Saas Fee School 2019 – Gaia tutorials

Jason Sanders

January 31, 2019

I provide a number of worked examples of using the Gaia data to illustrate some of the points made in the 'Fine structure of the stellar disc' lecture series. There are five tutorials that each demonstrate some basic data analysis technique using examples from Gaia. All of the tutorials use `python`.

The first section describes how to set up a virtual environment with the necessary packages. The examples obviously require data. We will show how to download Gaia data using the TAP protocol. For manipulating large datasets this becomes impractical. Therefore, we also will use locally stored data. These are available on USB sticks at the school (SF19_1, SF19_2) or can be downloaded from the provided sources.

Preliminaries

For these tutorials we will use Python 3.7 (available from https://www.python.org/downloads/). We first set up an environment using `conda` (if `conda` not installed, download miniconda from https://conda.io/miniconda.html)

```
conda create -n saas_fee_py python=3.7 anaconda
```

This sets up an environment distinct from your regular python installation. This is useful as we can edit and modify this environment independent of other projects. We use the environment using

```
source activate saas_fee_py
```

Then we want to install two further packages. First `galpy`. `galpy` requires `gsl`. The easiest way to do this is (installs a pre-compiled version)

```
conda install galpy -c conda-forge
```

Second `healpy`,

```
conda install healpy -c conda-forge
```

and then `astroquery`, `emcee`, `corner`.

```
pip install astroquery emcee corner
```

This should give you the tools you need to look at and analyse the Gaia data, as well as follow all of the examples given here. Now inside the folder with the notebooks, run

```
jupyter notebook
```

and you will see the available notebooks in the browser.

You can also run notebooks remotely. This is useful when working with large files and you only have a laptop available. First, run

```
jupyter notebook --no-browser
```

on the computer that will run the notebooks. Note which port the notebook is using (likely 8888). Then on your computer set up an ssh tunnel to that computer like

```
ssh -N -f -L localhost:8080:localhost:port_on_remote_compute name_of_remote_computer.
```

Now open the page `localhost:8080` in the browser.

Tutorials

There are five tutorials.

- Tutorial 1: the vertical density profile – simple accessing of Gaia data and reproducing thin-thick disc vertical density profile
- Tutorial 2: Coordinates and orbits – using astropy and galpy to do coordinate transformation and integrate orbits. Illustrated with dwarf spheroidal galaxy orbits.
- Tutorial 3: Actions, angles and frequencies – using galpy to find these coordinates, reproducing local substructure and the selection effects.
- Tutorial 4: Dissecting local chemo-kinematic structure – cross-matching the Gaia data with spectroscopic catalogues to use chemical abundances. Splitting velocity structure based on metallicity.
- Tutorial 5: Generalized Oort constants – fitting a model to data using emcee. Measure the generalized Oort constants from local proper motion and radial velocity data.

The notebooks are self-contained. They are repeated here in pdf form.

Tutorial1

January 31, 2019

1 Tutorial 1: The vertical density profile

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        warnings.filterwarnings('ignore')

        %matplotlib inline
```

In this tutorial we will inspect the vertical density profile of the Milky Way. This will illustrate basic access of the Gaia data.

We use the TAP functionality in astroquery to collect Gaia DR2 data. This runs an ADQL query remotely and retrieves the data -- ADQL is like SQL with added functions to specifically querying astronomical datasets). You will get lots of warnings as astropy doesn't like the units used by Gaia -- I have switched these off with the warnings package. This query gets the top 100 stars with positive parallaxes and parallax/parallax_error>3. You can replace the ADQL query with your own ADQL query (see https://gea.esac.esa.int/archive-help/adql/examples/index.html for some examples of Gaia ADQL queries.

To get the names of the columns, we have to look here https://gea.esac.esa.int/archive/documentation/GDR2/Gaia_archive/chap_datamodel/

```
In [2]: from astroquery.gaia import Gaia
        job = Gaia.launch_job_async(
            "select top 100 * from gaiadr2.gaia_source "+
            "where parallax>0 and parallax_over_error>3. ") # Select `good' parallaxes

        table = job.get_results()
        print(table)
```

```
Created TAP+ (v1.0.1) - Connection:
        Host: gea.esac.esa.int
        Use HTTPS: False
        Port: 80
        SSL Port: 443
Query finished.
    solution_id             designation          ... epoch_photometry_url
                                                 ...
```

```
------------------- ---------------------------- ... -------------------
1635721458409799680 Gaia DR2 4918097648274495232 ...                  --
1635721458409799680 Gaia DR2 4917739271907207808 ...                  --
1635721458409799680 Gaia DR2 4918122520428945152 ...                  --
1635721458409799680 Gaia DR2 4917647810580457088 ...                  --
1635721458409799680 Gaia DR2 4918005358017184640 ...                  --
1635721458409799680 Gaia DR2 4917927911167389568 ...                  --
1635721458409799680 Gaia DR2 4917945468993221248 ...                  --
1635721458409799680 Gaia DR2 6683370266818726656 ...                  --
1635721458409799680 Gaia DR2 6683395796104579584 ...                  --
1635721458409799680 Gaia DR2 6683442113032077056 ...                  --
                ...                          ... ...                 ...
1635721458409799680 Gaia DR2 5040960371801363456 ...                  --
1635721458409799680 Gaia DR2 5041251226986310016 ...                  --
1635721458409799680 Gaia DR2 5041279505051422976 ...                  --
1635721458409799680 Gaia DR2 5041180033608649600 ...                  --
1635721458409799680 Gaia DR2 1977874809671653120 ...                  --
1635721458409799680 Gaia DR2 4090282078290887040 ...                  --
1635721458409799680 Gaia DR2 4090282005267633408 ...                  --
1635721458409799680 Gaia DR2 2092048543773473280 ...                  --
1635721458409799680 Gaia DR2 2091982671854169856 ...                  --
1635721458409799680 Gaia DR2 2092029680276575616 ...                  --
Length = 100 rows
```

```
In [3]: print(table.colnames)
```

```
['solution_id', 'designation', 'source_id', 'random_index', 'ref_epoch', 'ra', 'ra_error', 'dec'
```

We can access individual columns like this. Note the units.

```
In [4]: print(table['pmra'])
```

```
       pmra
      mas / yr
-------------------
-4.6941335793129895
  0.658964757014354
  9.345594944921816
 30.304557672810176
 12.931525689919681
  30.59712862858142
  3.345527098756507
-0.7851807922221252
  6.935896681096559
 -4.655488202353315
                ...
 -1.987250904276051
```

```
 7.389922975948233
 32.72382063517125
-6.788283763279843
-4.686360036756701
-2.913297957272082
0.9979362894872922
-5.918008814146078
1.2858349469844021
-1.1818414171791185
Length = 100 rows
```

We will now query some main sequence stars around the North Galactic Pole (Galactic latitude greater than 85 deg). The ADQL language implements various mathematical functions. Here we are using log10. We require good parallaxes (not negative and with small errors -- this might bias our sample in funny ways but for now it is good enough). We select a small range in colours $0.65 < (G_{BP} - G_{RP}) < 0.85$ and ensure they are main sequence by cutting on absolute magnitude. We don't worry about extinction as this effect is small when looking straight out of the disc.

```
In [5]: from astroquery.gaia import Gaia
        job = Gaia.launch_job_async(
            """select top 3000 phot_g_mean_mag,parallax """
            """from gaiadr2.gaia_source """
            """where b>85. """                                  # Around North Galactic Pole
            """and parallax>0 and parallax_over_error>3. """# Select `good' parallaxes
            """and bp_rp>0.65 and bp_rp<0.85 """      # Select upper main sequence colours
            """and phot_g_mean_mag-5*log10(100./parallax) < 6.""") # Remove giant contaminants

        table = job.get_results();

Query finished.
```

Another way to do this is using the ADQL built-in geometric functions. POINT is a function describing a point on the sky and BOX, POLYGON, CIRCLE are all functions that describe a region on the sky. Typically they are used in conjunction with CONTAINS. This will return True (1) is POINT is CONTAINed in CIRCLE. In theory these should work with Galactic coordinates but I can't get this to work. We therefore require the coordinates of the NGP, using astropy. In theory, using ra, dec is faster as the Gaia archive has been indexed on these entries.

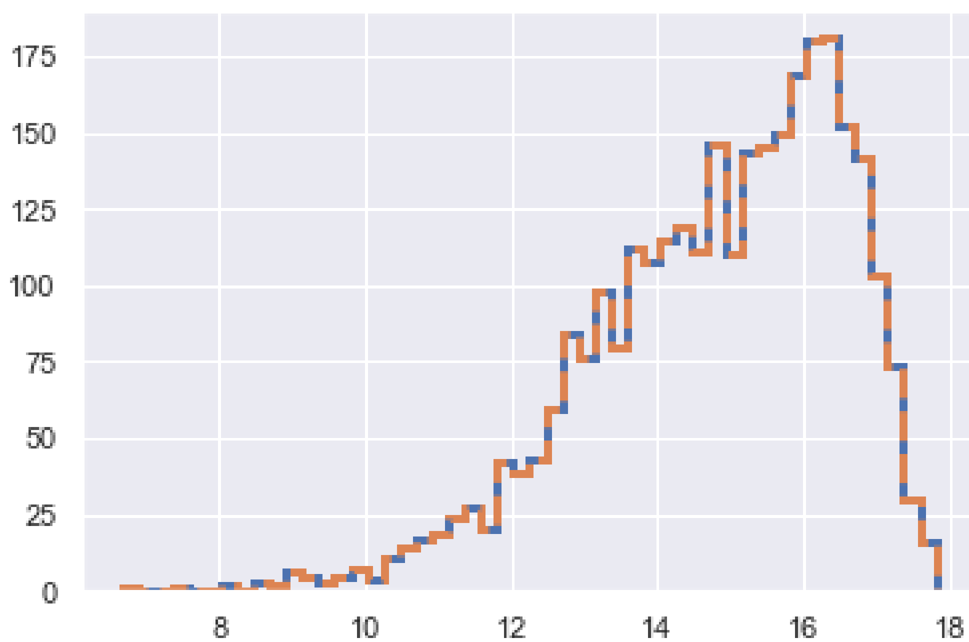
```
In [6]: from astropy.coordinates import SkyCoord, ICRS
        import astropy.units as u
        SkyCoord(l=300.*u.deg,b=90.*u.deg,frame='galactic').transform_to(ICRS)

Out[6]: <SkyCoord (ICRS): (ra, dec) in deg
            (192.85947789, 27.12825241)>
```

We then write the above query as

```
In [7]: from astroquery.gaia import Gaia
        job = Gaia.launch_job_async(
            """select top 3000 phot_g_mean_mag,parallax """
            """from gaiadr2.gaia_source """
            """where 1=CONTAINS(POINT('ICRS',ra,dec),CIRCLE('ICRS',192.85947789, 27.12825241,5))
            """and parallax>0 and parallax_over_error>3. """# Select `good' parallaxes
            """and bp_rp>0.65 and bp_rp<0.85 """     # Select upper main sequence colours
            """and phot_g_mean_mag-5*log10(100./parallax) < 6.""") # Remove giant contaminants

        table2 = job.get_results();
```

```
Query finished.
```

```
In [8]: plt.hist(table2['phot_g_mean_mag'],histtype='step',lw=3,bins=50);
        plt.hist(table['phot_g_mean_mag'],histtype='step',lw=3,bins=50,ls='dashed');
```

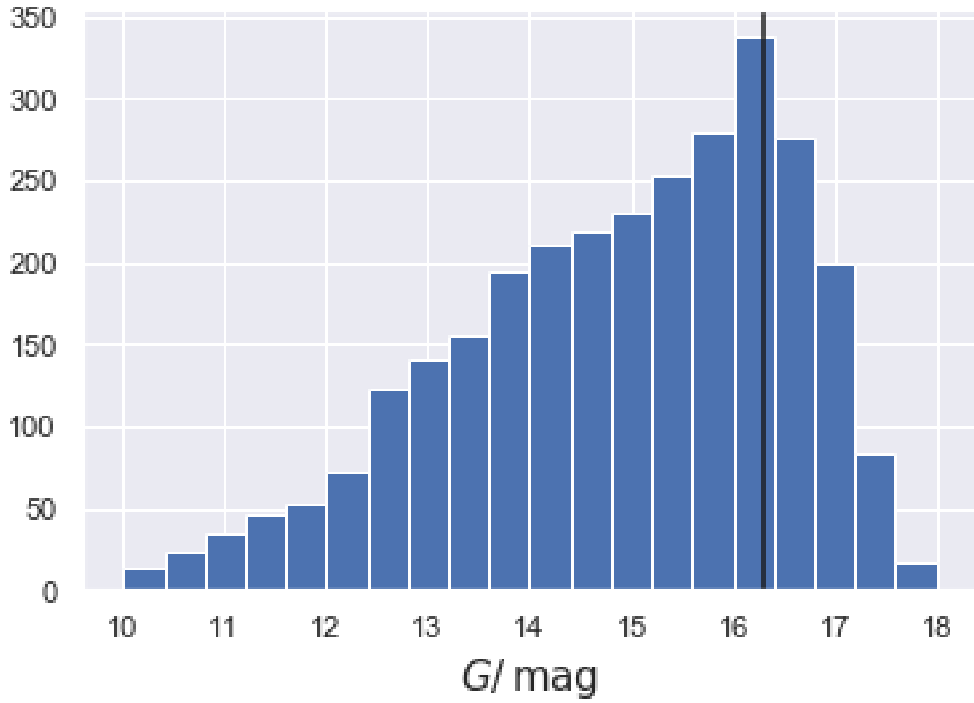


We have to worry about completeness. Gaia only observes down to a limiting magnitude. We can see in the magnitude distribution a truncation at around 16.3 mag. For our selected main sequence stars (with median magnitude 4.3) this corresponds to a limiting distance of 2.5 kpc.

