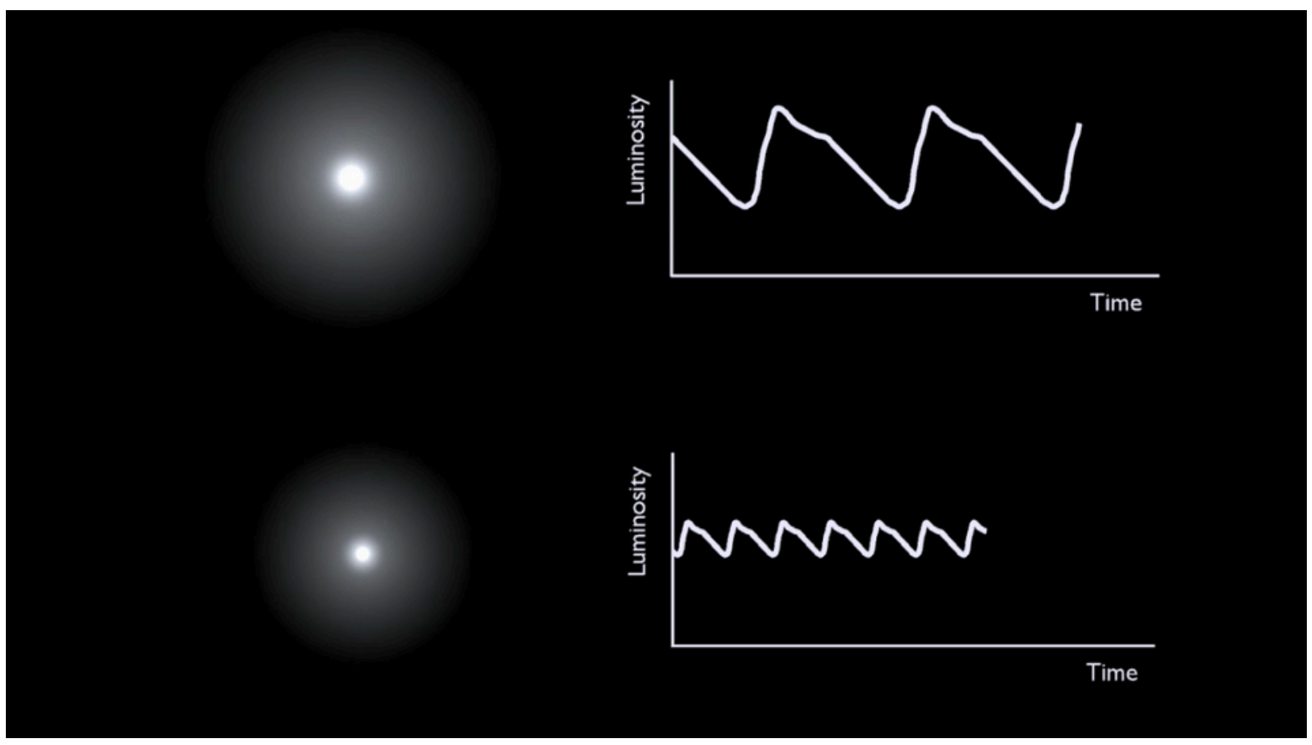
Monica Isabella Cue

NEW METHOD TO HELP FINDING "CANDLE" STARS USING SINUSOIDAL ML REGRESSION.

BY MAKING USE OF THE FOURIER TRANSFORM AND THE KEPLER SPACE TELESCOPE DATASET.

INTRODUCTION

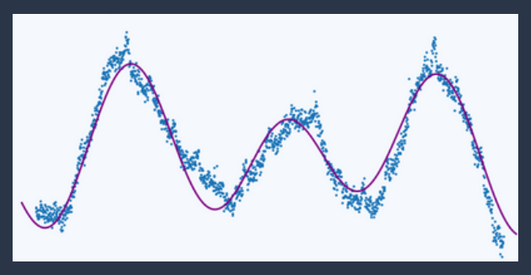
How do we measure stars distance? The astronomers have discovered ingenious methods to do it. The main two are parallax and the use of standard candles, that is events or objects that share common characteristics that allow to know their brightness. One type of candles are known as cepheids, which are **stars that shine in a periodic manner** in just days or less as it's showed below. In contrast to the rest of stars that change brightness only due to "random" fluctuations, or the star aging which delay millions of years. In this case we will look for them using Kepler's data.



GENERAL METHODOLOGY.

To find such cepheids, it was developed a new method **making use of Machine Learning** and the **Fast Discrete Fourier Transform**. Where in conjunction throws the error when trying to **fit a sinusoidal model** to the dataset. Method which is a complementary tool to the traditional periodogram used in the last decades.

RESULTS OVERVIEW.



The results of the new method are showed above. Also makes easier to get a significant loss when compared to the traditional method as showed below. Finally it has the advantage to fit skewsines. *Note: The second metric is a modified version of root mean squared loss.

35.7%

OF IMPROVEMENT IN ABSOLUTE ERROR LOSS

90.4%

OF IMPROVEMENT IN ROOT MEAN SQUARED LOSS*

AT EXPENSE OF A DECREMENT IN VELOCITY

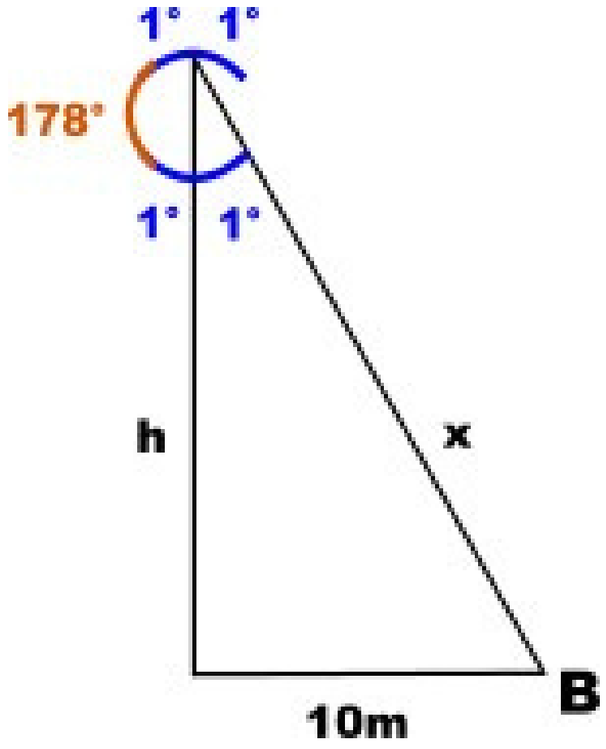
ASTRONOMY PRELIMINARIES.

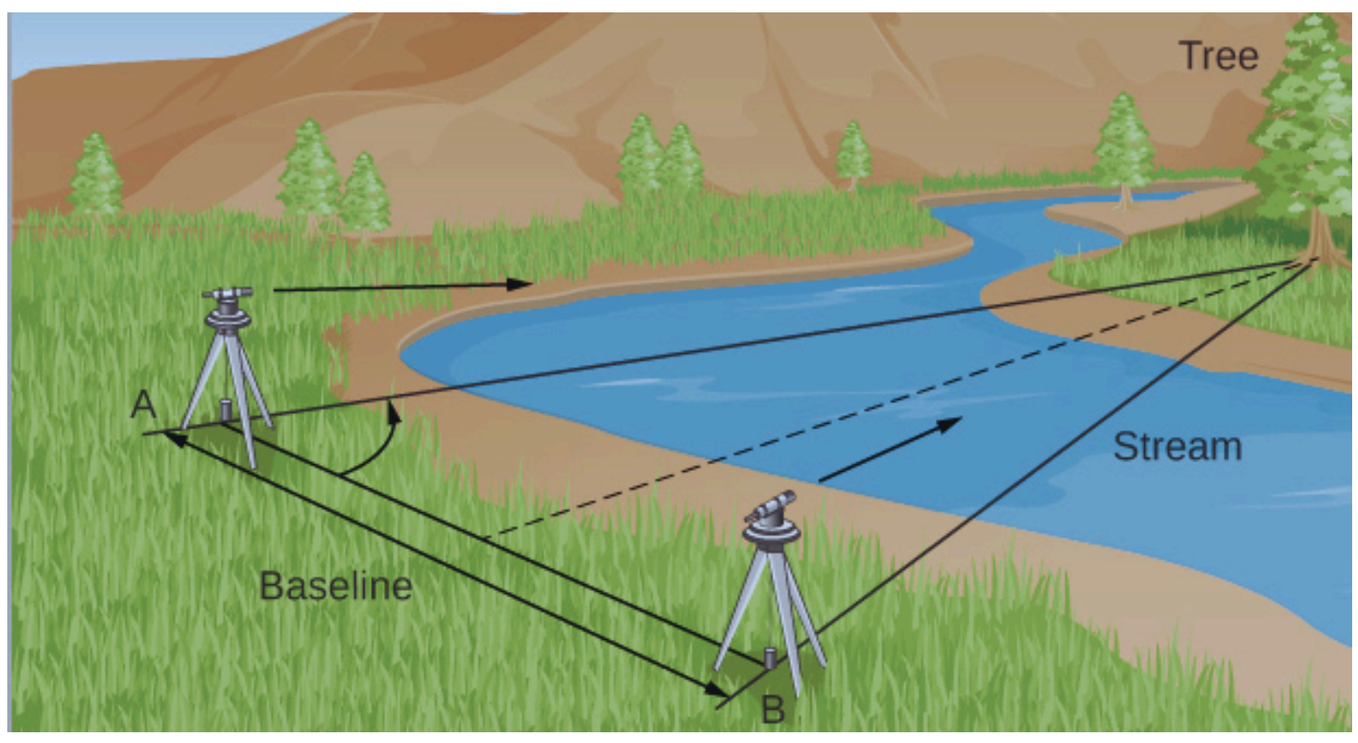
01 MEASURE STAR DISTANCES BY PARALLAX.

There are two main ways to do so. The first is using parallax. The basics are the following: Suppose you want to find the distance of a far away object like a tree. You may think that it will easy by just measure how big or small seems like. This reasoning is correct whenever you know the aproximate size of the tree. Maybe you have already identified that is an oak and not other specie. However how could you know when you don't have such information? Well the answer comes from trigonometry.

