AST1501 Notes

1. Coordinate Transformation

1.1. Stream to Equatorial Coordinates

$$\begin{bmatrix} \cos(\phi_1)\cos(\phi_2) \\ \sin(\phi_1)\cos(\phi_2) \\ \sin(\phi_2) \end{bmatrix} = \begin{bmatrix} -0.4776303088 & -0.1738432154 & 0.8611897727 \\ 0.510844589 & -0.8524449229 & 0.111245042 \\ 0.7147776536 & 0.493068392 & 0.4959603976 \end{bmatrix} \times \begin{bmatrix} \cos(\alpha)\cos(\delta) \\ \sin(\alpha)\cos(\delta) \\ \sin(\delta) \end{bmatrix} \quad (1)$$

So if we assume the left hand side matric is A and the others are B and C, respectively, we can get C matrix by:

$$\begin{aligned} A &= BC \\ B^{-1}A &= (B^{-1}B)C \\ B^{-1}A &= C, \end{aligned} \quad (2)$$

therefore, we get:

$$x = y + z \quad (3)$$

1.2. Cylindrical to Stream coordinates

Coversion from cylindrical coordinates (R, z, ϕ) to cartesian coordinates (x, y, z):

$$\begin{aligned} x &= R\cos(\phi) \\ y &= R\sin(\phi) \\ z &= z \end{aligned} \quad (4)$$

Conversion from cartesian coordinates (x, y, z) to equatorial coordinates (δ, α, d):

$$\begin{aligned} d &= \sqrt{x^2 + y^2 + z^2} \\ \delta &= \arcsin(\frac{z}{d}) \\ \alpha &= \arctan(\frac{y}{x}) \end{aligned} \quad (5)$$

Finally, conversion from equatorial (δ, α, d) to stream coordinates (ϕ_1, ϕ_2) can be done using equation 1.

2. Likelihood Procedure

$$\begin{aligned} L(\text{modelorbit}) &= P(V_c, q_\phi|\text{Data}) = & = \\ \prod_i P(\text{data}_i|\text{model}) &= \prod_i (P(\phi_{2,i}|\phi_{1,i}), P(D_i|\phi_{1,i}), P(Vrad_i|\phi_{1,i}), P(\mu_i|\phi_{1,i})) & (6) \end{aligned}$$

$$\ln(\mathcal{L}) = \sum_i P(\text{data}_i|\text{model}). \quad (7)$$

We can get the likelihood by computing this probablity. However, in Koposov et al. 2010, they marginalized over the oarameters rather than finding the likelihood. Marginalization can be done in the following way:

$$\text{P}(V_c, q_\phi|\text{data(D)}) = \int_7^{10} \text{P}(V_c, q_\phi|\text{D}, \text{R}_0)\text{P}(\text{R}_0)\text{dR}_0, \quad (8)$$

where D represents the data and $\text{P}(\text{R}_0)$ is given by:

$$\text{P}(\text{R}_0) = M(R_0|8.4, 0.42) = \frac{1}{2\pi} e^{\frac{-(R_0-8.4)^2}{2\times 0.42^2}}. \quad (9)$$

The value of our guess for R_0 being 8.4 with the error of 0.42 comes from Koposov et al. 2010. We should then do the same calculation with values of 8.2 for R_0 with the error of 0.1 .

We also need to optimize the initial points for calculating the orbit for each given V_c and q_ϕ. The steps for compouting the likelihood including the optimization is as follows:

1. Take an initial guess for (ϕ_1, ϕ_2) initial point (keep the ϕ_1 the same and change ϕ_2).
2. Convert the (ϕ_1, ϕ_2) to be in cylindrical coordintes
3. Compute potential and orbit for the specified V_c and q_ϕ
4. Find the likelihood value
5. Optimize orbit, which means maximize the the $\ln(\mathcal{L})$ (or minimize the χ^2 since $\chi^2 = -2\ln(\mathcal{L})$)
6. Get the initial position obtained from optimization
7. calculate the orbit with the obtained initial position
8. compute the likelihood from the calculated orbit
9. compute the marginalization

10. Do the above steps for each set of V_c and q_ϕ

11. Make a contour of likelihood

In order to optimize the likelihood, I should pass a function of one or more variables to the scipy.optimize() module. So I need to write a function of $\phi_2, \mathrm{d}, \mu_{\phi_1}, \mu_{\phi_2} \mathrm{and} V_c$ which does find the likelihood value for a set value of ϕ_1. Then I can pass this function to the optimizer function to get the parameter values that maximize the likelihood or minimize the χ^2.