```
In [9]: limiting_mag = 16.3 #
        plt.axvline(limiting_mag,color='k')
        plt.hist(table['phot_g_mean_mag'],range=[10.,18.],bins=20)
        plt.xlabel(r'$G/\,\mathrm{mag}$')
```

```
Mv = np.median(table['phot_g_mean_mag']-5.*np.log10(100./table['parallax']))
limiting_distance = 10.**(0.2*(-Mv+limiting_mag)-2.)
print('Limiting mag=',limiting_mag,'mag')
print('Median absolute mag=',Mv,'mag')
print('Limiting distance=',limiting_distance,'kpc')
```

```
Limiting mag= 16.3 mag
Median absolute mag= 4.2873335323882 mag
Limiting distance= 2.5265813933891805 kpc
```



We have taken a sample in a cone. As such, there is a larger volume over which we observe distant stars than the volume over which we observe nearby stars. To compute the density with Galactic height we require a Jacobian factor. we denote distance s and parallax ϖ.

$n(s) = s^2\rho(s)$

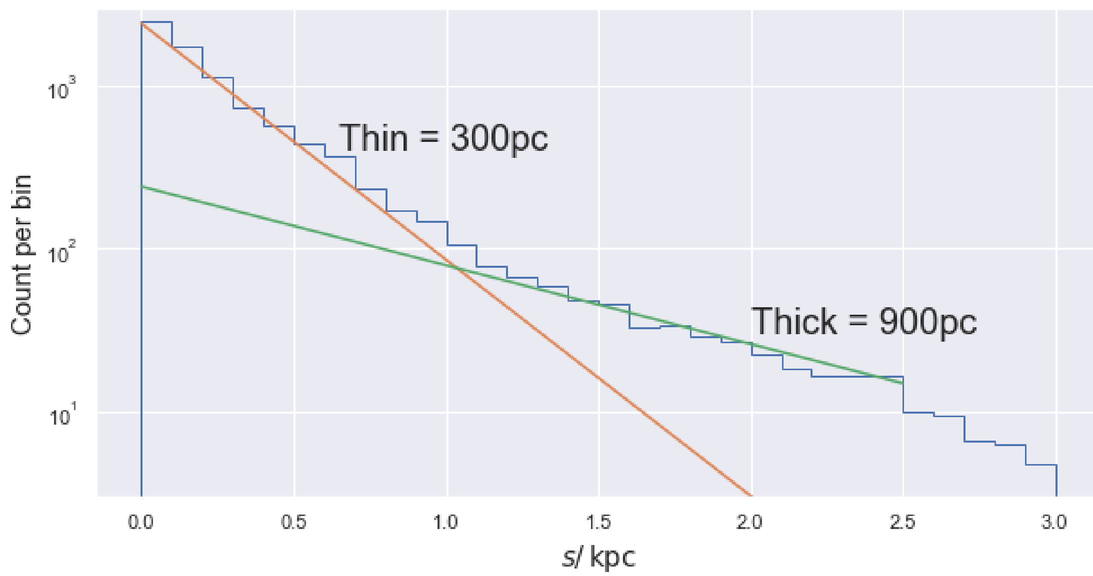
so $\rho(s) = \varpi^2 n(1/\varpi)$.

We have overplotted the classical thin/thick disc scaleheight exponentials. Note how beyond $s = 2.5\,\mathrm{kpc}$ incompleteness is an issue.

```
In [10]: fig = plt.figure(figsize=[10.,5.])
         xx = np.linspace(0.,2.5)
         nbins=30
         plt.hist(1./table['parallax'],range=[0.,3.],bins=nbins,
                  weights=table['parallax']**2,histtype='step');
```

```
plt.ylim(0.001*len(table),1.*len(table))

thin_disc_scale_height=0.3
thick_disc_scale_height=0.9
scaling = len(table)/(1.*nbins)

plt.plot(xx,24.*scaling*np.exp(-xx/thin_disc_scale_height))
plt.plot(xx,2.4*scaling*np.exp(-xx/thick_disc_scale_height))

plt.semilogy()
plt.xlabel(r'$s/\,\mathrm{kpc}$')
plt.ylabel(r'Count per bin')
plt.annotate('Thick = 900pc', xy=(2.,.3*scaling),fontsize=20)
plt.annotate('Thin = 300pc', xy=(.65,4.*scaling),fontsize=20)
```

```
Out[10]: Text(0.65, 400.0, 'Thin = 300pc')
```



We have confirmed the Gilmore-Reid result for the thin-thick vertical structure of the disc (not quite as we were looking at the North Galactic Pole). The original result used photometric distances so it is not trivial to reproduce this result. Other things to do would be

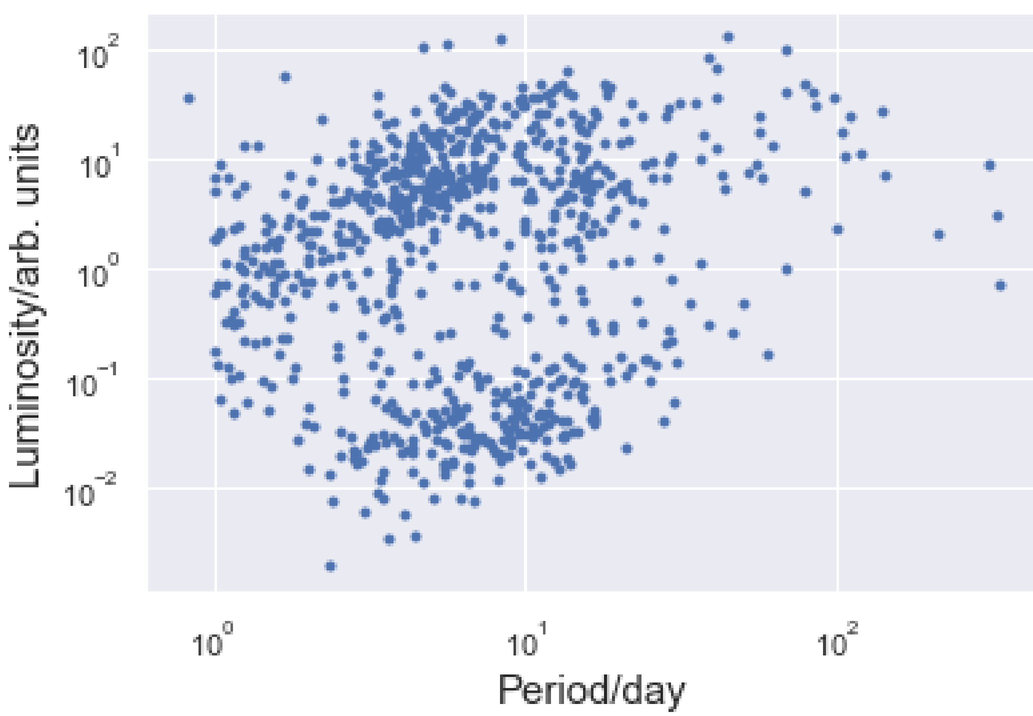
1. Look at different samples (other main sequence cuts, giants, etc. -- see Bovy 2017 TGAS paper),
2. Look away from North Pole (can we map the full density field in parallaxes? -- reproducing Juric et al. 2008 but over whole sky)
3. Look at perturbations relative to the smooth model (see Bennett and Bovy, 2018).

1.1 Joining with other tables

In addition to data in the Gaia DR2 source table, we might require additional data. Here we will grab results from the variability catalogue.

```
In [11]: from astroquery.gaia import Gaia
         job = Gaia.launch_job_async(
             """select top 1000 power(10.,-0.4*(phot_g_mean_mag-5.*log10(100./parallax))) as lum
             """from gaiadr2.gaia_source as G, gaiadr2.vari_cepheid as V where G.source_id=V.sou
             """and parallax_over_error>2."""
         )
         table3 = job.get_results();
         plt.plot(table3['period'],table3['lum'],'.')
         plt.semilogy()
         plt.semilogx()
         plt.xlabel('Period/day')
         plt.ylabel('Luminosity/arb. units');
```

```
Query finished.
```



1.2 Computing aggregate quantities

We can also compute aggregate quantities. Here we compute the mean, standard deviation (implemented specifically for the Gaia TAP+ service) and count for stars within 0.1 deg of the NGP.

```
In [12]: from astroquery.gaia import Gaia
         job = Gaia.launch_job_async(
             """select avg(bp_rp), stddev(bp_rp), count(*) """
             """from gaiadr2.gaia_source """
             """where 1=CONTAINS(POINT('ICRS',ra,dec),CIRCLE('ICRS',192.85947789, 27.12825241,0.
         table3 = job.get_results();
         print(table3)
```

```
Query finished.
       avg                stddev        count_all
------------------ ------------------ ---------
1.3012299208805478 0.6671705337696868        68
```

We can also compute averages binned by another column.

```
In [13]: from astroquery.gaia import Gaia
         job = Gaia.launch_job_async(
             """select avg(bp_rp) as mean_bprp, avg(pmra) as meanpmra, """
             """stddev(pmra) as std_pmra, count(*), round(bp_rp*5) as bin """
             """from gaiadr2.gaia_source """
             """where 1=CONTAINS(POINT('ICRS',ra,dec),CIRCLE('ICRS',192.85947789, 27.12825241,0.
             """group by bin order by bin"""
         )
         table3 = job.get_results();
         print(table3)
```

```
Query finished.
     mean_bprp            meanpmra            std_pmra       count_all bin
------------------- ------------------- ------------------ --------- ----
 0.2925529479980469  0.9967250993514118                 --         1  1.0
0.39504661560058596 -1.0437391618924852    1.3985565936889         5  2.0
 0.6297840118408203   -9.72065792224543 13.901853618271293         5  3.0
 0.8187069459394976  -5.403940264957905  9.735770728648967        11  4.0
 1.0233988080705916  -4.155133222253669  8.532296573983963         7  5.0
 1.2054602305094402 -17.024689929693437 3.9024275474255505         3  6.0
 1.4590363502502441 -12.840884089086014 16.691582183256596         4  7.0
 1.6041273389543806  -5.568259282869979  7.946967494249695         7  8.0
 1.8112707138061523  4.0610378898025985  7.313314913581323         2  9.0
 1.9614383697509765 -14.223314508195571  17.11847421314836         5 10.0
  2.191007614135742    4.63094220935661  4.932914063852677         2 11.0
 2.4071226119995117 -11.468381741555222  7.447031720971797         2 12.0
  2.539379119873047   6.870534525327603  6.545338752799496         2 13.0
  2.749445915222168   5.748151886686594  5.572400805195707         2 14.0
                 -- -14.871813449918099                 --        10   --
```

1.3 Using Vizier

We can also use the TAP Vizier service. This is useful for cross-matching to other catalogues. Although, there are many cross-matches available in the Gaia arcive anyway.

```
In [14]: from astroquery.utils.tap.core import TapPlus
         tap = TapPlus(url="http://TAPVizieR.u-strasbg.fr/TAPVizieR/tap")
         tables = tap.load_tables()
```

```
Created TAP+ (v1.0.1) - Connection:
        Host: TAPVizieR.u-strasbg.fr
        Use HTTPS: False
        Port: 80
        SSL Port: 443
Retrieving tables...
Parsing tables...
Done.
```

We have to find the name of the Gaia DR2 table in the Vizier database.

```
In [15]: for t in tables:
             if 'gaia' in t.name:
                 print(t.name)
```

```
"J/A+A/608/A148/origaia"
"I/337/gaia"
"I/345/gaia2"
"I/347/gaia2dis"
```

We now cross-match to LAMOST find Gaia stars within 1 arcsec (1/3600=0.00027777) of each LAMOST star.

```
In [16]: job = tap.launch_job_async('select top 10 * from "I/345/gaia2" as gaia, '+
                                    '"V/153/dr4" as lamost '+
                                   "where 1=CONTAINS(POINT('ICRS',gaia.ra,gaia.dec),"+
                                                 "CIRCLE('ICRS',lamost.raj2000,lamost.dej2000
         tableV = job.get_results();
         print(tableV)
```

```
Query finished.
     designation              ra       ra_error ...   DE1deg  AssocData FileName
                             deg         mas    ...
---------------------- -------------- -------- ... --------- --------- --------
  Gaia DR2 34361129088 45.00431616421   0.1322 ...   0.02097      fits     fits
  Gaia DR2 34361129088 45.00431616421   0.1322 ...   0.02097      fits     fits
 Gaia DR2 549755818112 45.04828219159   0.0358 ...  0.048314      fits     fits
Gaia DR2 1275606125952 44.99327098082   0.0612 ...  0.076347      fits     fits
```

```
Gaia DR2 1340029955712  44.9690763768   0.1227 ... 0.0844669      fits     fits
Gaia DR2 1653563247744 44.99606023501   0.0644 ... 0.0849587      fits     fits
Gaia DR2 1653563247744 44.99606023501   0.0644 ...   0.08496      fits     fits
Gaia DR2 2546916445184 45.05702841559   0.1309 ... 0.1150102      fits     fits
Gaia DR2 2546916445184 45.05702841559   0.1309 ... 0.1150102      fits     fits
Gaia DR2 2851858288640 45.13214374785   0.0515 ... 0.1377617      fits     fits
```

```
In [17]: tableV[['ObsID','RAJ2000','ra']]
```

```
Out[17]: <Table masked=True length=10>
           ObsID    RAJ2000          ra
                      deg           deg
           int32    float64      float64
         --------- ---------- --------------
            215109   45.00422 45.00431616421
          18112111   45.00422 45.00431616421
         367412199  45.048267 45.04828219159
          18112179  44.993274 44.99327098082
         300702156 44.9690933  44.9690763768
            215101 44.9960101 44.99606023501
         367412179  44.996028 44.99606023501
         252907206 45.0569813 45.05702841559
         300702174 45.0569813 45.05702841559
         300002179 45.1321253 45.13214374785
```

1.4 Gaia RVS selection function

One other unique function provided in the Gaia archive is gaia_healpix_index which gives the healpix number of each Gaia source. Healpix are a method of dividing up the sphere into equal area pixels. This is useful as we can bin the catalogue by Healpix. In the following we count all sources in bins of G and healpix for a small region on the sky.

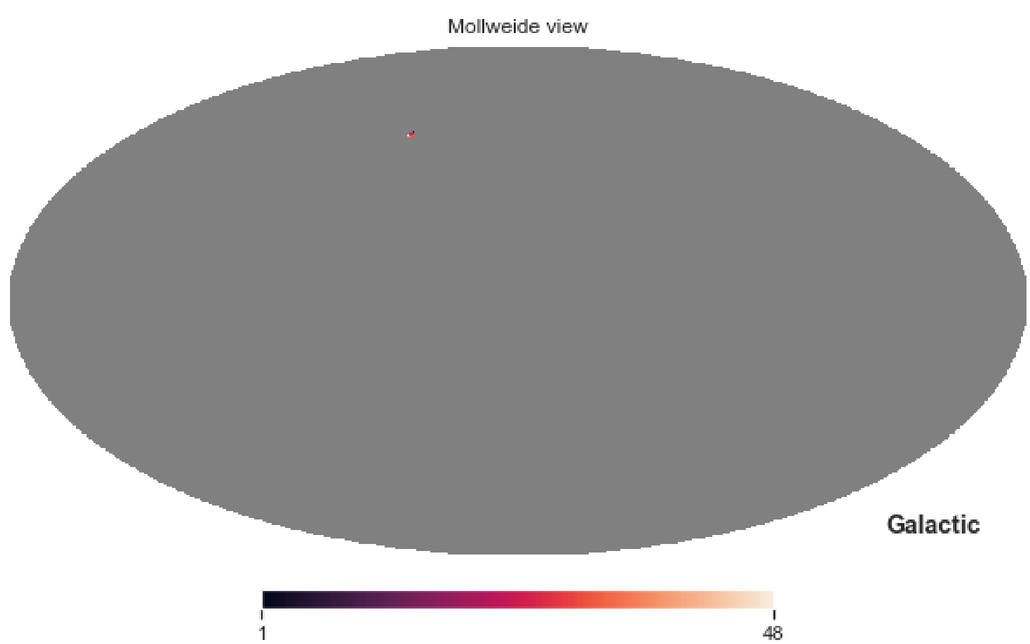
```
In [18]: from astroquery.gaia import Gaia
         job = Gaia.launch_job_async(
             """select gaia_healpix_index(8, source_id) AS hpx8, count(*), round(phot_g_mean_mag
             """from gaiadr2.gaia_source """
             """where 1=CONTAINS(POINT('ICRS',ra,dec),CIRCLE('ICRS',50,50, 1.)) """
             """ and phot_g_mean_mag>6. and phot_g_mean_mag<17. """
             """group by hpx8, G order by G"""
         )
         table3 = job.get_results();
         table3['gbin']=np.digitize(table3['g'],np.arange(6.,17.,0.5))
         print(np.unique(table3['g']))
```

```
Query finished.
 g
----
 7.0
```

```
 7.5
 8.0
 8.5
 9.0
 9.5
10.0
10.5
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0
15.5
16.0
16.5
17.0
```

We plot with healpy.

```
In [19]: import healpy as hp
         NSIDE = 256
         Gbin=20
         H = np.ones(hp.nside2npix(NSIDE))*np.nan
         H[table3['hpx8'][table3['gbin']==Gbin]]=table3['count_all'][table3['gbin']==Gbin]
         hp.mollview(H,nest=True,coord='G')
```



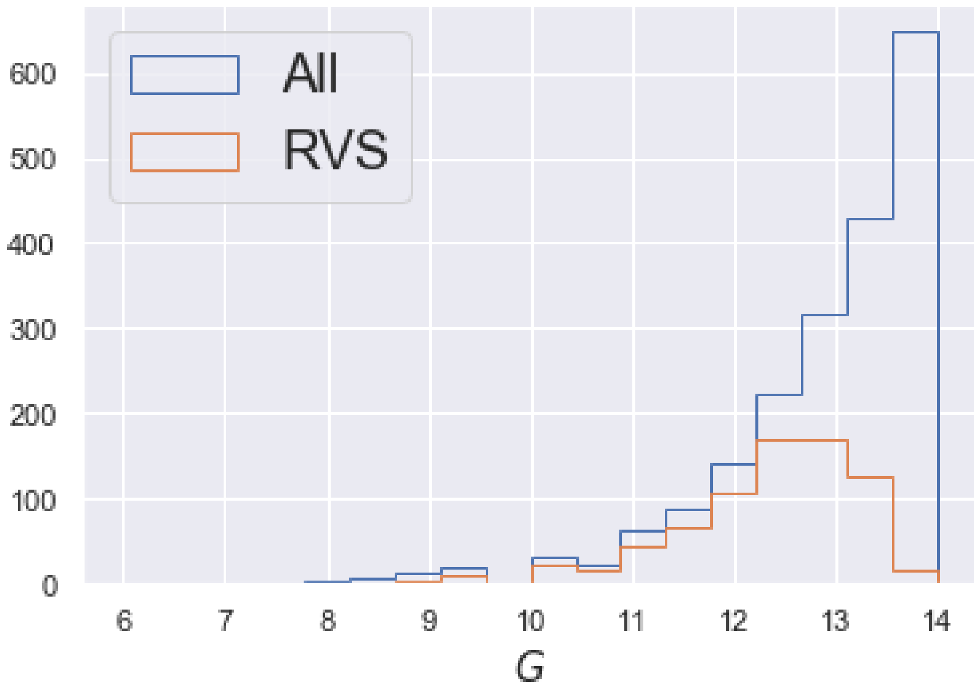
We perform the same calculation for those stars with radial velocities.

```
In [20]: from astroquery.gaia import Gaia
         job = Gaia.launch_job_async(
             """select gaia_healpix_index(8, source_id) AS hpx8, count(*), round(phot_g_mean_mag
             """from gaiadr2.gaia_source """
             """where 1=CONTAINS(POINT('ICRS',ra,dec),CIRCLE('ICRS',50,50, 1.)) """
             """ and phot_g_mean_mag>6. and phot_g_mean_mag<17. and radial_velocity>-1000 """
             """group by hpx8, G order by G"""
         )
         table3_RV = job.get_results();
         table3_RV['gbin']=np.digitize(table3_RV['g'],np.arange(6.,17.,0.5))
```

```
Query finished.
```

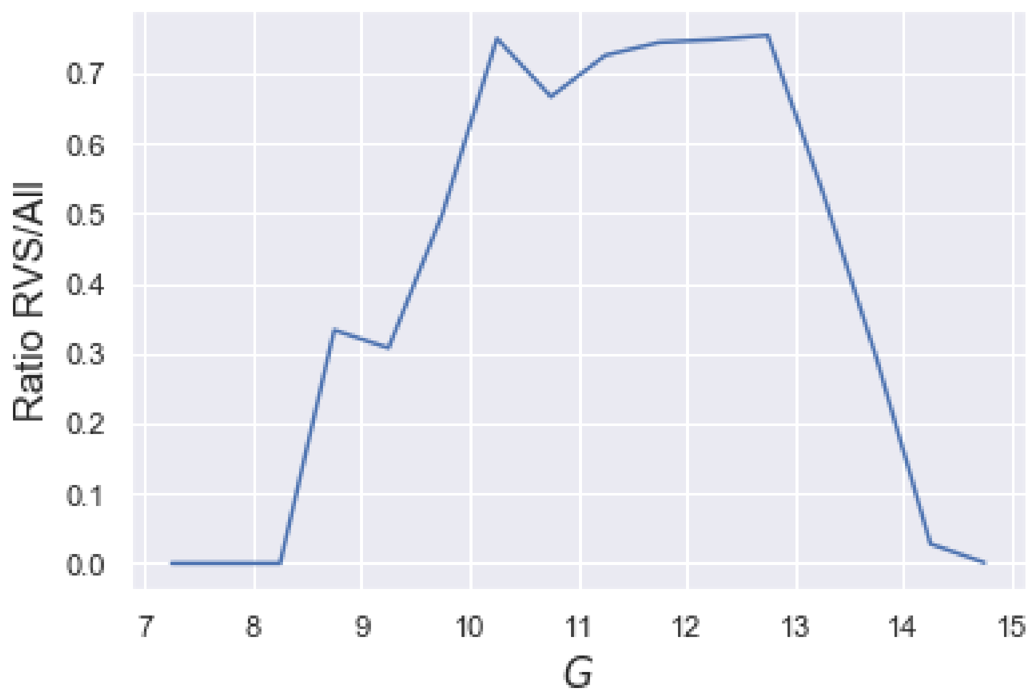
Plotting the ratio we see the approx. selection function of the RVS in this part of the sky. Approx. complete up to G=13 mag.

```
In [24]: plt.hist(table3['g'],range=[6.,14.],bins=18,histtype='step',label='All',weights=table3[
         plt.hist(table3_RV['g'],range=[6.,14.],bins=18,histtype='step',label='RVS',weights=tabl
         # plt.ylim(0.,80.)
         plt.xlabel(r'$G$')
         plt.legend(loc='upper left',fontsize=20);
```



```
In [26]: nn,bb = np.histogram(table3['g'],range=[6.,15.],bins=18,weights=table3['count_all'])
         nn2,bb = np.histogram(table3_RV['g'],bins=bb,weights=table3_RV['count_all'])
         plt.plot(.5*(bb[1:]+bb[:-1]),nn2/nn)

         plt.xlabel(r'$G$')
         plt.ylabel(r'Ratio RVS/All')
```

```
Out[26]: Text(0, 0.5, 'Ratio RVS/All')
```



2 Other methods

2.1 Reading downloaded VOT

Go to Gaia Archive (https://gea.esac.esa.int/archive/), query database, download vot (or other formats) and read in with astropy

```
In [ ]: from astropy.table import Table
        t = Table.read('/Users/jls/Downloads/15487447006230-result.vot')
        print(t)
```

2.2 Use TOPCAT

Download http://www.star.bris.ac.uk/~mbt/topcat/, query TAP services, etc.

```
In [ ]: from astroquery.simbad import Simbad
        result_table = Simbad.query_object("Hyades")
        print(SkyCoord(ra='{}h{}m{}s'.format(*result_table['RA'][0].split(' ')),
                       dec='{}d{}m{}s'.format(*result_table['DEC'][0].split(' '))))
```

Tutorial2

January 29, 2019

1 Tutorial 2: Coordinates and orbits

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from astropy.table import Table
        import urllib.request
        from astropy.coordinates import SkyCoord, Galactocentric
        import astropy.units as u

        %matplotlib inline
```

In this tutorial we will look at the orbits of the Milky Way dwarf spheroidal galaxies. Many of these had their proper motions robustly measured for the first time (Helmi et al. 2018, Simon 2018, Fritz et al. 2018, Massari et al. 2018). This illustrates conversion from observed coordinates (equatorial coordinates, proper motions, distances and radial velocities) to more physical coordinates with respect to the Galaxy. We will also use galpy's orbit functionality.

I have combined the Gaia proper motion measurements (from Fritz et al. 2018) with the McConnachie (2010) dwarf spheroidal galaxy catalogue (containing distances and radial velocities). It is available here: https://www.ast.cam.ac.uk/~jls/data/dwarf_spheroidal_data.fits . The table contains on-sky equatorial coordinates, distances (computed from distance moduli), radial velocities and proper motions. We have not provided uncertainties but in general these are significant.

```
In [2]: reqst = urllib.request.urlretrieve(
            "https://www.ast.cam.ac.uk/~jls/data/dwarf_spheroidal_data.fits",
            "dwarf_spheroidal_data.fits")
        data = Table.read("dwarf_spheroidal_data.fits")
        print(data)
```

```
satellite         ra          ...        pmra               pmdec
                  deg         ...      mas / yr           mas / yr
--------- ------------------ ... ------------------- -------------------
   Aqu II           338.4813 ...              -0.252               0.011
    Boo I            210.025 ... -0.5539999999999999              -1.111
   Boo II              209.5 ... -2.6860000000000004               -0.53
  Boo III              209.3 ...                 nan                 nan
 CanVen I 202.01458333333332 ...              -0.159              -0.067
CanVen II 194.29166666666663 ...              -0.342              -0.473
```

```
   Car I 100.40291666666668 ...               0.485               0.132
  Car II           114.1066 ...               1.867 0.08199999999999999
 Car III           114.6298 ... 3.0460000000000003               1.565
ComBer I  186.7458333333333 ...               0.471              -1.716
     ...                ... ...                 ...                 ...
   Sgr I          283.83125 ...              -2.736              -1.357
  Sgr II          298.16875 ...                 nan                 nan
  Tri II            33.3225 ...               0.588  0.5539999999999999
  Tuc II  342.9795833333333 ...               0.916              -1.169
 Tuc III             359.15 ...              -0.117 -1.7009999999999998
  Tuc IV               0.73 ...                 nan                 nan
 U Maj I             158.72 ... -0.6829999999999999              -0.72
U Maj II            132.875 ... 1.6909999999999998              -1.903
 U Min I 227.28541666666663 ...              -0.184 0.08199999999999999
   Wil 1           162.3375 ...               0.199              -1.342
Length = 43 rows
```

1.0.1 Galactocentric coordinates

We use astropy coordinates to find the Galactocentric positions and velocities. First, we initialize a SkyCoord object. We can then transform to any frame (here Galactocentric). Note that the output coordinates come complete with a description of the assumed solar position and velocity.

```
In [3]: skycoords = SkyCoord(ra=data['ra'],dec=data['dec'],
                             distance=data['distance'],
                             pm_ra_cosdec=data['pmra'],pm_dec=data['pmdec'],
                             radial_velocity=data['radial_velocity'])
```

```
In [4]: galcen = skycoords.transform_to(Galactocentric)
        print(galcen[0])
```

```
<SkyCoord (Galactocentric: galcen_coord=<ICRS Coordinate: (ra, dec) in deg
    (266.4051, -28.936175)>, galcen_distance=8.3 kpc, galcen_v_sun=(11.1, 232.24, 7.25) km / s,
    (28.4708803, 53.12710278, -86.07288446)
 (v_x, v_y, v_z) in km / s
    (91.67239149, 231.76259785, 130.25851467)>
```

1.0.2 Orbits

We now integrate orbits using galpy. We can either initialized the orbits using the SkyCoord objects.

```
In [5]: from galpy.potential import MWPotential2014
        from galpy.orbit import Orbit
        orbits = [Orbit(skycoords[ii]) for ii in range(len(skycoords))]
```

```
/Users/jls/anaconda/envs/saas_fee_py/lib/python3.7/site-packages/astropy/coordinates/baseframe.p
  'favor of `representation_type`', AstropyDeprecationWarning)
```

Or we can explicitly compute the required galpy quantities (with units) and then initialize explicitly. These differ slightly in their assumed solar position and velocity.

```
In [6]: # Compute quantities needed by galpy
        R = np.sqrt(galcen.x**2+galcen.y**2)
        vR = (galcen.v_x*galcen.x+galcen.v_y*galcen.y)/R
        vT = -(-galcen.v_x*galcen.y+galcen.v_y*galcen.x)/R
        phi = np.arctan2(galcen.x,galcen.y)+np.pi/2.*u.rad

        orbits2 = [Orbit(vxvv=[R[ii],vR[ii],vT[ii],
                               galcen.z[ii],galcen.v_z[ii],phi[ii]])
                   for ii in range(len(skycoords))]
```

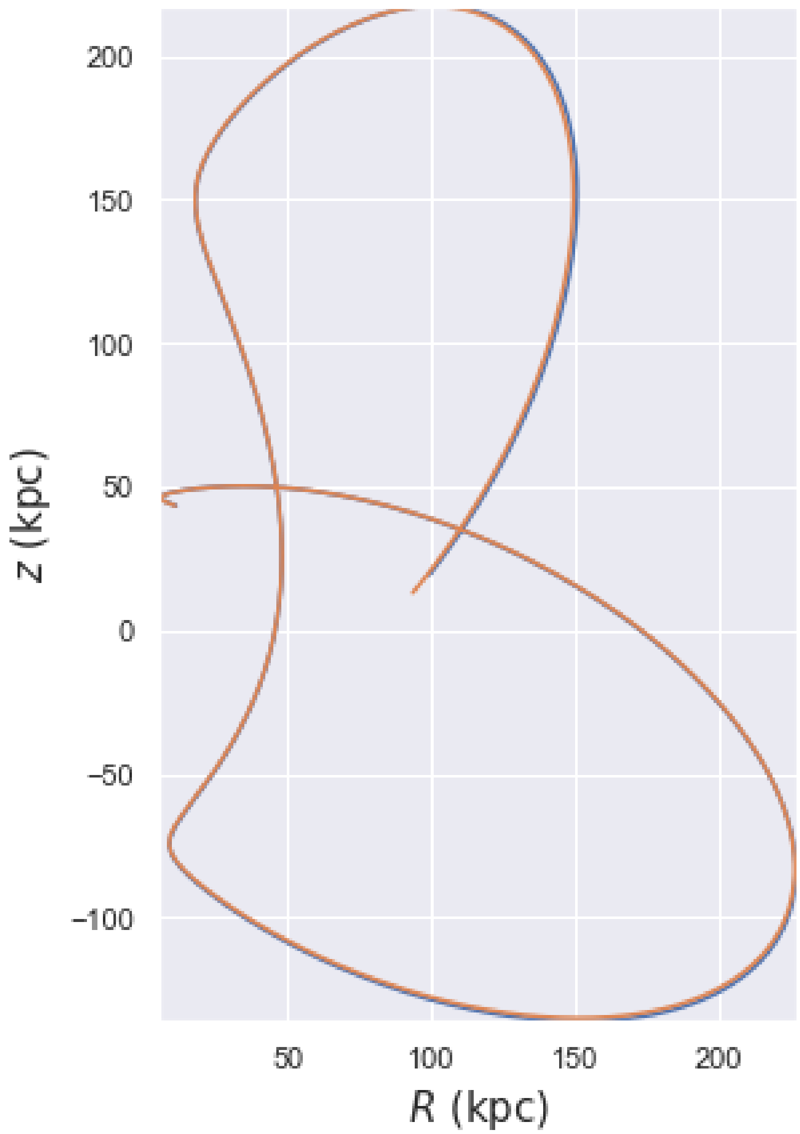
We integrate both sets of orbits forwards for 10 Gyr in the MWPotential2014

```
In [7]: times = np.linspace(0.,10.,500)*u.Gyr
        [o.integrate(t=times,pot=MWPotential2014) for o in orbits]
        [o2.integrate(t=times,pot=MWPotential2014) for o2 in orbits2];
```

