1. Viết giải thuật của region growing và code

Giải thuật:

- **Bước 1:** Đọc ảnh, tìm pixel đầu tiên không thuộc về và đặt nó là (x0,y0)
- **Bước 2:** Lấy (x0, Y0) làm trung tâm, xem xét 4 pixel lân cận (x, y) của (x0, Y0), nếu (x0, Y0) đáp ứng tiêu chí tăng trưởng, hợp nhất (x, y) và (x0, Y0) trong cùng một vùng, và đẩy (x, y) lên ngăn xếp.
- **Bước 3:** Lấy một pixel từ ngăn xếp và đưa nó trở lại bước 2 là (x0, Y0).
- **Bước 4:** Nếu như ngăn xếp trống thì quay về Bước 1
- **Bước 5:** Lặp lại các bước trên cho tới khi mỗi điểm trong ảnh có ghi nhận.

Code:

```python
import numpy as np
import cv2

class Point(object):
 def __init__(self,x,y):
  self.x = x
  self.y = y

 def getX(self):
  return self.x
 def getY(self):
  return self.y

def getGrayDiff(img,currentPoint,tmpPoint):
 return abs(int(img[currentPoint.x,currentPoint.y]) - int(img[tmpPoint.x,tmpPoint.y]))

def selectConnects(p):
 if p != 0:
  connects = [Point(-1, -1), Point(0, -1), Point(1, -1), Point(1, 0), Point(1, 1), \
     Point(0, 1), Point(-1, 1), Point(-1, 0)]
 else:
  connects = [ Point(0, -1), Point(1, 0),Point(0, 1), Point(-1, 0)]
 return connects

def regionGrow(img,seeds,thresh,p = 1):
 height, weight = img.shape
 seedMark = np.zeros(img.shape)
 seedList = []
 for seed in seeds:
  seedList.append(seed)
 label = 1
 connects = selectConnects(p)
 while(len(seedList)>0):
  currentPoint = seedList.pop(0)

  seedMark[currentPoint.x,currentPoint.y] = label
  for i in range(8):
   tmpX = currentPoint.x + connects[i].x
   tmpY = currentPoint.y + connects[i].y
   if tmpX < 0 or tmpY < 0 or tmpX >= height or tmpY >= weight:
    continue
   grayDiff = getGrayDiff(img,currentPoint,Point(tmpX,tmpY))
   if grayDiff < thresh and seedMark[tmpX,tmpY] == 0:
    seedMark[tmpX,tmpY] = label
    seedList.append(Point(tmpX,tmpY))
 return seedMark


img = cv2.imread('lean.png',0)
seeds = [Point(10,10),Point(82,150),Point(20,300)]
binaryImg = regionGrow(img,seeds,10)
cv2.imshow(' ',binaryImg)
cv2.waitKey(0)
```

2. Lấy ngẫu nhiên wi. Đọc K-mean++ để xem khác biệt lấy dữ liệu như thế nào?

Sự khác nhau giữa K-mean và K-Mean ++: Sự khác biệt là ở việc Lựa chọn các trung tâm xung quanh các cụm có tham gia. K-means ++ loại bỏ nhược điểm của K, có nghĩa là nó sẽ phụ thuộc vào việc khởi tạo tâm.

