دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشگاه صنعتی امیرکبیر
دانشکده مهندسی مکانیک

گزارش تشریحی **تمرین سری سوم**

درس پردازش تصاویر دیجیتال

استاد: دکتر آذرنوش

دانشجو: محمدجواد زلقی

شماره دانشجویی: ۹۸۱۲۶۰۷۹

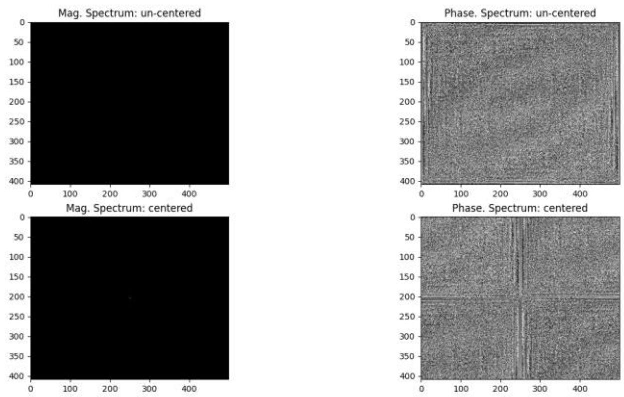
تاریخ: ۱۳۹۹/۹/۸

سوال اول: الف)

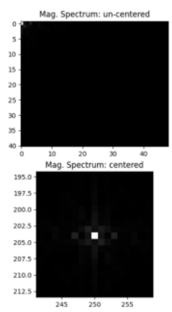
برای پیاده‌سازی تبدیل فوریه گسسته یا DFT بر روی تصویر از تابع `np.fft.fft2()` استفاده می‌کنیم که در خروجی تبدیل فوریه آرایه گسسته دو بعدی را بصورت یک آرایه شامل اعضای موهومی به ما می‌دهد. با استفاده از تابع `np.fft.fftshift()` آرایه تبدیل فوریه را به مرکز تصویر منتقل می‌کنیم. اگر این کار انجام نشود، فرکانس‌های صفر گذار در گوشه سمت چپ بالای تصویر قرار می‌گیرند که مطلوب ما نیست.

برای تشکیل اسپکتروم فاز از خروجی تبدیل فوریه، از تابع `np.angle()` استفاده می‌کنیم که بر اساس مقدار حقیقی و موهومی هر المان مقدار فوریه گسسته محاسبه شده، جهت‌گیری آن را محاسبه می‌کند. همچنین برای تشکیل اسپکتروم اندازه، ابتدا اندازه خروجی تبدیل فوریه را اندازه گیری می‌کنیم، سپس آن را نرمالایز می‌کنیم.

برای دو حالت سنتر نشده و سنتر شده تبدیل فوریه، اسپکتروم‌های فاز و اندازه به شرح زیر نمایش داده می‌شوند:



نمودار اسپکتروم فاز کاملا مشخص است، اما اسپکتروم اندازه در فوریه سنترد نشده بصورت ضربه کوچک در گوشه چپ بالا و در فوریه سنترد شده در مرکز قرار دارد. اگر زوم کنیم:



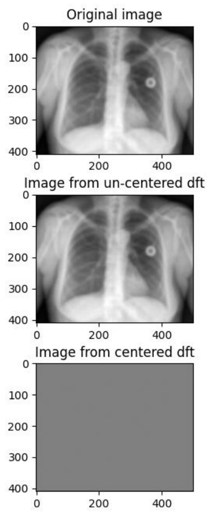
اگر بخواهیم این اسپکتروم‌های اندازه را بهتر ببینیم، باید با تبدیل شدتی مثلا لگاریتم گرفتن، توضیح اینتنسیتی را در سطوح نزدیک‌تر انجام دهیم (صورت سوال این مورد را نخواسته بود).

سوال اول: ب)

ما در قسمت اول تبدیل فوریه گسسته از تصویر گرفتیم. در این مرحله باید با اعمال معکوس تبدیل فوریه گسسته بر روی خروجی قسمت الف همین سوال، تصویر اصلی را تولید کنیم.