The key is to see the tree from two points A and B and not just one which we actually one their distance known as **baseline**, as it's showed below. Now the direction to the tree C in relation to the baseline is observed from each station. Note that C appears in different directions from the two stations. This apparent change in direction of the remote object due to a change in vantage point of the observer is called **parallax**. Just to clarify what it means below is an example.

Example: Suppose that you want to measure the distance of a tree T, from to observers A and B such that the length of the baseline AB is 20 meters. Recall the mountain seems static compared to the tree we can used it as a reference point to establish a radios of angles. In this case we measure that the angle of displacement is 2°. What is the distance to the tree?





By symmetry we can deduce that the measures of half of the triangle are the ones above. From there notice that:

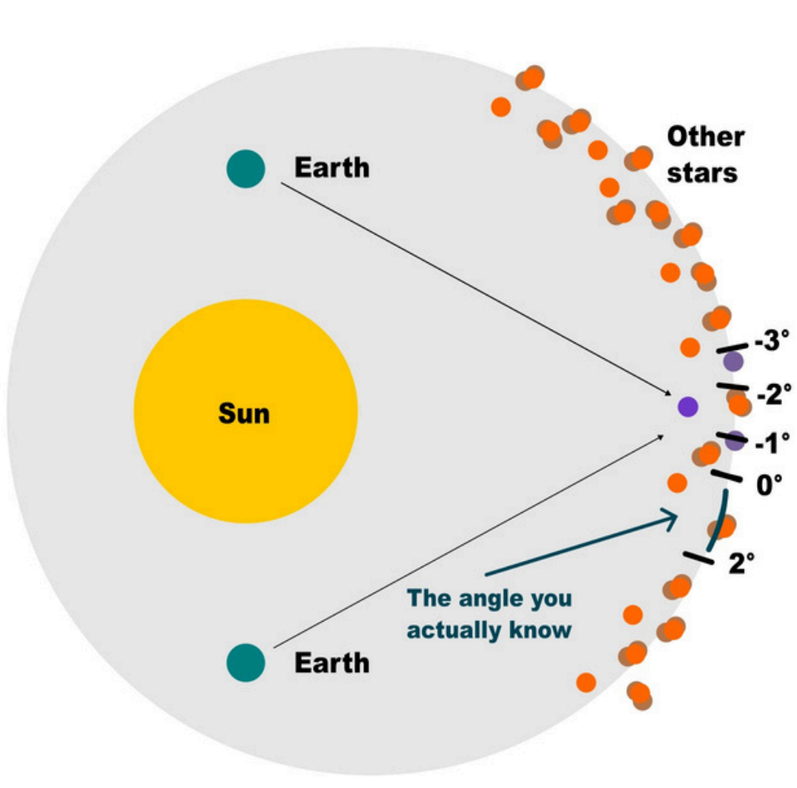
$$\frac{10}{sin(1^\circ)} = \frac{h}{sin(89^\circ)}$$

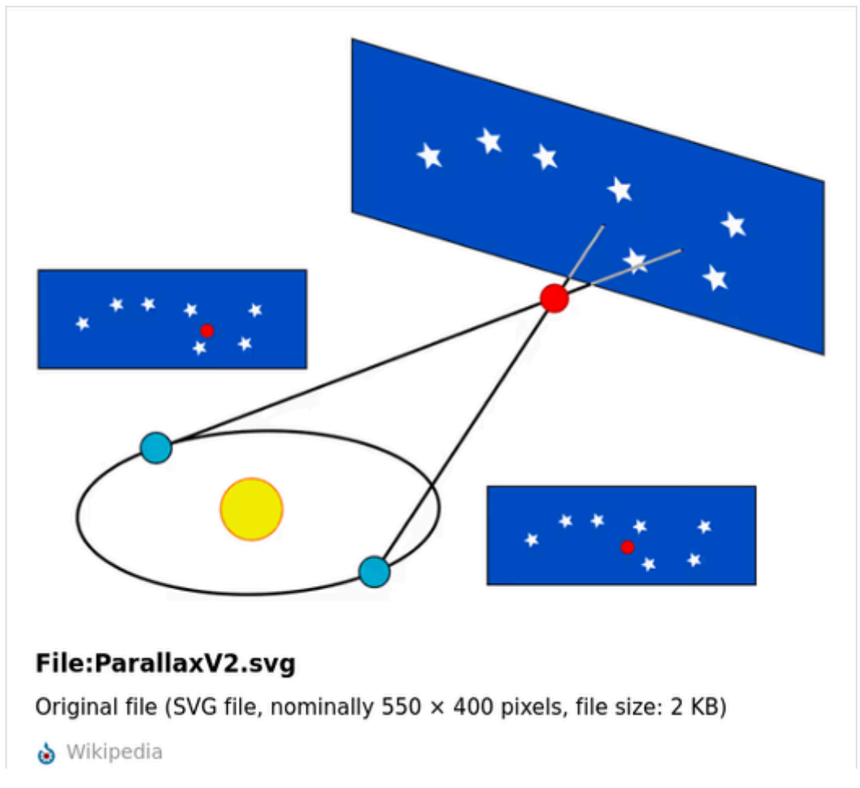
$$x = \sqrt{10^2 + h^2} = 572.08m$$

Thus the distance h is 572m from the first equation. The distance x will be 572.08 mtrs.

Unfortunately, nearly all astronomical objects are very far away. To measure their distances requires a very large baseline and highly precise angular measurements. And moreover in order to the parallax be measure we need a very big baseline. Such a baseline is provided by Earth's annual trip around the Sun. That is **a baseline of 300,000 millions of kilometers!**

However and contrary to the intuition, **this baseline is still to small to the farthest stars!** So we need another method to measure such distances, maybe we need to reconsider the first idea.

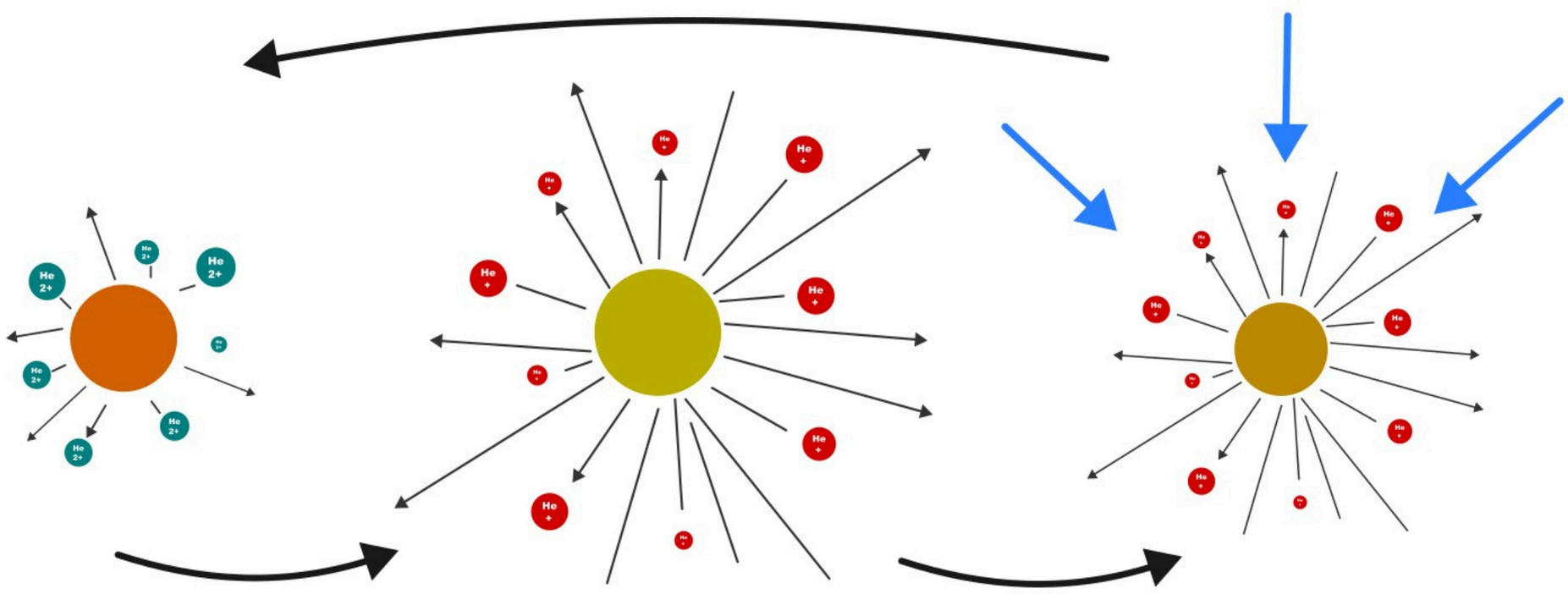




Pictures. *In this case the triangulation make use of the celestial sphere, that is how the target (purple dot) star moves relative to the rest (orange dots). The second picture shows the observation from Earth.*

