AST1100 - 1C.4

Andreas Helland

14. november 2016

1 Introduksjon

Vi ser på observasjoner av en rekke stjerner og måler deres bevegelse i forhold til oss over tid ved hjelp fra dopplereffekten. Vi ser også på fluksen, altså hvor mye lys som treffer oss, over tid. Med dette kan vi finne ut om det er planeter i bane rundt stjernen som påvirker bevegelsen, samt om gitte planeter formørker stjernen eller ikke (altså om de ligger i planet med oss og stjernen og reduserer fluksen i det de beveger seg foran stjernen). Dersom de gjør det kunne det muligens gitt oss informasjon om gitt planet. Ut ifra den relative hastigheten til planeten og omløpstiden den har skal vi også finne massen til planeten.

2 Metode & Fremgangsmåte

Programmet er strukturert slik at all relevant informasjon fra målingsfilene er lagret i de lokale variablene til dataset objektet. Vi oppretter et objekt for hvert sett med målinger. [1]

Vi har fått 5 tekstfiler fylt med observasjonsdata (med 88850, blir filene fra mappe 50 brukt). Disse filene inneholder tre set med verdier. Fluks, bølgelengde og tidspunkt observasjonen ble gjort (fra første observasjon ved tid $t = 0$). Vi begynner med å sortere dem i egne lister, slik at vi har en liste med tidspunkt samt den målte forskyvningen av $H\alpha$ bølgelengden og fluksen i disse tidspunktene. [2]

Objektet kalles ut ifra hvilken oppgave man vil løse (en enkel if test). I den første oppgaven regner vi først ut den radielle hastigheten til stjernen ved å bruke dopplerligningen.

$$v_r = \frac{\Delta\lambda}{\lambda_0} c \tag{1}$$

Vi finner dermed pekuliærhastigheten ved å se på gjennomsnittshastigheten i alle målingene. Dette er hastigheten stjernesystemet har i forhold til oss. Vi vil fjerne denne for å finne den resterende hastigheten (bevegelsen innad i systemet).

[1] Hver av oppgavene har en egen liten kjørbar fil som (for det meste) kaller objektet hvor de fleste operasjoner blir utført.

[2] gjennomgangen av koden i denne (og alle de andre) rapporten(e) er relativt overfladisk, men de fleste funksjonene skal være kommentert nokså detaljert dersom det ikke var klart hva som blir gjort.

Vi plotter den relative hastigheten og fluksen over tid for alle systemene. Koden er formatert slik at det lar seg lett plotte 2 subplots sammen, med tid på x-aksen. Her sender verdiene man vil se øverst til upperPlot og det man vil plotte under til lowerPlot (og skriver det hele ut med plotter funksjonen).

Dersom vi ser en betydelig nedgang i fluksen (altså, ikke støy), kan dette indikere at en planet bevegde seg foran stjernen og formørket den.

For å finne planetens masse (dersom det er en planet i bane rundt stjernen) bruker vi formelen vi har kommet frem til i oppgavesettet.

$$m_p \sin i \quad = \frac{m_*^{2/3} v_{*r} P^{1/3}}{(2\pi G)^{1/3}} \tag{2}$$

Dette finner vi først ved å gjøre en øye-estimering av perioden til planeten og den radielle hastigheten. Perioden finner vi ved å se hvor lang tid det tar for hastighetsmønsteret å repetere sin endring, mens den radielle hastigheten finner vi ved å se på et top-punkt eller et bunnpunkt i hastigheten. Det er i disse punktene vi ser den egentlige bevegelsen (altså radiellhastigheten) til planeten fordi all (eller størst mulig andel dersom $i \neq 90°$) bevegelse skjer rettet mot oss (observatøren) eller fra oss.

