07.2ex_Algorithms

March 14, 2020

1. **Maximum wind speed prediction at the Sprogø station**

The exercise goal is to predict the maximum wind speed occurring every 50 years even if no measure exists for such a period. The available data are only measured over 21 years at the Sprogø meteorological station located in Denmark.

The annual maxima are supposed to fit a normal probability density function. However such function is not going to be estimated because it gives a probability from a wind speed maxima. Finding the maximum wind speed occurring every 50 years requires the opposite approach, the result needs to be found from a defined probability. That is the quantile function role and the exercise goal will be to find it. In the current model, it is supposed that the maximum wind speed occurring every 50 years is defined as the upper 2% quantile.

By definition, the quantile function is the inverse of the cumulative distribution function. The latter describes the probability distribution of an annual maxima. In the exercise, the cumulative probability p_i for a given year i is defined as $p_i = i/(N+1)$ with $N = 21$, the number of measured years. Thus it will be possible to calculate the cumulative probability of every measured wind speed maxima. From those experimental points, the scipy.interpolate module will be very useful for fitting the quantile function. Finally the 50 years maxima is going to be evaluated from the cumulative probability of the 2% quantile.

Practically, load the dataset:

```
import numpy as np
max_speeds = np.load('max-speeds.npy')
years_nb = max_speeds.shape[0]
```

Compute then the cumulative probability p_i (`cprob`) and sort the maximum speeds from the data. Use then the UnivariateSpline from scipy.interpolate to define a quantile function and thus estimate the probabilities.

In the current model, the maximum wind speed occurring every 50 years is defined as the upper 2% quantile. As a result, the cumulative probability value will be:

```
fifty_prob = 1. - 0.02
```

So the storm wind speed occurring every 50 years can be guessed as:

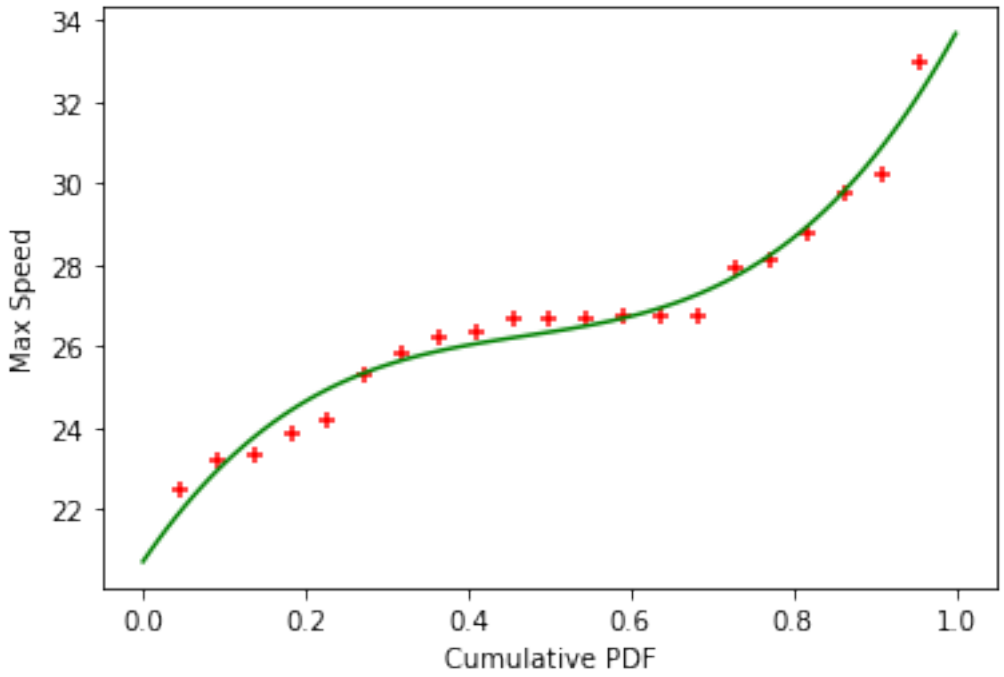
```
fifty_wind = quantile_func(fifty_prob)
```

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy.interpolate import UnivariateSpline
```

```
max_speeds = np.load('max-speeds.npy')
years_nb = max_speeds.shape[0]
cprob = (np.arange(years_nb, dtype=np.float32) + 1)/(years_nb + 1)
sorted_max_speeds = np.sort(max_speeds)
quantile_func = UnivariateSpline(cprob, sorted_max_speeds)
nprob = np.linspace(0, 1, 100)
fitted_max_speeds = quantile_func(nprob)
fifty_prob = 1. - 0.02
fifty_wind = quantile_func(fifty_prob)
print("Maximum speed: ",fifty_wind)

%matplotlib inline
plt.plot(np.linspace(0,1,100),quantile_func(np.linspace(0,1,100)), c="green" )
plt.scatter(cprob, sorted_max_speeds, marker="+", c="red")
plt.xlabel("Cumulative PDF")
plt.ylabel("Max Speed")
```

