پاسخ تمرین چهارم **مهدیه نادری: ۹۸۵۲۲۰۷۶**

سوال اول) محاسبه ی تبدیل فوریه تصویر زیر:

۲	۳
۱	۴

فرمول های زیر برای تبدیل فوریه ی گسسته میباشند:

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{+j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

در این مسئله $N = 2$ و $M = 2$ است. برای راحتی ابتدا مقادیر $e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})}$ را برای برای ۴ حالت بدست می آوریم و سپس با کمک آن به محاسبه ی ضرایب $F(u,v)$ میپردازیم.

میدانیم:

$$e^{jx} = \cos(x) + j.\sin(x)$$

از طرفی طبق شکل مقادیر f به شرح زیر است:

$$f(0,0) = 2,\quad f(0,1) = 3,\quad f(1,0) = 1,\quad f(1,1) = 4$$

الف) u=0و v=0:

$$e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} = e^{-j2\pi(\overset{0}{\cancel{\frac{0\times x}{2}}}+\frac{0\times y}{2})} = \cos(0) + j.\sin(0) = 1$$

$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{0\times x}{M}+\frac{0\times y}{N})} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) = 2+3+1+4 = 10$$

ب) u=0و v=1:

$$e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} = e^{-j2\pi(\overset{\frac{y}{2}}{\cancel{\frac{0\times x}{2}}}+\frac{y}{2})} = \cos(-\pi y) + j.\sin(-\pi y) = \cos(-\pi y) = \cos(\pi y)$$

$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{0\times x}{2}+\frac{y}{2})} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y).\cos(\pi y)$$
$$= f(0,0)\times\cos(0) + f(0,1)\times\cos(\pi) + f(1,0)\times\cos(0) + f(1,1)\times\cos(\pi)$$
$$= 2 + 3\times(-1) + 1 + 4\times(-1) = 2 - 3 + 1 - 4 = -4$$

ج) u=1و v=0:

$$e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} = e^{-j2\pi(\frac{\frac{x}{2}}{2}+\frac{0\times y}{2})} = \cos(\pi x) + j.\sin(\pi x) = \cos(\pi x)$$

$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{x}{2}+\frac{0\times y}{2})} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y).\cos(\pi x)$$
$$= f(0,0)\times\cos(0) + f(0,1)\times\cos(0) + f(1,0)\times\cos(\pi) + f(1,1)\times\cos(\pi)$$
$$= 2 + 3 + 1\times(-1) + 4\times(-1) = 2 + 3 - 1 - 4 = 0$$

د) u=1و v=1:

$$e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} = e^{-j2\pi(\frac{x}{2}+\frac{y}{2})} = \cos(-\pi(x+y)) + j.\sin(-\pi(x+y)) = \cos(\pi(x+y))$$

$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{x}{2}+\frac{y}{2})} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y).\cos(\pi(x+y))$$
$$= f(0,0)\times\cos(0) + f(0,1)\times\cos(\pi) + f(1,0)\times\cos(\pi) + f(1,1)\times\cos(2\pi)$$
$$= 2 + 3(-1) + 1\times(-1) + 4\times(1) = 2 - 3 - 1 + 4 = 2$$

منبع: اسلاید ها و صحبت های استاد در ویدیو۵و۶

سوال دوم) بخش اول: ابعدا فضا در بخش حقیقی دامنه ی فرکانسی تصویر با ابعاد $n \times n$

باید از فرمول های زیر(در اسلاید های جلسه ی ششم بودند)برای این سوال استفاده کرد:

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{+j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

همانطور که از فرمول بالا مشخص است $F(u,v)$ دو بخش حقیقی و موهومی داردکه در سوال بخش حقیقی را میخواهد.

$$Real(F(u,v)) = Real(\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)})$$
$$= Real(\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)(\cos(-2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right) + j.sin(-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)))$$

که باید در آن $M = n$ و $N = n$ را جایگذاری کنیم.

$$Real(F(u,v)) = Real(\sum_{x=0}^{n-1}\sum_{y=0}^{n-1} f(x,y)e^{-j2\pi(\frac{ux}{n}+\frac{vy}{n})}) =$$
$$= Real(\sum_{x=0}^{n}\sum_{y=0}^{n} f(x,y)\left(\cos\left(-2\pi\left(\frac{ux+vy}{n}\right)\right)+j.\sin\left(-2\pi\left(\frac{ux+vy}{n}\right)\right)\right))$$
$$= \sum_{x=0}^{n-1}\sum_{y=0}^{n-1} f(x,y)\left(\cos\left(-2\pi\left(\frac{ux+vy}{n}\right)\right)\right) = \sum_{x=0}^{n-1}\sum_{y=0}^{n-1} f(x,y)(\cos(2\pi\left(\frac{ux+vy}{n}\right)))$$

