Laboration 5 - Databehandling

Cepheidvariabler och kurvanpassning - för lärare

I princip allt praktiskt vetenskapligt arbete (inklusive CanSat) går ut på att samla in data och analysera den. Databehandling är ett mycket stort område. I denna labb kommer vi beröra en liten del av det, att hitta trender i data. Om man kan hitta en trend kan man visa hur en storhet beror av en annan, t.ex. hur lufttryck beror av höjd.

Vi kommer att analysera data från så kallade cepheidvariabler. En cepheidvariabel är en typ av variabel stjärna som varierar i ljusstyrka på ett regelbundet sätt. Vi ska undersöka sambandet mellan en cepheidvariabels period och ljusstyrka. Till grund för dessa mätningar har vi avståndsmätningar gjorda med parallax, vilket man kan göra utan att behöva veta något om stjärnan.

Intro till Python

För att manipulera och analysera data vill vi använda programmering men det vi har gjort hittills har varit anpassat för att styra elektroniska kretsar, något mer mångsidigt behövs. Vi kommer använda ett språk som heter Python. Python är ett av de mest använda programmeringsspråken och kan användas till allting från enkla beräkningar och automation till avancerat vetenskapligt arbete. Språkets största styrkor är att det är enkelt att lära sig och att det finns ett mycket stort antal bibliotek till språket vilket kan göra komplicerade uppgifter mycket enklare.

Python har många likheter med C som vi har använt hittills. Precis som med programmering av Arduinon så rekommenderas det att ha koll på referenserna. För Python i allmänhet finns den officiella dokumentationen[1] och w3schools Python-tutorial[2]. För NumPy (som vi kommer använda senare) finns dess officiella dokumentation[3].

Om ni aldrig har använt Python förut kan man använda sidan Repl.it[4] gratis utan att behöva installera något eller registrera konto. I den här labbinstruktionen utgår från att Repl.it används.

[1] https://docs.python.org/3/reference/index.html
[2] https://www.w3schools.com/python/default.asp
[3] https://docs.scipy.org/doc/numpy/reference/index.html
[4] https://repl.it/languages/python3

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

I Python behöver man inte skriva semikolon efter varje rad. Man behöver inte heller skriva klammerparenteser efter funktioner, den funktionen uppfylls av indrag (skrivs med tab). Detta betyder att i Python är det mycket viktigt att indragen är korrekta eftersom det är det som avgör flödet i koden till skillnad från C där det bara gjorde koden lite lättare att läsa.

Variabler definieras på ett liknande sätt som i C. Skillnaden är att man inte behöver ange datatypen i Python, den klurar ut det själv. Funktioner fungerar också som i C. För att anropa en funktion skriver man dess namn följt av en parentes med eventuella argument, separerade med kommatecken om det är fler än ett. Kommentarer finns också och markeras med ett `#`.

Ett enkelt exempel är ett kort program som skriver ut texten Hello world! I Python skrivs det helt enkelt

- `print("Hello World!")`

Citationstecknet behövs för att markera att det är text och inte ett variabelnamn, samma som i C. Funktionen `print()` skriver ut argumentet i konsolen, liknande `Serial.println()`. Notera att vi inte skriver detta i någon funktion till skillnad från när vi arbetade i C. Man kan skapa funktioner i Python men man måste inte. Funktioner definieras med `def namn():` och sedan koden inom indrag.

Precis som i C kan man inkudera bibliotek i Python. Detta görs med kommandot

- `import namn`

man kan även utöka kommandot och ge det importerade biblioteket ett "smeknamn" genom att skriva

- `import namn as annatNamn`

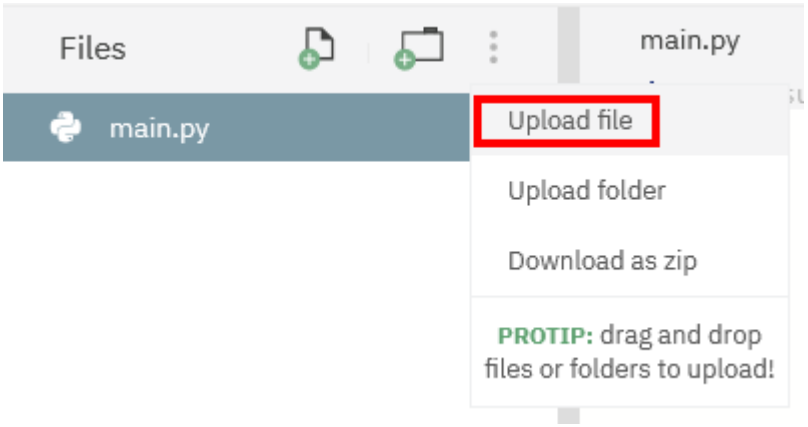
Detta är väldigt användbart för att förkorta bibliotekens namn, vilket är smidigt om man behöver anropa det många gånger i koden.

Intro databehandling

Vi kommer börja med att experimentera på en fil med data. Datan finns i filen `exemple.txt` och ser ut som på bilden.

```
923.4:310.0:621.7:038.6
068.6:157.2:746.4:886.4
072.1:594.3:212.1:799.4
287.8:938.4:518.6:248.4
```

Denna "data" består i 16 slumpmässiga decimaltal fördelat på 4 rader. Skiljetecknet mellan olika tal är ett kolon, :. Det första steget är att läsa in filen med data i Python. Filen ska ligga i rätt mapp, vilket är samma mapp som du sparar ner din python kod. Om man använder repl.it behöver man ladda upp filen.



I Python öppnar man en fil med kommandot `f = open(namn, läge)` där *namn* är namnet på filen som ska öppnas, *läge* är vilket läge vi vill öppna filen i dvs för att läsa ur filen eller för att skriva och f är namnet på den variabel vi använder till att förvara filen i. Vi vill läsa filen så vi använder läge `"r"` vilket gör kommandot till

- `f = open("example.txt", "r")`

Funktionen `readlines()` läser en fil och svarar med alla rader i filen. `readlines()` finns inom filobjektet som skapas av `open()` och kan därför bara användas på sådana objekt. En funktion som finns "inuti" ett objekt på detta vis kallas för en *metod* och de anropas genom att man skriver namnet på variablen man vill använda metoden på följt av en punkt och metodens namn, `f.readlines()`. Vi vill använda `readlines()` på f och spara resultatet i en lista som vi kallar `line_list`, så vi skriver

- `line_list = f.readlines()`

För att stänga filen skriver vi

- `f.close()`.

Det är viktigt att stänga filer som man öppnar.

```
1   f = open("example.txt", "r") # Open file for reading
2
3   line_list = f.readlines() # Save all lines to line_list
4
5   f.close() # Remember to close the file
6
```

Vi har nu en lista där varje objekt är en rad från filen. En lista i Python är en typ av variabel som innehåller flera värden i följd. Om en lista definieras explicit görs det med hakparenteser, t.ex.

- `myList = [1,2,3].`

Man kan även skapa tomma listor, t.ex. om man vill fylla i dom senare i en loop,

- `myEmptyList = [].`

Man kan anropa ett objekt (kallat element) i en lista genom att skriva listans namn följt av hakparenteser med elementets position i listan. Kom ihåg att programmeringsspråk nästan alltid räknar det första elementet som noll. t.ex.

- `one = myList[0]`

gör att variabeln one får värdet 1. Man kan skapa en lista genom att ta en del av en annan lista på liknande sätt. Om man skriver en två positioner med ett kolon emellan så betyder det ett spann, t.ex.

- `one_and_two = myList[0:1]`

gör att `one_and_two` blir en lista med värdena 1 och 2. Om vi bara skriver ett av värdena och lämnar den andra sidan tom tolkas det som att vi bara anger ett start- eller stoppvärde, dvs. hela resten av listan tas med. Det finns en metod som används på listor som du borde veta om, det är `.append()`. `.append()` lägger till det man skickar den sist i listan den sitter på, t.ex.

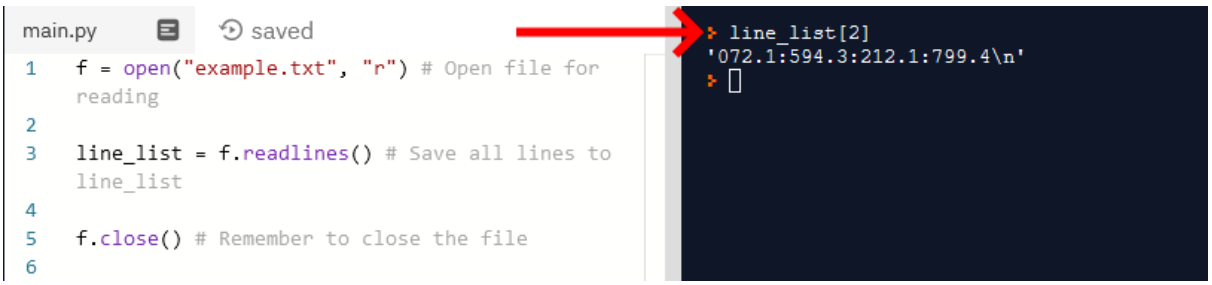
- `myList.append(4)`

gör att `myList` innehåller 1,2,3,4.

Testa skriva ut några rader från `line_list` genom att skriva

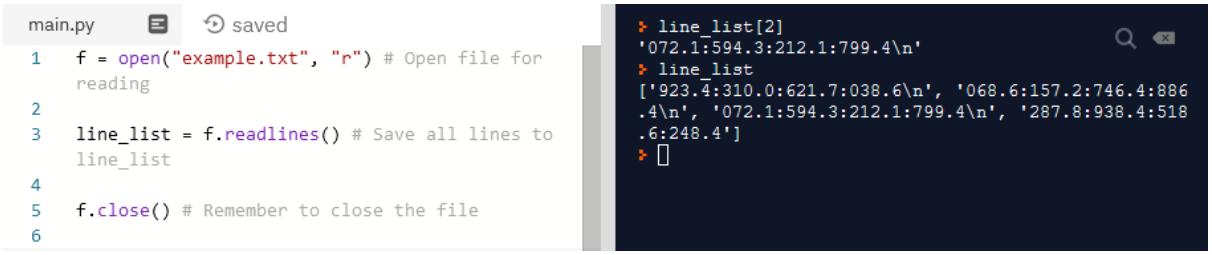
- line_list[2]

i konsolen. Det är en bra idé att få koden att skriva ut vad den gör genom att använda `print(variable)` på ställen i koden där man vill se vad som händer. Detta kommer inte ändra hur koden fungerar, bara göra dess funktion mer synlig.



för att se hela listan, skriv

- `line_list`

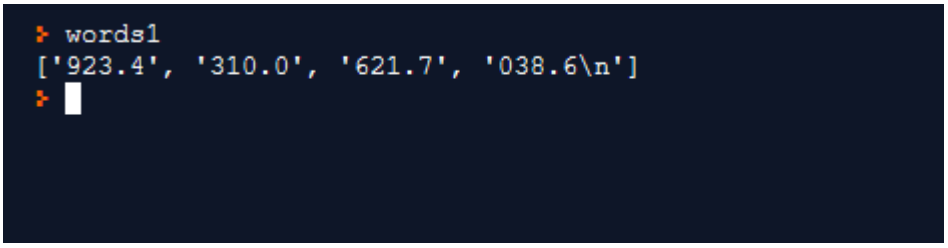


Nu kan vi se samma data i textfilen som programmet skriver ut. Datan från *`example.txt`* är sparad som en lista med textsträngar i `line_list`. , en för varje rad. Vi ser att \n betyder ny rad.

Nu vill vi organisera om datan så att den första siffran på varje rad hamnar i en lista, den andra siffran i en annan lista osv. Vi kommer också behöva byta datatyp eftersom datan i textfilen är text och vi vill ha siffrorna som siffror. Det första vill vi göra för att nå dit är att separera varje rad till de enskilda värdena. För att dela upp en sträng vid vissa tecken använder vi metoden `.split()`. `.split()` används på strängar och delar upp strängen längs det tecken man ger den. Det tecken som separerar datan i vår fil är ett kolon. Vi definierar en ny variabel för att ta emot listan av "ord" som `.split()` ger. För att använda `split` på den första raden i `line_list` skriver vi

- `words1 = line_list[0].split(":")`

Skriver vi sedan ut words1 i konsolen ser vi hur det ser ut.



words1 är nu en lista med fyra strängar som förut var en sträng. Funktionen `float(data)` gör om *data* till en float, så vi kan ta ut det första värdet och göra det till en float med `float(words1[0])` och skriva in det i en ny lista. Vi skulle kunna göra samma sak för de tre andra raderna och sedan manuellt skriva in de fyra värdena i de fyra listorna i en nya lista, men det är väldigt mycket jobb. Med kod kan vi automatisera den processen. En for-loop är vad vi behöver.

> **Inforuta, kom ihåg:** En for-loop är en struktur som repeterar koden i sig ett visst antal gånger eller tills ett kriterium är uppfyllt.

Ett sätt att skriva en for-loop i Python påminner om det vi har sett tidigare,

- ```
  for i in range(int):
      kod
  ```

där *int* är en siffra. Denna kod kommer utföra koden *kod int* antal gånger. `range()` är en funktion som ger en serie från en siffra till en annan, om man anger bara en siffra så blir serien från 0 till den angivna siffran. i är en variabel som tar värdet av en siffra i serien per varv. Det här går att expandera. Variabeln behöver inte ta värden som är siffror och listan behöver inte vara en serie siffror. Om vi skriver

- ```
  for var in list:
      kod
  ```

kommer koden i for-loopen utföras för varje objekt i listan *list*. Koden i for-loopen körs så många gånger som det finns objekt i listan och *var* blir varje objekt i listan, ett i taget. För att illustrera, om man skriver

- ```
  siffer_lista = [1,9,4,7]
  ```
- ```
  for siffra in siffer_lista:
      print(siffra)
  ```

blir utskriften

1

9

4

7

Innan vi börjar skriva koden för att fylla i fyra listor ska vi skapa de fyra listorna. Skriv

- ```
  first_list = []
  ```
  ```

```
second_list = []
third_list = []
fourth_list = []
```

först i koden. Sedan skriver vi våran loop. Vi vill göra vissa saker för varje rad i `line_list`, vi vill dela upp raden i "ord", lägga till orden i listorna som är definierade längst upp i koden och omvandla orden till floats. Vi använder `.append()` för att lägga till varje "ord" i sin lista efter att den har blivit omvandlad till float. Vi skriver en for-loop så här

- ```
  for line in line_list:
      words = line.split(":")
      first_list.append(float(words[0]))
      second_list.append(float(words[1]))
      third_list.append(float(words[2]))
      fourth_list.append(float(words[3]))
  ```

```
1   first_list = [] # create empty lists
2   second_list = []
3   third_list = []
4   fourth_list = []
5
6   f = open("example.txt", "r") # Open file for reading
7
8   line_list = f.readlines() # Save all lines to line_list
9
10  f.close() # Remember to close the file
11
12  for line in line_list: # loop over the lines in line_list
13    words = line.split(":") # split the string
14    first_list.append(float(words[0])) # add word to the end of the list
15    second_list.append(float(words[1]))
16    third_list.append(float(words[2]))
17    fourth_list.append(float(words[3]))
18
```

Det borde räcka. Om du skriver ut first_list i konsolen borde du se en lista av floats som är de första från varje rad.

```
> first_list
[923.4, 68.6, 72.1, 287.8]
> >
```
```

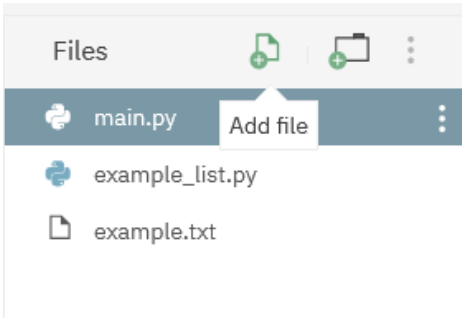
Relation mellan period och avstånd för cepheidvariabler

I filen *gaia_cepheider.txt* finns en tabell med data om ett antal cepheidvariabler. Detta är äkta data som kommer direkt ifrån ESAs rymdteleskop Gaia[5] via databastjänsten ARI's Gaia TAP Service[6]. Kolonnerna är Namn, period, parallax, uppskattat fel i parallax, observerad magnitud i G-bandet, effektiv temperatur och extinktion. För att undersöka sambandet mellan period och absolut ljusstyrka behöver vi period, parallax, magnitud och temperatur. Vissa stjärnor har negativa värden för parallax. Detta är orimligt och saknar fysiskt betydelse och kan därför ses som ett mätfel. Dessa värden behöver filtreras ut, vi kommer göra det senare i koden.

designation	pf	parallax	parallax_error	phot_g_mean_mag	teff_val	a_g_val
"Gaia DR2 4057701830728920064"	7.017529307010724	3.4314367740607445	0.2019658107192288	4.316964	5249.0	
"Gaia DR2 6058439910929477120"	3.342476897508827	1.7777437608620148	0.08731238219966553	5.2445936	6044.0	1.2997
"Gaia DR2 1820309639468685824"	8.386033915357054	0.6425529353755096	0.09315620122279228	5.718552	5076.0	
"Gaia DR2 5506374096132016512"	5.6929616525813325	1.2609953624454222	0.0642818738097271	5.445048	5790.3335	
"Gaia DR2 1857884212378132096"	2.217210945265626	1.6738045061961986	0.08914132808424952	5.440939	4707.25	
"Gaia DR2 4175017625462647168"	17.12484063933547	1.361059042016258	0.08344344352410059	5.5648613	3859.0	
"Gaia DR2 5873984023533350400"	5.2759671782252315	1.7449062341279333	0.34502242607837436	5.700312	5192.0	
"Gaia DR2 1964855904803120640"	3.3323345324660067	1.1505517896475126	0.06634840475810791	5.7184625	6198.0	
"Gaia DR2 5855852527008107008"	9.65890011040995	1.131409494710051	0.11669511285155915	5.9243245	5404.25	
"Gaia DR2 5338359442320395904"	38.71791440244922	0.4789725460644777	0.029341750317122366	6.7958097	5143.45	
"Gaia DR2 1870258975238302208"	16.390621155118538	0.8982672586263467	0.027007756189095483	6.9698887	5143.45	
"Gaia DR2 5855468247702904704"	7.510275510767402	1.0001819840417556	0.029112687842940146	6.257463	5076.4966	
"Gaia DR2 5835124087174043136"	9.753758609111381	1.061575091220469	0.04220963229268318	6.150488	5065.75	
"Gaia DR2 5823134325151372032"	6.3166264238077305	1.0752614679438075	0.030129764968007598	6.266616	5356.5	
"Gaia DR2 6054829806275577216"	6.732870416019966	1.1397475833324917	0.027632002427898993	6.3766994	5134.0	
"Gaia DR2 4092905375639902464"	6.0933262327518305	1.4600813301331412	0.044714486570567306	6.501302	4287.565	
"Gaia DR2 6060173364061645696"	4.68971493100182	1.0214829266427539	0.04481854435113779	9.266351		
"Gaia DR2 5868451040512196224"	5.622827508200463	0.4838172467245708	0.15447789694472336	6.4429145	5169.2354	
"Gaia DR2 5877533315817003648"	7.066260258518632	1.0148424692881757	0.031891770789733785	6.4461155	5100.3335	
"Gaia DR2 5824464493705913472"	3.389323187056138	1.4754483958055171	0.03711945461449232	6.415912	5220.0	
"Gaia DR2 5546476927338700416"	41.464114200882854	0.5844378748729412	0.026037014270696043	6.6014204	3955.0	
"Gaia DR2 5891675303053080704"	5.494569706156711	1.3396571859609512	0.04474311950096547	6.6745715	5210.0	
"Gaia DR2 1825621002188696448"	7.990139955058473	1.0529894731273517	0.038967296479367426	6.6911983	3975.0	0.728
"Gaia DR2 5339394082770287232"	7.720813253754169	0.7956404722729643	0.03458194424832424	6.569681	5500.67	
"Gaia DR2 4085919765884068736"	5.191731401385837	1.2484550307176254	0.04109278267302299	6.5938225	4419.25	
"Gaia DR2 5125243616130406 40"	8.377013305874744	1.29631304913415	0.03133304208251316	6.6373186	4113.202	
"Gaia DR2 2027951173435143680"	44.87620944377743	0.3728552994205654	0.030349340157764505	6.8699703	3986.0598	0.5307
"Gaia DR2 5523162573544337408"	20.497635428698278	0.5467438169841664	0.03168055353176288	7.956587	5095.6333	
"Gaia DR2 6026412893938675712"	14.597655937025346	1.130587582649599	0.0552264461216046	6.7184496	4919.75	
"Gaia DR2 2031776202613700480"	3.8459870643308443	1.169541947125911	0.05156865807706845	6.817692	5420.0	
"Gaia DR2 4066429066901946368"	2.7894091130718697	1.119032405819382	0.05268460297603898	6.8778343	4730.0	
"Gaia DR2 1744890980111145216"	6.464663189704652	1.041957404440374	0.06371154896837697	7.0238175	4642.0	
"Gaia DR2 5302258008774271488"	6.696948853426342	0.7603316846403194	0.02271110440530145	7.1692224	4861.3335	
"Gaia DR2 5351161399785606016"	18.873022302484596	0.5120823364649894	0.041380091581450944	7.3261256	4483.5	
"Gaia DR2 4703611143398494 72"	7.91560526872026	0.7822751522923913	0.039722031724278344	7.175379	4197.3765	

Spara den kod som har gjort hittills och börja på en ny fil. Om repl.it används så måste den kod som ska köras ligga i main.py, den gamla koden kan kopieras över till en ny fil och main.py återanvändas.

[5] https://www.esa.int/Science_Exploration/Space_Science/Gaia_overview
[6] http://gaia.ari.uni-heidelberg.de/tap.html



Först ska vi ladda upp filen med data från Gaia. Det görs på samma sätt som example.txt gjordes på. Den nya koden ska börja på samma sätt som den förra, med att öppna en textfil och läsa in raderna i en lista. Prova skriv ut några rader av listan för att se att den är läst som den ska.

```
1   f = open("gaia_cephider.txt", "r") # Open file for reading
2
3   line_list = f.readlines() # Save all lines to line_list
4
5   f.close() # Remember to close the file
6
```

```
> line_list[2]
'"Gaia DR2 4057701830728920064"|7.017529307010724 |3.4314367740607445 |0.2019658107192288 |
4.316964       |5249.0    |       \n'
> line_list[0]
'         designation          |       pf        |     parallax      |   parallax_error   |
phot_g_mean_mag|teff_val |a_g_val\n'
> line_list[100]
'"Gaia DR2 2055014277739104896"|5.955900846425911 |0.6972293343957737 |0.038827214095027554|
9.039626       |3938.5598|       \n'
>
```

De två första raderna av vår data bara tabellhuvud som vi inte vill ta med. Vi omdefinierar därför line_list som line_list fast utan de två första raderna. Vi skriver

- `line_list = line_list[2:]`.

Nu är vi redo att börja plocka ut data ur våra rader. Vi vill åstadkomma ett par saker. I slutet vill vi ha fyra listor med siffror och inga orimliga värden. För att göra det behöver vi plocka ut datan vi är intresserade av, omvandla den från text till siffror och kolla om värdena är rimliga. Vi gör allt detta i en for-loop som tidigare.

För att plocka ut de tre värden vi är intresserade av från varje rad använder vi `.split()` igen. I vår data separeras varje typ av data med ett lodrätt streck |. Vi definierar en ny variabel för att ta emot listan av "ord" som `.split()` ger. Vi skriver

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

Nu vill vi fylla på fyra olika listor med rätt av data. Börja med att skapa fyra tomma listor med beskrivande namn i början av koden. De värden vill vi lagra i listorna vill vi omvandla från text till siffror med `float()` som tidigare. Vi vill använda `float()` eftersom datan vi hanterar är decimaltal och vi ska använda den i beräkningar senare. Funktionen tar hand om eventuella extra mellanslag och liknande automatiskt. Vi använder `.append()` för att lägga till varje "ord" i sin lista igen. Notera att denna kod inte kommer fungera eftersom vår data fortfarande innehåller värden som programmet inte kan hantera, specifikt så innehåller listan på temperaturer flera tomma poster. Dessa orsakar fel när man försöker konvertera dom till floats.

```
1   period_list = []  # Create an empty list to store values in
2   parallax_list = []
3   mag_list = []
4   t_list = []
5
6   f = open("gaia_cephider.txt", "r") # Open file for reading
7
8   line_list = f.readlines() # Save all lines to line_list
9
10  f.close() # Remember to close the file
11
12  line_list = line_list[2:] # Remove the header
13
14  for line in line_list: # loop over the lines in line_list
15      words = line.split("|") # split the string
16      period_list.append(float(words[1])) # add word to the end of the list
17      parallax_list.append(float(words[2]))
18      mag_list.append(float(words[4]))
19      t_list.append(float(words[5]))
20
```

Vi vill kolla om radens data är bra och ska användas eller dålig och ska förkastas. Det är två saker vi vill kolla, att värdet för parallaxen är rimligt och att det finns ett värde för temperaturen för den raden. Båda kriterierna måste uppfyllas för att en rad ska accepteras. Detta beslut fattas med en if-sats.

Inforuta, kom ihåg: En if-sats är en struktur som utvärderar ett kriterium

och kör en bit kod endast om kriteriet är sant.

if-satser i Python skrivs nästan exakt som i C fast man behöver inga parenteser och ett kolon efter villkoret, och indrag används för att avgöra vilken kod som ligger i den. Det bästa stället att lägga if-satsen på är precis efter `.split()`. Villkoret för att kolla att parallaxen är ok borde vara parallax > 0 (`float(words[2]) > 0`), villkoret för temperaturen kan vara att kolla att första tecknet i temperatur-”ordet” inte är ett mellanslag `(words[5][0] != ' '`),

- `if float(words[2]) > 0 and words[5][0] != ' ':`

Strängar i Python kan öppnas på liknande sätt som listor dvs man kan plocka ut en viss bokstav genom att skriva dess index efter strängens namn, som om strängen var en lista med bokstäver. Om det är så, som i vårt fall, att strängen i sig är ett element i en lista så skriver man bara det andra indexet efter det första. För att kombinera två villkor till ett använder man de logiska operatorerna `and` eller `or`. `and` gör att villkoret blir sant om båda delarna är sanna, `or` gör att villkoret blir sant om minst ett av villkoren är sanna. Vi vill att båda ska vara sanna så vi använder `and`.

Det är en bra idé att räkna hur många rader som förkastas. Detta görs enklast med en variabel som går upp med 1 varje gång if-satsen inte godkänner villkoret. Vi skapar en variabel som vi kallar `bad_lines` som startar på 0. Varje gång if-satsen inte är sann ökar vi den med 1.

- `else:`

```
        bad_lines = bad_lines + 1
```

```
period_list = []  # Create an empty list to store values in
parallax_list = []
mag_list = []
t_list = []

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

line_list = f.readlines() # Save all lines to line_list

f.close() # Remember to close the file

line_list = line_list[2:] # Remove the header
bad_lines = 0 # count how many lines are discarded

for line in line_list: # loop over the lines in line_list
    words = line.split("|") # split the string
    if float(words[2]) > 0 and words[5][0] != ' ':  # Check for bad data
        period_list.append(float(words[1])) # add word to the end of the list
        parallax_list.append(float(words[2]))
        mag_list.append(float(words[4]))
        t_list.append(float(words[5]))
    else:  # If data is bad, discard
        bad_lines = bad_lines + 1

```

De kommande funktionerna vi vill använda är del i biblioteket Numpy. Numpy är ett stort och mycket användbart bibliotek med många matematiska funktioner. Vi inkluderar det genom att skriva

- `import numpy as np`

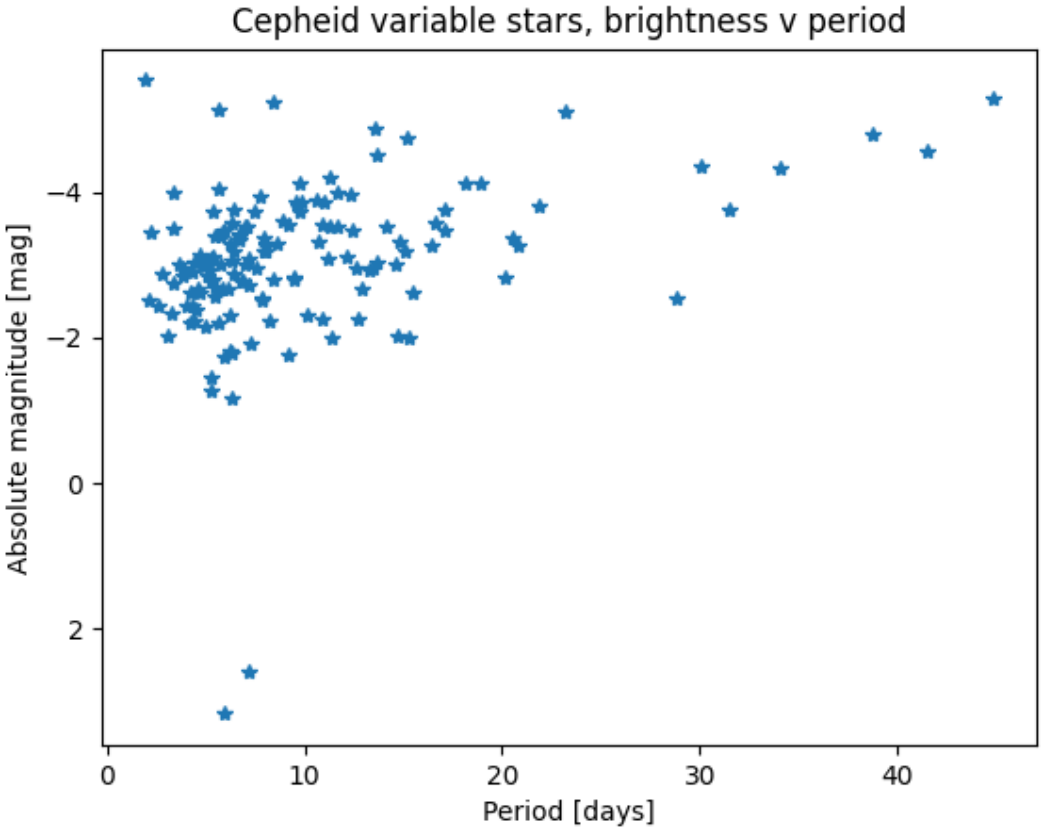
längst upp i koden. Nu kan vi använda funktioner från numpy genom att skriva `np.`*`namn`* där *namn* är namnet på funktionen. Det första vi vill göra är att göra om listorna på magnituder och parallax till array-er. Det kommer göra att vi kan enkelt göra beräkningar på alla värden samtidigt istället för att göra det för varje element enskilt. Ett array kan enkelt skapas från en listan genom kommandot `np.array()`. Skriv listan som argument och definiera resultatet som en ny variabel.

- ```
  mag_array = np.array(mag_list)
  parallax_array = np.array(parallax_list)
  ```

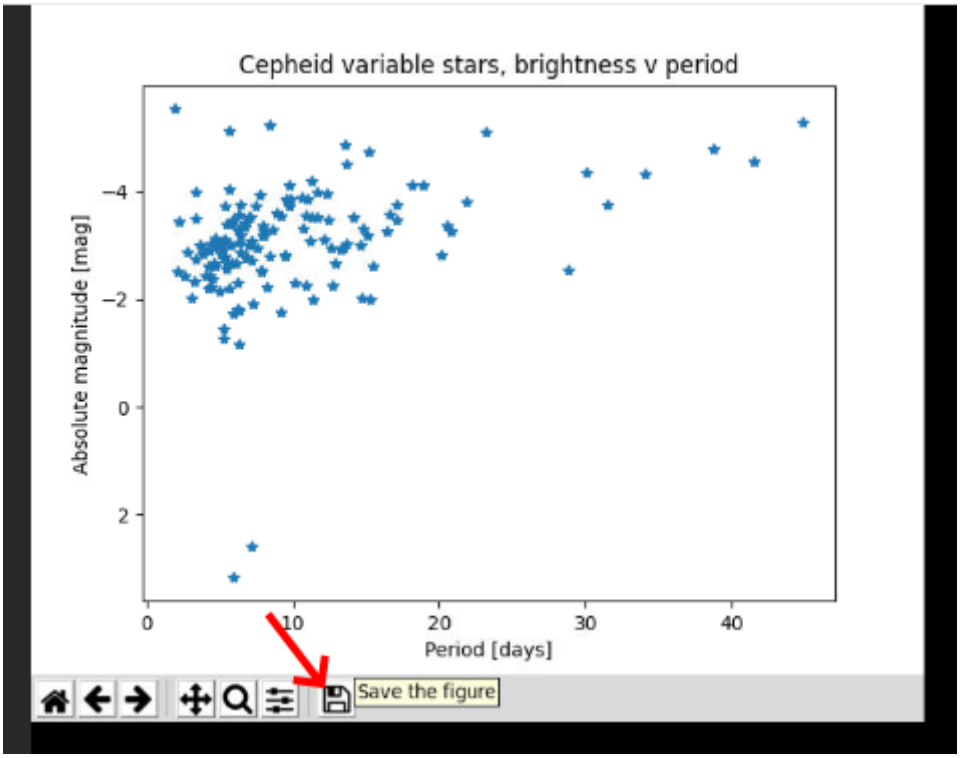
Nu ska vi göra beräkningarna för att räkna om parallax till avstånd och observerad magnitud till absolut magnitud. Formeln för avstånd från parallax är helt enkelt

$$d = \frac{1}{p}$$
  ```

där *p* är parallax i bågsekunder och *d* är avstånd i parsek. Notera att i vår data är parallax angett i tusendels bågsekunder så ni behöver kompensera för det.

- `pc_array = 1. / (0.001 * parallax_array)`

Formeln för att absolut magnitud från observerad magnitud och avstånd är

$$M_{abs} = m_{obs} - 5 * log_{10}(dist_{pc}) + 5$$

där *M_abs* är absolut magnitud, *m_obs* är observerad magnitud och *dist_pc* är avståndet i parsek. Logaritmer skrivs enklast som `np.log10(var)`.

- `m_abs_list = mag_array - 5. * np.log10(pc_array) + 5.`

Att skriva en punkt efter en siffra gör att den definieras som en float istället för en int. Att skriva `1.` är samma sak som att skriva `1.0`.

```
1   import numpy as np
2
3   period_list = []  # Create an empty list to store values in
4   parallax_list = []
5   mag_list = []
6   t_list = []
7
8   f = open("gaia_cephider.txt", "r") # Open file for reading
9
10  line_list = f.readlines() # Save all lines to line_list
11
12  f.close() # Remember to close the file
13
14  line_list = line_list[2:] # Remove the header
15  bad_lines = 0 # count how many lines are discarded
16
17  for line in line_list: # loop over the lines in line_list
18      words = line.split("|") # split the string
19      if float(words[2]) > 0 and words[5][0] != ' ':  # Check for bad data
20          period_list.append(float(words[1])) # add word to the end of the list
21          parallax_list.append(float(words[2]))
22          mag_list.append(float(words[4]))
23          t_list.append(float(words[5]))
24      else:  # If data is bad, discard
25          bad_lines = bad_lines + 1
26
27  mag_array = np.array(mag_list)  # convert to an array
28  parallax_array = np.array(parallax_list)
29
30  pc_array = 1. / (0.001 * parallax_array)  # Convert parallax to parsec
31  # Convert observed magnitude to absolte magnitude
32  m_abs_list = mag_array - 5. * np.log10(pc_array) + 5.
```

Rita grafer

Det vi har uppnått nu är ett dataset med x- och y-koordinater. Y-koordinaterna är magnituderna och x-koordinaterna är perioderna. Vi kan få Python att rita upp en graf med datan åt oss. Till det behöver vi ett till bibliotek som kallas matplotlib. Det är ett väldigt stort bibliotek och vi behöver bara en liten del som heter pyplot. För att göra det enklare för oss kan vi säga åt Python att bara hämta pyplot från matplotlib genom att skriva

- `from matplotlib import pyplot`

Vi kan även ge pyplot ett smeknamn genom att lägga till ett `as` *namn* efter det om vi vill. För att skriva en graf anger vi de egenskaper vi vill ha följt av ett kommando för att visa grafen.
För att lägga in data i en graf använder man

- `pyplot.plot(`*`x, y`*`)`

Vi skriver in vår data för x- och y-koordinaterna. Eftersom våran data inte är sorterad vill vi inte rita linjer mellan datapunkter. För att göra det använder vi ett tredje argument i plot-kommandot som talar om vilken typ av markör man vill ha. Vi skriver `'*'`. Man vill oftast ange en titel och skriva rubriker och enheter på axlarna, det gör vi genom att skriva

- ```
  pyplot.title("Cepheid variable stars, brightness v period")
  pyplot.xlabel("Period [days]")
  pyplot.ylabel("Absolute magnitude [mag]")
  ```

Eftersom magnitud är en "omvänd" skala där mer negativa tal motsvarar ljusstarkare stjärnor behöver vi vända på y-axeln. I moderna versioner av matplotlib är det enkelt, man lägger bara in kommandot

- `pyplot.gca().invert_yaxis()`

Slutligen säger man åt Python att visa grafen på skärmen genom kommandot

- `pyplot.show()`
  ```

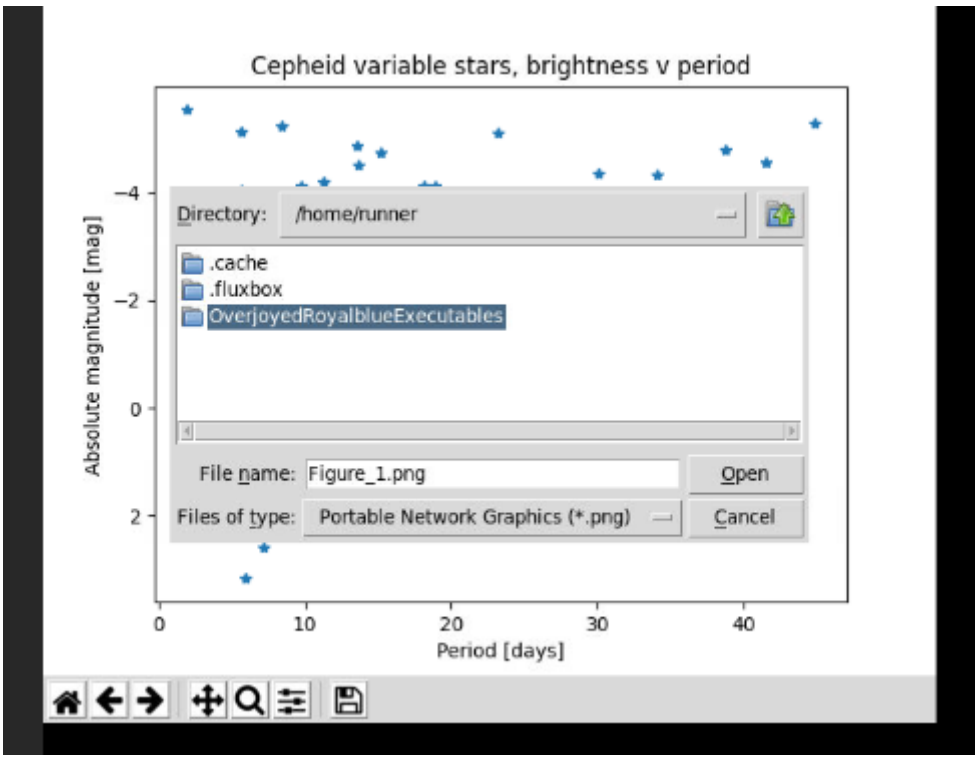
```python
import numpy as np
import matplotlib.pyplot as pyplot

period_list = []  # Create an empty list to store values in
parallax_list = []
mag_list = []
t_list = []

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

line_list = f.readlines() # Save all lines to line_list

f.close() # Remember to close the file

line_list = line_list[2:] # Remove the header
bad_lines = 0 # count how many lines are discarded

for line in line_list: # loop over the lines in line_list
    words = line.split("|") # split the string
    if float(words[2]) > 0 and words[5][0] != ' ':  # Check for bad data
        period_list.append(float(words[1])) # add word to the end of the list
        parallax_list.append(float(words[2]))
        mag_list.append(float(words[4]))
        t_list.append(float(words[5]))
    else:  # If data is bad, discard
        bad_lines = bad_lines + 1

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

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

pyplot.title("Cepheid variable stars, brightness v period")
pyplot.xlabel("Period [days]")
pyplot.ylabel("Absolute magnitude [mag]")
pyplot.gca().invert_yaxis()  # Flip the y-axis
pyplot.plot(period_list, m_abs_list, '*')
pyplot.show()  # Print the plot
```



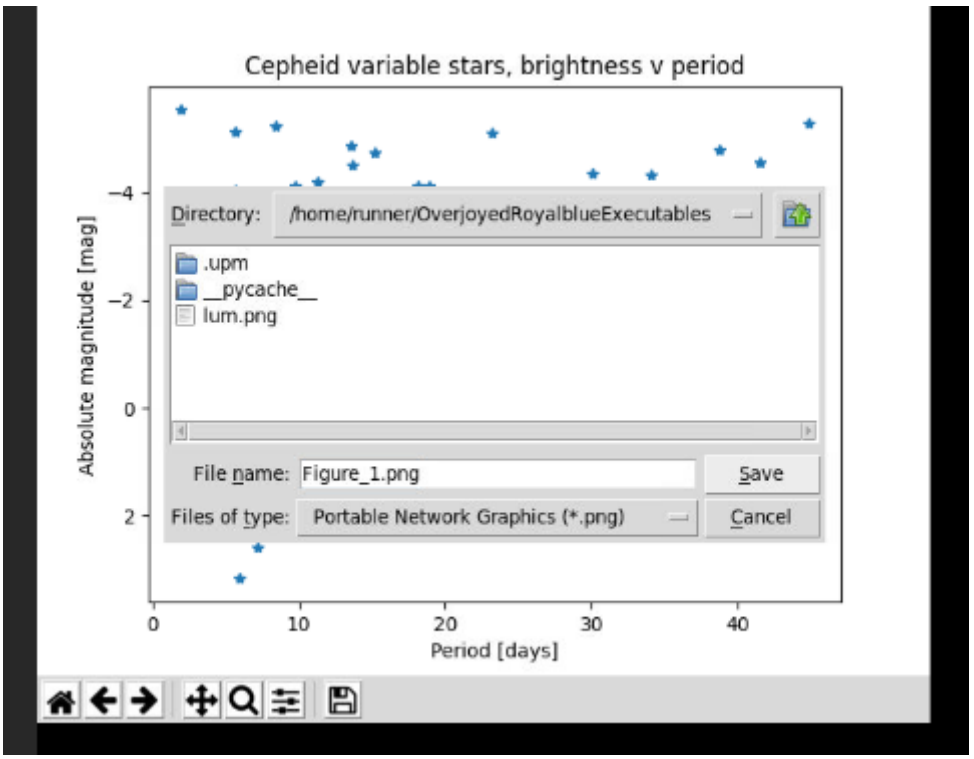
Om man använder Repl.it och vill spara bilder gör man såhär. Klicka på knappen för att spara bilden.



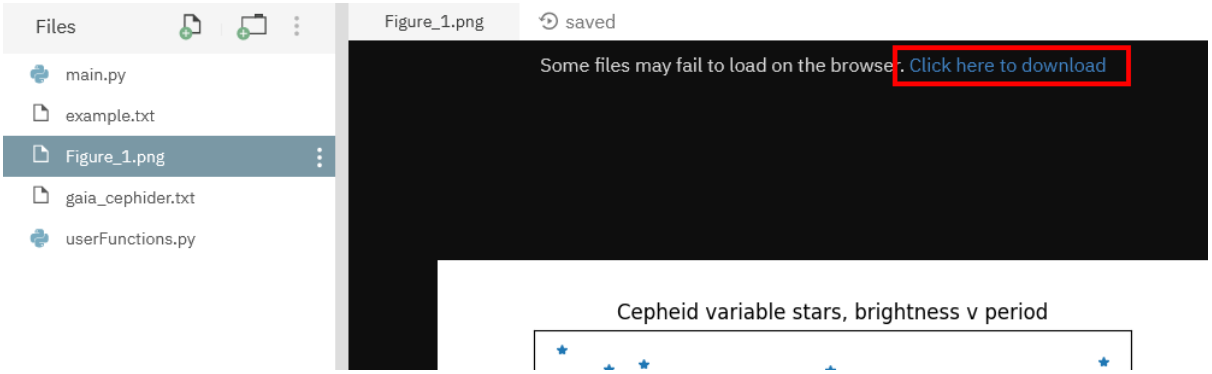
Öppna mappen med ditt slumpade namn (den enda mappen som inte börjar med en punkt).



Spara bilden.



Klicka på bilden som har dykt upp i listan på filer och klicka på **Click here to download.**



Nu har vi visualiserat vår data vilket är bra men ärligt talat inte så hjälpsamt än. Det finns två saker vi kan göra som kommer leda oss till en slutsatts. De är att göra om datan till att vara linjär och att anpassa en trendlinje till vår data. Magnitud är en logaritmisk enhet för ljusstyrka. Den linjära enheten för ljusstyrka heter luminositet.

Astronomiruta: I normala fall räknas luminositet fram med hjälp av så kallad bolometrisk magnitud vilket är magnituden av en stjärna i alla färger samtidigt. Vi har inte tillgång till några bolometriska magnituder, vi har bara magnituderna i G-bandet vilket ligger helt i infrarött. För att komma fram till en luminositet från det behöver man en bolometrisk korrektion. Eftersom stjärnor är ungefärliga svartkroppar kan man räkna fram ungefär hur starkt de borde lysa om man vet temperaturen och ljusstyrkan i någon viss del av spektrat. Det går att räkna fram en god approximation av den bolometriska korrektionen utifrån temperaturen men det är lite besvärligt, så ni får en funktion som gör det åt er.

```python
import numpy as np
import matplotlib.pyplot as pyplot
from userFunctions import bolometricCorrection # user made lib

period_list = []  # Create an empty list to store values in
parallax_list = []
mag_list = []
t_list = []

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

line_list = f.readlines() # Save all lines to line_list

f.close() # Remember to close the file

line_list = line_list[2:] # Remove the header
bad_lines = 0 # count how many lines are discarded

for line in line_list: # loop over the lines in line_list
    words = line.split("|") # split the string
    if float(words[2]) > 0 and words[5][0] != ' ':  # Check for bad data
        period_list.append(float(words[1])) # add word to the end of the list
        parallax_list.append(float(words[2]))
        mag_list.append(float(words[4]))
        t_list.append(float(words[5]))
    else:  # If data is bad, discard
        bad_lines = bad_lines + 1

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

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

lum_list = []
for i in range(len(t_list)):
  # Compute the luminosity (linear unit) for each star
  lum_list.append(np.exp(-0.4*(m_abs_list[i] + bolometricCorrection(t_list[i]) + 4.74)))

pyplot.title("Cepheid variable stars, brightness v period")
pyplot.xlabel("Period [days]")
```

Eftersom det är besvärligt att räkna fram luminositet utan all data får ni en färdig funktion som gör det. Den tar effektiv temperatur som argument och ger den bolometriska korrektionen i magnituder som svar. Man kan importera funktioner från andra Python-filer genom att importera dom på samma sätt man importerar bibliotek. Ladda upp filen userFunctions.py på samma sätt som datafilerna och skriv

- `from userFunctions import bolometricCorrection`

längst upp i koden så kan ni använda `bolometricCorrection()` som om det var en inbyggd funktion.

Formeln för luminositet från absolut magnitud i en viss färg, bolometrisk korrektion i samma färg och en referensmagnitud är

$$L = Exp(-0,4 * (M_{abs} + BC(T) - M_{ref}))$$

där *Exp* är exponentialfunktionen dvs *e* upphöjt till det inom parentesen (`np.exp(var)` i Python), *M_abs* är den absoluta magnituden vi kom fram till tidigare, *BC* är den bolometriska korrektionen som beror av temperaturen, *M_ref* = 4,74 eftersom det är solens absoluta bolometriska magnitud vilket är vår referens och *L* är luminositeten i sol-enheter dvs hur stor andel av solens luminositet stjärnan är. Man kan behålla den enheten eller multiplicera med solens luminositet (3,828*10^26 W), kom bara ihåg att skriva rätt enhet på diagrammet senare. Eftersom vi behöver anropa bolometricCorrection() för varje värde kan vi inte göra alla beräkningar samtidigt utan behöver göra dom en i taget. Vi skriver en for-loop som gör detta.

```
lum_list = []
for i in range(len(t_list)):
    lum_list.append(np.exp(-0.4*(m_abs_list[i] + bolometricCorrection(t_list[i]) + 4.74)))
```

Det andra vi vill göra är att se hur vårt ena mätvärde (luminositeten) beror av det andra (perioden). Vi gör det genom att få Python att rita en trendlinje. Det finns en funktion som skapar sådana linjer i numpy som heter `polyfit`. Den bestämmer vilket polynom som bäst stämmer överens med den data den får. Den tar tre argument, x-data, y-data och vilken grad polynomet ska ha. Vår x-data är listan över perioder, y-datan är luminositeterna vi just räknade ut och graden är 1 eftersom vi vill ha en linje. Funktionen kommer ge en array med två värden som är linjens koefficienter.

```
poly = np.polyfit(period_list, lum_list, 1)
```

För att faktiskt rita linjen i en graf behövs dess x- och y-koordinater för ett antal punkter. Som tur är finns en sådan funktion också i numpy och den heter polyval. `polyval` tar en lista på koefficienter (precis en sån som `polyfit` ger) och en lista på x-koordinater att räkna ut värdet på polynomet i. `polyval` ger en listan på y-värden för polynomet som motsvarar de x-värden man matade in. Dessa kan sedan användas för att rita upp en graf.

```
values_poly = np.polyval(poly, period_list)
```

```
35
36  lum_list = []
37  for i in range(len(t_list)):
38      # Compute the luminosity (linear unit) for each star
39      lum_list.append(np.exp(-0.4*(m_abs_list[i] + bolometricCorrection(t_list[i]) + 4.74)))
40
41  # Find polynomial coefficients of fitted polynomial
42  poly = np.polyfit(period_list, lum_list, 1)
43  # Create an array of values of the polynomial
44  values_poly = np.polyval(poly, period_list)
45
46  pyplot.title("Cepheid variable stars, brightness v period")
```

För att rita två linjer i samma graf behöver man bara anropa `pyplot.plot()` två gånger innan man anropar `pyplot.show()`. Skriv kod som skriver ut en till graf med luminositeter och trendlinjen i. Om man använder Repl.it kommer den bara visa den senaste grafen man ritar. För att slippa ta bort den kod som skapar den första grafen kan man göra om de raderna till kommentarer istället genom att skriva ett #-tecken först på varje rad. Då kommer Python ignorera den texten och bara gå vidare. Detta är även ett smidigt sätt att ta bort kod som man misstänker orsakar fel.

Den nya datan plottas med kommandona

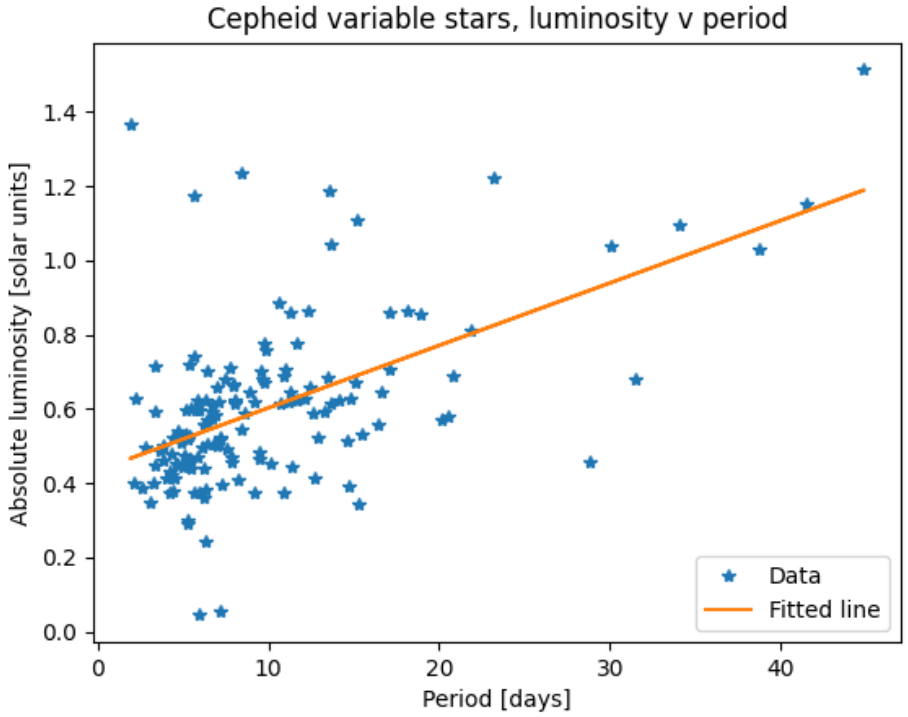
- `pyplot.plot(period_list, lum_list, '*', label="Data")`
  `pyplot.plot(period_list, values_poly, label="Fitted line")`

För att få en förklaring på vilken data som är vad använder man en legend. För att använda en sådan behöver man namnge datan i grafen, vilket görs med kommandot `label = namn` sist i plot-kommandot och ett kommando för att visa legenden,

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

`loc` talar om var rutan ska hamna, `"best"` gör att programmet själv väljer den "bästa" platsen.

```
29  mag_array = np.array(mag_list)  # convert to an array
30  parallax_array = np.array(parallax_list)
31
32  pc_array = 1. / (0.001 * parallax_array)  # Convert parallax to parsec
33  # Convert observed magnitude to absolte magnitude
34  m_abs_list = mag_array - 5. * np.log10(pc_array) + 5.
35
36  lum_list = []
37  for i in range(len(t_list)):
38      # Compute the luminosity (linear unit) for each star
39      lum_list.append(np.exp(-0.4*(m_abs_list[i] + bolometricCorrection(t_list[i]) + 4.74)))
40
41  # Find polynomial coefficients of fitted polynomial
42  poly = np.polyfit(period_list, lum_list, 1)
43  # Create an array of values of the polynomial
44  values_poly = np.polyval(poly, period_list)
45
46  #pyplot.title("Cepheid variable stars, brightness v period")
47  #pyplot.xlabel("Period [days]")
48  #pyplot.ylabel("Absolute magnitude [mag]")
49  #pyplot.gca().invert_yaxis()  # Flip the y-axis
50  #pyplot.plot(period_list, m_abs_list, '*')
51  #pyplot.show()  # Print the plot
52
53  pyplot.title("Cepheid variable stars, luminosity v period")
54  pyplot.xlabel("Period [days]")
55  pyplot.ylabel("Absolute luminosity [solar units]")
56  pyplot.plot(period_list, lum_list, '*', label="Data")
57  pyplot.plot(period_list, values_poly, label="Fitted line")
58  pyplot.legend(loc="best")
59  pyplot.show() # Print the plot
60
```



Den graf som produceras ser inte särskilt entydig ut, det finns datapunkter (stjärnor) både långt över och under linjen. Detta beror på att den absoluta magnituden är framräknad av två mätvärden, ljusstyrka och parallax, och bägge de värdena har felkällor som inte är tagna hänsyn till här. Dessutom är detta ett förenklat sätt att göra dessa beräkningar på, t.ex. så har inte extinktion inte tagits med i beräkningarna, se extraövningen. Men vår slutsats är ändå värdefull. Vi ser ett signifikant samband mellan period och absolut luminositet för dessa stjärnor, det är detta samband som låter astronomer avgöra avstånd till stjärnformationer som är för avlägsna för att mätas med parallaxmetoden.

Äkta vetenskapligt arbete innehåller ofta datapunkter som inte stämmer med slutsattsen, därför finns det statistiska verktyg för att avgöra hur “sann” en slutsats är utifrån datan som har använts. Dessa verktyg är väldigt viktiga men ligger utanför denna kurs.

Extra övning: Ta med interstellär extinktion

Interstellär extinktion är ett mått på hur mycket svagare en stjärna ser ut att vara på grund av damm och liknande i rymden som blockerar ljuset. I *data_cepheider.txt* finns det en till kolonn med data som är interstellär extinktion. Värdena är i magnituder. Det finns inte värden för alla observationer. Formeln för absolut magnitud från observerad magnitud, avstånd och extinktion är

$$M_{abs} = m_{obs} - 5 * log_{10}(dist_{pc}) + 5 - A$$

där *A* är extraktionen. Kan du modifiera koden så att extraktionen tas hänsyn till för de stjärnor den finns för?

Lösningsförslag:

Lägg till en till tom lista,

- `ext_list = []`

och en if-sats i for-loopen

- ```
  if words[6][0] != ' ':
        ext_list.append(float(words[6]))
  else:
       ext_list.append(0.0)
  ```

Denna if-sats kollar om det finns ett värde i sista "ordet" dvs om det första tecknet inte är ett mellanslag. Om det finns ett värde så läggs det till i `ext_list` annars läggs värdet 0 till istället. Gör om `ext_list` till ett array på samma sätt som de andra listorna. Ändra formeln till att ha med extraktionen,

- `m_abs_list = mag_array - 5. * np.log10(pc_array) + 5. - ext_array`

Använd den nya listan med magnituder på samma sätt som den förra.
  ```