Chapter 2

Discovering exoplanets: The radial velocity method

2.1 The radial velocity method

When a planet rotates around a star, the star also performs a rotating motion. If this motion is not exactly in the plane of the sky, then there will be a *radial velocity* component of the stellar motion with respect to the line of sight to the star. This motion can be detected by astronomers, because the spectrum of the star is then doppler-shifted, and this doppler-shift is varying with time because the radial velocity describes an oscillation.

Since planets have a relatively low mass compared to the star, this motion of the star is, of course, very small. For a Jupiter-mass planet it is in the order of hundreds of meters per second, but for an Earth-mass planet it is more of the order of *decimeter per second*. These are tiny doppler shifts (compare to the light speed), but they can nowadays be measured.

Using this radial velocity method (often abbreviated to *RV method*) many exoplanets have been discovered since 1995.

For a circular motion with semi-major axis a of a planet around a star the math is easy. The orbital velocity of the star around the barycenter is:

$$v_* = \frac{M_p}{M_* + M_p} v_K \tag{2.1}$$

where the Kepler velocity is

$$v_K = \sqrt{\frac{G(M_* + M_p)}{a}} \tag{2.2}$$

where M_* is the stellar mass and M_p is the planetary mass.

Now consider that the orbit of the planet has an inclination i with respect to the line of sight, then the radial velocity as a function of time is:

$$v_{\mathrm{radial}}(t) = v_0 + v_* \sin(i) \cos(\Omega_K t + \phi_0) \tag{2.3}$$

where v_0 is the systemic radial velocity (the radial velocity of the combined star+planet system with respect to us), and Ω_K is the Kepler frequency

$$\Omega_K = \sqrt{\frac{G(M_* + M_p)}{a^3}} \tag{2.4}$$

and t is the time. The orbital period is

$$P = \frac{2\pi}{\Omega_K} \tag{2.5}$$

and ϕ_0 is a phase constant.

Observers measure the doppler shift, and reconstruct $v_{\mathrm{radial}}(t_i)$ for a series of *observation epochs* t_i. These are then published in tables that can be, for instance, downloaded from the NASA exoplanet archive[1].

[1]`https://exoplanetarchive.ipac.caltech.edu`

This orbital period is often relatively easily extracted from the observations, because the radial velocity signal will display a certain periodicity, which can be accurately determined if sufficient orbits have been measured.

Exercise 13: Which quantity can one determine from the amplitude of the RV signal? Does the measurement give us a *lower limit* or an *upper limit*?

Exercise 14: If the inclination (by luck) happens to be 90 degrees (i.e. the orbit lies edge-on with respect to our line of sight), you can *also* measure transits. Explain why this means that we can then infer the average *density* of the planet.

2.2 The case of 51 Pegasi

Now that we have a model of the radial velocity of the star as a function of the known variables M_* and the unknown variables inclination i, planet mass M_p and period P, we can try to fit actual data. The star 51 Pegasi has a *hot Jupiter* circling around it every 4.230785 days. You can download the RV data of the paper of Howard & Fulton (2016) from:

```
https://exoplanetarchive.ipac.caltech.edu/data/ExoData/0113/0113357/data/UID_0113357_RVC_005.tbl
```

Exercise 15: Plot these data in a way where on the x-axis the phase is given (i.e. wrap time around a period of 4.230785 days).

Now let us fit our model (Eq. 2.3)through this data. There are several methods for doing this. We will discuss the *least-squares* method, the *maximum likelihood* method and the *Markov-Chain Monte Carlo* method. Though we will only discuss these methods in extreme brevity.

Suppose we define x to be the phase divided by 2π. Then the time variable of the RV data converts to values of x between 0 and 1. The RV data itself are values of the radial velocity in m/s. Let us call these y. The RV data also has error estimates. Let us call these Δy. The RV dataset is thus a table with three columns: x, y and Δy.

Let us simplify our model as:

$$y(x) = y_0 + A\cos(2\pi x + \phi_0) \tag{2.6}$$

The parameters we wish to fit are (A, y_0, ϕ_0).