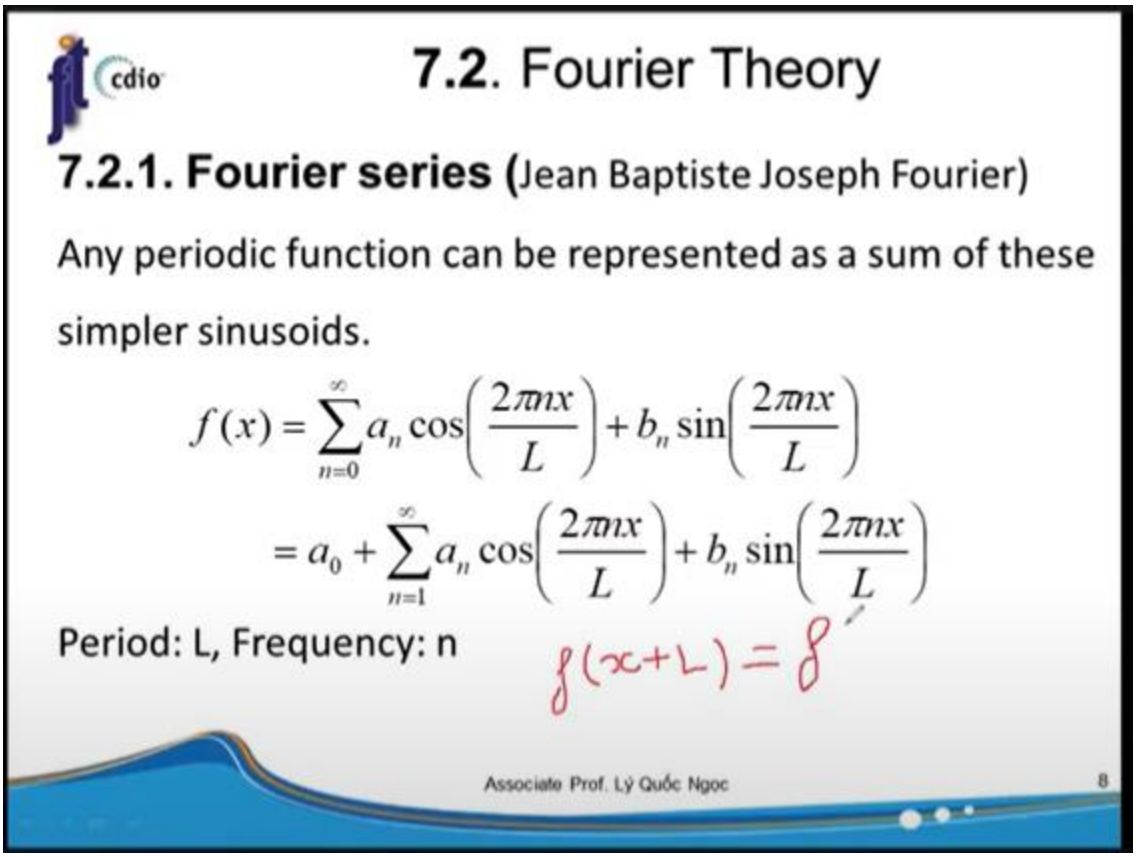
Gán (chọn) wj= il như thế nào? il lấy từ đâu ra?

Điều kiện dừng của vòng lặp: Until((w1m= w1m-1) & (w2m= w2m-1)….. (wkm= wkm-1))

Điều kiện kết nạp il với Cj*:

```
1. for l in range (1,n){
2.     minDistance= distance(i[l],w[i])
3.     minIndex=1
4.     for j* in range (1,k){
5.         if(distance(i[l],w[j*]<minDistance)){
6.             minDistance=distance(i[l],w[j*]);
7.             minIndex=j*;
8.         }
9.     }
10.      C[minIndex].add(i[l])
```

1. Thay f(x + L) vào f(x) thì hàm có thay đổi không?



$$f(x) = a_0 \sum_{n=1}^{\infty} a_n \cos\left(\frac{2\pi nx}{L}\right) + b_n \sin\left(\frac{2\pi nx}{L}\right)$$

$$f(x+L) = a_0 \sum_{n=1}^{\infty} a_n \cos\left(\frac{2\pi nx}{L} + 1\right) + b_n \sin\left(\frac{2\pi nx}{L} + 1\right)$$

$$= a_0 \sum_{n=1}^{\infty} a_n \cos\left(\frac{2\pi nx}{L}\right) + b_n \sin\left(\frac{2\pi nx}{L}\right)$$

➔ Không thay đổi

2. Tầm quan trọng của định lý Convolution theorem (dòng đầu) của phép biến đổi Fourier trong Làm trơn ảnh và phát hiện biên cạnh.

