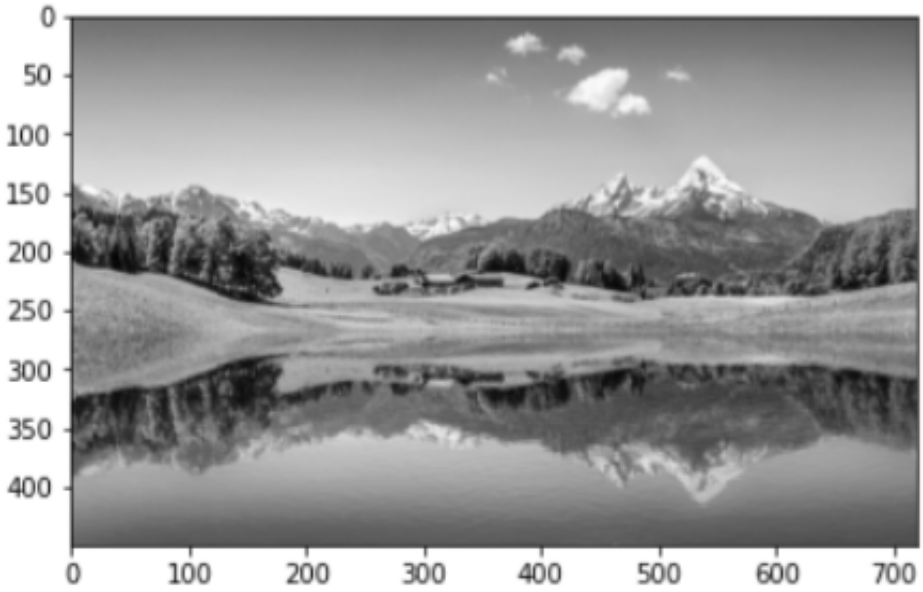
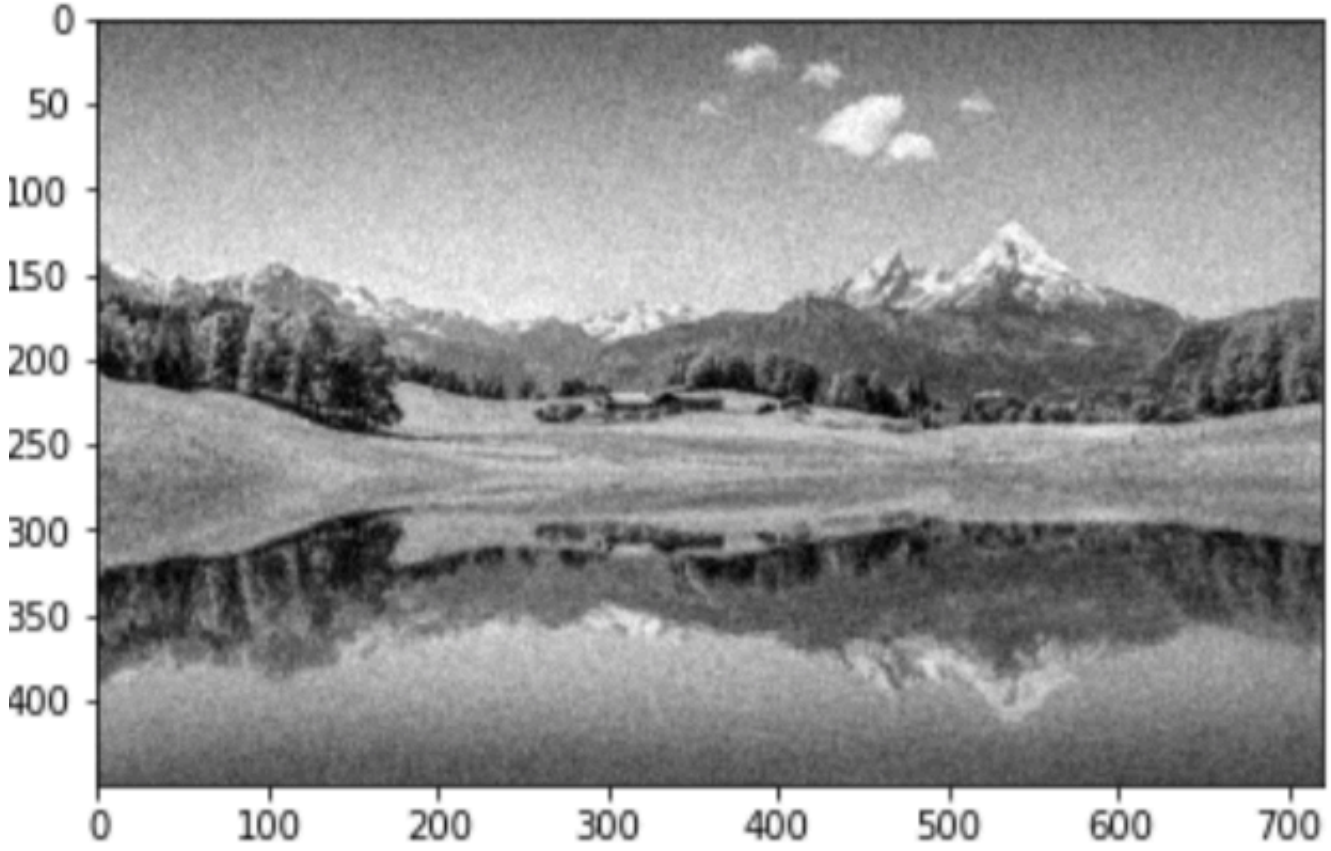
a)

convert image to grayscale:

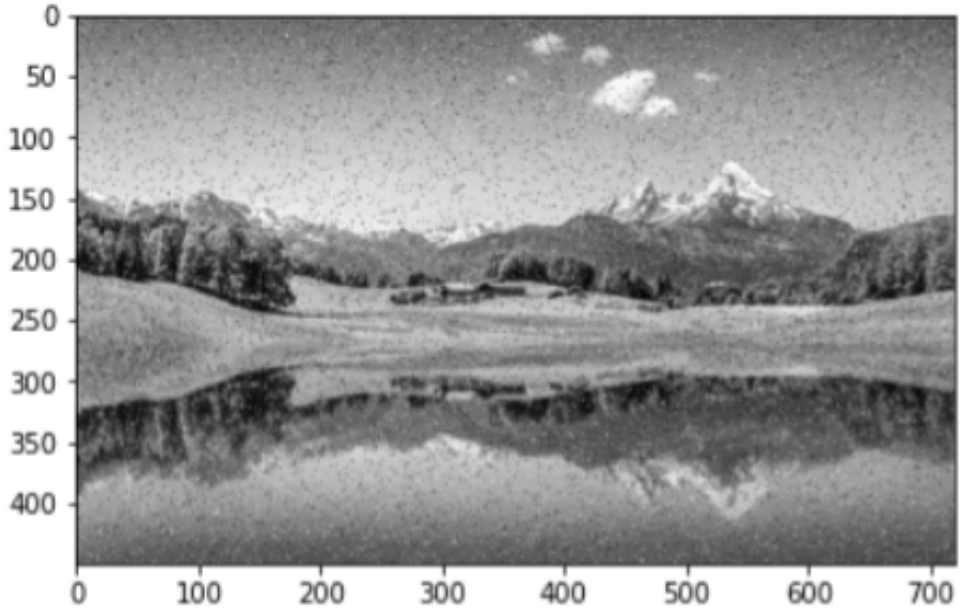


b)

