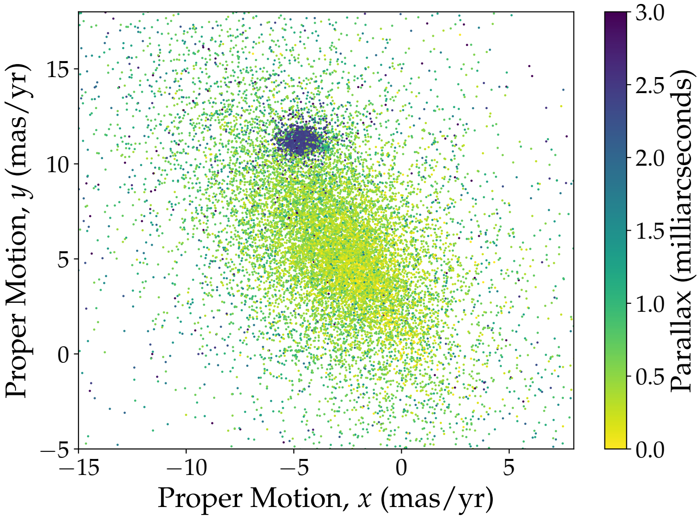
Statistics, Data Analysis, and Machine Learning for Physicists
Timothy Brandt
Spring 2020

Homework 3

This homework will provide a brief introduction to some of the other topics we have covered in class, including mixture models, bootstrap resampling, MCMC and PCA.

Part 1: Outlier Treatment and Bootstrapping

This part of the homework will examine a real case of digging a signal out of a pile of nuisance data, which you might call outliers. You have a Jupyter notebook to get you started on the data, to read it in and make an illustrative plot. I reproduce that plot below for reference:



a. Start by just considering the parallaxes. Plot a histogram of the parallaxes of all stars (you may cut off your histogram at a parallax of 4 or 5; there is no need to include the handful of outliers representing a few nearby stars). The spike at around 2.5 milliarcseconds is from stars in the cluster. Parallax $\sim$ 1/distance, so small parallaxes correspond to distant stars.

b. Assume that all members of this star cluster have the exact same parallax in the absence of measurement error (a pretty good approximation). Construct a reasonable distribution for the outliers (nonmembers of the cluster) by looking at your histogram from part (a) and roughly matching the population *not* in the spike at $\sim$2.5 milliarcseconds. Then use either a mixture model with this outlier distribution or iteratively reweighted least squares (or a combination of the two methods) to infer the cluster's parallax. Compute the standard error on this parallax ($\Delta \ln \mathcal{L} = -0.5$). Finally, use bootstrap resampling on the entire data set to compute another estimate of the uncertainty on the parallax. For your prior on a star being a cluster member, you may use either $\frac{1}{2}$ or the approximate fraction of stars that seem to be members by their parallax (which you can estimate). The latter choice will be a bit more robust.

c. Now use a mixture model in three dimensions. For this step, you should also construct background distributions in proper motion (you may include the observed covariance in the background proper motions in x and y if you wish, but you don't have to). Don't worry about getting these distributions exactly right–Gaussians with means and variances that roughly match the data are good enough. For your model of the cluster stars, use the following (I use $\mu_{x,i}$, $\mu_{y,i}$, and ϖ_i to represent proper motion in x and y, and parallax, respectively for a given star, and $\mu_{x,\mathrm{cl}}$, $\mu_{y,\mathrm{cl}}$, and ϖ_{cl} to represent the cluster proper motions and parallax):

$$p(\mu_{x,i}, \mu_{y,i}, \varpi_i|\mathrm{member}) = A \exp\left[-\frac{1}{2}\left(\frac{(\mu_{x,i}-\mu_{x,\mathrm{cl}})^2}{\sigma_{x,i}^2+\sigma_{x,\mathrm{cl}}^2} + \frac{(\mu_{y,i}-\mu_{y,\mathrm{cl}})^2}{\sigma_{y,i}^2+\sigma_{y,\mathrm{cl}}^2} + \frac{(\varpi_i-\varpi_{\mathrm{cl}})^2}{\sigma_{\varpi,i}^2}\right)\right] \tag{1}$$

with

$$A = \frac{1}{(2\pi)^{3/2}\sqrt{(\sigma_{x,i}^2+\sigma_{x,\mathrm{cl}}^2)(\sigma_{y,i}^2+\sigma_{y,\mathrm{cl}}^2)(\sigma_{\varpi,i}^2)}}. \tag{2}$$

In these two equations, you will fit for five parameters: $\mu_{x,\mathrm{cl}}$, $\mu_{y,\mathrm{cl}}$, ϖ_{cl} (the cluster's proper motion and parallax), plus $\sigma_{x,\mathrm{cl}}^2$ and $\sigma_{y,\mathrm{cl}}^2$ (measures of the internal velocity dispersion, and indirectly, of the cluster's mass). The $\sigma_{x,i}^2$ and $\sigma_{y,i}^2$, and $\sigma_{\varpi,i}^2$ are given in the data table supplied with this homework.

