Fitting Blazar Light Curves with Gaussian Modelling

R Abhay Chand, MS20023

May 26, 2024

Gaussian Distribution

Gaussian distribution is a continuous probability distribution for a real-random variable. The general form of a Gaussian is:

$$G = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \frac{-(x-\mu)^2}{2\sigma^2} \tag{1}$$

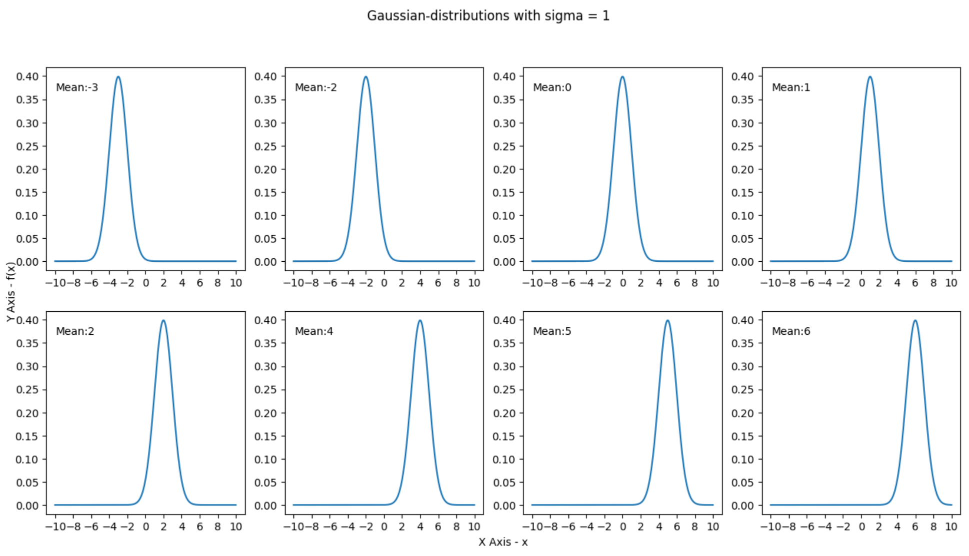
It has two parameters mean(mu) and standard deviation(sigma) which dictates the shape of the curve. The curve is centered around the mean and the standard deviation is the measure of the spread of data points around the mean.

Gaussian functions vary useful in variety of scientific fields. In machine learning Gaussian distribution is used to model noise . Noises are independently and identically distributed random variable,i.e. they have the same probability distribution(same mean and sigma) and are mutually independent. Moreover, as Gaussian integrals are easy to evaluate.

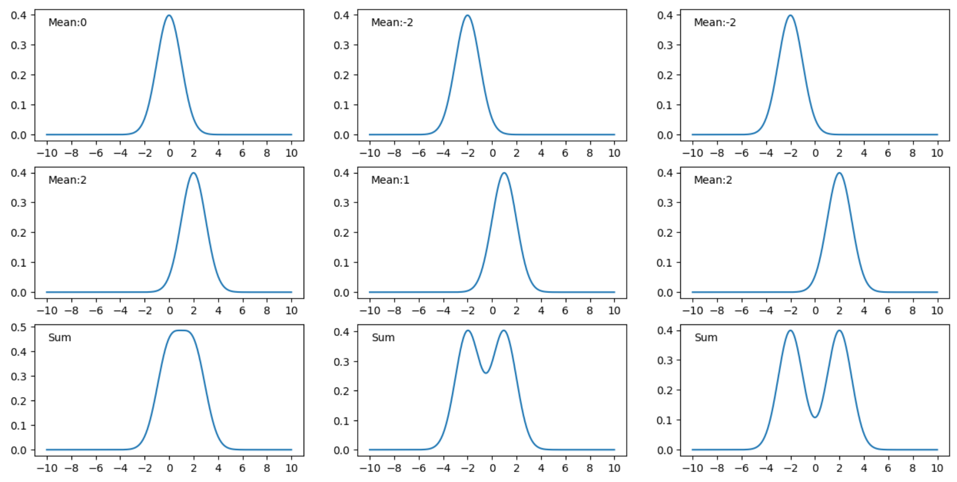
According to Do and Lee (2007)Some properties of Gaussian include :

1. Normalization - Integral of Gaussian function from -infinity to +infinity is 1.
2. Marginalisation - The marginal density of a multivariate Gaussian distribution is also Gaussian.
3. Conditioning - The conditional probabilities that we obtain from a multivariate Gaussian is also Gaussian.

1 To look at Gaussian distributions with different means

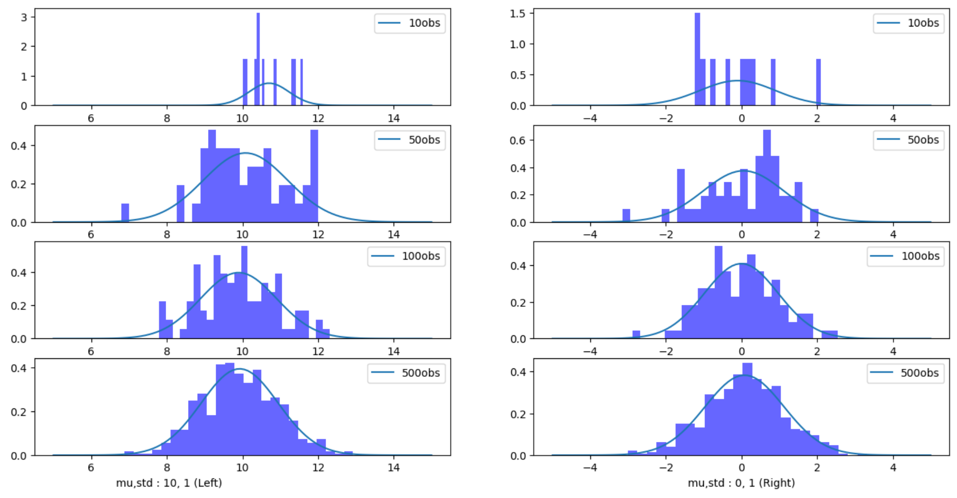


2 To look at the sum of Gaussian distributions with different means

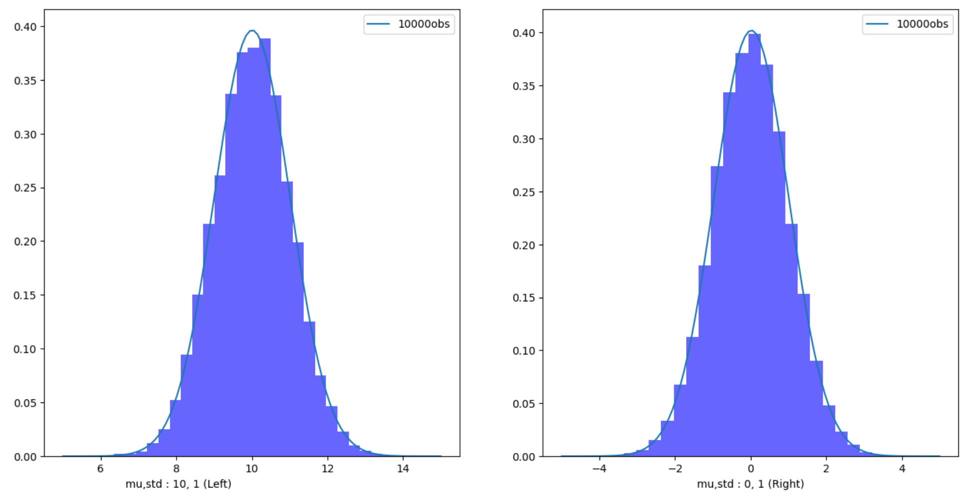


We can see that the sum of two G.D s does not always represent a Gaussian.