J1: Gaussian white noise added with variance 0.01

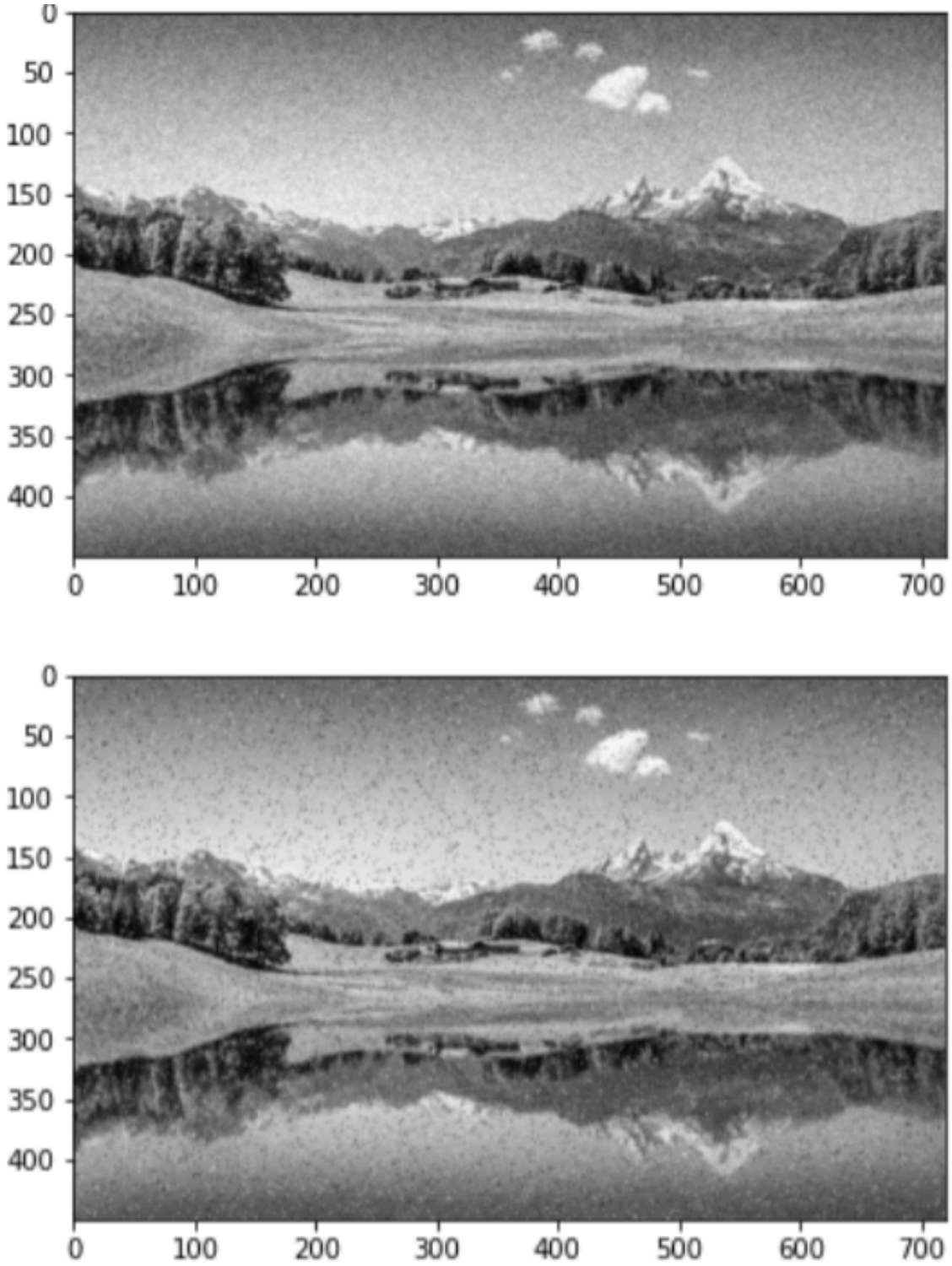


J2: Salt-and-pepper noise, affecting approximately 5% of pixels

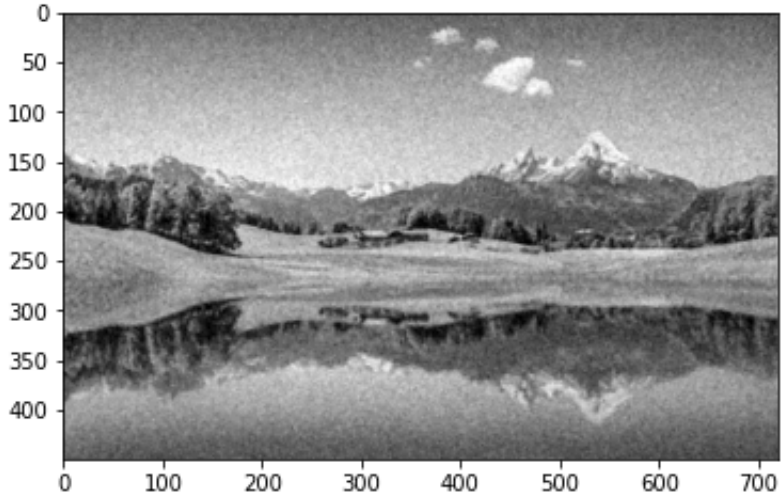


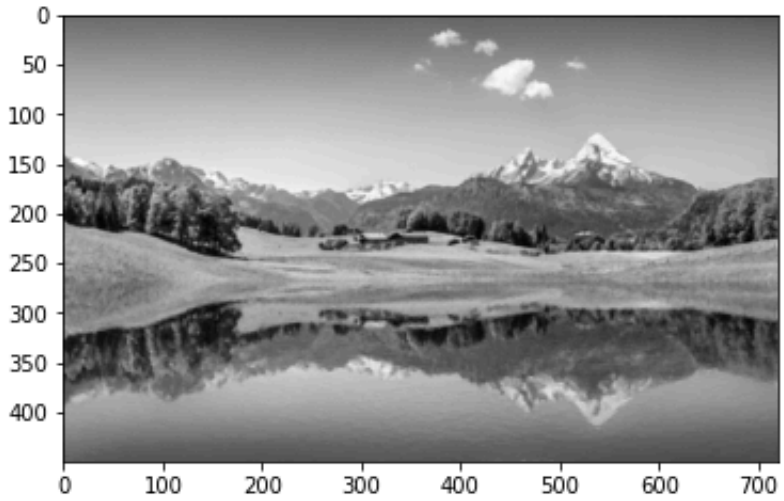
c)

