$\mathcal{L}IRA$ Mini-Tutorial

A. Connors, with help from D. van Dyk and N. M. Stein

A. Getting your own copy:

A.1. Contacts:

1. Nathan Stein : nmstein (fas dot harvard dot edu) or nathanmstein (gmail dot com)
2. Alanna Connors : aconnors (eurekabayes dot com)
3. David van Dyk : david.a.vandyk (gmail dot com)

A.2. Building and installing.

1. Install R from cran: `cran.r-project.org/doc/manuals/R-admin.html`
2. If you wish, install R's fitsIO package, from Harris of STSci. It is listed under 'packages', so you can use the handy R-GUI package installer (menu bar at top).
3. Get a LIRA tar-ball either from the contacts above, or from the github repository: `github.com/vkashyap/LIRA` .
4. If needed un-gzip and un-tar the file (tar xzvf lira-etc.gz)
5. Make sure your machine's gcc matches that of the R-build: e.g. for Mac darwin: sudo select gcc 4
6. Change your working directory to `lira/`.
7. For unix/linux–like operating systems, do the usual R-package procedure:
    (a) R CMD BUILD or sudo R CMD BUILD
    (b) R CMD INSTALL or sudo R CMD INSTALL
8. LIRA has not yet been built on a Windows machine. Theoretically one could follow the procedures documented in `cran.r-project.org/doc/manuals/R-admin.html`.
9. If you wish, you can try out some of the R-command files in `tests/`.

10. Check out the documentation! Hopefully during the INSTALL, the `docs/` directory moved from `inst/` (the *install* directory) to the main directory.

That's it! Have fun!

B. Quick Summary of Steps to Run LIRA

For those who already understand MMIs, MCMC, Low-count Poisson, Bayesian priors and hyperpriors, etc., etc., we offer this quick outline of how to run LIRA.

B.1. INPUTS/OUTPUTS

1. INPUTS:
   (a) Required inputs:
      i. A $2^n \times 2^n$ matrix of data, in the form of cts/bin;
      ii. Number of iterations, thinning, names of output files. Typically one starts with iter=500 ot iter=1000; thin=1. This allows one to calculate burn-in and auto-correlation length. Next one does several runs, with thin$\geq$auto-correlation length; and enough iterations so that one gets about $10^4 - 10^5$ good samples. Defaults are available for all of these if none were given.

   (b) Optional:
      i. A $2^n \times 2^n$ background or best-fit model or null model map as the *baseline.*
      ii. An exposure map of matching dimensions. If there is an exposure map, then the units of the exposure map times the background or model map should be cts/bin, matching the data map.
      iii. A map of the point-spread function, or instrument smearing. The bin-size should match that of the data, but the size can be arbitrary (but no larger than the data map).

2. OUTPUTS:
   (a) A log file, *.Rout, that echoes the inputs and keeps track of the usage time. Any error messages should end up in here.
   (b) An ascii parameter file, typically *.par, that has a header designated by "#"; and a list of parameters and hyper-parameters for each MCMC iteration. If "thin" is set > 1, then only every "thin" iterations will be written out.
   (c) An ascii output of the MCMC images, typically *.out. If the input data map has dimension $2^n \times 2^n$, then each row in the file will contain 2^n ascii entries. If "thin" is set > 1, then only every "thin" iterations will be written out. Hence the entries written to the "param" file and the sample images written to this "*.out" file will match.

B.2. SHORT RUN FOR CHECKING:

1. Do a few short runs (iter at least 500), with a minmum of three different widely separated starting values.
2. Plot traces and scatter plots. Looks OK? Burn-in? Auto-correlation Length? Example scripts to do these (in RPlotsUtils):
   (a) `Simple_01_Rplot_traces_128x128FaintE_conv_Gauss2D_unconv_noPSF_test01.sh`
   (b) (in python)

   `quicko_autocor_write.py`

   (c) `Simple_01_Rplot_traces_128x128FaintE_conv_Gauss2D_unconv_noPSF_autocor.sh`
3. Look at movies (it's fun!). Do the images make sense? Example scripts:
   (a) `Simple_04_RUtil_AsciiRoutToFitsImageMovies.sh`
   (b) `Simple_04_RUtil_AccumulatedCleanFitsImageMoviesToMoments.sh`
4. If wished, look at the mean and higher moments (after burn-in). Do these images make sense?
   (a) `Simple_04_RUtil_AccumulatedCleanFitsImageMoviesToMoments.sh`
5. Given the limits you want, estimate resources for the longer runs: How many separate runs with separate starting maps (at least three)? How many iterations for each run (so in the end you have a total, over all the separate runs, of $\sim 10^4 - 10^5$ good samples)? How long will each take? (Careful! $\sim 10^4 - 10^5 \times\ 128 \times 128$ or 256×256 images starts being quite large.)

   If one wants only a rough visual representation, and not quantitative limits, one can stop here.

B.3. QUANTITATIVE ANALYSIS 1: LONGER ANALYSIS RUN -

1. If it all works out, do several longer runs (iter 1000 or more good samples).
2. Again, plot traces and scatter plots. All OK?
3. Look at the movies from separate runs. All OK?
4. Combine, after chopping burn-in, to calculate mean and higher moments

5. Scrutinize combinations of the fit-parmaters. Which is the appropriate summary statistic for your problem? Since we haven't done the 'baseline/null' runs for comparison, at this stage you might use just the total counts in the Multiscale Component.

6. If you wish, sort the samples according to your summary statistic. Select the ones that are around n% and 100-n% of the tail. Averaging these two sets of images will give you your upper and lower confidence bounds. One rather clumsy example:

   (a) (in python)

   ```
   run_second_sort_imagemovie.py
   ```

B.4. QUANTITATIVE ANALYSIS 2: COMPARING WITH BASELINE/NULL MODEL FOR GOODNESS OF FIT:

To get statistical significance, or a 'distance' of the data from one's baseline/null, one needs to run the exact same analysis on data-sets simulated from the baseline map. This is a short description, but running all the nulls can take a long time.

1. Create several ($\geq 5 - 10$) Poisson realizations of your baseline/null model.

   (a) (in python)

   ```
   howto_just_get_simpler_poiss.py
   ```

2. Analyze in exactly the same way as in 2 and 3, above.

3. COMPARE the scatter plots and especially the summary statistic distributions. How much overlap? In RPlotsUtils, see:

   ```
   Simple_04_RPlot_FaintEParam_scatterplots_hist.sh
   ```

4. COMPARE the Baseline/Null and Interesting images at the same values of the summary statistic that gave the +/- confidence bounds. As noted before, a sample script that sorts only on total MS Counts is in PyUtils:

   ```
   run_second_sort_imagemovie.py
   ```

5. Make sense of the result!

B.5. QUANTITATIVE ANALYSIS 3: MORE ADVANCED SUMMARY STATISTICS:

Now suppose the scatter plots reveal that the Baseline/Null differs from the interesting data in several of the fit parameters. How to make use of *all* the info in the fit-parameters? We will define a simple 'distance' between the Baseline and Interesting data.

1. First, one wants the distributions about each parameter to be (relatively) symmetric. That implies non-linear transformations for the total MS counts (usually square-root or log) and background pre-factor (usually log).

2. Second, find the *medians* $\mu_{i,B}$ *and* $\mu_{i,D}$ of each parameter in the total Baseline distribution; and the Interesting Data distribution.

3. Third, the parameters all have different scales. How to 're-norm' them so that they all have the same scale? We chose to calculate the *variance about the median* for each parameter, in the Baseline and Interesting parameter samples. Let $\sigma_{i,B}$ and $\sigma_{i,D}$ represent the square root of the variance of parameter i, for the Baseline and Interesting data, respectively. We now define the re-normed parameters as:

$$\rho_{s,i,B} = (par_{s,i,B} - \mu_{i,B})/\sigma_{i,B} \;\; and \;\; \rho_{s,i,D} = (par_{s,i,B} - \mu_{i,D})/\sigma_{i,B}. \tag{B1}$$

   Here s stands for the iteration number of a good McMC sample. Notice that we have subtracted $\mu_{i,B}$ and divided by $\sigma_{i,B}$ for both the Baseline and Interesting data parameters. Hence the re-normed Baseline/Null distribution parameters will all have a median of zero.