Python `scipy.optimize` provides the `curve_fit()` method, which performs a *least-squares fit*. It tries to find the values of (A, y_0, ϕ_0) for which

$$\sum_i \frac{(y_i - y_{\mathrm{model}}(x_i))^2}{\Delta y_i^2} \tag{2.7}$$

has its minimum. In other words: it tries to find the model for which the difference to the data (taking into account the data uncertainties) is the smallest. Here is a code snippet:

```
#
# Model
#
def rvmodel(x,ampl,offset,phase):
    return offset+ampl*np.cos(2*np.pi*x+phase)

#
# Parameters: initial values, and bounds
#
par0   = np.array([50., 0., 0.])
lobnd  = np.array([10., -50., 0.])
upbnd  = np.array([100., 50., 2*np.pi])
bounds = (lobnd,upbnd)

```

```
#
# Now perform least-squares fit
#
q = optimization.curve_fit(rvmodel, x, y, par0, yerr, bounds=bounds)
ampl_lsq, offset_lsq, phase_lsq = q[0]
```

You can find this snippet in `snippets/snippet_lsq.py`.

Exercise 16: Use this Python least squares fitting tool to fit our model through this data, and determine the three unknown parameters. Note that the inclination and the planet mass are degenerate, so define instead a combined parameter for these two.

Exercise 17: Now overplot the model over the data and see if it is indeed a succesful fit.

Exercise 18: The least-squares method returns, in addition to the best-fit model parameters, also a *covariance matrix*: in the snippet this would be `q[1]`. What is the meaning of this matrix?

The maximum likelihood method does not try to minimize the difference to the data, but tries to maximize the likelihood that the data could have been generated by the model. The *likelihood function* is the likelihood that the data are the way they are, given the model and the data uncertainties and the parameters θ_k (where in our example $\theta_0 = A$, $\theta_1 = y_0$ and $\theta_2 = \phi_0$). It is given by

$$\ln p(y|x,\Delta y,\theta) = -\frac{1}{2}\sum_i \left[\frac{(y_i - y_{\mathrm{model}}(x_i))^2}{\Delta y_i^2} + \ln(2\pi\Delta y_i^2)\right] \tag{2.8}$$

Here is a code snippet:

```
import scipy.optimize as op
#
# Model
#
def rvmodel(x,ampl,offset,phase):
    return offset+ampl*np.cos(2*np.pi*x+phase)

#
# Likelihood function
#
def lnlike(theta, x, y, yerr):
    ampl, offset, phase = theta
    model = rvmodel(x,ampl,offset,phase)
    sigma2 = yerr**2
    return -0.5*(np.sum((y-model)**2*inv_sigma2 + np.log(2*np.pi*sigma2)))

#
# Bounds
#
bounds = [(10.,100.),(-50.,50.),(0.,2*np.pi)]

#
# Maximum likelihood fit
#
nll = lambda *args: -lnlike(*args)
result = op.minimize(nll, [50., 10., 3.], args=(x, y, yerr), bounds=bounds)
ampl_ml, offset_ml, phase_ml = result["x"]
```

You can find this snippet in `snippets/snippet_ml.py`.

Exercise 19: Do the same, but now with the maximum likelihood method.

The least-squares and maximum-likelihood methods are both relatively similar. The least-squares method gives error estimates for the derived (=fitted) parameters in the covariance matrix. But sometimes the likelihood function in parameter space is very non-linear and the linear covariance matrix may not give a good enough idea of the uncertainties of the parameters. This is especially the case if there are near-degeneracies in the model.

A way to get a better feeling for the degeneracies and uncertainties is to use a *Monte Carlo sampling method*. The idea is to use random numbers to "try out" various parameter sets. A particular subclass of these methods is the *Markov Chain Monte Carlo* (MCMC) method. This method lets a set of *N samplers* perform a random walk through parameter space, according to certain rules that let them automatically be attracted toward the best-fit solution, but still randomly swirl around it in a way that samples the uncertainties. The set of rules that determine this random motion is derived from *Bayesian analysis*, which is a rigorous statistical analysis of how a model fits the data. We will not go into any details here, but you are encouraged to read up on *Bayesian inference* because this is a very widely used, and mathematically robust way of extracting information from observed data.

We shall use the `emcee` Python package here. You may need to install it on your system, because it is not standard in Python. But a simple `pip install emcee` usually works.