```
Maximum speed:  32.97989825386221
```

```
Out[1]: Text(0, 0.5, 'Max Speed')
```



2. **Curve fitting of temperature in Alaska**

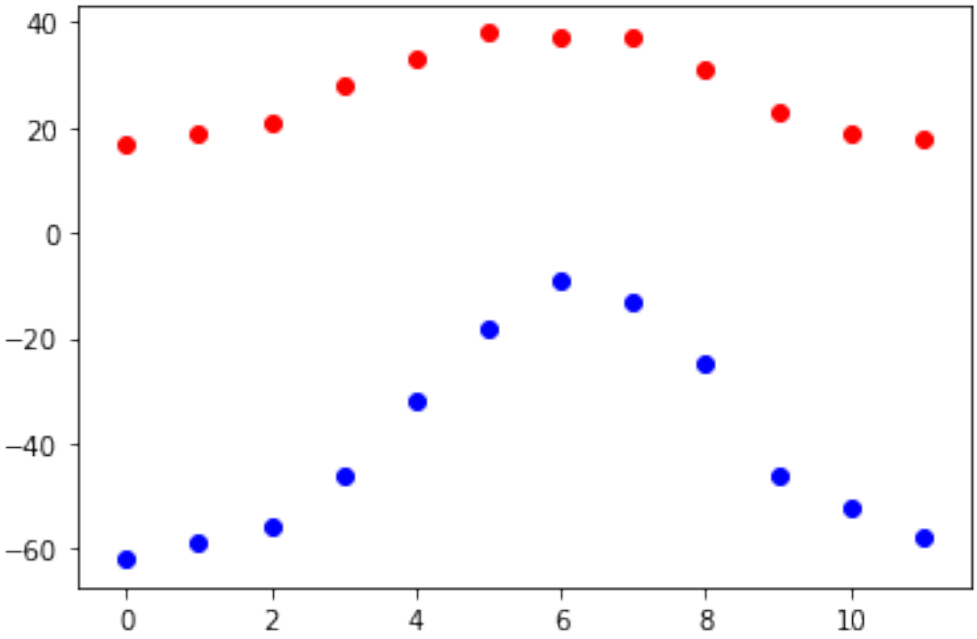
The temperature extremes in Alaska for each month, starting in January, are given by (in degrees Celcius):

max: 17, 19, 21, 28, 33, 38, 37, 37, 31, 23, 19, 18

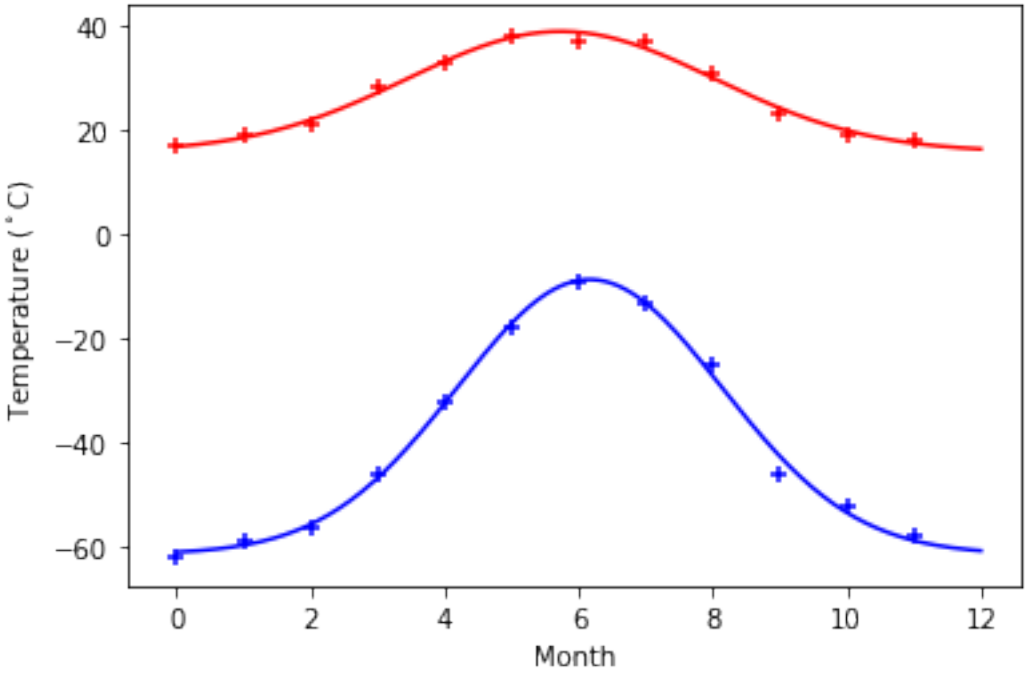
min: -62, -59, -56, -46, -32, -18, -9, -13, -25, -46, -52, -58

- Plot these temperature extremes.
- Define a function that can describe min and max temperatures.
- Fit this function to the data with scipy.optimize.curve_fit().
- Plot the result. Is the fit reasonable? If not, why?
- Is the time offset for min and max temperatures the same within the fit accuracy?

In [2]:
```
import scipy as sp
from scipy import optimize

temp_max = np.array([17,  19,  21,  28,  33,  38, 37,  37,  31,  23,  19,  18])
temp_min = np.array([-62, -59, -56, -46, -32, -18, -9, -13, -25, -46, -52, -58])

%matplotlib inline
plt.scatter(np.arange(12), temp_max, c="red", marker="o")
plt.scatter(np.arange(12), temp_min, c="blue", marker="o")
plt.show()

#i will use a gaussian with offset
#i don t know why but if i use the same curve one does not converge
def curve_temp_max(times, cost, mean, sigma, offset):
    return (cost*np.exp(-(times-(6+mean))**2/(2*sigma)**2)+offset)
def curve_temp_min(times, cost, mean, sigma, offset):
    return (cost*np.exp(-(times-mean)**2/(2*sigma)**2)+offset)
months = np.arange(12)
max_par, cov_max = optimize.curve_fit(curve_temp_max, months, temp_max)
min_par, cov_min = optimize.curve_fit(curve_temp_min, months, temp_min)

print(max_par)
print(min_par)

plt.figure()
plt.scatter(months, temp_max, c="red", marker="+")
plt.plot(np.linspace(0,12,365), curve_temp_max(np.linspace(0,12,365), *max_par), c="red
plt.scatter(months, temp_min, c="blue", marker="+")
plt.plot(np.linspace(0,12,365), curve_temp_min(np.linspace(0,12,365), *min_par), c="blu
plt.xlabel('Month')
plt.ylabel('Temperature ($^\circ$C)')