برای اعمال معکوس تبدیل فوریه گسسته دو بعدی، از تابع `np.fft.ifft2()` استفاده می‌کنیم. همچنین باید ذکر گردد که بخاطر محسابات عددی و الگوریتم این تابع، مقادیر بسیار کوچک موهومی در خروجی نیز وجود دارند. برای راحت شدن از آنها از تابع `np.real()` استفاده می‌کنیم که فقط بخش حقیقی تصویر را ذخیره می‌کند.

حال نتیجه اعمال این تابع را بر روی دو خروجی تبدیل فوریه گسسته بخش اول (سنتر نشده و سنتر شده) نمایش می‌دهیم:



مشخص است معکوس نسخه سنتر نشده تبدیل فوریه گسسته دقیقا همان تصویر اصلی را برمی‌گرداند اما معکوس نسخه سنتر شده، چیزی نمی‌دهد. دلیل نیز این است که مبدا فرکانس صفر در گوشه بالا سمت چپ قرار گرفته است و در سایر نقاط در این تصویر ترنزیشن خاصی نداریم. بدیهی است اگر در کاری مثل حذف نویز در فضای فرکانس، از نسخه سنتر شده تبدیل فوریه گسسته یک تصویر استفاده کردیم، قبل از برگرداندن تصویر به حوزه مکان باید فرکانس صفر از مرکز به گوشه منتقل شود که برای این کار از تابع numpy.fft.ifftshift() می‌توان استفاده کرد.

سوال اول: ج)

برای چرخش تصویر حول مرکز تصویر بگونه‌ای که تصویر آینه شود، کافیست مزدوج تبدیل فوریه تصویر سنتر شده را بدست آوریم. زیر با این کار فاز می‌چرخد اما اندازه تغییر نمی‌کند.

```
mirrored_about_cented_freq = np.conj(chest_img_freq_domain_centered)
```

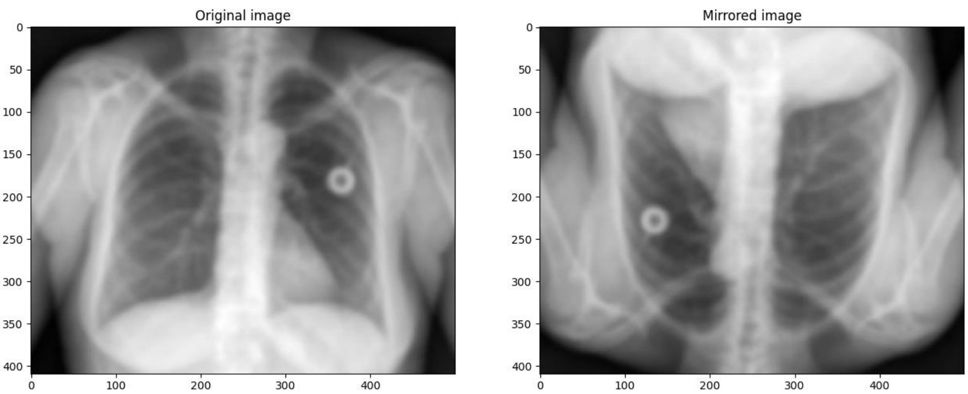
سپس آن را به گوشه شیفت بدهیم:

```
mirrored_about_cented_freq = np.fft.ifftshift(mirrored_about_cented_fre
q)
```

و در نهایت از آن تبدیل فوریه گسسته معکوس بگیریم تا به مکان ببریم و فقط بخش حقیقی آن را مد نظر قرار بدهیم (زیر بخش موهومی در فضای موقعیت معنا ندارد و آنچه در خروجی نیز می‌ماند، بخاطر الگوریتم-های عددی محاسبه عکس تبدیل فوریه است که مقادیر بسیار ناچیزی هستند):

```
mirrored_about_center_pos = np.fft.ifft2(mirrored_about_cented_freq)
mirrored_about_center_pos = np.real(mirrored_about_center_pos)
```