1 Gaussian filter with sigma = 2, results for J1 and J2 are shown as the first and second image respectively

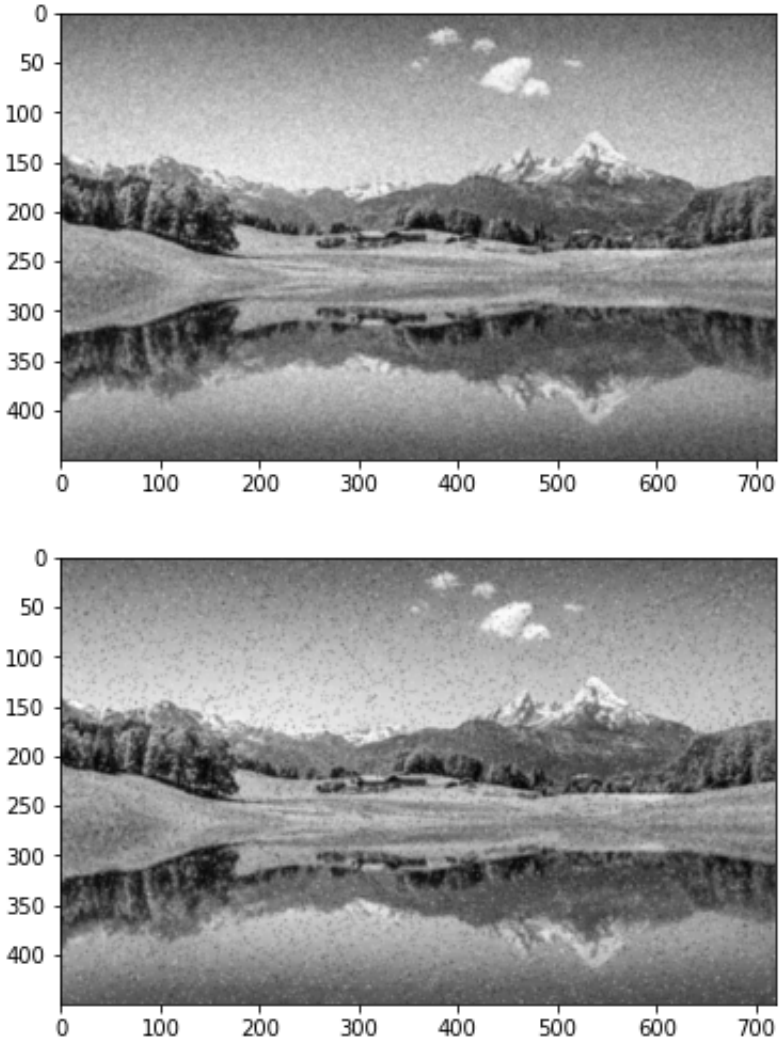


2 Median filtering, results for J1 and J2 are shown as the first and second image respectively

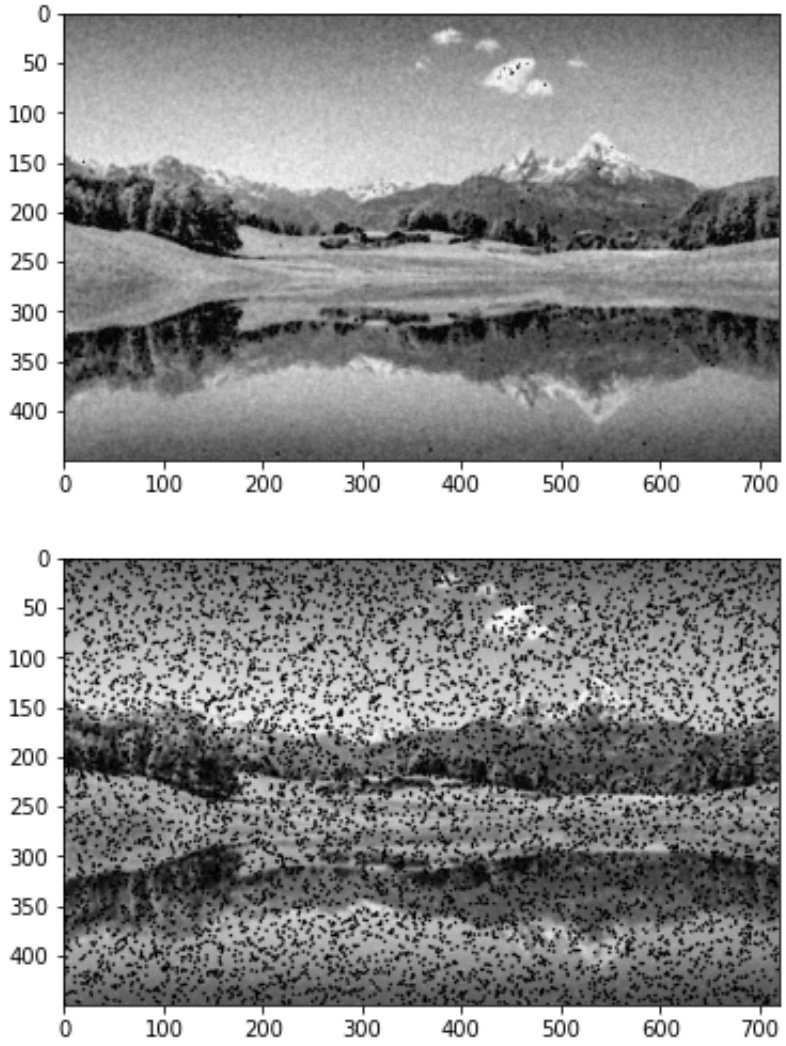




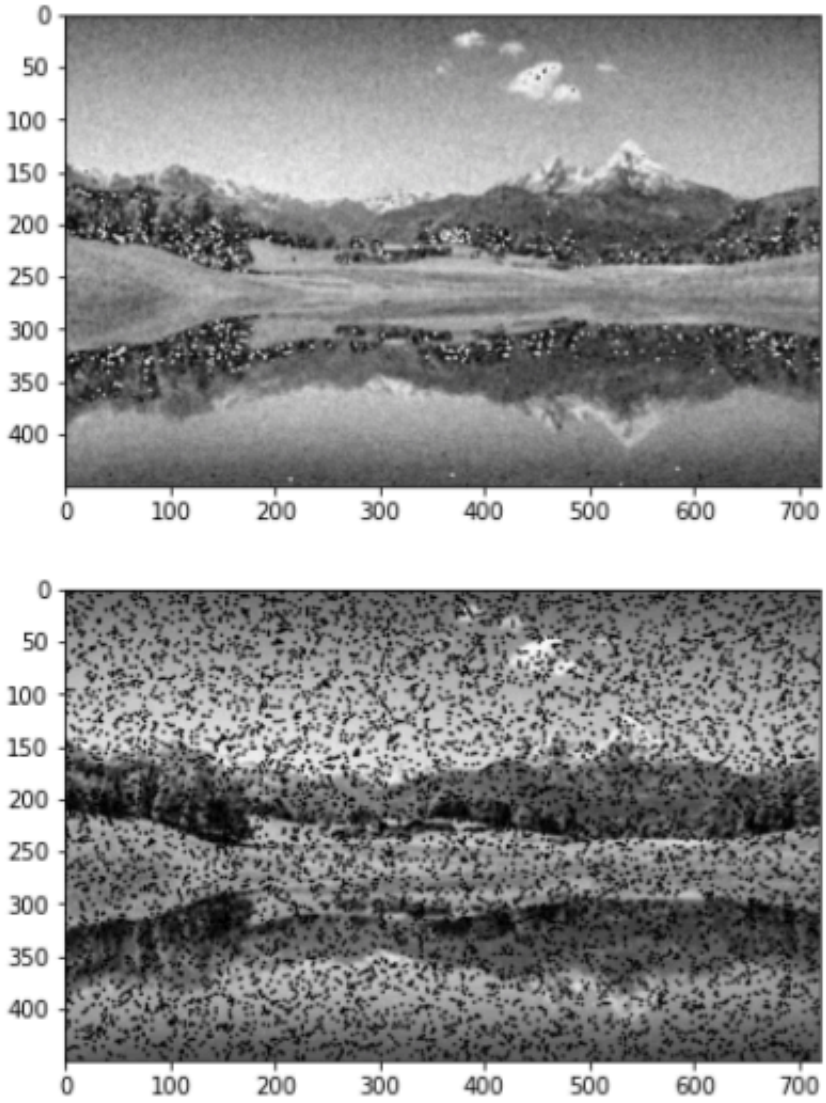
3 Arithmetic mean filter for J1 and J2



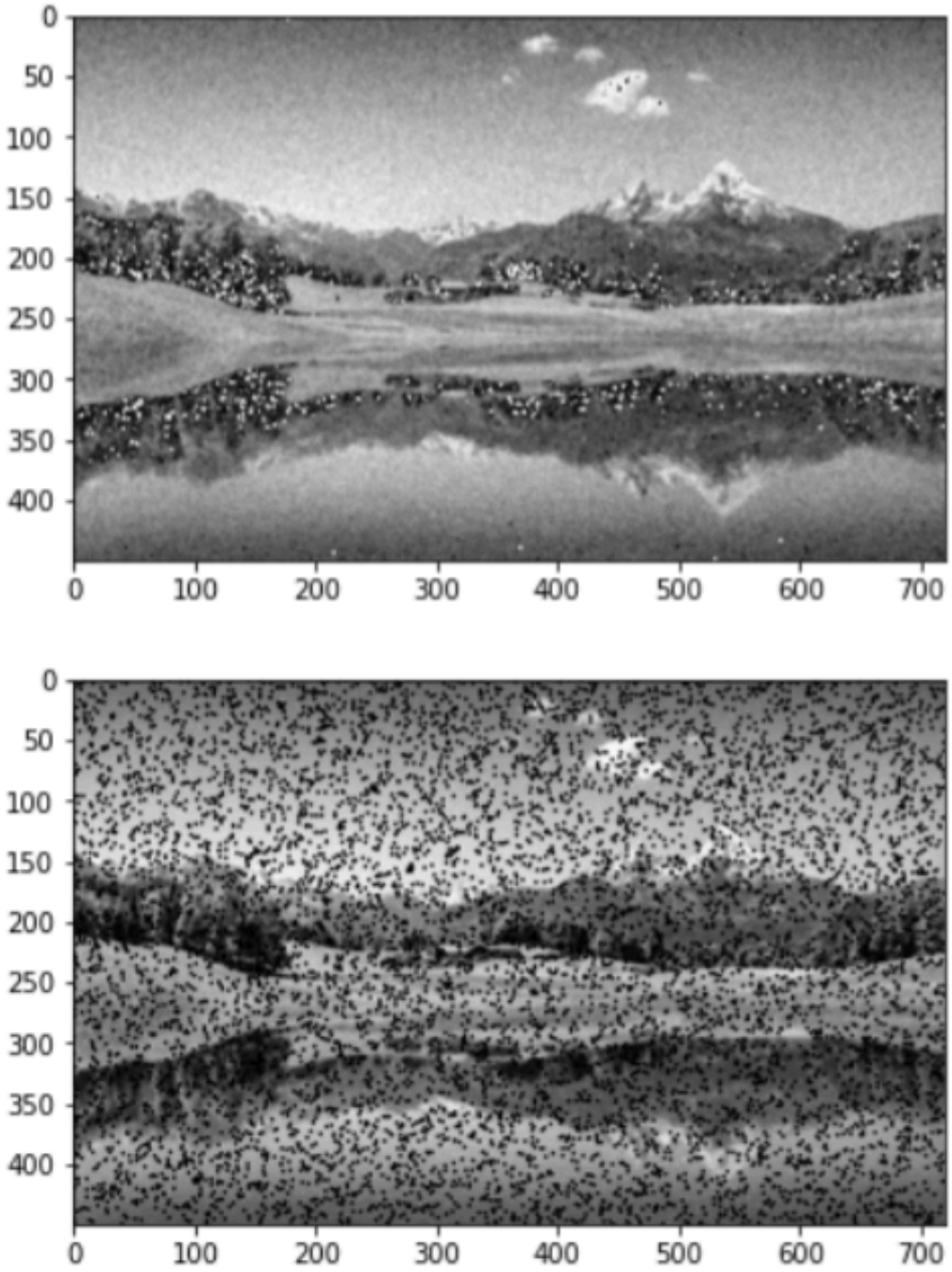
4 Geometric mean filter for J1 and J2



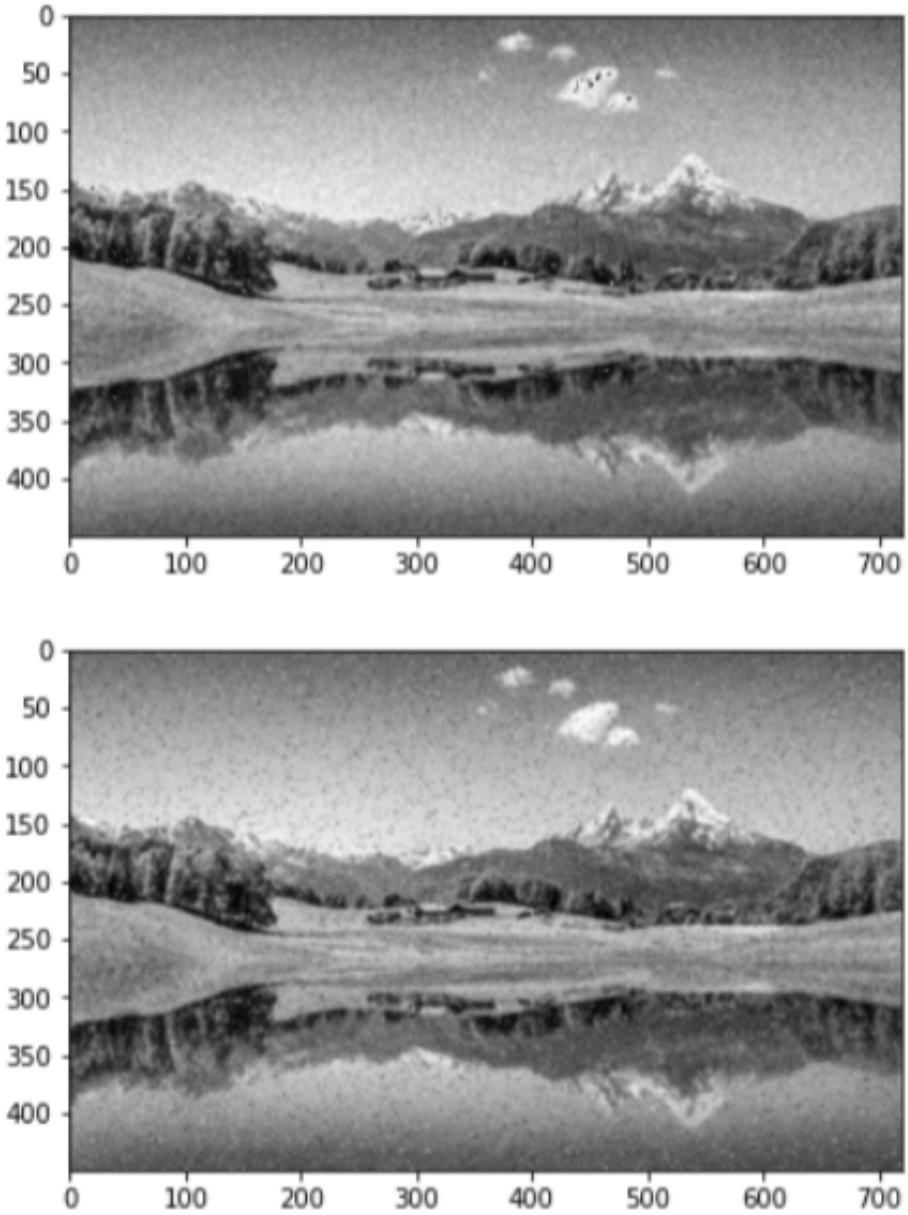
5 Harmonic mean filter for J1 and J2



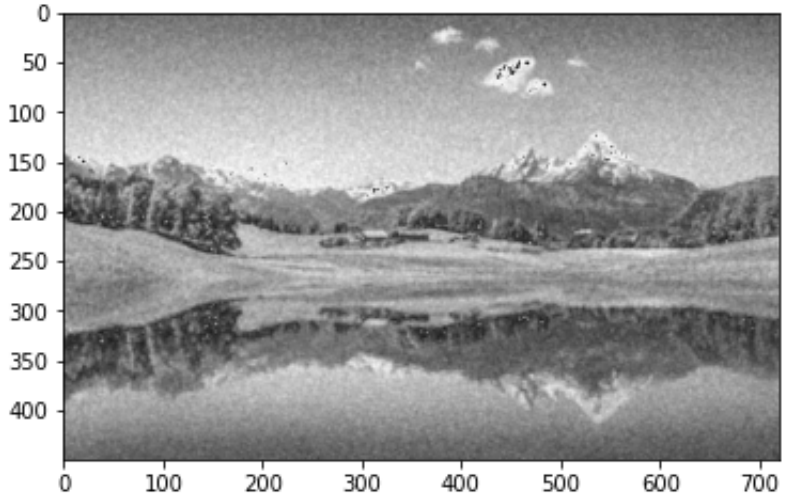
6 Contraharmonic mean filter with $Q = -1$ for J1 and J2