Chuyển đổi từ việc lấy từng phần tử trong h tích với các phần tử trong f qua tính trực tiếp phép biển đổi Fourier ngược của F.H:

$$(f * h)(x, y) = \xi^{-1}(F(u, v).H(u, v))$$

3. **Định lý Convolution điều 2 dùng ở đâu? Dùng trong việc lọc ảnh trong miền tần số.**

$$\xi\{f(x,y).h(x,y)\} = (F * H)(u,v)$$

4. **Độ phức tạp tính toán của Convolution Theroem: O((N*N)*(M*M)) nếu trong miền không gian, khi dùng Convolution Theorem bằng bao nhiêu?**

$O((N * N)$

5. **Biện luận Ideal Lowpass Filters. Nếu D0 càng lớn/càng bé chuyện gì sẽ xảy ra?**

$$H(u,v) = \begin{cases} 1 \; if \; D(u,v) \leq D_0 \\ 0 \; if \; D(u,v) > D_0 \end{cases}$$

$$\mathrm{D(u,v)} = \left[\left(u - \frac{M}{2}\right)^2 + \left(v - \frac{N}{2}\right)^2\right]^{\frac{1}{2}}$$

D_0 là tần số cắt. D_0 càng lớn thì sẽ khử nhiễu kém chỉ khử về 0 với những tần số cao hơn nhưng những chỗ không bị nhiễu sẽ được bảo toàn.

D_0 càng bé thì khử nhiễu tốt nhưng những chỗ không bị nhiễu sẽ bị khử theo. Nên ảnh sẽ mờ.

6. **Butterworth Lowpass Filter khi D(u,v) bằng bao nhiêu thì H tiến về 0 và ngược lại khi nào H tiến về 1?**

$$H(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_0}\right]^{2n}}$$

$$\mathrm{D(u,v)} = \left[\left(u - \frac{M}{2}\right)^2 + \left(v - \frac{N}{2}\right)^2\right]^{\frac{1}{2}}$$

Từ công thức H(u,v), ta dễ dàng thấy: $D(u,v)$ càng cao thì H(u,v) sẽ càng tiến về 0 và ngược lại D(u,v) càng thấp thì H(u,v) sẽ tiến về 1

7. Cho 1 hàm $f_p(t) = \begin{cases} 0 \text{ nếu } t < 0 \\ e^{-at} \text{ nếu } t \geq 0 \end{cases}$ với a>0. Tính $f(t), F(\omega)$:

$$F(\omega) = \frac{1}{2\pi}\int_{-\infty}^{0} f(t)e^{-i\omega t}dt = \frac{1}{2\pi}\int_{-\infty}^{0} e^{-at-i\omega t}dt = \frac{-e^{-t(a+i\omega)}}{2\pi(a+i\omega)}$$

$$= \frac{1-\lim_{t\to\infty} e^{-t(a+i\omega)}}{2\pi(a+i\omega)} = \frac{1}{2\pi(a+\ i\omega)} \ (\lim_{t\to\infty} e^{-t(a+i\omega)} = 0)$$

$$f(t) = \int_{-\infty}^{+\infty} F(\omega)e^{i\omega t}dt$$

1. So sánh độ phức tạp tính toán giữa phương pháp lọc ảnh trong miền không gian và miền tần số.

- Trong miền không gian: g = f * h, nên độ phức tạp tính toán là (n^2, m^2)

$$\Sigma\Sigma f(x-i, y-j)\ h(i,j)$$

- Trong miền tần số:
  - $fourier\{f\} = f(u,v) \rightarrow \frac{O(n^2 * n^2)}{Fast\ fourier\ transform: O(n^2 * logn)}$
  - $fourier\{h\} = h(u,v) \rightarrow O(n^2)$
  - $f(u,v) * h(u,v) \rightarrow O(n^2)$
  - $fourier_{-1}\{F.H\} \rightarrow \frac{O(n^2, n^2)}{Fast\ fourier\ transform: O(n^2 * logn)}$

⇒ Trong trường hợp m,n nhỏ thì miền không gian nhanh hơn vì độ phức tạp ít hơn, nhưng khi m, n lớn thì tần số có độ phức tạp nhỏ hơn.

