موضوع گزارش : اعمال انواع فیلتر ها بر روی عکس و بررسی روش های انجام آن

نام استاد درس : دکتر فدایی

نام دانشجو : محمد شکری

شماره دانشجویی : ۹۸۱۵۳۱۰۲۷

ترم بهار ۱۴۰۲

در بخش اول سوال باید یک function را بنویسیم که فیلتر های متفاوتی را بسازد . این فانکشن باید چندین ورودی مختلف از جمله omega و سایز فیلتر را بگیرد و سپس شروع به ساخت فیلتر بکند .

بگذارید اول نگاه کنیم که فیلتر چیست ؟

فیلتر کردن فرکانس های ناخواسته از تصویر را فیلتر می گویند. هدف از فیلتر کردن تصویر پردازش تصویر است تا نتیجه مناسب تر از تصویر اصلی برای یک برنامه خاص باشد. فیلتر تصویر به فرآیندی اطلاق می شود که نویز را حذف می کند، تصویر دیجیتال را برای کاربردهای متنوع بهبود می بخشد.
مراحل اساسی :

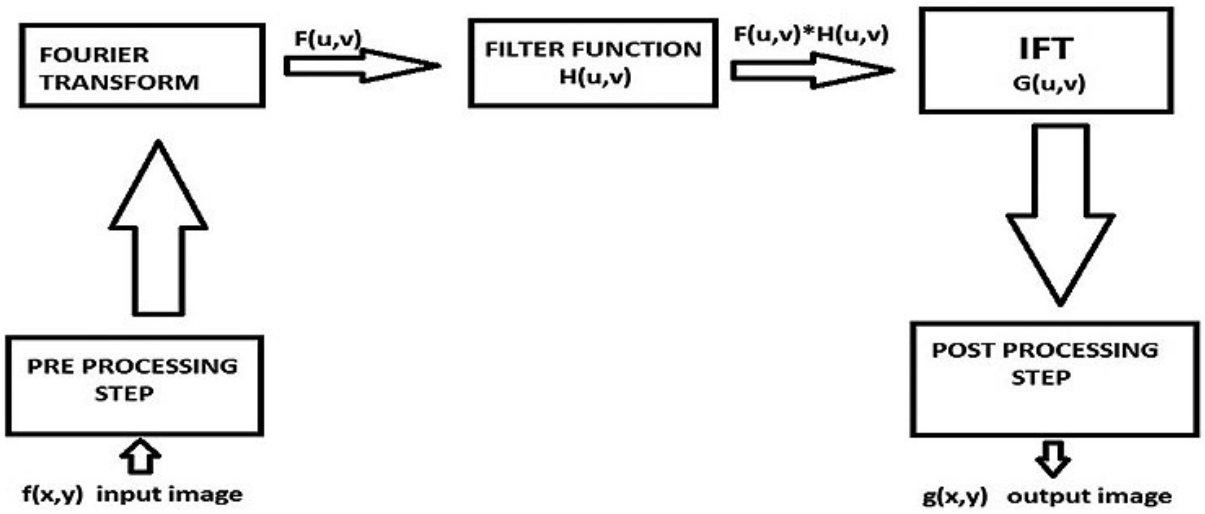


Figure 1: basic steps for filtering in frequency domain

Fourier transform

تبدیل فوریه فرکانس های قسمت های تناوبی تصویر را منعکس می کند. با اعمال تبدیل فوریه معکوس می توان فرکانس های نامطلوب یا ناخواسته را حذف کرد و به این حالت ماسک یا فیلتر می گوییم. یک فیلتر یک ماتریس است و اجزای فیلترها معمولاً از 0 تا 1 تغییر می کنند. اگر جزء 1 باشد، فرکانس اجازه عبور داده می شود و اگر جزء 0 باشد فرکانس به بیرون پرتاب می شود.

طیف وسیعی از وظایف پردازش تصویر را می توان با استفاده از فیلترهای مختلف پیاده سازی کرد. فیلتری که فرکانس های بالا را در حین عبور از فرکانس های پایین تضعیف می کند فیلتر پایین گذر نامیده می شود.

علاوه بر این، باند گذر (band reject pass filter) روی باندهای فرکانس خاصی کار می کند. فیلترهای ناچ روی فرکانس های خاصی کار می کنند. فیلترهای پایین گذر، بالا گذر و رد باند اغلب فیلترهای ایده آل نامیده می شوند، اگرچه همانطور که در شکل نشان داده شده است پرش دارند.

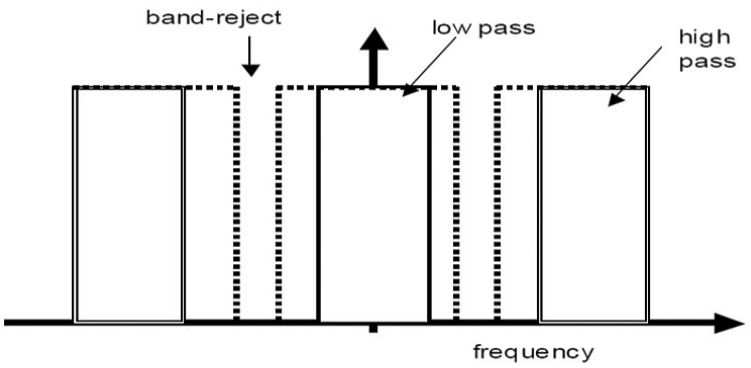


Figure 2: ideal filters

چهار نوع فیلتر وجود دارند : ideal filter, gaussian filter , notch filter , butterworth filter

نحوه انجام کار به این صورت است که باید تمام عکس ها را از حوزه مکان به حوزه فرکانس ببریم و در انجا به اعمال فیلتر بر روی عکس به حوزه فرکانس برده شده بپردازیم .

برای انتقال عکس از حوزه مکان به فرکانس باید از ان fft بگیریم .

انواع اعمال فیلتر ها : Low pass filter , high pass filter , band pass filter , band stop filter

در low pass filter باید جاهایی که مقدار برابر با ۱ است را اجاز عبور بدهیم و بقیه جاها را stop کنیم در High pass filter جاهایی که انرژی بیشتری دارن عبور داده می شوند .

در band pass filter یک باند وجود دارد که تمام اطلاعات و فرکانس هایی که در آن قرار دارند عبور می دهد و سایر جاها باید stop شوند .

در band stop filter نیز شبیه بالا است ولی بر عکس عمل می کنیم یعنی جاهایی که درون باند قرار دارند باید stop شوند .

تعداد 192 عکس موجود است که در انتها آورده شده و title هر عکس مشخص است و در توضیحات به آن ارجاع داده می شود.

کلاس فیلتر :

```
class Filters:
    @staticmethod
    def _loop(dim, callback):
        matrix = [[0 for j in range(dim[1])] for i in range(dim[0])]
        for u in range(dim[0]):
            for v in range(dim[1]):
                matrix[u][v] = callback(u, v)
        return matrix

    @staticmethod
    def idealLPF(dim, d0):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            return 0 if d0 > d else 1

        return Filters._loop(dim, callback)

    @staticmethod
    def idealHPF(dim, d0):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            return 1 if d0 > d else 0

        return Filters._loop(dim, callback)

    @staticmethod
    def idealBPF(dim, d0, bw):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            return 0 if d <= (d0 - bw / 2) or d >= (d0 + bw / 2) else 1

        return Filters._loop(dim, callback)

    @staticmethod
    def idealBSF(dim, d0, bw):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            return 1 if d <= (d0 - bw / 2) or d >= (d0 + bw / 2) else 0
```

```
        return Filters._loop(dim, callback)

    @staticmethod
    def gaussianLPF(dim, d0):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            return math.e ** (- (d ** 2 / (2 * d0 ** 2)))

        return Filters._loop(dim, callback)

    @staticmethod
    def gaussianHPF(dim, d0):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            return 1 - math.e ** (- (d ** 2 / (2 * d0 ** 2)))

        return Filters._loop(dim, callback)

    @staticmethod
    def gaussianBPF(dim, d0, bw):  # search
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            if bw * d == 0:
                return math.exp(- ((d ** 2 - d0 ** 2) / (2 * 0.000001)) ** 2)
            return math.exp(- ((d ** 2 - d0 ** 2) / (2 * bw * d)) ** 2)

        return Filters._loop(dim, callback)

    @staticmethod
    def gaussianBSF(dim, d0, bw):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            if bw * d == 0:
                return 1 - math.exp(- ((d ** 2 - d0 ** 2) / (2 * 0.000001))
** 2)
            return 1 - math.exp(- ((d ** 2 - d0 ** 2) / (2 * bw * d)) ** 2)

        return Filters._loop(dim, callback)

    @staticmethod
    def butterworthLPF(dim, d0, _2n):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            return 1 / (1 + (d / d0) ** (2 * _2n))

        return Filters._loop(dim, callback)

    @staticmethod
    def butterworthHPF(dim, d0, _2n):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            if d == 0:
                return 1 / (1 + (d0 / 0.000001) ** (2 * _2n))
            return 1 / (1 + (d0 / d) ** (2 * _2n))

        return Filters._loop(dim, callback)
```

```
    @staticmethod
    def butterworthBPF(dim, d0, bw, _2n):  ##### search
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            if d == 0:
                return 1 / (1 + ((d ** 2 - d0 ** 2) / (0.000001)) ** (2 *
_2n))
            return 1 / (1 + ((d ** 2 - d0 ** 2) / (d * bw)) ** (2 * _2n))

        return Filters._loop(dim, callback)

    @staticmethod
    def butterworthBSF(dim, d0, bw, _2n):
        def callback(u, v):
            d = math.sqrt((u - dim[0] / 2) ** 2 + (v - dim[1] / 2) ** 2)
            if d ** 2 - d0 ** 2 == 0:
                return 1 / (1 + ((d * bw) / (0.000001)) ** (2 * _2n))
            return 1 / (1 + ((d * bw) / (d ** 2 - d0 ** 2)) ** (2 * _2n))

        return Filters._loop(dim, callback)

    @staticmethod
    def notchIdeal(dim, d0, u0, v0):
        def callback(u, v):
            d1 = math.sqrt((u - dim[0] / 2 - u0) ** 2 + (v - dim[1] / 2 - v0)
** 2)
            d2 = math.sqrt((u - dim[0] / 2 + u0) ** 2 + (v - dim[1] / 2 + v0)
** 2)
            return 0 if d1 <= d0 or d2 <= d0 else 1

        return Filters._loop(dim, callback)

    @staticmethod
    def notchGaussian(dim, d0, u0, v0):
        def callback(u, v):
            d1 = math.sqrt((u - dim[0] / 2 - u0) ** 2 + (v - dim[1] / 2 - v0)
** 2)
            d2 = math.sqrt((u - dim[0] / 2 + u0) ** 2 + (v - dim[1] / 2 + v0)
** 2)
            return 1 - math.exp(-((d1 * d2) / (2 * d0 ** 2)))

        return Filters._loop(dim, callback)

    @staticmethod
    def notchButterworh(dim, d0, u0, v0,_2n):
        def callback(u, v):
            d1 = math.sqrt((u - dim[0] / 2 - u0) ** 2 + (v - dim[1] / 2 - v0)
** 2)
            d2 = math.sqrt((u - dim[0] / 2 + u0) ** 2 + (v - dim[1] / 2 + v0)
** 2)
            if d1 * d2 == 0:
                return 1 / (1 + (d0 ** 2 / (0.000001)))
            return 1 / (1 + (d0 ** 2 / (d1 * d2))**_2n)

        return Filters._loop(dim, callback)
```

- تمامی متد های این کلاس استاتیک هستند زیرا وابستگی وجود ندارد.
- متد loop_ ، ابعاد و callback را به عنوان ورودی می گیرد و با حلقه زدن روی آن ابعاد ، برای هر بار callback را صدا می زند و در آن نقطه خروجی callback را در ماتریس فیلتر قرار می دهد.
- تمامی متد های دیگر، فیلتر ها هستند که تعریف آن ها بدیهی است و با توجه به الگوریتم تعریف شده اند و نیاز به توضیح ندارد.

قرار دادن فیلتر ها روی عکس :

```
def applyFilterThenPlot(name,src, filter, *args):
    img = Image.open(src).convert('L')
    imgMatrix = np.array([[img.getpixel((i, j)) for j in range(img.size[1])]
for i in range(img.size[0])])

    plt.figure(num=name+'-image')
    plt.imshow(imgMatrix, cmap='gray')
    plt.show()

    args = list(args)
    args.insert(0, img.size)
    strFilter = 'Filters.{0}(*args)'.format(filter)

    filter = np.array(eval(strFilter, {'args': args, 'Filters': Filters}))
    plt.figure(num=name+'-filter')
    plt.imshow(filter, cmap='gray')
    plt.show()

    f = np.fft.fft2(np.array(imgMatrix))
    f = numpy.fft.fftshift(numpy.array(f))
    plt.figure(num=name + '-fft')
    plt.imshow(abs(f), cmap='gray')
    plt.show()

    out = f * filter
    out = np.fft.ifftshift(out)
    out = np.fft.ifft2(out)
    out = np.abs(out)

    newImg = Image.new(img.mode, (img.size), 'white')
    [[newImg.putpixel((i, j), int(out[i][j])) for j in range(img.size[1])]
for i in range(img.size[0])]
    newImg.save('test.bmp')
    plt.figure(num=name+'-image')
    plt.imshow(out, cmap='gray')
    plt.show()
```

- تابع applyFilterThenPlot ، نام و مسیر عکس و نام فیلتر و ورودی های فیلتر را می گیرد سپس عکس را نشان می دهد سپس فیلتر را ساخته و نمایش می دهد سپس fft عکس را گرفته و آن را نمایش می دهد سپس عکس فیلتر شده را نمایش می دهد.
  - هدف از تعریف این تابع راحتی کار است که با توجه به نمونه های استفاده می بینید که خیلی پویا و راحت است.