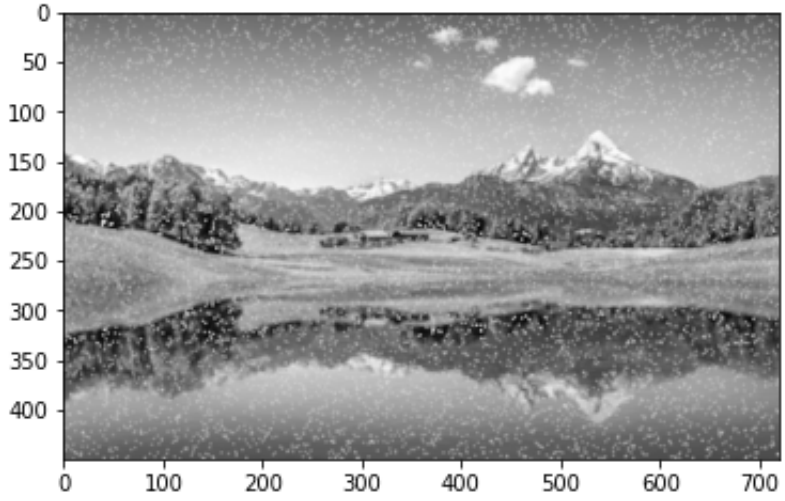


Contraharmonic mean filter with Q = 0 for J1 and J2

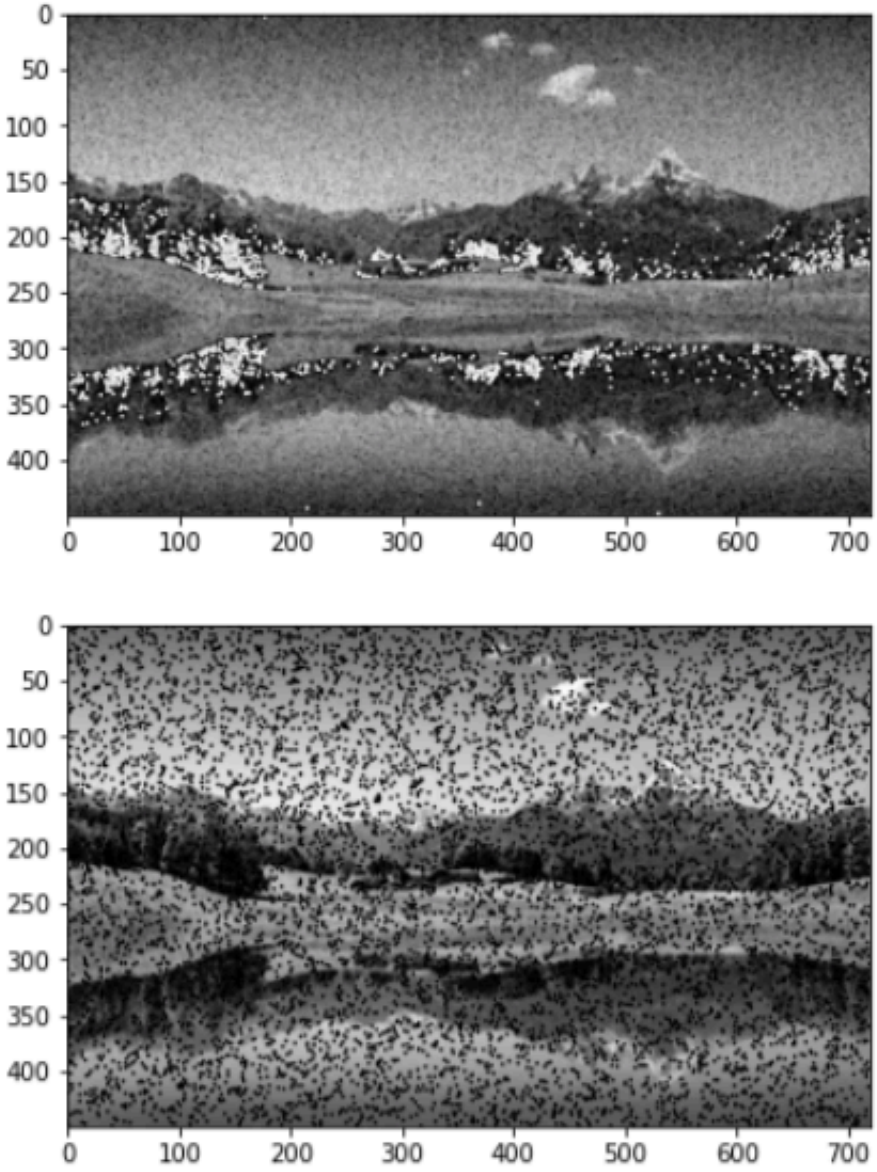


Contraharmonic mean filter with Q = 1 for J1 and J2

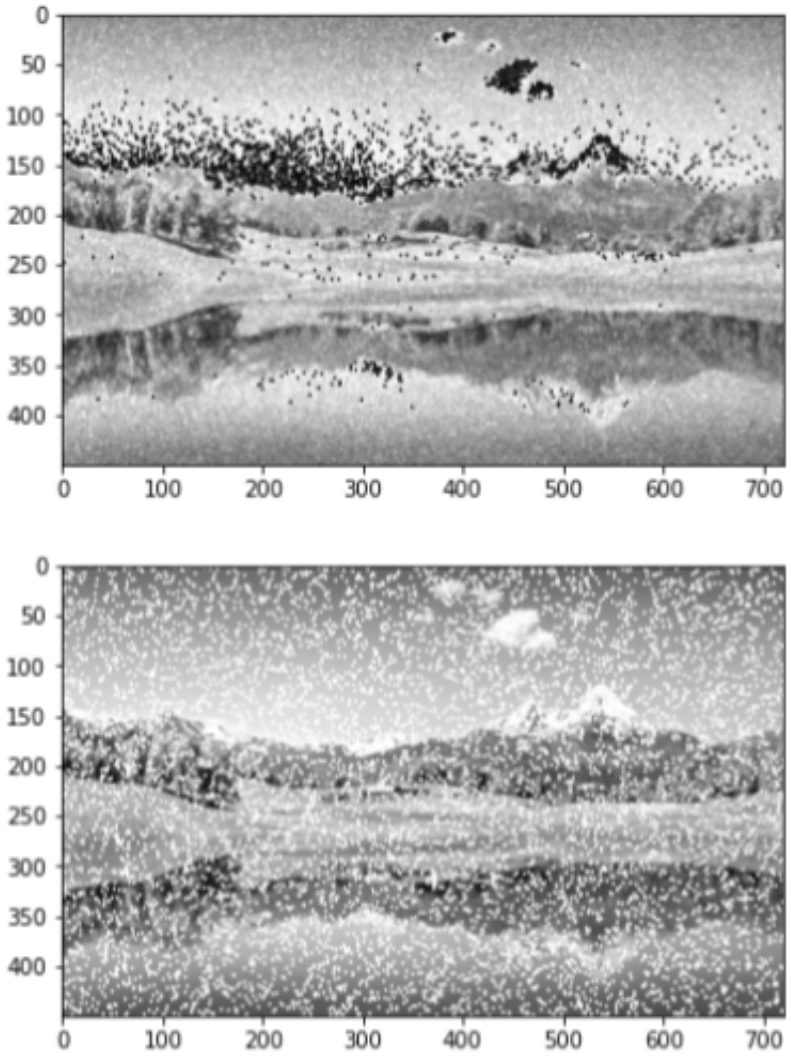




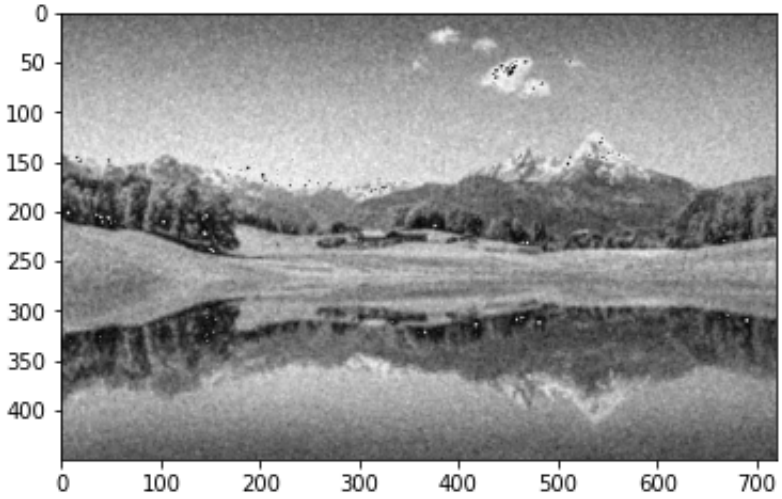
7 Minimum filter for J1 and J2

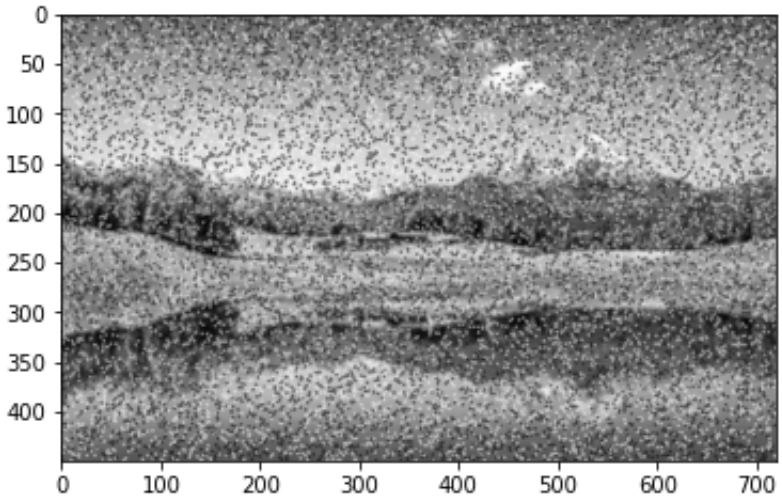


8 Maximum filter for J1 and J2



9 Midpoint filter for J1 and J2





d)

1 What denoising technique do you recommend for removing Gaussian noise?

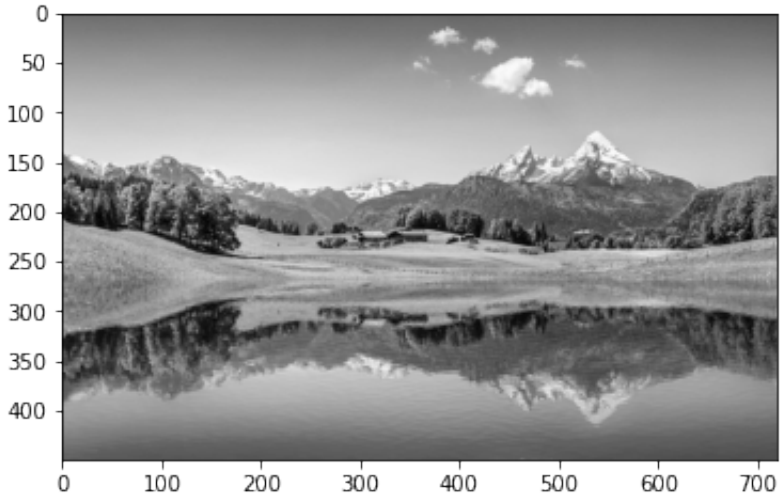
For Gaussian noise, Gaussian fitering, Median filtering, Arithmetic mean filtering, Geometric mean filtering, Contraharmonic mean filter with $Q = 0$ and 1, Midpoint filtering work fine, but cannot significantly reduce Gaussian noise.

For salt and pepper noise, Median filtering is very efficient and highly recommended. Gaussian filtering, Arithmetic mean filtering work fine with salt and pepper noise, but not as efficient as Median filering. Geometric mean, Harmonic mean filtering(Contraharmonic mean filter with $Q = -1$) work terribly with salt and pepper noise. Minimum filtering only removes salt noise, not pepper noise; on the contrary, Maximum filtering only removes pepper noise, not salt noise.

In [3]:
```
#PIL, scipy.ndimage, skimage
import scipy.misc as spm
import scipy.ndimage as ndi
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
from sklearn.cluster import k_means
np.random.seed(0)
from skimage import io
import random
import cv2 #OpenCV
from statistics import harmonic_mean
%matplotlib inline
```

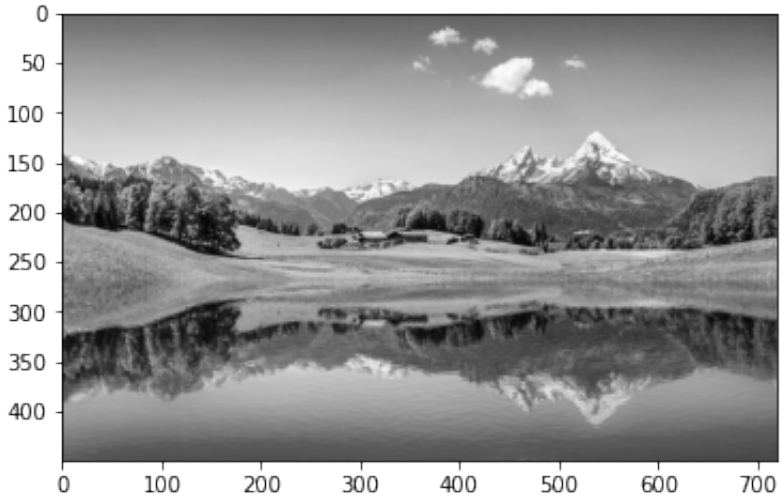
In [7]:
```
#convert RGB to grayscale
#https://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python
I = io.imread('theAlps.jpg', as_gray=True)
plt.imshow(I,cmap='gray')
I.shape
```

Out[7]: (450, 720)



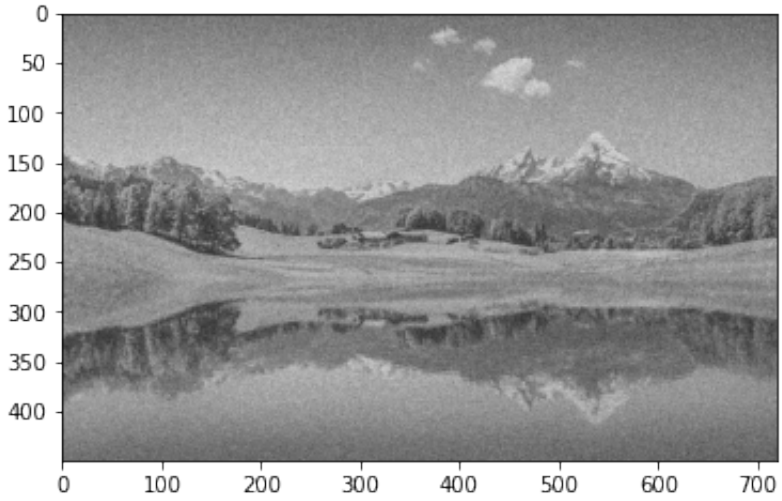
In [5]:
```
I = cv2.imread('theAlps.jpg', cv2.IMREAD_GRAYSCALE).astype(float)
plt.imshow(I,cmap='gray')
```