Phương pháp Karhunen Loeve

- Phát biểu bài toán: Cần tìm phép biến đổi tuyến tính để:
  - Giảm số chiều không gian đặc trưng nhưng vẫn giữ được đượccasc đặc trưng chính của ảnh.
  - Cực tiểu hóa việc mất mát thông tin, đồng nghĩa với việc giữ được nhiều thông tin nhất có thể từ ảnh gốc.
- Phương pháp:
  - y = AT x

- Tìm xấp xỉ của x đẻ giảm số chiều nhưng vẫn thỏa yêu cầu cực tiểu mất mát thông tin.

  $\hat{x} = \sum_{i=0}^{m-1} y_i a_i$

  Cực tiểu $E(\|x - \hat{x}\|^2)$

- Các bước thực hiện:
  - Tính $ATB = \frac{1}{M} \sum_{i=1}^{M} I_i(x, y)$
  - Tính ảnh độ lệch $X_i = I_i - ATB$
  - Tìm M giá trị riêng của $C'_x = X^T X$ và giữ lại K giá trị lớn nhất
  - Tính mặt riêng là các vector riêng của $C_x = XX^T$
    - $F = X.F'_M$
    - $F_i(j) = \sum_{l=1}^{M} X_i(j) F'_j(l)$
  - Chiếu tập ảnh xuống không gian vector con hợp bởi K vector riêng của Cx.
    - $Y = A'^T.X$
    - $Y_i(j) = \sum_{l=1}^{N} F_j(l) X_i(l)$
    - A' chính là A nhưng chỉ gồm K vector riêng
  - Xấp xỉ ảnh bởi hình chiếu của nó xuống không gian con hợp bởi các vector riêng của Cx
    - $I_i = ATB + \sum a_{ij} F_j$
    - Vector đặc trưng của Ii là $\{a_{ij}, j = 1, 2, ..., k\}$

1. Qui nạp: Giá trị trung bình độ xám của 8 vùng Region growing

$$m(R_i^{(k)}) = \frac{1}{N(R_i^{(k)})} . \sum_{(k,l) \in R_i^{(k)}} f(k,l)$$

$$\sigma(R_i^{(k)}) = \sqrt{\frac{1}{N(R_i^{(k)})} . \sum_{(k,l) \in R_i^{(k)}} (f(k,l) - m(R_i^{(k)}))^2}$$

$$\Rightarrow m(R_i^{(k+1)}) - m(R_i^{(k)})$$

$$= [\frac{1}{N(R_i^{(k)})} + 1][f(x,y) + N(R_i^{(k)}) * m(R_i^{(k)})] - \frac{1}{N(R_i^{(k)})} . \sum_{(k,l) \in R_i^{(k)}} f(k,l)$$

$$= [\frac{1}{N(R_i^{(k)})} + 1][f(x,y) + N(R_i^{(k)}) * \frac{1}{N(R_i^{(k)})} . \sum_{(k,l) \in R_i^{(k)}} f(k,l)] - \frac{1}{N(R_i^{(k)})} . \sum_{(k,l) \in R_i^{(k)}} f(k,l)$$

$$= [\frac{1}{N(R_i{}^{(k)})} + 1][f(x,y) * \frac{1}{N(R_i{}^{(k)})} * \sum_{(k,l)\in R_i{}^{(k)}} f(k,l) + \sum_{(k,l)\in R_i{}^{(k)}} f(k,l)\ ]$$