I made my likelihood function faster by eliminating the for loop existed in the previous version. It now takes arrays of data and model values and computes the likelihood for each set of parameter (such as $\phi_2, V_{\mathrm{rad}} and \mu$ using numpy.tile() function. The numpy.tile() generates an array that has the array you pass to it repeated n times. So in order to have the calculations correct, I made an array as shown in equation (10):

$$\begin{bmatrix} \mathrm{data1} & \mathrm{data2} & \cdots & \mathrm{dataN} \\ \mathrm{data1} & \mathrm{data2} & \cdots & \mathrm{dataN} \\ \mathrm{data1} & \mathrm{data2} & \cdots & \mathrm{dataN} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{data1} & \mathrm{data2} & \cdots & \mathrm{dataN} \end{bmatrix}, \tag{10}$$

where the number of rows is equal to the length of model and the number of the columns is equal to the length of data.

Likewise, we can write down a similar matrix for the model values as in equation (11):

$$\begin{bmatrix} \mathrm{model\,1} & \mathrm{model\,1} & \cdots & \mathrm{model\,1} \\ \mathrm{model\,2} & \mathrm{model\,2} & \cdots & \mathrm{model\,2} \\ \mathrm{model\,3} & \mathrm{model\,3} & \cdots & \mathrm{model\,3} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{model\,N} & \mathrm{model\,N} & \cdots & \mathrm{model\,N} \end{bmatrix}, \tag{11}$$

where the number of rows is equal to the length of model and the number of columns is equal to the length of data. Then, in order for the matrices to have the same shape so that we will be able to do mathematical operations on them, we have to take the transpose of the model matrix. Finally, to compute the integral of each column in the final value of the tiled matrix, I used simps integration function including axis=0 as an argument so that it takes the integral of the columns and returns the values of the integra of each column as an array to avoid using for loops. Then we will be able to the likelihood calculations as normal.

3. General Notes

- The actions of stars in the cluster are not conserved (because the self-gravity of the cluster is important), but that the actions of stream members freeze once they are stripped.

- The angle difference between stars in a stream and the progenitor increases linearly with time.
- Computing the logarithm of the sum of exponentials can be handles in a more stable way in scipy using scipy.misc.logsumexp() function. This way we make sure we do not get underflows such as getting inf and nan as a result of using np.log(np.sum(np.exp(a))).

4. How to use NEMO and gyrfalcON

A set of examples for running NEMO and gyrfalcON is in here:

`https://github.com/jobovy/dyn-modeling-streams-2014/tree/master/sim`

4.1. NEMO

Below is an example run in NEMO:

```
mkking out=gd1.nemo nbody=10000 W0=2. mass=20000 r_t=0.07 WD_units=t
```

Here is the list of arguments used to run the above run NEMO:

1. mk + name_of_the_density_profile: For instance, mkking refers to King's profile and mkplummer refers to Plummer profile.
2. out= : specifies the output filename
3. nbody: number of particles in the simulation
4. W0: dimensionless central potential for King's model
5. mass: total mass of the system
6. r_t: tidal radius which is defined as the radius where the potential Ψ becomes 0 and the density vanishes. i.e it is the outermost limit of a cluster (Ψ is the potential taht we solve for in Poisson equations)
7. WD_units: that is, positions are specified in kpc, velocities in kpc/Gyr, times in Gyr, and G=1.