Dersom vi finner planeter i bane rundt stjernene skal vi dermed finne radius og tettheten til planeten. Dette kan vi gjøre ved å se på transisjonsperioden fra når fluksen begynner å formørke planeten og når formørkelsen har stabilisert seg. Dette er perioden hvor planeten delvis dekker forsiden av stjernen. Når formørkelsen stabiliserer seg, dekker hele planeten deler av stjernen. Ved å se på hvor lang tid det tok for planeten å begynne å dekke stjernen til når den var helt innenfor stjernens areal (og hastigheten planeten beveger seg rundt stjernen) kan vi finne diameteren ved følgende formel.

$$2R_p \quad = (v + v_p)(t_1 - t_0) \tag{3}$$

Når vi har radiusen til planeten, samt massen, kan vi også finne tettheten.

$$\rho_p \quad = \frac{m_p}{\frac{4}{3}\pi R_p^3} \tag{4}$$

Etter vi har gjort øye-estimat, skal vi bruke minste kvadraters metode til å finne den beste modellen for radiell hastighet gitt i oppgaven.

$$v_r^{modell}(t)p \quad = v_r \cos(\tfrac{2\pi}{P}(t - t_0)) \tag{5}$$

Dette gjør vi ved å sette opp en metode for å finne et tidintervall vi vet topppunktet til hastighetskurven befinner seg, et intervall vi vet den faktiske topphastigheten befinner seg og et intervall vi vet omløpstiden befinner seg. Grunnen til at vi vil se på intervall og ikke bare finne de beste verdiene er fordi det er såpass mye støy i målingene at vi heller setter opp en modell for n antall av gitte verdier og sammenligner dem med målingene. Den modellen som gir minst forskjell (lavest $\Delta(t_0, P, v_r)$) gir oss det beste estimatet av de faktiske verdiene.

$$\Delta(t_0, P, v_r) \quad = v_r \sum_t (v_r^{data}(t) - v_r^{modell}(t, t_0, P, v_r)) \tag{6}$$

Programmet er satt opp slik at den finner intervallene av seg selv ved å dele datasettet i to. Den finner dermed 2 topppunkt i hastigheten og tar utgangspunktet i en periode mellom disse (altså, midten av intervallet er $t_2 - t_1$). Hastighetsintervallet tar utgangspunkt i en av disse toppunktene og går fra $-v_{topp}/2$ til $v_{topp}/2$. Det samme gjøres for tidsintervallet ($-t_{v_{topp}}/2$ til $t_{v_{topp}}/2$). Denne automatiske metoden for å finne intervall uten å gjøre øyeestimat fungerer bare dersom datasettet kan deles i to, og de to øverste punktene faktisk er topppunkt som følger hverandre. Hvis dette kriterie ikke møtes kan en tidligere funksjon brukes for å først kutte ned målingene (reduceData) slik at kun to toppunkt befinner seg i målingene man håndtterer.[3]

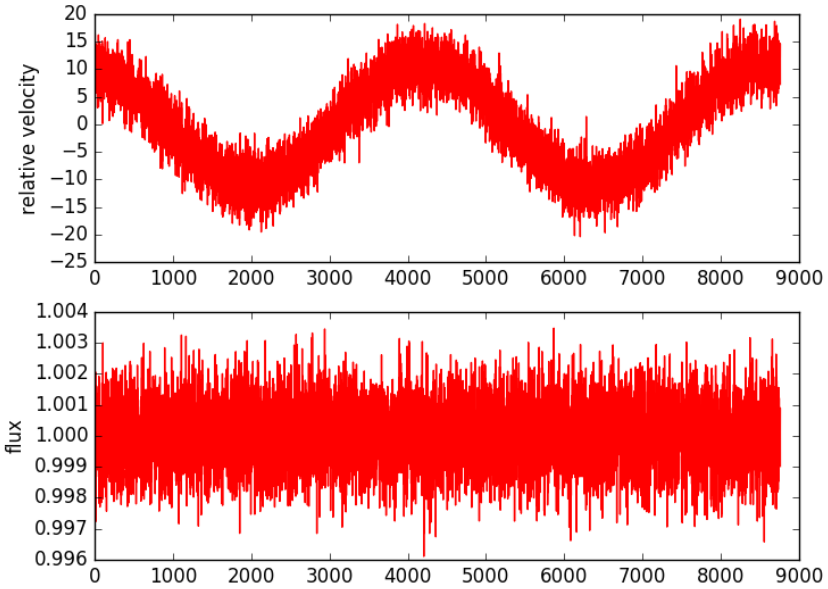
3 Resultater

Kjører vi den første filen (1.py) får vi følgende plot (figur 1, figur 2,figur 3,figur 4 og figur 5).