$$- \frac{1}{N(R_i{}^{(k)})} . \sum_{(k,l)\in R_i{}^{(k)}} f(k,l)$$

$$= \frac{1}{N(R_i{}^{(k)})} * f(x,y) * \frac{1}{N(R_i{}^{(k)})} * \sum_{(k,l)\in R_i{}^{(k)}} f(k,l) + \frac{1}{N(R_i{}^{(k)})} * \sum_{(k,l)\in R_i{}^{(k)}} f(k,l) + f(x,y)$$

$$* \frac{1}{N(R_i{}^{(k)})} * \sum_{(k,l)\in R_i{}^{(k)}} f(k,l)$$

$$+ \sum_{(k,l)\in R_i{}^{(k)}} f(k,l) - \frac{1}{N(R_i{}^{(k)})} . \sum_{(k,l)\in R_i{}^{(k)}} f(k,l)$$

$$= (\frac{1}{N^2(R_i{}^{(k)})} * f(x,y) + \frac{1}{N(R_i{}^{(k)})} + f(x,y) * \frac{1}{N(R_i{}^{(k)})} + 1 - \frac{1}{N(R_i{}^{(k)})}) * \sum_{(k,l)\in R_i{}^{(k)}} f(k,l)$$

$$= [f(x,y) * \frac{1}{N(R_i{}^{(k)})} * (\frac{1}{N(R_i{}^{(k)})} + 1) + 1] * \sum_{(k,l)\in R_i{}^{(k)}} f(k,l)$$

$$= [f(x,y) * \frac{1}{N(R_i{}^{(k)})} * (\frac{1}{N(R_i{}^{(k)})} + 1) + 1] * \sum_{(k,l)\in R_i{}^{(k)}} f(k,l)$$

Qui nạp:

$$m(R_i{}^{(k+1)}) = [\frac{1}{N(R_i{}^{(k)})} + 1][f(x,y) + N(R_i{}^{(k)}) * m(R_i{}^{(k)})]$$

$$\sigma(R_i{}^{(k+1)}) = \sqrt{\frac{1}{N(R_i{}^{(k)}) + 1} + N(R_i{}^{(k)}) * \sigma^2(R_i{}^{(k)}) + \frac{N(R_i{}^{(k)})}{N(R_i{}^{(k)}) + 1}(f(x,y) - m(R_i{}^{(k)}))}$$

2. Giải thuật region growing

```
import os
import cv2
```

```
import numpy as np


def region_growing(img, x, y, delta=15):
    """
        x: relative coord (0-1)
        y: relative coord (0-1)
    """
    def is_valid_px(x, y, img):
        if x>=0 and x<img.shape[1] and y>=0 and y<img.shape[0]:
            return True
        return False
    mask = np.zeros(img.shape[:2])
    y_abs = int(y * img.shape[0])
    x_abs = int(x * img.shape[1])

    loop_points = []
    loop_points.append((x_abs, y_abs))
    px_intensity = img[y_abs, x_abs, :]
    blue = px_intensity[0]
    green = px_intensity[1]
    red = px_intensity[2]

    while len(loop_points) != 0:
        (cx, cy) = loop_points.pop()
        current_intensity = img[cy, cx, :]
        b = int(current_intensity[0])
        g = int(current_intensity[1])
        r = int(current_intensity[2])
```

```
        if abs(r-red)<delta and abs(g-green)<delta and abs(b-blue)<delta:
            mask[cy, cx] = 255
            neighbors = [
                (cx-1, cy),
                (cx+1, cy),
                (cx-1, cy-1),
                (cx, cy-1),
                (cx+1, cy-1),
                (cx-1, cy+1),
                (cx, cy+1),
                (cx+1, cy+1)
            ]
            for (nx, ny) in neighbors:
                if is_valid_px(nx, ny, img) and mask[ny, nx] == 0:
                    loop_points.append((nx, ny))
        else:
            pass
    return mask


def main(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_COLOR)
    KERNEL_WIDTH = KERNEL_HEIGHT = 5
    SIGMA_X = SIGMA_Y = 2
    img[:,:,0] = cv2.GaussianBlur(img[:,:,0], ksize=(KERNEL_WIDTH, KERNEL_HEIGHT),
sigmaX=SIGMA_X, sigmaY=SIGMA_Y)
    img[:,:,1] = cv2.GaussianBlur(img[:,:,1], ksize=(KERNEL_WIDTH, KERNEL_HEIGHT),
sigmaX=SIGMA_X, sigmaY=SIGMA_Y)
    img[:,:,2] = cv2.GaussianBlur(img[:,:,2], ksize=(KERNEL_WIDTH, KERNEL_HEIGHT),
sigmaX=SIGMA_X, sigmaY=SIGMA_Y)
    cv2.imwrite('out_blur_'+os.path.basename(img_path), img)
```

```
    x = 0.5
    y = 0.6
    delta = 20
    mask = region_growing(img=img, x=x, y=y, delta=delta)
    cv2.imwrite('out_mask.jpg', mask)

    img_color = cv2.imread(img_path, cv2.IMREAD_COLOR)
    img_color[mask>0,:] = (0, 255, 255)

    cv2.circle(img_color, (int(x*img.shape[1]), int(y*img.shape[0])), radius=5, color=(0, 0,
255), thickness=2)
    cv2.imwrite('out_seg_'+os.path.basename(img_path), img_color)

if __name__ == "__main__":
    print('Running...')
    main('jung_yun.jpg')
    print('* Follow me @ ' + "\x1b[1;%dm" % (34) + '
https://www.facebook.com/minhng.info/' + "\x1b[0m")
    print('* Join GVGroup for discussion @ ' + "\x1b[1;%dm" % (34) +
'https://www.facebook.com/groups/ip.gvgroup/' + "\x1b[0m")
    print('* Thank you ^^~')
```