We can shift the position and velocities of the run forward in time to today:

```
snapshift input_file.nemo output_file_shifted.nemo rshift
    =-11.63337239,-10.631736273934635,-20.76235661 vshift
    =-128.8281653,79.172383882274971,42.88727925
```

The above command shifts to 12.4,1.5,7.1,107.0,-243.0,-105.0 (kpc, km/s, today) = -11.63337239,-20.76235661,-10.631736273934635,-128.8281653,42.88727925,79.172383882274971 (kpc, km/s, 5 Gyr ago)

Because of the definition of the logarithmic potential in NEMO, it cannot be flattened in z, so to use a flattened logarithmic potential, one has to flip y and z between galpy and NEMO (one can flatten in y). That is why the location of y and z and also vy and vz has been flipped.

We can then evolve the system using the following command:

```
gyrfalcON in=gd1_shifted.nemo out=gd1_evol.nemo tstop=5.125 eps=0.0015
    step=0.125 kmax=6 Nlev=10 fac=0.01 accname=LogPot accpars
    =0,48400.,0.,1.0,0.9
```

The parameters are:

1. in = shifted file as an input
2. out = output filename
3. tstop: stop time
4. eps: softening length
5. step: time step
6. kmax: the longest time step is taken to be $\tau = 2^{-\mathrm{kmax}}$
7. Nlev:
8. fac:These factors control the average time step of a body to be the minimum
9. accname: external acceleration field name (in this case LogPot means the logarithmic potential)
10. accparse: parameters used for the acceleration field specified in accname

We can then convert the file to a readable file using:

```
s2a filenname_evolved.nemo filename_evolved.dat
```

In order to use the output file in galpy, we need to flip y and z columns as well as vy and vz columns to get the correct values.

In order to get high time resolution in the simulation, we can decrease the value of "step" while keeping everything else the same.

To get a simulation with higher velocity dispersion, we increase the "eps" value (for instance we make it 10 times larger). With increasing the "eps" the mass and the tidal radius should also increase (why?!)

5. Finding initial parameters using optimization

Below is a list of parameters obtained from my optimization code as well as their true values and the percentage errors for the cases of 1D, 2D and 5D:

Table 1: 1D parameter varying for $V_c = 220$ km/s and q_ϕ=0.9

	Koposov value	Obtained value	percentage error
varying only ϕ_2 (deg)	-1.9964	-1.8808	5.8
varying only D (kpc)	16.50467	16.5146	0.06
varying only μ_{ϕ_1}	0.9794	0.9809	0.15
varying only μ_{ϕ_2}	-2.9904	-3.0053	0.5
varying only $V_{\rm rad}$	-78.6004	-78.2320	0.47

Table 2: 2D, two parameters varying at the same time

	Koposov ϕ_2	Obtained ϕ_2	% error ϕ_2	Koposov	Obtained	% error
ϕ_2 and D (kpc) varying	-1.9964	-2.2448	12.5	16.5047	16.5350	0.18
ϕ_2 and μ_{ϕ_1} varying	-1.9964	-2.1468	7.5	0.9794	0.9829	0.36
ϕ_2 and μ_{ϕ_2} varying	-1.9964	-1.8949	5.1	-2.9904	-2.9945	0.13
ϕ_2 and $V_{\rm rad}$ varying	-1.9964	-2.3997	20.2	-78.6004	-77.0369	1.9

Table 3: 5D, all parameters varying

	Koposov value	Obtained value	percentage error
ϕ_2 (deg)	-1.9964	-2.64076829	32.3
D (kpc)	16.50467	16.6010	0.58
μ_{ϕ_1}	0.9794	0.9673	1.2
μ_{ϕ_2}	-2.9904	-2.9635	0.9
$V_{\rm rad}$	-78.6004	-76.7780	2.3

5.1. Computing $\rho(r)$ from $\psi(r)$ for King Model

We have the Poisson equation for King's model as in the equation below:

$$\frac{d}{dr}\left(r^2\frac{d\Psi(r)}{dr}\right) = -4\pi G\rho_1 r^2\left[e^{(\frac{\Psi(r)}{\sigma^2})}\text{erf}\left(\frac{\sqrt{\Psi(r)}}{\sigma}\right) - \sqrt{\frac{4\Psi(r)}{\pi\sigma^2}}\left(1+\frac{2\Psi(r)}{3\sigma^2}\right)\right] \tag{12}$$