We can inspect the individual orbits (e.g. Coma Berenices). Note the very small difference between the two orbits due to the two different conventions.

```
In [8]: ii=9
        plt.figure(figsize=[7.,7.])
        print(data['satellite'][ii])
        orbits2[ii].plot(gcf=True)
        orbits[ii].plot(overplot=True,color=sns.color_palette()[1]);
        plt.gca().set_aspect('equal')
```

```
ComBer I
```

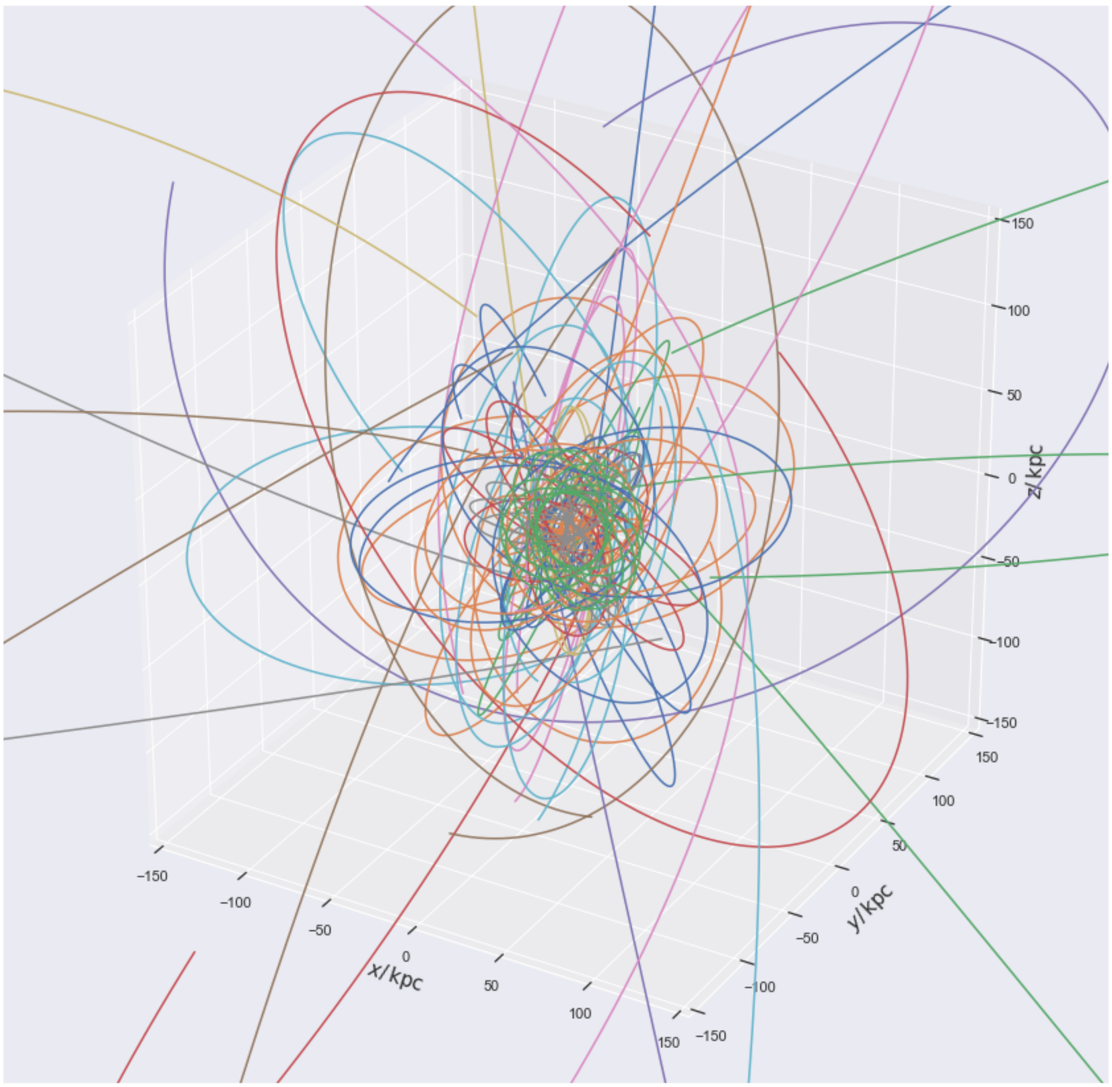


Now we plot all orbits in 3d.

```
In [9]: fig = plt.figure(figsize=[15.,15.])
        ax = fig.add_subplot(111, projection='3d')

        for ii in range(len(orbits)):
            orbits[ii].plot3d(overplot=True)

        ax.set_xlim(-150.,150.)
        ax.set_ylim(-150.,150.)
        ax.set_zlim(-150.,150.)
        ax.set_xlabel(r'$x/\,\mathrm{kpc}$')
```

```
    ax.set_ylabel(r'$y/\,\mathrm{kpc}$')
    ax.set_zlabel(r'$z/\,\mathrm{kpc}$');
```

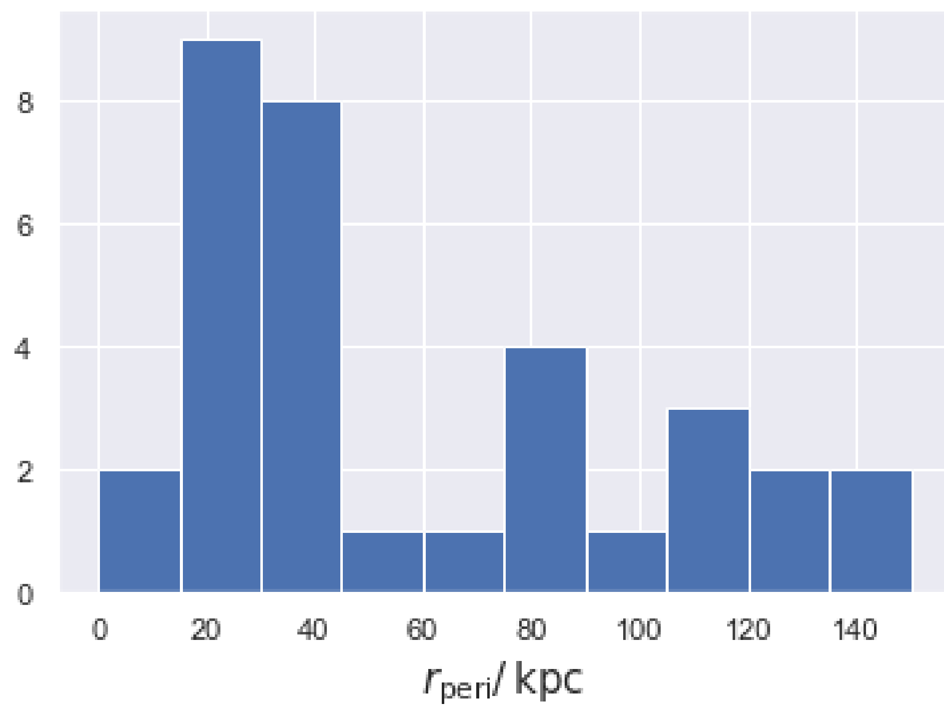
```
/Users/jls/anaconda/envs/saas_fee_py/lib/python3.7/site-packages/numpy/core/_methods.py:32: Runt
  return umr_minimum(a, axis, None, out, keepdims, initial)

/Users/jls/anaconda/envs/saas_fee_py/lib/python3.7/site-packages/numpy/core/_methods.py:28: Runt
  return umr_maximum(a, axis, None, out, keepdims, initial)
```



We can also inspect properties of the ensemble. Like the distribution of pericentric distances.

```
In [10]: plt.hist([o.rperi() for o in orbits],range=[0.,150.]);
         plt.xlabel(r'$r_\mathrm{peri}/\,\mathrm{kpc}$');
```

```
/Users/jls/anaconda/envs/saas_fee_py/lib/python3.7/site-packages/numpy/core/fromnumeric.py:83: R
  return ufunc.reduce(obj, axis, dtype, out, **passkwargs)

/Users/jls/anaconda/envs/saas_fee_py/lib/python3.7/site-packages/numpy/lib/histograms.py:754: Ru
  keep = (tmp_a >= first_edge)

/Users/jls/anaconda/envs/saas_fee_py/lib/python3.7/site-packages/numpy/lib/histograms.py:755: Ru
  keep &= (tmp_a <= last_edge)
```



We have inspected the orbits of the dwarf spheroidal galaxies in the Milky Way. The next things to do would be to look at:

1. Varying the potential
2. Calculating the uncertainties (Monte Carlo sampling)
3. Inspecting distributions of orbital quantities and their correlations

(see Fritz et al. 2018, Helmi et al. 2018).

Tutorial3

January 29, 2019

1 Tutorial 3: Actions, angles and frequencies

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import pandas as pd
        from astropy.table import Table
        from astropy.coordinates import SkyCoord, Distance, Galactocentric
        import astropy.units as u
        from galpy.potential import MWPotential2014
        from galpy.actionAngle import actionAngleStaeckel
        from galpy.orbit import Orbit

        import warnings
        warnings.filterwarnings('ignore')

        %matplotlib inline
```

In this tutorial we will demonstrate how to compute action, angle and frequency coordinates using the Gaia RVS sample. We begin by grabbing data via TAP. Here we limit ourselves to the first 1000 stars with radial velocities and proper motions and within 200 pc.

```
In [2]: from astroquery.gaia import Gaia
        job = Gaia.launch_job_async(
            """select top 3000 ra,dec,parallax,pmra,pmdec,radial_velocity """
            """from gaiadr2.gaia_source """
            """where radial_velocity>-1000 and pmra>-1000 and parallax>5""")
        table = job.get_results();
```

```
Created TAP+ (v1.0.1) - Connection:
        Host: gea.esac.esa.int
        Use HTTPS: False
        Port: 80
        SSL Port: 443
Query finished.
```

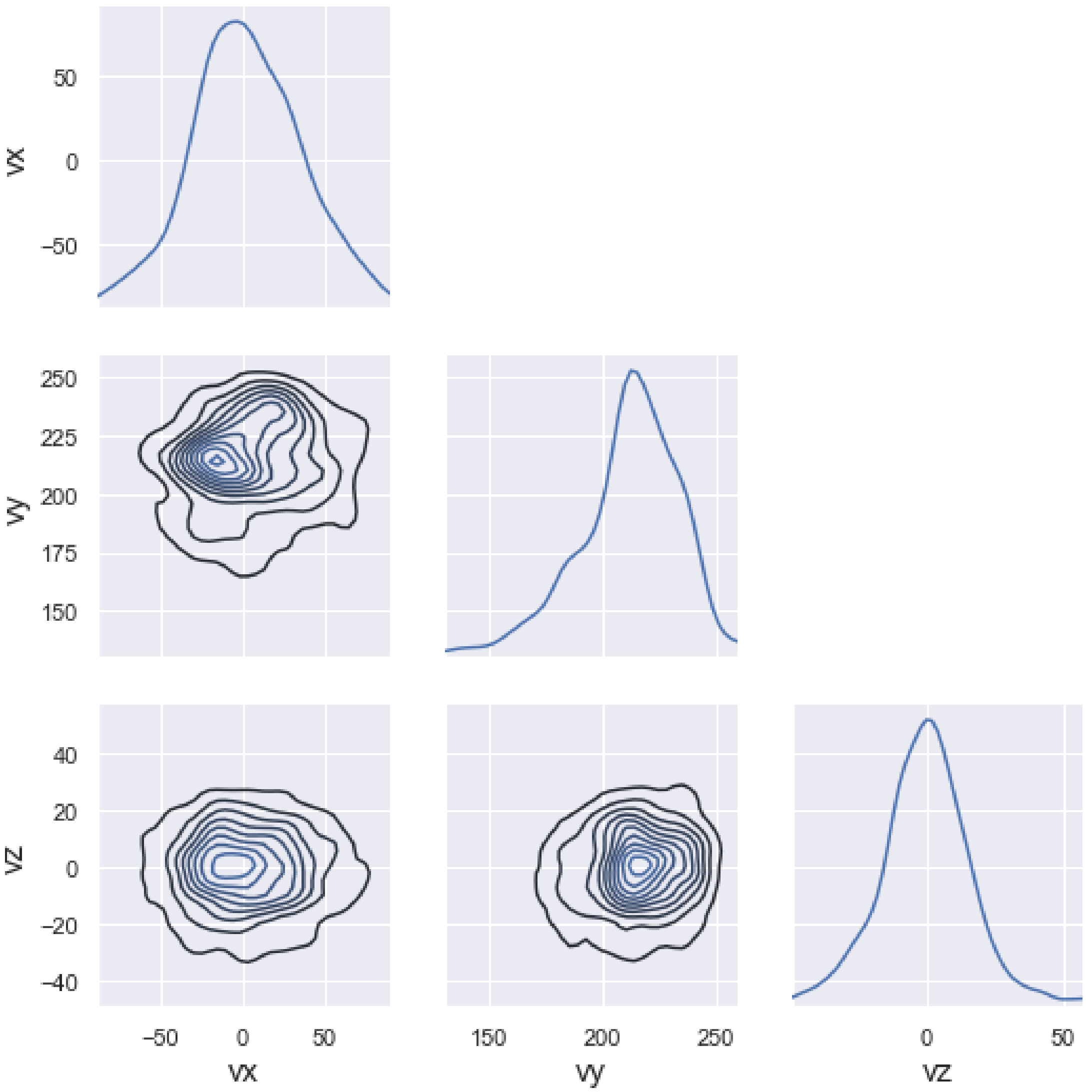
As in Tutorial 2, we construct SkyCoord objects. The difference here is we can construct a Distance object from parallaxes. In theory, the measured parallax could be negative (we have restricted ourselves to positive parallax), in which case we set allow_negative=True to propagate negative parallaxes through the calculation -- they get NaN distances.

In [3]:
```
# allow_negative means the distance=nan if parallax<0
s = SkyCoord(ra=table['ra'],dec=table['dec'],
             distance=Distance(
                 parallax=table['parallax'].data.data*u.mas,
                 allow_negative=True),
             pm_ra_cosdec=table['pmra'],pm_dec=table['pmdec'],
             radial_velocity=table['radial_velocity'])
galcen = s.transform_to(Galactocentric)
```

In [4]:
```
def pair_plot(data):
    g = sns.PairGrid(data, diag_sharey=False)
    g.map_lower(sns.kdeplot)
    g.map_diag(sns.kdeplot)
    for i, j in zip(*np.triu_indices_from(g.axes, 1)):
        g.axes[i, j].set_visible(False)
    for i, j in zip(*np.tril_indices_from(g.axes, 1)):
        g.axes[i,i].set_ylim(*np.percentile(data.values[:,i],[1.,99.]))
        g.axes[i,i].set_xlim(*np.percentile(data.values[:,j],[1.,99.]))
    for i in range(np.shape(g.axes)[0]):
        g.axes[i,i].set_xlim(*np.percentile(data.values[:,i],[1.,99.]))
```

In velocity space we see the characteristic structures. In particular, in vx against vy we see the Hyades, Hercules and Sirius very clearly.

In [5]:
```
pair_plot(pd.DataFrame(np.array([galcen.v_x,galcen.v_y,galcen.v_z]).T,
                       columns=['vx','vy','vz']))
```



1.0.1 Action, angles and frequencies

We now use the action-angle estimation routines in galpy. The standard method is the 'Staeckel fudge' which integrates the equations for the actions using those for a Staeckel potential. We use the MWPotential2014 from galpy. A choice of a parameter Δ is required for the method. We can use the SkyCoord object in the Orbit class so the coordinate transformations are handled internally. galpy outputs the actions and frequencies in its internal coordinates where distances in units of 8 kpc and velocities 220 km/s so we scale to physical.

In [6]:
```
# Initialize action-finder and compute actions, converting to physical units
aAS= actionAngleStaeckel(pot=MWPotential2014,delta=0.4,c=True)
action_unit = 8.*u.kpc * 220.*u.km/u.s
freq_unit = 1./(8.*u.kpc) * 220.*u.km/u.s
```

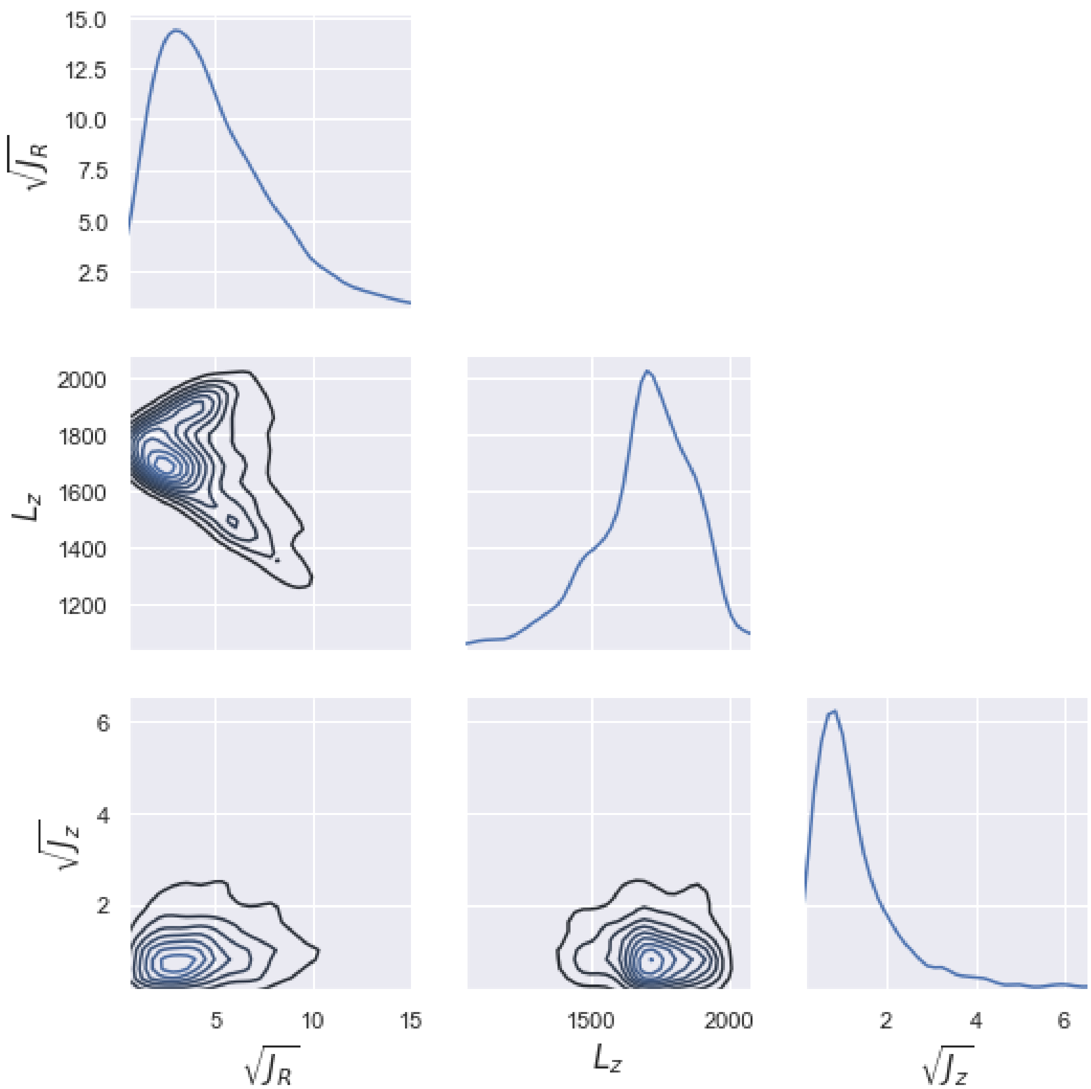
```
In [7]: o = Orbit(s)
        jj = aAS.actionsFreqsAngles(o.R()*u.kpc,
                                    o.vR()*u.km/u.s,
                                    o.vT()*u.km/u.s,
                                    o.z()*u.kpc,
                                    o.vz()*u.km/u.s,
                                    o.phi()*u.rad)