3. Prototype K mean method và so sánh với K-means++(hint: Chọn vector đặc trưng (centroid) khởi thủy như thế nào ở step 1 Kmean?)

K-means

B1: Chia lưới ảnh

B2: Mỗi ô lưới chọn ra vector đặc trưng

B3: Chọn ra 1 vector đặc trưng và đặt làm khởi thủy

K-means	K-means++
Khởi tạo vector đặc trưng (centroid) khởi thủy = lấy ngẫu nhiên vector đặc trưng của vùng từ ảnh đã bị chia ô lưới rồi tính "loss with cost-sensitivities"/ trung bình cộng/...	Chọn ngẫu nhiên centroid đầu tiên từ các điểm dữ liệu. Đối với mỗi điểm dữ liệu, tính toán khoảng cách của nó từ trung tâm gần nhất, đã chọn trước đó

4. Điều kiện dừng của vòng lặp

Khi các vùng (cluster) đã ổn định.

$$\frac{\partial E(t)}{\partial t} = 0 \approx \frac{E(t) - E(t-1)}{t - (t-1)} = 0 \Rightarrow |E(t) - E(t-1)| < \varepsilon$$

```
stop_condition:
 bool flag = true;
 for (int j=1; j <= k; j++){
  If (w_j^k != w_j^(k-1)) flag = false;
  Else continue;
 }
If (flag == true) break;
Else update prototype; goto stop_condition;
```

5. Điều kiện kết nạp ứng viên vào cluster C_j* và nearest w_j* như thế nào? (Hint:Tính distance Step 2)

Tính độ dị biệt của vector đặc trưng (centroid) đang xét với mỗi vùng (cluster)

Cluster mình chọn để kết nạp phải làm cho đối số độ dị biệt của vector đặc trưng đang xét với vector đặc trưng khởi thủy là cực tiểu.