02 MEASURE STAR DISTANCES BY STANDARD CANDLES.

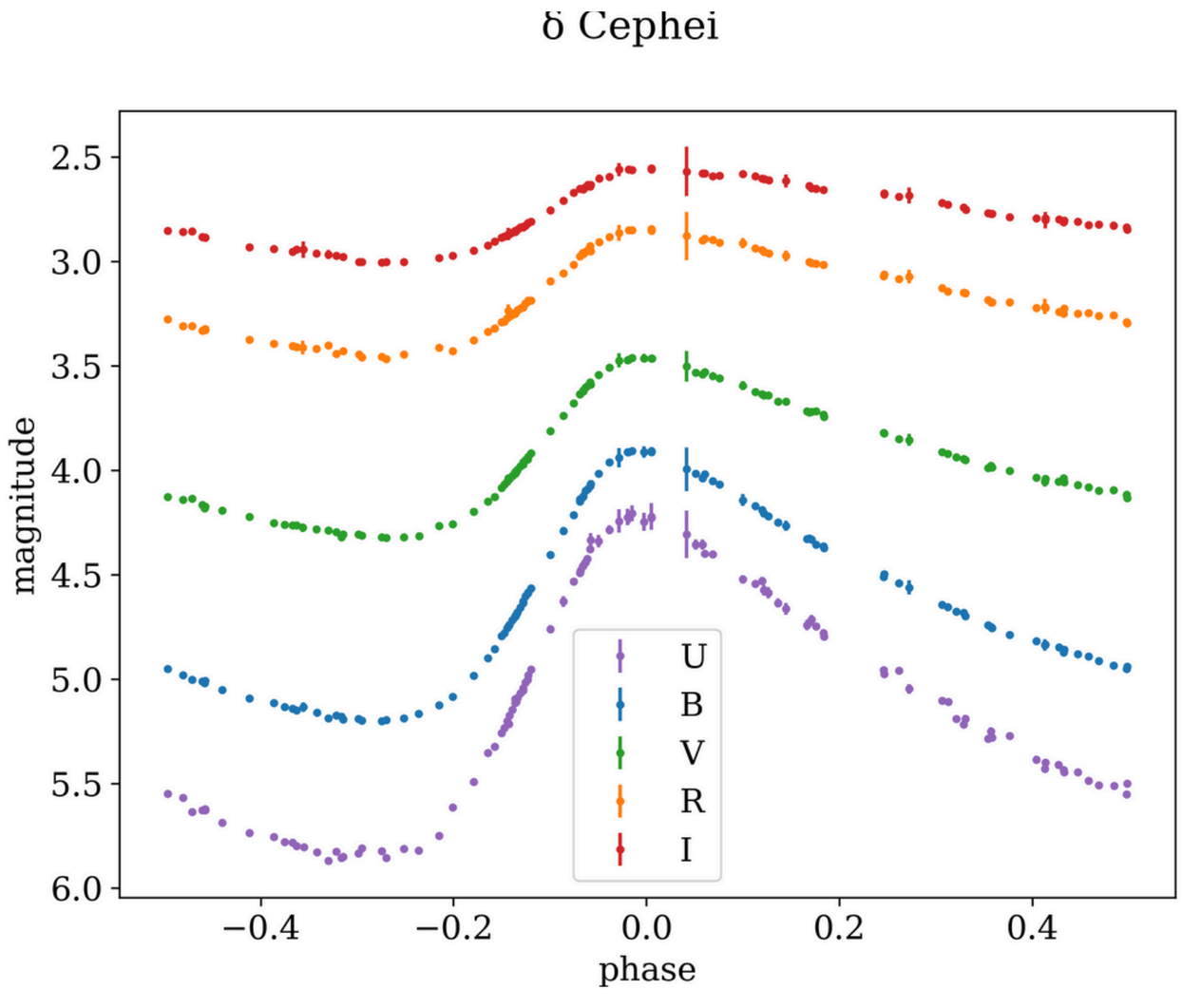
Returning to the first idea as it will be convenient to know the specie of the tree we are seeing at by making use of the fact that we know the mean size of such trees; **if we know the type of the star and such star and such type always emits the same brightness, then we could know their distance precisely!**



Picture. *Helium 2+ is denoted by turquoise color and Helium + by red dots. We can see that when the star is in the first stage Helium 2+ blocks a lot of light contrary to when it expands. Finally in the third stage the gravity force make the star contracts again, in this case is represented by the last blue arrows.*

The reason is the physical properties of it. Summarizing a lot, is due to the change of the emission of ionized Helium which varies from Helium + to Helium 2+. That is from loosing 1 electron to loosing two, because the star expands due to the nuclear fusion, and hence kick out more electrons from Helium. However although the star is expanding Helium 2+ is more opaque leading less photons to escape and the result of this is that the star appears less bright. However when the star is too big, its loose energy so the star again emits Helium + instead of 2+ making it more bright. However when the star is too big it contract due to gravity making it more compact. And when is too compact emits more energy producing Helium 2+ again and the cycle start again.

How we measure the rest of stars that are not cepheids is more difficult and less precise. However we can make some assumptions. By looking at the star around it, because all its neighborhood (cumulus) have approximately the same distance, then we could approximately measure the distance of the stars near to it. (That is the idea although the details are much more complicated). Also is important to note that cepheids aren't the unique standard candles. The main characteristic of star candles is that they must be objects or events which brightness is proportional to other of its features so that we can directly know its distance. Cepheids are objects, but by events we mean supernovas for example, that is the explosion of a star, although in this article we are not going to look for them.



Now returning to cepheids what is important for us is that they radiate its energy in a periodic way that we can measure. The model that we looks like the picture above.

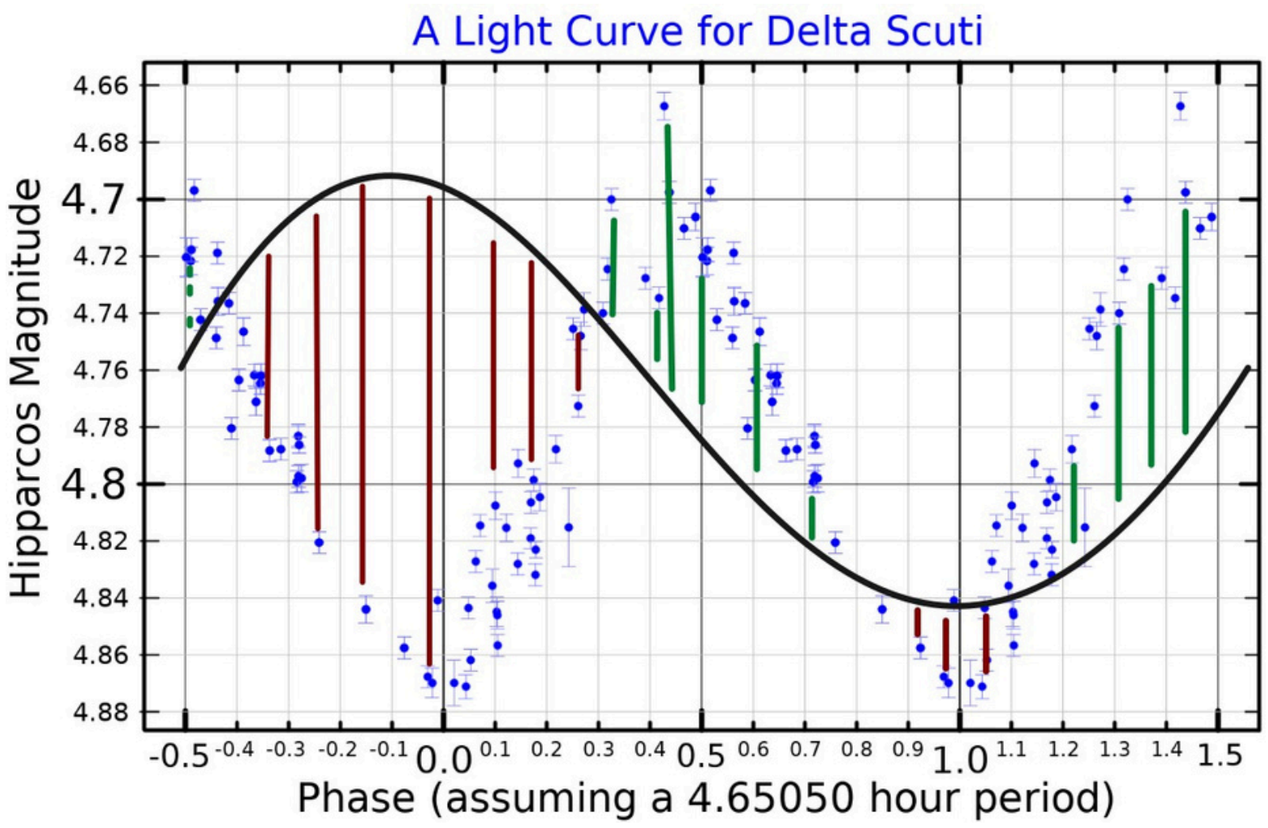
This image show the ideal model for all the types of light going from infrared to ultraviolet, of a Delta Scuti Cepheid. However in our dataset we are only going to use one type of light. So to identify this type of stars we must find which files in our dataset have light curves similar to that.
In other word we have to perform a sinusoidal regression to fit our data. In this case by a function f*(x) and a error denoted by E.

THE MACHINE LEARNING CHALLENGE.

03 LINEAR REGRESSION VS SINUSOIDAL REGRESSION.

In most of Machine Learning books and courses, teach to perform linear regression and some of them even polynomial regression. The reason is that we can easily compute the loss function to then use Gradient Decent or more advanced optimization algorithms to perform the fitting process.

The most basic metrics that any loss function could use are Absolute Mean Loss and Squared Mean Loss and its partners. Is easy to see why this methods works for linear regression however when using sinusoidal regression we could see some problems in the implementation.



In the picture we can see what happen when we try to fit one sinusoidal model to our data. In a real world example where we are trying to fit the most simple of all models, the sine function. But of course, we don't know neither the period or the amplitude or the vertical shift neither the horizontal shift because that what we want to predict!

Suppose I start with a random sine function. In this case it seems that the amplitude A and the frequency Ω as well as the vertical shift V are correct. We just need to find the horizontal shift φ. But when we apply our loss function note that half of the points of the model are below the curve which subtracts loss and half of the points add loss. Suppose the red errors add up to -50 and the green errors +55. Then for some loss measure, let's say the most simple, Mean Absolute Loss, we have:

$$Ef^*(x) = E(Asin(\omega x + \phi) + V) = |-50| + 60 = 110$$

And now we can see that when we change our horizontal shift φ the loss E is still the same!

$$Ef^*(x) = E(Asin(\omega x + \phi) + (V + 0.5)) = |-60| + 50 = 110$$

With a little of thinking we could imagine why this happen. And even worse, you may figure out this happens almost all the parameters. We need a different approach to do the work.

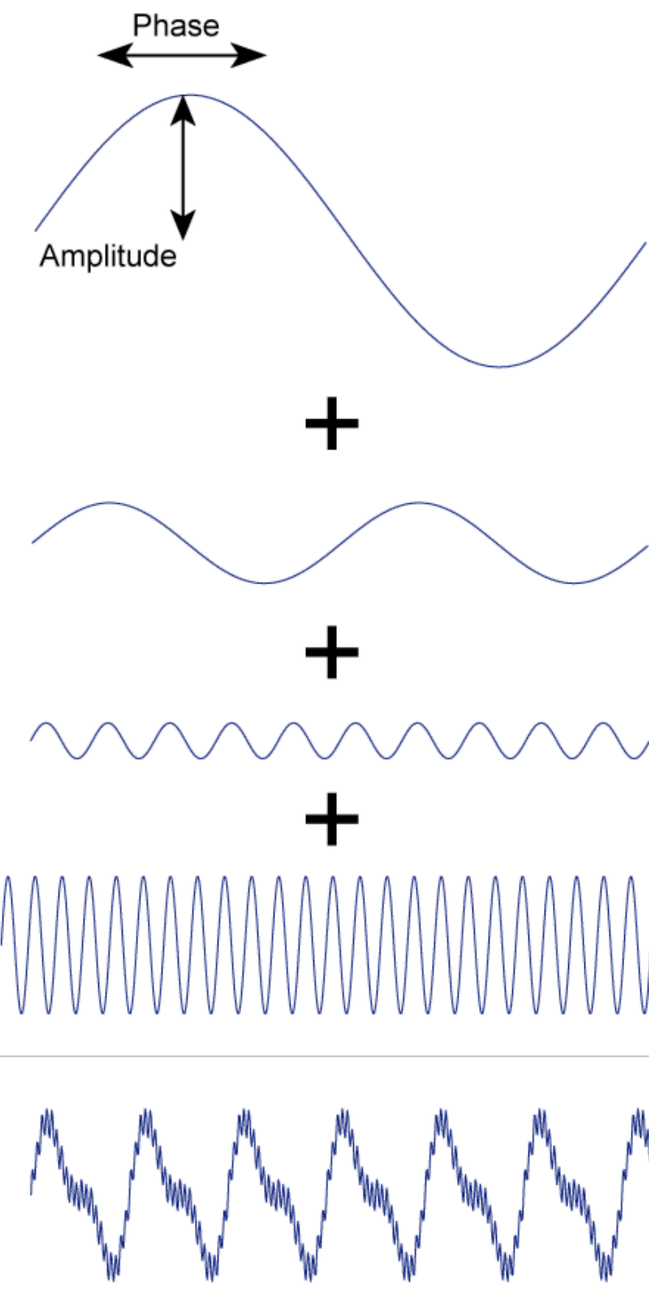
THE FOURIER TRANSFORM.

04 THE FOURIER TRANSFORM.

The **Fourier Transformation*** is a way to decompose some signal into individual sine and cosines functions. In this article we will limit ourselves to the discrete version and the use of just the sines values which mathematically corresponds to the imaginary values that complement the real cosine values. Each sine function will be called a **trigonometric monomial*** and the combination (sum) of those will be called a trigonometric polynomial. Hence the goal of the Fourier Transformation is to convert continuous signals into trigonometric polynomials.

And as mention before **every sine function can be represented or determined by four parameters**, amplitude A, frequency Ω, the horizontal displacement also known as phase φ and finally the center or vertical displacement V. Thus every trigonometric monomial can be store in a 4 tuple.

$$Asin(\omega x + \phi) + V$$



Picture. Representation of sinusoidal addition.

***Note.** The Fourier transform has many different formulas and not just one with different interpretations. In must books contrary to this article a trigonometric monomial of a function f(x) is defined as :

$$e^{n(i2\pi x)} = cos(n2\pi x) + sin(n2\pi x)i$$

Or written with the amplitude and using or notation ω we end up that the ω's monomial of f(x) is:

$$A_\omega e^{\omega(i2\pi x)} = A_\omega cos(\omega 2\pi x) + A_\omega sin(\omega 2\pi x)i$$

We can note a lot differences with or definition of trigonometric monomial we defined before. First note that the terms φ and V doesn't appear . Second is a complex function and not real anymore. And third is a combination of sines and cosines and not just sines.

And the reason of this is that first V is an isolated term of the function, so its a monomial itself which is not of such form. In some books about the Fourier Transform is called the term a/2 or similar.

Second the reason why we express our monomial in terms of sines and cosines is that is not possible to find the phase φ only choosing either only sine functions or cosines sine functions. More information about this in the following video made by the Reducible YouTube channel minute 18:33 about the phase problem.

https://youtu.be/yYEMxqreA10

So we need both functions express in the form above. And third we use complex number because of simplicity in notation as we can write such formula using the Euler notation e. Although we can use the 2-dimensional real plane also.

Summarizing in the code implementation using the **scipy.fft library**, we will make use the complex sines and cosines implementation because the phase φ is unknown for us however because as is explained in the video sine and cosine are orthogonal, after we compute the phase we can use only sines to express our function as the Wikipedia link below shows (in such article the frequency ω is denote as the inverse of the period T, and the terms of the summary use a dummy variable n also instead of ω as we do, also note that V in this case is denoted by D zero):

https://en.wikipedia.org/wiki/Fourier_series

Fourier series, amplitude-phase form

$$s_N(x) = D_0 + \sum_{n=1}^{N} D_n \cos\left(2\pi \frac{n}{P} x - \varphi_n\right) \quad \textbf{(Eq.1)}$$

Fourier series, sine-cosine form

$$s_N(x) = A_0 + \sum_{n=1}^{N} \left(A_n \cos\left(2\pi \frac{n}{P} x\right) + B_n \sin\left(2\pi \frac{n}{P} x\right)\right) \quad \textbf{(Eq.2)}$$

Fourier series, exponential form

$$s_N(x) = \sum_{n=-N}^{N} C_n \, e^{i2\pi \frac{n}{P} x} \quad \textbf{(Eq.3)}$$

05 SCIPY FFT IMPLEMENTATION.

As you may notice there is not a standard notation to express the Fourier Transform or the Discrete version. Some books use the frequency ω other 1 over the period T, other use n or k and many other j instead of i to denote the imaginary unit. In this case scipy uses the following notation in the documentation: https://docs.scipy.org/doc/scipy/tutorial/fft.html

The FFT *y[k]* of length N of the length-N sequence *x[n]* is defined as

$$y[k] = \sum_{n=0}^{N-1} e^{-2\pi j \frac{kn}{N}} x[n] \,,$$

Where y[k] is the trigonometric polynomial we want to find f*(x) in our notation. Because our data-frame have a finite size of N points we just have to sum over a specific finite number of frequencies by the Shannon-Nyquist Sampling Theorem explained also in the Reducible video in the minute 31:40. Or in the video of Akash Murty. Also note that N will be our period.

https://www.youtube.com/watch?v=yYEMxqreA10&t=1236s

https://www.youtube.com/watch?v=vrXGaFV1AmE

Instead of using the letter i as our imaginary unit uses j. As a remark we multiply by 2π to make the function Z periodic instead of 2π periodic. Which means that a function f(x) of period 2π has the property that.

$$f(x) = f(x + 2\pi)$$

In contrast to the Z periodic functions that have the property that:

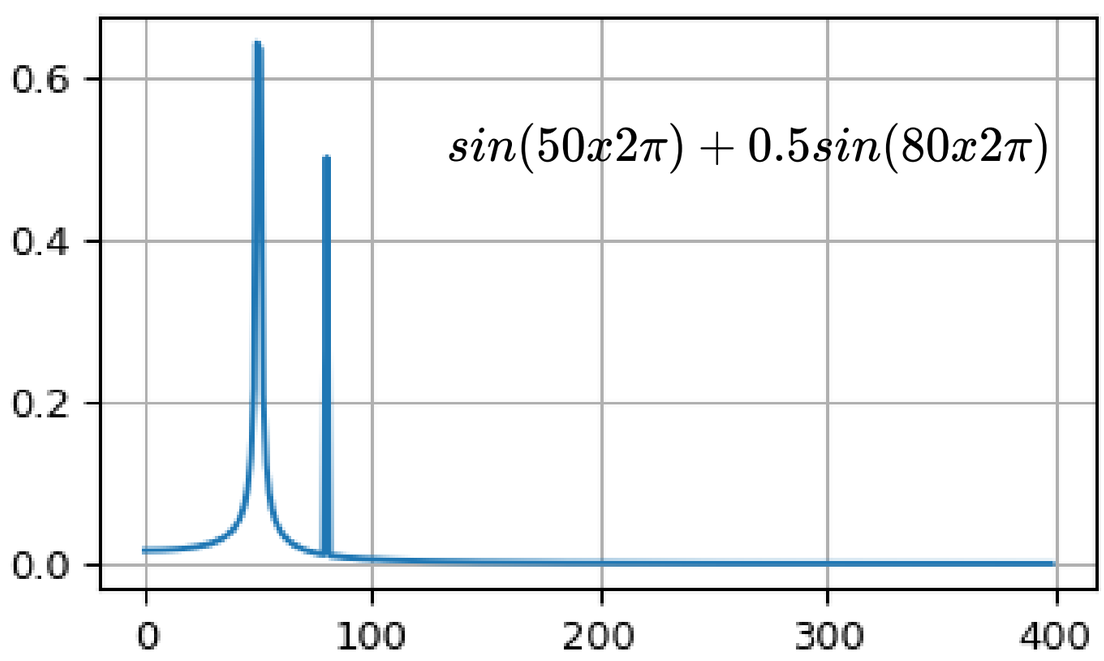
$$f(x) = f(x + n), \forall n \in \mathbb{Z}$$

The documentation uses k instead of x as our independent variable, and instead of write the frequency ω uses 1/N. Finally we use a negative sign to denote that the rotation in the complex plane is going clockwise instead of counter-clockwise. Is just a convention because the formula will work using counter-clockwise rotation just being careful of being congruent. And of course is a summary and not an integral because is finite. Now is clear what the formula tells us. And is clear that is the same formula that Wikipedia and many books described although the scipy notation is not the standard one.

As you can read scipy implement this using the fft and ifft modules of the Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT) respectively.

06 THE PERIODOGRAM.

The graphical way to see what is the DFT doing is the plotting a graph with the amplitudes of different frequencies. Such graph is called the periodogram. For example that is a periodogram of the sum of two sines:



THE TRADITIONAL ANALYSIS.

07 DOWNLOADING THE DATA.

Now, it's describe how cepheids are currently found using the traditional techniques which is by the using of main frequencies analysis (a periodogram analysis). So to show it with our real world example in astronomy all starts by downloading the data.

First install lightkurve library which contains itself the Kepler's dataset. One way to download the dataframe of a particular star could be the following code.

```
import lightkurve as lk
import pandas as pd
search = lk.search_lightcurve("KIC 9832235", mission = "Kepler")
lightcurve = search[0].download()
star = lightcurve.to_pandas()
```

We obtain a dataframe like the following. The most important columns are **time** and **flux.** Flux is somewhat another name to denote brightness. And is important to mention that time is measure in Earth's days. The data that we obtain look like the picture below.

time	flux	flux_err	quality	timecorr	centroid_col	centroid_row	cadenceno	sap_flux	sap_flux_err	sap_bkg
131.512133	15200.517578	5.355344	0	0.001139	32.368120	419.716114	1105	10270.300781	3.198522	313.378632
131.532567	15195.247070	5.360495	0	0.001140	32.368298	419.715750	1106	10267.377930	3.198226	313.043732
131.553002	15199.755859	5.355632	0	0.001141	32.368213	419.715850	1107	10270.215820	3.198484	313.236725
131.573436	15189.259766	5.353740	0	0.001141	32.368264	419.716400	1108	10262.474609	3.197757	313.368683
131.593870	15191.637695	5.354620	0	0.001142	32.367503	419.716326	1109	10265.097656	3.198110	313.303192
...	...	...	...	...	...	...	...	...	...	...
164.901770	15153.568359	5.301619	0	0.002279	32.349899	419.725684	2739	10049.682617	3.171345	289.742645
164.922204	15155.409180	5.309118	0	0.002280	32.350781	419.725780	2740	10052.774414	3.171373	288.822906

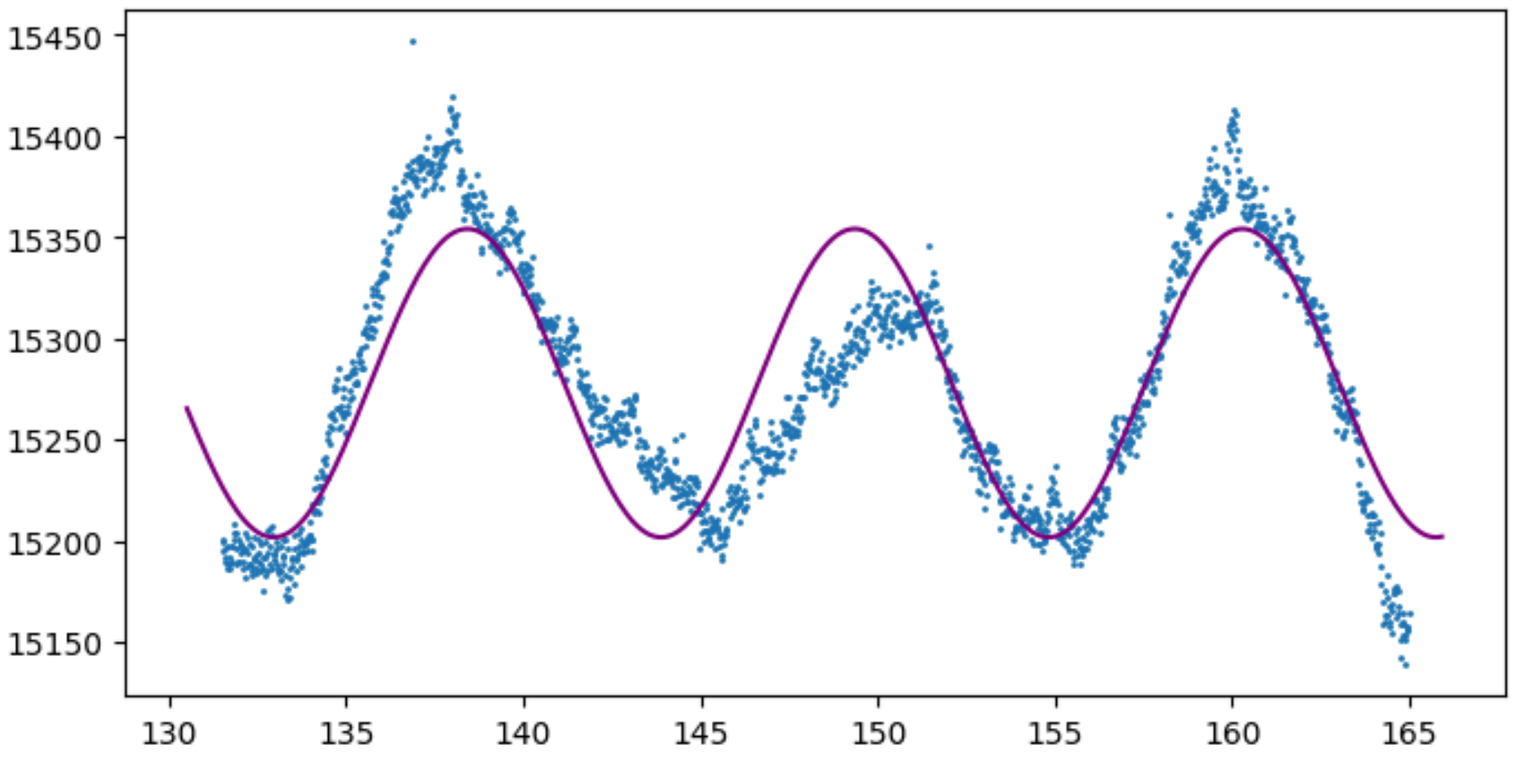
Of course if we want to get the data for multiple stars we can make use of some code like the one below.

```
def create_dataset():
  dataframes_folder = "some path"
  for i in range(1, 1001):
    try:
      search_result = lk.search_lightcurve(f"Kepler-{i}", author="Kepler", cadence="long")
      lightcurve = search_result[0].download()
    except:
      print(f"Star number {i} file does not exist.")
      continue
    dataframe = lightcurve.to_pandas()
    dataframe.dropna(subset=["flux"], inplace=True)
    dataframe.to_csv(dataframes_folder+f'object{i}.csv')
```

08 GET THE MAIN FREQUENCIES.

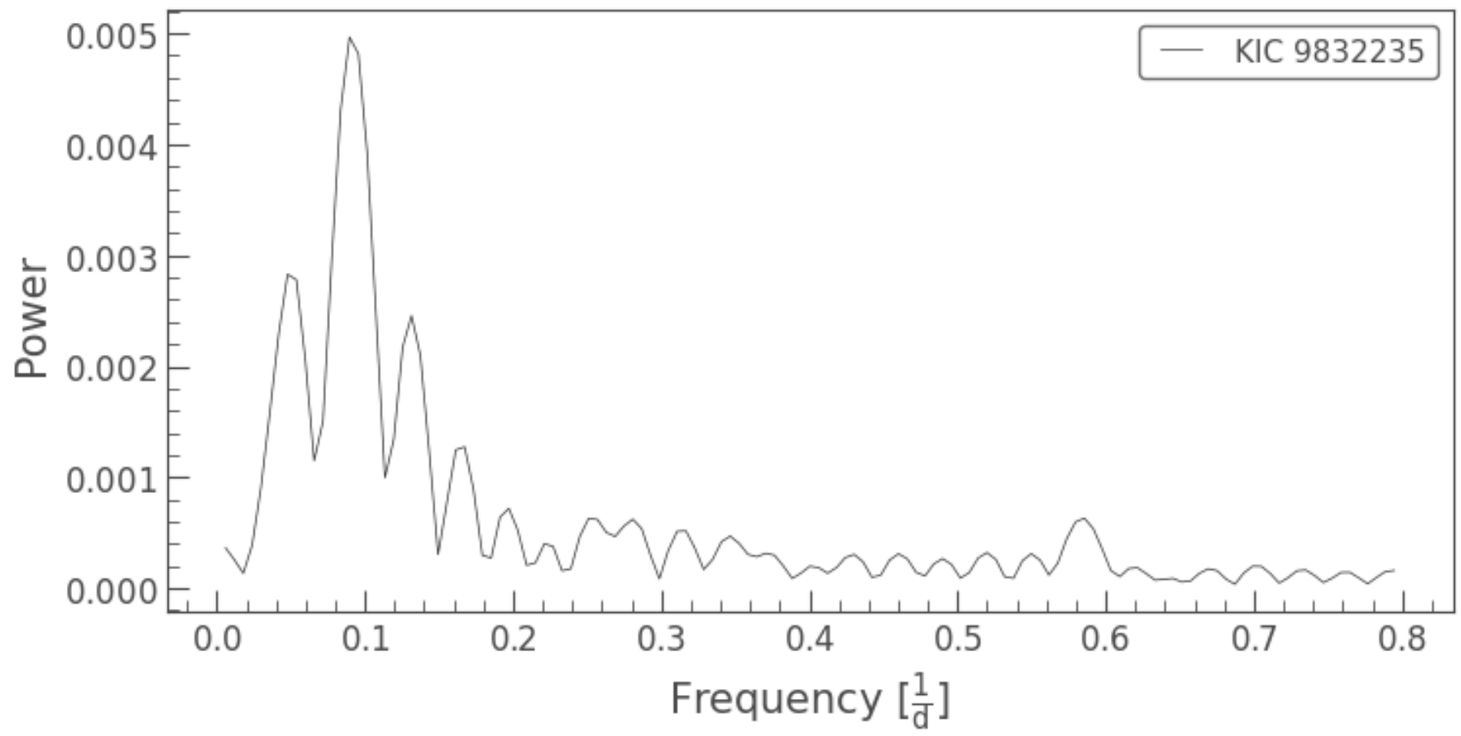
The strategy will be the following. If some star is periodic, then some frequency in the data will display a very high peak compared to the other. Such peak will be the called the **main harmonic**. It means that if we wish to approximate a function with just one sine wave, such frequency will represent the most fitting sine function of all. That is the main harmonic represents the target parameters that we are looking for.

For example given the star of the example above it looks like this.



As you could see the main harmonic is the sine function display above. In this case this sine function was found using the tool developed in this article, however it's easy to verify that is the same frequency that the Periodogram shows us. You can get such periodogram using the code below.

```
periodogram = lightcurve.normalize().to_periodogram(maximum_frequency=0.8)
max_power = periodogram.period_at_max_power
periodogram.plot()
```



Also the lightkurve library has an integrated property that gives os the harmonic frequency in terms of the period. In this case the max_power is 11.15 days or a frequency of 1 cycle each 0.09 days.

THE TRADITIONAL ANALYSIS

In case we want to know the other peaks we must create our own method. To do so we have to implement the Fast Fourier directly from Spicy instead of using Lightkurve because such method is not implemented there. The code looks like the following:

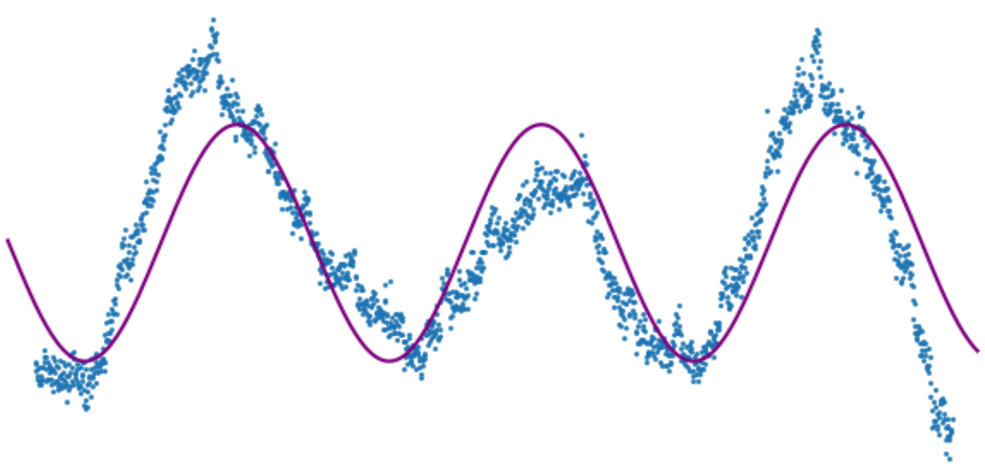
```
def traditional_predict(n_peaks, entries, values):
    mean = np.mean(values)
    parameters = []
    entries = np.array(entries)
    values = np.array(values)
    N = len(entries)
    T = entries[1] - entries[0]
    xf = spicy.fft.fftfreq(N, T)[:N//2]
    yfft = spicy.fft.fft(values - np.mean(values))
    yf = 2.0/N * np.abs(yfft[0:N//2])

    peak_indices1 = np.argsort(yf)[-n_peaks:]
    peak_amplitudes = []
    for index in peak_indices1:
        peak_amplitudes.append(yf[index])
    peak_amplitudes = sorted(peak_amplitudes, reverse=True)

    peak_indices2 = np.argsort(yf)[::-1][:n_peaks]
    peak_frequencies = xf[peak_indices2]

    horizontal_shifts = []
    for index in peak_frequencies:
        horizontal_shift = 1 / index
        horizontal_shifts.append(horizontal_shift)

    for index in range(n_peaks):
        parameters.append(peak_amplitudes[index])
        parameters.append(2.*np.pi*peak_frequencies[index])
        parameters.append(horizontal_shifts[index])
    combined_amplitude = sum(np.array(parameters[0::3])**2)**0.5
```

The first step is to convert our data, that is our points (expressed in time versus flux) into a numpy array. Then is clear that the number of points is the number of entries. And then we start by setting the parameters for the **Fast Discrete Fourier Transform Algorithm**. First the spacing between each entry is given by T. Second we obtain the sample frequencies to store in in **xf**, using the **spicy.fft.fftfreq** method and we pass all our points. Because this method will return positive and negative result, which are symmetric we will just extract the positive values that is half of the results using the **[:N//2]** slicing (// means floor division in Python). Finally we compute the Fourier Transform of the values which will return a complex valued array. Thus if we want to measure just the frequencies which their respective amplitudes then we need to transform our original **yfft** variable into a new numpy array with real values numbers, in particular we want the modules of the complex valued values and then multiplied each value by **2.0/N** to get amplitudes instead of powers. Such new array will be named **yf**.

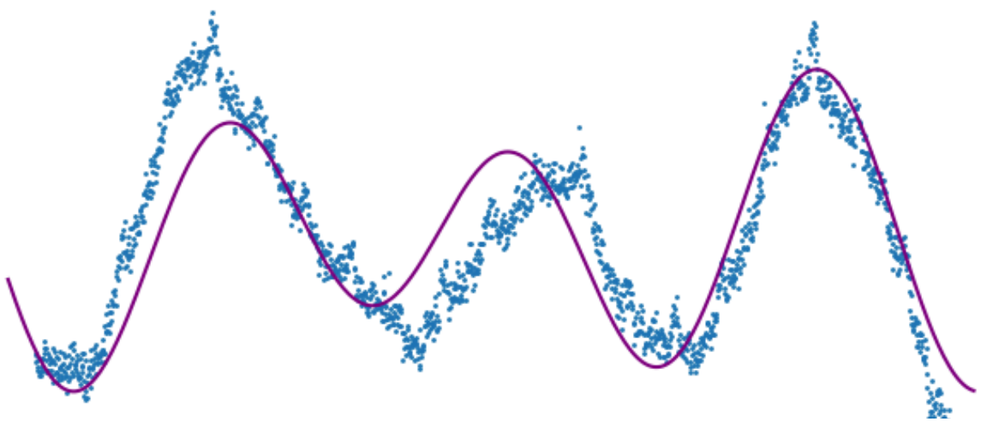
09 MEASURING THE RESULTS.

One of the most difficult challenges is to **obtain meaningful metrics of the results.** The main problem is that is we use the usual metric of Mean Squared Error or Root Mean Squared Metric, are so much sensitive to the points that are distant from the mean. Thus the result of using this metric is that the fittest models is just a line that passes through the mean. In order to avoid such problem is preferable to use **Absolute Error** divided by the range or the Mean Squared Loss (or the Root version) but with the peculiarity that is only going to consider the entries such that their flux in in the interval of the Mean + Amplitude and Mean-Amplitude and (if is desired multiplied by 5 in order to make it look congruent with respect to Absolute Error). This last error is called **Masked Mean Squared Error** in the code. Some of the results are showed below, using the traditional method in the left. However you can see that the regression in the right are much better. Such regressions are those that use the new developed method. What is doing is discussed in the next section.

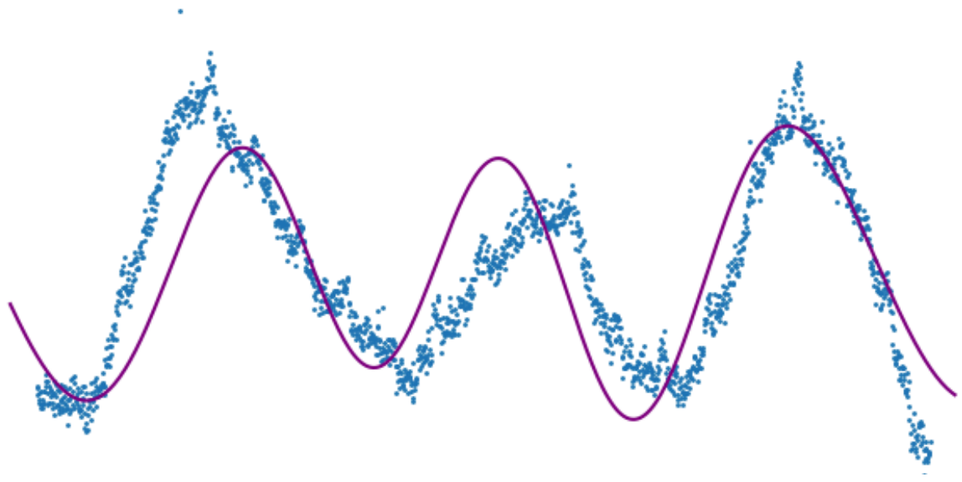
The traditional method.



Absolute error = 8. 0702
Root mean squared error = 8. 1477

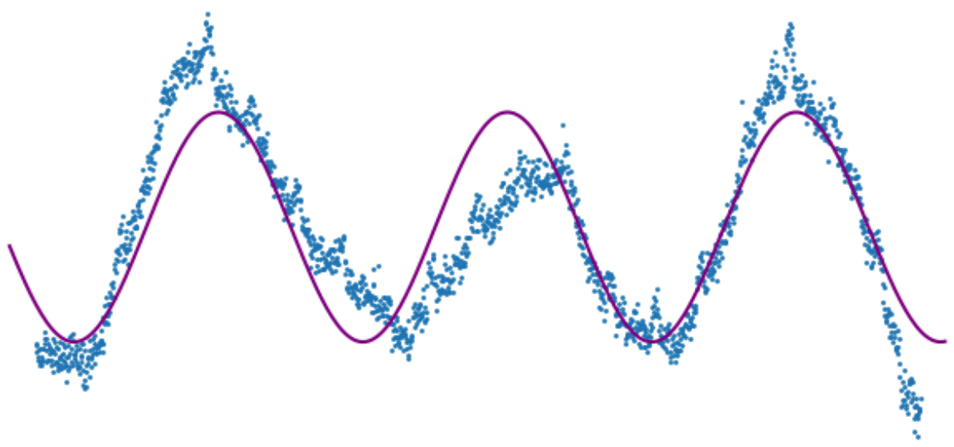


Absolute error = 8. 1477
Root mean squared error = 7. 3206

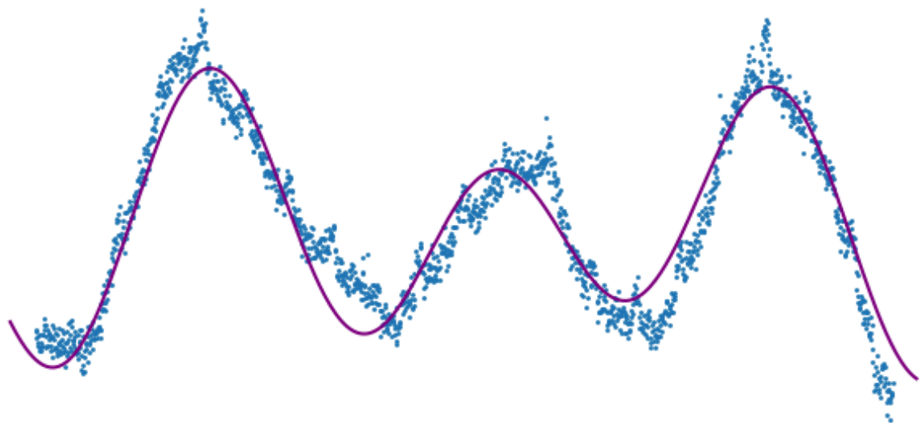


Absolute error = 10. 2473
Root mean squared error = 9. 0163

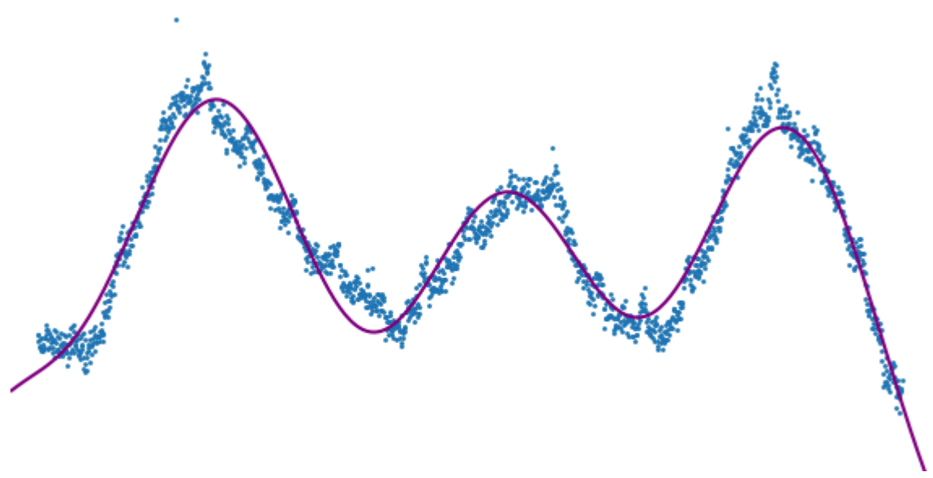
The developed method.



Absolute error = 7. 8394
Root mean squared error = 2. 3358



Absolute error = 5. 3190
Root mean squared error = 0. 3672



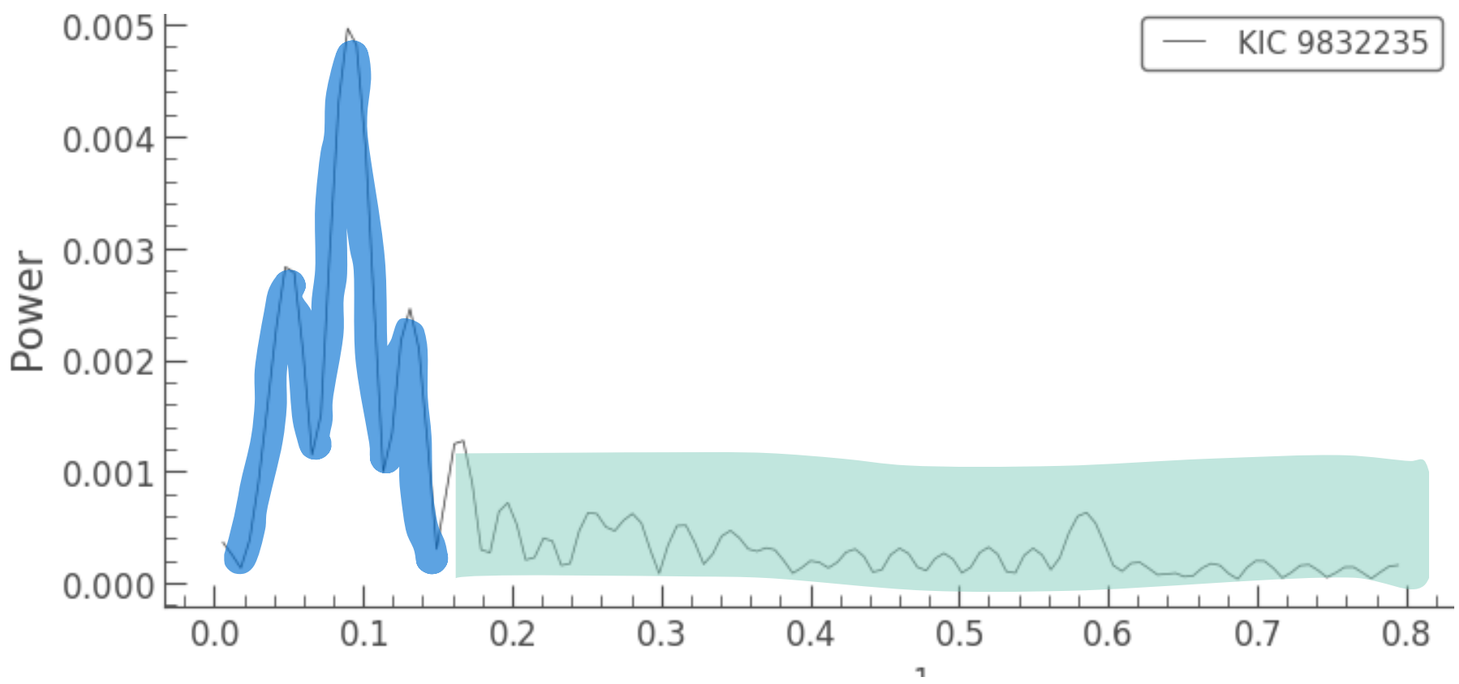
Absolute error = 2. 1543
Root mean squared error = 0. 3327

THE DEVELOPED METHOD.

10 WHY THE PREVIOUS METHOD DOESN'T WORK?

In order to improve the results we get in the previous analysis which ironically seems to get words as we add more trigonometric terms, let's analyze what's happening. To understand it better let's see an example.

Suppose we want to fit a trigonometric trinomial to the data, as is illustrated in the last picture of the previous page. Now let's see what we are doing visually using our periodogram.



As we can see the only information of the periodogram that we are considering to do the analysis are the three main peaks (in the above picture colored in blue). However all the information of the rest of the peaks is not considered (colored in green). This is really important because although the frequencies mean and amplitudes are improved as we sum more terms, the phase is becoming terribly worse which depends in the sum of all terms. In other words to get the actual phase we have to use the trigonometric polynomial that results of the addition of all frequencies which is a terrible idea recall this will lead to a perfect overfitting getting nothing that a useless model.

11 HOW TO FIX THE PROBLEM? FINDING THE MEAN.

In order to use all the information we have to find the optimal parameters in a another way. Some way which doesn't make use of the traditional ML regression or directly the Discrete Fourier Transform. Maybe we have to combine both ideas.

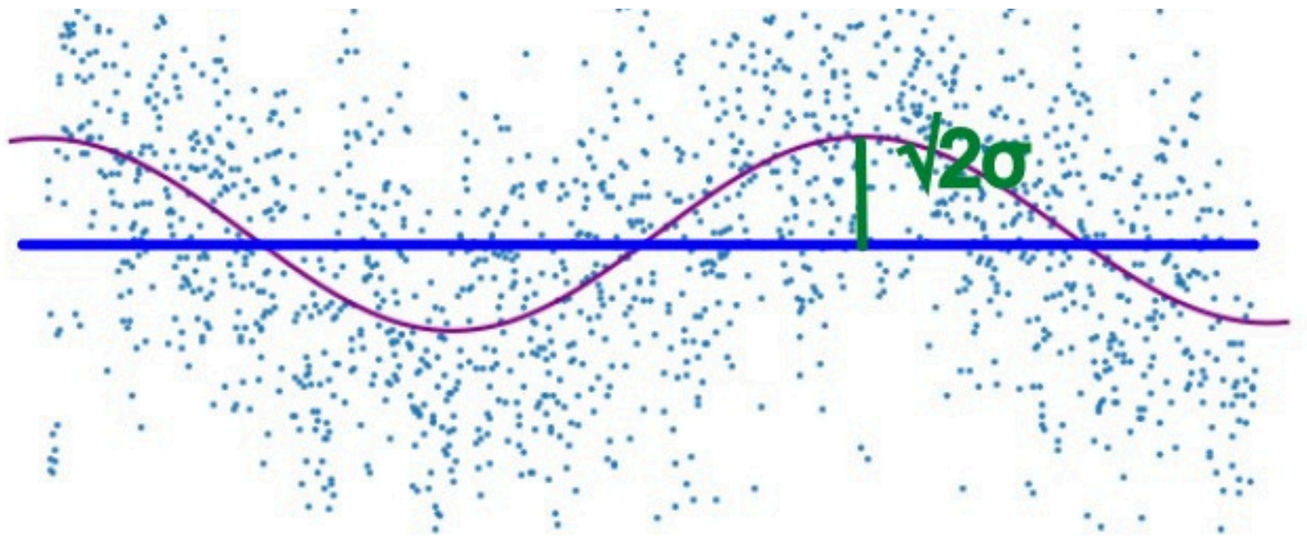
First let's think how to find the parameters without using any of those methods. Which one will be the easier to find. Well notice that the mean is easy to compute because is literally the mean of the fluxes, although we have to first clean the dataframe. We can use a code like the one below. And a picture of this idea is on the next page.

```
z = np.abs(spicy.stats.zscore(dataframe["flux"]))
outlier_indices = np.where(z > 2)[0]
dataframe.drop(outlier_indices, inplace=True)
guess_mean= np.mean(flux)
```

12 FINDING THE AMPLITUDE.

To find the amplitude we can just find the mean and find the standard deviation. Making some tests we can see that the amplitude will be approximately the standard deviation times root of 2. A picture is this is below.

$$A = \sqrt{2}\sigma$$



13 FINDING THE FREQUENCY AND PHASE.

In order to use all the information we will use the information provided by the Discrete Fourier Transform in a clever way. The key is to use another spicy implementation. Which is called **spicy.optimize_curve.** Although in order to make it work we have to make some initial guess. In this case we will use the harmonic (the main sine term) as our guess and append zeros to the rest of parameters. However another guess could be use the information that the **traditional_predict()** method give us. The credit about how we can combine this two methods is for the following StackOverflow user who also follows our reasoning about using the mean and standard deviation as we do although he or she only gives us the answer for a single sine function, so we have to modify the code using the **ntrigs()** function showed below:

Credit: https://stackoverflow.com/questions/16716302/how-do-i-fit-a-sine-curve-to-my-data-with-pylab-and-numpy

```
def trig(x, a, b, c):
  return a * np.sin(b * x + c)

def ntrigs(time, mean, *params):
  result = mean
  for index in range(0, len(params), 3):
    result += trig(time, params[index], params[index+1], params[index+2])
  return result
```

Anyway the code will work although it has some limitations. The main disadvantages are that is doesn't always find the optimal parameters and second its slower than the first method because we have to perform the same algorithm as before but now also **a Machine Learning Algorithm.**

The code is below.

```
guess_mean= np.mean(flux)
guess_amp = np.std(flux) * 2.**0.5
guess_freq = abs(ff[np.argmax(Fflux[1:])+1]) # excluding the zero frequency "peak", which is related to mean
guess = [guess_mean, guess_amp, 2.*np.pi*guess_freq, 0.0]
if n_equation > 1:
  for i in range(n_equation-1):
    guess.append(0)
    guess.append(guess_freq) #Each frequency can be improved if we used the guess of the traditional algorithm
    guess.append(0)
guess = np.array(guess)
optimization, convolution = spicy.optimize.curve_fit(equation, xdata=time, ydata=flux, p0=guess)
```

14 A SECOND ADVANTAGE OF USING THE NEW METHOD.

Although have meaningful metrics of how well our model is doing by consider the rest of information, actually the inspiration of this code came by looking from cepheids. And Although cepheids look like a single sine function is true that is not a perfect sine but instead a skewed sine. Thus if we want to find skewed sines instead of a regular sine wave we have to use a different function, some function like the one below and see how easy is to implement it using this new code.

$$Acos(\omega x + \phi) - \frac{A}{5}sin(2\omega x + \phi) + V$$

```
def skewsin(time, mean, amplitude, frequency, h_shift):
  character1 = amplitude * np.cos(frequency*time + h_shift)
  character2 = (amplitude / 5) * np.sin(2*frequency*time + h_shift)
  return character1 - character2 + mean

prediction_skewsin = predict(dataframe["time"], dataframe["flux"], -1)
```

As a reference each equation has an associated number. The reason of using numbers instead of their name as a string is that we can call **ntrigs()** of n_equation parameters whenever n_equation is bigger than or equal to 2. But for special functions like **skewsin()** we can make use of the negative numbers that are empty and so free to use. We will associate then **skewsin()** with the number -1.

```
def get_equation(n_equation):
  if n_equation == 0:
    return linearfunc
  elif n_equation == 1:
    return sinfunc
  elif n_equation == -1:
    return skewsin
  else:
    return ntrigs
```

RESULTS AND SUMMARY.

Finally using the same metrics that we have already explained in Section 9, we can now understand the results of the front page. The first two are the improvement with respect to the mean errors of both methods and finally we loose in computational time because we have to compute two algorithms instead of one.

35.7%

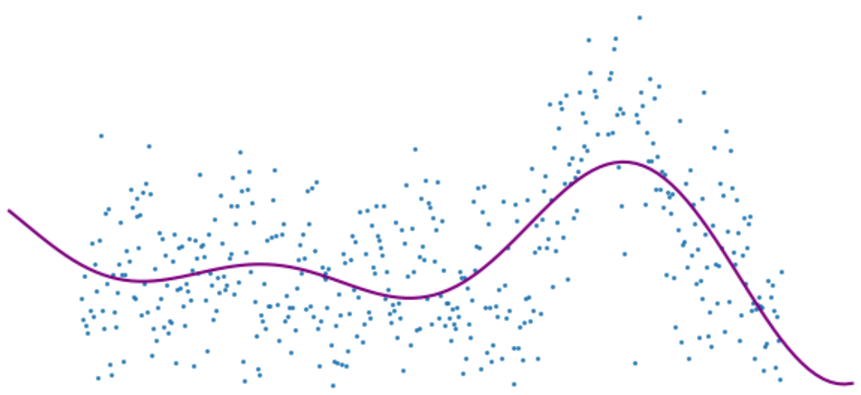
OF IMPROVEMENT IN ABSOLUTE ERROR LOSS

35.7%

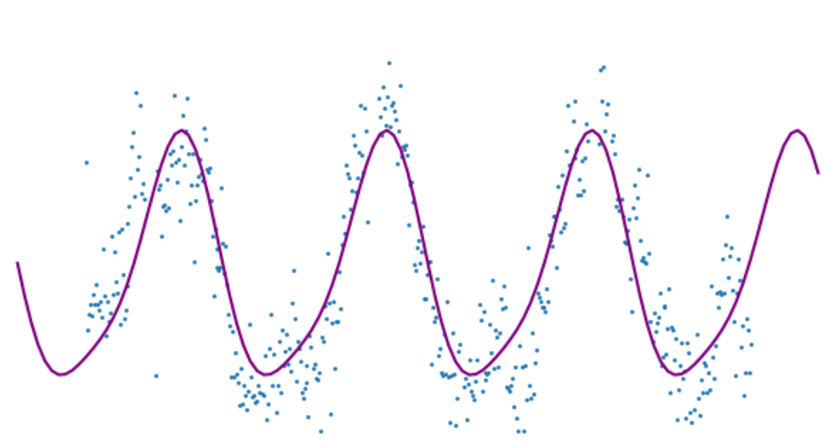
OF IMPROVEMENT IN ROOT MEAN SQUARED LOSS*

AT EXPENSE OF A DECREMENT IN VELOCITY

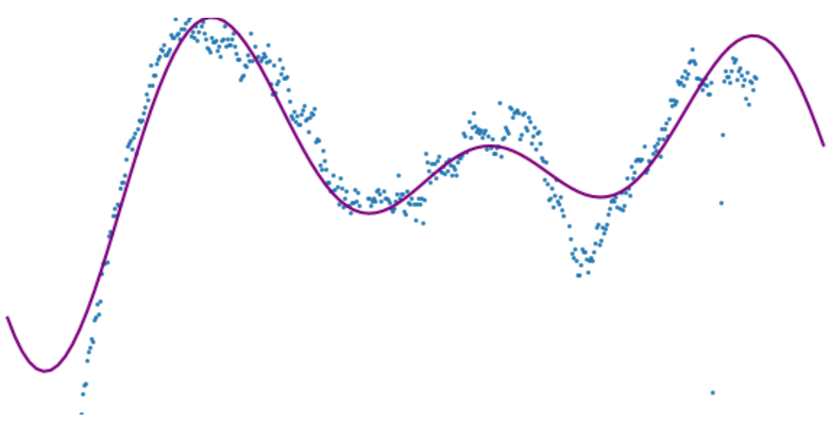
Some pictures are showed below for two trigonometric monomials and for skewsine. Notice that in the code it was implemented a linear regression as well to discard those stars where it's clear there isn't any periodic component. Also some stars cannot be represent by just one or two terms.



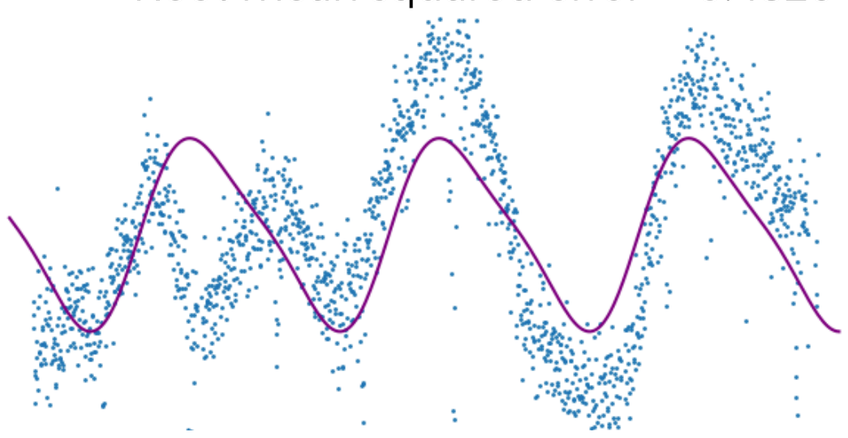
Absolute error = 11. 6896
Root mean squared error = 0. 0887



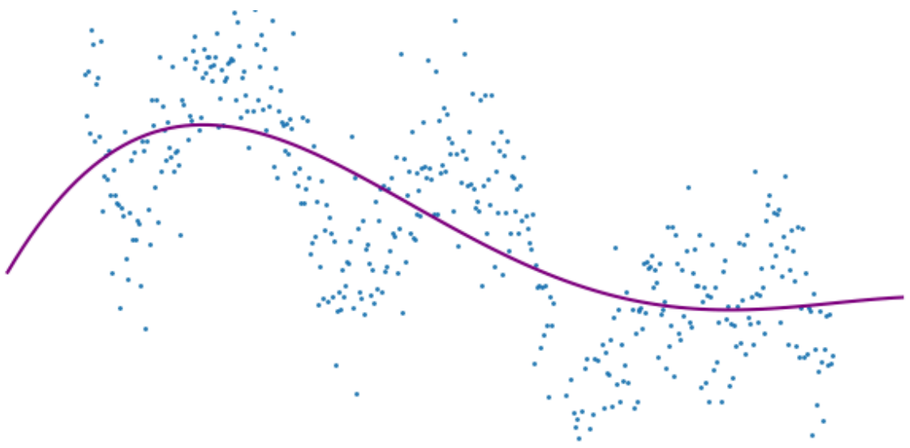
Absolute error = 9. 7342
Root mean squared error = 0. 1826



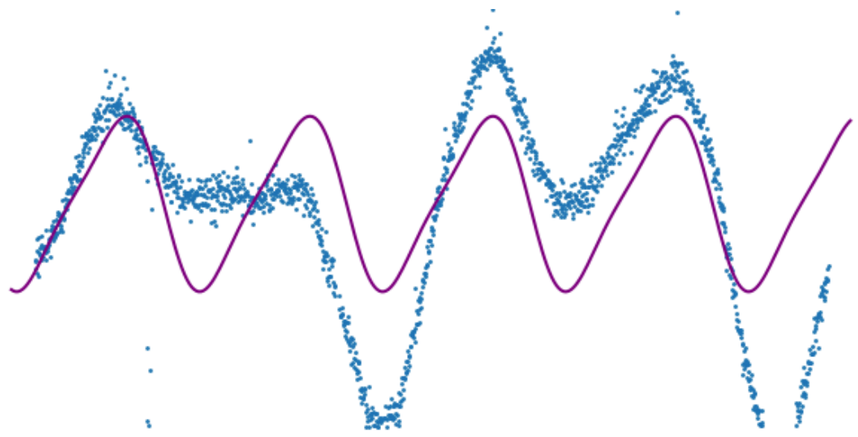
Absolute error = 5. 7691
Root mean squared error = 0. 4043



Absolute error = 13. 3155
Root mean squared error = 0. 1711



Absolute error = 13. 3921
Root mean squared error = 0. 1452



Absolute error = 13. 4599
Root mean squared error = 0. 5269