اعمال فیلتر ها :

- ```
  applyFilterThenPlot('idealLPF-45','./lena.bmp','idealLPF',45) #pi/4
  applyFilterThenPlot('idealLPF-90','./lena.bmp','idealLPF',90) #pi/2
  applyFilterThenPlot('idealLPF-120','./lena.bmp','idealLPF',120) #2pi/3

  applyFilterThenPlot('gaussianLPF-45','./lena.bmp','gaussianLPF',45) #pi/4
  applyFilterThenPlot('gaussianLPF-90','./lena.bmp','gaussianLPF',90) #pi/2
  applyFilterThenPlot('gaussianLPF-120','./lena.bmp','gaussianLPF',120) #2pi/3

  applyFilterThenPlot('butterworthLPF-45','./lena.bmp','butterworthLPF',45,2) #pi/4
  applyFilterThenPlot('butterworthLPF-90','./lena.bmp','butterworthLPF',90,2) #pi/2,n=2
  applyFilterThenPlot('butterworthLPF-120','./lena.bmp','butterworthLPF',120,2) #2pi/3,n=2

  applyFilterThenPlot('idealHPF-45','./lena.bmp','idealHPF',45) #pi/4
  applyFilterThenPlot('idealHPF-90','./lena.bmp','idealHPF',90) #pi/2
  applyFilterThenPlot('idealHPF-120','./lena.bmp','idealHPF',120) #2pi/3

  applyFilterThenPlot('gaussianHPF-45','./lena.bmp','gaussianHPF',45) #pi/4
  applyFilterThenPlot('gaussianHPF-90','./lena.bmp','gaussianHPF',90) #pi/2
  applyFilterThenPlot('gaussianHPF-120','./lena.bmp','gaussianHPF',120) #2pi/3

  applyFilterThenPlot('butterworthHPF-45','./lena.bmp','butterworthHPF',45,2) #pi/4,n=2
  applyFilterThenPlot('butterworthHPF-90','./lena.bmp','butterworthHPF',90,2) #pi/2,n=2
  applyFilterThenPlot('butterworthHPF-120','./lena.bmp','butterworthHPF',120,2) #2pi/3,n=2

  applyFilterThenPlot('notchIdeal1','./noisyimage1.bmp','notchIdeal',75,108,6) #u0=108,v0=6
  applyFilterThenPlot('notchGaussian1','./noisyimage1.bmp','notchGaussian',75,108,6)
  #u0=108,v0=6
  applyFilterThenPlot('notchButterworh1','./noisyimage1.bmp','notchButterworh',75,108,6,2)
  #u0=108,v0=6
  applyFilterThenPlot('notchIdeal2','./noisyimage2.bmp','notchIdeal',75,6,108) #u0=6,v0=108
  applyFilterThenPlot('notchGaussian2','./noisyimage2.bmp','notchGaussian',75,6,108)
  #u0=6,v0=108
  applyFilterThenPlot('notchButterworth2','./noisyimage2.bmp','notchButterworh',75,6,108,2)
  #u0=6,v0=108

  applyFilterThenPlot('idealBSF1-22_5','./noisyimage1.bmp','idealBSF',75,22.5)
  applyFilterThenPlot('idealBSF1-11_25','./noisyimage1.bmp','idealBSF',75,22.5/2)
  applyFilterThenPlot('idealBSF2-22_5','./noisyimage2.bmp','idealBSF',75,22.5)
  applyFilterThenPlot('idealBSF2-11_25','./noisyimage2.bmp','idealBSF',75,22.5/2)

  applyFilterThenPlot('gaussianBSF1-22_5','./noisyimage1.bmp','gaussianBSF',75,22.5)
  applyFilterThenPlot('gaussianBSF1-11_25','./noisyimage1.bmp','gaussianBSF',75,22.5/2)
  applyFilterThenPlot('gaussianBSF2-22_5','./noisyimage2.bmp','gaussianBSF',75,22.5)
  applyFilterThenPlot('gaussianBSF2-11_25','./noisyimage2.bmp','gaussianBSF',75,22.5/2)

  applyFilterThenPlot('butterworthBSF1-
  22_5','./noisyimage1.bmp','butterworthBSF',75,22.5,2)
  applyFilterThenPlot('butterworthBSF1-
  11_25','./noisyimage1.bmp','butterworthBSF',75,22.5/2,2)
  applyFilterThenPlot('butterworthBSF2-
  22_5','./noisyimage2.bmp','butterworthBSF',75,22.5,2)
  applyFilterThenPlot('butterworthBSF2-
  11_25','./noisyimage2.bmp','butterworthBSF',75,22.5/2,2)
  ```
  ```

```
applyFilterThenPlot('idealBPF1-22_5','./noisyimage1.bmp','idealBPF',75,22.5)
applyFilterThenPlot('idealBPF1-11_25','./noisyimage1.bmp','idealBPF',75,22.5/2)
applyFilterThenPlot('idealBPF2-22_5','./noisyimage2.bmp','idealBPF',75,22.5)
applyFilterThenPlot('idealBPF2-11_25','./noisyimage2.bmp','idealBPF',75,22.5/2/2)

applyFilterThenPlot('gaussianBPF1-22_5','./noisyimage1.bmp','gaussianBPF',75,22.5)
applyFilterThenPlot('gaussianBPF1-11_25','./noisyimage1.bmp','gaussianBPF',75,22.5/2)
applyFilterThenPlot('gaussianBPF2-22_5','./noisyimage2.bmp','gaussianBPF',75,22.5)
applyFilterThenPlot('gaussianBPF2-11_25','./noisyimage2.bmp','gaussianBPF',75,22.5/2)