fra disse ser vi at stjerne 0, 2, 3 og 4 har planeter i bane rundt seg. Kun stjerne 3 og 4 har planeter som formørker lyset til stjernen. Lyskurven blir dempet akkurat i punktene man ville forventet at en planet går foran stjernen (når den relative radielle hastigheten er lik null, etter den har vært negativ, altså rett etter planeten beveger seg mot oss og er nærmere oss enn stjernen sin).

Bruker vi formel (2) med følgende øye-estimat får vi verdiene i tabell 1 hvor planetmassen er målt i jupitermasser.

[3]Dette er ikke nødvendig med de gitte målingene, fordi ingen av dem inneholder mer enn to omløpsperioder, men det skal poengteres at dette ikke er en spesielt generell måte å løse problemet på. Det fungerer derimot akkurat her.



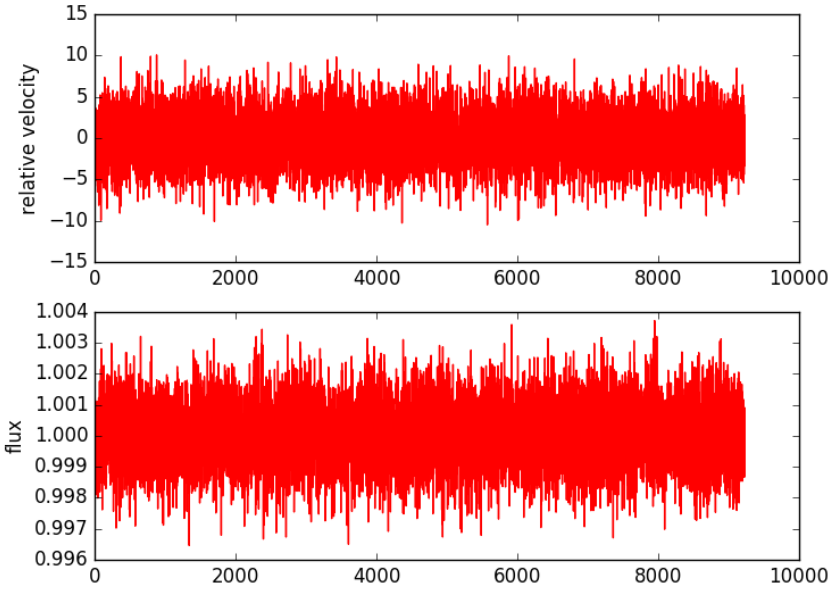
Figur 1: Radiell hastighet og fluks som funksjon av tid for stjerne 0.

Stjerne	v_r	Omløpstid P (dager)	Masse $m_{jupiter}$
0	10	4200	0.82
2	20	3000	1,86
3	50	3800	4,36
4	50	5000	5.11

Tabell 1: Øye-estimat av v_r og omløpstid og den utregnede massen til planeten gitt i jupiter masser ($1.898 \times 10^{27} kg$) ut ifra disse verdiene.

Dessverre har vi ikke nok informasjon til å finne en radius for disse planetene, selv ikke dem som formørker stjernen sin. Fordi det er såpass mange målinger gjort, kan vi se på et mindre sett med data og gjøre fluksen akkurat rundt formørkelsene tydeligere. Disse plottene finner man i figur 6 og 7. Her ser vi at det ikke er gjort noen målinger i perioden hvor planeten kun er delvis formørket.

Bruker vi modellfunksjonen for å finne verdiene vi tidligere kun har funnet ved øye-observasjon, får vi verdiene fra tabell 2.