plt.show()
```




```
[23.09207078 -0.26418443  1.62240804 15.77934614]
[ 52.64929131   6.16452598   1.39717225 -61.32093406]
```



3. **2D minimization of a six-hump camelback function**

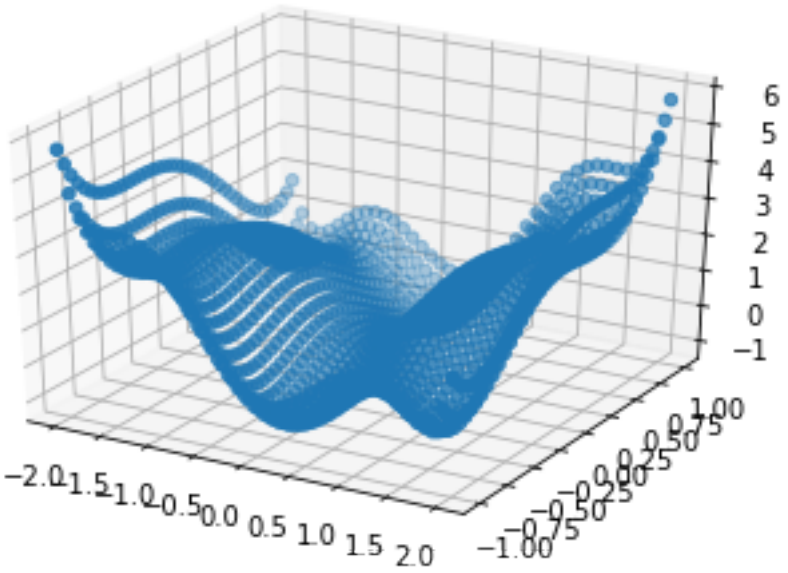
$$f(x,y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right) x^2 + xy + (4y^2 - 4)y^2$$

has multiple global and local minima. Find the global minima of this function.
Hints:

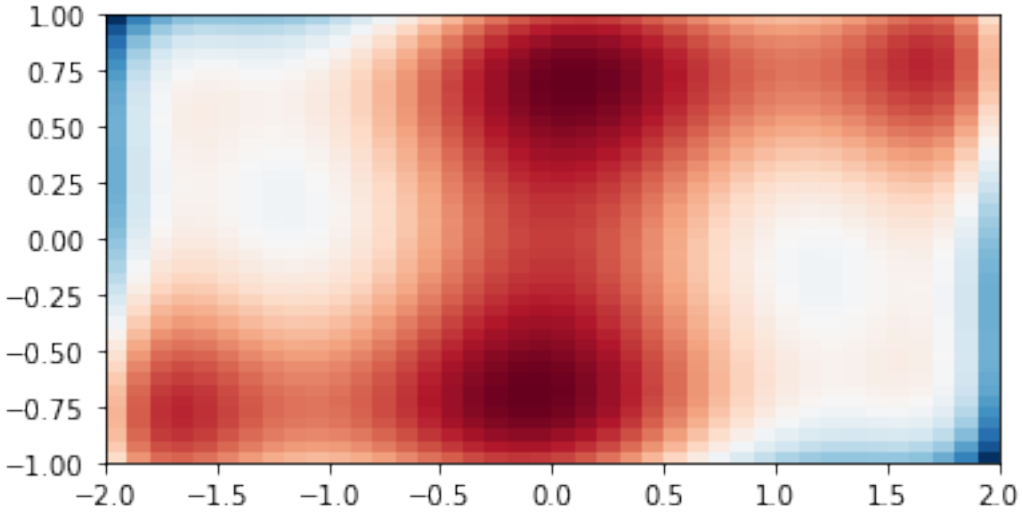
- Variables can be restricted to $-2 < x < 2$ and $-1 < y < 1$.
- Use numpy.meshgrid() and pylab.imshow() to find visually the regions.
- Use scipy.optimize.minimize(), optionally trying out several of its methods.

How many global minima are there, and what is the function value at those points? What happens for an initial guess of $(x,y) = (0,0)$?

```
In [3]: import scipy as sp
        from scipy.optimize import minimize

        def f(x,y):
            return (4-2.1*x**2+(x**4)/3.0)*x**2 + x*y + 4*(y**2-1)*y**2
        def f2(coord):
            x=coord[0]
            y=coord[1]
            return (4-2.1*x**2+(x**4)/3.0)*x**2 + x*y + 4*(y**2-1)*y**2

        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D as plt3d

        n=40
        x=np.linspace(-2,2,n)
        y=np.linspace(-1,1,n)
        x,y=np.meshgrid(x,y)
        z=np.array(f(x,y))
        #print(x)
        #print(y)
        #print(z)

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(x,y,z)
        plt.show()