Điều kiện cần của cực tiểu:

$$\frac{\partial E(t)}{\partial t} = 0 \approx \frac{E(t) - E(t-1)}{t - (t-1)} = 0 \Rightarrow |E(t) - E(t-1)| < \varepsilon$$

```
For l in range(1,n):
minDistance=distance(i_L, w_i)
minIndx=1
  For j* in range(1,k):
    If (distance(i_L, w_j*)< minDistance):
      minDistance= distance(i_L, w_j*)
      minIndx = j*
  cluster[minIndx].add(i_L)
```

6. Cập nhật lại centroid cho tất cả các vùng

Trung bình cộng

$$prototype\ khởi\ thủy\ w_j = \sum_{vector\ đặc\ trưng\ i_L\ của\ \in\ từng\ vùng\ C_j\ đã\ phân\ đoạn} \frac{i_L}{|C_j|}$$

1. Xác định độ phức tạp tính toán của 2 thuật toán Fourier f*h và F*H bằng cách lập bảng so sánh

$$\zeta\{(f * h)(x,y)\} = F(u,v).H(u,v)$$
$$\zeta\{f(x,y).h(x,y)\} = (F * H)(u,v)$$

f*h	F*H
$f(x,y)*h(x,y)=\frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}f(x-m,y-n)h(m,n)$ Độ phức tạp của phương pháp lọc ảnh trong miền không gian: O(n^2 *m^2)	Bước 1: Fourier(f)= F(u,v) O(n^4) giảm còn O(n^2log2 n): FFT Bước 2: Fourier(h)= H(u,v): O(n^2) Bước 3: F(u,v).H(u,v) : O(n^2) Bước 4: Fourier^-1{F.H}: O(n^2log2 n) Độ phức tạp của phương pháp lọc ảnh trong miền tần số: O(n^2log2 n)

Kết luận: Phương pháp nào nhanh hơn sẽ phụ thuộc vào giá trị của m và n.

2. Cái thứ 2 dùng để làm gì Discrete Fourier Transform

Sử dụng để lọc ảnh trong miền tần số.

3. Tầm quan trọng của định lí trong 2 tác vụ làm trơn ảnh và phát hiện biên cạnh 7.2.2 Convolution (2 tác vụ dùng định lí này ở chỗ nào)

• Ứng dụng hữu ích 1: Sử dụng không gian tần số để hiểu tác dụng của bộ lọc

- Ví dụ: Biến đổi Fourier của một Gaussian là một Gaussian

- Do đó: làm giảm tần số cao

Ứng dụng hữu ích 2: Tính toán hiệu quả - Biến đổi Fourier nhanh (FFT) mất thời gian O (n log n)

- Do đó, có thể thực hiện tích chập trong thời gian O (n log n + m log m)

- Tăng hiệu quả lớn nhất cho các bộ lọc lớn

4. $$f(t) = \begin{cases} 0, t < 0 \\ e^{-at}, t \geq 0, a > 0 \end{cases}$$

a) F(w)

$$F(w) = \frac{1}{2\pi}\int_0^{\infty} e^{-iwt}dt = \frac{e^{-(a+iw)t}|_0^{\infty}}{-2\pi(a+iw)} = \frac{1 - \lim_{t\to\infty} e^{-(a+iw)t}}{2\pi(a+iw)} = \frac{1 - \lim_{t\to\infty} |e^{-(a+iw)t}|}{2\pi(a+iw)}$$
$$= \frac{1 - \lim_{t\to\infty} |e^{-at}.e^{iwt}|}{2\pi(a+iw)} = \frac{1 - \lim_{t\to\infty} |e^{-at}|}{2\pi(a+iw)} = \frac{1-0}{2\pi(a+iw)} = \frac{1}{2\pi(a+iw)}$$

b) f(t)

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{e^{iwt}}{a+iw}dw$$

5. Giữ cái nào bỏ cái nào?

Để cực tiểu sai số $E(||x - \hat{x}||^2)$

$$= \sum_{i=m}^{n-1} a_i{}^T.E[xx^T].a_i = \sum_{i=m}^{n-1} a_i{}^T.\lambda_i.a_i = \sum_{i=m}^{n-1} \lambda_i$$

thì ta sắp theo thứ tự giảm dần rồi

- giữ $\lambda_0 \to \lambda_{m-1}$: lớn nhất
- bỏ $\lambda_m \to \lambda_{n-1}$: bé nhất