در انتها نتایج را نمایش می‌دهیم:



سوال دوم)

برای اعمال فیلترهای مطرح شده در فضای فرکانسی بر روی یک تصویر، تابع `freq_filtering(img, filter, parameters)` نوشته شده است که در ورودی تصویر مبدا، نوع فیلتر و پارامترهای مورد نیاز را می‌گیرد. برای پیاده‌سازی فیلتر در حوزه فرکانس، طبق کتاب گنزالس مرحله به مرحله تابع را گسترش دادیم.

در گام اول داریم:

1. Given an input image $f(x, y)$ of size $M \times N$, obtain the padding parameters P and Q from Eqs. (4.6-31) and (4.6-32). Typically, we select $P = 2M$ and $Q = 2N$.

```
f = img.copy()
A, B = f.shape
D0 = parameters[0]
```

در خصوص شعاع $cutt-off$ قرار می‌کنیم بعنوان اولین پارامتر برای هر نوع فیلتری، مقدار داده شده مربوط به این شعاع است.

در گام دوم داریم:

2. Form a padded image, $f_p(x, y)$, of size $P \times Q$ by appending the necessary number of zeros to $f(x, y)$.

پس با استفاده از تصویر ورودی و با تعیین Q و P طبق خواسته، آن را $zero\ pad$ می‌کنیم:

```
P = 2 * A
Q = 2 * B
# padded image
f_p = np.zeros((P,Q))
f_p[0:A,0:B] = f
```

در گام سوم داریم:

3. Multiply $f_p(x, y)$ by $(-1)^{x+y}$ to center its transform.

برای اعمال این گام از حلقه استفاده می‌کنیم:

```
f_p_centered = f_p.copy()
for x in range(P):
  for y in range(Q):
    f_p_centered[x,y] = (-1)**(x+y) * f_p[x,y]
```

برای گام چهارم داریم:

4. Compute the DFT, $F(u, v)$, of the image from step 3.

بسیار راحت به محاسبه تبدیل فوریه تصویر پد شده سنتر شده میپردازیم:

```
F = np.fft.fft2(f_p_centered)
```

برای گام پنجم داریم:

5. Generate a real, symmetric filter function, $H(u, v)$, of size $P \times Q$ with center at coordinates $(P/2, Q/2)$.† Form the product $G(u, v) = H(u, v)F(u, v)$ using array multiplication; that is, $G(i, k) = H(i, k)F(i, k)$.

طبق خواسته سوال، ما شش فیلتر در نظر گرفته‌ایم:

1. `filter == 'LP_Ideal'`
2. `filter == 'HP_Ideal'`
3. `filter == 'LP_Butterworth'`
4. `filter == 'HP_Butterworth'`
5. `filter == 'LP_Gaussian'`
6. `filter == 'HP_Gaussian'`

مکانیزم ایجاد فیلترها دقیقا مشابه است و فقط مقداردهی به آن‌ها تفاوت دارد که توضیح داده می‌شود. همچنین فقط در فیلتر گوسین بجز شعاع کات آف، پارامتر n نیز بعنوان ورودی داده شود. برای ایجاد یک فیلتر، اگر *Low pass* بود، ابتدا یک ماتریس *zero* (اگر *High pass* بود، ماتریس *ones*) با ابعاد $P \times Q$ می‌سازیم:

```
H = np.zeros((P,Q))     or     H = np.ones((P,Q))
```

سپس حول پیکسل مرکزی H یک اسلایس مربعی با طول یال D_0 در نظر می‌گیریم:

```
    square_slice = H[p_h-d0_h:p_h+d0_h, q_h-d0_h:q_h+d0_h]
```