        plt.imshow(z,extent=[-2,2,-1,1], cmap='RdBu')
```



Out[3]: <matplotlib.image.AxesImage at 0x7efbb2821fd0>



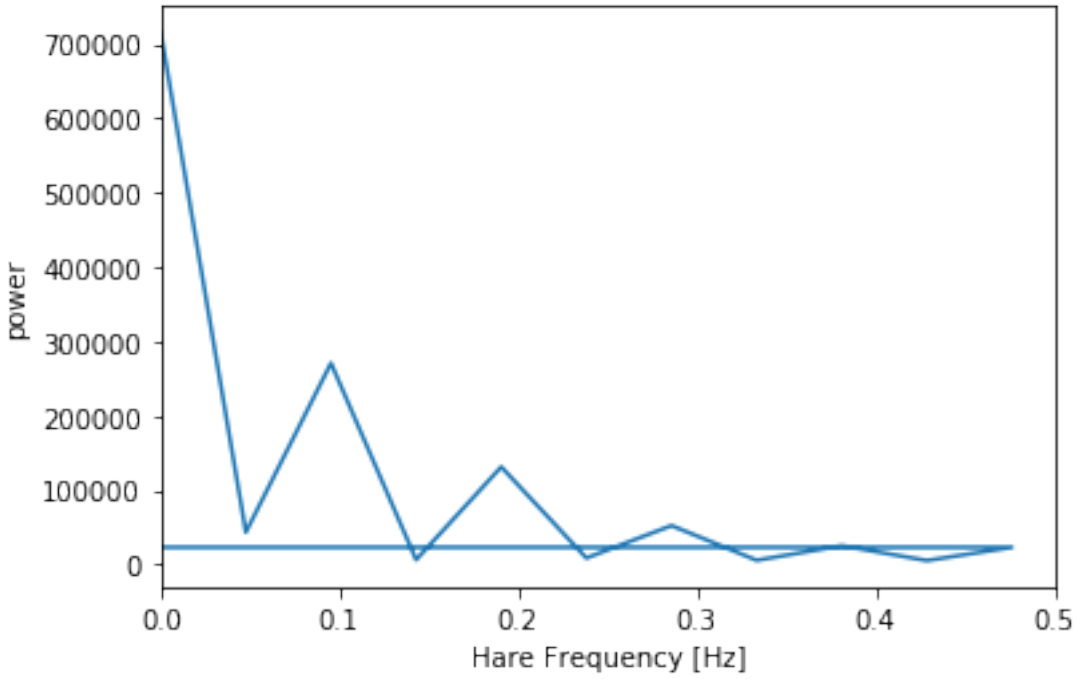
In [4]:
```
limits=[(-2,2)]
minima=minimize(f2,limits)
print(minima)
print("minimum coords: ", minima.x)
print("minimum value:  ", minima.fun)
```

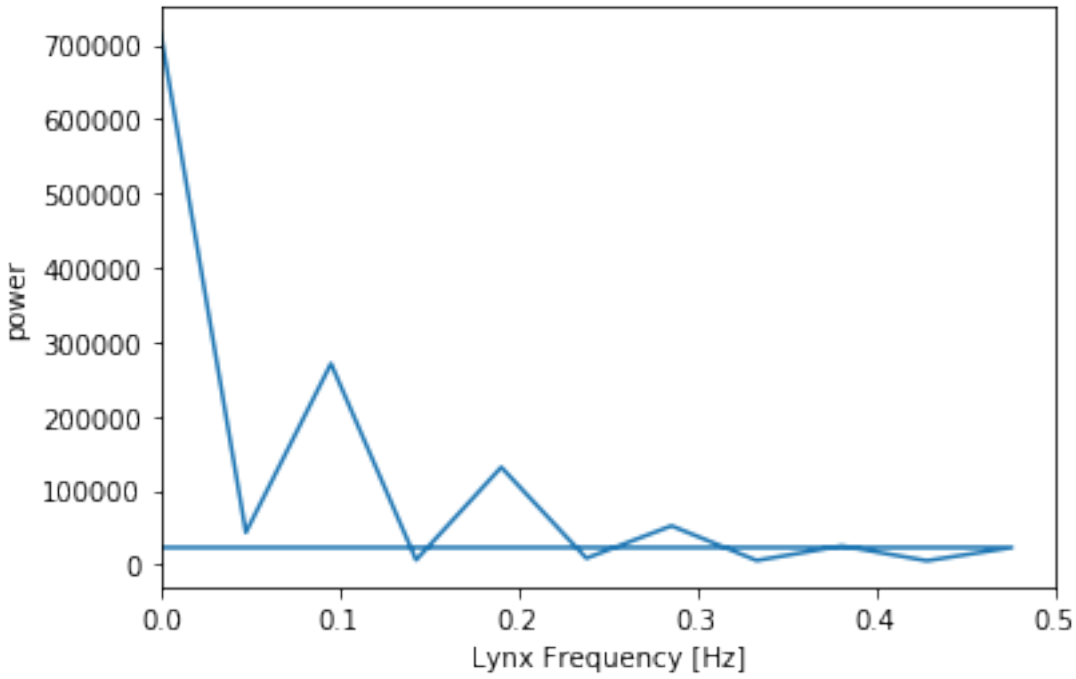
```
      fun: -1.0316284534877955
 hess_inv: array([[ 0.12502046, -0.00569839],
       [-0.00569839,  0.0603727 ]])
      jac: array([ 4.45544720e-06, -4.55975533e-06])
  message: 'Optimization terminated successfully.'
     nfev: 48
      nit: 10
     njev: 12
   status: 0
  success: True
        x: array([-0.08984141,  0.71265608])
minimum coords:  [-0.08984141  0.71265608]
minimum value:   -1.0316284534877955
```

4. **FFT of a simple dataset**
Performe a periodicity analysis on the lynxs-hares population

```
In [5]: import numpy as np
        import pandas as pd
        from scipy import fftpack
        from matplotlib import pyplot as plt

        data=pd.read_csv("populations.txt", delimiter="\t")

        hare_fft = fftpack.fft(data["hare"])
        hare_pow =np.abs(hare_fft)
        lynx_fft = fftpack.fft(data["hare"])
        lynx_pow =np.abs(lynx_fft)

        sample_freq = fftpack.fftfreq(data["# year"].size)

        # Plot the FFT power

        plt.plot(sample_freq, hare_pow)
        plt.xlabel('Hare Frequency [Hz]')
        plt.ylabel('power')
        plt.xlim([0,0.5])
        plt.show()

        plt.plot(sample_freq, lynx_pow)
        plt.xlabel('Lynx Frequency [Hz]')
        plt.ylabel('power')
        plt.xlim([0,0.5])
        plt.show()
```

