

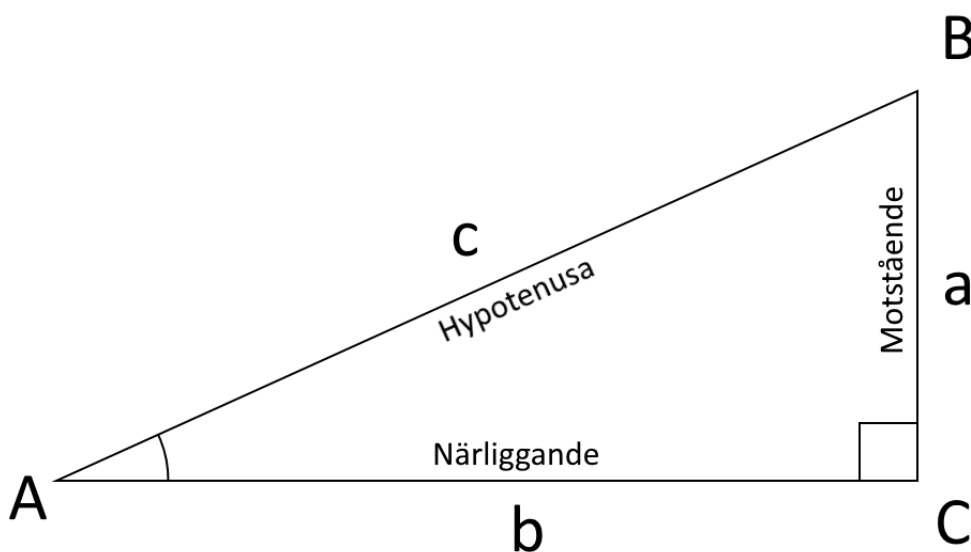
# Laboration 6 - Tidsserieanalys

## Trigonometri och avancerad kurvanpassning

I den förra labben undersökte vi sambandet mellan period och ljusstyrka för cepheid variabler. Denna gång ska vi titta närmare på hur man kan få fram datan vi använde förra gången. Först måste vi dock prata en del om den matematiska grunden vi behöver använda.

### Trigonometri

Trigonometri är läran om vinklar och trianglar. Trigonometri används också för att räkna med vågor och därmed alla typer av system med periodiskt återkommande egenskaper. Det gör att trigonometri används inom områden så vitt skilda som astronomi, musikteori, ingenjörskonst och speldesign. Grunderna till trigonometrin ligger i förhållandena mellan sidorna i en rätvinklig triangel. Bilden nedan förklarar notationen och namnen som används.



Hörnen och deras vinklar har namn som är stora bokstäver och sidorna är namngivna med små bokstäver efter det hörn som är mitt emot dem. Hypotenusan kallas alltid den sidan som är mitt emot den räta vinkeln. Från hörnet A kallas sidan a för motstående och sidan b för närliggande. De trigonometriska funktionerna sinus, cosinus och tangens av vinkeln i A definieras som förhållandet mellan dessa sidor.