        actions = np.array(jj[:3])*action_unit
        frequencies = np.array(jj[3:6])*freq_unit
        angles = np.array(jj[6:])*u.rad
        angles[1]=angles[1]-2.*np.pi*u.rad*(angles[1]>np.pi*u.rad)

        # Convenient to work with sqrt(J_R) and sqrt(J_z)
        sqrt_actions = np.array([np.sqrt(actions[0]),actions[1],np.sqrt(actions[2])])
```

```
In [8]: actions[:,0], frequencies[:,0], angles[:,0]
```

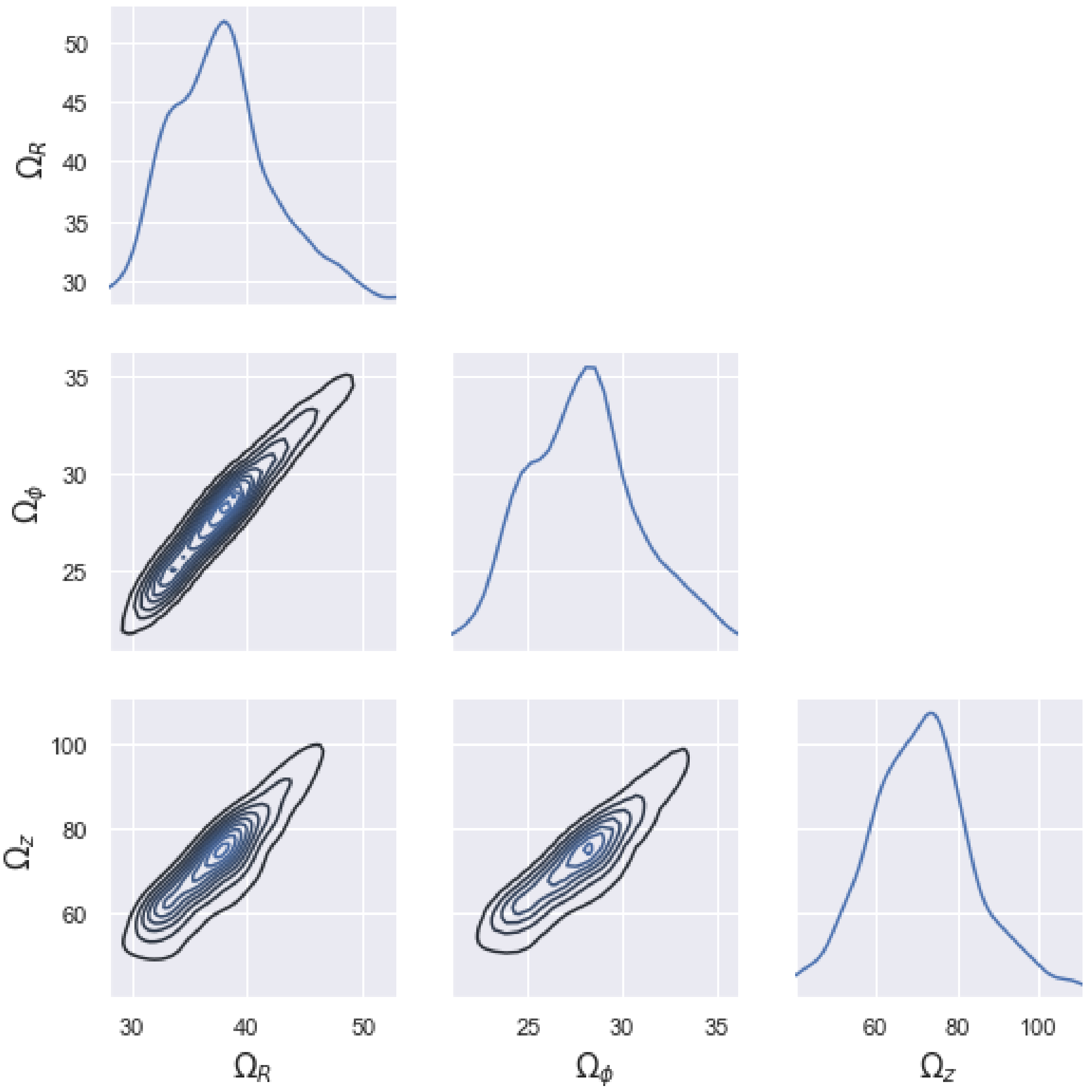
```
Out[8]: (<Quantity [7.33880794e+00, 1.84714611e+03, 4.11417501e-02] km kpc / s>,
         <Quantity [34.4022353 , 25.72916766, 67.73227455] km / (kpc s)>,
         <Quantity [5.81893182, 0.04455854, 3.63249353] rad>)
```

Inspecting the pair plots for these quantities we see the familiar features -- spurs due to the moving groups, edges due to the selection volume.

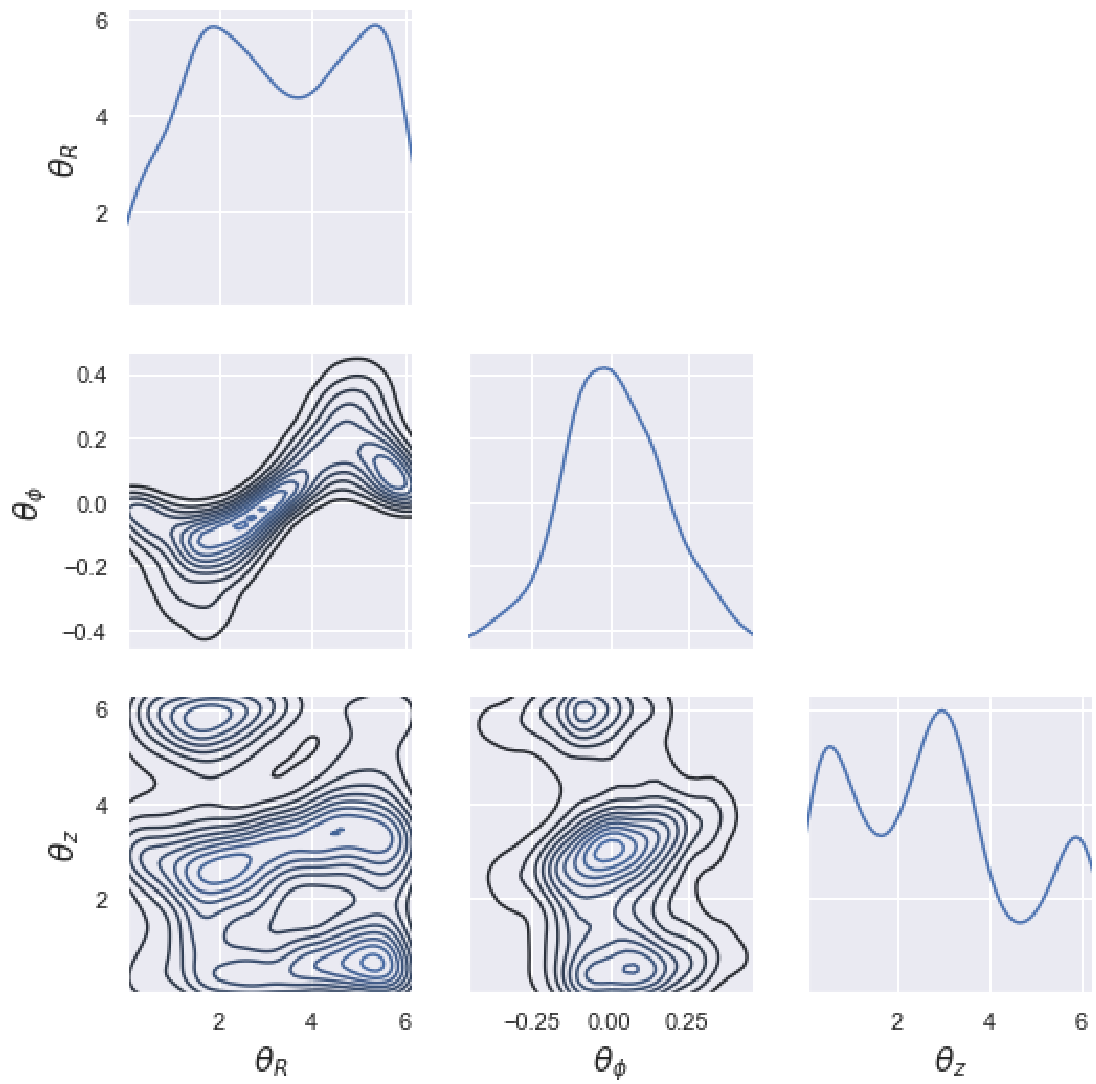
```
In [9]: pair_plot(pd.DataFrame(sqrt_actions.T,
                               columns=[r'$\sqrt{J_R}$',r'$L_z$',r'$\sqrt{J_z}$']))
```




```
In [10]: pair_plot(pd.DataFrame(frequencies.T,
                               columns=[r'$\Omega_R$',r'$\Omega_\phi$',r'$\Omega_z$']))
```




```
In [11]: pair_plot(pd.DataFrame(angles.T,
                         columns=[r'$\theta_R$',r'$\theta_\phi$',r'$\theta_z$']))
```



We have seen how to compute action, angle and frequency coordinates. There are several things we can do next.

1. Propagate uncertainties through the calculation.
2. Attempt to find clumps in the action space.
3. Look at how these distributions change with Galactic location.

See Trick et al. (2018).

Tutorial4

January 29, 2019

1 Tutorial 4: Dissecting local chemo-kinematic structure

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import pandas as pd
        import glob
        from astropy.table import Table, hstack, vstack
        from astropy.coordinates import SkyCoord, Distance, Galactocentric
        import astropy.units as u
        import warnings
        warnings.filterwarnings('ignore')

        %matplotlib inline
```

In this tutorial we will complement the Gaia data with other spectroscopic data. This demonstrates the crucial function of cross-matching large catalogues.

Gaia does not provide any metallicity or abundance information. In order to understand the chemo-kinematic structure of the Galaxy we must complement with results from other spectroscopic surveys. Here we will use GALAH as it has observations of many nearby bright stars. This uses the pre-downloaded data tables (available on USB sticks, by downloading from

http://cdn.gea.esac.esa.int/Gaia/gdr2/gaia_source_with_rv/csv/

and

https://www.ast.cam.ac.uk/~jls/data/galah_dr2.fits

or

https://datacentral.org.au/teamdata/GALAH/public/

). Note that as GALAH provides radial velocities we could cross-match to the entire Gaia catalogue and only use Gaia proper motions and parallaxes.

```
In [2]: def read_gaia_rvs(f):
            gaia_rvs_full = pd.read_csv(f)
            fltr = (gaia_rvs_full['parallax']>5.)&(gaia_rvs_full['parallax_over_error']>3.)
            fltr &= (gaia_rvs_full['radial_velocity']>-1000.)
            fltr &= (np.abs(gaia_rvs_full['b'])>8.)
            fltr &= (gaia_rvs_full['dec']<10.)
            gaia_rvs = gaia_rvs_full[fltr].reset_index(drop=True)
            return Table.from_pandas(gaia_rvs)
```

```
gaia_rvs = vstack([read_gaia_rvs(f) for f in glob.glob('GaiaSource_*.csv')])

galah = Table.read('galah_dr2.fits')
```

Here we find the entries in the Gaia RVS table that correspond to entries in the GALAH DR2 table. This will find the entry that is closest in $(\Delta\alpha\cos\delta, \Delta\delta)$.

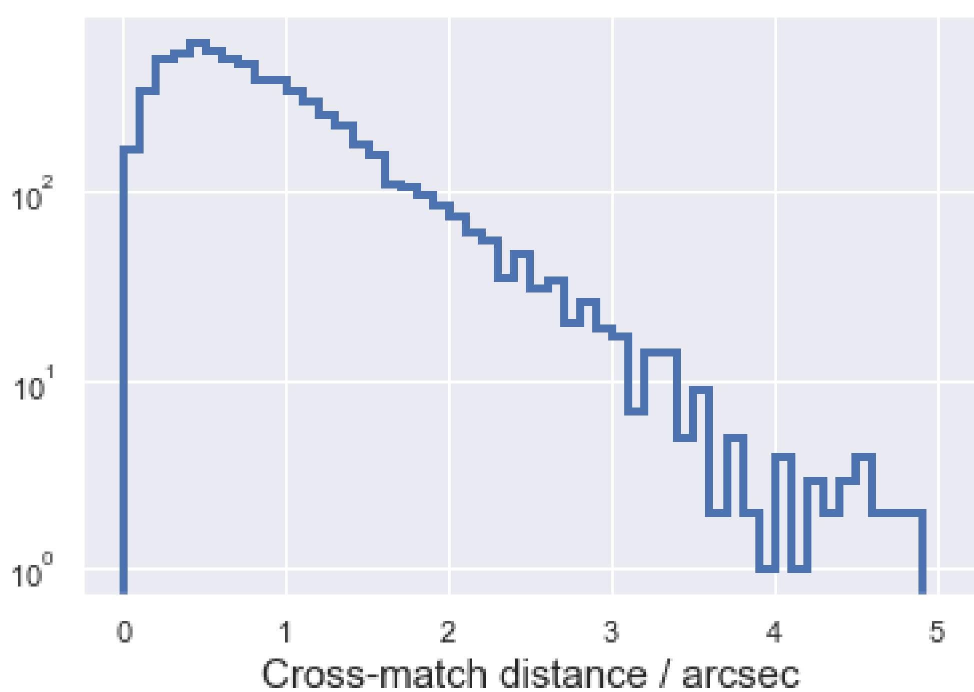
```
In [3]: gaia_rvs_skycoord = SkyCoord(ra=gaia_rvs['ra']*u.deg,
                                     dec=gaia_rvs['dec']*u.deg)
        galah_skycoord  = SkyCoord(ra=galah['raj2000']*u.degree,
                                   dec=galah['dej2000']*u.degree)
        idx, dist_onsky, d3d = galah_skycoord.match_to_catalog_sky(
                            gaia_rvs_skycoord)
```

We join the tables and then only keep entries where the cross-match distance was less than 3 arcsec.

```
In [4]: gaia_xm = hstack([galah,gaia_rvs[idx]])[dist_onsky<3.*u.arcsec]
        print('No. of XMatches=',len(gaia_xm))
        plt.hist(dist_onsky.arcsec,range=[0.,5.],histtype='step',lw=3,bins=50);
        plt.semilogy()
        plt.xlabel('Cross-match distance / arcsec')
```

```
No. of XMatches= 6698
```

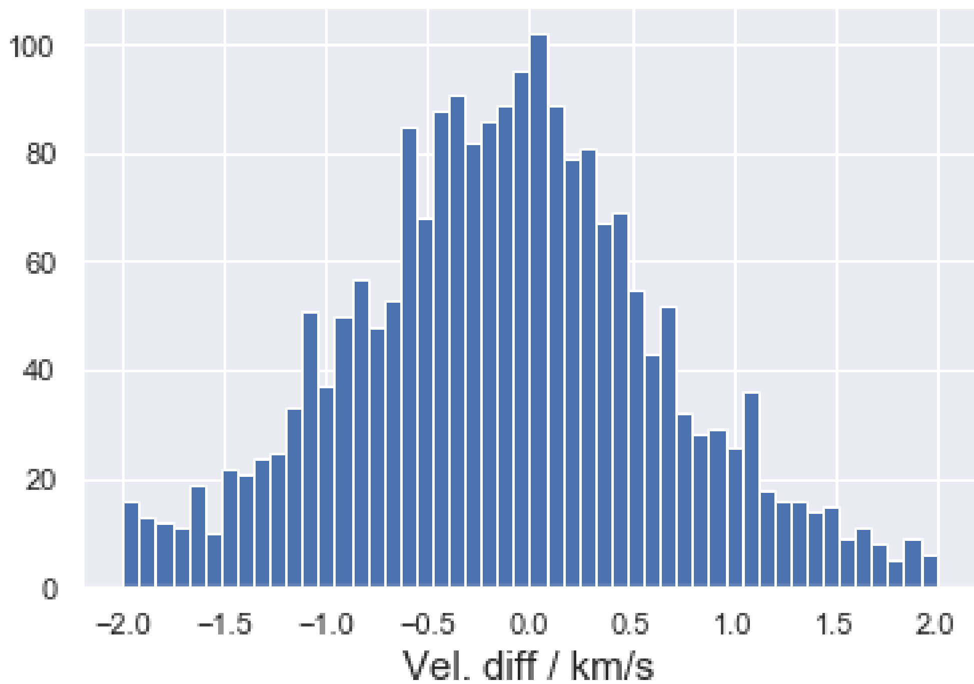
```
Out[4]: Text(0.5, 0, 'Cross-match distance / arcsec')
```



We can compare the radial velocities from Gaia and GALAH. Pretty good.

In [5]:
```
plt.hist(gaia_xm['rv_obst']-gaia_xm['radial_velocity'],range=[-2.,2.],bins=50);
plt.xlabel('Vel. diff / km/s')
```

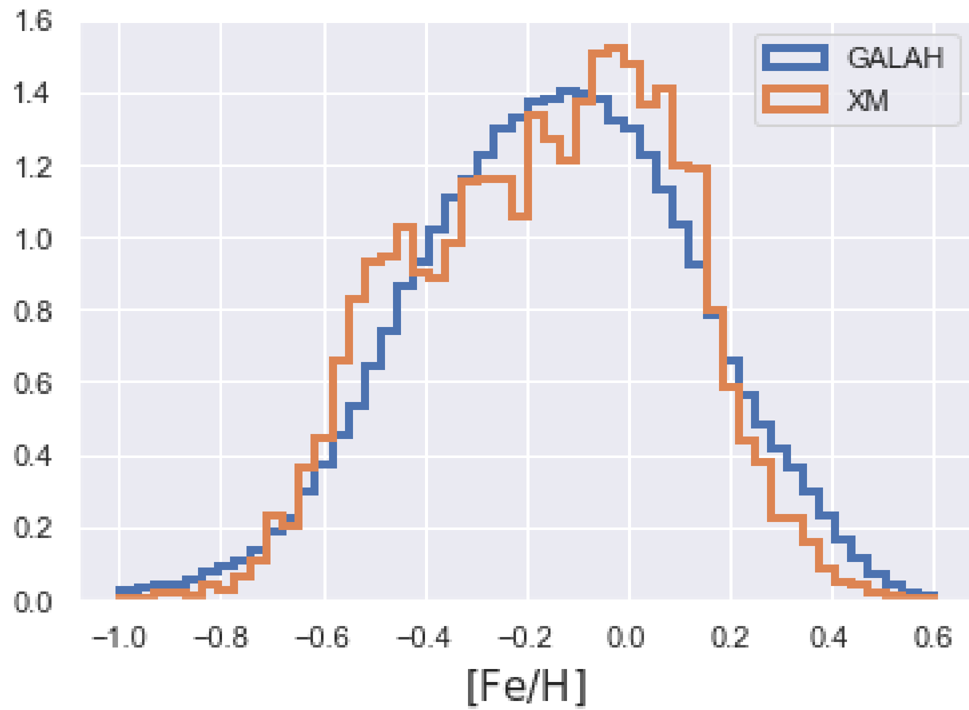
Out[5]:
```
Text(0.5, 0, 'Vel. diff / km/s')
```



The cross-match metallicity distribution is similar to all of GALAH.

In [6]:
```
plt.hist(galah['fe_h'],range=[-1.,0.6],normed=True,
         histtype='step',lw=3,bins=50,label='GALAH')
plt.hist(gaia_xm['fe_h'],range=[-1.,0.6],normed=True,
         histtype='step',lw=3,bins=50,label='XM')
plt.xlabel(r'$[\mathrm{Fe}/\mathrm{H}]$')
plt.legend()
```

Out[6]:
```
<matplotlib.legend.Legend at 0x1a1df6ff98>
```

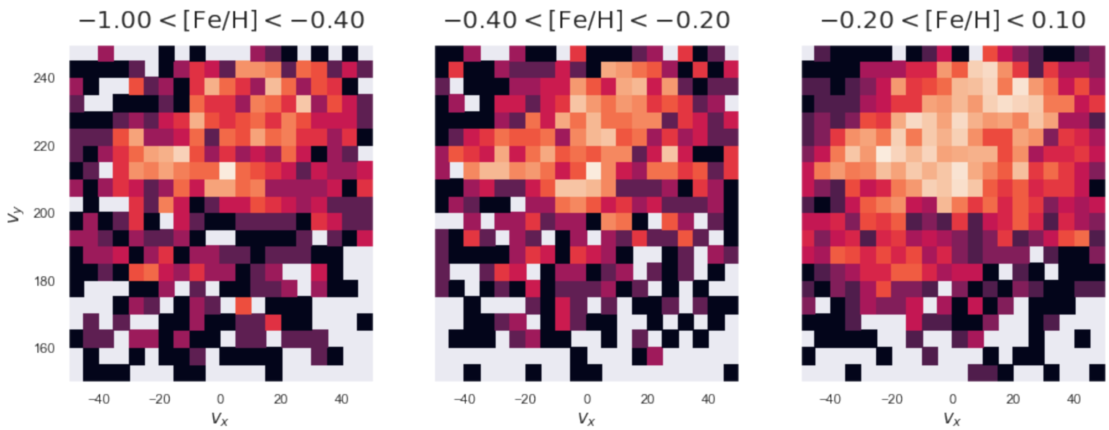


1.1 Velocity distributions split by metallicity

In [7]:
```
# allow_negative means the distance=nan if parallax<0
s = SkyCoord(ra=gaia_xm['ra']*u.deg,dec=gaia_xm['dec']*u.deg,
             distance=Distance(
                 parallax=gaia_xm['parallax'].data.data*u.mas,
                 allow_negative=True),
             pm_ra_cosdec=gaia_xm['pmra']*u.mas/u.yr,
             pm_dec=gaia_xm['pmdec']*u.mas/u.yr,
             radial_velocity=gaia_xm['radial_velocity']*u.km/u.s)
galcen = s.transform_to(Galactocentric)
```

In [8]:
```
f,a=plt.subplots(1,3,figsize=[15.,5.],sharey=True)
feh_range = [-1.,-0.4,-0.2,0.1]
from matplotlib.colors import LogNorm
for f in range(len(feh_range)-1):
    plt.sca(a[f])
    fltr = (gaia_xm['fe_h']>feh_range[f])&(gaia_xm['fe_h']<feh_range[f+1])
    plt.hist2d(galcen.v_x.value[fltr],galcen.v_y.value[fltr], bins=20,
              range=[[-50,50],[150.,250.]],norm=LogNorm(),normed=True)
    plt.annotate(r'$%0.2f<[\mathrm{Fe}/\mathrm{H}]<%0.2f$'\
                 %(feh_range[f],feh_range[f+1]),
                xy=(0.5,1.05),xycoords='axes fraction',
                 ha='center',fontsize=20)
```

```
        plt.xlabel(r'$v_x$')
    plt.sca(a[0])
    plt.ylabel(r'$v_y$');
```



The Hercules is definitely more visible in the metal-rich bin.

We have demonstrated how to cross-match catalogues. This is necessary if we want to use Gaia with any chemical abundance information. We have used this to inspect the local velocity structure separated by metallicity.

Things to do next:

1. Subdivide based on other abundances
2. Inspect action, angle, frequency diagrams split by abundance
3. Investigate vertical distributions (Gaia spiral)

See Bland-Hawthorn et al. (2018)

Tutorial5

January 29, 2019

1 Tutorial 5: Generalized Oort constants

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from astropy.table import Table, vstack
import glob
from astropy.coordinates import SkyCoord, Galactic
import astropy.units as u
from scipy.optimize import minimize
import emcee
import corner

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

In this tutorial, we look at fitting models probabilistically to data. We demonstrate how to use emcee (https://emcee.readthedocs.io) -- an MCMC sampling package. We will use a simple example from the lectures of measuring the Generalized Oort constants (Lecture 4, Slides 25 onwards). In theory, this problem could be analysed using simple linear least-squares. However, that method has a tendency to underestimate uncertainties and for more complex models MCMC is necessary.

The Oort Constants are related to the averaged proper motions μ and radial velocities $v_{||}$ across the sky like

$$\begin{aligned}
\bar{v}_{||}\varpi &= (K + C\cos 2\ell + A\sin 2\ell)\cos^2 b \\
&\quad - \varpi(u\cos\ell\cos b + v\sin\ell\cos b + w\sin b), \\
\bar{\mu}_\ell &= (B + A\cos 2\ell - C\sin 2\ell)\cos^2 b \\
&\quad - \varpi(u\sin\ell + v\cos\ell), \\
\bar{\mu}_b &= -(K + C\cos 2\ell + A\sin 2\ell)\sin b\cos b \\
&\quad + \varpi(u\cos\ell\sin b + v\sin\ell\sin b - w\cos b).
\end{aligned} \tag{1}$$

ϖ is the parallax, (u, v, w) is the solar peculiar motion, (K, A, B, C) the Oort constants (Here we have ignored the constant 4.74 that relates proper motions to velocities $v = 4.74\mu/\varpi$.

We will use the Gaia RVS sample again, restricting ourselves to nearby main sequence stars.

```
In [2]: def read_gaia_rvs(f):
            gaia_rvs_full = pd.read_csv(f)
            fltr = (gaia_rvs_full['parallax']>2.)&(gaia_rvs_full['parallax_over_error']>3.)
            fltr &= (gaia_rvs_full['bp_rp']>0.6)&(gaia_rvs_full['bp_rp']<0.7)
            fltr &= (gaia_rvs_full['phot_g_mean_mag']
                         -5.*np.log10(100./gaia_rvs_full['parallax'])>1.5)
            fltr &= (gaia_rvs_full['phot_g_mean_mag']
                         -5.*np.log10(100./gaia_rvs_full['parallax'])<4.5)
            gaia_rvs = gaia_rvs_full[fltr].reset_index(drop=True)
            return Table.from_pandas(gaia_rvs)
        gaia_rvs = vstack([read_gaia_rvs(f) for f in glob.glob('GaiaSource_*.csv')])
```

We transform to Galactic coordinates -- in particular we require the proper motions in Galactic coordinates. We also complement the catalogue with a 'line-of-sight proper motion' i.e.

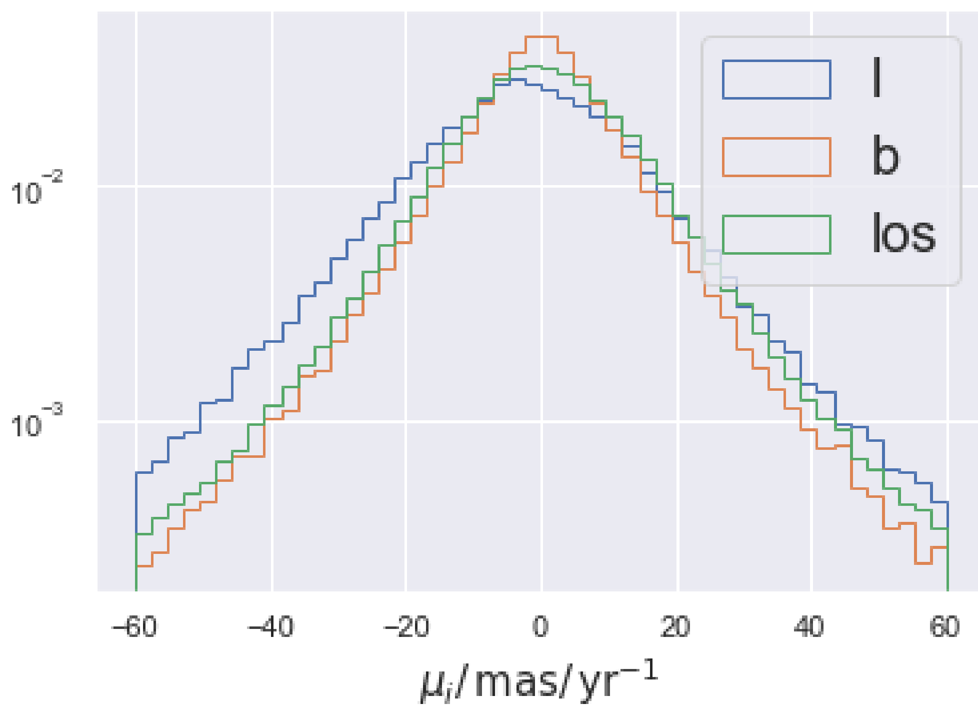
$$\mu_{\mathrm{los}} = \frac{v_{\mathrm{los}}\varpi}{4.74}$$

```
In [3]: PM_Const = 4.74057170372*u.km/u.s*u.yr

        s = SkyCoord(ra=gaia_rvs['ra']*u.deg,
                     dec=gaia_rvs['dec']*u.deg,
                     pm_ra_cosdec=gaia_rvs['pmra']*u.mas/u.yr,
                     pm_dec=gaia_rvs['pmdec']*u.mas/u.yr)
        gal = s.transform_to(Galactic)

        gaia_rvs['pml']=gal.pm_l_cosb
        gaia_rvs['pmb']=gal.pm_b
        gaia_rvs['pmlos'] = (gaia_rvs['radial_velocity']*u.km/u.s*\
                                  gaia_rvs['parallax']*u.mas)*1./PM_Const
```

First, we correct the proper motions for the solar peculiar motion (we use Schoenrich et al. 2010) using the parallaxes.

```
In [4]: solar_peculiar = np.array([11.1,12.24,7.25])*u.km/u.s

        def reflex_velocity(data):
            ''' Reflex velocity in l, b and los'''
            l = np.deg2rad(data['l'])
            b = np.deg2rad(data['b'])

            rv = solar_peculiar[0]*np.sin(l)\
                   -solar_peculiar[1]*np.cos(l),\
                 solar_peculiar[0]*np.cos(l)*np.sin(b)\
                   +solar_peculiar[1]*np.sin(l)*np.sin(b)\
                   -solar_peculiar[2]*np.cos(b),\
                -solar_peculiar[0]*np.cos(l)*np.cos(b)\
```

```
                -solar_peculiar[1]*np.sin(l)*np.cos(b)\
                -solar_peculiar[2]*np.sin(b)
        return rv

    rv = reflex_velocity(gaia_rvs)

    for ii,k in enumerate(['l','b','los']):
        gaia_rvs['pm%s_rf'%k]=gaia_rvs['pm%s'%k]\
                        -rv[ii]/PM_Const*gaia_rvs['parallax']*u.mas
        plt.hist(gaia_rvs['pm%s_rf'%k],histtype='step',bins=50,
                 range=[-60.,60.],normed=True,label=k);
    plt.semilogy()
    plt.legend(fontsize=20)
    plt.xlabel(r'$\mu_i/\,\mathrm{mas/\,yr}^{-1}$');
```



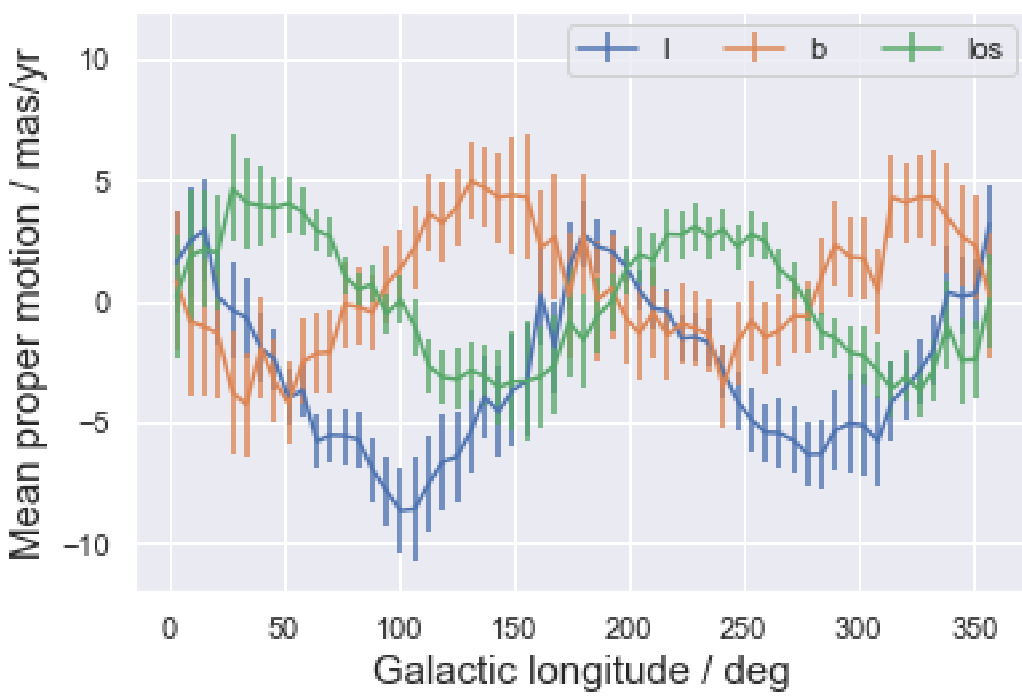
We include a further geometric factor in b that relates the velocities and distances to in-plane velocities and distances. Dividing by this factor gives us the mean quantities related to e.g. $K + C\cos 2\ell + A\sin 2\ell$.

```
In [5]: gaia_rvs['pml_rf_g']=gaia_rvs['pml_rf']/np.cos(np.deg2rad(gaia_rvs['b']))**2
        gaia_rvs['pmb_rf_g']=gaia_rvs['pmb_rf']/np.cos(np.deg2rad(gaia_rvs['b']))/\
                                    np.sin(np.deg2rad(gaia_rvs['b']))
        gaia_rvs['pmlos_rf_g']=gaia_rvs['pmlos_rf']/np.cos(np.deg2rad(gaia_rvs['b']))**2
```

We bin the data in ℓ and compute the mean and standard error in each bin. This neglects uncertainties in the proper motions, which we could subtract off using a median error.

```
In [6]: def sdm(x):
            '''
                Error in median -- using percentiles as more robust than standard deviation
                Factor is sqrt(pi/2)
            '''
            return 1.253*np.diff(np.percentile(x,[84.,16.]))/np.sqrt(len(x))

        ## Bin the data and compute means/errors
        groups = gaia_rvs[['l','pml_rf_g','pmb_rf_g','pmlos_rf_g']].group_by(
            np.digitize(gaia_rvs['l'],np.linspace(0.,360.,60))
            )
        Means = groups.groups.aggregate(np.median)
        Errors = groups.groups.aggregate(sdm)

        for i in ['l','b','los']:
            plt.errorbar(Means['l'],Means['pm%s_rf_g'%i],Errors['pm%s_rf_g'%i],label=i)
        plt.legend(ncol=3)
        plt.ylim(-12,12)
        plt.xlabel('Galactic longitude / deg')
        plt.ylabel('Mean proper motion / mas/yr')
```

```
Out[6]: Text(0, 0.5, 'Mean proper motion / mas/yr')
```



We use a simple model where the likelihood is given by

$$\mathcal{L} = \prod_i \mathcal{N}(\bar{\mu}_i | K_1 + K_2 \cos 2\ell_i + K_3 \sin 2\ell_i, \sigma_{\mu,i}) \tag{2}$$

Here the product is over ℓ bins and the $\mathcal{N}(x, \mu, \sigma)$ is a Gaussian in x centred on μ with width σ. In each ℓ bin we have a measurement of the mean proper motion $\bar{\mu}_i$ with uncertainty $\sigma_{\mu,i}$. We know that the mean should follow one of the 'Oort functions' which in general look like $K_1 + K_2 \cos 2\ell + K_3 \sin 2\ell$ where K_i are different depending on which component we consider. Taking the logarithm we find

$$\log \mathcal{L} = -\sum_i \frac{(\bar{\mu}_i - (K_1 + K_2 \cos 2\ell_i + K_3 \sin 2\ell_i))^2}{2\sigma_{\mu,i}^2} \tag{3}$$

where we have ignored the constant normalization term.

We define the log-likelihood function

```
In [7]: def lnL(p,l,data,invvar):
            ''' l in rad,
                data=mean proper motion (reflex and geometry corrected),
                invvar = inverse variance of measurements'''
            K1,K2,K3 = p[0],p[1],p[2]
            OortFn = K1 + K2*np.cos(2.*l) + K3*np.sin(2.*l)
            return -5.*np.sum(invvar*(data-OortFn)**2)
```

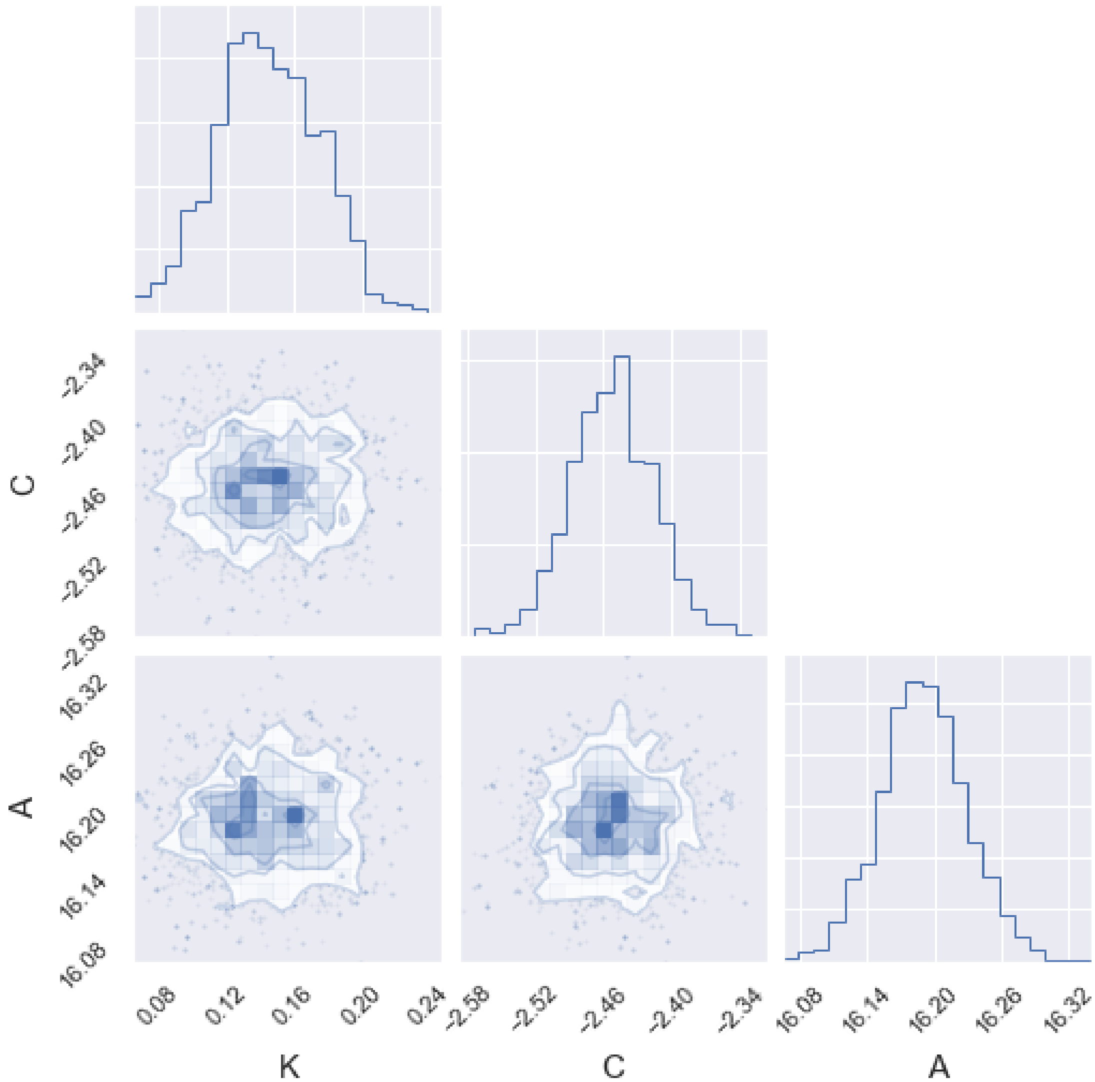
And set up a function to sample. We find an initial guess of the parameters using the minimize function in scipy.optimize.

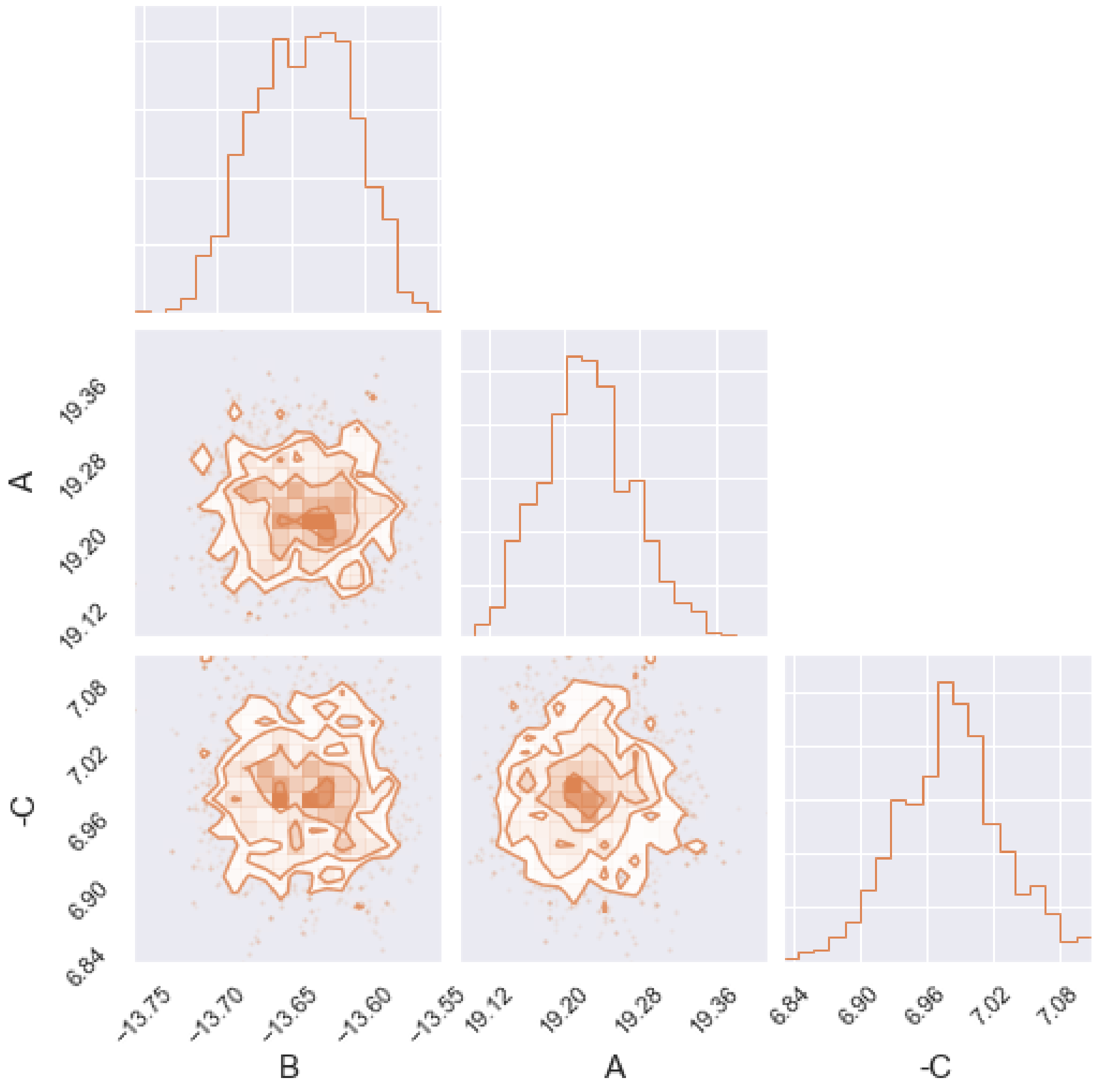
```
In [8]: def run_sampler(column, nwalkers=50, nsamples=100):
            ''' Run model on column data -- use nwalkers, run nsamples where half are burn-in'''
            ndim = 3

            # Set up data
            l = np.deg2rad(Means['l'])
            data = Means[column+'_rf_g']
            invvar = 1. / Errors[column+'_rf_g']**2

            ## Initialize walkers using the maximum likelihood solution
            optimum = minimize(lambda x: -lnL(x,l,data,invvar),
                               np.array([-12.,15.,3.])/PM_Const.value).x
            print('Minimum found at',optimum*PM_Const.value)
            p0 = np.array([np.random.normal(loc=optimum,
                                            scale=.01*np.ones(ndim),
                                            size=ndim)
                           for i in range(nwalkers)])

            sampler = emcee.EnsembleSampler(nwalkers, ndim, lnL,
                                            args=[l,data,invvar],threads=4)
            sampler.run_mcmc(p0, nsamples)

            return sampler.flatchain[-int(nsamples*nwalkers/2):,:]
```

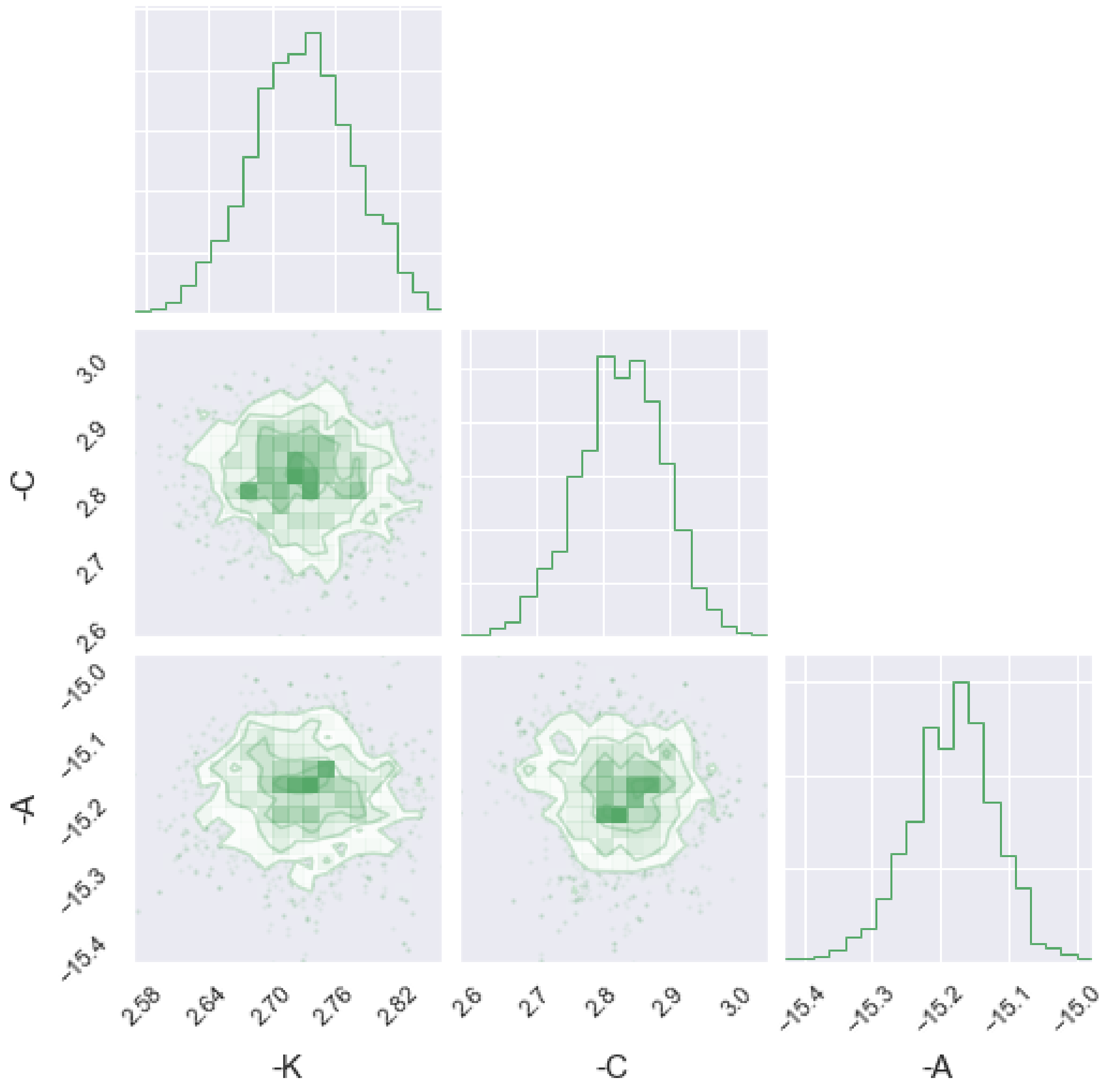
Run the samples and generate the standard corner plots. Here the constants are in units of proper motion (mas/yr).

```
In [9]: samplesLOS = run_sampler('pmlos',nsamples=150,nwalkers=50)

        samplesL = run_sampler('pml',nsamples=150,nwalkers=50)

        samplesB = run_sampler('pmb',nsamples=150,nwalkers=50)

        corner.corner(samplesLOS*PM_Const.value,color=sns.color_palette()[0],
                      labels=['K','C','A']);
        corner.corner(samplesL*PM_Const.value,color=sns.color_palette()[1],
                      labels=['B','A','-C']);
        corner.corner(samplesB*PM_Const.value,color=sns.color_palette()[2],
                      labels=['-K','-C','-A']);
```

```
Minimum found at [ 0.14542286 -2.45276189 16.1854335 ]
Minimum found at [-13.64419249  19.22829198   6.98550693]
Minimum found at [  2.73091762   2.81328797 -15.19090281]
```


C
-2.34
-2.40
-2.46
-2.52
-2.58
A
16.32
16.26
16.20
16.14
16.08
0.08
0.12
0.16
0.20
0.24
K
-2.58
-2.52
-2.46
-2.40
-2.34
C
16.08
16.14
16.20
16.26
16.32
A


A
-C
B
A
-C
19.36
19.28
19.20
19.12
7.08
7.02
6.96
6.90
6.84
−13.75
−13.70
−13.65
−13.60
−13.55
19.12
19.20
19.28
19.36
6.84
6.90
6.96
7.02
7.08



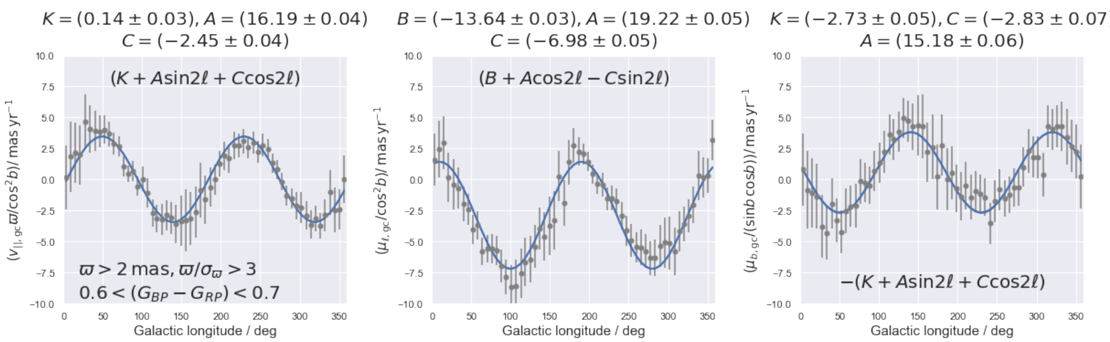

```
In [10]: def plot_models(samples,N=10):
             for i in np.random.randint(0,len(samples),N):
                 plt.plot(Means['l'],samples[i,0]+
                          samples[i,1]*np.cos(2.*np.deg2rad(Means['l']))+
                          samples[i,2]*np.sin(2.*np.deg2rad(Means['l'])),
                          color=sns.color_palette()[0],
                          alpha=0.5)
```

We compare the models to the data. We have plotted many samples from the model, although you cannot tell. The formal uncertainty in the parameters is very small.

```
In [11]: f,a=plt.subplots(1,3,figsize=[20.,5.])
         plt.subplots_adjust(wspace=0.3)
```

```
plt.sca(a[0])
plt.annotate(r'$\varpi>2\,\mathrm{mas}, \varpi/\sigma_\varpi>3$'+'\n'
             +r'$0.6<(G_{BP}-G_{RP})<0.7$',
             xy=(0.05,0.),xycoords='axes fraction', fontsize=20,
             ha='left', va='bottom')

for i,K in enumerate(['pmlos','pml','pmb']):
    a[i].errorbar(Means['l'],Means['%s_rf_g'%K],Errors['%s_rf_g'%K],
                  fmt='o',ms=5,color='grey');

plt.sca(a[0])
plot_models(samplesLOS)
plt.ylabel(r'$(v_{||,\mathrm{gc}}\varpi/\cos^2b)/\,\mathrm{mas\,yr^{-1}}$')
plt.annotate(r'$(K+A\sin2\ell+C\cos2\ell)$',xy=(0.5,0.95),xycoords='axes fraction',
             fontsize=20,ha='center',va='top')
plt.annotate(r'$K=(%0.2f\pm%0.2f),$'%(np.median(samplesLOS[:,0]*PM_Const.value),
                                     np.std(samplesLOS[:,0]*PM_Const.value))
            +r'$A=(%0.2f\pm%0.2f)$'%(np.median(samplesLOS[:,2]*PM_Const.value),
                                    np.std(samplesLOS[:,2]*PM_Const.value))
             +'\n'+r'$C=(%0.2f\pm%0.2f)$'%(np.median(samplesLOS[:,1]*PM_Const.value),
                                          np.std(samplesLOS[:,1]*PM_Const.value)),
             xy=(0.5,1.02),xycoords='axes fraction',
             fontsize=20,ha='center',va='bottom')

plt.sca(a[1])
plot_models(samplesL)
plt.ylabel(r'$(\mu_{\ell,\mathrm{gc}}/\cos^2b)/\,\mathrm{mas\,yr^{-1}}$')
plt.annotate(r'$(B+A\cos2\ell-C\sin2\ell)$',xy=(0.5,0.95),xycoords='axes fraction',
             fontsize=20,ha='center',va='top')
plt.annotate(r'$B=(%0.2f\pm%0.2f),$'%(np.median(samplesL[:,0]*PM_Const.value),
                                     np.std(samplesL[:,0]*PM_Const.value))
            +r'$A=(%0.2f\pm%0.2f)$'%(np.median(samplesL[:,1]*PM_Const.value),
                                    np.std(samplesL[:,1]*PM_Const.value))
             +'\n'+r'$C=(%0.2f\pm%0.2f)$'%(np.median(-samplesL[:,2]*PM_Const.value),
                                          np.std(samplesL[:,2]*PM_Const.value)),
             xy=(0.5,1.02),xycoords='axes fraction',
             fontsize=20,ha='center',va='bottom')

plt.sca(a[2])
plot_models(samplesB)
plt.ylabel(r'$(\mu_{b,\mathrm{gc}}/(\sin b\,\cos b))/\,\mathrm{mas\,yr^{-1}}$')
plt.annotate(r'$-(K+A\sin2\ell+C\cos2\ell)$',xy=(0.5,0.05),
             xycoords='axes fraction',fontsize=20,ha='center',va='bottom')
plt.annotate(r'$K=(%0.2f\pm%0.2f),$'%(np.median(-samplesB[:,0]*PM_Const.value),
                                     np.std(-samplesB[:,0]*PM_Const.value))
            +r'$C=(%0.2f\pm%0.2f)$'%(np.median(-samplesB[:,1]*PM_Const.value),
                                    np.std(-samplesB[:,1]*PM_Const.value))
```

```
                    +'\n'+r'$A=(%0.2f\pm%0.2f)$'%(np.median(-samplesB[:,2]*PM_Const.value),
                                                   np.std(-samplesB[:,2]*PM_Const.value)),
                    xy=(0.5,1.02),xycoords='axes fraction',
                    fontsize=20,ha='center',va='bottom')
    for ii in range(3):
        plt.sca(a[ii])
        plt.xlabel(r'Galactic longitude / deg')
        plt.xlim(0.,360.)
        plt.ylim(-10.,10.)
```



Things to do next (e.g. Bovy 2017 with TGAS data):

1. Consider the (correlated) uncertainties in the observables.
2. Repeat calculation for different stellar populations.