```
/home/alessandro/anaconda3/lib/python3.7/site-packages/scipy/fft/_pocketfft/helper.py:95: Futu
  return np.array(x, copy=not x.flags['ALIGNED'])
```



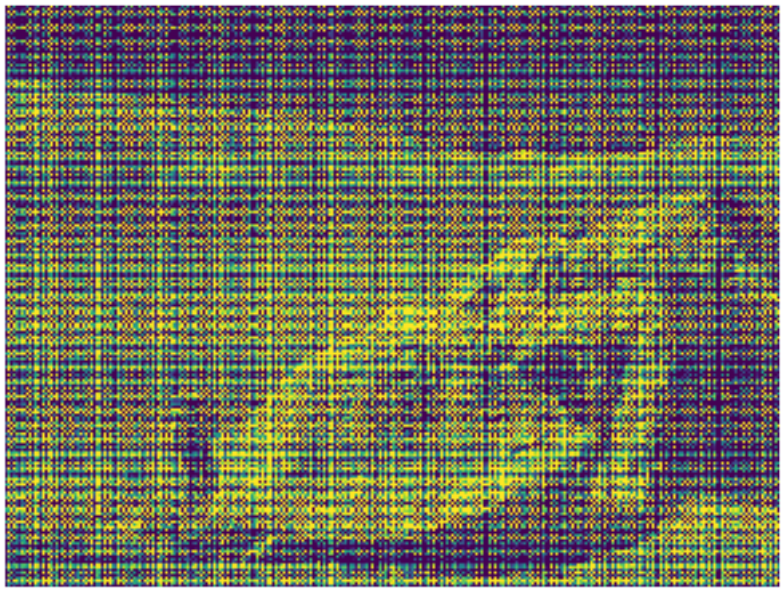


5. **FFT of an image**

- Examine the provided image `moonlanding.png`, which is heavily contaminated with periodic noise. In this exercise, we aim to clean up the noise using the Fast Fourier Transform.

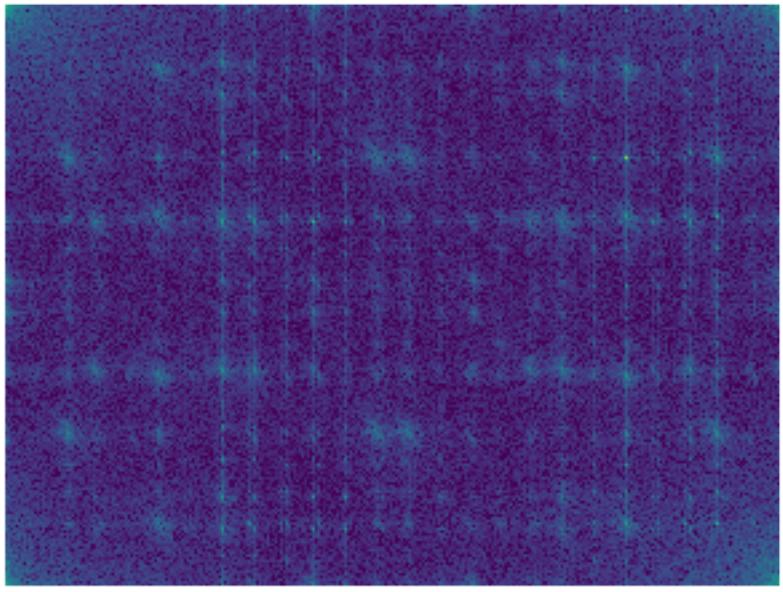
- Load the image using pylab.imread().
- Find and use the 2-D FFT function in scipy.fftpack, and plot the spectrum (Fourier transform of) the image. Do you have any trouble visualising the spectrum? If so, why?
- The spectrum consists of high and low frequency components. The noise is contained in the high-frequency part of the spectrum, so set some of those components to zero (use array slicing).
- Apply the inverse Fourier transform to see the resulting image.

```
In [6]: import scipy as sp
        import numpy as np
        from scipy import fftpack
        import pylab
        import matplotlib.pyplot as plt

        image=pylab.imread("moonlanding.png")
        image_tuple=tuple(image)

        plt.imshow(image)
        plt.axis("off")
        plt.show()

        image_fft=fftpack.fft2(image_tuple)
        plt.imshow(np.abs(image_fft))
        plt.axis("off")
        plt.show()
```



The color map is not able to show anything since the values are probably too spread, i have to use a logarithmic color map

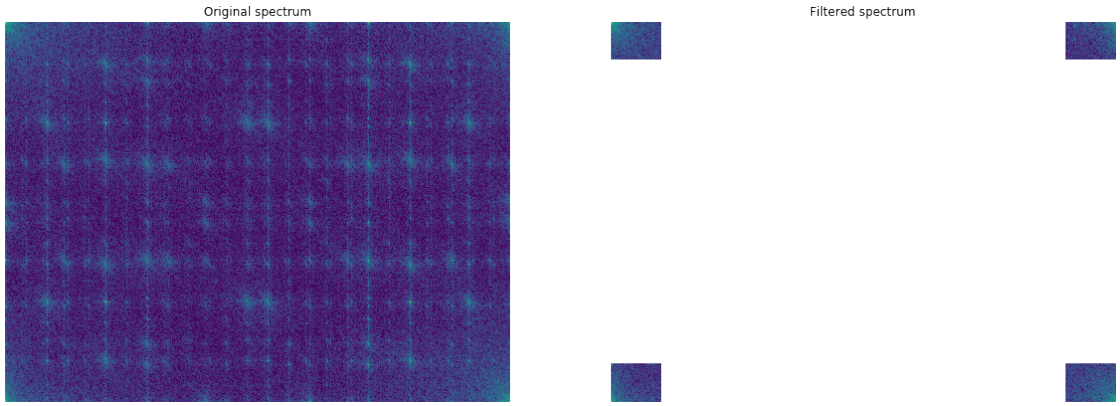
```
In [7]: from matplotlib.colors import LogNorm
        plt.imshow(np.abs(image_fft), norm=LogNorm(vmin=10))
        plt.axis("off")
        plt.show()
```

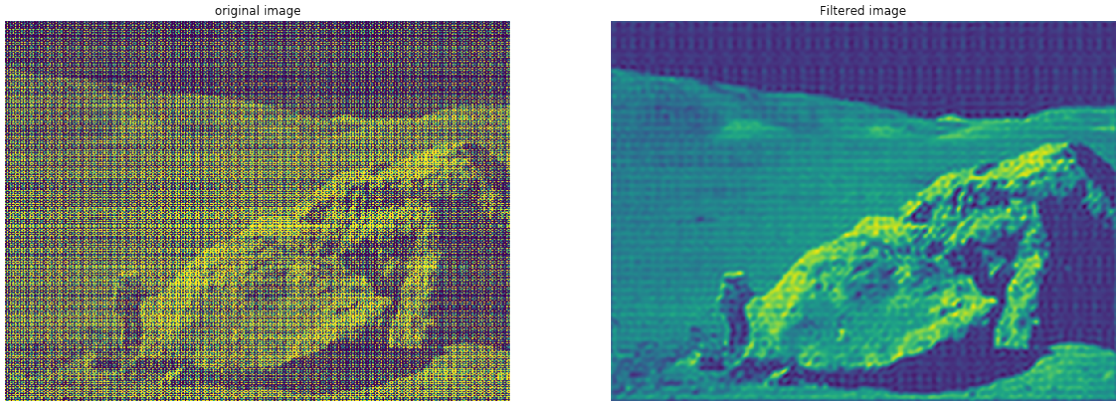


In [8]:
```
keep=0.1
image_copy=image_fft.copy()