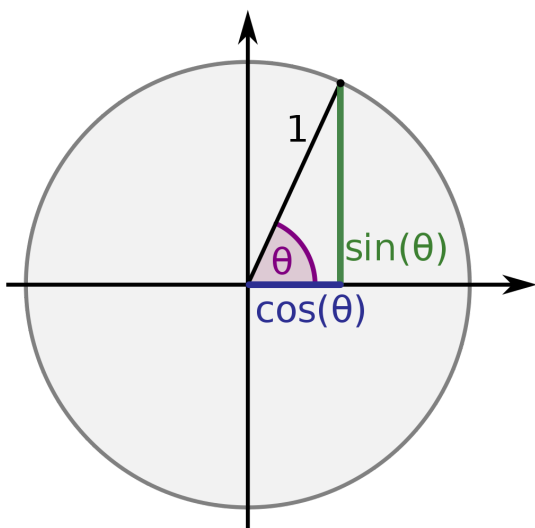


$$\sin(A) = \frac{\text{Motstående}}{\text{Hypotenusan}}$$

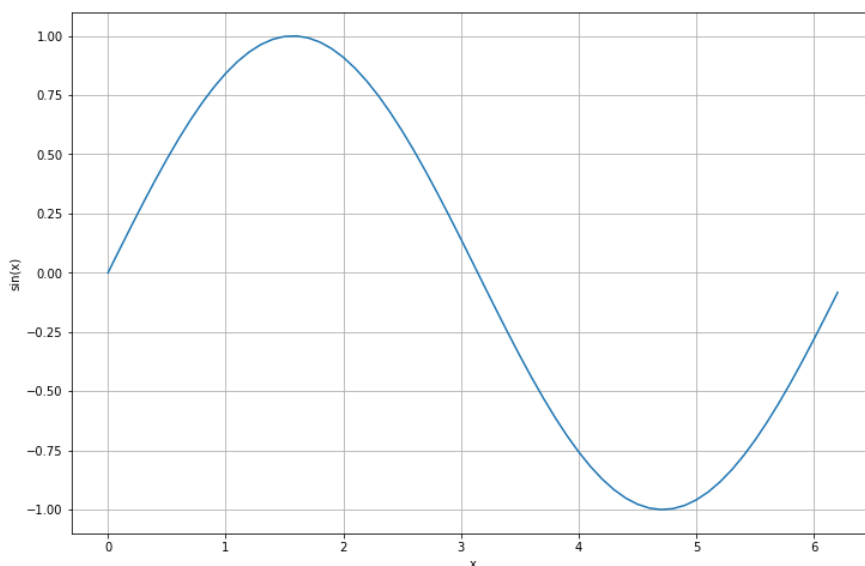
$$\cos(A) = \frac{\text{Närliggande}}{\text{Hypotenusan}}$$

$$\tan(A) = \frac{\text{Motstående}}{\text{Närliggande}}$$

Enhetscirkeln är ett användbart verktyg för att illustrera dessa samband. Enhetscirkeln är en cirkel med radie 1 (en enhet) med centrum i origo (mitten på ett koordinatsystem) och en vinkel theta ( $\theta$ ) mellan en linje från mitten till kanten. Om man ritat en rätvinklig triangel med linjen som hypotenusan och ena kanten längs x-axeln kommer den vågräta kanten ha längden  $\cos(\theta)$  och den lodräta kanten ha längden  $\sin(\theta)$ .



Så hur är det här relaterat till vågor, kan man undra? Om man ritat en graf över värdet på sinus medans  $\theta$  går ett helt varv ser man denna figur.





## Astronomisk data

```
Time, Flux cepheid, Flux referance star
0.0 106434 82717
1.929 91419 85260
5.947 120214 84432
8.996 120741 89749
11.991 92946 84008
15.03 109158 85489
16.93 124252 87863
19.887 103821 86779
23.916 99767 74080
26.914 127669 88864
29.981 112669 89661
32.916 96869 89802
33.892 96755 72111
39.969 104375 70546
41.973 93100 82763
44.96 107175 85829
48.908 112264 84908
50.948 94373 65277
56.0 113433 83417
```

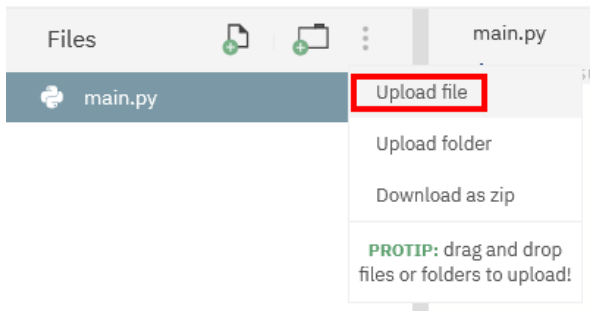
Att mäta perioden på en variabel stjärna låter kanske enkelt, men är inte så lätt i praktiken. Problemet består i att man sällan har möjlighet att göra tillräckligt många observationer för att se ett tydligt mönster. Vår data finner nu i filen `cephheid_obs.txt`. Datan denna gång är tidpunkt, flux från en cepheid och flux från en annan stjärna med konstant ljusstyrka. Flux är ett mått på ljusstyrka. Genom att ha värden för två stjärnor för alla tidpunkter kan man dela ljusstyrkorna med varandra och därmed automatiskt kompensera för variationer mellan mätningarna från saker som förändringar i atmosfären, störande ljus eller olika kalibreringar av instrumenten. Nackdelen är att man förlorar det absoluta måttet på hur ljusstark stjärnan är, men vi behöver inte den informationen ändå.

Även i denna labb är det en bra idé att skriva `print(data)` på ställen man vill synliggöra vad programmet gör och hur det funkar.

Vi vill läsa in denna data på samma sätt som i förra labben. Börja med att ladda upp filen, om ni använder repl.it<sup>1</sup>.

---

<sup>1</sup> <https://repl.it/languages/python3>



Öppna och läs filen med

- `f = open("cephheid_data.txt", "r")`
- `line_list = f.readlines()`
- `f.close()`

Om man tittar i den ursprungliga filen eller skriver ut `line_list` så ser man att den första raden är en rubrik och inte data, så den tar vi bort genom att definiera om listan.

- `line_list = line_list[1:]`

Reservera tre listor, en för varje fält i filen.

- `time = []`  
`flux_ceph = []`  
`flux_ref = []`

Skriv en for-loop som går igenom varje rad i listan med rader,

- `for line in line_list:`

och dela varje rad i flera "ord", där varje ord är ett värde, med `split`. Håll i åtanke att vi har andra skiljetecken, mellanslag istället för `|`, och inte behöver göra några kontroller för orimlig data.


- `words = line.split(" ")`

Fyll sedan listorna med värdena, konverterade till siffror

- `time.append(float(words[0]))`
- `flux_Fc.append(float(words[1]))`
- `flux_Fr.append(float(words[2]))`

Och kom ihåg att göra om listorna till arrayer,

- `time = np.array(time)`
- `flux_ceph = np.array(flux_ceph)`
- `flux_ref = np.array(flux_ref)`



```

1  import numpy as np
2
3  time = [] # Reserve lists
4  flux_ceph = []
5  flux_ref = []
6
7  f = open("cepheid_obs.txt", "r") # Open the file for reading
8
9  line_list = f.readlines() # Read all line in the file
10
11 f.close() # Remember to close the file
12
13 line_list = line_list[1:] # Remove the header
14
15 for line in line_list:
16     words = line.split(" ")
17     time.append(float(words[0])) # Convert to float and add to list
18     flux_ceph.append(float(words[1]))
19     flux_ref.append(float(words[2]))
20
21 time = np.array(time) # Convert to arrays
22 flux_ceph = np.array(flux_ceph)
23 flux_ref = np.array(flux_ref)
24

```

Sedan kan vi dela fluxen av cepheiden med fluxen på referensstjärnan för att få den relativa fluxen.

- `flux_ratio = flux_ceph/flux_ref`

Om man ska skriva ut data utan att rita linjer mellan punkterna kan man använda `pyplot.plot(x, y, '*')` som vi gjorde förra gången eller `pyplot.scatter(x, y)`, resultatet blir samma. Scatter är en annan typ av graf som är till för att visa spridning och i detta fall funkar den bra. Skriv en graf genom att inkludera

- `import matplotlib.pyplot as pyplot`

längst upp och

- `pyplot.title("Cepheid observations")`
- `pyplot.xlabel("Time [days]")`
- `pyplot.ylabel("Flux ratio [A.U.]")`

Följt av

- `pyplot.scatter(time, flux_ratio)`

Man kan lägga till ett rutnät i bakgrunden med kommandot

- `pyplot.grid()`

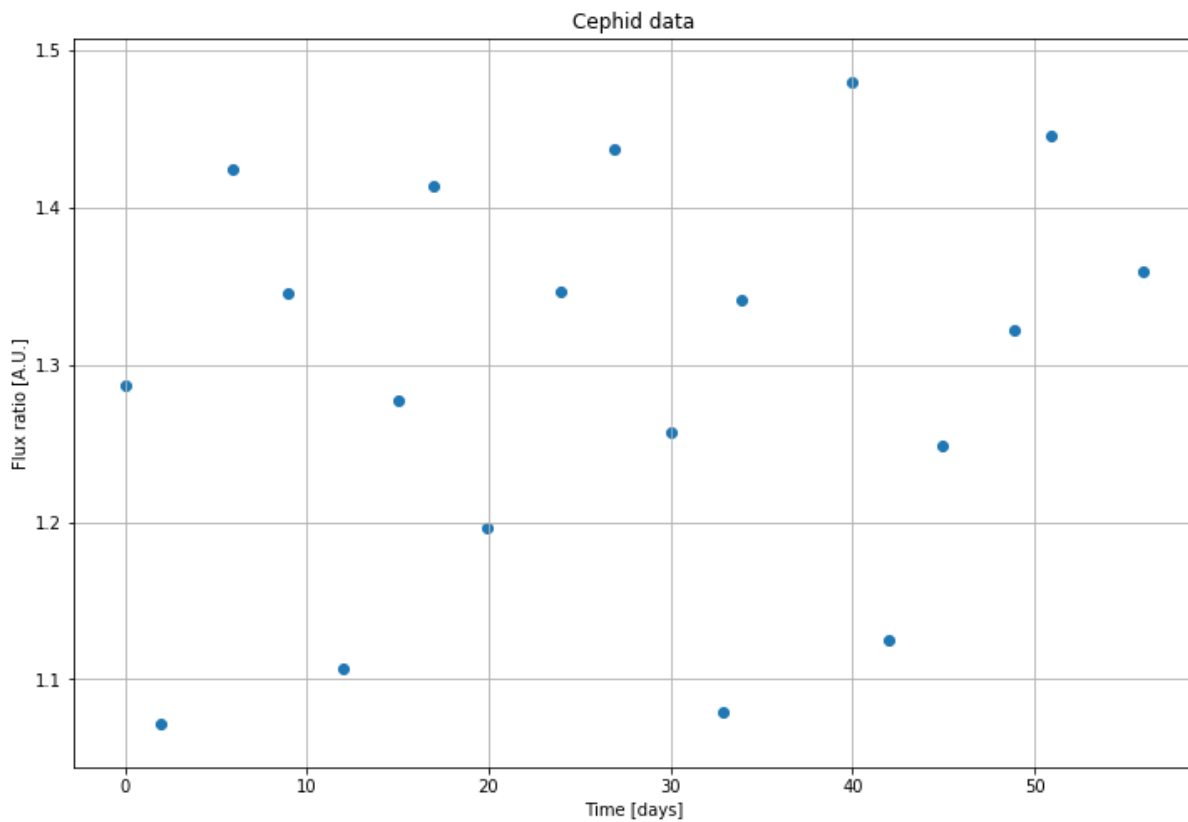
glöm inte

- `pyplot.show()`

i slutet.

```
1 import numpy as np
2 import matplotlib.pyplot as pyplot
3
4 time = [] # Reserve lists
5 flux_ceph = []
6 flux_ref = []
7
8 f = open("cepheid_obs.txt", "r") # Open the file for reading
9
10 line_list = f.readlines() # Read all line in the file
11
12 f.close() # Remember to close the file
13
14 line_list = line_list[1:] # Remove the header
15
16 for line in line_list:
17     words = line.split(" ")
18     time.append(float(words[0])) # Convert to float and add to list
19     flux_ceph.append(float(words[1]))
20     flux_ref.append(float(words[2]))
21
22 time = np.array(time) # Convert to arrays
23 flux_ceph = np.array(flux_ceph)
24 flux_ref = np.array(flux_ref)
25
26 flux_ratio = flux_ceph/flux_ref # Compute the flux ratio
27
28 pyplot.title("Cepheid observations")
29 pyplot.xlabel("Time [days]")
30 pyplot.ylabel("Flux ratio [A.U.]")
31 pyplot.scatter(time, flux_ratio) # Plot but with only points
32 pyplot.grid() # Add grid
33 pyplot.show() # Print the plot
34
```

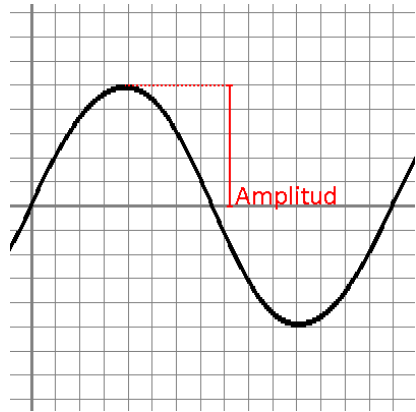
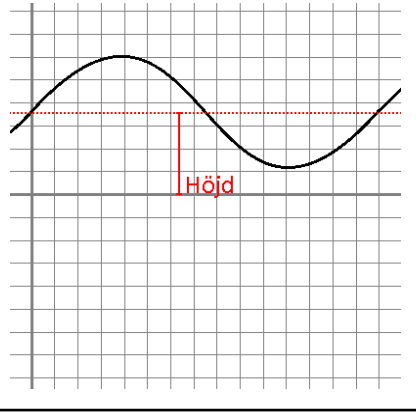
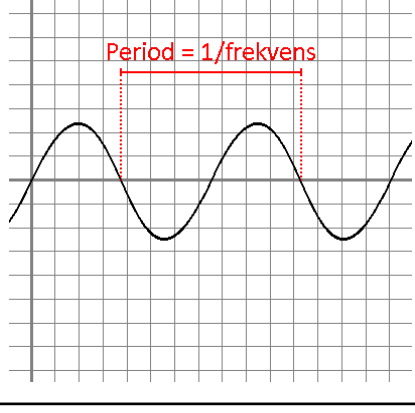
Om man skriver ut resultatet av divisionen över tiden får man följande figur.



Det är inte väldigt enkelt att se hur många dagar det tar för stjärnan att pulsera. Vi kan få datorn att anpassa en sinuskurva till denna data för att se vilken period en sådan kurva har. Principen här är samma som i förra labben, vi låter datorn räkna fram den kurva som passar minst fel till datan vi ger den. Skillnaden är att denna gång vill vi ha en kurva som är mycket mer komplicerad än ett polynom så vi måste använda ännu starkare verktyg.

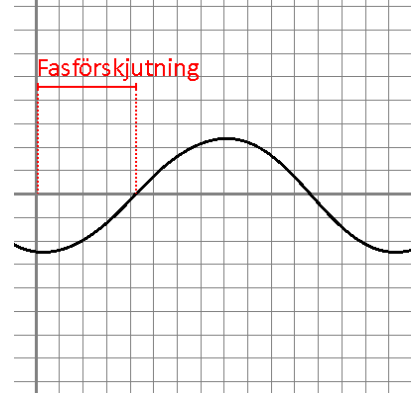
Det finns 4 möjliga parametrar att ändra på en sinuskurva. Dessa förklaras i tabellen nedan.



Namn	Standardvärde	Beskrivning	Bild
Amplitud	1	Avstånd från mitten av kurvan till högsta punkten	
Höjd	0	Avstånd från x-axeln till mitten på kurvan	
Svängningshastighet (Period)	1	Svängningshastighet, inversen av avståndet från en topp till nästa	





Förskjutning	0	Avstånd från y-axeln till en punkt på kurvan där den skulle ha korsat x-axeln om de andra parametrarna hade sina standardvärden.	
--------------	---	--	--

Av dessa värden är svängningshastigheten den viktigaste för oss eftersom den kommer tala om för oss perioden av cepheidvariabeln. Avståndet från en topp till nästa kallas en period. 1 delat med perioden är frekvensen dvs antal svängningar per sekund, och frekvensen gånger ett varv är svängningshastigheten. Ett varv är 360 grader men det är ofta bättre att använda den matematiskt välgrundade enheten radianer.  $2 * \pi$  radianer är ett varv. Frekvensen är därför  $2\pi$  delat med perioden.

Ekvationen för en sinuskurva med alla parametrar ser ut såhär

$$y = H + A * \sin(x * S + F)$$

där parametrarna representeras av sina initialer.

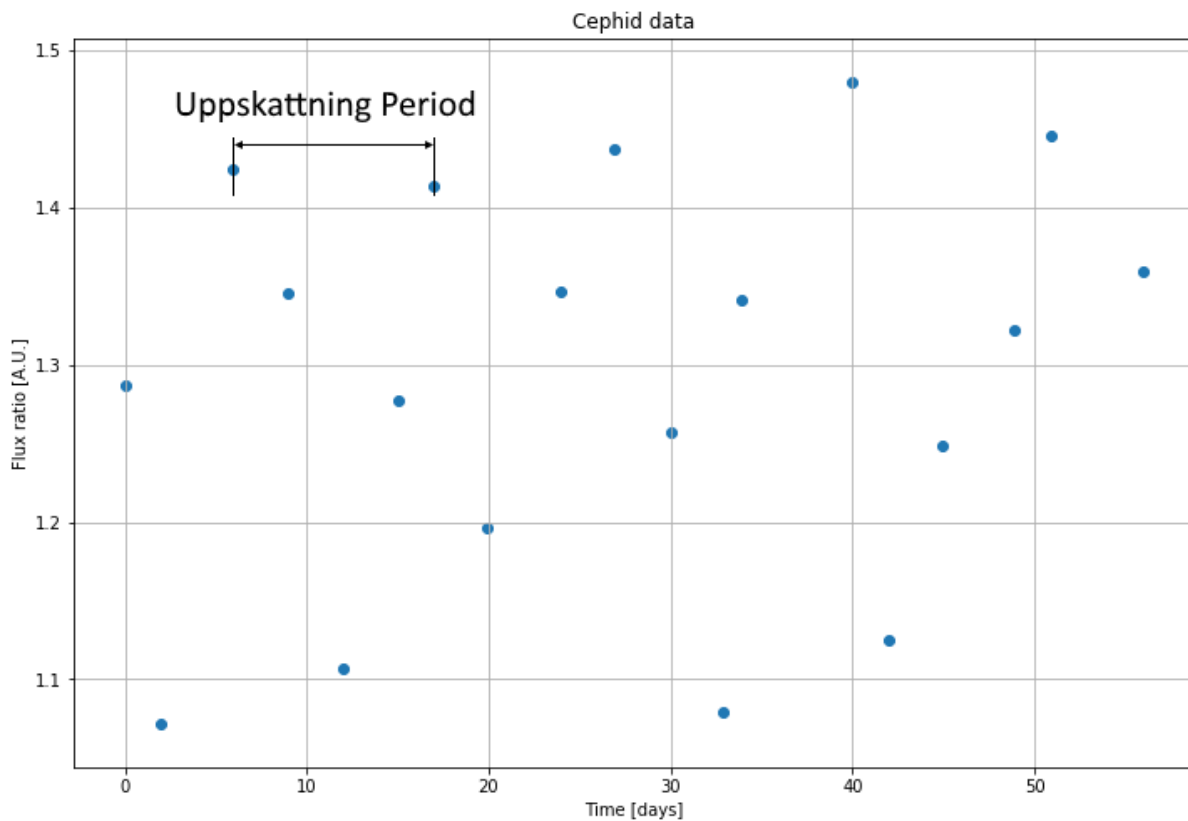
## Avancerad kurvanpassning

Funktionen som anpassar en godtycklig funktion till data är väldigt kraftfull men också svåränvänd. Vi tillhandahåller därför en något förenklad version som bara fungerar på sinuskurvor. Den finns i biblioteket `userFunctions` och importeras med namnet `sineFit` på samma sätt som i förra labben, med

- `from userFunctions import sineFit`

Funktionen `sineFit` tar tre argument, dessa är x-data, y-data och startgissningar för alla parametrar i sinusfunktionen. Funktionen svarar med en array med de justerade värdena på parametrarna, en tätare array med x-värden och en array med y-värden för en anpassade sinusfunktionen som motsvarar arrayen med x-värden. Detta är inte en perfekt process och man kan få olika svar från olika gissningar av startvärden.

`sineFit` bygger på `curve_fit` som gör minsta-kvadrat-anpassning, se dokumentationen för SciPy v1.4.1 Reference Guide<sup>2</sup> för detaljer.



Vi behöver komma med gissningar för alla 4 parametrar för en sinuskurva. Höjden kan man anta är nära snittvärdet av all y-data vilket vi kan få ut genom att använda funktionen `mean` från `numpy`. Den returnerar helt enkelt medelvärdet av datan den får. Amplituden är antagligen nära hälften av skillnaden mellan det största och minsta y-värdet. Här kan vi använda funktionerna `min` och `max`, dom ger det minsta och största värdet från en lista man det dom. Perioden kan man uppskatta att vara ungefär avståndet mellan två punkter nära toppen, enligt bild nedan. Förskjutningen är väldigt svår att uppskatta och dessutom oviktig så vi nöjer oss med att gissa att den är noll.

---

<sup>2</sup> [https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve\\_fit.html](https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html)



Vi skriver in våra gissningar som variabler,

- `guess_H = np.mean(flux_ratio)`
- `guess_A = (max(flux_ratio) - min(flux_ratio))/2.0`
- `guess_P = 12.0 # Only used to compute guess_S`
- `guess_S = 2.0*np.pi/guess_P`
- `guess_F = 0.0`

så är de enklare att hitta och ändra om vi skulle behöva. Vi skriver in alla i en lista som är lätt att skicka till `sineFit`. Notera att ordningen i listan är viktig eftersom programmet inte kan komma på själv vilket värde som är vilket, så de måste komma i ordningen:

Höjd, Amplitud, Svängningshastighet, Färförskjutning,


- `p0=[guess_H, guess_A, guess_S, guess_F]`

```

1 import numpy as np
2 import matplotlib.pyplot as pyplot
3 from userFunctions import sineFit # user made lib
4
5 time = [] # Reserve lists
6 flux_ceph = []
7 flux_ref = []
8
9 f = open("cepheid_obs.txt", "r") # Open the file for reading
10
11 line_list = f.readlines() # Read all line in the file
12
13 f.close() # Remember to close the file
14
15 line_list = line_list[1:] # Remove the header
16
17 for line in line_list:
18     words = line.split(" ")
19     time.append(float(words[0])) # Convert to float and add to list
20     flux_ceph.append(float(words[1]))
21     flux_ref.append(float(words[2]))
22
23 time = np.array(time) # Convert to arrays
24 flux_ceph = np.array(flux_ceph)
25 flux_ref = np.array(flux_ref)
26
27 flux_ratio = flux_ceph/flux_ref # Compute the flux ratio
28
29 pyplot.title("Cepheid observations")
30 pyplot.xlabel("Time [days]")
31 pyplot.ylabel("Flux ratio [A.U.]")
32 pyplot.scatter(time, flux_ratio) # Plot but with only points
33 pyplot.grid() # Add grid
34 pyplot.show() # Print the plot
35
36 guess_H = np.mean(flux_ratio) # Guess for height
37 guess_A = (max(flux_ratio) - min(flux_ratio))/2.0 # Guess for amplitude
38 guess_P = 12.0 # Only used to compute guess_S
39 guess_S = 2.0*np.pi/guess_P # Guess for the speed of swings
40 guess_F = 0.0 # Skip guessing the phase shift
41
42 p0=[guess_H, guess_A, guess_S, guess_F] # Arrange guesses in a list

```

Funktionen `sineFit` svarar med tre arrayer. För att definiera tre variabler som svaret på en funktion skriver man variablerna efter varandra separerade med ett kommatecken. För att göra



hela funktionsanropet behöver vi skriva variablerna vi vill definiera med funktionsanropet (`params, t_dense, sine_array`), funktionsanropet (`sineFit`), vår x-data (`t`), vår y-data (`flux_ratio`) och vår lista på gissningar på startvärden för parametrarna (`p0`),

- `params, t_dense, sine_array = sineFit(t, flux_ratio, p0)`

`params` blir nu en array med de optimerade parametrarna. Det är vi ute efter är perioden så vi räknar fram den från svängningshastigheten,

- `period = 2.0*np.pi/params[2]`

och sedan skriver vi ut den med `print`. Python kan man sätta ihop två strängar till en med `+` och man kan göra om siffror (oavsett float eller int) till en sträng med funktionen `str()`. Vi kan därför skapa en sträng som innehåller vårt svar genom att skriva `"Period is " + str(period)`, och lägga in den i en `print` för att skriva ut den, så behöver vi inte definiera några fler variabler,

- `print("Period is " + str(period))`

Vi vill rita in den nya sinuskurvan tillsammans med datan för att se att lösningen programmet kom fram till är rimlig. Om man använder Repl.it behöver man kommentera bort den tidigare plotten, som förra gången. Börja med att skriva grafens namn och axlarnas namn och enhet,

- `pyplot.title("Cepheid observations with fitted sine")`
- `pyplot.xlabel("Time [days]")`
- `pyplot.ylabel("Flux ratio [A.U.]")`

A.U. står för arbitrary units eftersom vi delar en flux med en flux, så vi har bara ett relativt värde utan riktiga enheter. `sineFit` ger oss två färdiga arrayer som är sinuskurvas x- och y-värden. I exemplet kallas x-datan `t_dense` och y-datan `sine_array`. Om man har flera linjer eller annan data i samma graf kan man vilja ha en legend. I `pyplot` skapas en sådan genom att man lägger till en etikett på varje kommando som ritas in mer data så här


- `pyplot.scatter(time, flux_ratio, label='Data')`
- `pyplot.plot(t_dense, sine_array, label='Fitted function')`

och sedan inkluderar man ett kommando som skapar en legend innan man skriver ut grafen,

- `pyplot.legend(loc='best')`

avsluta med

- `pyplot.grid()`
- `pyplot.show()`

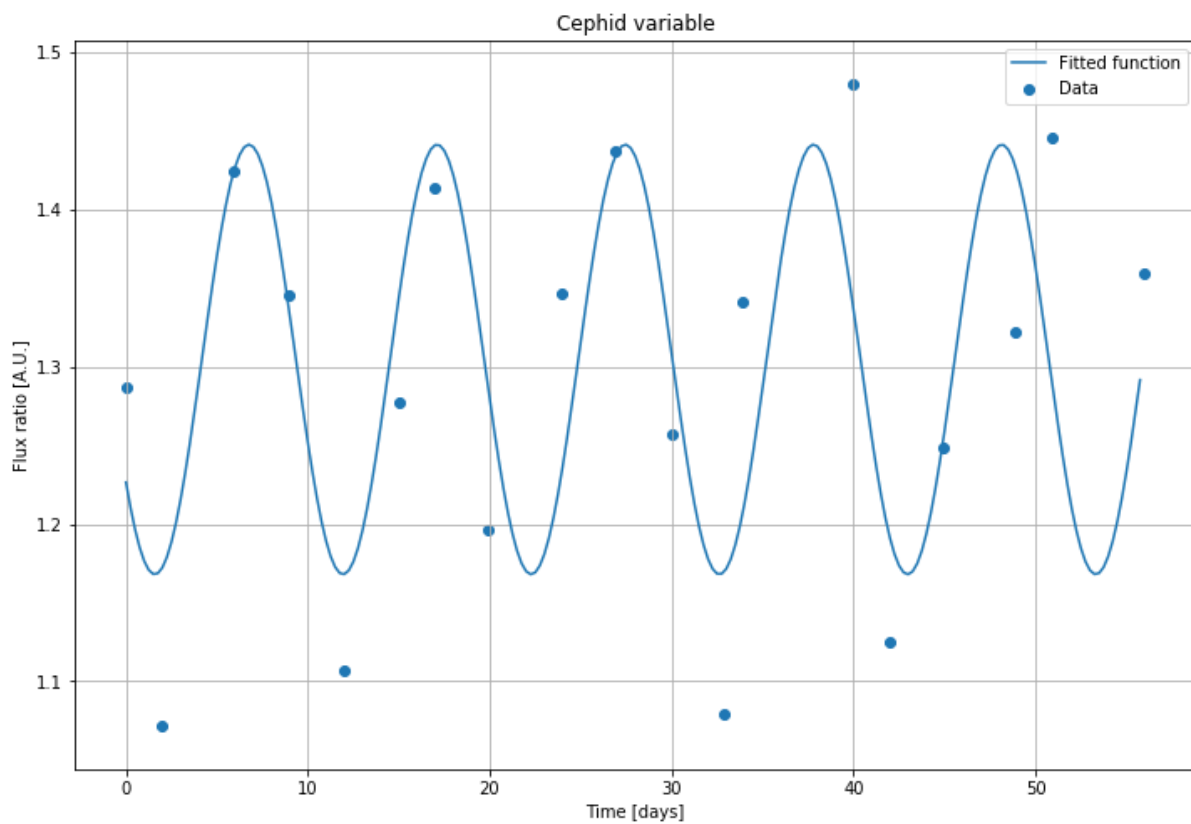


```

29 #pyplot.title("Cepheid observations")
30 #pyplot.xlabel("Time [days]")
31 #pyplot.ylabel("Flux ratio [A.U.]")
32 #pyplot.scatter(time, flux_ratio) # Plot but with only points
33 #pyplot.grid() # Add grid
34 #pyplot.show() # Print the plot
35
36 guess_H = np.mean(flux_ratio) # Guess for height
37 guess_A = (max(flux_ratio) - min(flux_ratio))/2.0 # Guess for amplitude
38 guess_P = 12.0 # Only used to compute guess_S
39 guess_S = 2.0*np.pi/guess_P # Guess for the speed of swings
40 guess_F = 0.0 # Skip guessing the phase shift
41
42 p0=[guess_H, guess_A, guess_S, guess_F] # Arrange guesses in a list
43
44 params, t_dense, sine_array = sineFit(time, flux_ratio, p0) # Call user made function
45
46 period = 2.0*np.pi/params[2] # Convert frequency to period
47
48 print("Period is " + str(period)) # Print answer
49
50 # Plot answer to see if it is reasonable
51 pyplot.title("Cepheid observations with fitted sine")
52 pyplot.xlabel("Time [days]")
53 pyplot.ylabel("Flux ratio [A.U.]")
54 pyplot.scatter(time, flux_ratio, label='Data') # Plot but with only points
55 pyplot.plot(t_dense, sine_array, label='Fitted function')
56 pyplot.legend(loc='best') # Add legend at automatic position
57 pyplot.grid() # Add grid
58 pyplot.show() # Print the plot
59

```

Om den anpassade funktionen ser rimlig ut så kan man tro på att den uppskattade perioden är nära sanningen, annars får man prova ändra gissningarna.



### Extra övning: Analysera sineFit

Öppna `userFunctions` och läs igenom funktionen `sineFit`. Den första delen kollar den inkommande datan och att varnar användaren om den givna datan är orimlig. Den andra delen är själva uträkningarna.

Återskapa uppgiften i labben genom att använda `curve_fit` direkt, utan `sineFit`.

### Extra övning 2: Anpassa en logaritmisk funktion till logaritmisk data

Ta datan från del 1 och använd `curve_fit` för att anpassa en logaritmisk kurva till den logaritmiska datan.