We can expand the first term on the left hand side to get:

$$r^2\frac{d^2\Psi(r)}{dr^2} + 2r\frac{d\Psi(r)}{dr} = -4\pi G\rho_1 r^2\left[e^{(\frac{\Psi(r)}{\sigma^2})}\text{erf}\left(\frac{\sqrt{\Psi(r)}}{\sigma}\right) - \sqrt{\frac{4\Psi(r)}{\pi\sigma^2}}\left(1+\frac{2\Psi(r)}{3\sigma^2}\right)\right] \tag{13}$$

Once we have this equation, we can re-write the right hand side that depends on $\Psi(r)$ as a function $f(\Psi(r))$, so that we can make the above equation as a second order differential equation that can be solved numerically only. I solved this using scipy ode.int by making the equation into two first order equations by changing variables as is done below, where we set:

$$\begin{aligned} x_1(r) &= \Psi(r) \\ x_2(r) &= \dot{\Psi}(r) \end{aligned} \tag{14}$$

The initial condition are that $\frac{d\Psi(0)}{dr} = 0$ which makes $x_2(0) = 0$ and that $W_0 = \frac{\Psi(0)}{\sigma^2} = 3$ (or any other value we want) which makes $x_1(0) = 3\sigma^2$. Then, we can integrate $x_1(r)$ and $x_2(r)$ to get:

$$\begin{aligned} \dot{x_1}(r) &= \dot{\Psi}(r) = x_2(r) \\ \dot{x_2}(r) &= \ddot{\Psi}(r) = -4\pi G\rho_1 f(\Psi(r)), \end{aligned} \tag{15}$$

where:

$$f(\Psi(r)) = [e^{(\frac{\Psi(r)}{\sigma^2})}\mathrm{erf}\left(\frac{\sqrt{\Psi(r)}}{\sigma}\right) - \sqrt{\frac{4\Psi(r)}{\pi\sigma^2}}\left(1 + \frac{2\Psi(r)}{3\sigma^2}\right) - \frac{2}{r}\dot{\Psi}(r) \tag{16}$$

So, in the end our two first order differential equations would become (we substituted $\Psi(r)$ with $x_1(r)$ and $\dot{\Psi}(r)$ with $x_2(r)$:

$$\begin{aligned} \dot{x_1}(r) &= x_2(r) \\ \dot{x_2}(r) &= -4\pi G\rho_1 \left[e^{(\frac{x_1(r)}{\sigma^2})}\mathrm{erf}\left(\frac{\sqrt{x_1(r)}}{\sigma}\right) - \sqrt{\frac{4x_1(r)}{\pi\sigma^2}}\left(1 + \frac{2x_1(r)}{3\sigma^2}\right)\right] - \frac{2}{r}x_2(r) \end{aligned} \tag{17}$$

We can pass these two final differential equations to scipy to numerically integrate and give the results back.

Once we have the values for $\Psi(r)$ from scipy, we can interpolate the values to get $\Psi(r)$ as a function of r.

5.2. Probability Calculation

Here are a few probability rules that we will use during the calculations:

$$P(A, B) = P(A \cap B) = P(A|B).P(B) \tag{18}$$

$$P(A|B) = \frac{P(B|A).P(A)}{P(B)} \tag{19}$$

$$P(A|U) = \frac{P(A, U)}{P(U)} \tag{20}$$

$$\begin{aligned} P(B,C) &= P(B|C).P(C) \\ P(A,B,C) &= P(A).P(B|A).P(C|A,B) \end{aligned} \tag{21}$$

Now, assuming $U = (B, C)$ and using the above equation we can write $P(A|B, C)$ as:

$$P(A|B,C) = \frac{P(A,(B,C))}{P(B,C)} = \frac{P(A\ capB \cap C)}{P(B,C)} = \frac{P(A).P(B|A).P(C|A,B)}{P(B|C).P(C)} \tag{22}$$