تابع کسینوسی نسبت به محور مرکز مختصات قرینه است در اینجا چون $\frac{ux+vy}{n}$ به u و v هم بستگی دارد یعنی مثلا حالت x=0,y=1 با حالت x=1,y=0 جواب یکسانی ندارد. برای همین این تابع سیگما نسبت به محور x,y قرینه نیست. برای همین ابعاد آن میشود $\frac{n\times(n+1)}{2}$.

سوال دوم) بخش دوم: رابطه ی تبدیل فوریه نقطه ی مبدا با مقادیر بقیه نقاط تصویر:

فرکانس صفر معادل مجموع شدت روشنایی همه ی نقاط تصویر است. منبع: ویدیو کلاس و صحبت استاد

اثبات:

فرمول تبدیل فوریه:

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{+j2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

جایگذاری نقطه ی (0,0):

$$F(0,0) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{0\times x}{M}+\frac{0\times y}{N})} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)$$

مقدار تبدیل فوریه نقطه ی مبدا = مجموع شدت روشنایی نقاط تصویر

$$f(0,0) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)$$

مقدار تابع در نقطه ی مبدا = میانگین مقادیر تبدیلات فوریه تصویر

سوال سوم)الف)محاسبه ی نتیجه ی اعمال یک کرنل روی یک تصویر سیاه و سفید:

برای اینکار هر درایه ی کرنل در یکی از پیکسل های همسایه ی نقطه ی مورد نظر ضرب و حاصل ها جمع میشوند. نقطه ی وسط کرنل نظیر با خود نقطه ی مورد نظر است. برای نقاط گوشه یا دیواره هم روش های مختلفی مثل صفر در نظر گرفتن مقدار همسایه هایی که موجود نیستند، وجود دارد.

```
    2D array, representing a grayscale image.
kernel: ndarray
    2D array, representing a linear kernel.
Returns
-------
ndarray
    The result of convolving `image` with `kernel`.
"""
result = np.zeros(image.shape)
########################
# Your code goes here. #
col = int(len(kernel)/2)

row = int(len(kernel[0])/2)

r_num = image.shape[0]

c_num = image.shape[1]

for x in range(r_num):
  for y in range(c_num):
    for i in range(-row,row):
      for j in range(-col,col):
          if((x - i >= 0 and y-j >= 0) and (x-i < r_num and y - j< c_num)):
              result[x, y] += kernel[i + row,j + col] * image[x - i, y - j]
########################
return result
```

در قسمت مشخص شده تابعی را نوشتیم که فرمول اعمال کرنل یک بعدی روی تصویر دو بعدی را اجرا میکند.

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t).f(x+s,y+t)$$

که در آن w مقادیر کرنل و f مقادیر شدت روشنایی نقاط تصویر است.

سوال سوم)ب) پیاده سازی تابع تولید کرنل میانگین گیر:

تولید یک فیلتر m*n مستلزم تعیین mn ضریب در کرنل است.

در کرنل میانگین گیر در تصویر حاصل، مقدار هر پیکسل برابر با میانگین مقادیر پیکسل های همسایه است. که میشود ۱ تقسیم بر تعداد پیکسل های کرنل ضرب در ماتریسی که همه ی درایه ای آن ۱ هستند.

```
result = np.ones((size, size))
########################
# Your code goes here. #
pixels_num = size * size
for i in range(size):
    for j in range(size):
        result[i,j] = (1 / pixels_num)
########################
return result
```

خواندن تصویر با نویز فلفل و نمک:

برای اینکه در ادامه ی سوال ورودی دادن تصویر به تابع اول به مشکل نخوریم در کانال ۰ تصویر را خواندم و گرنه rgb میشد.

```
In [12]: im = cv2.imread('images/Q3/salt_and_pepper_low.jpeg',0)
         plt.imshow(im)
         plt.axis('off')
Out[12]: (-0.5, 224.5, 224.5, -0.5)
```

نتیجه ی حاصل از فیلتر میانگین گیر:

```
kernel = averaging_kernel(3)
im_smoothed = filter_2d(im, kernel)
plt.imshow(im_smoothed)
plt.axis('off')
```

```
(-0.5, 224.5, 224.5, -0.5)
```

سوال سوم)پ)پیاده سازی فیلتر میانه گیر و مقایسه با فیلتر میانگین گیر:

نوعی پیکسل مرتبه ای است. خطی نیست و برای همین نمیشود با کانولوشن آن را پیاده سازی کنیم چون نیاز به sort دارد. معمولا وقتی شدت روشنایی ها را مرتب کنیم، نقاط ابتدایی و انتهایی بازه نویزی محسوب میشوند و نقاط وسط تصویر اصلی هستند. در فیلتر میانه، میانه ی پیکسل های درون کرنل را به دست می آوریم و در پیکسل مرکزی جایگذاری میکنیم.

```
    ----------
    image: ndarray
        2D array, representing a grayscale image.
    size: int
        Size of the window for median calculation.
    Returns
    -------
    ndarray
        The result of convolving `image` with `kernel`.
    """
    result = np.zeros(image.shape)
    ########################
    # Your code goes here. #
    row = image.shape[0]
    col = image.shape[1]
    for x in range(row):
        for y in range(col):
            kernel_image = []
        for i in range(size):
            for j in range(size):
                if((x + i > row or y + j > col)):
                    kernel_image.append(image[x - i, y - j])
                else:
                    kernel_image.append(image[x + i, y + j])

        kernel_image.sort()
        result[x,y] = kernel_image[int((len(kernel_image) - 1)/2)]
    ########################
    return result
```

نتیجه ی اعمال این فیلتر روی تصویر دارای نویز با نمک و فلفل شدید تر:

```
im = cv2.imread('images/Q3/salt_and_pepper_high.jpeg', 0)
plt.imshow(im)
plt.axis('off')
```

```
(-0.5, 224.5, 224.5, -0.5)
```

```
im_smoothed = median_filter(im, size=3)
plt.imshow(im_smoothed)
plt.axis('off')
```

```
(-0.5, 224.5, 224.5, -0.5)
```

سمت چپ تصویر اولیه و سمت راست تصویر بعد از اعمال فیلتر است.

همانطور که مشخص است تصویر جدید نویز کمتری نسبت به تصویر اصلی دارد اما خب تصویر خیلی تار شده است. در تصویر دارای نویز نمک و فلفلی چون این نویز جمع شونده نیست و به طور کلی رنگ یک نقطه از تصویر سفید یا سیاه میشود معمولا از همین نویز های غیر خطی استفاده میکنیم.

فیلتر میانگین گیر قسمت بالا هم نویز را کاهش داده اما نقاط نویز باقی مانده روی تصویر بزگتر و مشخص تر دیده میشوند اما در میانه گیر نقاط نویز محو تر و کوچکتر دیده میشوند اما جزئیات تصویر هم مشخص نیست و این از معایب آن است.

از طرفی فیلتر میانگین گیر چون بعد اضرب کرنل در نقاط تصویر فقط یک تقسیم دارد که با نامپای به سرعت میشود انجام داد اما در میانه گیر نیاز به مرتب سازی هست و این سرعت را پایین می آورد.

سوال سوم)ت)فیلتر محاسبه ی مشتق تصویر افقی یا عمودی:

رابطه ی مشتق مرتبه اول تصویر یک بعدی با تقریب یک جمله ای از بسط تیلور:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+1) - f(x-1)}{2}$$

رابطه ی مشتق مرتبه اول تصویر دو بعدی با تقریب یک جمله ای از بسط تیلور:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x-1,y)}{2}$$

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+1) - f(x,y-1)}{2}$$

```
derivative_kernel = np.array([
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0],
])

########################
# Your code goes here. #
derivative_kernel[1,0] = -1
derivative_kernel[1,2] = 1
########################
```

سمت چپ تصویر اصلی و تصویر راست بعد از اعمال فیلتر مشتق:

```
im_smoothed = filter_2d(im, derivative_kernel)
plt.imshow(im_smoothed)
plt.axis('off')
```

```
(-0.5, 255.5, 255.5, -0.5)
```

```
im = cv2.imread('images/Q3/cameraman_noisy.png',0)
plt.imshow(im)
plt.axis('off')
```

```
(-0.5, 255.5, 255.5, -0.5)
```

مشتق گیری برای مشخص تر کردن جزئیات تصویر خوبه و اگر نقطه ای اختلاف بیتری با نقاط همسایه داشته باشد این میزان اختلاف شدت روشنایی شدید تر میشود. اگر درکرنل اختلاف اعداد را کمتر کنیم تصویر هموار تر میشود. همچنین اگر از نقاط بیشتری استفاده کنیم مشتق تصویر دقیق تر محاسبه میشود و فقط اختلاف دو نقطه تقسیم بر دو که تقریب زیادی دارد نیست.میتوان از مشتق مرتبه بالاتر هم استفاده کرد. در اینجا ما فقط مشتق افقی استفاده کردیم. اگر مشتق عمودی و دیگرزوایا هم استفاده شود بهتر است. در کل برای نویز نمک فلفلی خوب نیست اما برای نویز های رندم یا جمع شونده میتواند خوب باشد.

سوال چهارم)الف)حذف نویز تصویر داده شده:

لینک های استفاده شده:

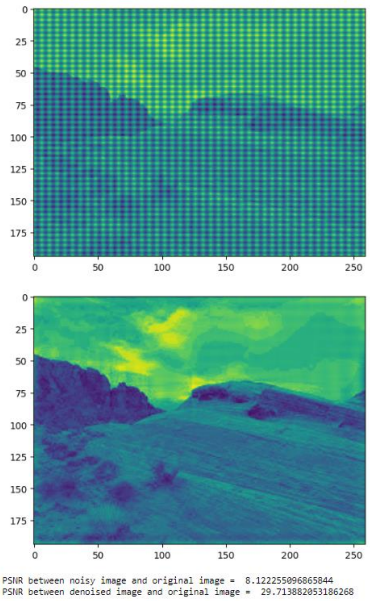
https://numpy.org/doc/stable/reference/generated/numpy.fft.fft.html

تصویر اولیه را به fft2 میدهیم تا فضای تصویر تغییر کرده و وارد فضای قوریه شود. سپس برای بهتر دیدن نویز ها آن را به تابع fftshft میدهیم و میفهمیم نویز ها در دو خط عمود منصف اضلاع مستطیل قرار گرفتند و آن را به ۴ قسمت تبدیل کرده اند. با حلقه های for نقاط مورد نظر را صفر میکنیم و پردازش لازم روی پیکسل ها را انجام میدهیم. سپس با توابع ishift و ifft2 فضای حالت تصویر را به صورت ابتدا در می آوریم. برای اطمینان از درستی مقدار پیکسل ها از real استفاده میکنیم که بخش های موهومی کاملا حذف شوند.

```
denoised = image.copy()
########################
# Your code goes here. #
denoised = np.fft.fft2(denoised)
denoised = np.fft.fftshift(denoised)
xs = image.shape[0]
ys = image.shape[1]
row_mid = int(xs / 2)
col_mid = int(ys / 2)
for i in range(row_mid - 4, row_mid + 4):
    for j in range(col_mid - 4 * 4):
        denoised[i, j] = 0
    for j in range(col_mid + 4 * 4, len(denoised[i])):
        denoised[i, j] = 0
for i in range(col_mid - 4, col_mid + 4):
    for j in range(row_mid - 4 - 4 * 4):
        denoised[j, i] = 0
    for j in range(row_mid + 4 + 4 * 4, xs):
        denoised[j, i] = 0
denoised = np.fft.ifftshift(denoised)
denoised = np.fft.ifft2(denoised)
denoised = np.real(denoised)
########################
```

نتیجه :

تصویر بالا اصلی به علاوه ی نویز و پایین بعد از دینوز شدن است.



سوال چهارم)ب)بهبود حاصل در تصویر:

لینک مورد استفاده: [Python | Peak Signal-to-Noise Ratio (PSNR) - GeeksforGeeks]

نسبت سیگنال به نویز پیک (PSNR)نسبت بین حداکثر توان ممکن یک تصویر و قدرت نویز مخرب است که بر کیفیت نمایش آن تأثیر می گذارد. برای تخمین PSNR یک تصویر، باید آن تصویر را با یک تصویر تمیز ایده آل با حداکثر توان ممکن مقایسه کرد. در واقع نسبت سیگنال مفید به نویزی میباشد.

$$PSNR = 10log_{10}(\frac{(L-1)^2}{MSE}) = 20log_{10}(\frac{L-1}{RMSE})$$

در اینجا، L تعداد حداکثر سطوح شدت ممکن (حداقل سطح شدت فرض کنید 0 باشد) در یک تصویر است.

MSE میانگین مربعات خطا است و به صورت زیر تعریف می شود:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}(O(i,j) - D(i,j))^2$$

جایی که O نشان دهنده داده های ماتریس تصویر اصلی است. D داده های ماتریسی تصویر تخریب شده را نشان می دهد. m نشان دهنده تعداد ردیف های پیکسل و i نشان دهنده شاخص آن ردیف از تصویر است. n نشان دهنده تعداد ستون های پیکسل و j نشان دهنده شاخص آن ستون تصویر است.