applyFilterThenPlot('butterworthBPF1-
22_5','./noisyimage1.bmp','butterworthBPF',75,22.5,2)
applyFilterThenPlot('butterworthBPF1-
11_25','./noisyimage1.bmp','butterworthBPF',75,22.5/2,2)
applyFilterThenPlot('butterworthBPF2-
22_5','./noisyimage2.bmp','butterworthBPF',75,22.5,2)
applyFilterThenPlot('butterworthBPF2-
11_25','./noisyimage2.bmp','butterworthBPF',75,22.5/2,2)
```

- در فیلتر ها با توجه به داکیمونت اعمال شده اند مثلا idealBPF1-22_5 ، فیلتر idealBPF است با اندازه pi/4

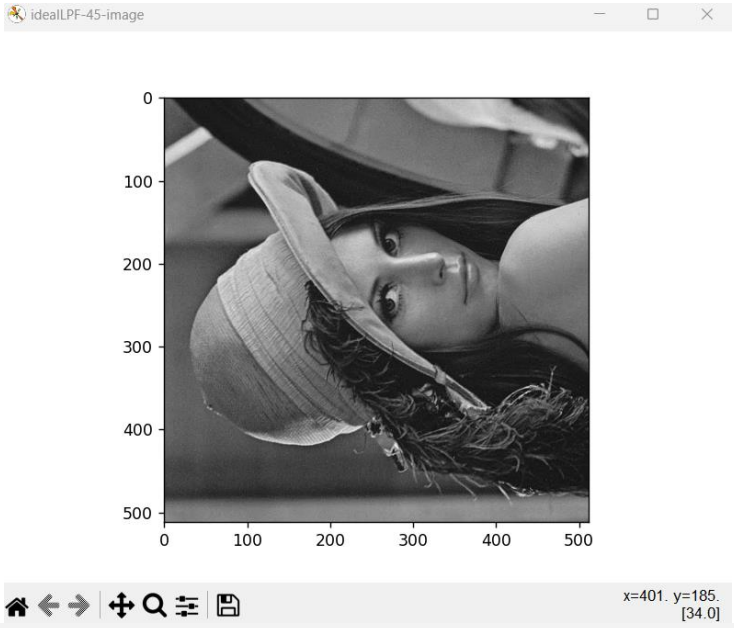
تحلیل ها :

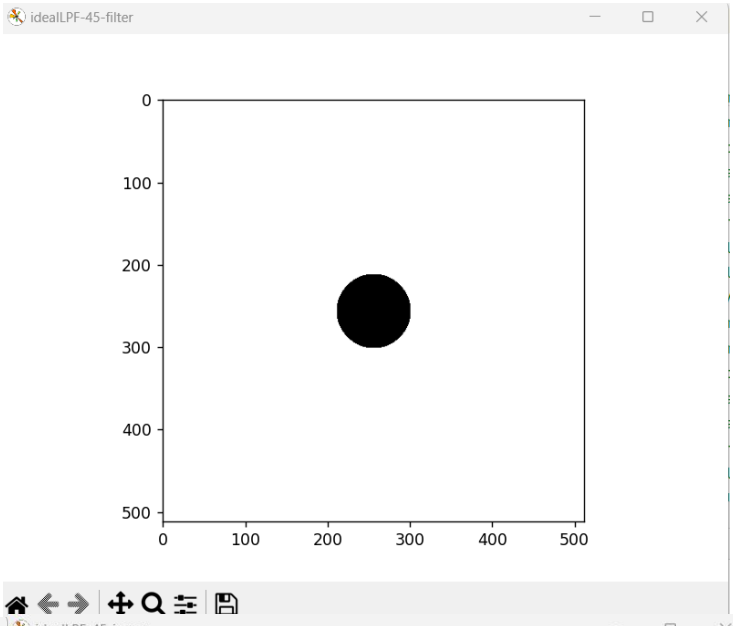
- تصویر lena و ideal مختف با فرکانس قطع pi/4,pi/2,2pi/3
  - این فیلتر برای حفظ فرکانس های پایین و حذف فرکانس های بالا است
- تصویر lena و gaussian مختف با فرکانس قطع pi/4,pi/2,2pi/3
  - در این فیلتر عکس blur می شود و برای حذف نویز استفاده می شود
- تصویر lena و butterworth مختف با فرکانس قطع pi/4,pi/2,2pi/3
  - این فیلتر پاسخ مسطح فرکانس است

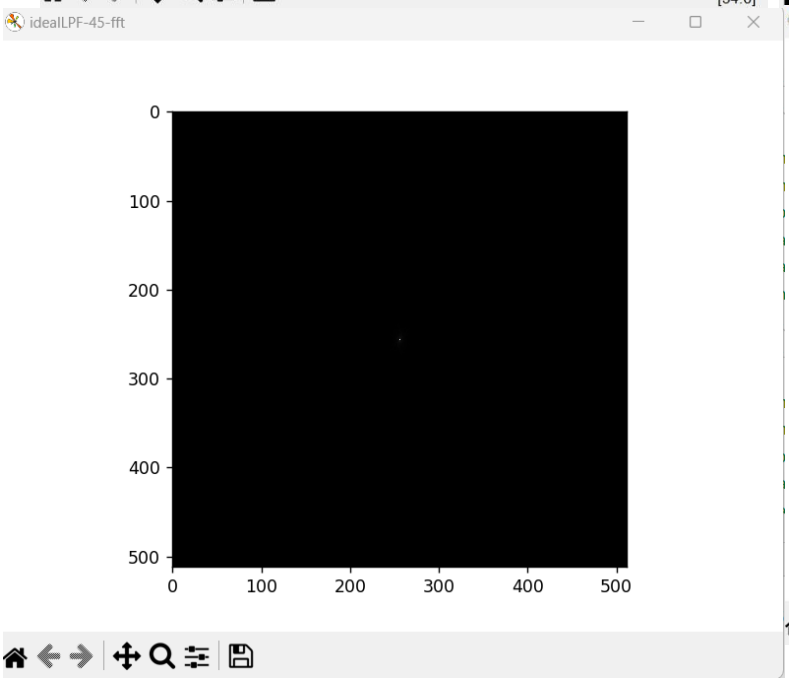
- تصویر noisyimage1 و noisyimage2 برای انجام دو کار :
  - حذف نویز
    - باید عکس را در حوزه فرکانس مشاهده کرد تا بتوان محل نویز را پیدا کرد و به دو فیلتر زیر ورودی داد
    - Notch filter
      - در اینجا این فیلتر بهتر عمل می کند
    - Band stop filter
  - نمایش نویز
    - از Band pass filter ها و Band stop و Reverse Notch فیلتر ها نویز را استخراج کرده و نمایش داده.
  - Notch فیلار برای پاس دادن دامنه خاصی از فرکانس است

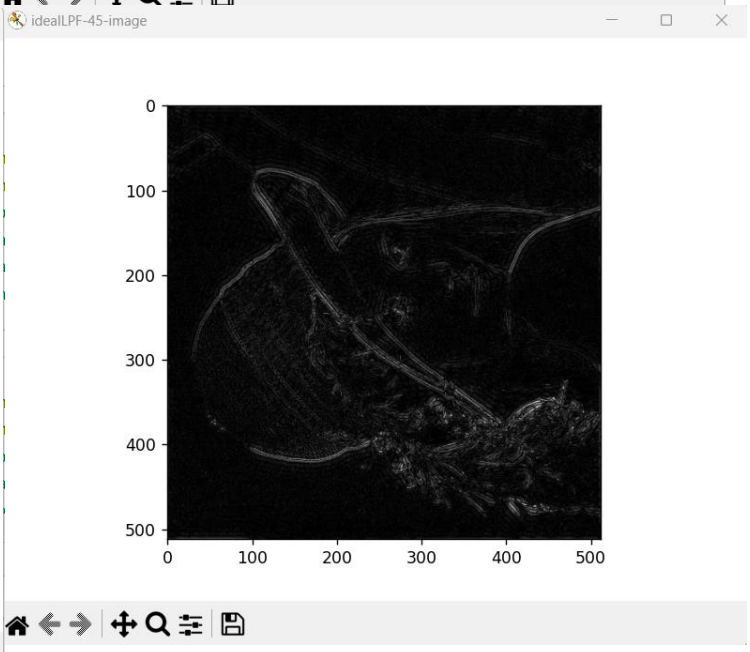
توضیح عکس ها :