Out[5]: <matplotlib.image.AxesImage at 0x12dba92b0>



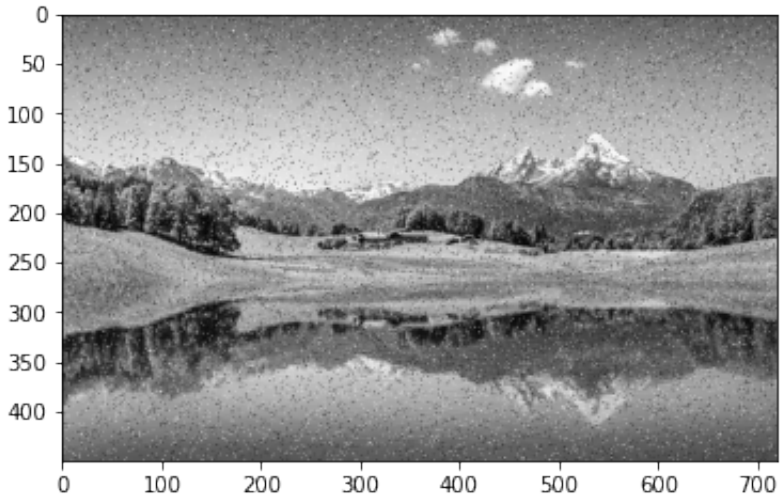
In [54]:
```
#zero mean Gaussian white noise
g = np.random.normal(0,0.1,size=I.shape).reshape(I.shape)
J1 = I + g*np.max(I)
#J1 = np.round(J1*255).astype(int)
#J1 = np.uint8(J1)
plt.imshow(J1,cmap='gray')
```

Out[54]: <matplotlib.image.AxesImage at 0x1305fe2b0>

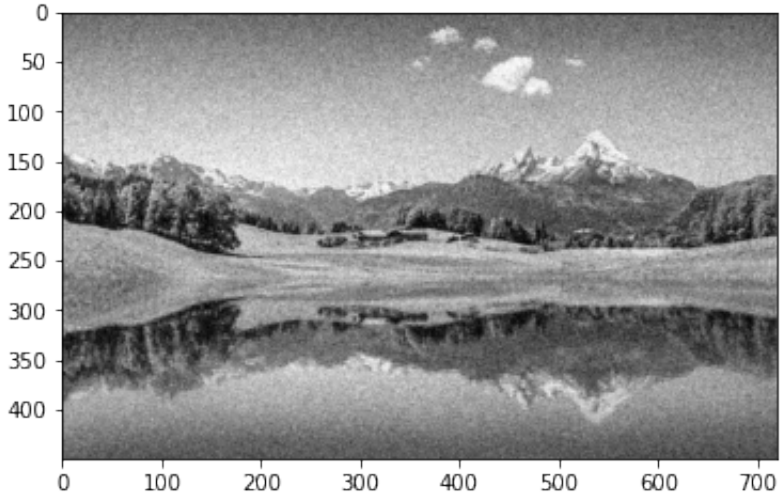


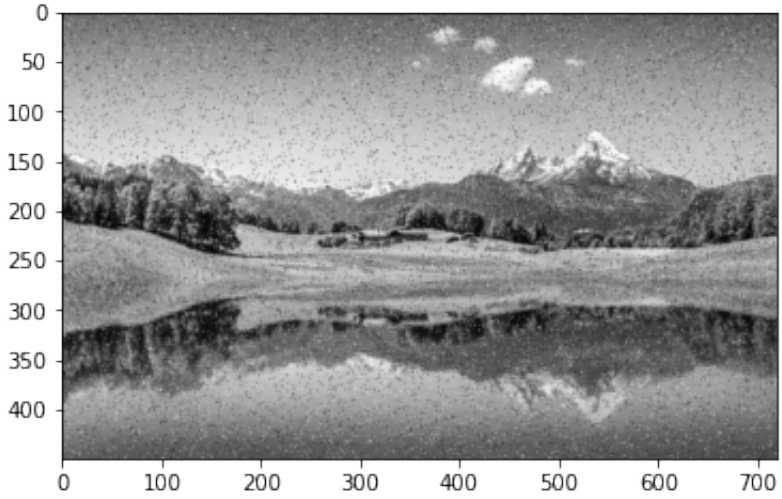
In [9]:
```
#5% chance of turning a pixel either black or white
#https://stackoverflow.com/questions/22937589/how-to-add-noise-gaussian-salt-and-pepper-etc-to-image-in-python-with-opencv
def sp_noise(image,prob):
    '''
    Add salt and pepper noise to image
    prob: Probability of the noise
    '''
    output = np.zeros(image.shape)
    thres = 1 - prob
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rdn = random.random()
            if rdn < prob:
                output[i][j] = 0
            elif rdn > thres:
                output[i][j] = 1
            else:
                output[i][j] = image[i][j]
    return output

J2 = sp_noise(I,0.025)
plt.imshow(J2,cmap='gray')
```

Out[9]: <matplotlib.image.AxesImage at 0x12ee86278>

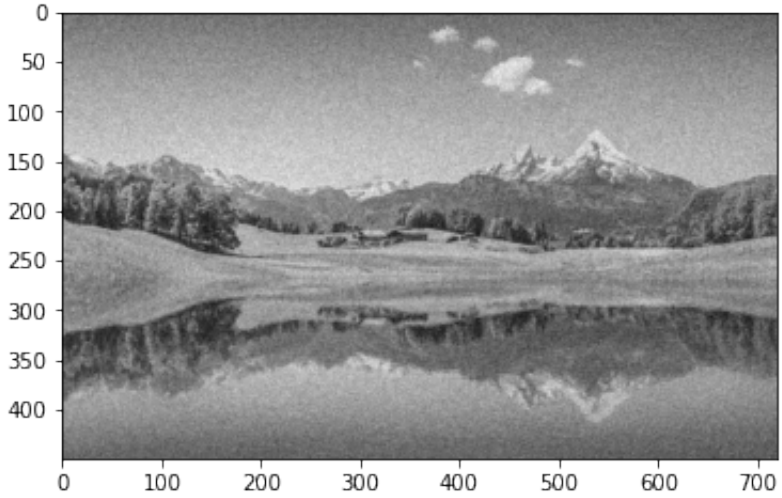


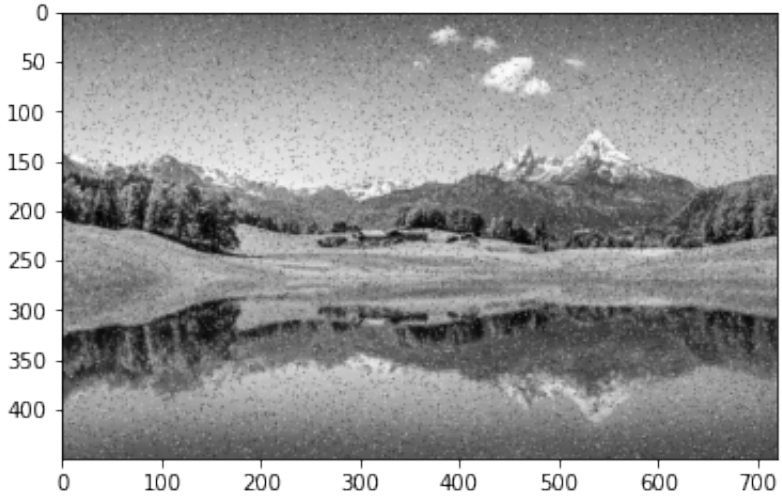
In [19]:
```
#Gaussian kernel
X = np.linspace(-1,1,3)
Y = np.linspace(-1,1,3)
W = np.zeros((3,3))
sigma = 2
for i in range(3):
    for j in range(3):
        W[i,j] = (1/2*np.pi*sigma**2)*np.exp(-(X[i]**2+Y[j]**2)/2*sigm
a**2)

W = W/np.sum(W)

plt.figure()
J1_Gaus = ndi.convolve(J1,W)
plt.imshow(J1_Gaus,cmap='gray')
plt.show()

plt.figure()
J2_Gaus = ndi.convolve(J2,W)
plt.imshow(J2_Gaus,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

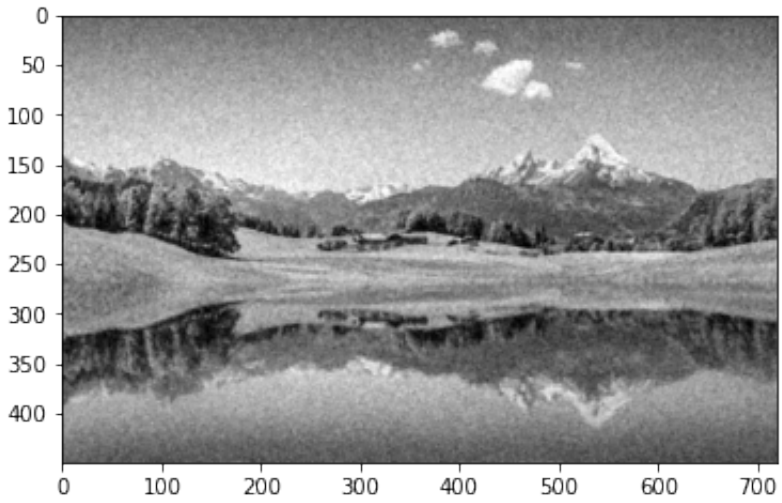

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

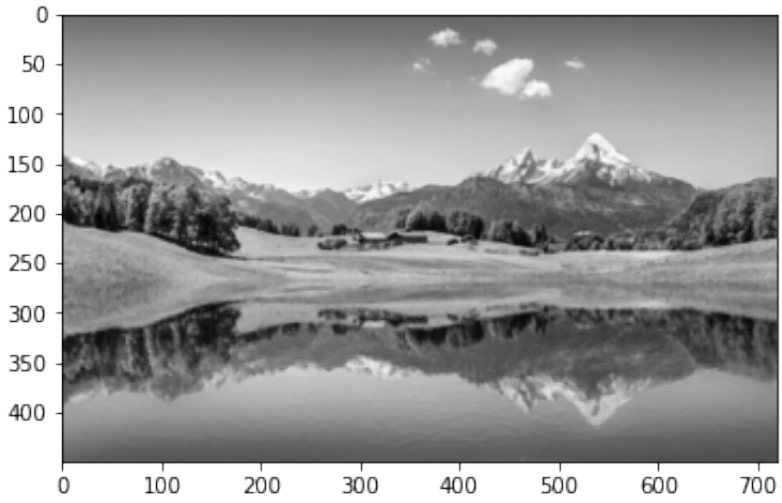
In [7]:
```
#Gaussian kernel
X = np.linspace(-1,1,3)
Y = np.linspace(-1,1,3)
W = np.zeros((3,3))
sigma = 2
for i in range(3):
    for j in range(3):
        W[i,j] = np.exp(-(X[i]**2+Y[j]**2)/2*sigma**2)

W = W/np.sum(W)

plt.figure()
J1_Gaus = ndi.convolve(J1,W)
plt.imshow(J1_Gaus,cmap='gray')
plt.show()

plt.figure()
J2_Gaus = ndi.convolve(J2,W)
plt.imshow(J2_Gaus,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

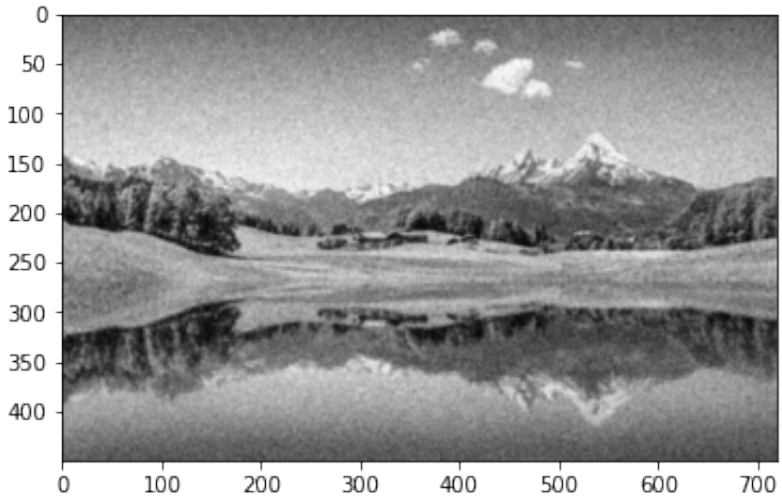

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

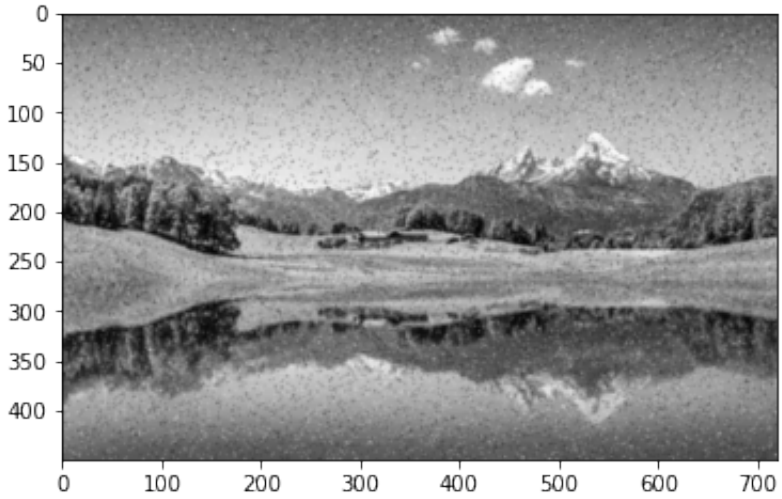
In [20]:
```
#median filtering
plt.figure()
J1_Median = ndi.median_filter(J1,size=3)
plt.imshow(J1_Median,cmap='gray')
plt.show()

plt.figure()
J2_Median = ndi.median_filter(J2,size=3)
plt.imshow(J2_Median,cmap='gray')
plt.show()
```



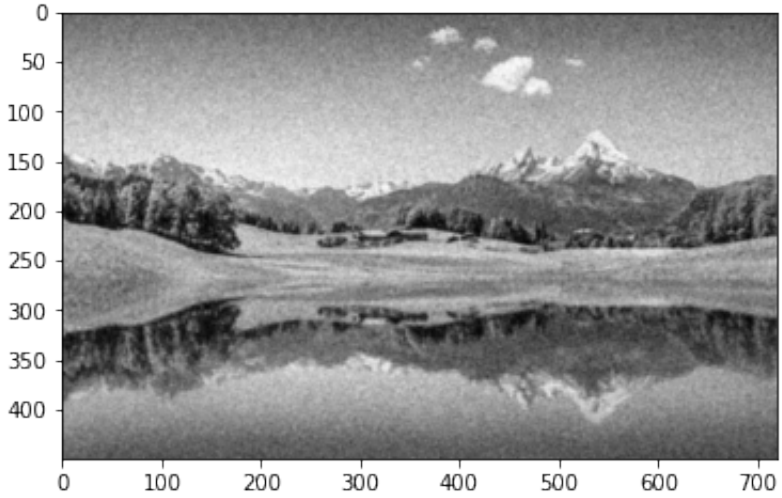


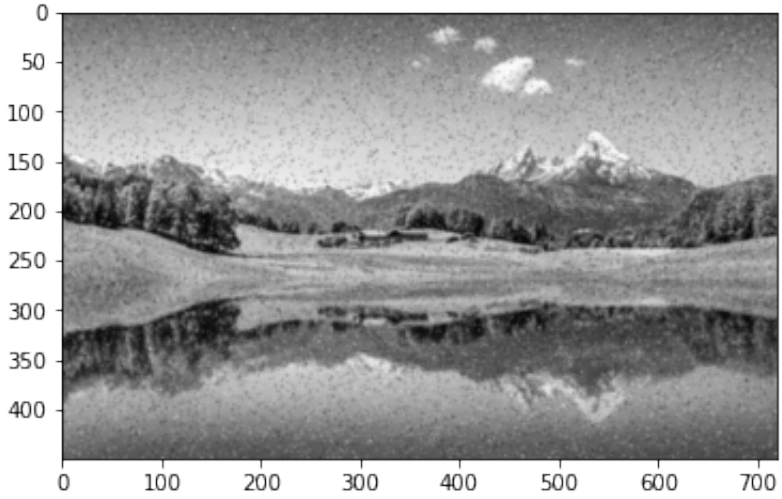
In [21]:
```
#arithmetic mean filtering with cv2
#https://machinelearningknowledge.ai/python-opencv-image-smoothing-using-averaging-gaussian-blur-and-median-filter/
plt.figure()
J1_mean = cv2.blur(J1,(3,3))
plt.imshow(J1_mean,cmap='gray')
plt.show()

plt.figure()
J2_mean = cv2.blur(J2,(3,3))
plt.imshow(J2_mean,cmap='gray')
plt.show()
```





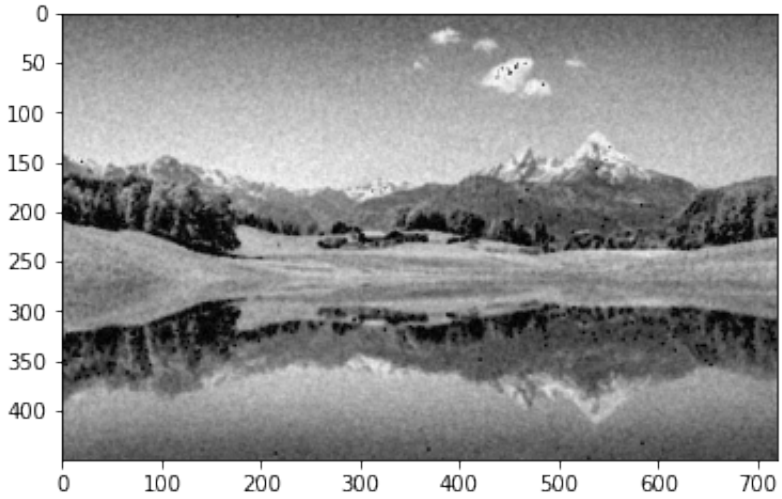
In [28]:
```
#arithmetic mean filtering
rows, cols = J1.shape[:2]
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
mean1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        mean1[r, c] = np.mean(pad_img[r:r+ksize, c:c+ksize])
mean1 = (np.round(mean1*255)).astype(int)
mean1 = np.uint8(mean1)
plt.figure()
plt.imshow(mean1,cmap='gray')
plt.show()

rows, cols = J2.shape[:2]
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
mean2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        mean2[r, c] = np.mean(pad_img[r:r+ksize, c:c+ksize])
mean2 = (np.round(mean2*255)).astype(int)
mean2 = np.uint8(mean2)
plt.figure()
plt.imshow(mean2,cmap='gray')
plt.show()
```

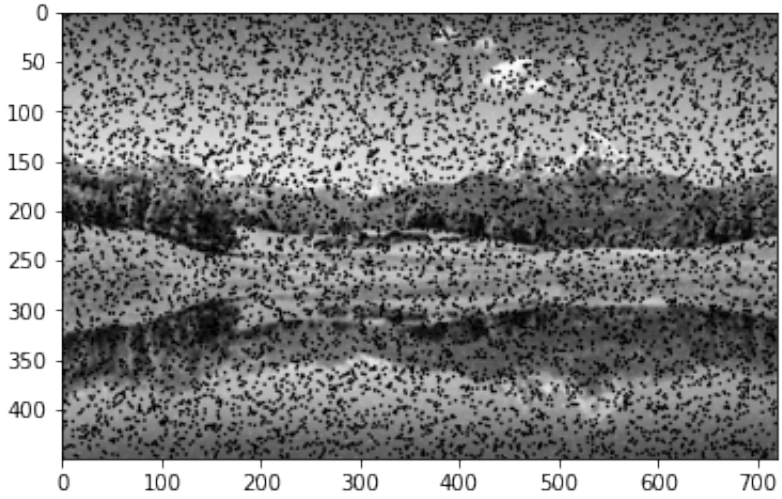

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

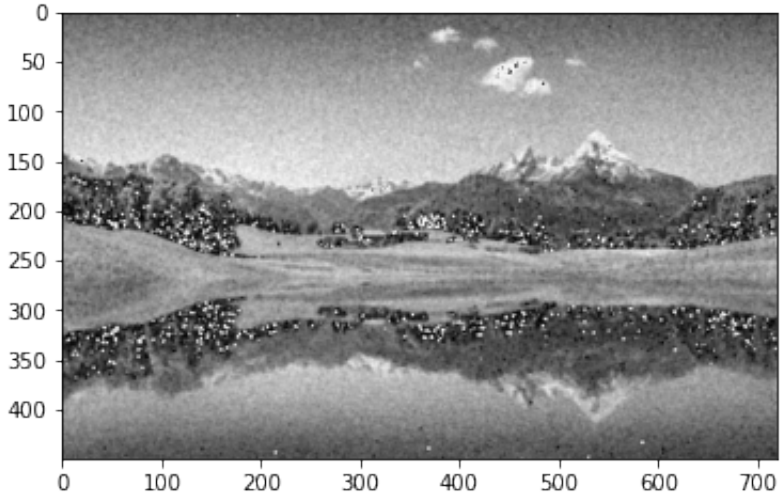
In [38]:
```
#Geometric mean filtering
#https://stackoverflow.com/questions/48065702/geometric-mean-filter-with-opencv

rows, cols = J1.shape[:2]
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
geomean1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        geomean1[r, c] = np.prod(pad_img[r:r+ksize, c:c+ksize])**(1/(k
size**2))
geomean1 = (np.round(geomean1*255)).astype(int)
geomean1 = np.uint8(geomean1)
plt.figure()
plt.imshow(geomean1,cmap='gray')
plt.show()


rows, cols = J2.shape[:2]
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
geomean2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        geomean2[r, c] = np.prod(pad_img[r:r+ksize, c:c+ksize])**(1/(k
size**2))
geomean2 = (np.round(geomean2*255)).astype(int)
geomean2 = np.uint8(geomean2)
plt.figure()
plt.imshow(geomean2,cmap='gray')
plt.show()
```

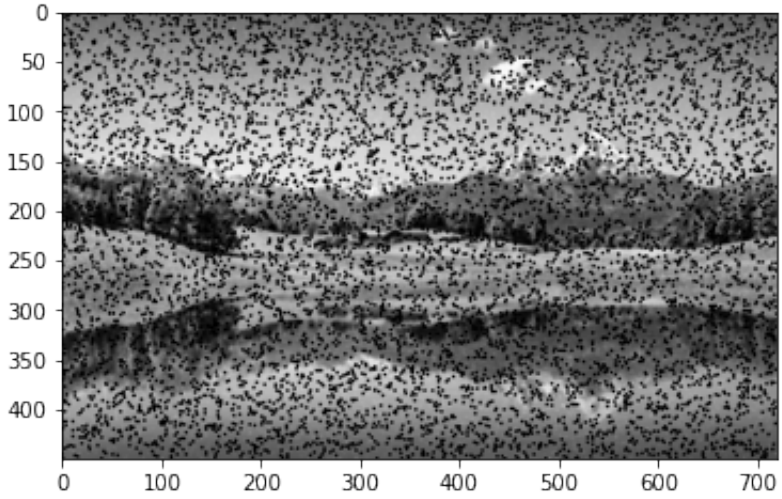
```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:11: Run
timeWarning: invalid value encountered in double_scalars
  # This is added back by InteractiveShellApp.init_path()
```





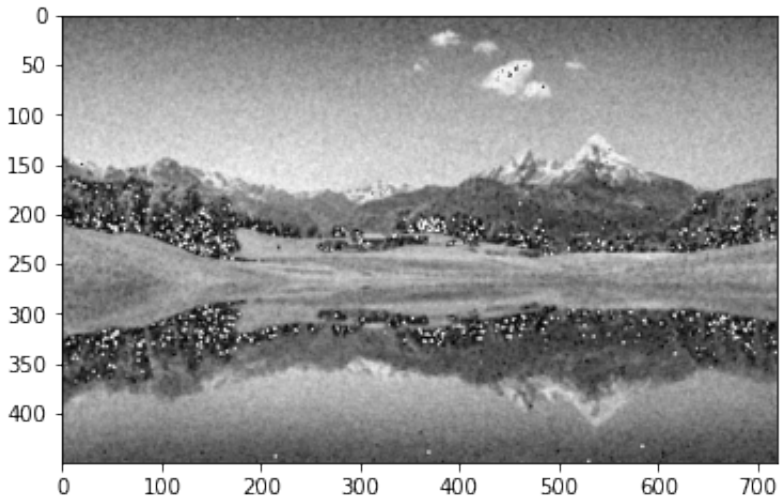
In [43]:
```
#harmonic mean filtering
import warnings
warnings.filterwarnings('ignore')
#I = io.imread('theAlps.jpg', as_gray=True)
#I = cv2.imread('theAlps.jpg', cv2.IMREAD_GRAYSCALE).astype(float)
#g = np.random.normal(0,0.1,size=I.shape).reshape(I.shape)
#J1 = I + g*np.max(I)
def Func(a):
    a = a.reshape((3,3))
    return 9/np.sum(1/a)
harmean1 = ndi.filters.generic_filter(J1,Func,size=3)
harmean1 = (np.round(harmean1*255)).astype(int)
harmean1 = np.uint8(harmean1)
plt.figure()
plt.imshow(harmean1,cmap='gray')
plt.show()

rows, cols = J2.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
harmean2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        harmean2[r, c] = 1/np.mean(1/pad_img[r:r+ksize, c:c+ksize])
harmean2 = (np.round(harmean2*255)).astype(int)
harmean2 = np.uint8(harmean2)
plt.figure()
plt.imshow(harmean2,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

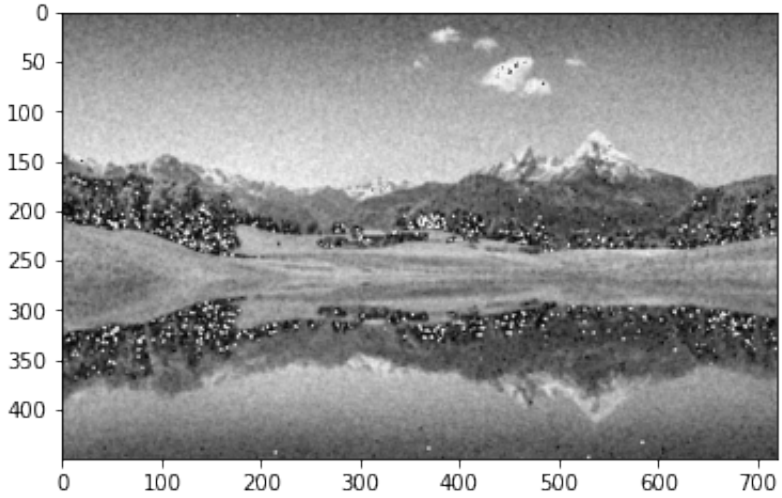

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

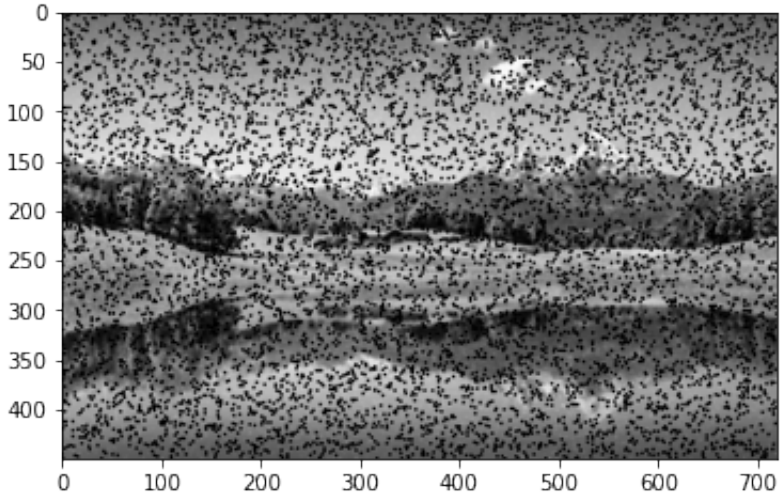
In [40]:
```
#harmonic mean filter
#I = cv2.imread('theAlps.jpg', cv2.IMREAD_GRAYSCALE).astype(float)
rows, cols = J1.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
harmean1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        harmean1[r, c] = ksize**2/np.sum(1/pad_img[r:r+ksize, c:c+ksiz
e])
harmean1 = (np.round(harmean1*255)).astype(int)
harmean1 = np.uint8(harmean1)
plt.imshow(harmean1,cmap='gray')
```

Out[40]: <matplotlib.image.AxesImage at 0x12da27358>

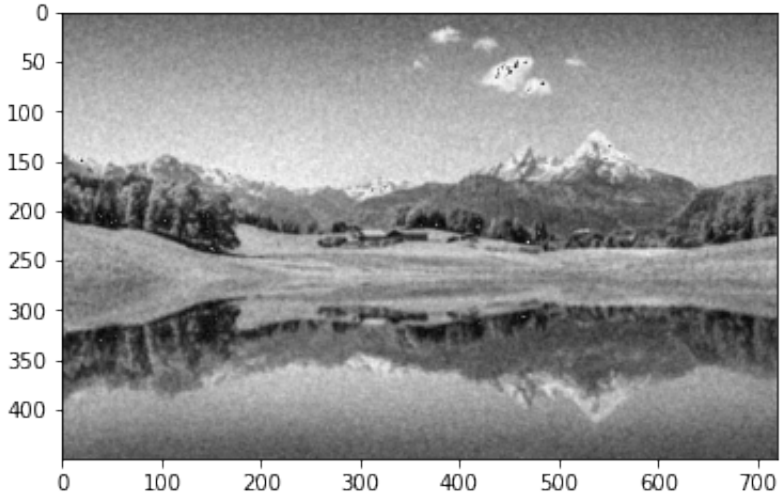


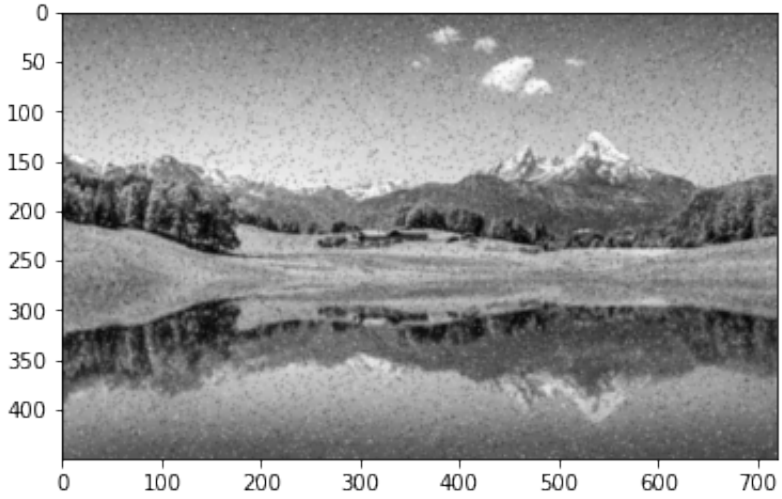
In [45]:
```
#Contraharmonic mean filter
#Q=-1
rows, cols = J1.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
conmean1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        conmean1[r, c] = np.sum(pad_img[r:r+ksize, c:c+ksize]**0)/np.s
um(pad_img[r:r+ksize, c:c+ksize]**(-1))
conmean1 = (np.round(conmean1*255)).astype(int)
conmean1 = np.uint8(conmean1)
plt.figure()
plt.imshow(conmean1,cmap='gray')
plt.show()

#Q=-1
rows, cols = J2.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
conmean2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        conmean2[r, c] = np.sum(pad_img[r:r+ksize, c:c+ksize]**0)/np.s
um(pad_img[r:r+ksize, c:c+ksize]**(-1))
conmean2 = (np.round(conmean2*255)).astype(int)
conmean2 = np.uint8(conmean2)
plt.figure()
plt.imshow(conmean2,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

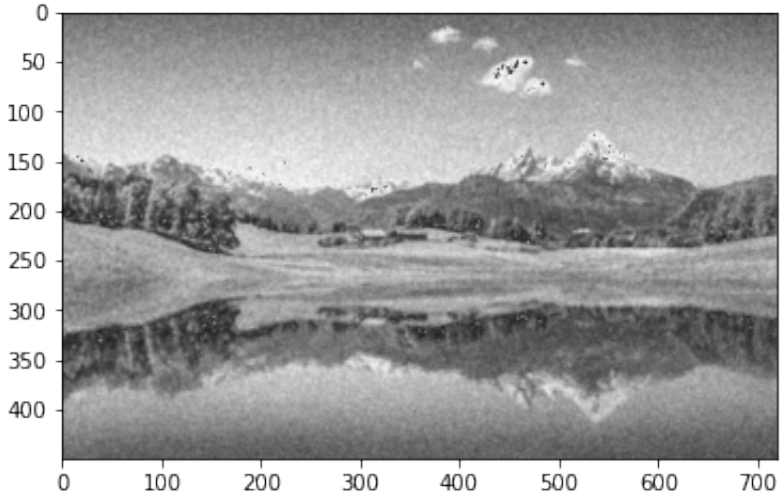

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

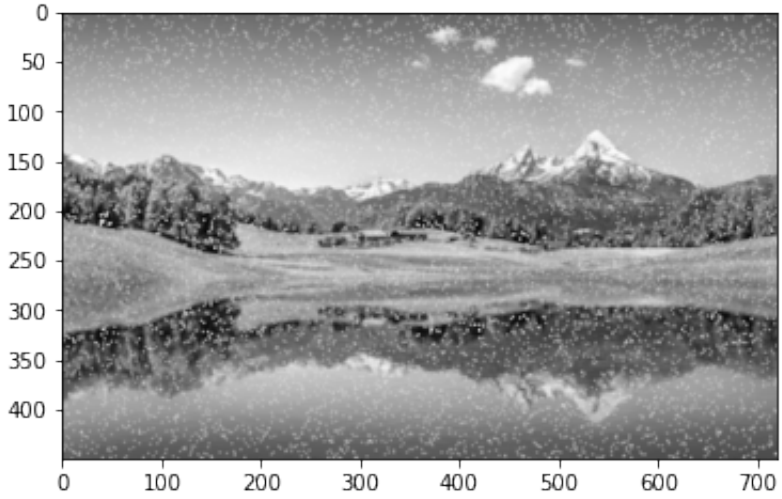
In [46]:
```
#Q=0
rows, cols = J1.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
conmean1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        conmean1[r, c] = np.sum(pad_img[r:r+ksize, c:c+ksize]**1)/np.s
um(pad_img[r:r+ksize, c:c+ksize]**0)
conmean1 = (np.round(conmean1*255)).astype(int)
conmean1 = np.uint8(conmean1)
plt.figure()
plt.imshow(conmean1,cmap='gray')
plt.show()

#Q=0
rows, cols = J2.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
conmean2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        conmean2[r, c] = np.sum(pad_img[r:r+ksize, c:c+ksize]**1)/np.s
um(pad_img[r:r+ksize, c:c+ksize]**0)
conmean2 = (np.round(conmean2*255)).astype(int)
conmean2 = np.uint8(conmean2)
plt.figure()
plt.imshow(conmean2,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

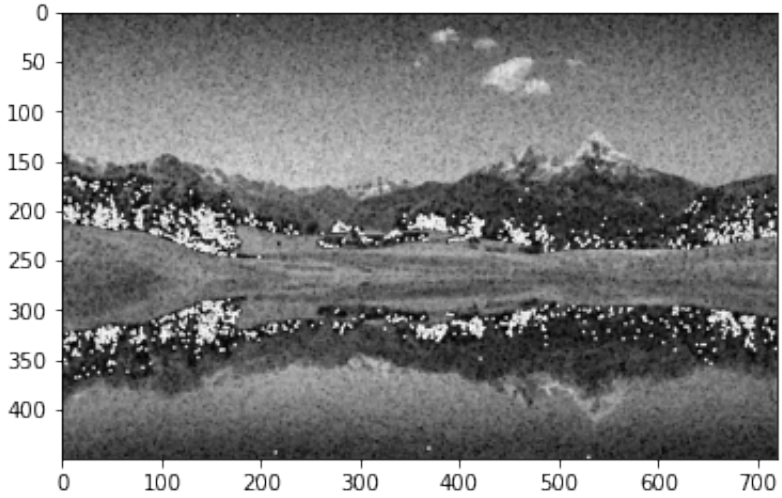

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

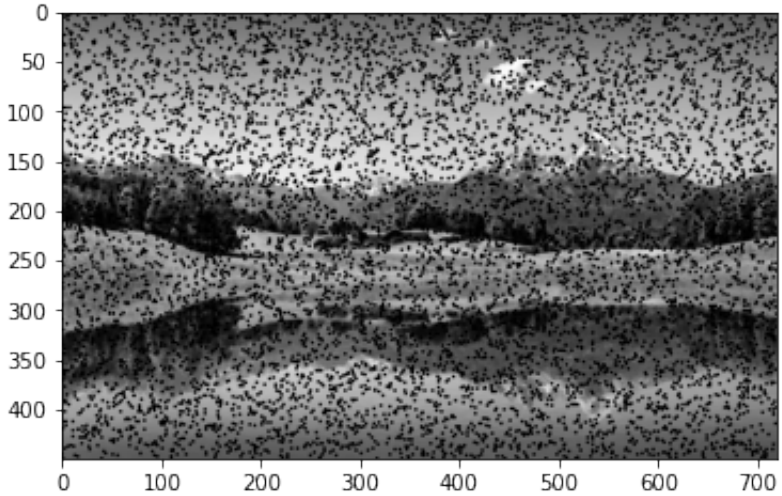
In [47]:
```
#Q=1
rows, cols = J1.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
conmean1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        conmean1[r, c] = np.sum(pad_img[r:r+ksize, c:c+ksize]**2)/np.s
um(pad_img[r:r+ksize, c:c+ksize]**1)
conmean1 = (np.round(conmean1*255)).astype(int)
conmean1 = np.uint8(conmean1)
plt.figure()
plt.imshow(conmean1,cmap='gray')
plt.show()

#Q=1
rows, cols = J2.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
conmean2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        conmean2[r, c] = np.sum(pad_img[r:r+ksize, c:c+ksize]**2)/np.s
um(pad_img[r:r+ksize, c:c+ksize]**1)
conmean2 = (np.round(conmean2*255)).astype(int)
conmean2 = np.uint8(conmean2)
plt.figure()
plt.imshow(conmean2,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

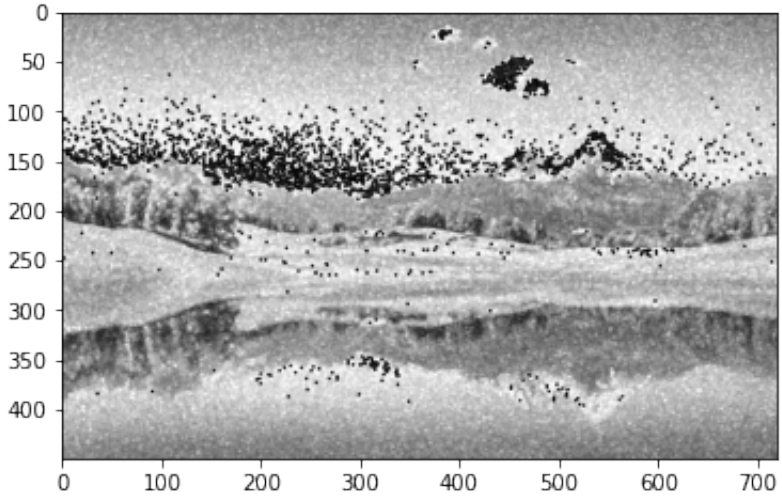

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

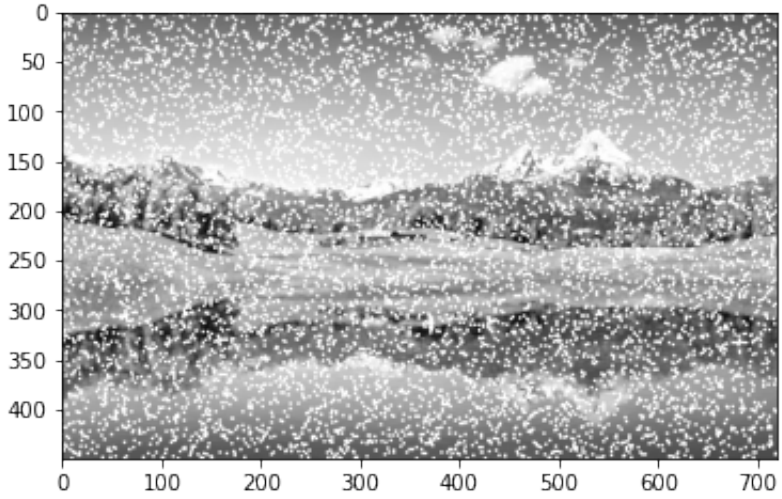
In [50]:
```
#Minimum filter
rows, cols = J1.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
min1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        min1[r, c] = np.min(pad_img[r:r+ksize, c:c+ksize])
min1 = (np.round(min1*255)).astype(int)
min1 = np.uint8(min1)
plt.figure()
plt.imshow(min1,cmap='gray')
plt.show()

#Minimum filter
rows, cols = J2.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
min2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        min2[r, c] = np.min(pad_img[r:r+ksize, c:c+ksize])
min2 = (np.round(min2*255)).astype(int)
min2 = np.uint8(min2)
plt.figure()
plt.imshow(min2,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

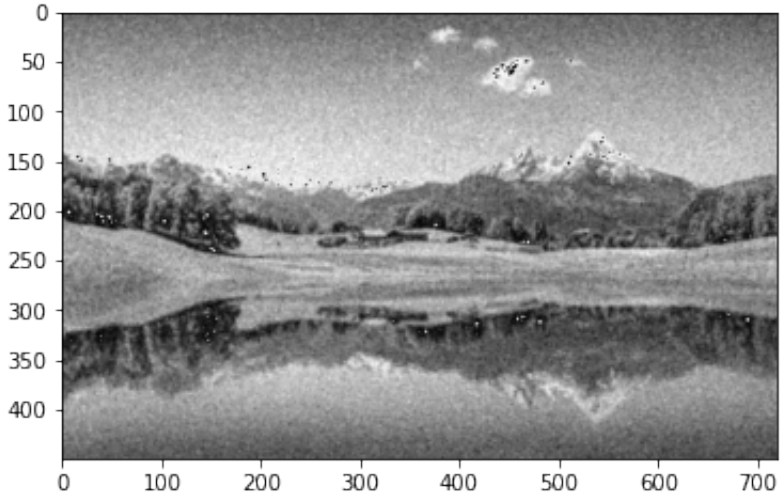

0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

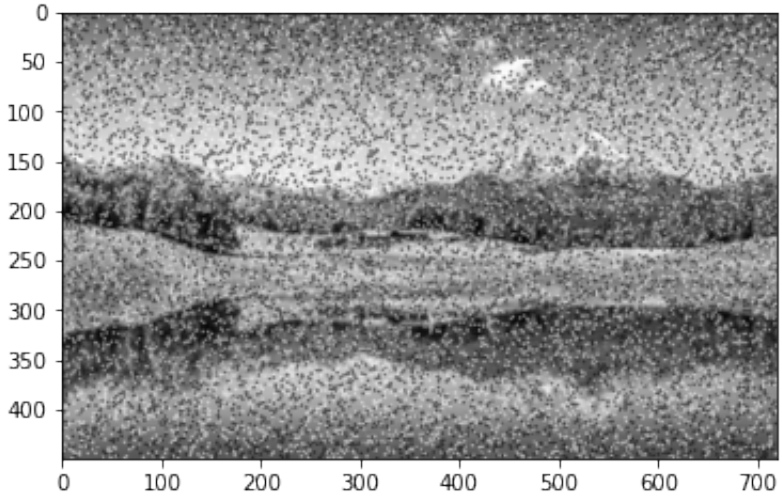
In [51]:
```
#Maximum filter
rows, cols = J1.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
max1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        max1[r, c] = np.max(pad_img[r:r+ksize, c:c+ksize])
max1 = (np.round(max1*255)).astype(int)
max1 = np.uint8(max1)
plt.figure()
plt.imshow(max1,cmap='gray')
plt.show()

#Maximum filter
rows, cols = J2.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
max2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        max2[r, c] = np.max(pad_img[r:r+ksize, c:c+ksize])
#max2 = np.uint8(max2)
max2 = (np.round(max2*255)).astype(int)
max2 = np.uint8(max2)
plt.figure()
plt.imshow(max2,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

In [52]:
```
#Midpoint filter
rows, cols = J1.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J1, *[padsize]*4, cv2.BORDER_DEFAULT)
mid1 = np.zeros_like(J1)
for r in range(rows):
    for c in range(cols):
        mid1[r, c] = 0.5*(np.min(pad_img[r:r+ksize, c:c+ksize]) + np.m
ax(pad_img[r:r+ksize, c:c+ksize]))
mid1 = (np.round(mid1*255)).astype(int)
mid1 = np.uint8(mid1)
plt.figure()
plt.imshow(mid1,cmap='gray')
plt.show()

#Midpoint filter
rows, cols = J2.shape
ksize = 3
padsize = int((ksize-1)/2)
pad_img = cv2.copyMakeBorder(J2, *[padsize]*4, cv2.BORDER_DEFAULT)
mid2 = np.zeros_like(J2)
for r in range(rows):
    for c in range(cols):
        mid2[r, c] = 0.5*(np.min(pad_img[r:r+ksize, c:c+ksize]) + np.m
ax(pad_img[r:r+ksize, c:c+ksize]))
mid2 = (np.round(mid2*255)).astype(int)
mid2 = np.uint8(mid2)
plt.figure()
plt.imshow(mid2,cmap='gray')
plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

In [ ]:

In [ ]: