Laboration 6 - Tidsserieanalys

Trigonometri och avancerad kurvanpassning - för lärare

I den förra labben undersökte vi sambandet mellan period och ljusstyrka för cepheidvariabler. Denna gång ska vi titta närmare på hur man kan få fram datan vi använde förra gången.

I elevversionen av detta dokument finns här en snabb, grundläggande genomgång av trigonometri. Om ytterligare material behövs eller annat material vill användas istället finns det bland ordinarie matematik.

Astronomisk data

```
Time, Flux cephid, 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
```

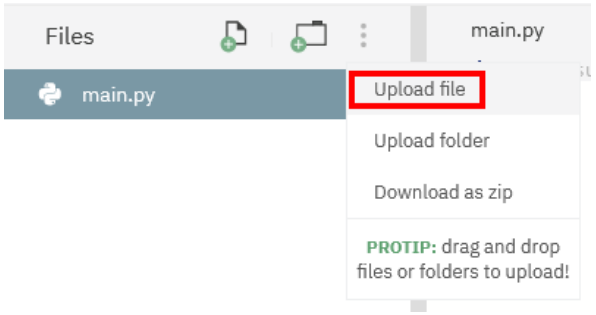
Att mäta perioden på en variabel stjärna låter kanske enkelt, men är inte så lätt i praktiken. Problemet består i att man sällan har möjlighet att göra tillräckligt många observationer för att se ett tydligt mönster. Vår data finner nu i filen *cepheid_obs.tx*t. Datan denna gång är tidpunkt, flux från 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.

E-post: cansat@au.se
Telefon: 070-000 90 56

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[1].



Öppna och läs filen med

- `f = open("cepheid_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

[1] https://repl.it/languages/python3

- `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()`

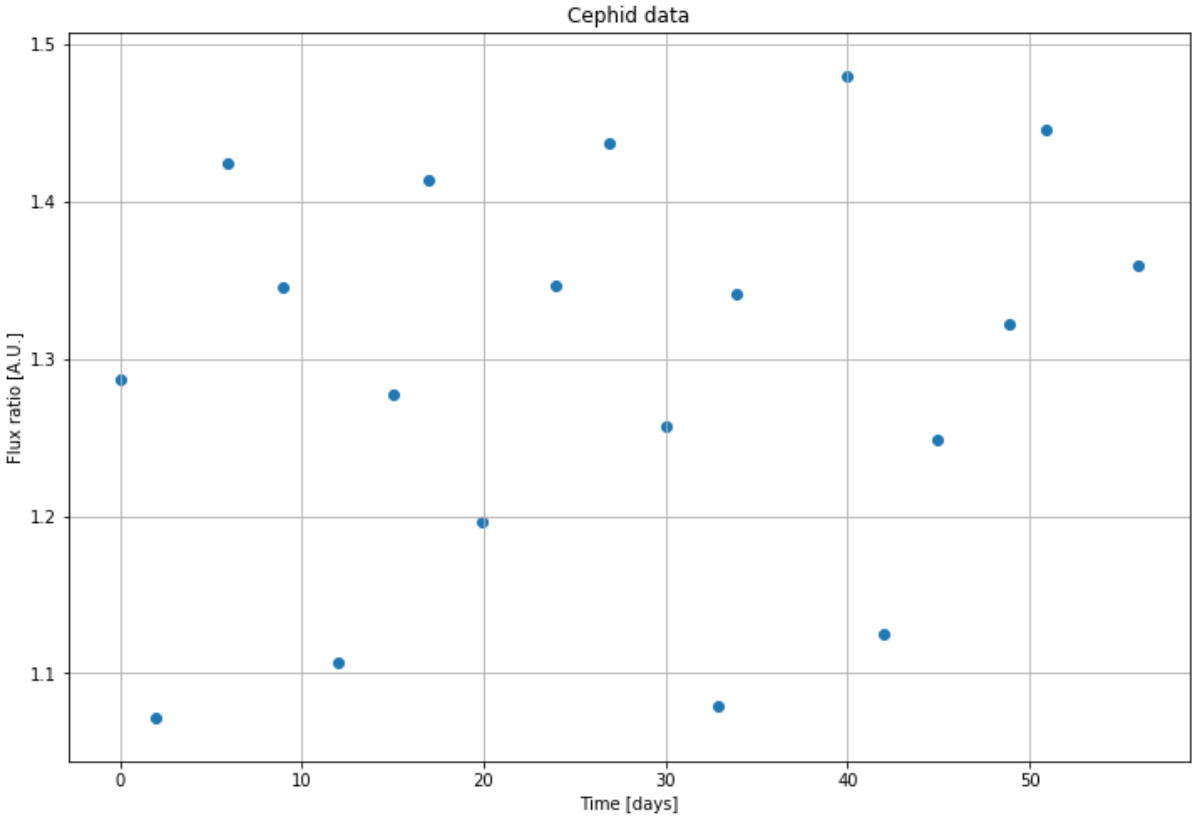
glöm inte

- `pyplot.show()`

i slutet.

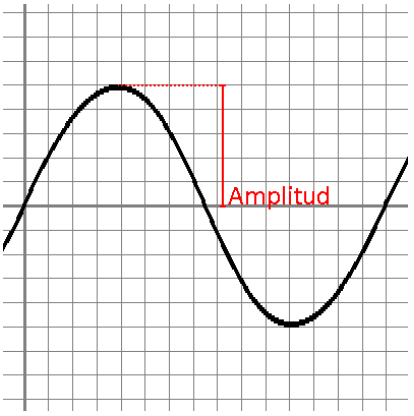
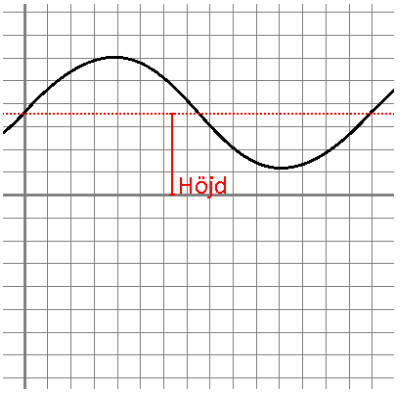
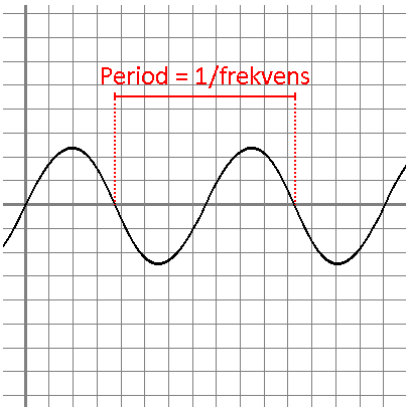
```python
import numpy as np
import matplotlib.pyplot as pyplot

time = [] # Reserve lists
flux_ceph = []
flux_ref = []

f = open("cepheid_obs.txt", "r") # Open the file for reading

line_list = f.readlines() # Read all line in the file

f.close() # Remember to close the file

line_list = line_list[1:] # Remove the header

for line in line_list:
    words = line.split(" ")
    time.append(float(words[0])) # Convert to float and add to list
    flux_ceph.append(float(words[1]))
    flux_ref.append(float(words[2]))

time = np.array(time) # Convert to arrays
flux_ceph = np.array(flux_ceph)
flux_ref = np.array(flux_ref)

flux_ratio = flux_ceph/flux_ref # Compute the flux ratio

pyplot.title("Cepheid observations")
pyplot.xlabel("Time [days]")
pyplot.ylabel("Flux ratio [A.U.]")
pyplot.scatter(time, flux_ratio) # Plot but with only points
pyplot.grid() # Add grid
pyplot.show() # Print the plot

```

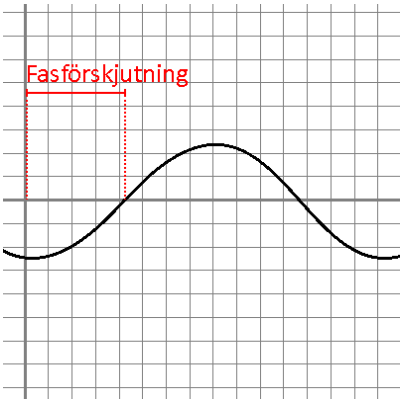
Om man skriver ut resultatet av divisionen över tiden får man följande figur.



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	
Svängnings-hastighet (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 cepheidvariablen. Avståndet från en topp till nästa (eller annan punkt till motsvarande en cykel senare) 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 2π 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

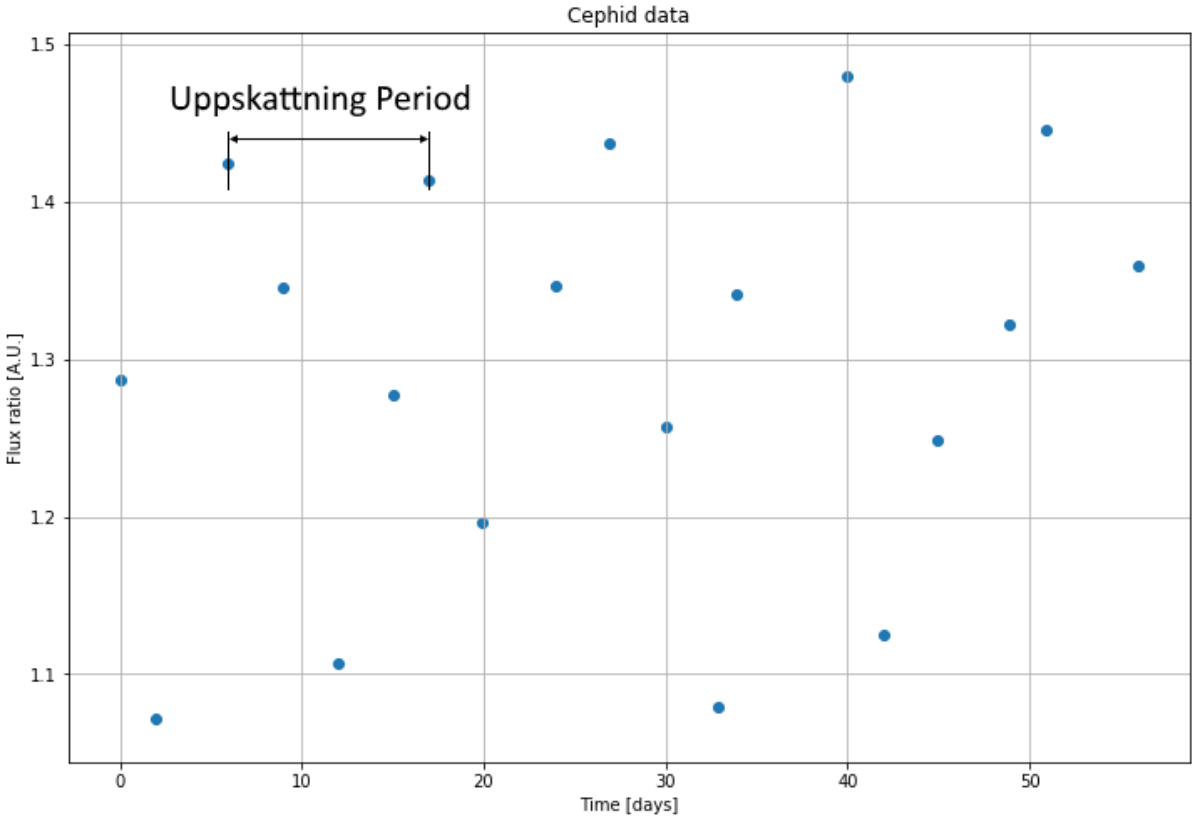
Funktionen som anpassar en godtycklig funktion till data är väldigt kraftfull men också svåranvänd. Vi tillhandahåller därför en något förenklad version som bara fungerar på sinuskurvor. Den finns i biblioteket `userFunctions` och importeras med namnet `sineFit` på samma sätt som i förra labben, med

- `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[2] 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.



[2] 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, Fasförskjutning,

- `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))`

Detta sätt att skriva ut siffror tar med alla decimaler, vilket är onödigt för oss. Vi kan använda metoden `.format` för att göra utskriften snyggare. Metoden anropas på en sträng med en (eller flera) platshållare i form av klammerparenteser. Innehållet i klammerparenteserna bestämmer hur datan i `.format` formateras. `:f` betyder att datan ska hanteras som en float. Utskriften blir då

`print("Period is {:f} days".format(period))`.

För mer detaljer se https://pyformat.info/.

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("Cephid 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 sinuskurvans 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 ritar 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,

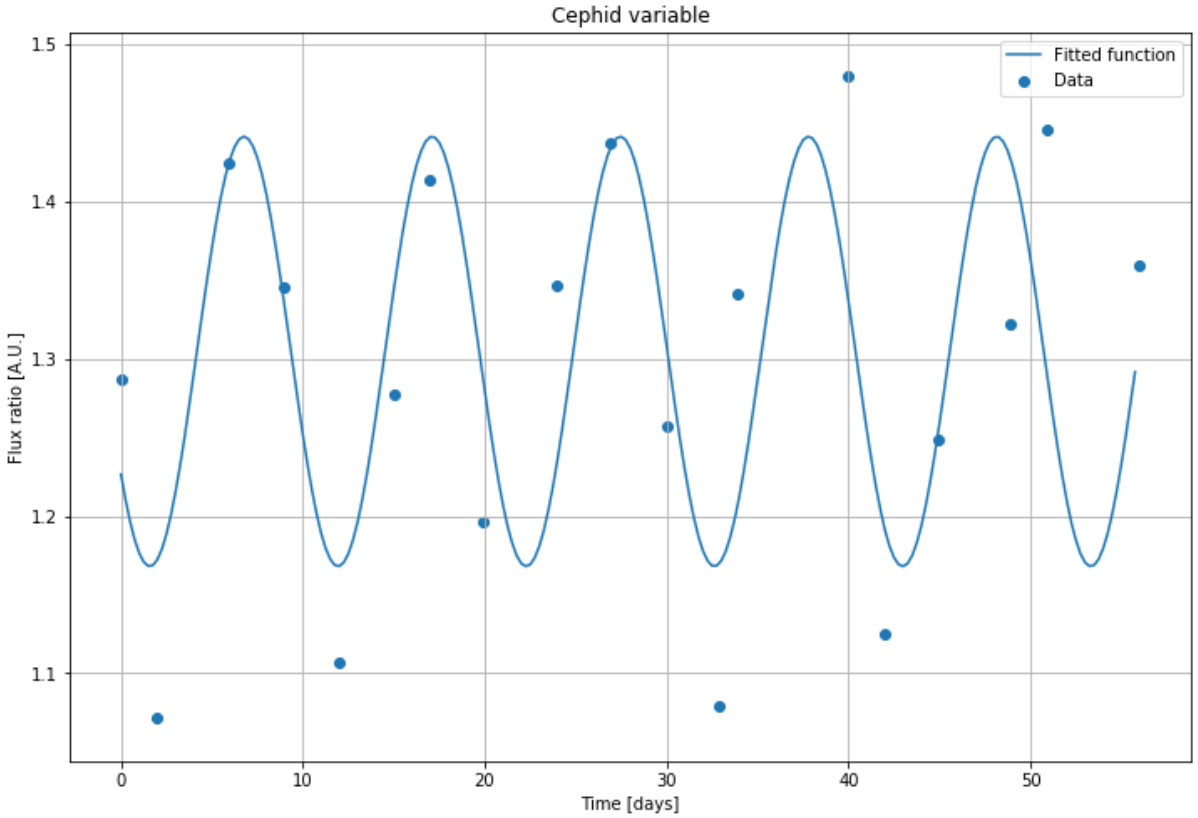
- `pyplot.legend(loc='best')`

avsluta med

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

```
#pyplot.title("Cepheid observations")
#pyplot.xlabel("Time [days]")
#pyplot.ylabel("Flux ratio [A.U.]")
#pyplot.scatter(time, flux_ratio) # Plot but with only points
#pyplot.grid() # Add grid
#pyplot.show() # Print the plot

guess_H = np.mean(flux_ratio) # Guess for height
guess_A = (max(flux_ratio) - min(flux_ratio))/2.0 # Guess for amplitude
guess_P = 12.0 # Only used to compute guess_S
guess_S = 2.0*np.pi/guess_P # Guess for the speed of swings
guess_F = 0.0 # Skip guessing the phase shift

p0=[guess_H, guess_A, guess_S, guess_F] # Arrange guesses in a list

params, t_dense, sine_array = sineFit(time, flux_ratio, p0) # Call user made function

period = 2.0*np.pi/params[2] # Convert frequency to period

print("Period is " + str(period)) # Print answer

# Plot answer to see if it is reasonable
pyplot.title("Cephid observations with fitted sine")
pyplot.xlabel("Time [days]")
pyplot.ylabel("Flux ratio [A.U.]")
pyplot.scatter(time, flux_ratio, label='Data') # Plot but with only points
pyplot.plot(t_dense, sine_array, label='Fitted function')
pyplot.legend(loc='best') # Add legend at automatic position
pyplot.grid() # Add grid
pyplot.show() # Print the plot

```

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.`

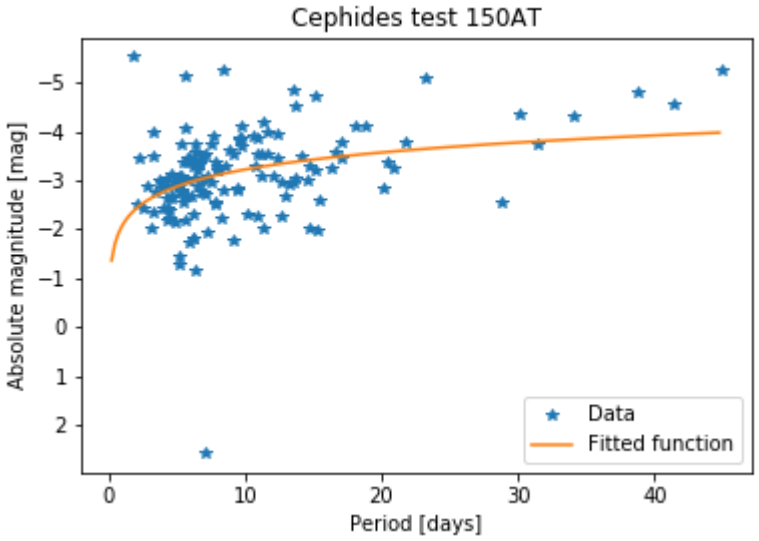
Extra övning 2: Anpassa en logaritmisk funktion till logaritmisk data

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

Lösningsförslag:

main.py saved

```
import numpy as np # Needed for fitting the data
import matplotlib.pyplot as plt
from scipy import optimize

name = "data_cephider.txt" # The name of the file
period_list = [] # Create an empty list to store values in
parallax_list = []
mag_list = []
ext_list = [] # include extinction, extra task
t_list = []

f = open(name, "r") # Open the file for reading

temp_list = f.readlines()

f.close() # Remember to close the file

temp_list = temp_list[2:-1]

drop = 0 # Keep track of how many entries were discarded

for line in temp_list:
    words = line.split("|")
    if float(words[2]) > 0 and words[5][0] != ' ':  # Check for bad data
        period_list.append(float(words[1]))
        parallax_list.append(float(words[2]))
        mag_list.append(float(words[4]))
        t_list.append(float(words[5]))
    else:  # If data is bad, discard
        drop += 1

print("Discarded {:d} results from bad parallax data or missing temp data".format(drop)
)

mag_array = np.array(mag_list) # convert to an array
parallax_array = np.array(parallax_list)
ext_array = np.array(ext_list)

pc_array = 1./(0.001 * parallax_array) # Convert parallax to parsec
m_abs_list = mag_array - 5. * np.log10(pc_array) + 5.

def test_funktion(x, H, A): # Function with unknown parameters to fit to data
    y = H + A * np.log(x)
    return y

guess_H = 0
guess_A = -0.4*(max(m_abs_list) - min(m_abs_list))/2.0
params, params_covariance = optimize.curve_fit(test_funktion, period_list, m_abs_list,
p0=[guess_H, guess_A])
print(params)
t_dense = np.arange(0.25, max(period_list), 0.25) # Get an array of x-data to plot the
sine curve on
log_array = test_funktion(t_dense, params[0], params[1]) # y-data of fitted curve

plt.title("Cephides test 150AT")
plt.xlabel("Period [days]")
plt.ylabel("Absolute magnitude [mag]")
plt.gca().invert_yaxis()
plt.plot(period_list, m_abs_list, '*')
plt.savefig("lum.png")  # Needed on repl.it
plt.show() # Print the plot

plt.clf() # Close first figure to draw the second correctly

plt.title("Cephides test 150AT")
plt.xlabel("Period [days]")
plt.ylabel("Absolute magnitude [mag]")
plt.gca().invert_yaxis()
plt.plot(period_list, m_abs_list, '*', label='Data') # Plot but with only points
plt.plot(t_dense, log_array, label='Fitted function')
plt.legend(loc='best') # Add legend at automatic position
plt.savefig("trend.png")  # Needed on repl.it
plt.show() # Print the plot
```



Gissningarna är svåra att komma fram till. Ett sätt att visualisera gissningarna och göra det lättare att hitta rätt är att rita ut test-funktionen med gissningarna innan de ändras av curve_fit, `pyplot.plot(t_dense, test_funktion(t_dense, guess_H, guess_A))`.