The following people have participated in creating these solutions: Nicolaas E. Groeneboom, Magnus Pedersen Lohne, Karl R. Leikanger
NOTE: There might be errors in the solution. If you find something which doens't look right, please let me know

Partial solutions to problems: Lecture 3

Problem 1

1. $v_r \approx 12$ m/s, yes!

2. $v_r \approx 0.1$ m/s, no!

3. Using formulae in the text, you should find that $v_r \propto 1/\sqrt{a}$ such that the radial velocity decreases for larger distances a.

4. 1.2×10^6 km.

Problem 2

1. We assume that the orbits of all planets are circular. In the previous exercise, we found the speed of the sun's wobbling due to Jupiter was given as $v_* = 12.20$m/s. The radius a_* the sun wobbles is given from

$$v_* = \frac{2\pi a_*}{P}$$

such that

$$a_* = \frac{v_* P}{2\pi}$$

What we are interested in is the maximum distance from us to the star, and this is given by $\theta = (a_*/d)$. **Note:** When using this approximation ($\tan\theta \sim \theta$ for small values of θ), it is important to perform the actual computations using *radians* and not *degrees*. Now,

$$d = 2 * \frac{a_*}{\theta} = \frac{v_* P}{\pi\theta}$$

where $\theta = 0.1'' = \frac{1}{60}\frac{1}{60}\frac{2\pi}{360} = 4.848 \cdot 10^{-7}$ radians. The extra factor 2 is added, as the star wobbles with a magnitude a_* in *two* directions, doubling the effect.

When inserting the values for Jupiter, one obtains

$$d \approx 3.04 \cdot 10^{15} m = 0.32\text{ly}$$

which is the *distance from us* the star needs to be in order to detect any wobbling. Note that the distance to Sun's closest neighbour Proxima Centauri is 4.22 ly.

2. $d \approx 0.0002$ly.

3. First of all, wikipedia states that *Proxima Centauri* contains 0.123 solar masses. We use a formula presented in chapter 2, giving a relation between different orbit sizes. We have:

$$a_1 = \frac{\mu a}{m_1}$$

where $a_1 = a_*$ is the radius of the orbit of the wobbling star about the center of mass, $m_1 = m_*$, $a \equiv a_d$ is the distance between the star and the planet while the reduced mass $\hat{\mu} \sim m_p$ can be approximated to be the mass of the orbiting planet. Thus,

$$m_p = \frac{m_1 a_1}{a} = \frac{m_* a_*}{a_d}$$

We need to decide on a_*. Now, in order to detect wobbling, $\theta = 0.1''$, and we know $d = 4.22 ly$ such that a_* must be

$$2a_* = d\theta$$

inserting this into 3, we find

$$m_p = \frac{m_* d\theta}{2a_d} = \frac{0.123 \cdot \text{Msun} \cdot 4.22\text{ly} \cdot \frac{0.1 \cdot 2\pi}{60 \cdot 60 \cdot 360}}{1\text{AU} \cdot 2} \approx 8 \text{ Jupiter masses}$$

4. $1.6 m_{\text{jupiter}}$

Problem 3

1. Answer found in the text.

2. 14 km/s.

3. $v_{*r} \approx 445$ m/s, $P \approx 97$ hours.

4. $m_p \approx 4.4 m_{\text{jupiter}}$.

5. Answer found in the text./

6. $\Delta t \approx (1.685 - 1.665) \times 10^5 \approx 2000$ seconds, $v_p \approx 140$ km/s, $r_p \approx 140 \times 10^3$ km, $\rho \approx 700 kg/m^3$

7. gas planet.

Problem 4

Here is an outline of a Python code which can be used:

```
from scitools.all import *


#STAR CLASS
class star:
```

```python
#Constructor
def __init__(self, filename):
   #Declare self-variables/lists
   self.filename = filename
   self.time = []
   self.flux = []
   self.lambda_obs = []
   ...
   ...


   #Calculations/call functions
   self.read_file()
   self.rad_velocity()
   self.pec_velocity()
   self.light_curve()
   self.vel_curve()
   self.model()


#Function that reads data from the file self.filename
def read_file(self):
   ...
   self.time.append(...)          #Store data in self-lists
   self.flux.append(...)
   self.lambda1.append(...)
   ...
   #Make arrays
   ...


#Function that calculates the radial velocity for each observed wavelength
def rad_velocity(self):
   self.rad_vel = ...       #Use Doppler's formula


#Function that calculates the peculiar velocity
def pec_velocity(self):
   self.pec_vel = sum(..)/len(..)  #sum(array) - sum all elements in an array
                                   #len(array) - length of array
   #Write to file
   ...


#Function that plots the light-curve
def light_curve(self):
   #Plot flux vs time
   ...


#Function that plots the velocity-curve
```

```python
def vel_curve(self):
    #Relative velocity - Deviation of the radial velocity from the peculiar
    #(average) velocity
    self.rel_vel = self.rad_vel - self.pec_vel

    #Plot relative velocity vs time
    ...



#Function that evaluates the best values of the constants v_r, P and t_0
#in the cosine-model, based on the least-square method
def model(self):
    #Ask in screen
    planet = float(raw_input('Does %s have a planet eclipsing? (yes/no)' % (self.name)))

    if planet == 'yes':
        #Read in min and max possible value of t0, vr and P from screen
        t0_min = float(raw_input(' ... '))
        t0_max = ...
        vr_min = ...
        ...

        #Declare t0, vr and P - arrays
        t0 = linspace(t0_min, t0_max, 'number of evaluation-points per constant')
        vr = ...
        P = ...

        #Declare best-variables (set equal the first value in their arrays)
        best_t0 = ...
        best_vr = ...
        best_P = ...

        #Go through every combinations of the constants t0, vr and P, one
        #for-loop for each constant-array
        for i in range(len(t0)):
            for j in (...):
                for k in (...):
                    #Calculate the array of model-velocity, use t0[i], vr[j] and P[k]
                    rel_vel_model = ...

                    #Calculate the sum of the difference between all the elements
                    #in the self.rel_vel-array and the rel_vel_model-array squared.
                    delta = sum((self.rel_vel - rel_vel_model)**2)

                    #Check if you have a better constant-set (t0[i], vr[j] and P[k])
                    #than the previously stored best values
                    if delta < best_delta:
                        ...
                        ...
```

```
         #Plot the result, use hold('on') to plot two graphs in the same
         #figure, remember to use hold('off') afterwords
         ...

         #Write the best constant-values to file - write all star-values in 1 file
         if self.name = 'star0':
            file = open('filename', 'w')
         else:
            file = open('filename', 'a')
         ...                              #Write to file
         file.close()


#____
#MAIN

#Make star-objects
star0 = star('star0.txt')
star1 = star('star1.txt')
star2 = star('star2.txt')
star3 = star('star3.txt')
star4 = star('star4.txt')
star5 = star('star5.txt')
star6 = star('star6.txt')
star7 = star('star7.txt')
star8 = star('star8.txt')
star9 = star('star9.txt')
```

The best program approach is to make a star-class. In this way we get a
compact and well-presented program where we don't have too many variables
to think of. For each star object that we make (in the beginning of main), with
argument 'filename', the constructor function automatically runs. Since all the
functions are called from the constructor, all the calculations are carried out
by just making the object. If we were to do more calculations in main, and
especially if we needed variables inside the star objects, the gain in the class
approach would be even clearer. But even in this problem it should be fairly
obvious what we gain by the class approach.

In the following we will present results for star0, a similar apporach should
be used for the other stars.

1. We do not present all the plots here, but give a summary of what you
   should have found. The following table shows the results for star0-9:
   Star0 has peculiar velocity 10 km/s.

2. We again assume that the inclination $i = 90°$. We then use the formula

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

From the radial velocities of **star0** above, note that $v_{max} = 220$m/s,

Star	Planet	Eclipse	M
0	Y	Y	0.8
1	N	N	2.8
2	N	N	0.5
3	Y	Y	0.5
4	Y	Y	1.8
5	Y	N	0.7
6	Y	N	1.6
7	N	N	2.1
8	Y	N	7.0
9	Y	N	8.0

orbiting a period of $P = (t_1 - t_0) \sim 3.5 \cdot 10^5$s. Inserting, we find

$$m_p = 3.15 \cdot 10^{27} \sim 1.5 \, \text{Jupiter masses.}$$

This is a Jupiter-sized planet.

3. From the light curves, it is easy to see that star0 has an eclipsing planet, but it is hard to estimate the exact slope of the curve. By looking at the second "bump", we estimate $(t_1 - t_0) \sim 1 \cdot 1300$ seconds, such that

$$R \sim \frac{1}{2} \cdot ((220 + 220 * (m_*/m_p))\text{m/s} \cdot 1 \cdot 10^3\text{s} \sim 82000 km.$$

or a little more than the radius of Jupiter, again a Jupiter-sized planet. The density is thus

$$\rho = \frac{m_p}{4/3! pi R^3} \approx 1200 kg/m^3$$

which is a gas planet as expected. Note that the final answer may differ quite a lot depending on the value you find for R. And there will be large uncertainties in the value you find for R because of uncertainties in reading off the values from the light curve (which again comes from the noise in the data).

4. We now have a data set for the radial velocities taken at different times, namely v_i. We are now interested in fitting a cosine-curve **model** to the data, and thus determining three parameters: The orbital period P, the initial time t_0 and the amplitude v_r. This is done by using **least-square fitting**, we calculate

$$lsq \sim \sum_{data} (\text{data} - \text{model})^2 \tag{0.1}$$

We assume the cosine model to follow

$$v_r^{model} = v_r \cos\left((2\pi/P)(t - t_0)\right)$$

where we let v_r, P and t_0 vary linearly within a selected interval. Select fitting intervals for varying parameters "by eye" from the graph of the radial velocities of star0: $[v_{start}, v_{stop}] = [150, 300]$ m/s, $[P_{start}, P_{stop}] =$

$[500, 700]$ (measured in index elements of the data vector) and $[t_{0start}, t_{0stop}] = [400, 550]$ (measured in index elements of the data vector). It may be convenient to operate with the index elements instead of proper time in seconds in the code. When you have finally found the result for P and t_0 you can convert from index to seconds. The final results will depend a bit on how the grid is chosen, but should for star0 not be very different from the 'bye eye' result above.