4. Now we can better combine information from differing parameters. Specifically, the $\{\rho_{s,i,D}\}$, for each s, are now vectors showing the direction and distance of s from the Null distribution medians.

5. Finally, we can use these to calculate our choice of Summary Statistic. Suppose we choose several parameters (e.g. the background scale pre-factor and the total MS counts; or all the smoothing hyper-parameters; or all the parameters) for the Summary Statistic. Then the *re-normed parameter medians* $\nu_{i,D} = (\mu_{i,D} - \mu_{i,B})/\sigma_{i,B}$, where i runs through all the parameters one has chosen, define a vector. We will calculate our Summary Statistic $\mathcal{S}_s$ as the *distance along this vector*, for each good sample s of parameters. That is, if $\hat{\nu}_{i,D}$ is defined as *the unit vector in the direction of* $\nu_{i,D}$, then, for each good McMC sample of parameters s, the Summary Statistic is defined as the dot-product:

$$\vec{\mathcal{S}}_{s,D} = \hat{\nu}_D \dot{\vec{\rho}}_{s,D}. \tag{B2}$$

6. scripts for calculating these can be found under RPlotsUtils in:

   ```
   Simple_04_RUtil_RenormParams_GetSummaryStats.sh
   ```

7. Histograms of the Baseline/Null vs Interesting Summary Statistics will now show the distributions of their relative posterior probailities. Sample scripts can be found in:

   ```
   Simple_04_RPlot_RenormParam_SumStat_scatterplots_hist.sh
   Simple_04_RPlot_RenormParam_SumStats_Hyp_BkgMSC_Tot_scatterplots_hist.sh
   ```

B.6. VISUALIZING QUANTITATIVE ANALYSIS 4: THE IMAGES

Each sample image s now has a corresponding Summary Statistic $\mathcal{S}_s$. Hence one can sort the images by $\mathcal{S}$. One can then calculate the mean of the images within slices of $\mathcal{S}$. To dislay what the mean images look like that correspond to (say) the $\pm n\%$ tails of the summary statistic, one can use the RPlotsUtlis script:

```
Simple_04_RUtil_AccumulatedCleanFitsImageMoviesToSliceMeans.sh
```

Careful – for large data-sets and large samples, this can become memory-limited and hence very slow.

That's it! Have fun!

C. Special Notes for Users:

1. LIRA **REQUIRES** unprocessed counts, since it uses the Poisson formulation. The data CANNOT be background-subtracted (data-cuts are fine, though); or averaged or extrapolated (as in some CGRO/EGRET all-sky maps), or 'corrected' by an exposure factor (as is often done with TeV telescope data). All such processing belongs in the 'Instrument' model in the forward fitting: effective area; PSF; exposure map; and background. The process will simply NOT WORK (i.e. give you very weird results) if you try running on data that have been 'corrected' in some fashion.

2. As it is now written, LIRA REQUIRES the data to be in square maps of dimension 2^n. Any accompanying model, exposure, and background maps must match the data. In theory, Nowak and Kolaczyk have found a way, in one dimension, to lift the 2^n requirement - by implicitly embedding it in a larger 2^n vector. We have not done this work, in 2D, for LIRA.

3. Note that LIRA has smoothing intrinsically built into the multi-scale component. The multi-scale portion is NOT engineered to detect point-sources (although it can – but the fluxes will tend to be a little low).

4. No longer always convex, as we have introduced more competing model components (see next). SO should try multiple starting maps to make sure they all converge to the same final chain. (see next section.)

5. Although this is only important when running with a non-zero background model, it is often a good idea to include MCMC runs with: 1) starting values too high (i.e. puts all counts in the MS component, to start); and 2) starting values very low (i.e. puts all counts in the background component, to start). If there is going to be any problem with multiple modes, this can illustrate it rather quickly.