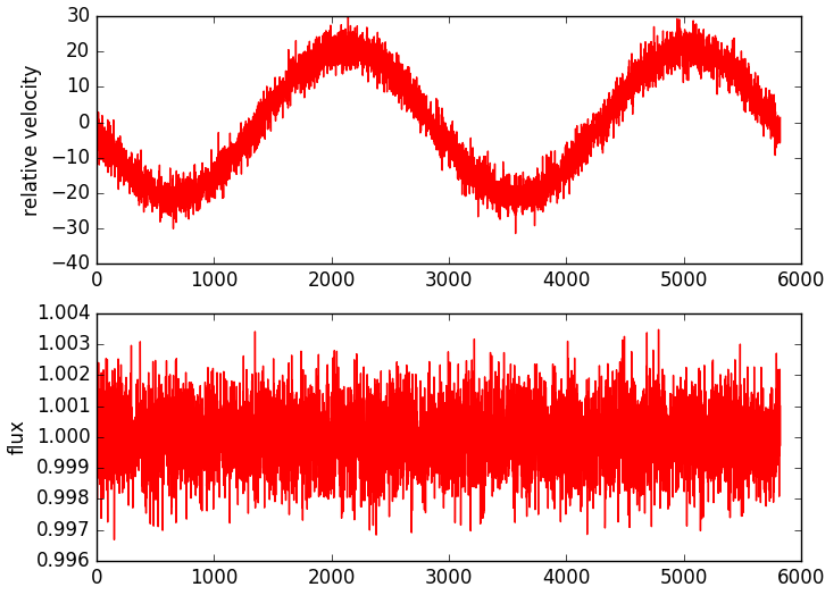


Figur 2: Radiell hastighet og fluks som funksjon av tid for stjerne 1.

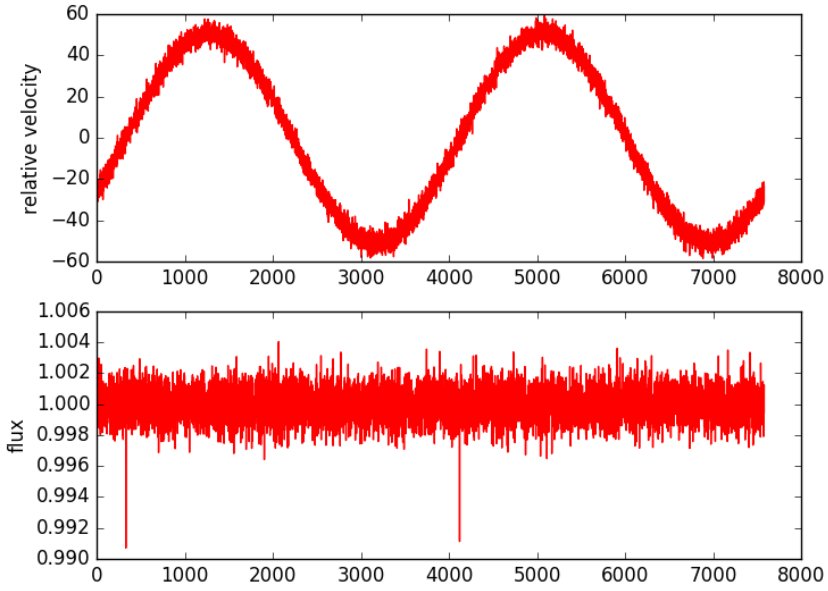
4 Diskusjon og konklusjon

Man kan gjøre et estimat som er relativt brukbart bare ved å lese av lys og hastighetskurver, men man vil kunne forbedre resultatet og skaffe seg mer nøyaktige verdier ved hjelp av modellering med minste kvadraters metode. Hvis man vil ha et godt estimat gjennom støyet og få en mer nøyaktig verdi for massen til en planet i bane rundt observerte stjerner er det noe man sansynligvis bør gjøre.

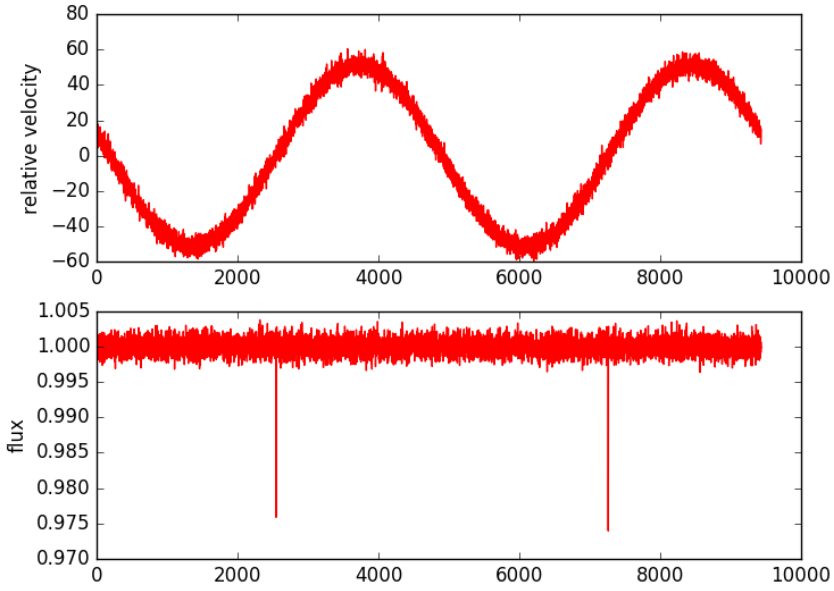
A Vedlegg



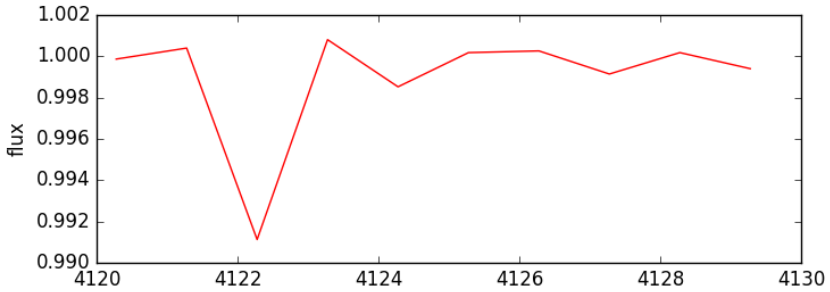
Figur 3: Radiell hastighet og fluks som funksjon av tid for stjerne 2.



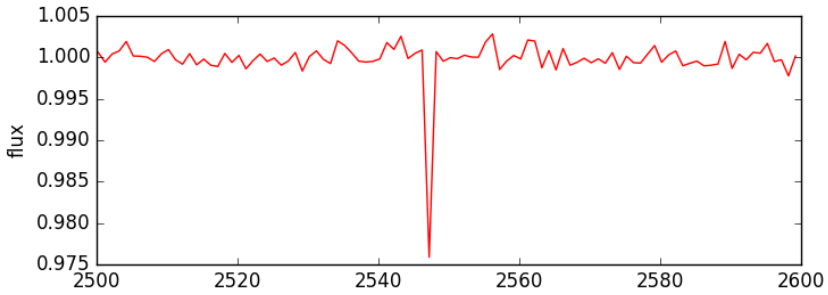
Figur 4: Radiell hastighet og fluks som funksjon av tid for stjerne 3.



Figur 5: Radiell hastighet og fluks som funksjon av tid for stjerne 4.



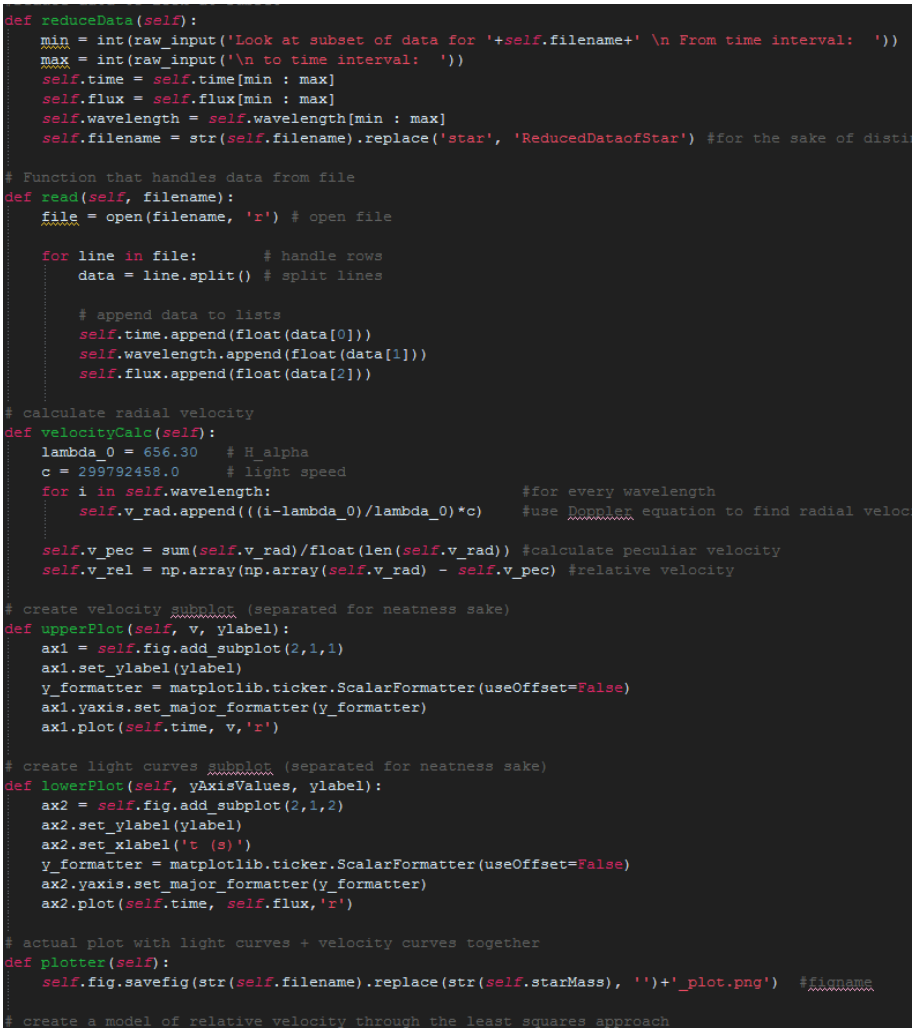
Figur 6: Fluksen til stjerne 3 i et mindre tidsintervallet hvor en planet formørker den.



Figur 7: Fluksen til stjerne 4 i et mindre tidsintervallet hvor en planet formørker den.

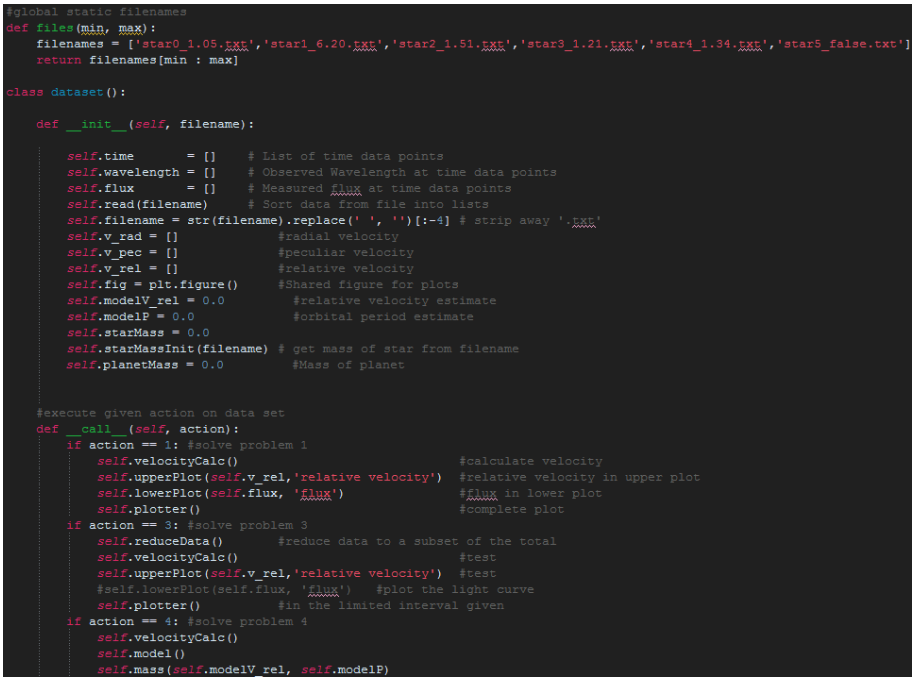
Stjerne	v_r	Omløpstid (dager)	$masse_{jupiter}$
0	8.3	3932.5	0.83
2	19.96	2750.6	1.81
3	47.64	3809	4.15
4	50.4	4703.4	5.05

Tabell 2: v_r og omløpstid P funnet ved hjelp av modellfunksjonen og den utregnede massen til planeten gitt i jupiter masser ($1.898 \times 10^{27} kg$) ut ifra disse verdiene.



```
def reduceData(self):
    min = int(raw_input('Look at subset of data for '+self.filename+' \n From time interval:  '))
    max = int(raw_input('\n to time interval:  '))
    self.time = self.time[min : max]
    self.flux = self.flux[min : max]
    self.wavelength = self.wavelength[min : max]
    self.filename = str(self.filename).replace('star', 'ReducedDataofStar') #for the sake of disti

# Function that handles data from file
def read(self, filename):
    file = open(filename, 'r') # open file

    for line in file:       # handle rows
        data = line.split() # split lines

        # append data to lists
        self.time.append(float(data[0]))
        self.wavelength.append(float(data[1]))
        self.flux.append(float(data[2]))

# calculate radial velocity
def velocityCalc(self):
    lambda_0 = 656.30   # H_alpha
    c = 299792458.0     # light speed
    for i in self.wavelength:                           #for every wavelength
        self.v_rad.append(((i-lambda_0)/lambda_0)*c)    #use Dopplex equation to find radial veloc

    self.v_pec = sum(self.v_rad)/float(len(self.v_rad)) #calculate peculiar velocity
    self.v_rel = np.array(np.array(self.v_rad) - self.v_pec) #relative velocity

# create velocity subplot (separated for neatness sake)
def upperPlot(self, v, ylabel):
    ax1 = self.fig.add_subplot(2,1,1)
    ax1.set_ylabel(ylabel)
    y_formatter = matplotlib.ticker.ScalarFormatter(useOffset=False)
    ax1.yaxis.set_major_formatter(y_formatter)
    ax1.plot(self.time, v,'r')

# create light curves subplot (separated for neatness sake)
def lowerPlot(self, yAxisValues, ylabel):
    ax2 = self.fig.add_subplot(2,1,2)
    ax2.set_ylabel(ylabel)
    ax2.set_xlabel('t (s)')
    y_formatter = matplotlib.ticker.ScalarFormatter(useOffset=False)
    ax2.yaxis.set_major_formatter(y_formatter)
    ax2.plot(self.time, self.flux,'r')

# actual plot with light curves + velocity curves together
def plotter(self):
    self.fig.savefig(str(self.filename).replace(str(self.starMass), '')+'_plot.png')  #figname

# create a model of relative velocity through the least squares approach
```

Figur 8: Funksjonene i dataset objektet som blir brukt i den første delen av oppgaven.



```
#global static filenames
def files(min, max):
    filenames = ['star0_1.05.txt','star1_6.20.txt','star2_1.51.txt','star3_1.21.txt','star4_1.34.txt','star5_false.txt']
    return filenames[min : max]

class dataset():

    def __init__(self, filename):

        self.time       = []    # List of time data points
        self.wavelength = []    # Observed Wavelength at time data points
        self.flux       = []    # Measured flux at time data points
        self.read(filename)     # Sort data from file into lists
        self.filename = str(filename).replace(' ', '')[:-4] # strip away '.txt'
        self.v_rad = []             #radial velocity
        self.v_pec = []             #peculiar velocity
        self.v_rel = []             #relative velocity
        self.fig = plt.figure()     #Shared figure for plots
        self.modelV_rel = 0.0         #relative velocity estimate
        self.modelP = 0.0             #orbital period estimate
        self.starMass = 0.0
        self.starMassInit(filename) # get mass of star from filename
        self.planetMass = 0.0         #Mass of planet


    #execute given action on data set
    def __call__(self, action):
        if action == 1: #solve problem 1
            self.velocityCalc()                             #calculate velocity
            self.upperPlot(self.v_rel,'relative velocity')  #relative velocity in upper plot
            self.lowerPlot(self.flux, 'flux')               #flux in lower plot
            self.plotter()                                  #complete plot
        if action == 3: #solve problem 3
            self.reduceData()       #reduce data to a subset of the total
            self.velocityCalc()                             #test
            self.upperPlot(self.v_rel,'relative velocity')  #test
            #self.lowerPlot(self.flux, 'flux')   #plot the light curve
            self.plotter()          #in the limited interval given
        if action == 4: #solve problem 4
            self.velocityCalc()
            self.model()
            self.mass(self.modelV_rel, self.modelP)
```

Figur 9: constructor og call i dataset.

```
# create a model of relative velocity through the least squares approach
def model(self):
    t, t0, vr, P = self.modelValues() #automated estimate for intervals
    delta = [0,0,0]
    best_delta = 10000000000000000

    # Calculate minimal delta by least square approach
    for i in t0:
        for j in vr:
            for k in P:
                model = j*cos((2*pi/k*(i-t)))              #Calculate model for radial velocity
                deltaCurrent = sum((self.v_rel - model)**2) #Calculate change
                # if current delta is smaller than previous smallest delta
                if deltaCurrent < best_delta:
                    best_delta = deltaCurrent #set currentDelta as best estimate
                    delta[0], delta[1], delta[2] = i, j, k

    self.modelP, self.modelV_rel = delta[2], delta[1] # store best estimate of orbital period & velocity

#find interval values for the least square approach function.
#This sloppy method will only work as long as there is a peak velocity in each half of the data set
#If you want it to work in a different data set, just set the limit between two peaks and use the reduceData func
def modelValues(self):
    t1, t2,vr,vr2= self.time[0],self.time[0],self.v_rel[0],self.v_rel[0]    #radial velocity is highest and/or lo
    for i in range(len(self.v_rel)):    #for every v_rel value
        if self.v_rel[i] > vr and i<=len(self.time)/2: #if v is higher than previous highest (first half)
            vr = self.v_rel[i]  #store v
            t1 = self.time[i]   #store time
        if self.v_rel[i] > vr2 and i>=len(self.time)/2: #if v is higher than previous highest (second half)
            vr2 = self.v_rel[i] #store a second v
            t2 = self.time[i]   #store a second time stamp at the second v peak

    P = t2-t1 #period = time steps between the two peaks
    return t1, np.linspace(t1-100, t1+100, 20), np.linspace(vr-10, vr+10, 20), np.linspace(P-100,P+100, 20)

#calculate planet mass from formula (5)
def mass(self, v, P):
    m_sun = 1.98892*10**30   #solar mass
    G = 6.67428*10**(-11)    #gravitational constant
    self.planetMass = ((self.starMass * m_sun)**(2.0/3)*v*(P*86400)**(1.0/3))/((2*pi*G)**(1.0/3))

#find mass of star from filename
def starMassInit(self, filename):
    starMassString = str(filename).replace('star', '')          #remove 'star' string
    for i in xrange(6): #For every star number in range (change range for larger number of data sets)
        starMassString = starMassString.replace(str(i)+'_', '') #remove number from text file
    starMassString = starMassString.replace('.txt', '')         #remove .txt
    try:
        self.starMass = float(starMassString)                   #store remaining numeric value
    except (IndentationError, ValueError):
        pass
```

Figur 10: modelleringsfunksjonen samt diverse andre funksjoner i objektet.