- عکس ها به صورت چهار تایی برای هر بخش خواسته شده در داکیومنت تولید شده اند
- applyFilterThenPlot('idealLPF-45','./lena.bmp','idealLPF',45)
- مثلا دستور بالا
  - ابتدا خود عکس
  - سپس عکس فیلتر
  - سپس عکس در حوزه فرکانس
  - سپس عکس فیلتر شده
- تمام چیز هایی که در داکیومنت خواسته شده در کد apply ها اعمال شده.
- هر عکس را که باز می کنید بالا سمت چپ آن نامش نوشته شده مه چه فیلتر و چه زاویه و ... دارد.
- نیمی از عکس ها زیر آمده است اما به دلیل زمان بالا تمامی عکس های تولید شده در دایرکتوری جداگونه ای به همین تریتب 4 تایی گذاشته شده اند

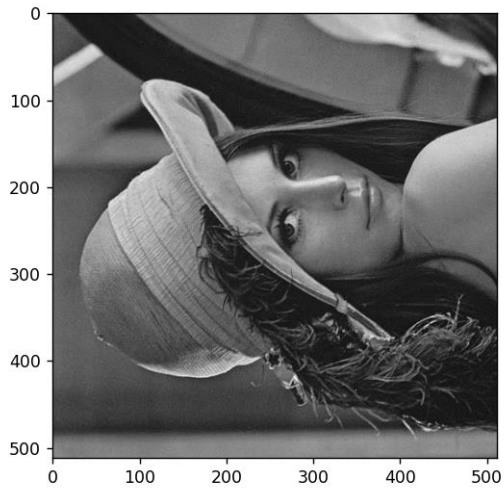

idealLPF-45-image
0
100
200
300
400
500
x=401. y=185.
[34.0]


idealLPF-45-filter
0
100
200
300
400
500

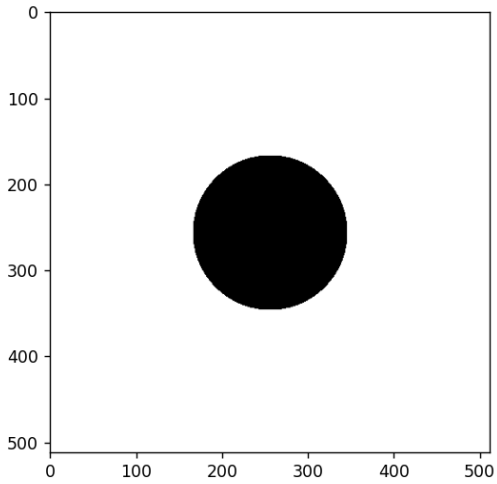

idealLPF-45-fft
0
100
200
300
400
500


idealLPF-45-image
0
100
200
300
400
500

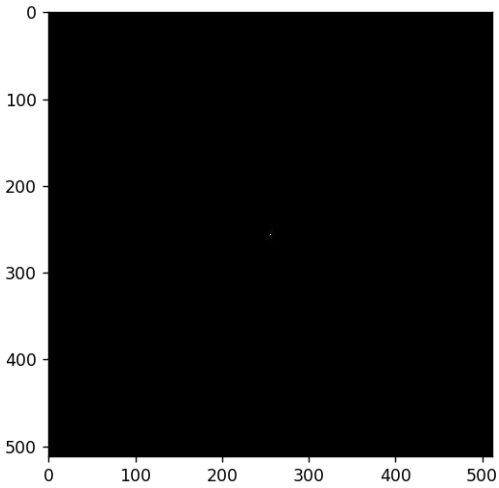
idealLPF-90-image



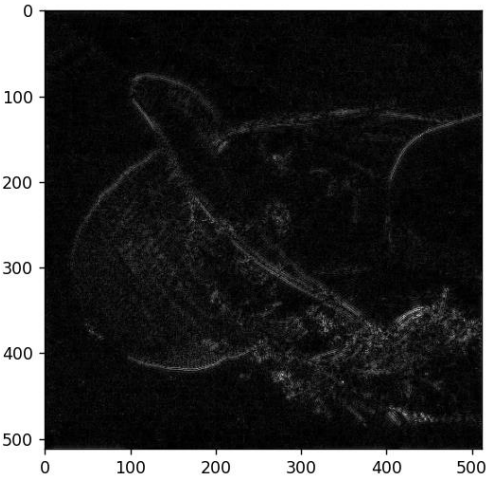
idealLPF-90-filter



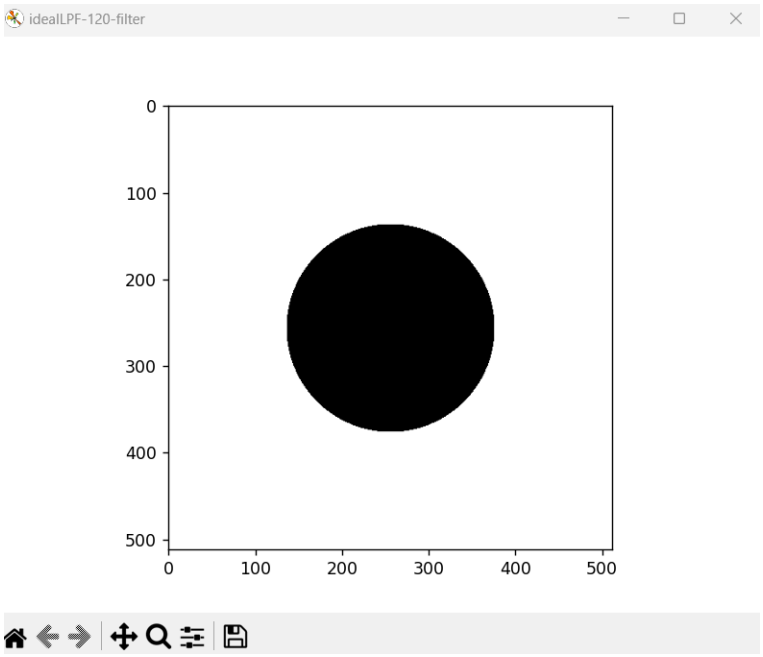
idealLPF-90-fft

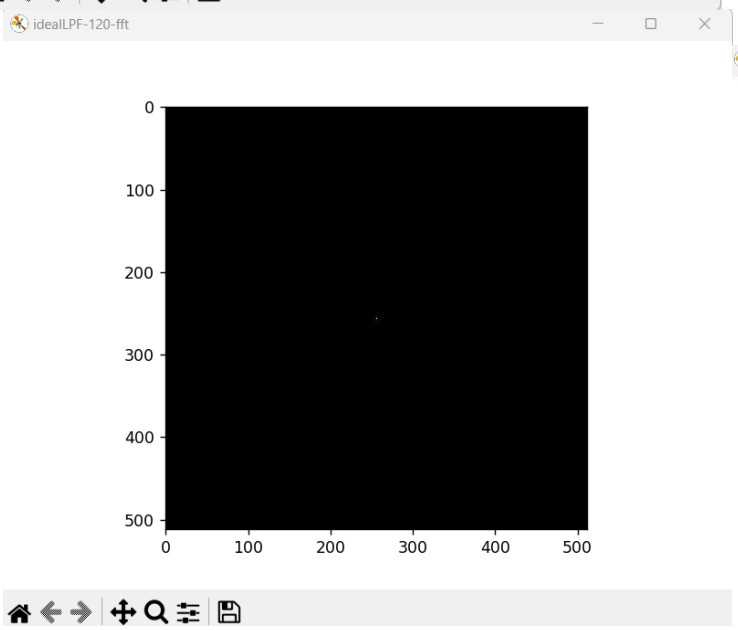


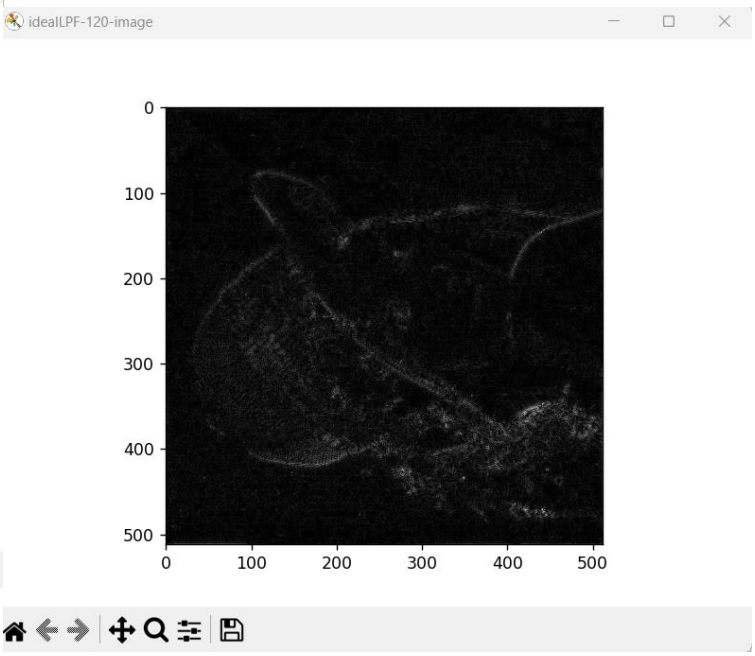
idealLPF-90-image



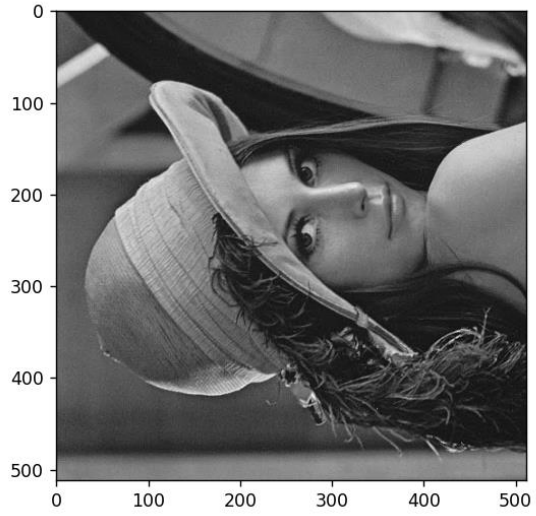

idealLPF-120-image
0
100
200
300
400
500


idealLPF-120-filter
0
100
200
300
400
500

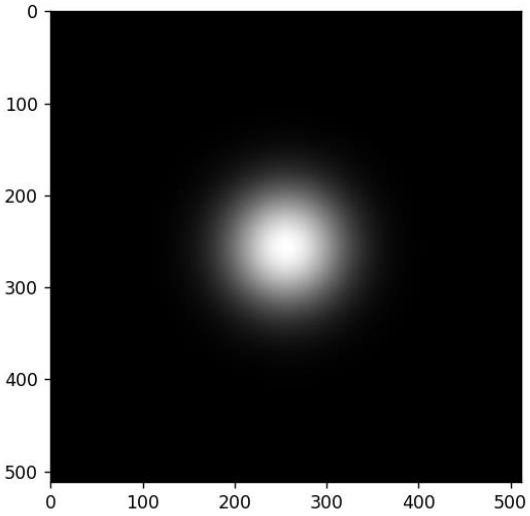

idealLPF-120-fft
0
100
200
300
400
500


idealLPF-120-image
0
100
200
300
400
500

gaussianLPF-45-image

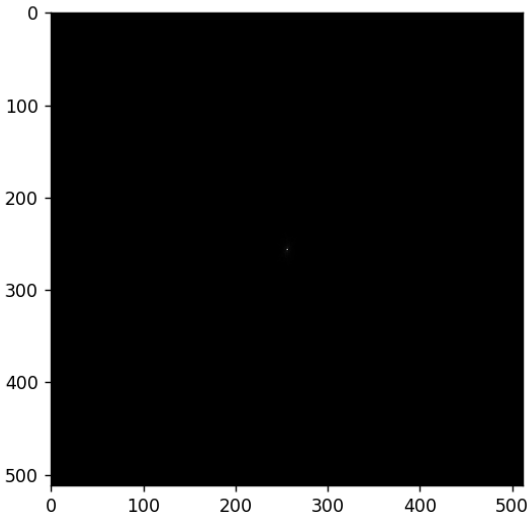


gaussianLPF-45-filter

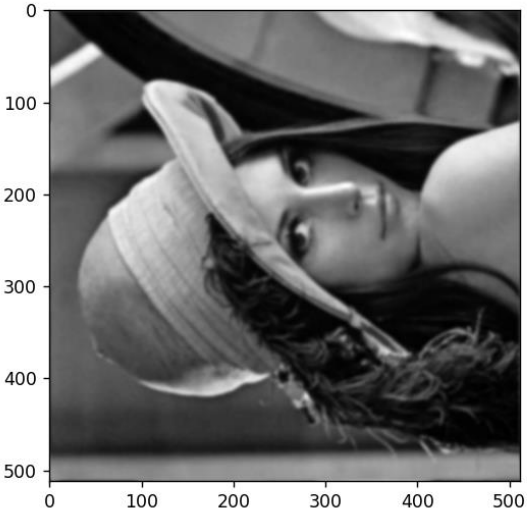


x=129. y=−0.5
[2.e-09]