RMSE ریشه میانگین مربعات خطا است.

پاسخ تمرین چهارم **مهدیه نادری: ۹۸۵۲۲۰۷۶**

با دانستن مطالب بالا در مورد تابع گفته شده میتوان فهمید هر چه مقدار تابع بیشتر باشد، نویز کمتری داریم و طبق نتیجه برای تصویر دارای نویز مقدار تابع 8.122255096865844 و بعد از از بین بردن نویز آن به 29.713882053186268 رسید که یعنی نویز تصویر تا حد خیلی خوبی کاهش پیدا کرده است.

سوال چهارم)پ)نوع نویز تصویر و تفاوت نویز جمع شونده و ضرب شونده:

لینک مورد استفاده:

What are additive and multiplicative noise? - Quick Image Processing Research Guide (gofastresearch.com)

```
X,Y = original_image.shape
noise = np.zeros((X,Y))
for i in range(X):
  for j in range(Y):
    noise[i,j] = f(i,j)*100
noisy_image = original_image + noise
```

با توجه به کد استفاده شده ما طبق یک فرمول از قبل تعیین شده به اندازه ی پیکسل های تصویرمان یک ماتریس ساختیم و یک سری عدد به عنوان نویز در این ماتریس طبق فرمول ذخیره کردیم. سپس برای تولید تصویر نویزی ماتریس ساخته شده را با تصویر اصلی جمع کردیم در نتیجه نویز این تصویر جمع شونده میباشد.

به طور کلی جزء فرکانس بالای تصویر به عنوان نویز نامیده می شود.

ویژگی نویز جمع شونده:

1. نویز جمع شونده سیگنال ناگهانی ناخواسته ای است که به برخی از سیگنال های واقعی اضافه می شود.
2. مدل نویز جمع شونده به شرح رو به رو است: Noisy_image[t] = original_image[t] + noise[t]
3. حذف نویز جمع شونده از تصویر کار چندان دشواری نیست زیرا مدل های بازیابی تصویر زیادی برای حذف نویز جمع شونده موجود است.
4. نویز گاوسی بهترین مثال برای نویز جمع شونده است. این نویز در بسیاری از کاربردها مانند تصویربرداری نوری و تعداد کمی از تصاویر پزشکی مانند سی تی اسکن و غیره وجود دارد.
5. منابع اصلی: در طول اکتساب تصویر مثل یکنواخت بودن شدت نور محیط.
6. تأثیر این نویز بر روی تصویر کمتر از نویز ضرب شونده است زیرا در اینجا سیگنال نویز به سیگنال اصلی اضافه می شود در حالی که در ضرب، نویز چند برابر می شود.
7. دارای الگوی توزیع نرمال است.

پاسخ تمرین چهارم **مهدیه نادری: ۹۸۵۲۲۰۷۶**

ویژگی نویز ضرب شونده:

1. نویز ضرب شونده سیگنال ناگهانی ناخواسته ای است که به سیگنال واقعی ضرب می شود.

2. مدل نویز ضرب شونده به شرح زیر است:

Noisy_image[t] = original_image[t] * noise[t]

3. حذف نویز ضرب شونده از تصویر کار بسیار دشواری است زیرا مدل های بازیابی تصویر بسیار کمی برای حذف این نوع نویز موجود است. بنابراین کار بر روی این نویز برای محققان جدید بسیار چالش برانگیزتر و خوب است زیرا آنها دامنه بسیار بیشتری برای کار دارند.

4. راه دیگری برای مدیریت این نویز با استفاده از مدل‌های بازیابی تصویر از نویز افزودنی وجود دارد. این روش با تبدیل ماهیت ضربی به افزودنی با استفاده از تبدیل لگاریتمی به دست می آید. با استفاده از تبدیل log، نویز ضربی به نویز افزایشی تبدیل می شود و اکنون می توان هر روش فیلتری را برای آن اعمال کرد. و بعداً از لاگ معکوس برای به دست آوردن نتیجه صحیح استفاده می شود.

5. منابع اصلی: در حین ضبط، انتقال یا پردازش های دیگر.

6. این نویز تأثیر نامطلوبی بر روی تصویر می گذارد زیرا سیگنال نویز به سیگنال اصلی چند برابر می شود.

7. دارای الگوی توزیع گاما است.

8. نویز لکه ای بهترین مثال برای نویز ضرب شونده است.

9. این نوع نویز عمدتاً در تصاویر رادار و تصاویر اولتراسوند یافت می شود.