Here is the snippet:

```
#
# Model
#
def rvmodel(x,ampl,offset,phase):
    return offset+ampl*np.cos(2*np.pi*x+phase)

#
# The Bayesian formula:  P_posterior(model|data) =
#         P_prior(model) * P(data|model) / someconstant
#
def lnprob(theta, x, y, yerr):
    lp = lnprior(theta)
    if not np.isfinite(lp):
        return -np.inf
    return lp + lnlike(theta, x, y, yerr)

#
# Define the prior
#
bounds = [(10.,100.),(-50.,50.),(0.,2*np.pi)]
def lnprior(theta):
    global bounds
    ampl, offset, phase = theta
    if bounds[0][0] < ampl < bounds[0][1] and \
       bounds[1][0] < offset < bounds[1][1] and \
       bounds[2][0] < phase < bounds[2][1]:
        return 0.0
    return -np.inf

#
# Likelihood function
#
def lnlike(theta, x, y, yerr):
    ampl, offset, phase = theta
    model = rvmodel(x,ampl,offset,phase)
    sigma2 = yerr**2
    return -0.5*(np.sum((y-model)**2*inv_sigma2 + np.log(2*np.pi*sigma2)))

#
# Initialize the walkers of MCMC
#
ndim, nwalkers = 3, 100
pos = []
```

```
for i in range(nwalkers):
    w = []
    for k in range(ndim):
        w.append(bounds[k][0]+(bounds[k][1]-bounds[k][0])*np.random.rand())
    pos.append(np.array(w))

#
# Set up the Emcee
#
import emcee
sampler = emcee.EnsembleSampler(nwalkers, ndim, lnprob, args=(x, y, yerr))

#
# Now run the MCMC chain
#
nchain = 500
sampler.run_mcmc(pos, nchain)

#
# See how the population evolves the first 10 steps
#
figevol = plt.figure()
ism = 50
plt.plot(sampler.chain[:,ism,0],sampler.chain[:,ism,1],'.')
for ism in range(0,50,10):
    plt.plot(sampler.chain[:,ism,0],sampler.chain[:,ism,1],'.')
plt.xlabel('ampl')
plt.ylabel('offset')

#
# Triangle plot, discarding first 50 samples
#
import corner
samples = sampler.chain[:, 50:, :].reshape((-1, ndim))
fig3 = corner.corner(samples, labels=["ampl", "offset", "phase"])

plt.show()
```

You can find this snippet in `snippets/snippet_mcmc.py`.

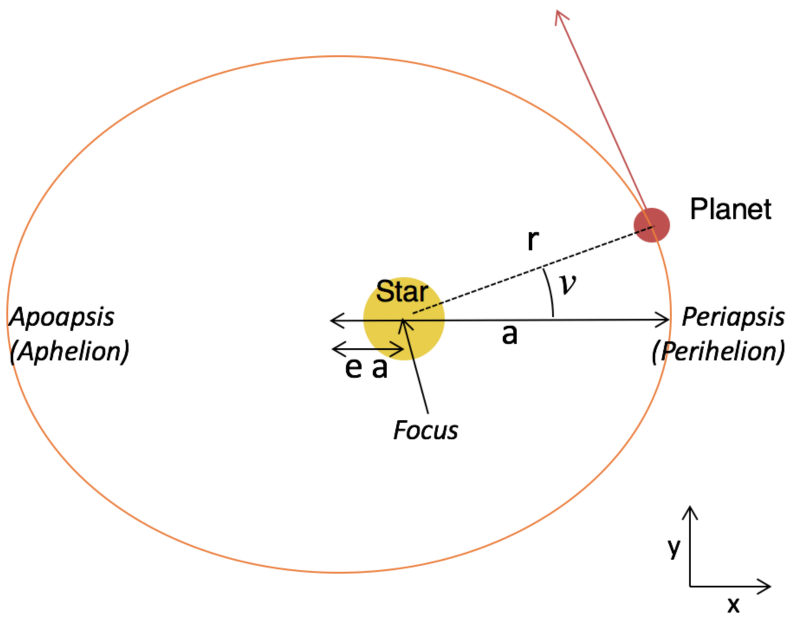
Exercise 20: Do the same, but now with the MCMC method. What is the meaning of the triangular corner plot?

Exercise 21: If you would keep M_p and $\sin(i)$ as two *independent* parameters in the fitting procedure, what do the triangular corner plots look like, in particular for the correlation between M_p and $\sin(i)$?

2.3 Kepler orbit semi-analytically

If the planet describes such a nice circular orbit, it is easy to infer from the RV measurements the data we need. But often exoplanetary orbits have *eccentricity*, i.e. their orbits are elliptic. This complicates matters, because now the number of unknown parameters of the orbit increases.

We numerically integrated planetary orbits in the previous chapter. However for a 2-body problem the orbit can be formulated *nearly* fully analytically. There is only one tiny step in the calculation where a numerical iteration is required.



In the cartoon of an elliptic kepler orbit the symbols are as follows: a is the semi-major axis, e the eccentricity, r is the radial distance between the planet and the star, $\nu(t)$ is the angle that the location of the planet has at time t.

To find the location coordinates $x_p(t)$ and $y_p(t)$ of the planet for a given time t after periastron passage, given the semi-major axis a, eccentricity e, stellar mass M_* and planet mass M_p, you carry out the following procedure:

1. First compute the Kepler angular frequency:

$$\Omega_K = \sqrt{\frac{G(M_* + M_p)}{a^3}} \tag{2.9}$$

2. Compute the *mean anomaly* M:

$$M = \Omega_K t \tag{2.10}$$

3. Compute the *eccentric anomaly* E by *numerically solving* the following equation:

$$M = E - e\sin(E) \tag{2.11}$$

   One simple way to solve this equation for E, given M and e, is to do an iteration procedure. Start with $E_0 = M$. Then iterate:

$$E_{i+1} = M + e\sin(E_i) \tag{2.12}$$

   For not too large value of e (i.e. not too close to 1) this equation converges reasonably fast. This is the only numerical step in this procedure.

4. Now compute the cosine of the *true anomaly* ν:

$$\cos(\nu) = \frac{\cos(E) - e}{1 - e\cos(E)} \tag{2.13}$$

5. The sine of ν follows from

$$\sin(\nu) = \pm\sqrt{1 - \cos^2(\nu)} \tag{2.14}$$

   The sign of $\sin(\nu)$ is taken to be the same as the sign of $\sin(M)$.

6. Compute the r from

$$r = \frac{a(1 - e^2)}{1 + e\cos(\nu)} \tag{2.15}$$

   The distance r is the distance between the star with the planet.

7. To compute the distance of the planet r_p (or star r_*) to the center of mass (barycenter) we compute:

$$r_p = \frac{M_*}{M_* + M_p} r \tag{2.16}$$

$$r_* = -\frac{M_p}{M_* + M_p} r \tag{2.17}$$

8. Now compute x_p and y_p:

$$x_p = r_p \cos(\nu), \qquad y_p = r_p \sin(\nu) \tag{2.18}$$

or equivalently x_* and y_*:

$$x_* = r_* \cos(\nu), \qquad y_* = r_* \sin(\nu) \tag{2.19}$$

This procedure is exact, as long as the numerical solution for E is computed with sufficient accuracy. But even if a small error tolerance is left for E, the long-term evolution of the Kepler orbit with this procedure is perfectly stable, in contrast to numerical integrations.

We can also compute the velocity vector $(v_{\mathrm{p},x}, v_{\mathrm{p},y})$ at a given t, from a given solution of the location (x_p, y_p). This is done with the following procedure:

1. Use the *vis-viva equation*

$$v^2 = G(M_* + M_p)\left(\frac{2}{r} - \frac{1}{a}\right) \tag{2.20}$$

to compute the square length of $\mathbf{v} = d\mathbf{x}/dt = d(\mathbf{x}_p - \mathbf{x}_*)/dt$.

2. The angle ϕ of the velocity vector with respect to circular is given by the following equation:

$$\tan(\phi) = \frac{e\sin(\nu)}{1 + e\cos(\nu)} \tag{2.21}$$

3. The velocity components v_x and v_y (for planet or star) are then computed first by computing the perpendicular version: $v_{px} = -yv/r$, $v_{py} = xv/r$, and then rotating this *clockwise* by the angle ϕ: $v_x = \cos(\phi)v_{px} + \sin(\phi)v_{py}$, $v_y = -\sin(\phi)v_{px} + \cos(\phi)v_{py}$.

This lecture provides a Python function `kepler()` that does all these things. You can find this function in `snippets/kepler.py`.

Exercise 22 (voluntary): Try to analyze the RV data of the object HD 118203 with a model of an eccentric planetary orbit. Your model parameter space will increase by at least 2 parameters.