سپس برای نقاطی که درون دایره محاط شده در مربع قرار می‌گیرند، بر حسب نوع فیلتر مقدار تعیین می‌کنیم. مثلا برای فیلتر *Ideal low pass* ، داریم:

```
    for u in range(square_slice.shape[0]):
      for v in range(square_slice.shape[1]):
        D = ((u-int(square_slice.shape[0]/2))**2 + (v-
int(square_slice.shape[1]/2))**2)**0.5
        if D<=d0_h:
          square_slice[u,v] = 1
```

سپس وقتی بروز رسانی تمامی المان‌های اسلایس به پایان رسید، اسلایس را درون H قرار می‌دهیم:

```
H[p_h-d0_h:p_h+d0_h, q_h-d0_h:q_h+d0_h] = square_slice
```

توجه داریم برای برای فیلترهای دیگر نیز فقط بخش‌های تغییر کرده به شرح زیر است:

Ideal high pass filter:

```
H = np.ones((P,Q))
        square_slice[u,v] = 0
```

Butterworth low pass filter:

```
H = np.zeros((P,Q))
        square_slice[u,v] = 1 / (1 + D/D0)**(2*n)
```

Butterworth high pass filter:

```
H = np.ones((P,Q))
        square_slice[u,v] = 1 - 1 / (1 + D/D0)**(2*n)
```

Gussian low pass filter:

```
H = np.zeros((P,Q))
        square_slice[u,v] = e**(-50*(D**2)/(2*(D0**2)))
```

Gussian high pass filter:

```
H = np.ones((P,Q))
        square_slice[u,v] = 1 - e**(-50*(D**2)/(2*(D0**2)))
```

پس برای تمام فیلترها H متناظر در تابع پدید آمده است. حال با $G(i,k) = H(i,k)F(i,k)$ داریم:

```
G = np.multiply(H,F)
```

برای گام ششم داریم:

6. Obtain the processed image:

$$g_p(x, y) = \left\{\text{real}\left[\Im^{-1}[G(u, v)]\right]\right\}(-1)^{x+y}$$

در تابع براحتی داریم:

```
g_p_centered = np.real(np.fft.ifft2(G))
```

دلیل بوجود آمدن قسمت موهومی در g_p الگوریتم محاسباتی است که ما آنها را خودمان حذف می‌کنیم. سپس برای کرنر کردن `g_p_centered` با حلقه داریم:

```
g_p = g_p_centered.copy()
for x in range(P):
  for y in range(Q):
    g_p[x,y] = (-1)**(x+y) * g_p_centered[x,y]
```

در گام هفتم و پایانی داریم:

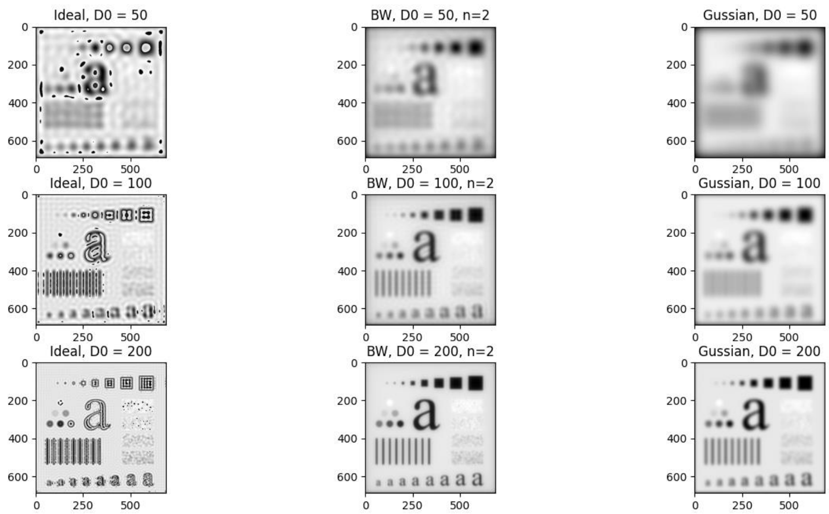
7. Obtain the final processed result, $g(x, y)$, by extracting the $M \times N$ region from the top, left quadrant of $g_p(x, y)$.