nrow, ncol = image_copy.shape
image_copy[int(nrow*keep):int(nrow*(1-keep))] = 0
image_copy[:, int(ncol*keep):int(ncol*(1-keep))] = 0
```

In [9]:
```
fig, ax= plt.subplots(nrows=2, ncols=2, figsize=(20,20))

ax[0,0].imshow(np.abs(image_fft), norm=LogNorm(vmin=10))
ax[0,0].axis("off")
ax[0,0].set_title("Original spectrum")

ax[0,1].imshow(np.abs(image_copy), norm=LogNorm(vmin=10))
ax[0,1].axis("off")
ax[0,1].set_title("Filtered spectrum")

ax[1,0].imshow(np.abs(image))
ax[1,0].axis("off")
ax[1,0].set_title("original image")

ax[1,1].imshow(fftpack.ifft2(image_copy).real)
ax[1,1].axis("off")
ax[1,1].set_title("Filtered image")
```

Out[9]:
```
Text(0.5, 1.0, 'Filtered image')
```





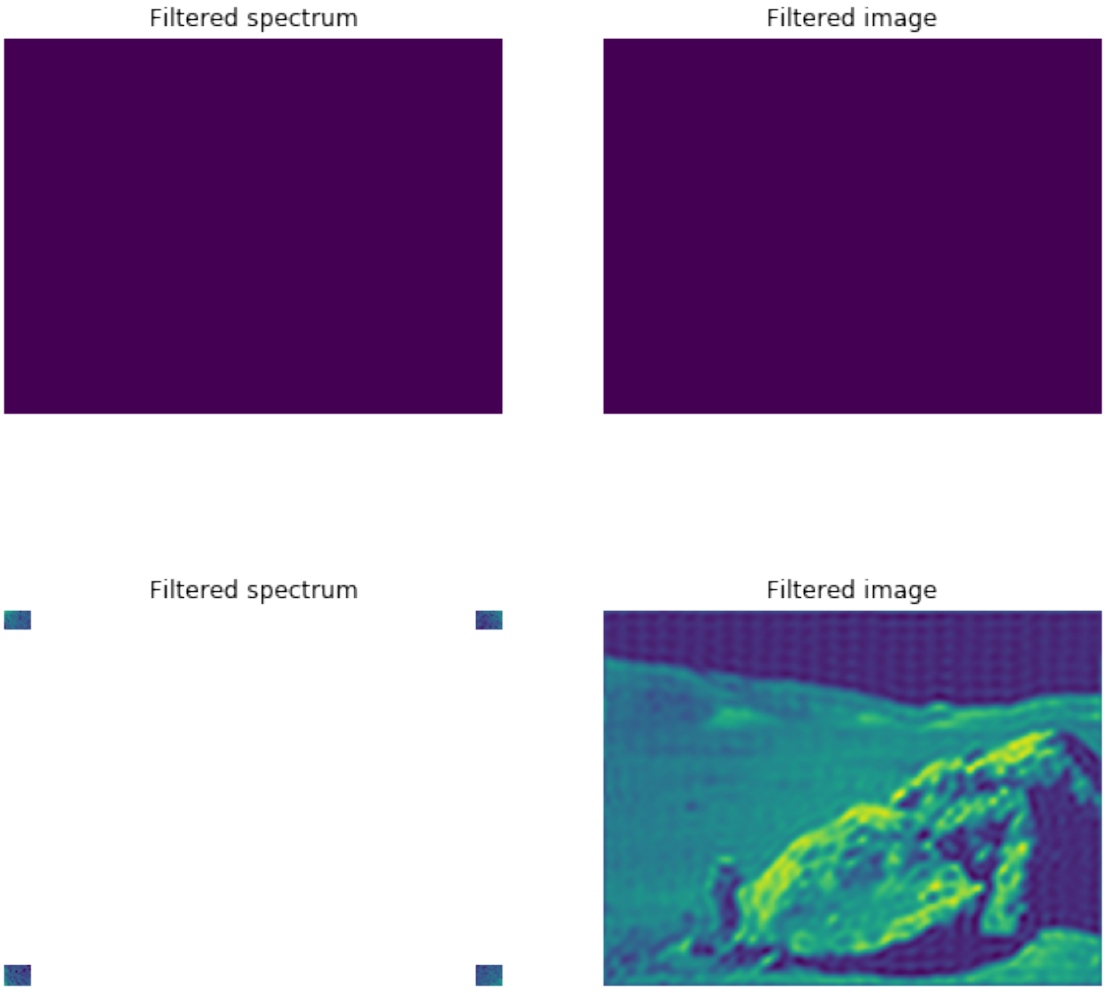
we can see how this works for many values of keep

```
In [10]: keeps=np.linspace(0,0.5, 10)
```

```
for k in keeps:
    image_copy=image_fft.copy()
    nrow, ncol = image_copy.shape
    image_copy[int(nrow*k):int(nrow*(1-k))] = 0
    image_copy[:, int(ncol*k):int(ncol*(1-k))] = 0
    fig, ax= plt.subplots(nrows=1, ncols=2, figsize=(10, 20))
    ax[0].imshow(np.abs(image_copy), norm=LogNorm(vmin=10))
    ax[0].axis("off")
    ax[0].set_title("Filtered spectrum")
    ax[1].imshow(fftpack.ifft2(image_copy).real)
    ax[1].axis("off")
```

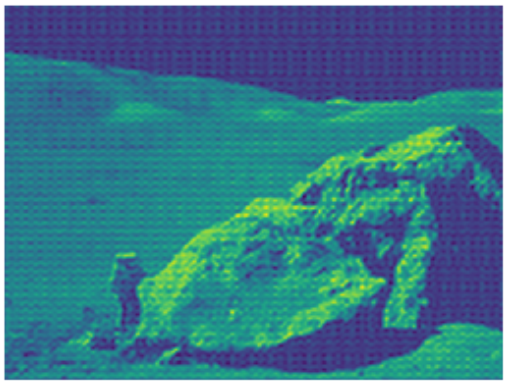
```
            ax[1].set_title("Filtered image")
            fig.show()
```