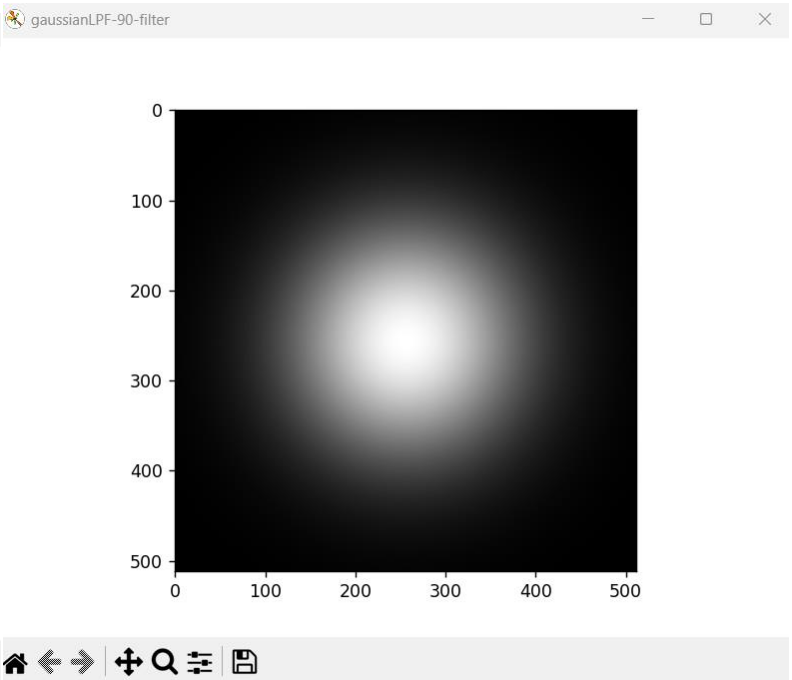
gaussianLPF-45-fft

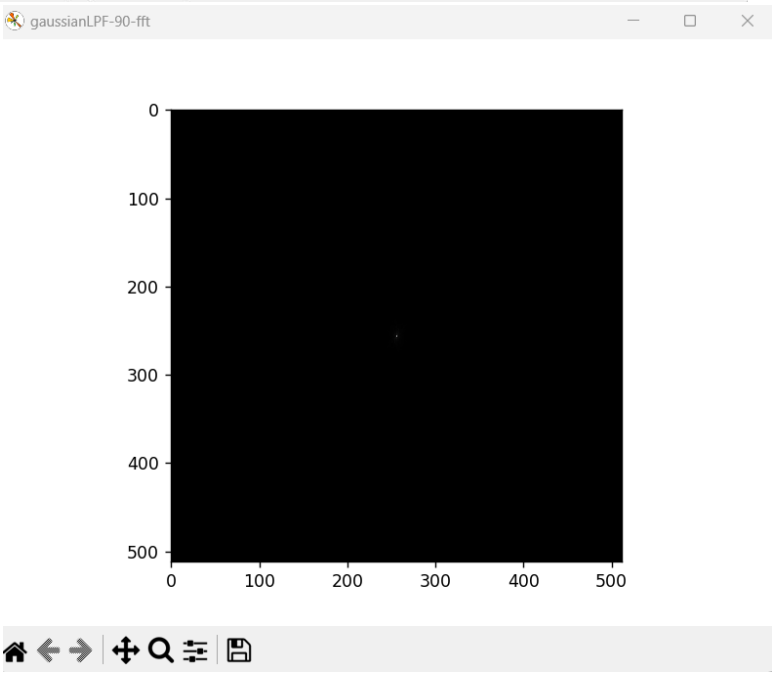


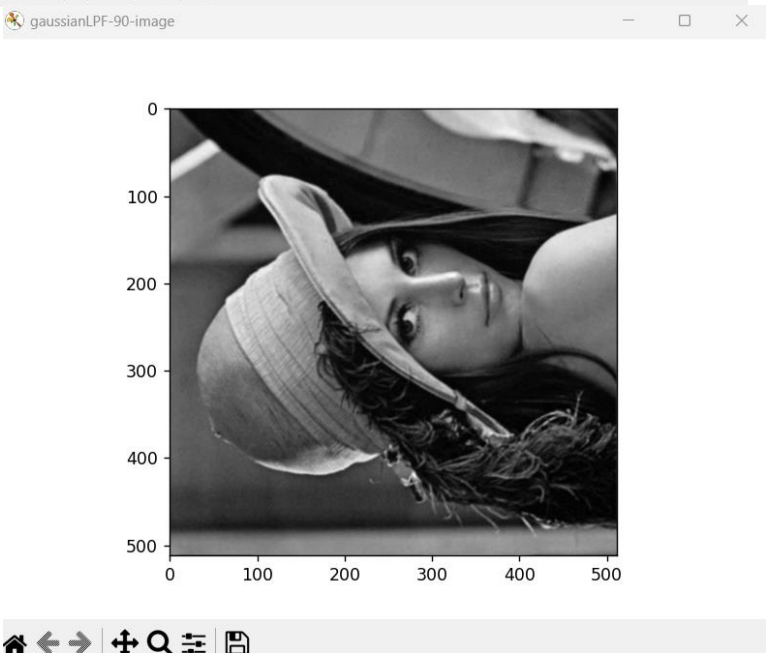
gaussianLPF-45-image



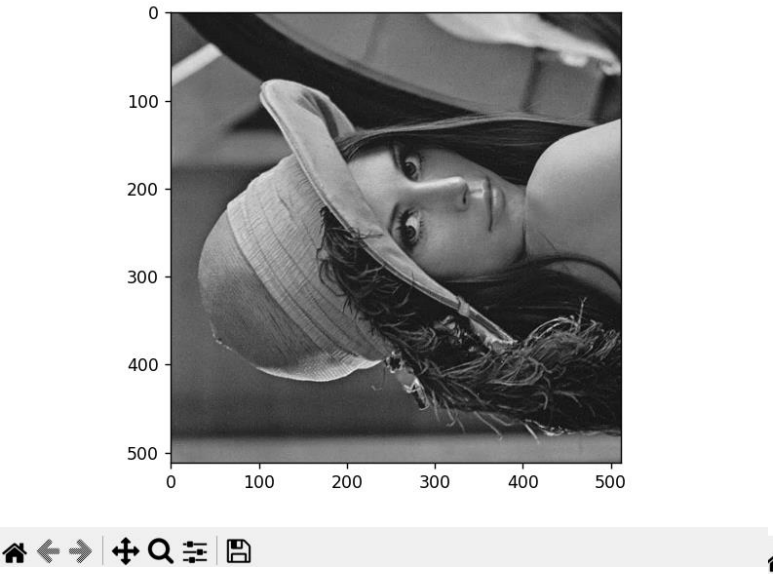

gaussianLPF-90-image
0
100
200
300
400
500


gaussianLPF-90-filter
0
100
200
300
400
500

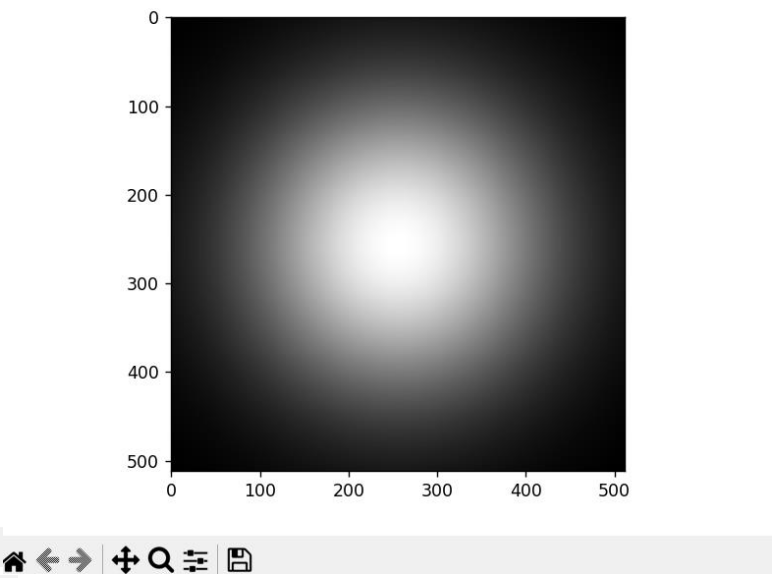

gaussianLPF-90-fft
0
100
200
300
400
500


gaussianLPF-90-image
0
100
200
300
400
500

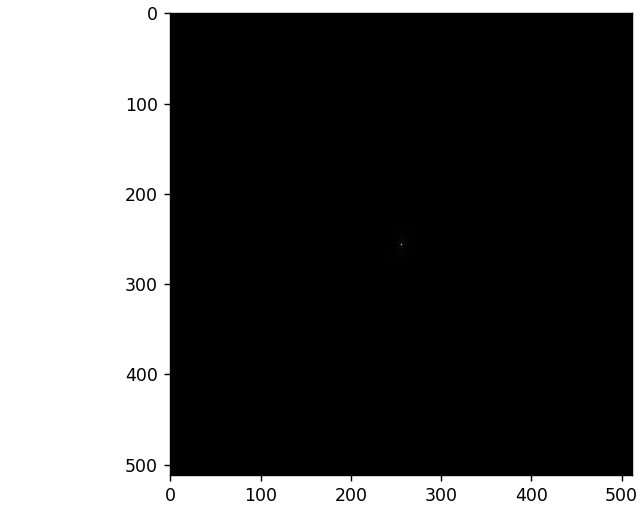
gaussianLPF-120-image



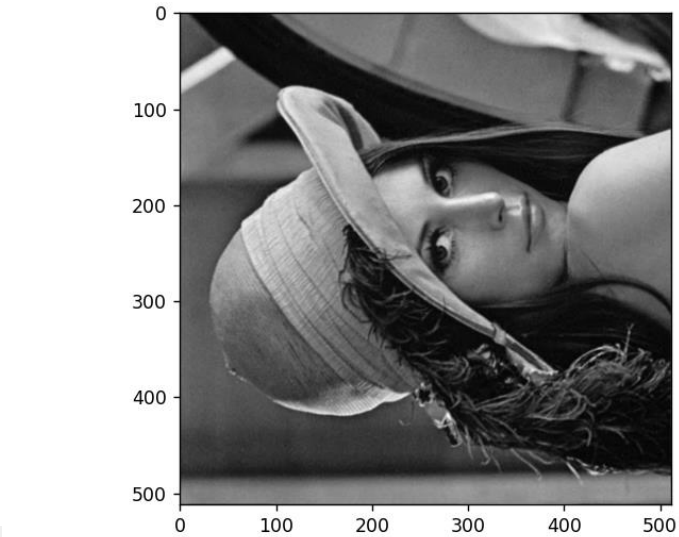
gaussianLPF-120-filter



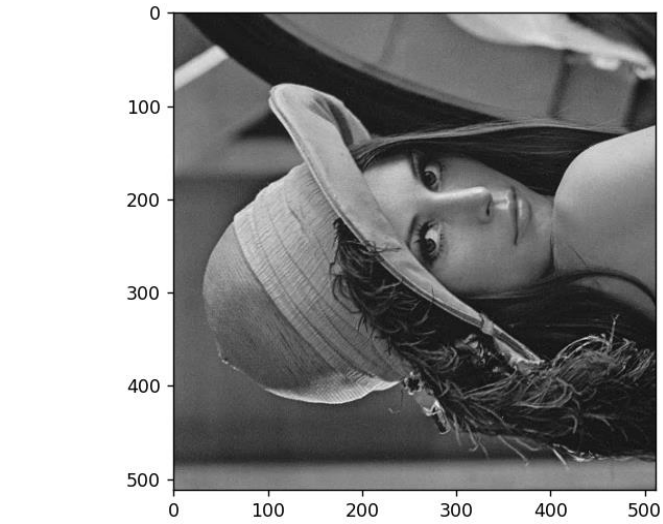
gaussianLPF-120-fft



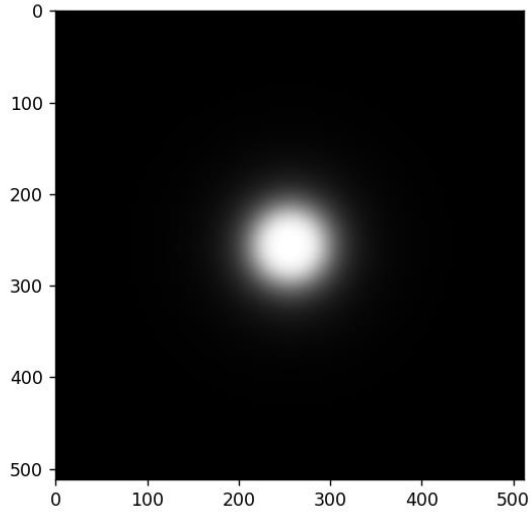
gaussianLPF-120-image



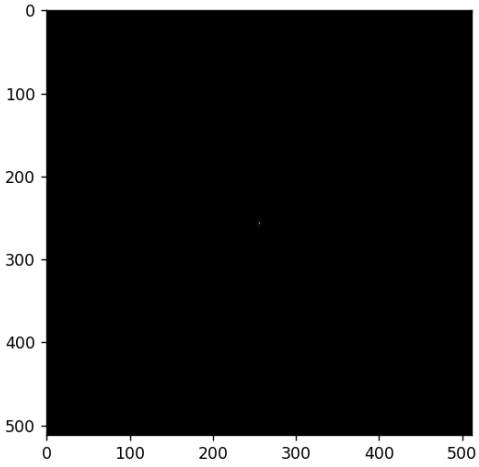
butterworthLPF-45-image



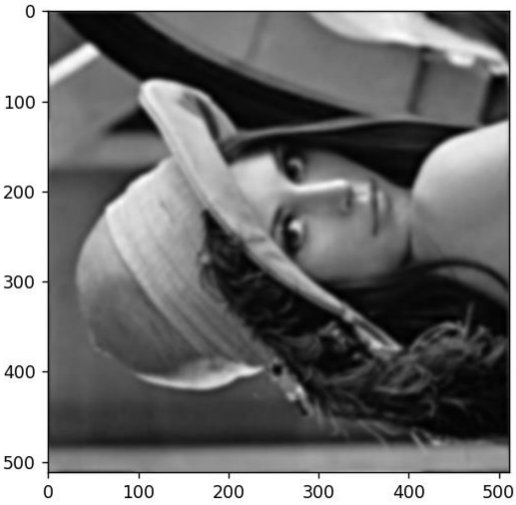
butterworthLPF-45-filter



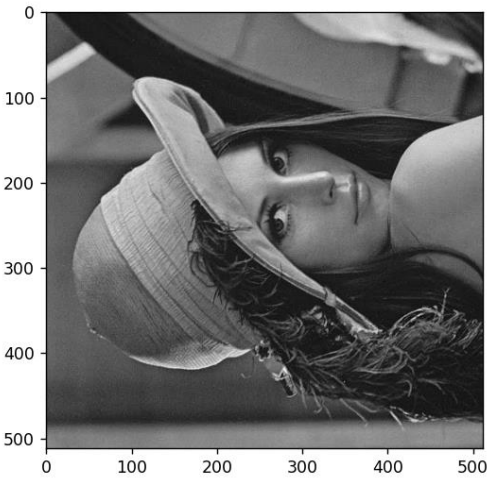
butterworthLPF-45-fft



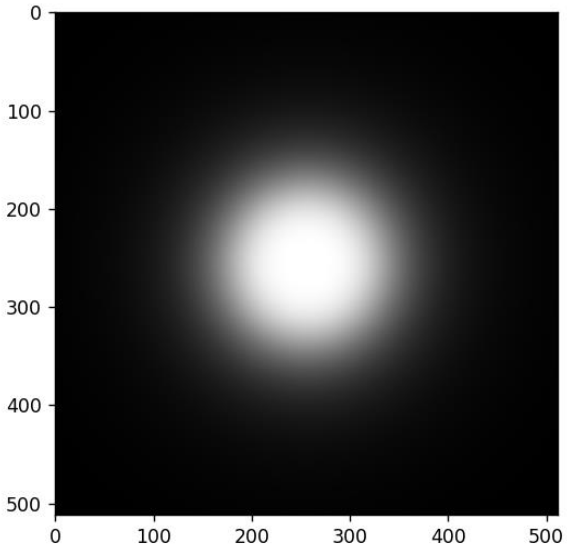
butterworthLPF-45-image



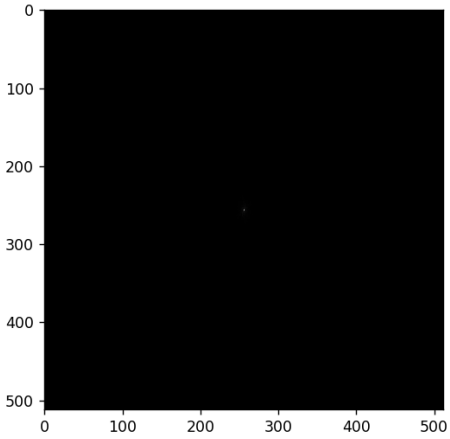
butterworthLPF-90-image



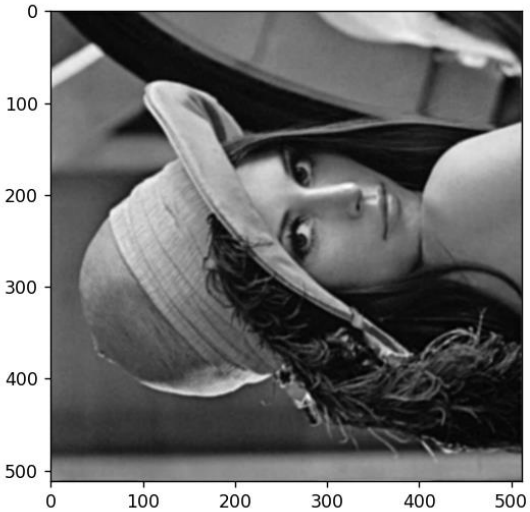
butterworthLPF-90-filter



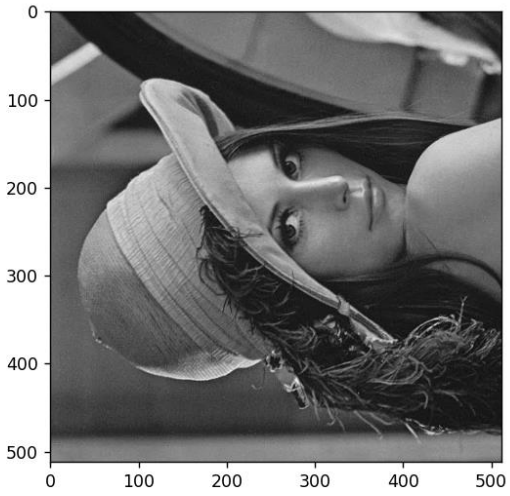
butterworthLPF-90-fft



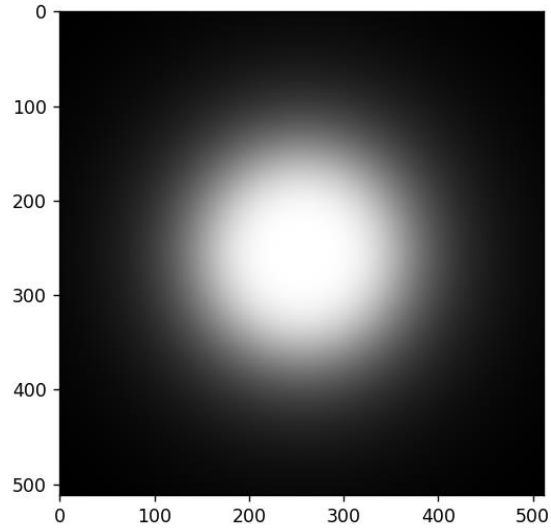
butterworthLPF-90-image



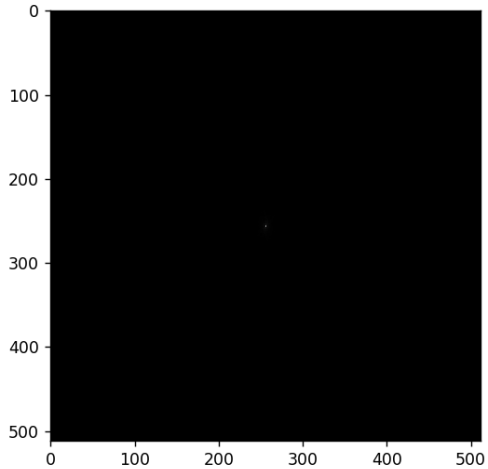
butterworthLPF-120-image



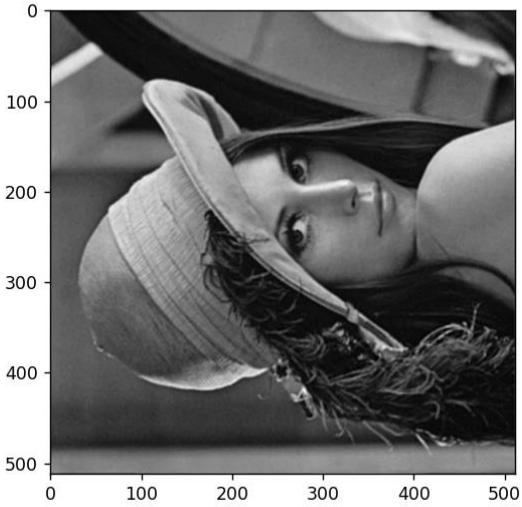
butterworthLPF-120-filter



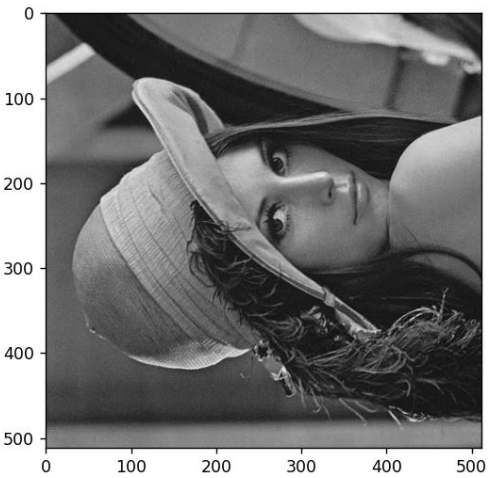
butterworthLPF-120-fft



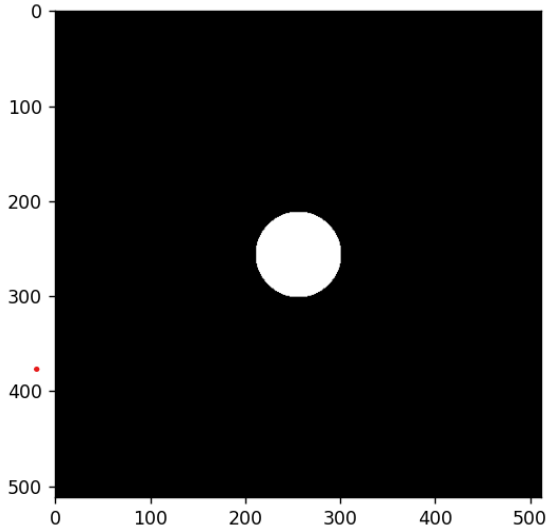
butterworthLPF-120-image



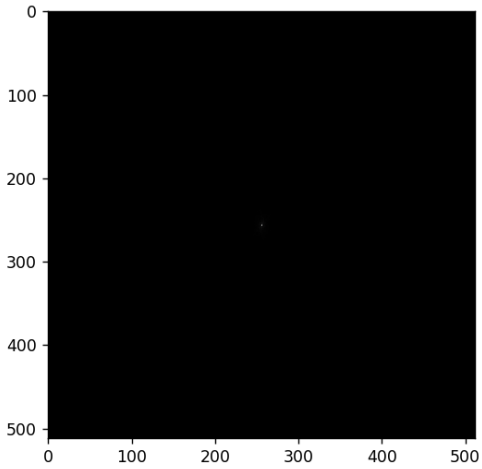
idealHPF-45-image



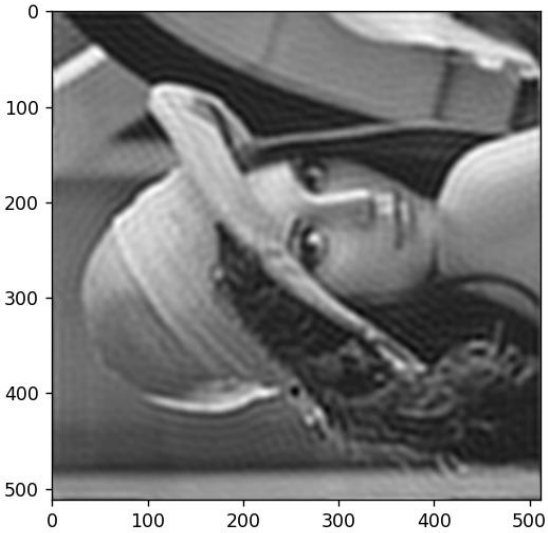
idealHPF-45-filter



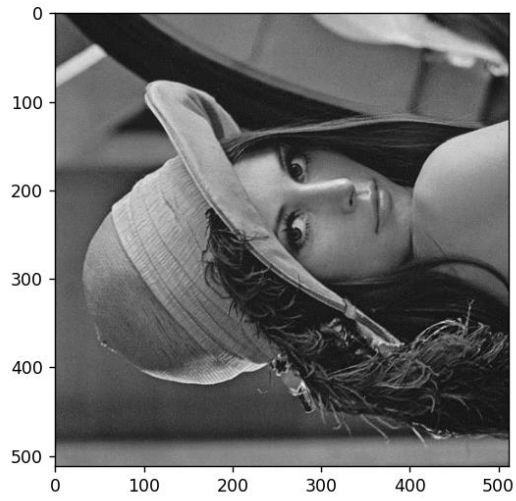
idealHPF-45-fft



idealHPF-45-image

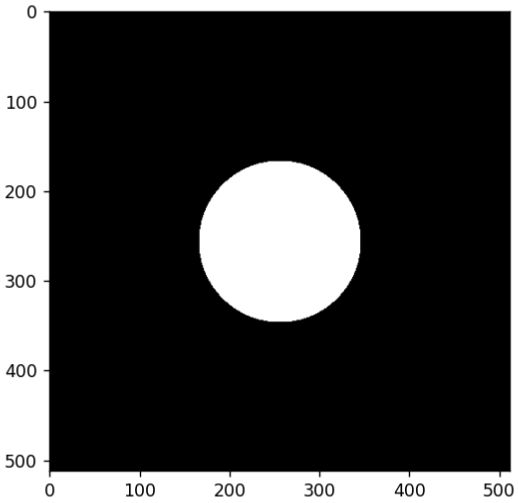


idealHPF-90-image

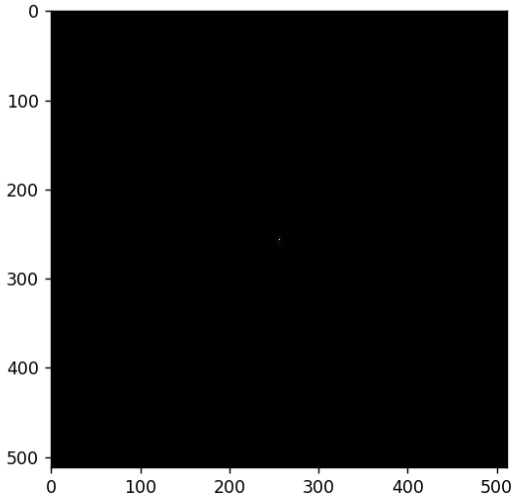


x=300. y=99.
[86.0]

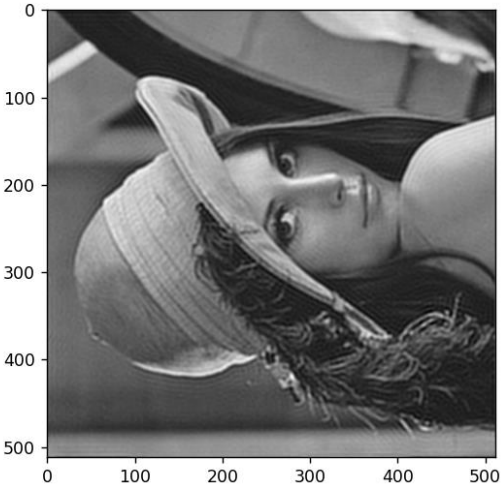
idealHPF-90-filter



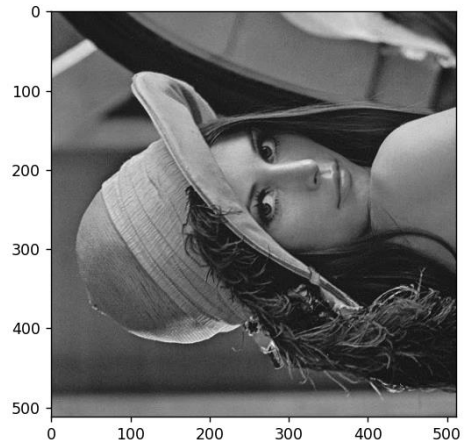
idealHPF-90-fft



idealHPF-90-image

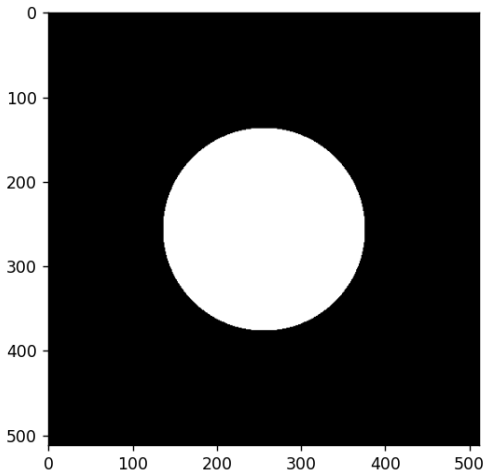


idealHPF-120-image

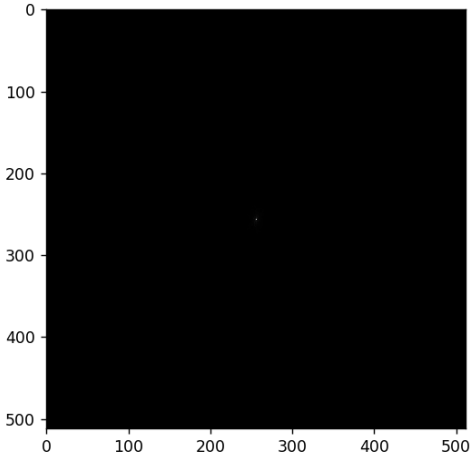


x=502. y=35.
[40.0]

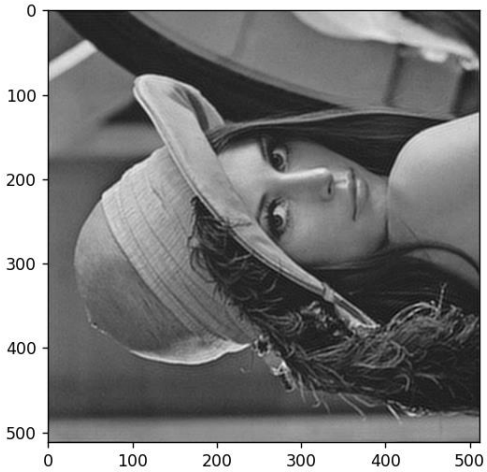
idealHPF-120-filter



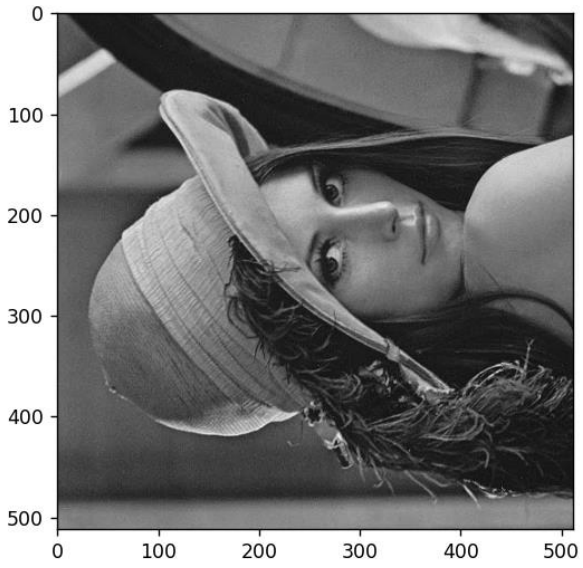
idealHPF-120-fft



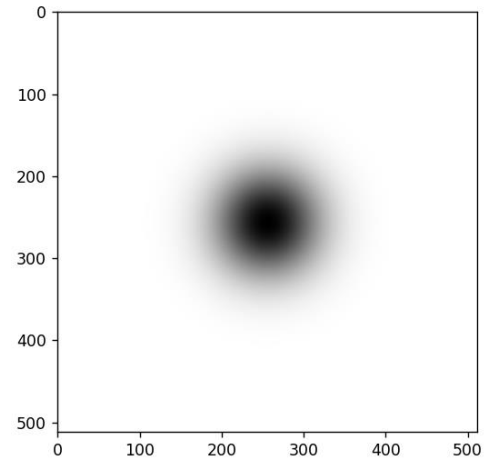
idealHPF-120-image



gaussianHPF-45-image

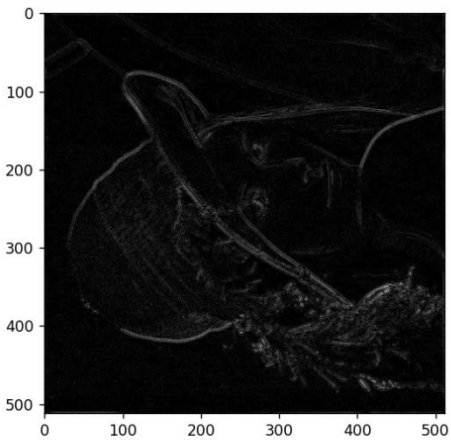


gaussianHPF-45-filter

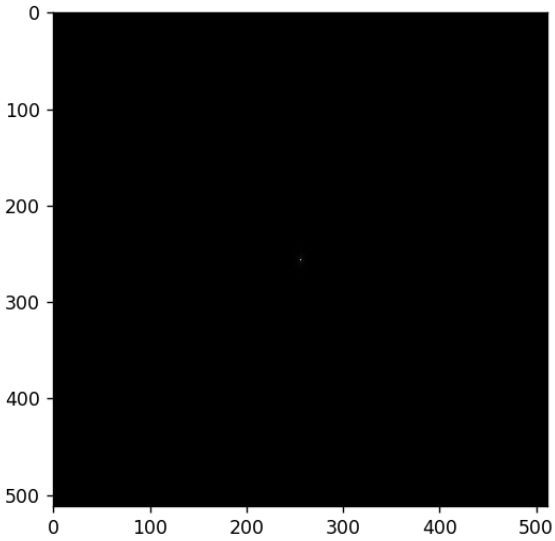


x=278. y=56.
[1.00]

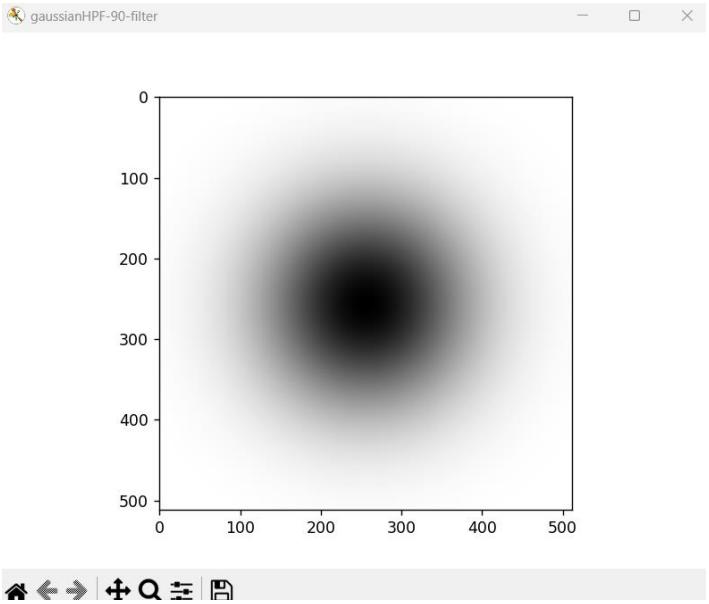
gaussianHPF-45-image

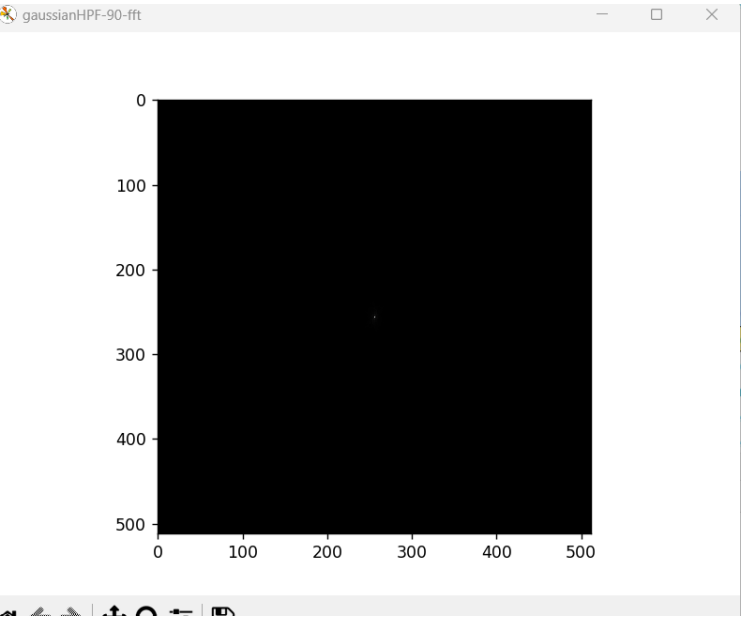


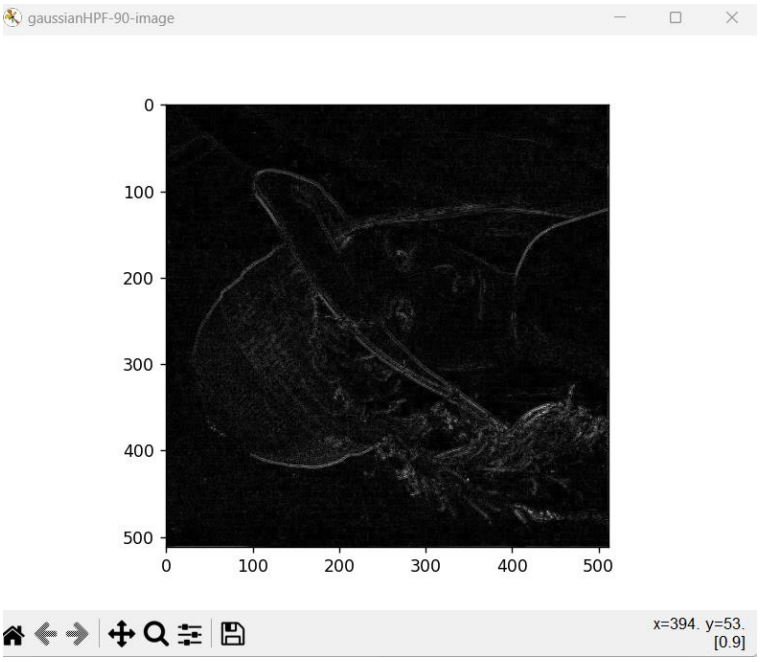
gaussianHPF-45-fft



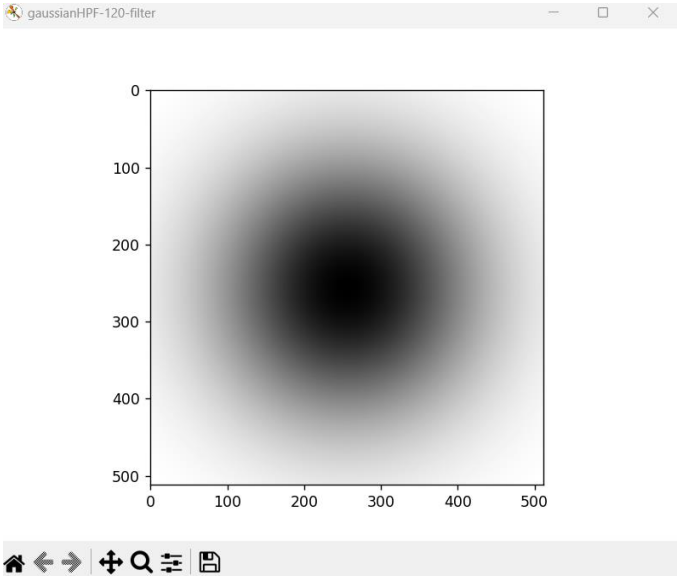

gaussianHPF-90-image
0
100
200
300
400
500
0
100
200
300
400
500

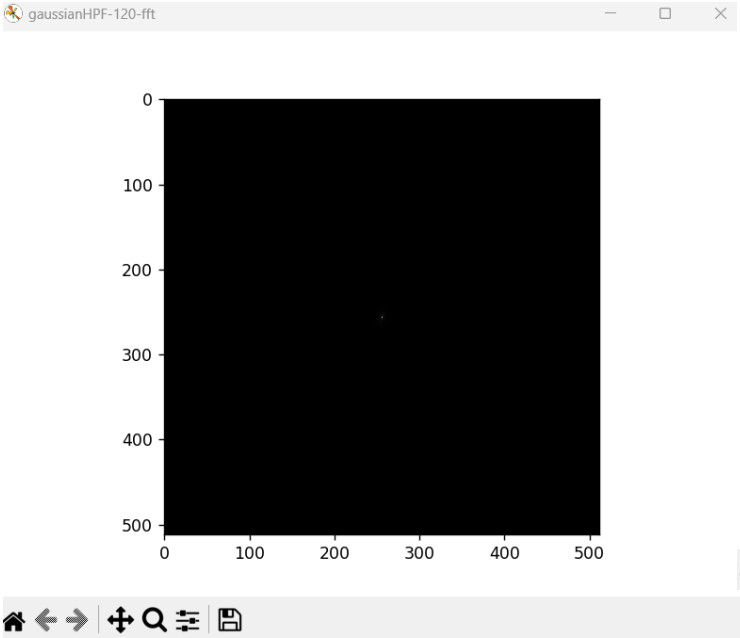

gaussianHPF-90-filter
0
100
200
300
400
500
0
100
200
300
400
500

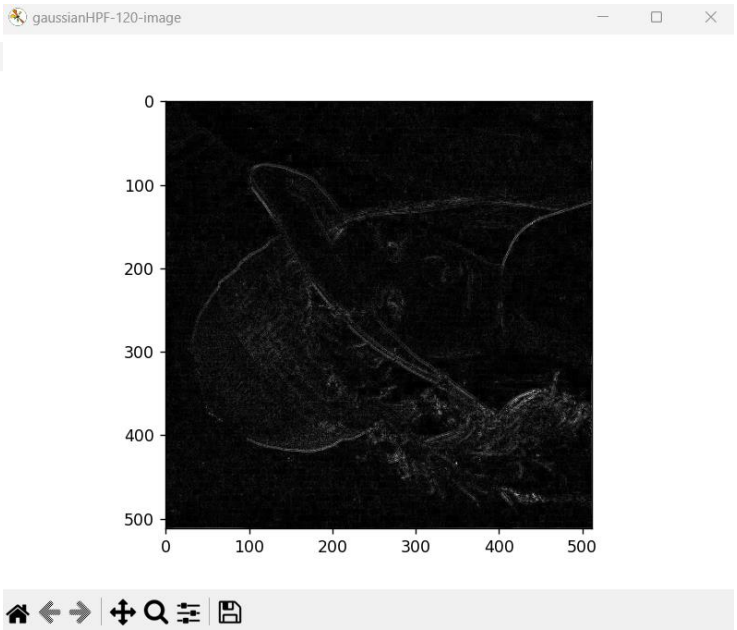

gaussianHPF-90-fft
0
100
200
300
400
500
0
100
200
300
400
500


gaussianHPF-90-image
0
100
200
300
400
500
0
100
200
300
400
500
x=394. y=53.
[0.9]

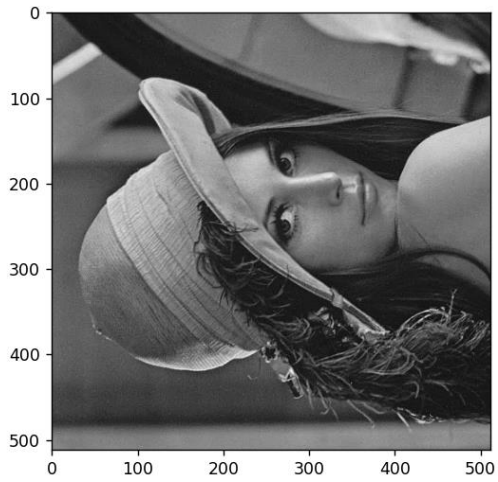

gaussianHPF-120-image
0
100
200
300
400
500
0
100
200
300
400
500


gaussianHPF-120-filter
0
100
200
300
400
500
0
100
200
300
400
500

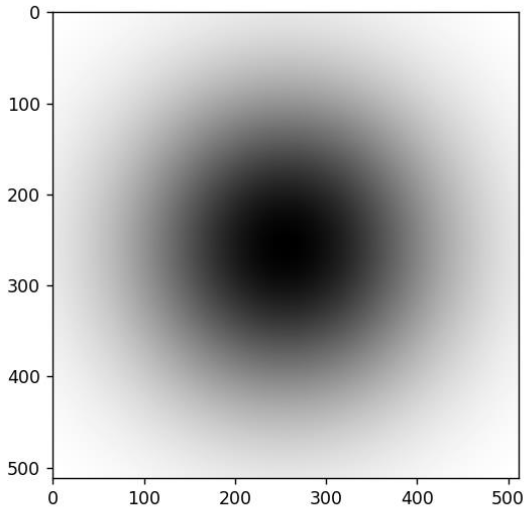

gaussianHPF-120-fft
0
100
200
300
400
500
0
100
200
300
400
500


gaussianHPF-120-image
0
100
200
300
400
500
0
100
200
300
400
500

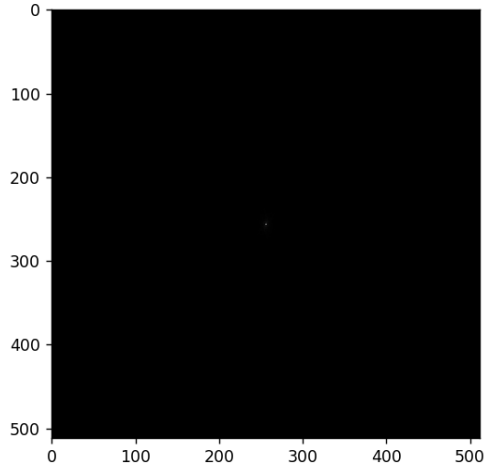
gaussianHPF-120-image



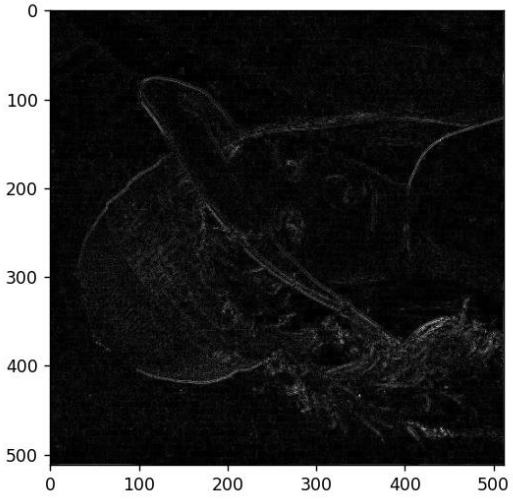
gaussianHPF-120-filter



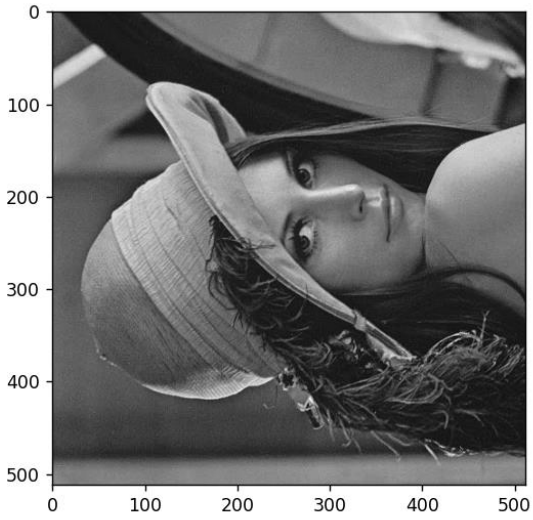
gaussianHPF-120-fft



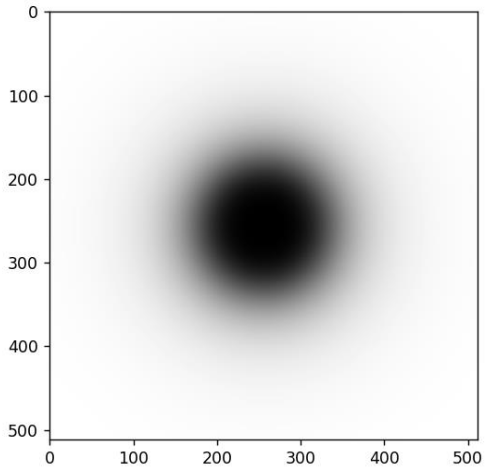
gaussianHPF-120-image



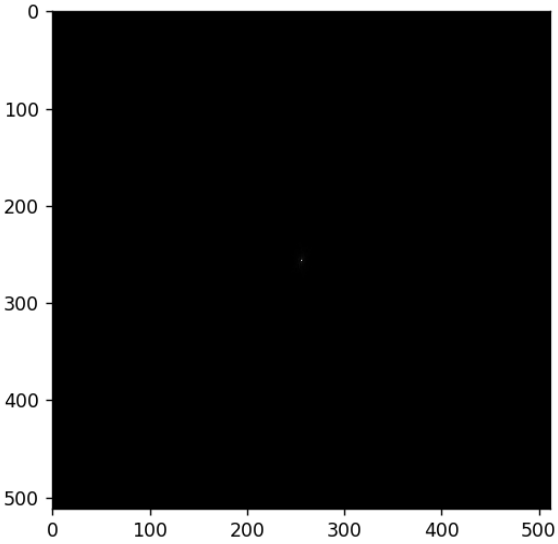
butterworthHPF-90-image



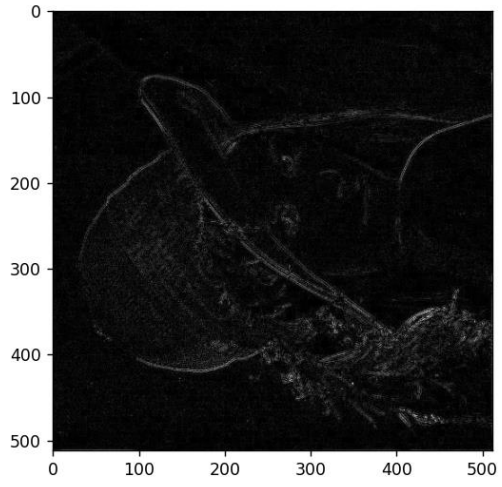
butterworthHPF-90-filter



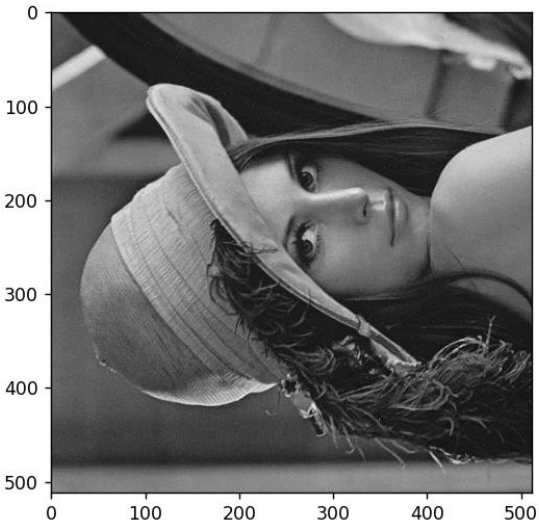
butterworthHPF-90-fft



butterworthHPF-90-image

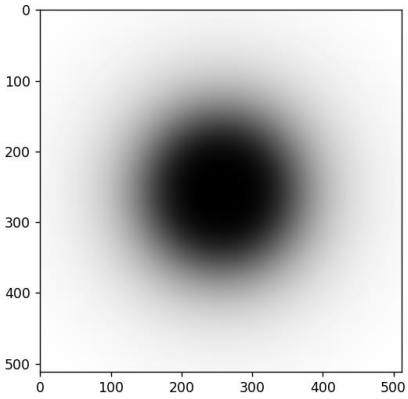


butterworthHPF-120-image

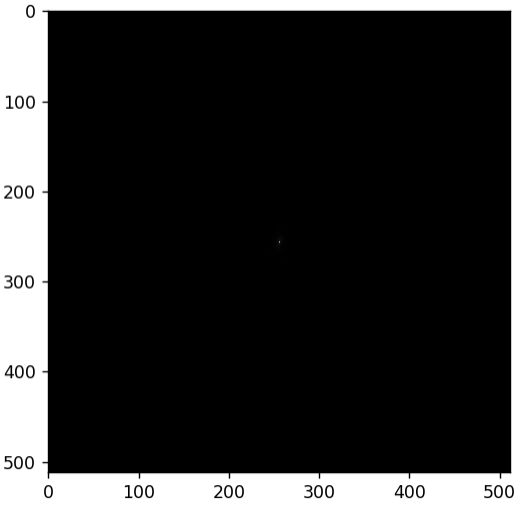


x=205. y=26.
[124.0]

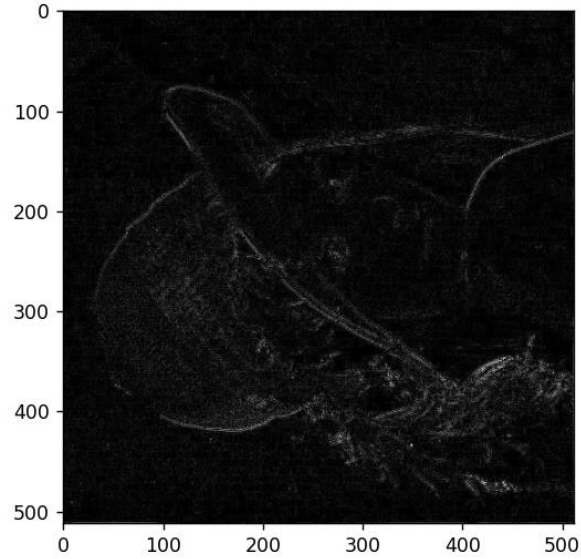
butterworthHPF-120-filter



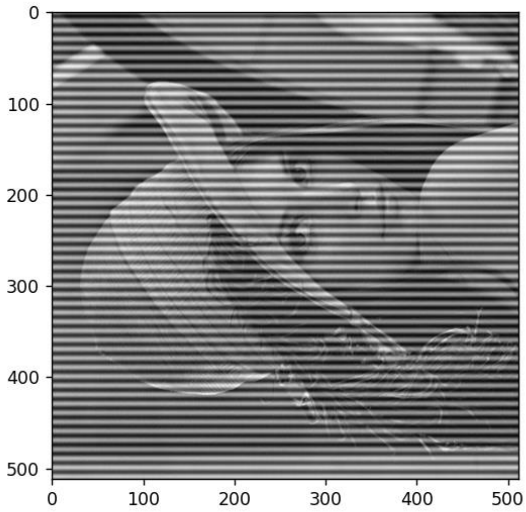
butterworthHPF-120-fft



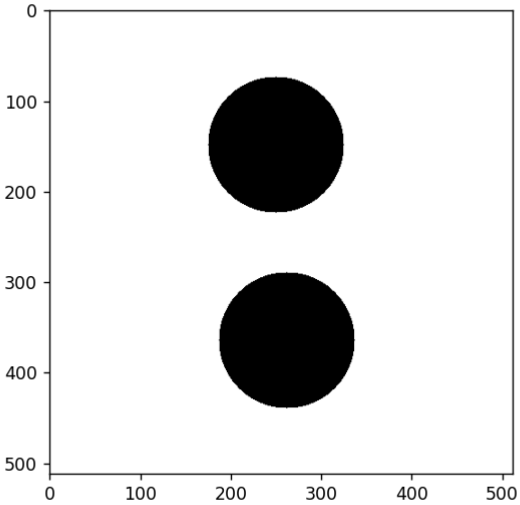
butterworthHPF-120-image



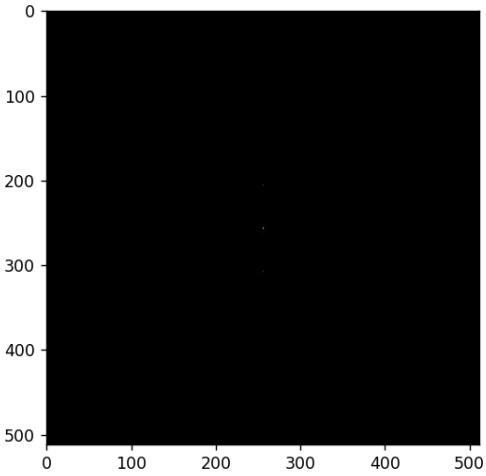
notchIdeal1-image



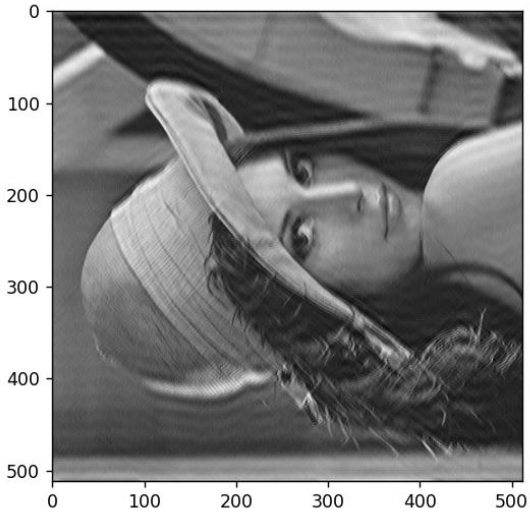
notchIdeal1-filter



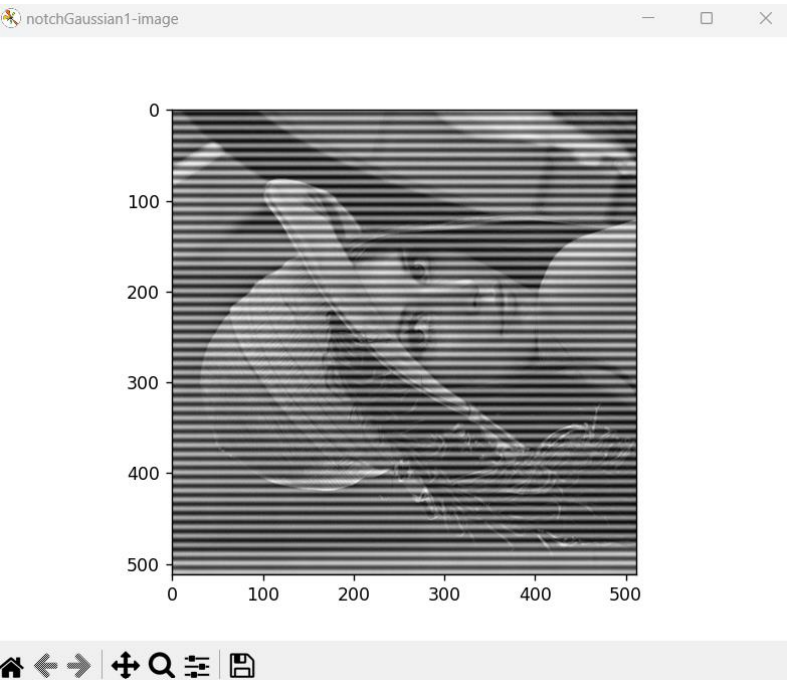
notchIdeal1-fft

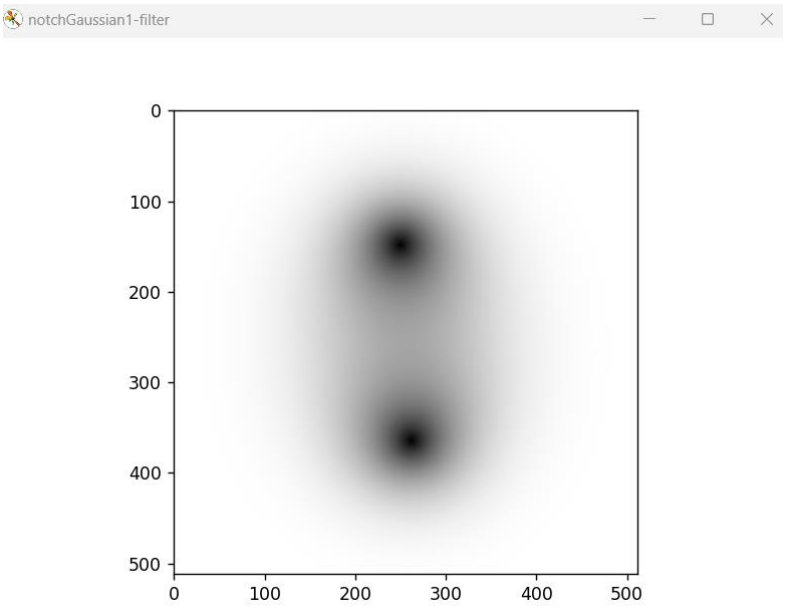


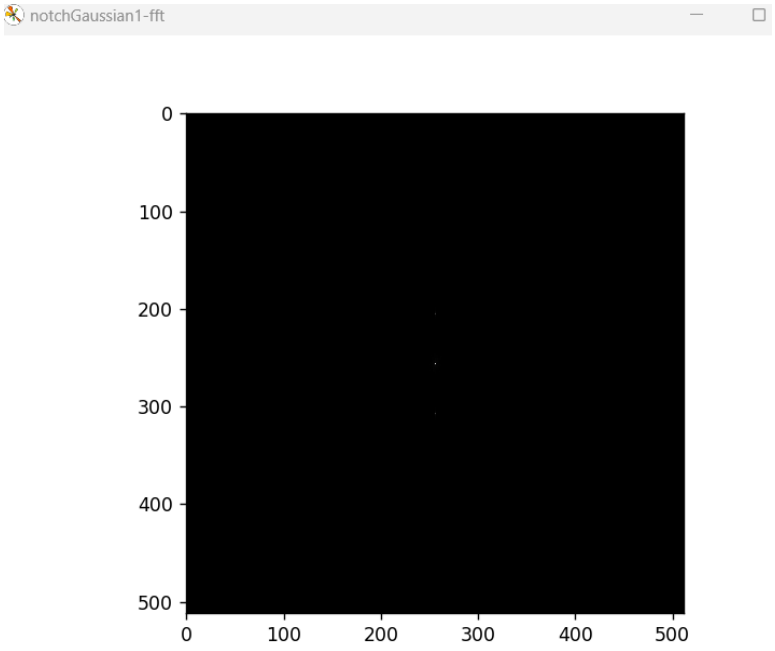
notchIdeal1-image

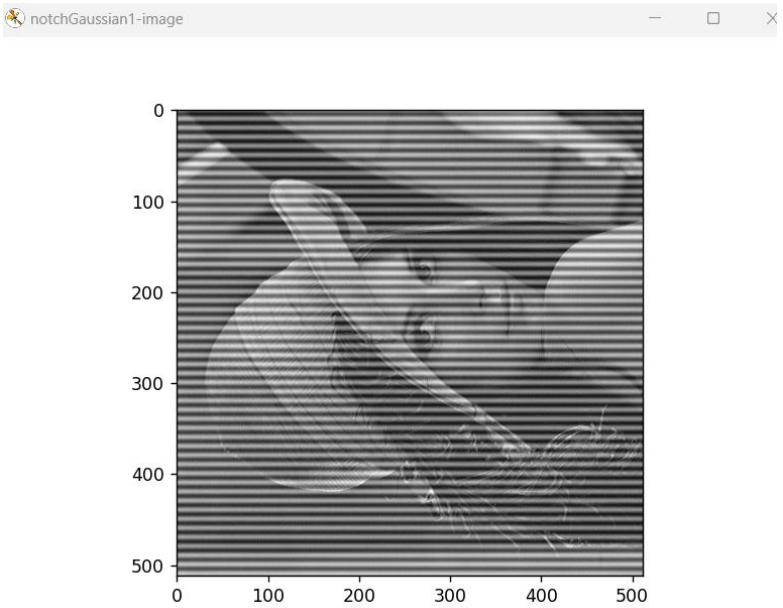


x=392. y=100.
[127.3]

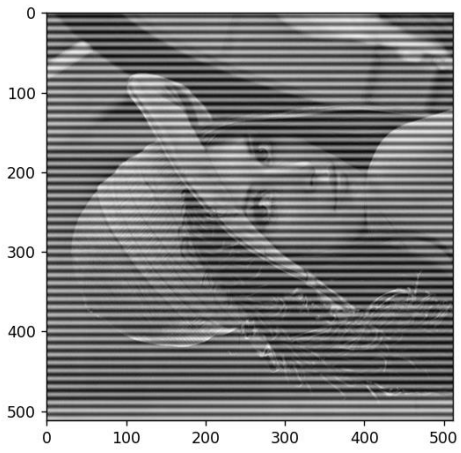

notchGaussian1-image
0
100
200
300
400
500
0
100
200
300
400
500


notchGaussian1-filter
0
100
200
300
400
500
0
100
200
300
400
500

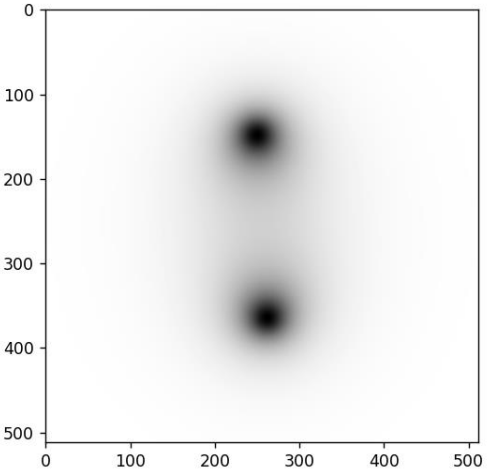

notchGaussian1-fft
0
100
200
300
400
500
0
100
200
300
400
500


notchGaussian1-image
0
100
200
300
400
500
0
100
200
300
400
500

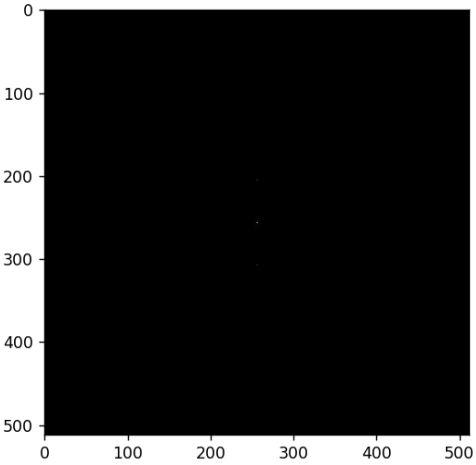
notchButterworh1-image



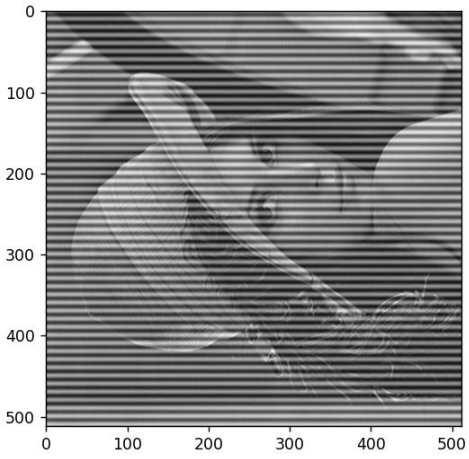
notchButterworh1-filter

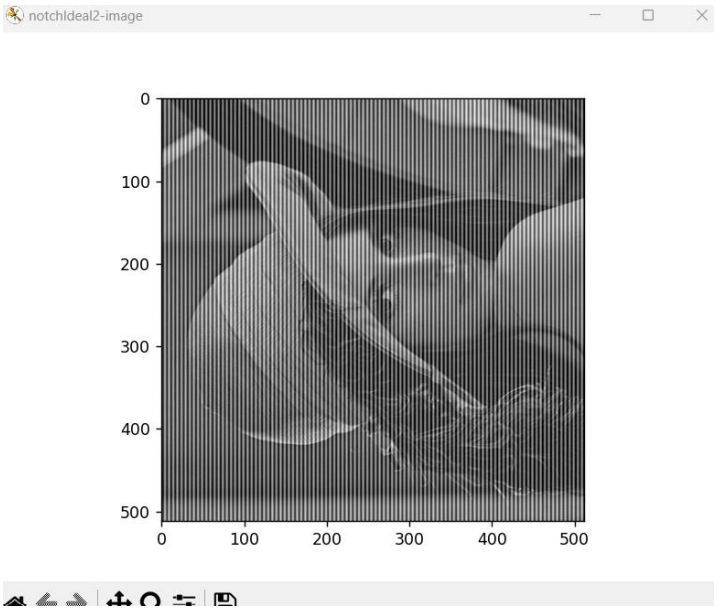


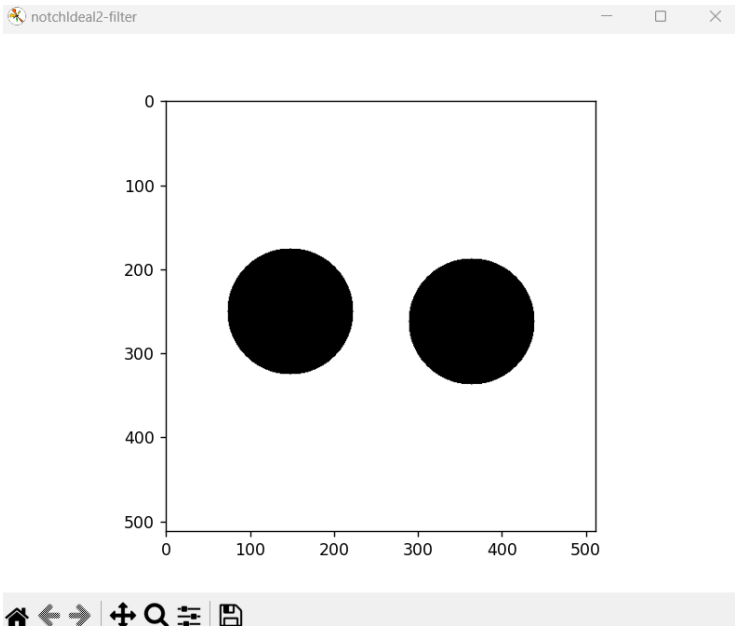
notchButterworh1-fft

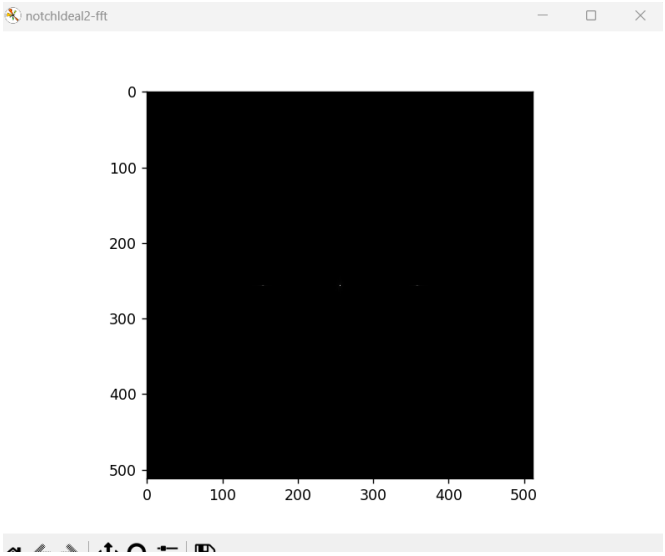


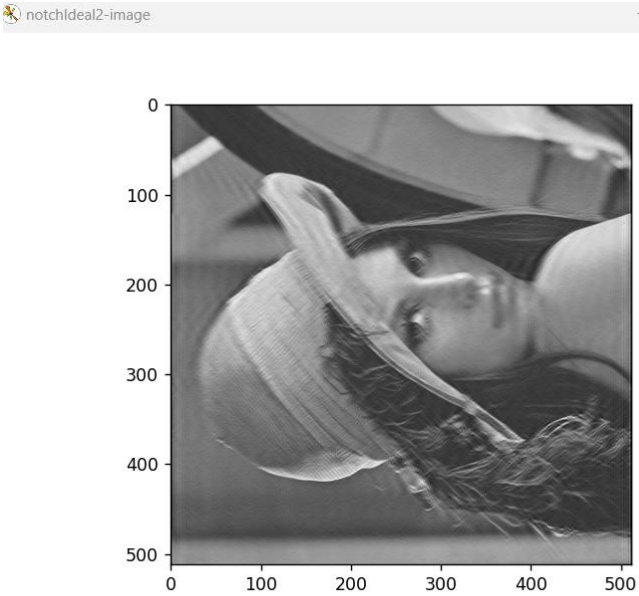
notchButterworh1-image



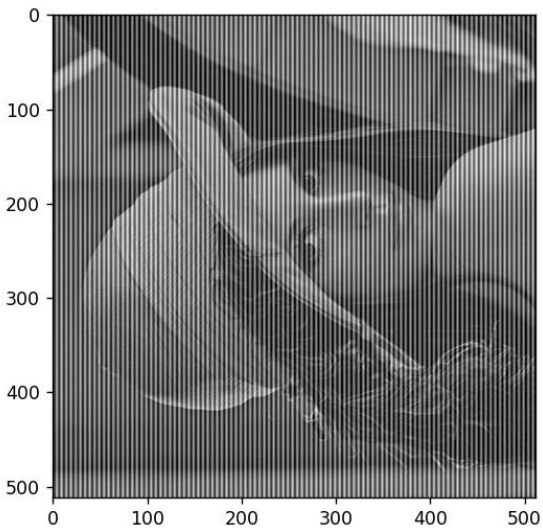

notchIdeal2-image
0
100
200
300
400
500


notchIdeal2-filter
0
100
200
300
400
500

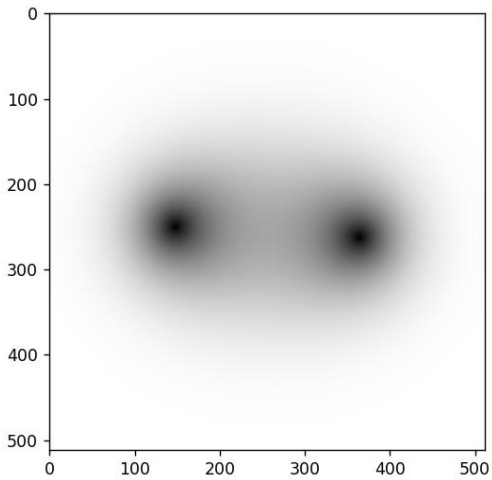

notchIdeal2-fft
0
100
200
300
400
500


notchIdeal2-image
0
100
200
300
400
500

notchGaussian2-image

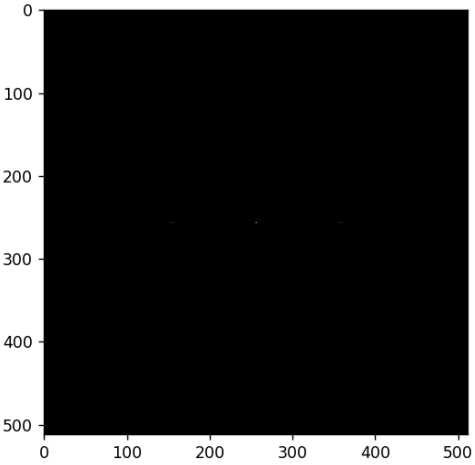


notchGaussian2-filter

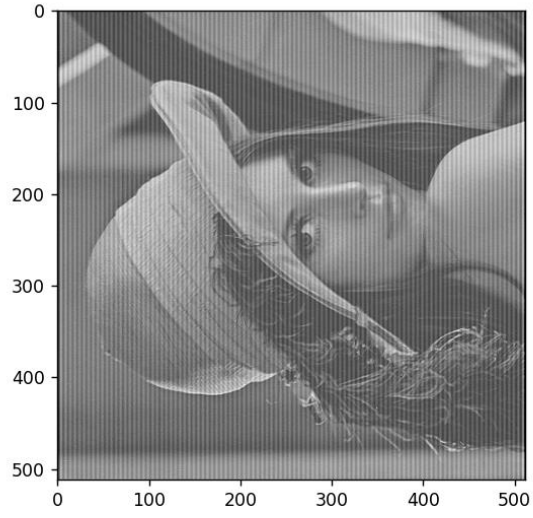


x=418. y=33.

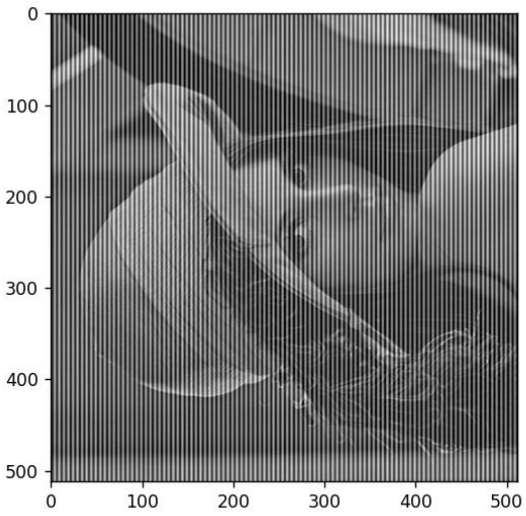
notchGaussian2-fft



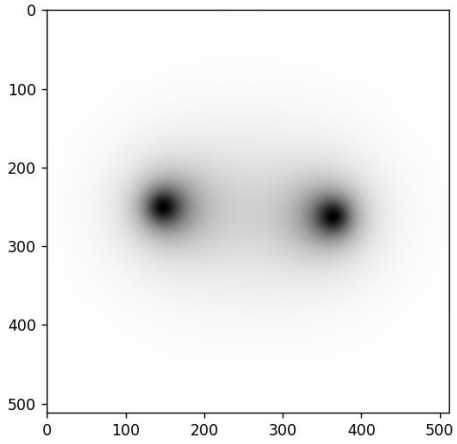
notchGaussian2-image



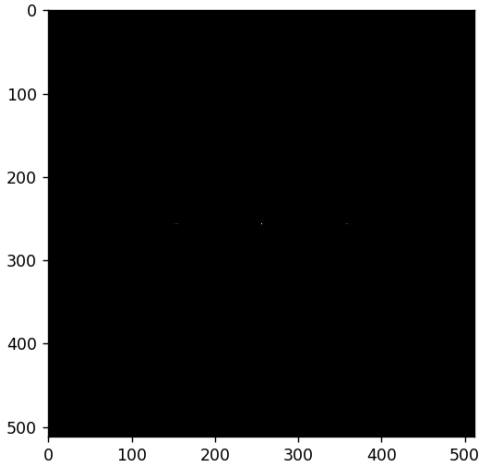
notchButterworth2-image



notchButterworth2-filter



notchButterworth2-fft



notchButterworth2-image