Use your favorite nonlinear optimization routine to fit for these five parameters or, if you prefer, you can use a linear optimization routine in $\mu_{x,\mathrm{cl}}$, $\mu_{y,\mathrm{cl}}$, and ϖ_{cl} combined with nonlinear optimization in $\sigma_{x,\mathrm{cl}}^2$ and $\sigma_{y,\mathrm{cl}}^2$ (depending on how you approach the mixture model). Use bootstrap resampling to estimate the uncertainties on all five parameters.

d. Use your 3D mixture model, together with prior probabilities of either $\frac{1}{2}$ or your estimated membership fraction for cluster membership for all stars, to compute posterior membership probabilities for all stars. Remake the plot above, but color-code by posterior probability of membership rather than by parallax.

Part 2: MCMC

We will now revisit Homework 2 and see how to apply MCMC to derive parameters and confidence intervals. This sort of analysis is very common and was actually used in the paper.

First, implement your own MCMC based on the Metropolis-Hastings algorithm. Start with Part 2a, the full optimization with the linear model of the background (and a diagonal covariance matrix), but allowing the exponent to be free. You may marginalize out the linear part analytically, including c_0, but if you do, pay close attention to the solutions to Part 2c. You'll have to multiply the likelihood by the square root of the determinant of the *model*'s covariance matrix. Assuming you do marginalize over $\{a_0, \ldots, a_4, c_0\}$, you will have three nonlinear parameters: c_1, c_2, and the exponent c_3.

Try either using a single step in all directions at once (i.e. a proposed step in each of the variables) or a proposed step in one (random) variable at a time. Note that if you cycle through the variables in a pre-defined way you are breaking the symmetry of the proposal distribution, an assumption that MCMC needs to work. You should choose the typical step size (width of a Gaussian proposed step) to be comparable to (maybe a bit less than) the root variance in each direction that you found when you originally did Homework 2. Then create a Markov chain by running your analysis with 100,000 points. Measure your acceptance rate, and tune your step sizes if your acceptance rate is very far from 25%. You have to re-run the chain when you do this; you cannot tune your steps mid-chain or you will again break the assumption that steps are symmetric (unless you are careful and account for this asymmetry in your acceptance probability).

Once you have a chain, decide how much to discard as the "burn-in" by plotting the distribution of each parameter as a function of iteration number, and measuring when the typical values of the parameters are similar to their best-fit values. After you have discarded the burn-in, estimate the effective number of points for the rest of the chain by measuring the correlation length. You will measure a different correlation length for each parameter; be sure to choose the longest one. Finally, use use the python package `corner` to make a corner plot of the errors and covariances on your three nonlinear parameters governing the absorption profile.

Finally, implement the full nonlinear model of Part 1b combined with the additional three parameters for an absorption profile, fixing the exponent at a new value of 2.8. In other words, use the model

$$T[\nu] = b_0 \nu^{-2.5+b_1+b_2 \log \nu} e^{-b_3 \nu^{-2}} + b_4 \nu^{-2} + c_0 \exp\left[-\left| \frac{\nu - c_1}{c_2} \right|^{2.8} \right] \tag{3}$$

along with the full covariance matrix for the data. You should now have eight (nonlinear) parameters in your model, and you can simply call `emcee`, an MCMC package in python, to compute an MCMC chain on these parameters and generate corner plots. Routines like this can be very easy to use and make pretty plots, but it is worth thinking a bit before reaching for them. Note that even in this case you can marginalize three of these parameters out analytically, which would help convergence a lot. Once again, estimate your burn-in and the correlation length to get the number of independent points. With this value of 2.8 for the exponent, the distribution is strongly multimodal. How successful was emcee at finding the multiple peaks of the likelihood function, and what did that fact do to your correlation length?

Part 3: PCA

Now you will do an exercise in principal component analysis. I have provided you with a sequence of images of a star. You may load these images using the following code:

```
from astropy.io import fits images = fits.open(’images.fits’)[0].data
```

This will load the images into a three-dimensional array. The first dimension is the number of images, the second dimension is the number of pixels in the vertical direction, and the third dimension is the number of pixels in the horizontal direction.

First, compute the mean image and subtract it from each of the images in the array. Then perform PCA on this sequence using `numpy.linalg.svd` (suggestion: set `full_matrices=False`). In order to do this decomposition, you will have to reshape the array with `numpy.reshape` to make it two-dimensional. After doing the SVD, you will want to reshape your principal component matrix **U** back to the shape of the original set of images. Plot the mean image and the first two principal components. Also, plot the residual variance as a function of the number of principal components, and suggest an appropriate point at which to truncate the decomposition.

Next, add these three lines of code after loading the images:

```
badpix = (numpy.random.uniform(0, 1, images.shape) > 0.9999)*(images != 0)
badpixval = numpy.std(images)*100
images += badpix*numpy.random.uniform(0, badpixval, images.shape)
```

to make 0.01% of the pixels noisy (but only those pixels that did not have a value of zero assigned), with the noise being much larger than the typical pixel. Most of the variance in the data is still from the signal. Compare the principal components now with the ones you computed previously, paying special attention to the less significant PCs, say numbers 20–40. You should notice that the less-significant principal components look more structured when computed with noisier data. Explain why.