3 To obtain random numbers from a Gaussian distribution and analyse it's histogram



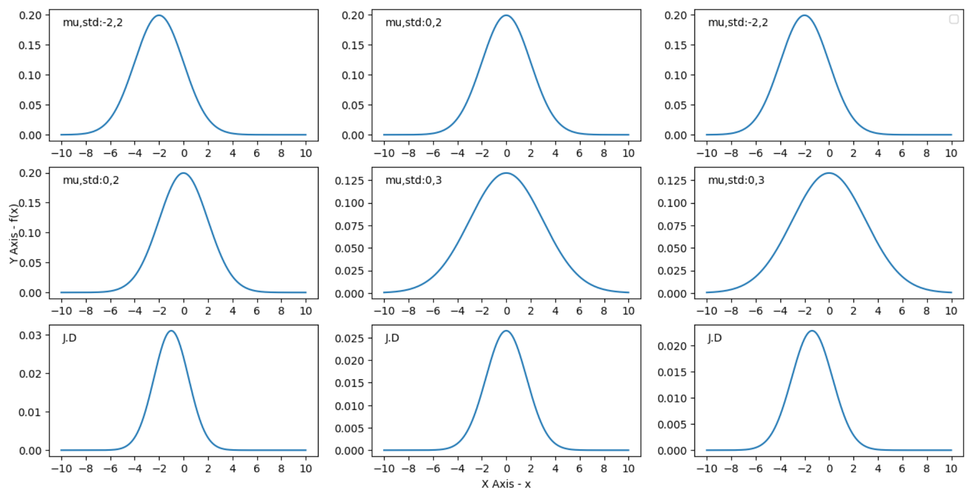
As the number of observations increases , the histogram starts to represent a Gaussian. At around 10000 observations the histogram plot closely resembles a Gaussian :



4 To analyse the joint probability distribution of G.D s

Joint Probability distribution is the probability of two events occurring simultaneously at the same time. If X and Y are two events then the probability density of $X \cap Y$ gives the joint probability distribution. For two independent events X and Y, the joint probability distribution is given by f(x)f(y) where , f(x) and f(y) are the probability of occurrence of X and Y respectively.

In the following figure, the bottom graph represents the joint probability distribution of above two graphs.



Conclusion: The joint probability distribution of G.D.s are always assume a Gaussian form.

5 Further Questions Addressed

Find the theoretical basis for product of G.D s being Gaussian for the following cases :

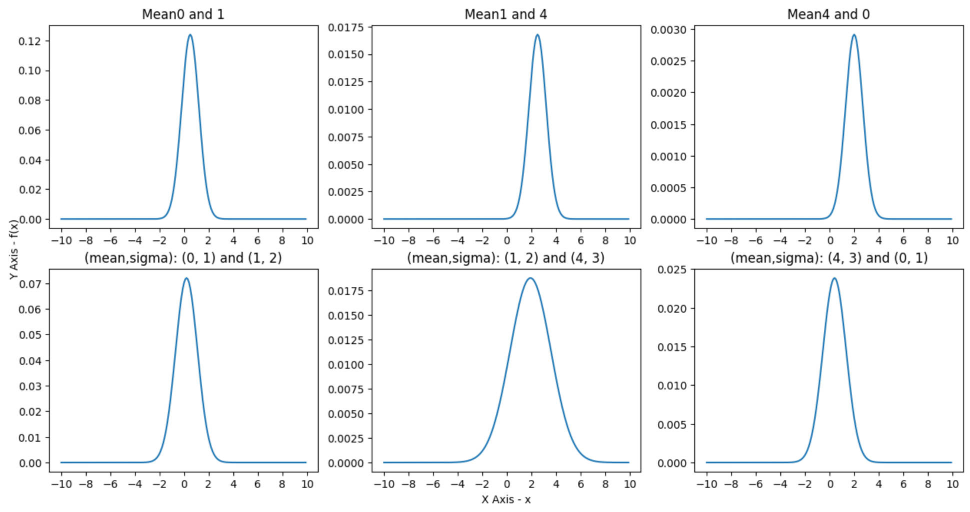
G.D s with

1. different means , and sigma same
2. Both mean and sigma different

Give theoretical plots for the above and verify geometrically.

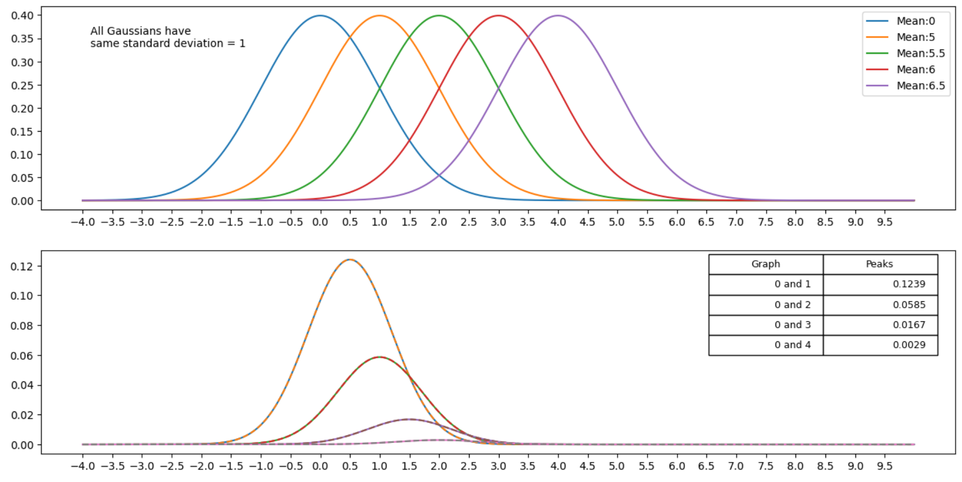
Ans.The theory behind the product of Gaussian distributions being Gaussian in the following PDF : https://drive.google.com/file/d/1uKuADPo7kYzrxjCpTPfLpxmv6QA1hpsQ/view?usp=sharing

The first row of plots show the product of Gaussian distributions with different means and same standard deviation. The second rows is the product of Gaussians with different means and standard deviation.



The product of Gaussians in different cases also looks like a Gaussian.

Following plot shows the product of various Gaussians:



Graph	Peaks
0 and 1	0.1239
0 and 2	0.0585
0 and 3	0.0167
0 and 4	0.0029

From the above figure, we can conclude that the theoretical plot of the product of Gaussian agrees with the actual observation. In this case , the standard deviation of each Gaussian is 1 and therefore the product of Gaussians have a mean which is the average of the two parent Gaussians and the standard deviation is $1/\sqrt{2}$.

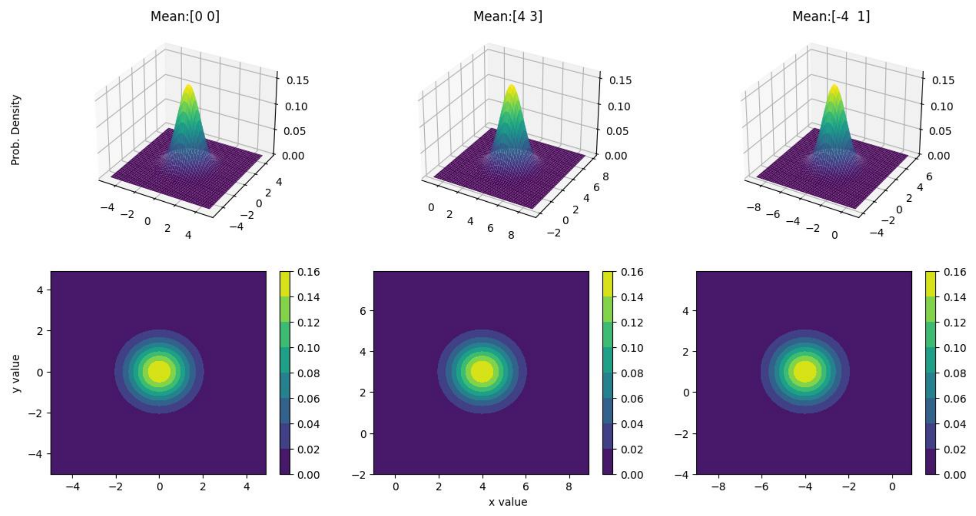
6 Multivariate Gaussian

Multivariate Gaussian can be represented as :

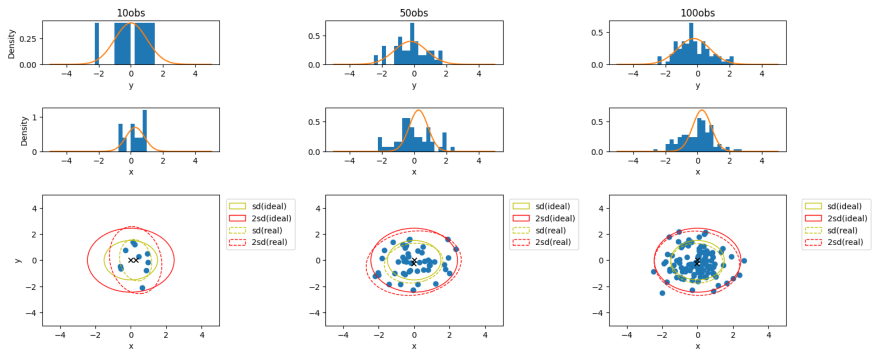
$$G = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) \tag{2}$$

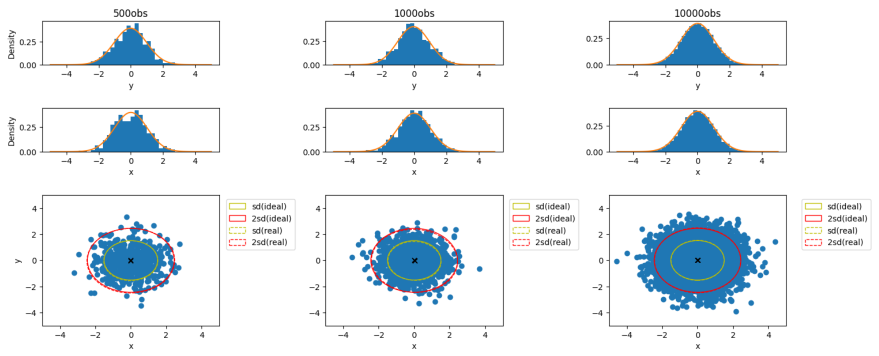
,where n is the number of independent variables, μ is the n dimensional mean , Σ is the n x n variance matrix

Following are the probability density distribution of the Gaussians with same covariance matrix - $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ but different means:



Different random samples were drawn from the same normal distribution with mean $\begin{pmatrix} 0 & 0 \end{pmatrix}$ and covariance matrix: $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ to obtain the following plots:





sd stands for standard deviation.

```
import numpy as np
from numpy import linalg as la
import matplotlib.pyplot as plt
from scipy.stats import norm
from matplotlib.patches import Ellipse

def plot_confidence_ellipse(mu, cov, alph, ax, clabel=None, label_bg='white', **
    ↪ kwargs):
    """Display a confidence ellipse of a bivariate normal distribution

    Arguments:
        mu {array-like of shape (2,)} -- mean of the distribution
        cov {array-like of shape(2,2)} -- covariance matrix
        alph {float btw 0 and 1} -- level of confidence
        ax {plt.Axes} -- axes on which to display the ellipse
        clabel {str} -- label to add to ellipse (default: {None})
        label_bg {str} -- background of clabel's textbox
        kwargs -- other arguments given to class Ellipse
    """
    c = -2 * np.log(1 - alph) # quantile at alpha of the chi_squarred distr. with df
        ↪  = 2
    Lambda, Q = la.eig(cov) # eigenvalues and eigenvectors (col. by col.)

    ## Compute the attributes of the ellipse
    width, height = 2 * np.sqrt(c * Lambda)
    # compute the value of the angle theta (in degree)
    theta = 180 * np.arctan(Q[1, 0] / Q[0, 0]) / np.pi if cov[1,0] else 0

    ## Create the ellipse
    if 'fc' not in kwargs.keys():
        kwargs['fc'] = 'None'
    level_line = Ellipse(mu, width, height, angle=theta, **kwargs)

    ## Display a label 'clabel' on the ellipse
    if clabel:
        col = kwargs['ec'] if 'ec' in kwargs.keys() and kwargs['ec'] != 'None' else '
            ↪ black' # color of the text
        pos = Q[:, 1] * np.sqrt(c * Lambda[1]) + mu # position along the heigth

        ax.text(*pos, clabel, color=col,
              rotation=theta, ha='center', va='center', rotation_mode='anchor', #
                  ↪ rotation
              bbox=dict(boxstyle='round', ec='None', fc=label_bg, alpha=1)) # white
                  ↪ box

    return ax.add_patch(level_line)

#Declaring Mean and Covarience
mean = np.array([0,0])
cov = np.array([[1,0],[0,1]])

#Creating Multinomial Gaussian Variable
x1,y1 = np.random.multivariate_normal(mean, cov, 10).T
x2,y2 = np.random.multivariate_normal(mean, cov, 50).T
x3,y3 = np.random.multivariate_normal(mean, cov, 100).T

# Original plot
fig = plt.figure()

# Creating Scatter Plots
#500 obs
ax31 = fig.add_axes([0.05, 0.1, 0.2, 0.3])
ax31.scatter(x1,y1)
plot_confidence_ellipse(mean, cov, 0.68, ax31 ,ec='y',label='sd(ideal)')
ax31.plot(*mean, 'kx')
plot_confidence_ellipse(mean, cov, 0.95, ax31 ,ec='r',label='2sd(ideal)')
ax31.plot(*mean, 'kx')
ax31.set_xlabel('x')
ax31.set_ylabel('y')
#1000 obs
ax33 = fig.add_axes([0.37, 0.1, 0.2, 0.3])
ax33.scatter(x2,y2)
plot_confidence_ellipse(mean, cov, 0.68, ax33 ,ec='y',label='sd(ideal)')
ax33.plot(*mean, 'kx')
plot_confidence_ellipse(mean, cov, 0.95, ax33 ,ec='r',label='2sd(ideal)')
ax33.plot(*mean, 'kx')
ax33.set_xlabel('x')
#10000 obs
ax35 = fig.add_axes([0.69, 0.1, 0.2, 0.3])
ax35.scatter(x3,y3)
plot_confidence_ellipse(mean, cov, 0.68, ax35 ,ec='y',label='sd(ideal)')
ax35.plot(*mean, 'kx')
plot_confidence_ellipse(mean, cov, 0.95, ax35 ,ec='r',label='2sd(ideal)')
ax35.plot(*mean, 'kx')
ax35.set_xlabel('x')

#Reconstructing Gaussian
#500obs
data1 = list(zip(x1,y1))
mean1 = np.mean(data1,axis=0)
cov1 = np.cov(data1,rowvar=False)
plot_confidence_ellipse(mean1, cov1, 0.68, ax31 ,ec='y',linestyle='dashed',label='sd
    ↪ (real)')
ax31.plot(*mean1, 'kx')
plot_confidence_ellipse(mean1, cov1, 0.95, ax31 ,ec='r',linestyle='dashed',label='2
    ↪ sd(real)')
```

```
ax31.plot(*mean1, 'kx')
ax31.legend(loc='upper right',bbox_to_anchor =(1.45, 1))
ax31.set_xlim(-5,5)
ax31.set_ylim(-5,5)
#10000obs
data2 = list(zip(x2,y2))
mean2 = np.mean(data2,axis=0)
cov2 = np.cov(data2,rowvar=False)
plot_confidence_ellipse(mean2, cov2, 0.68, ax33 ,ec='y',linestyle='dashed',label='sd
    ↪ (real)')
ax33.plot(*mean2, 'kx',linestyle='dashed')
plot_confidence_ellipse(mean2, cov2, 0.95, ax33 ,ec='r',linestyle='dashed',label='2
    ↪ sd(real)')
ax33.plot(*mean2, 'kx')
ax33.legend(loc='upper right',bbox_to_anchor =(1.45,1))
ax33.set_xlim(-5,5)
ax33.set_ylim(-5,5)
#10000obs
data3 = list(zip(x3,y3))
mean3 = np.mean(data3,axis=0)
cov3 = np.cov(data3,rowvar=False)
plot_confidence_ellipse(mean3, cov3, 0.68, ax35 ,ec='y',linestyle='dashed',label='sd
    ↪ (real)')
ax35.plot(*mean3, 'kx')
plot_confidence_ellipse(mean3, cov3, 0.95, ax35 ,ec='r',linestyle='dashed',label='2
    ↪ sd(real)')
ax35.plot(*mean3, 'kx')
ax35.legend(loc='upper right',bbox_to_anchor =(1.5, 1))
ax35.set_xlim(-5,5)
ax35.set_ylim(-5,5)

#Creating Histograms
#500obs
#Y-Axis
ax32 = fig.add_axes([0.05, 0.7, 0.2, 0.1])
ax32.hist(y1,40, histtype='stepfilled', orientation='vertical',range=[-5,5],density=
    ↪ True)
ax32.set_xlabel('y')
ax32.set_ylabel('Density')
ax32.set_title('10obs')
#X-Axis
ax21 = fig.add_axes([0.05, 0.5, 0.2, 0.1])
ax21.hist(x1,40, histtype='stepfilled', orientation='vertical',range=[-5,5],density=
    ↪ True)
ax21.set_xlabel('x')
ax21.set_ylabel('Density')
#10000obs
#Y-Axis
ax34 = fig.add_axes([0.37, 0.7, 0.2, 0.1])
ax34.hist(y2,40, histtype='stepfilled', orientation='vertical',range=[-5,5],density=
    ↪ True)
ax34.set_xlabel('y')
ax34.set_title('50obs')
#X-Axis
ax22 = fig.add_axes([0.37, 0.5, 0.2, 0.1])
ax22.hist(x2,40, histtype='stepfilled', orientation='vertical',range=[-5,5],density=
    ↪ True)
ax22.set_xlabel('x')
#10000obs
#Y-Axis
ax36 = fig.add_axes([0.69, 0.7, 0.2, 0.1])
ax36.hist(y3,40, histtype='stepfilled', orientation='vertical',range=[-5,5],density=
    ↪ True)
ax36.set_xlabel('y')
ax36.set_title('100obs')
#X-Axis
ax23 = fig.add_axes([0.69, 0.5, 0.2, 0.1])
ax23.hist(x3,40, histtype='stepfilled', orientation='vertical',range=[-5,5],density=
    ↪ True)
ax23.set_xlabel('x')

#Creating Gaussians

#500obs
#Y-Axis
my, sdy = norm.fit(y1)
ymin, ymax = -5, 5
y = np.linspace(ymin, ymax, 100)
py = norm.pdf(y, my, sdy)
ax32.plot(y, py)
#X-Axis
mx, sdx = norm.fit(x1)
xmin, xmax = -5, 5
x = np.linspace(xmin, xmax, 100)
px = norm.pdf(x, mx, sdx)
ax21.plot(x, px)
#10000obs
#Y-Axis
my, sdy = norm.fit(y2)
ymin, ymax = -5, 5
y = np.linspace(ymin, ymax, 100)
py = norm.pdf(y, my, sdy)
ax34.plot(y, py)
#X-Axis
mx2, sdx2 = norm.fit(x2)
xmin, xmax = -5, 5
x = np.linspace(xmin, xmax, 100)
px = norm.pdf(x, mx, sdx)
ax22.plot(x, px)
#1000obs
#Y-Axis
my, sdy = norm.fit(y3)
ymin, ymax = -5, 5
y = np.linspace(ymin, ymax, 100)
py = norm.pdf(y, my, sdy)
ax36.plot(y, py)
#X-Axis
mx3, sdx3 = norm.fit(x3)
xmin, xmax = -5, 5
x = np.linspace(xmin, xmax, 100)
px = norm.pdf(x, mx, sdx)
ax23.plot(x, px)
```

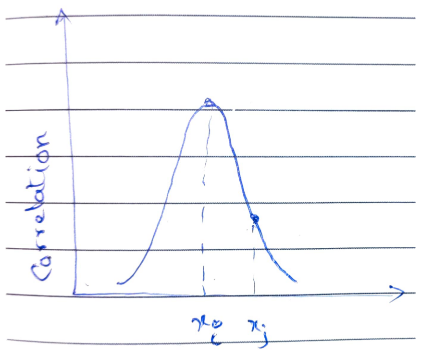
```
plt.show()
```

Q. Why do we use multi-variate Gaussian for time series ?

Gaussian process is a non-parametric tool ,i.e. it does not follow a fixed model and therefore can be used to model data about which we have very little knowledge, like astronomical data. The data obtained by telescopes may have noise or the multiple signals , which demands marginalisation or elimination of unwanted signals. Moreover, the data obtained might not be regular and hence we need to extrapolate or interpolate the obtained data. Gaussian processes can be used for all these functions.

In most cases, the error bar, or the error in the measurement of a point in astronomical data corresponds to a Gaussian.

Let's assume that the time series series of a astronomical data can be modelled using a continuous curve. Consider two points x1 and x2. If Δ x $\rightarrow$ 0 then $f(x1) = f(x2)$. We say that x1 and x2 are highly correlated. As Δ x >> 0 then f(x1) and f(x2) become significantly different and therefore the correlation between them approaches zero. It can be concluded that the correlation between two points as a function of δ x = (xi - xj) , where xi and xj are two points on the graph. Correlation is maximum when $\delta x = 0$ and decreases as δx increases.



This correlation function for two points is generally a Gaussian. For a correlation of a set of points , we can use a kernel function. This kernel function can be modelled using a multivariate Gaussian function.

Time series data represents data points at various instances of time. We assume a relation between time and the quantity that is measured , for example, photon flux. In Bayesian modelling ,a linear regression model is used, y = m x + c , where x represents the time , y is the measured quantity , m - slope and c - the noise. If we rewrite the equation, c = y - m x. Since the noise is normally distributed(, y - m x can be modelled using a Gaussian. Moreover by choosing various Gaussian kernels , we can represent the relation between various points on the graph.

Q. Can any light curve be modelled with a multi-variate Gaussian ?

GPs are commonly used to model stellar variability in photometric time series.

The process is generally used for modelling light curves of the following phenomena:

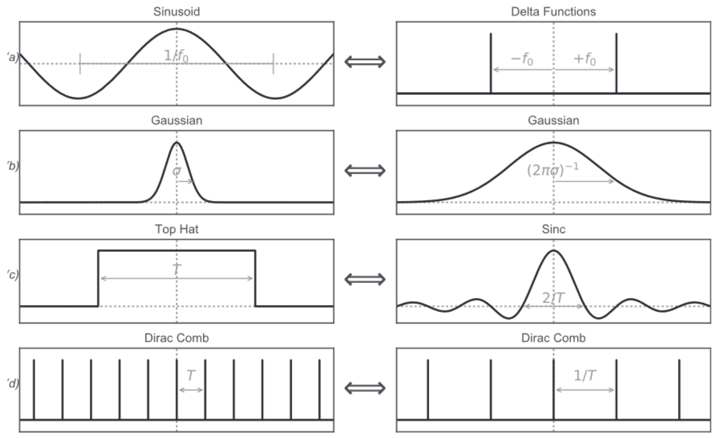
1. Quasi-periodic oscillation of stellar light curves
2. Light curves of transitioning exoplanets.

Q. Sometimes Fourier series is used to model light curves. How is it different from time series ?

Fourier transform converts a function into a form that describe frequencies present in the original function. For a function f(x) , the Fourier transform is given by

$$F(\xi) = \int_{-\infty}^{\infty} f(x) \exp{(2\pi\xi i)} dx$$

,where ξ is the frequency. The output is a complex function of the frequency. According to Fourier transform of some functions are as given below :

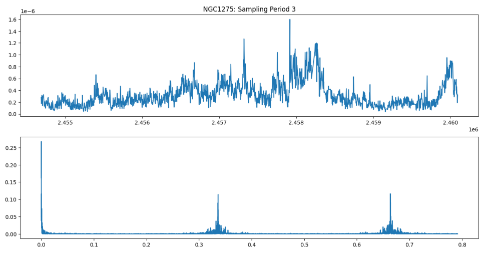


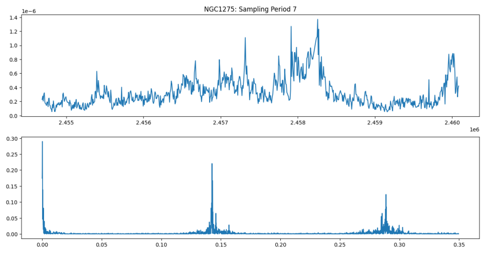
VanderPlas (2018)

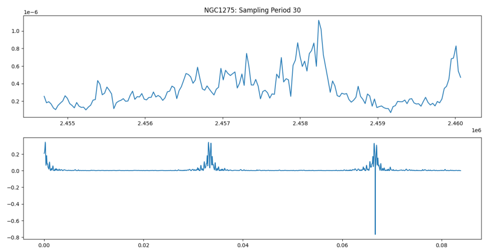
We can get rid of the complex value by taking the square of modulus of the transform.

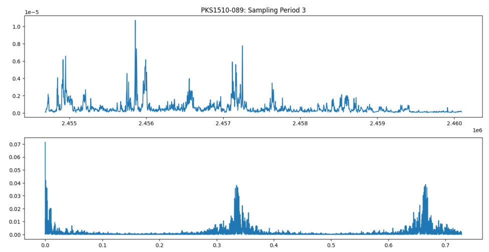
$$P_g = |F(\xi)|^2$$

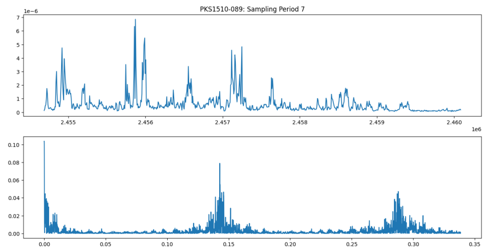
This gives us the Power Spectrum. Peaks in the Power Spectrum corresponds to frequencies which contributes the most to the signal. This provides a way to identify the dominant frequency in any given signal. Lomb Scargle Periodogram is an efficient way to compute the power spectrum of a signal. The Lomb Scargle Periodogram of two blazars were computed using the Astropy package :

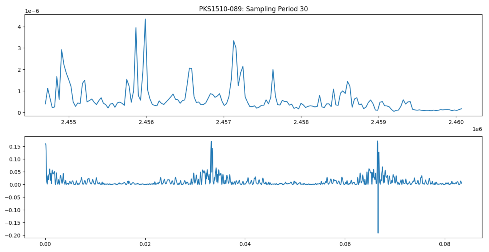










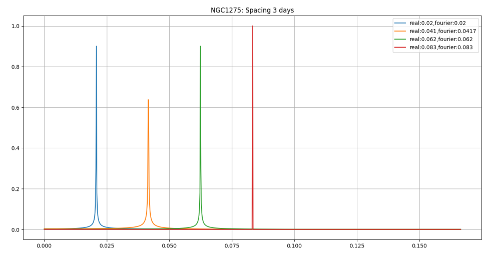


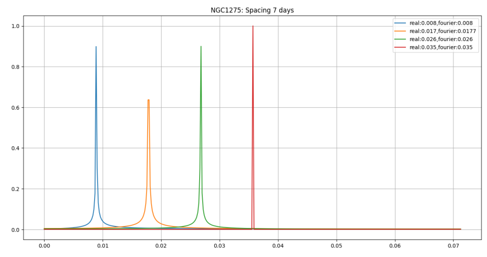
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from astropy.timeseries import LombScargle
import matplotlib.pyplot as plt

#File Location
path = {'NGC1275':{3 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
    ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\NGC 1275\4FGL_J0319.8+4130
    ↪ _daily_3_6_2023.csv"
                 , 7 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                     ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\NGC 1275\4FGL_J0319
                     ↪ .8+4130_weekly_31_5_2023.csv"
                 , 30 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                     ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\NGC 1275\4FGL_J0319
                     ↪ .8+4130_monthly_3_6_2023.csv"}
       ,'PKS1510-089':{3 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
           ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\PKS 1510-089\4FGL_J1512.8-0906
           ↪ _daily_3_6_2023.csv"
                 , 7 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                     ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\PKS 1510-089\4
                     ↪ FGL_J1512.8-0906_weekly_31_5_2023.csv"
                 , 30 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                     ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\PKS 1510-089\4
                     ↪ FGL_J1512.8-0906_monthly_3_6_2023.csv"}}

#Read CSV
def csv_reader(path):
    df = {}
    for i in path:
        cache = {}
        for j in path[i]:
            cache.update({j : pd.read_csv(path[i][j])})
        df.update({i : cache})
    return df
```

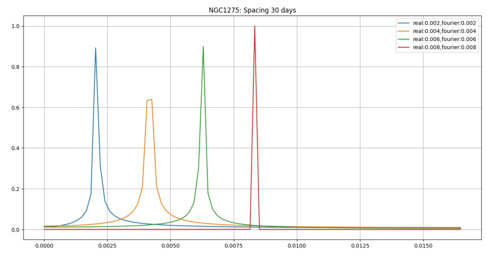
```
df = csv_reader(path)

#Removing Unwanted Data Points
def DataExt(df):
    Data = {}
    for i in df:
        cache = {}
        for j in df[i]:
            info = []
            for n,m in zip(df[i][j]['Julian Date'],df[i][j]['Photon Flux [0.1-100 GeV
                ↪ ](photons cm-2 s-1)']):
                try :
                    info.append([n,float(m)])
                except :
                    pass
            cache.update({j : info})
        Data.update({i : cache})
    return Data

Data = DataExt(df)

#Calculating frequency and power using astropy
def FPExt(Data):
    FP = {}
    for i in Data:
        cache = {}
        for j in Data[i]:
            frequency, power = LombScargle(np.array(Data[i][j])[:,0], np.array(Data[i
                ↪ ][j])[:,1]).autopower()
            cache[j] = {'Frequency' : frequency , 'Power' : power}
        FP[i] = cache
    return FP

FP = FPExt(Data)

for i in Data:
    for j in Data[i]:
        fig, (ax1,ax2) = plt.subplots(2)
        ax1.plot(np.array(Data[i][j])[:,0],np.array(Data[i][j])[:,1])
        ax2.plot(FP[i][j]['Frequency'],FP[i][j]['Power'])
        fig.add_subplot(111, frameon=False)
        plt.tick_params(labelcolor='none', top=False, bottom=False, left=False, right
            ↪ =False)
        plt.grid(False)
        plt.title(f'{i}: Sampling Period {j}')
        plt.show()
        plt.show()
```

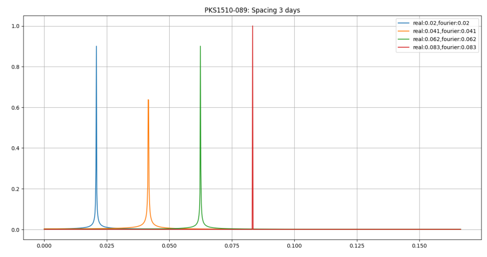
Sometimes, Fourier techniques are used for detecting and characterizing periodic signals in unevenly sampled data using Fourier techniques. For example, Lomb-Scargle periodogram involves calculating Fourier power spectrum to identify the periodicity in the data. In time series analysis, the data is directly fitted with a machine learning model. The time series data that is used to obtain the periodogram is actually the true signal mixed with a window function. In our case , especially astronomical data, the window function is a delta function. If we increase the time between observations, decreasing the spacing of the frequency comb, the true transform no longer "fits" inside the window transform.The result is a mixing of different portions of the signal, such that the true Fourier transform cannot be recovered from the transform of the observed data! This implies that if we have a regularly sampled function with a sampling rate of f0 = 1/T, we can only fully recover the frequency information if the signal is band limited between frequencies ±f0/2.VanderPlas (2018) Another disadvantage of using Fourier technique is that we cannot obtain normally distributed uncertainties like that in time series. Hence it is not possible to use Gaussians to model data after doing Fourier transform.

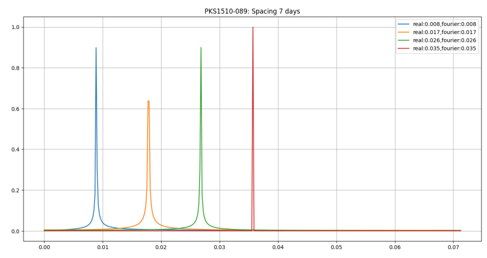
Sine curves were plotted with x values equal to the maximum and minimum of the data taken for the above graphs and equally spaced with time period T .The frequency of sine curves were taken equal to 2/T,4/T,8/T and 10/T of the and their PSD were computed:

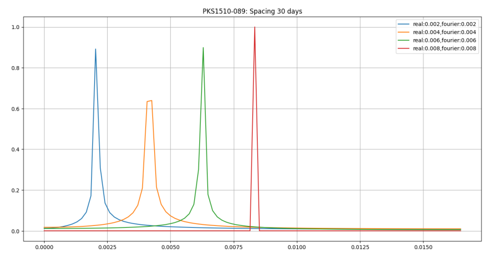










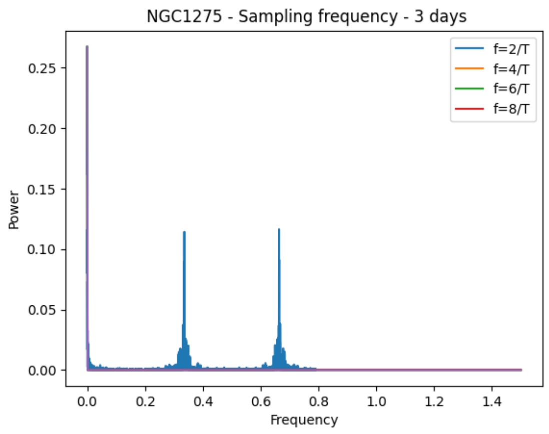


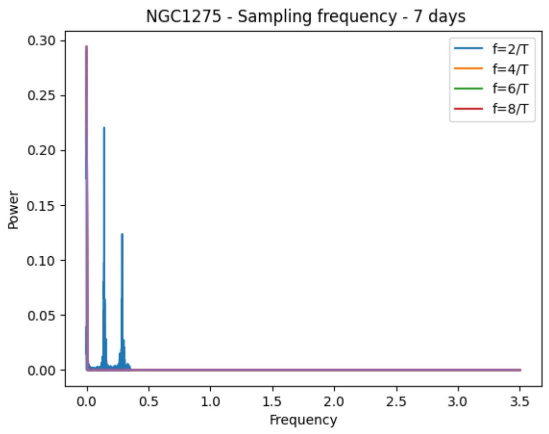
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from astropy.timeseries import LombScargle
import matplotlib.pyplot as plt
```

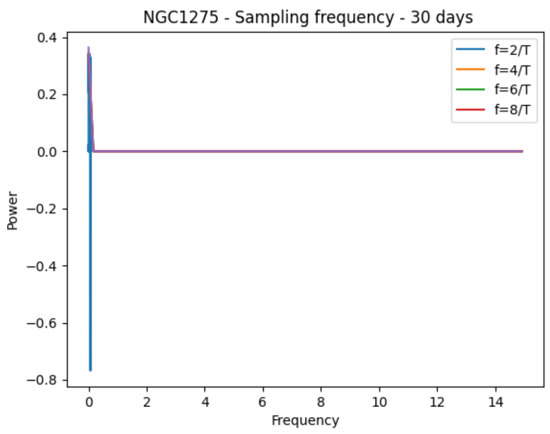
```
from scipy.fft import fft, rfft
from scipy.fft import fftfreq, rfftfreq
from scipy import signal
from peakdetect import peakdetect

#File Location
path = {'NGC1275':{3 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
    ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\NGC 1275\4FGL_J0319.8+4130
    ↪ _daily_3_6_2023.csv"
                , 7 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                    ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\NGC 1275\4FGL_J0319
                    ↪ .8+4130_weekly_31_5_2023.csv"
                , 30 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                    ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\NGC 1275\4FGL_J0319
                    ↪ .8+4130_monthly_3_6_2023.csv"}
      ,'PKS1510-089':{3 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
          ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\PKS 1510-089\4FGL_J1512.8-0906
          ↪ _daily_3_6_2023.csv"
                , 7 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                    ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\PKS 1510-089\4
                    ↪ FGL_J1512.8-0906_weekly_31_5_2023.csv"
                , 30 : r"C:\Users\abhay\OneDrive\Desktop\Everything\Academic\
                    ↪ Intern_Sem7\Astro_Gaussian_analysis\Data\PKS 1510-089\4
                    ↪ FGL_J1512.8-0906_monthly_3_6_2023.csv"}}

#Read CSV
def csv_reader(path):
    df = {}
    for i in path:
        cache = {}
        for j in path[i]:
            cache.update({j : pd.read_csv(path[i][j])})
        df.update({i : cache})
    return df

df = csv_reader(path)

#Removing Unwanted Data Points
def DataExt(df):
    Data = {}
    for i in df:
        cache = {}
        for j in df[i]:
            info = []
            for n,m in zip(df[i][j]['Julian Date'],df[i][j]['Photon Flux [0.1-100 GeV
                ↪ ](photons cm-2 s-1)']):
                try :
                    info.append([n,float(m)])
                except :
                    pass
            cache.update({j : info})
        Data.update({i : cache})
    return Data

Data = DataExt(df)
```

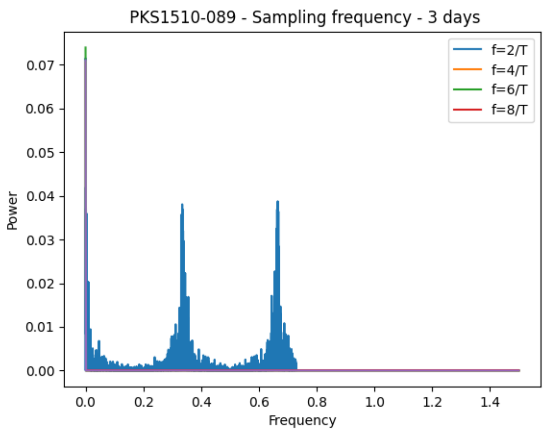
```
l = {'NGC1275':{3:{1:0.0417},7:{1:0.0177},30:{1:0.004}},
     'PKS1510-089':{3:{1:0.041},7:{1:0.017},30:{1:0.004}}}

for i in Data:
    for j in Data[i]:
        tstep = j
        sampling_freq = 1/tstep
        x = np.arange(np.array(Data[i][j])[0,0],np.array(Data[i][j])[-1,0]+1,j)
        N = len(x)

        Nyquist_freq = sampling_freq/2
        f0 = Nyquist_freq/8
        label = []
        for k in range(0,4):
            y = np.sin(2 * np.pi * (k+1)*f0 * x)

            fstep = sampling_freq / N
            f = np.linspace(0,(N-1)*fstep,N)
            fourier = np.fft.fft(y)
            fourier_mag = np.abs(fourier) / N
            fplot = f[0:int(N/2)]
            fmag = 2 * fourier_mag[0:int(N/2)]

            plt.plot(fplot,fmag)
            peaks = signal.find_peaks(fmag,threshold=0.1)
            freq_peaks = fplot[peaks[0]]
            print(freq_peaks)
            for m in freq_peaks:
                label.append(f'real:{int((k+1)*f0*1000)/1000},fourier:{int(m*1000)
                    ↪ /1000}')
        label.insert(1,f'real:{int((1+1)*f0*1000)/1000},fourier:{l[i][j][1]}')
        plt.title(f'{i}: Spacing {j} days')
        plt.legend(label,loc='upper right')
        plt.grid()
        plt.show()
```

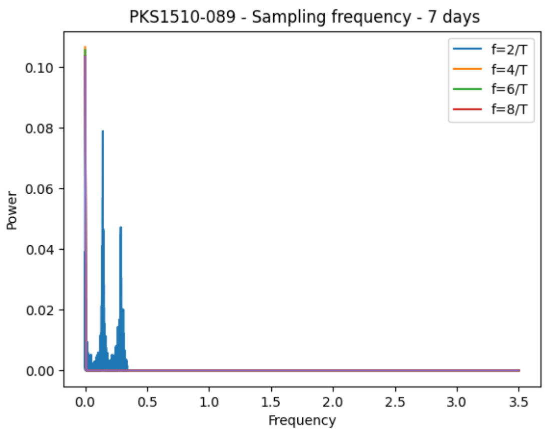
Resizing the sine curves and plotting it alongside the Lomb-Scargle periodogram we obtained earlier:

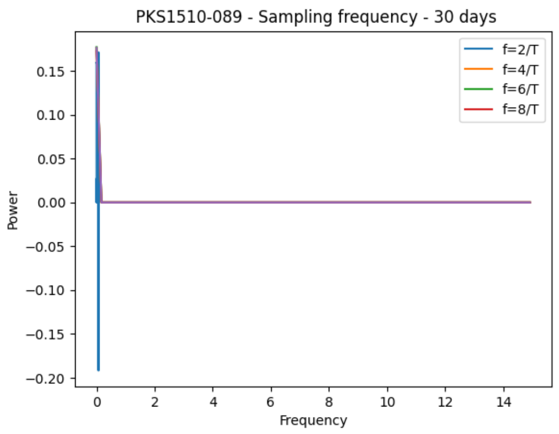












7 Kernels

In general, kernel is a function k which maps a pair of inputs x ϵ X , x' ϵ X into R. Kernel functions in Gaussian modelling ,also known as covariance function , provides information about the variation of one random variable with respect to the other with temporal/spacial variation.Williams and Rasmussen (2006) Kernel functions drive the degree of smoothness of the observed light curves, and can also identify periodic behaviors and define an important connection between autoregressive time series methods and GP analysis.Williams and Rasmussen (2006); Durrande et al. (2016); Foreman-Mackey et al. (2013)

Some of the important Gaussian kernels are as follows:

1. Sqaured Exponential/Radial Basis Function(RBF):

$$A \exp{(-\frac{r^2}{2L^2})} \tag{3}$$

   A>0 is the amplitude, r is the time separation and L is the length scale of exponential decay.If L is large , correlation between two data points will be stronger.Covino et al. (2020)

2. Rational Quadratic(RQ)

$$A[1+(\frac{r^2}{2\alpha L^2})^{-}\alpha] \tag{4}$$

This kernel can be seen as a scale mixture or sum of SE covariance functions with different characteristic length scales drawn from a gamma distribution.Covino et al. (2020)

3. Exponential Sine Squared(ESS)

$$A \exp\left[-\tau^2 sin(-\frac{\pi r}{P})^2\right] \tag{5}$$

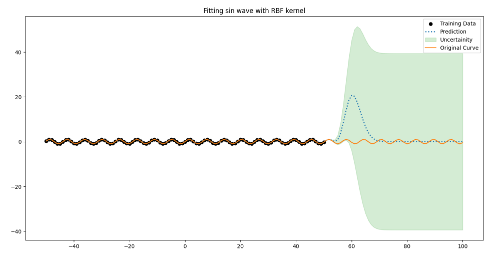
There is an additional parameter τ. If it is large, points separated by a period are strongly correlated, while the correlation is looser if τ becomes small.Covino et al. (2020)

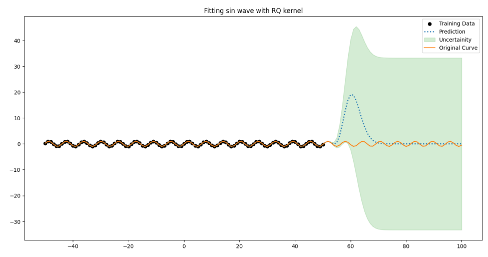
4. Cosine Kernel(CS)

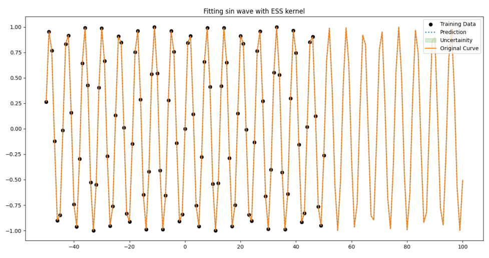
$$Acos(\frac{2\pi r^2}{P}) \tag{6}$$

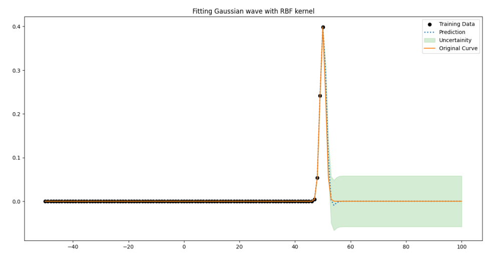
A>0 is the amplitude and r is the separation between data points. P is the period of the data. Covino et al. (2020)

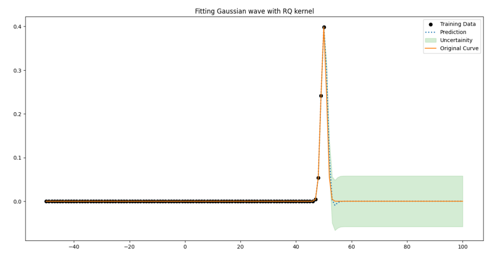
Let's use some of the kernels to model different types of signals:

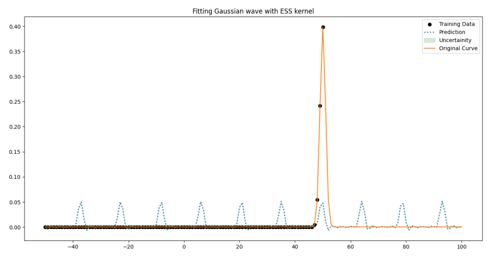


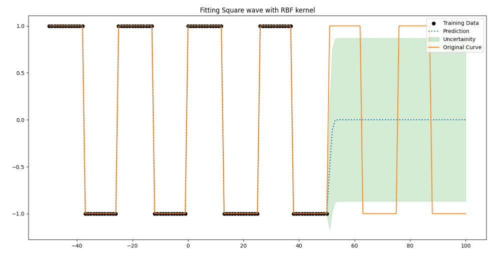


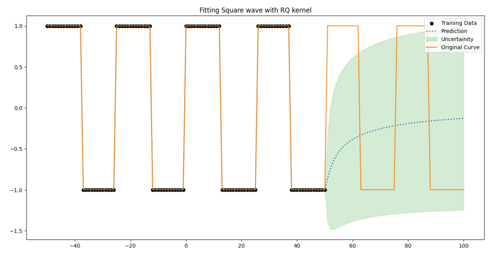


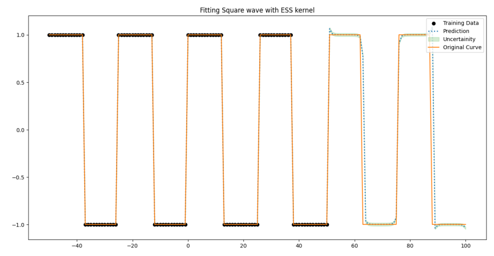












```
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process.kernels import RationalQuadratic
from sklearn.gaussian_process.kernels import ExpSineSquared
from scipy.stats import norm
from scipy import signal

x = np.arange(-50,51,1)
signals = {'sin':np.sin(x), 'Gaussian':norm.pdf(x,50,1), 'Square':signal.square(x/4)
    ↪ }
y = np.arange(-50,101,1)
original = {'sin':np.sin(y), 'Gaussian':norm.pdf(y,50,1), 'Square':signal.square(y
    ↪ /4)}

kernels = {'RBF' :1 * RBF(length_scale=1.0, length_scale_bounds=(-1e2, 1e2)),'RQ': 1
    ↪  * RationalQuadratic(length_scale=1.0, alpha=1.5)
         , 'ESS':1 * ExpSineSquared(length_scale=7, periodicity=1)}

for j in signals:
    for i in kernels:
        gaussian_process = GaussianProcessRegressor(kernel=kernels[i])
        gaussian_process.fit(x.reshape(-1,1), signals[j].reshape(-1,1))

        start_time = time.time()
        mean_predictions_gpr, std_predictions_gpr = gaussian_process.predict(
            y.reshape(-1,1),
            return_std=True,
        )

        plt.scatter(
            x,
            signals[j],
            color="black",
            label="Training Data Points",
        )

        # Plot the predictions of the gaussian process regressor
        plt.plot(
            y,
            mean_predictions_gpr,
            label="Gaussian process regressor",
            linewidth=2,
            linestyle="dotted",
        )

        plt.fill_between(
            y.ravel(),
            mean_predictions_gpr - std_predictions_gpr,
            mean_predictions_gpr + std_predictions_gpr,
            color="tab:green",
            alpha=0.2,
        )

        plt.plot(y,original[j])

        plt.title(f'Fitting {j} wave with {i} kernel')

        plt.legend(['Training Data','Prediction','Uncertainity','Original Curve'],loc
            ↪ ='upper right')

        plt.show()
```

Observations :

1. RBF - Increasing the length scale led to higher gaussian during extrapolation.
2. RQ - High alpha and low length scale led to high uncertainities. If both of the arameters are low, uncertainitites are low.
3. ESS - Kernel is sensitive to periodicity , if high periodicity is given , the predicted curve is smaller.

We can see that ESS kernel is best suited to model periodic signals like sine waves and square waves

8 Periodogram

Periodograms, which are used in signal processing and spectral analysis, often exhibit peaks or spikes at certain frequencies. These peaks represent the presence of significant components or harmonics in the analyzed signal. There are several reasons why periodograms may have peaks:

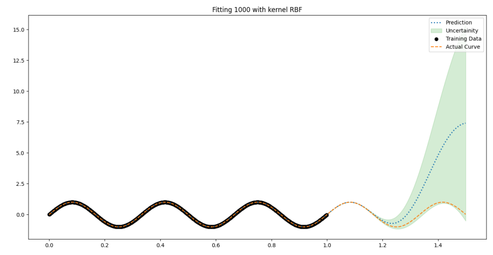
1. Presence of periodic or quasi-periodic components, such as sine waves, the periodogram will exhibit peaks at frequencies that correspond to that particular frequency. This helps in identification of dominant periods in time series data. The peaks depict the power or intensity of signal with that particular frequency in the given signal.
2. According to VanderPlas (2018) ,noise and interference can introduce peaks in the signal. Random fluctuations in the signal can cause peaks in the periodogram, which do not correspond to any meaningful components but rather represent noise energy.
3. In some cases, the signal may consist of discrete spectral lines at specific frequencies. These lines can arise from various sources, such as the presence of specific tones or frequencies in the original signal. The periodogram will exhibit peaks at these frequencies, indicating the existence of these spectral lines.

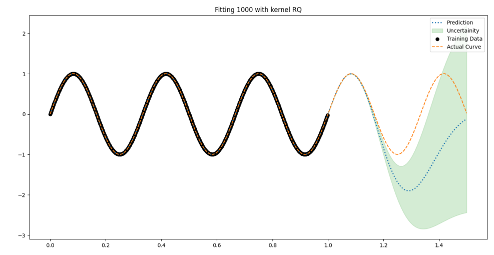
4. Sometimes, the sampling rate might be much less than the frequency components of the signal. This is known as aliasing. In such cases, the frequency components in the signal can ”fold back” or alias into the frequency range covered by the sampling rate. These aliased components can result in peaks in the periodogram at frequencies that differ from the original signal but are related due to aliasing.

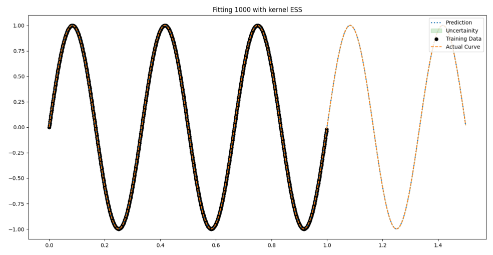
5. Application of windowing functions, especially to reduce spectral leakage. This leads to introduction of side lobes or sidelobes in the periodogram, which appear as additional smaller peaks around the main peaks.VanderPlas (2018)

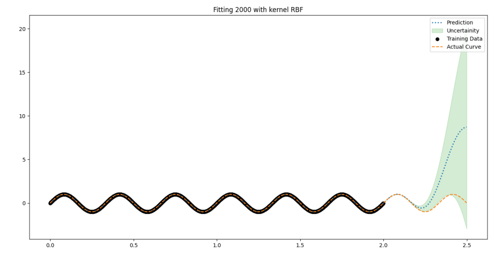
9 Gaussian Modelling

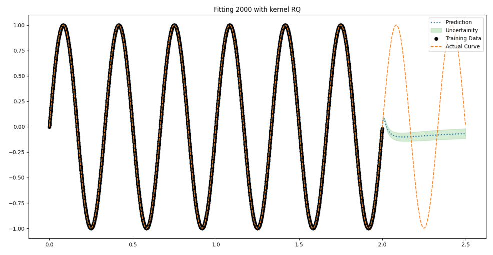
The given sine curves were were modelled using Bayesian regression:

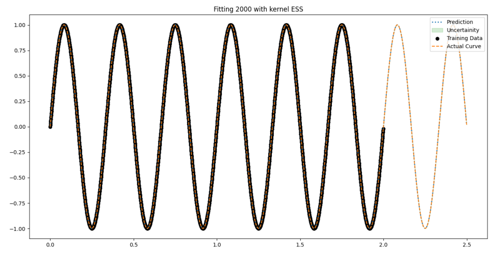












```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process.kernels import RationalQuadratic
from sklearn.gaussian_process.kernels import ExpSineSquared

sampling_freq = 1000
tstep = 1/sampling_freq
N = [1000,2000]

t={}
for i in N:
    t[i] = np.linspace(0,(i-1)*tstep,i)

signal= {}
for i in t:
    signal[i]= np.sin(2 * np.pi * 3 * t[i])

pred = {}
b = {}
for i in N:
    pred[i] = np.linspace(0,(i-1 + 500)*tstep,i)
    b[i] = np.sin(2 * np.pi * 3 * pred[i])

kernels = {'RBF':1 * RBF(length_scale=1.0, length_scale_bounds=(-1e2, 1e2)),
           'RQ':1 * RationalQuadratic(length_scale=1.0, alpha=1),
           'ESS':1 * ExpSineSquared(length_scale=1, periodicity=1)}

for n in t:
    for i in kernels:
        gaussian_process = GaussianProcessRegressor(kernel=kernels[i])
        gaussian_process.fit(t[n].reshape(-1, 1), signal[n].reshape(-1, 1))

        mean_prediction, std_prediction = gaussian_process.predict(pred[n].reshape
            ↪ (-1,1), return_std=True)

        # Plot the predictions of the gaussian process regressor
        plt.plot(
            pred[n],
            mean_prediction,
            label="Gaussian process regressor",
            linewidth=2,
            linestyle="dotted",
        )

        plt.fill_between(
            pred[n].ravel(),
            mean_prediction - std_prediction,
            mean_prediction + std_prediction,
            color="tab:green",
            alpha=0.2,
        )

        plt.scatter(
            t[n],
            signal[n],
            color="black",
        )

        plt.plot(pred[n],b[n],linestyle='dashed')

        plt.title(f'Fitting {n} with kernel {i}')

        plt.legend(['Prediction','Uncertainity','Training Data','Actual Curve'],loc='
            ↪ upper right')

        plt.show()
```

Observation:

1. ESS proves to be more useful while modelling sine curves.
2. If the number of points is less and the value of length factor is low, RBF kernel fits each point as a separate Gaussian.

References

Covino, S., Landoni, M., Sandrinelli, A., and Treves, A. (2020). Looking at blazar light-curve periodicities with gaussian processes. *The Astrophysical Journal*, 895(2):122.

Do, C. B. and Lee, H. (2007). Gaussian processes. *Stanford University, Stanford, CA, accessed Dec*, 5:2017.

Durrande, N., Hensman, J., Rattray, M., and Lawrence, N. D. (2016). Detecting periodicities with gaussian processes. *PeerJ Computer Science*, 2:e50.

Foreman-Mackey, D., Hogg, D. W., Lang, D., and Goodman, J. (2013). emcee: the mcmc hammer. *Publications of the Astronomical Society of the Pacific*, 125(925):306.

VanderPlas, J. T. (2018). Understanding the lomb–scargle periodogram. *The Astrophysical Journal Supplement Series*, 236(1):16.

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.