که در واقع قسمت گوشه که فقط تصویر فیلتر شده است را برای خروجی دادن استفاده می‌کنیم:

```
g = np.array(g_p[:A,:B], dtype=np.uint8)
```

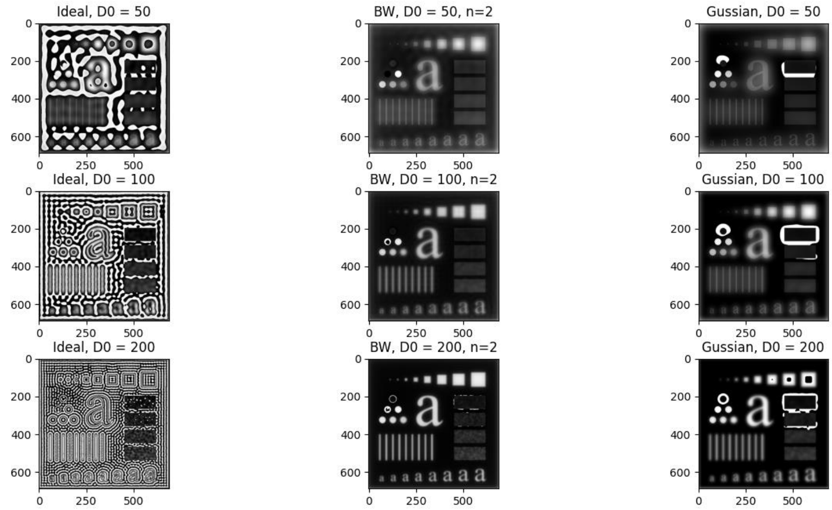
تا اینجا ساختار و گام‌های پیاده‌سازی تابع بیان شد. در ادامه در حالت‌های خواسته شده نتایج را بر روی تصویر می‌بینیم.

اعمال فیلترهای *Low pass* بر روی تصویر



بصورت کلی فیلترهای پایین گذر باعث می‌شوند ترنزیشن‌های با فرکانس بالاتر از کات آف حذف شوند. پس هرچه مقدار کات آف کم‌تر باشد، تصویر بلرتر می‌شود و هرچه کات آف بیشتر باشد، تصویر به تصویر واقعی نزدیک‌تر است زیرا ترنزیشن‌های با فرکانس بسیار بالا فقط فیلتر شده‌اند. در نتایج ما نیز همین اتفاق رخ داده است. تفاوت اصلی خروجی‌های نیز در پدیده رینگینگ می‌باشد بطوری که در فیلتر ایده آل رینگینگ به وضوح مشخص است اما در دو فیلتر دیگر رینگینگ کاهش یافته است. در باتر وارث مقدار بسیار کمی رینگینگ داریم و در فیلتر گوسین اصلا رینگینگ به چشم نمی‌آید. پس بطور کلی این فیلترها منجر به بلرشدن تصویر و کم‌اثر شدن مرز پله‌ها می‌شود. البته فیلتر ایده آل بخاطر ناپیوسته بودن کمی به محتوی تصویر ضربه می‌زند اما دو فیلتر دیگر خصوصا عملکرد بسیار خوبی داشته‌اند.

اعمال فیلترهای $High\ pass$ بر روی تصویر



در توضیحات این فیلتر، معکوس فیلتر پایین گذر را داریم. بدین معنی که فرکانس‌های شدت بالا حفظ شده و فرکانس‌های کم‌تر از کات آف حذف می‌شوند. عملا برای شارپ کردن و بولد کردن تغییرات شدید از این فیلتر می‌توان بهره برد. مجدد در دو فیلتر باتر ورث و گاوسین رینگینگ بسیار کم است و همچنین محتوی تصویر کاملا حفظ شده‌اند اما در فیلتر ایده آل بخاطر ناپیوستگی علاوه بر رینگینگ، محتوی مرز نیز دچار آسیب شده است. فارغ از نوع فیلتر، اینکه محاسبات نیز عددی می‌باشد و ما در آخرین مرحله فقط بخش حقیقی داده‌ها را در نظر می‌گیریم نیز شاید بی‌اثر نباشد.

سوال سوم)

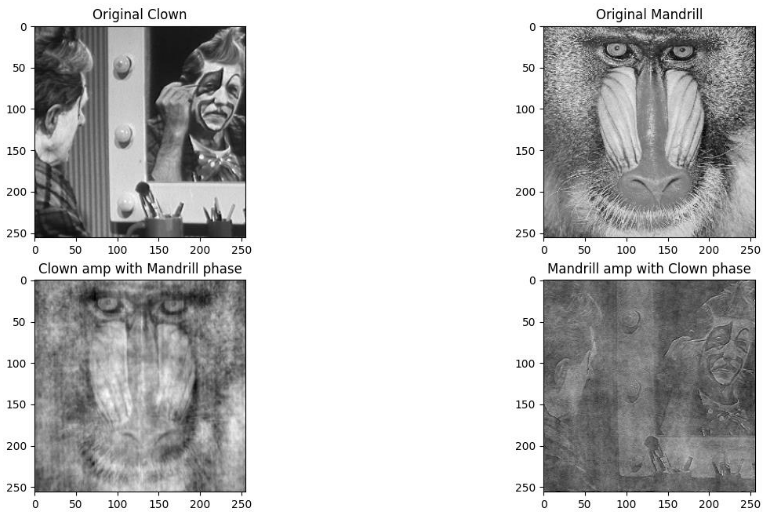
ابتدا در دو تصویر را خواندیم و تبدیل فوریه گسسته آن‌ها را نمایش دادیم. سپس برای اینکه براحتی فازهای آن‌ها را جابجا کنیم در حالیکه امپلیتیود ها را حفظ کنیم، دو تابع بسیار ساده و کاربردی نوشتیم. تابع اول جهت محاسبه امپلیتیود و فاز یک آرایه کامپلکس می‌باشد که در ورودی آرایه کامپلکس را می‌گیرد:

```
def amp_phase(x):
  return np.abs(x), np.angle(x)
```

تابع دوم یک تابع است که در وروی فاز و امپلیتیود را گرفته و طبق تعریف اویلر از عدد موهومی، برای ما یک آرایه موهومی ایجاد می‌کند:

```
def complex_array(amp,phase):
  return np.multiply(amp, np.exp(1j*phase))
```

در برنامه ابتدا فاز و امپلیتیود هر دو تبدیل فوریه دو تصویر را محاسبه می‌کنیم، سپس با امپلیتیود هر کدام و فاز دیگری و با تابع دوم، تبدیل فوریه تصویر جدید محاسبه می‌شود و سپس از آرایه کاپلکس ایجاد شده تبدیل فوریه گسسته معکوس می‌گیریم. در خروجی داریم:



تحلیل نتایج: مشاهده می‌شود مبنای بازسازی مجدد الگوهای تصویر جدید، قسمت فاز تبدیل فوریه می‌باشد. الگوهای تصویر در این قسمت قرار دارند و امپلیتیود نقشی در آن تقریبا ندارد. بصورت کلی در فاز اطلاعات مکانی قرار دارند که در بازسازی تصویر بسیار تعیین‌کننده می‌باشند اما در امپلیتیود شدت فرکانس‌ها قرار دارد و حاوی اطلاعات مکانی نمی‌باشد.

درباره اهمیت، هر کدام در یک کاربردی مهم هستند. برای بهبود تصویر از نظر نویز (حذف نویز) بررسی اسپکتروم فرکانس‌ها اطلاعات مهمی به ما می‌دهد اما برای کاری مثل این تمرین (بازسازی/بازیابی الگو) با توجه به اینکه اطلاعات مکانی درون فاز قرار دارند، فاز برای ما مهم‌تر می‌باشد.