```
/home/alessandro/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:17: UserWarning: I
/home/alessandro/anaconda3/lib/python3.7/site-packages/matplotlib/image.py:397: UserWarning: Wa
  dv = (np.float64(self.norm.vmax) -
/home/alessandro/anaconda3/lib/python3.7/site-packages/matplotlib/colors.py:1110: RuntimeWarnin
  mask |= resdat <= 0
/home/alessandro/anaconda3/lib/python3.7/site-packages/matplotlib/colors.py:1114: RuntimeWarnin
  resdat /= (np.log(vmax) - np.log(vmin))
/home/alessandro/anaconda3/lib/python3.7/site-packages/matplotlib/colors.py:527: RuntimeWarning
  xa[xa < 0] = -1
```



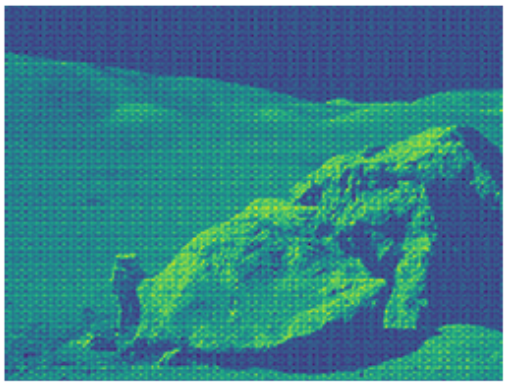
Filtered spectrum

Filtered image



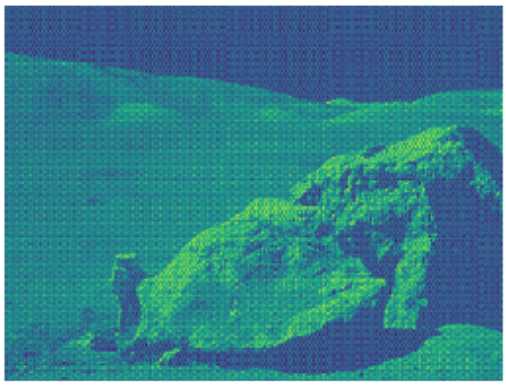
Filtered spectrum

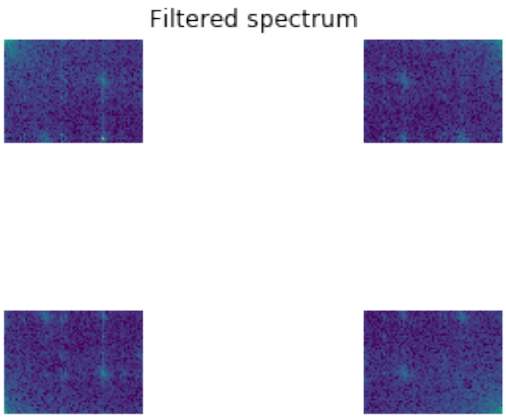
Filtered image

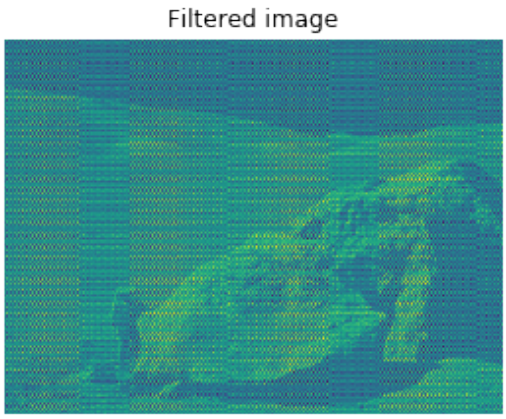


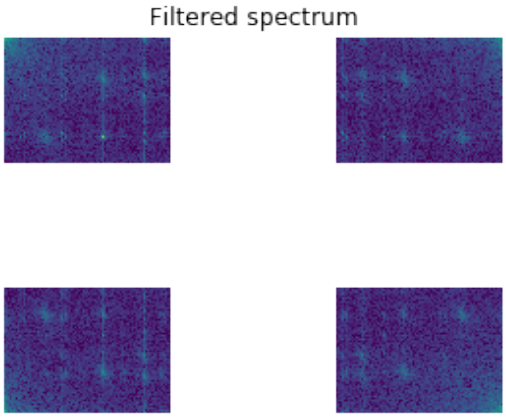
Filtered spectrum

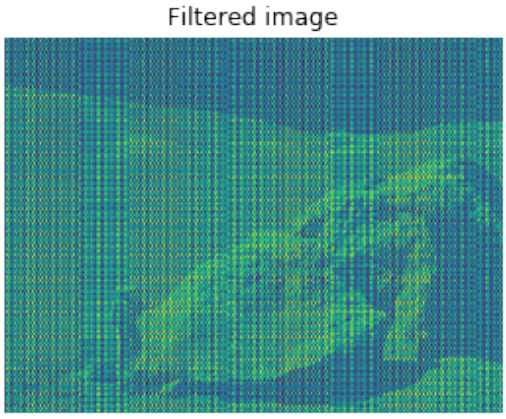
Filtered image

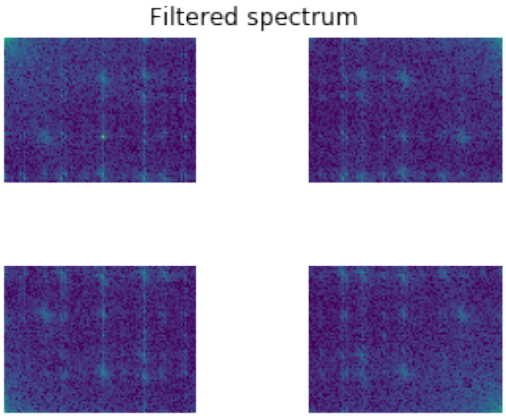


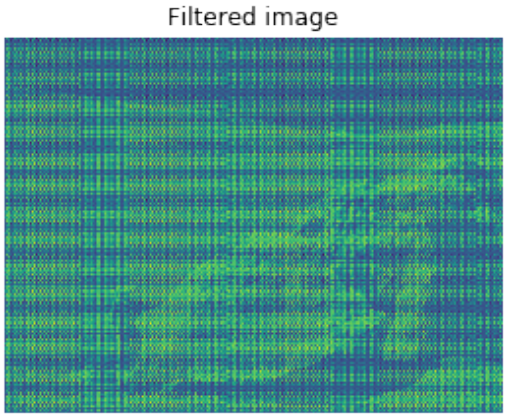

Filtered spectrum

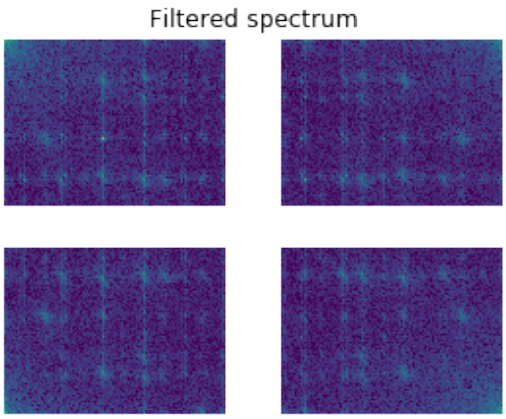

Filtered image

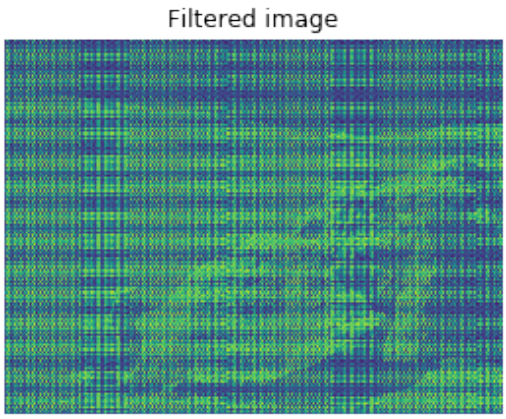

Filtered spectrum

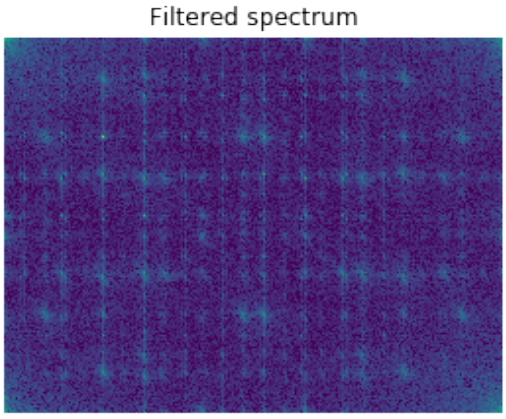

Filtered image

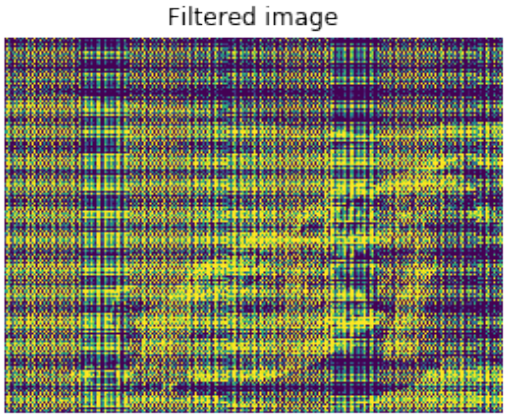

Filtered spectrum


Filtered image


Filtered spectrum


Filtered image


Filtered spectrum


Filtered image

In [ ]: