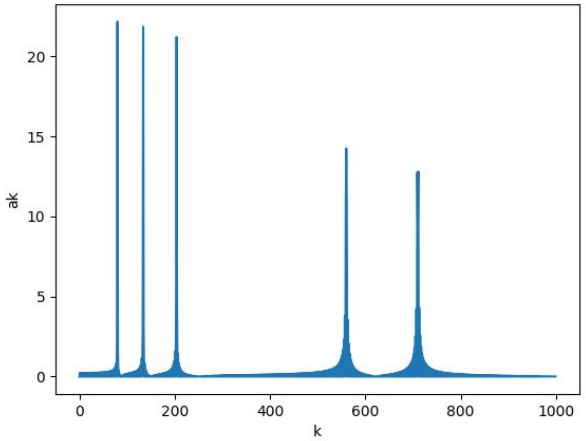
```
if __name__ == '__main__':
    # Do not modify these 2 lines
    signal_path = sys.argv[1]
    out_directory_path = sys.argv[2]
```

```
if __name__ == '__main__':
    im_path = sys.argv[1]
    output_path = sys.argv[2]
```

Some utility functions

```
 9
10 def plot_wave(x, path = './wave.png'):
11     # util function
12     plt.gcf().clear()
13     plt.plot(x)
14     plt.xlabel('n')
15     plt.ylabel('xn')
16     plt.savefig(path)
17
18 def plot_ak(a, path = './freq.png'):
19     # util function
20     plt.gcf().clear()
21
22     # Only plot the mag of a
23     a = np.abs(a)
24     plt.plot(a)
25     plt.xlabel('k')
26     plt.ylabel('ak')
27     plt.savefig(path)
```





FAQ

Q1. 找f1~f5是找a中最大的5個還是要對a取絕對值之後再找？

A. 對a取絕對值之後再找最大的5個。

Q & A

若有作業相關問題請到NTU COOL作業討論版發問

你的問題很可能也是其他同學的問題:）