Manual of the fakePipe

---- 2015-11-10 Song Huang ----

Basic Information:

- The code for the fakePipe can be found here: `https://github.com/clackner2007/fake-sources`
- This project is to add fake stellar and galaxy sources to the HSC data processing. We use galsim to generate fake galaxy sources. Stellar (PSF) sources are simply taken as the measured PSF. In the current incarnation, the pipeline accepts fake sources after the calibration but before measurement when processing single CCDs. This way, the fake sources don't affect things like PSF determination. Once the sources are added, they are deblended and measured just like the rest of the sources, and included in the output catalogs.

Setup:

- This code depends on the HSC pipeline (version 3.3.3 or later), and needs to have `Galsim` built against the pipeline python. For some of the additional scripts (makeRaDec, matchFakes) the `astropy` package is also necessary.
- For example, on Master@IPMU, one just needs to setup the following:

```
export PYTHONPATH=/home/clackner/.local/lib/python2.7/site-packages:${PYTHONPATH}
setup -v hscPipe 3.9.0
setup -v -j astrometry_net_data ps1_pv1.2c
setup -v -r path/to/fake-sources/
```

- The first `export` is only to get astropy on PYTHONPATH, so adjust according to your local installation.

- For more information about `Galsim`, please check [here https://github.com/GalSim-developers/GalSim]. The current version used by fakePipe is v1.3.0. To install `Galsim`, one also needs to install the `TMV` library (details can be found here: https://code.google.com/p/tmv-cpp/wiki/Installation).

- On Master@IPMU, one can setup `Galsim` by:

```
setup -v -j -r /home/clackner/src/tmv0.72/
setup -v -j -r /home/song/code/GalSim-1.3.0/
```

- Change the location to `Galsim` and `TMV` based on your installation.

Running:

We use several examples to show you how to run fakePipe.

1. MultiBand Test of Galaxy Photometry

(a). Prepare the input catalog

- The fakePipe accepts fake galaxies described by either single Sersic or double Sersic models. For the example here, we are only considering the single Sersic models. The single Sersic fitting results of galaxies down to F814W=25.0 mag on HST/ACS images are used as the parent catalog (private communication with Claire Lackner).
- The following columns are necessary for single Sersic models:
  - Magnitude: either magnitude in single band: `mag` ; or magnitude in different bands: `mag_g` , `mag_r` , et al.
  - `reff` (half light radius; in unit of arcsec); `sersic_n` (Sersic index; `b_a` (axis ratio), `theta` (position angle)
  - Possible shears: `g1` and `g2`

- To make sure that `Galsim` can generate images of fake galaxy without any problem, one need to make sure that `0.25 < b/a < 1.0` ; And, `sersic_n < 0.5` or `sersic_n > 6.0` often indicate unrealistic models; High Sersic index can also leads to longer time for fake generation and large snapshot image (We truncate the profile at 10 times of the Re).
- For this example, we are using `cosmos_23.5_multiband.fits` which includes 14113 fake galaxies down to 23.5 magnitude; the same magnitude value is used for all five bands. The catalog can be found at `/lustre/Subaru/SSP/rerun/song/fake` on Master@IPMU.

(b). Random (RA, DEC) assignment

- Now, we need to decide the (RA, DEC) of the fake galaxies on the real images. In principle, you can choose the coordinates in any way you want, and two columns: `RA` and `Dec` to the input catalog. But, in this example, we show you how to randomly assign coordinates to input catalog. There are two ways to do this:

[1]. makeSourceList.py (recommended):

- For most of the tests, we will be working at the **Tract** level. It is easier to use the fakePipe tool: `makeSourceList.py` :
- Basic usage:
  `bash makeSourceList.py DATA_DIR --rerun=RERUN_DIR \ --id tract=TRACT_ID filter='HSC-I' patch='4,4' \ -c inputCat=`
  - `DATA_DIR` : Locations of the reruns;
  - `--rerun` : Name of the data rerun
  - `--id` : HSC data ID for coadd image; The only useful one here is the Tract number; The Patch number and filter are dummy ones.
  - `outDir` : Location for the output catalogs.
  - `inputCat` : Name of the input fake galaxies catalog; If no catalog is provided, a catalog that only contains the RA, DEC will be saved.
  - `rhoFakes` : Number of fakes to be put on one Patch; Default number is 500;
  - `rejMask` : An optional mask in "well-known binary" format to exclude regions that have no useful data on it. This function depends on the `Shapely` : Python library. Please contact Song Huang in case you want to use this for certain Tract.
  - `innerTract` : Only add fakes to the InnerTract region; Default is True.
- Example command:

```
makeSourceList.py /lustre/Subaru/SSP \
     --rerun=yasuda/SSP3.8.5_20150725 \
     --id tract=8524 filter='HSC-I' patch='4,4' \
     -c inputCat='cosmos_23.5_multiband.fits' \
     rejMask='ssp385_wide_8524_HSC-I_nodata_big.wkb' \
     rhoFakes=300
```

- Since 5-bands magnitudes are included in the input catalog, 5 separated catalogs will be generated: `src_TRACT_radec_FILTER.fits`
- In the command line output, you can find the number of fake galaxies in the output catalogs, like:

```
## Filter through : ssp385_wide_8524_HSC-I_nodata_big.wkb
## 24295 out of 24300 objects left
```

[2]. makeRaDecCat.py:

- In case you want to add fakes to only 1 CCD, or in a square region larger than 1 Tract, you can use the `makeRaDecCat` function. Right now, makeRaDecCat.py accepts either a dataId ({visit, ccd} or {tract, patch, filter}) or a range of {RA, Dec} as input.
- The code will return a list of {RA,Dec} pairs; It also accepts an input fits catalog, and will add two columns to the catalog (RA, Dec). Make sure the number of galaxies in the input catalog is equal or smaller than the number of random RA,Dec pairs (This can be improved later). The output catalog will have a `_radec` suffix.
- And, an optional `rad` parameter is available as the minimum allowed separation (in unit of arcsec) between any of these random RA, Dec pairs.
- Example usages:

```
from fakes import makeRaDecCat
rangeRaDec = [123.4, 123.8, 12.0, 13.0]
inputCat = 'fakeExp.fits'
randomRaDec = makeRaDecCat(50, rangeRaDec=rangeRaDec, rad=10.0, inputCat=inputCat)
```

or

```
dataId = {tract:0, patch:'4,5', filter:'HSC-I'}
rootDir = '/lustre/Subaru/SSP/rerun/song/cosmos-i2'
inputCat = 'fakeExp.fits'
randomRaDec = makeRaDecCat(50, dataId=dataId, rad=10.0, inputCat=inputCat,                    rootDir=rootDir)
```

(c). Add fake galaxies to the images

[1] Decide the Visits you want to work on

- Right now, the `runAddFakes.py` still only works at single Visit level. This gives you the freedom to choose which Visits you want to work on. But, it also means that you need to manually figure out the input Visits to coadds of certain band, and manually add fakes to these images.
- Use the `tractFindVisits.py` tool in the fakePipe. Basic usage is:
  `bash tractFindVisits.py RERUN TRACT_ID --filter FILTER --patch PATCH_ID \ --dataDir DATA_DIR`
- Default `filter` is `HSC-I` ; and Default `dataDir` is: `/lustre/Subaru/SSP`
- Examples:

```
tractFindVisits.py 'yasuda/SSP3.8.5_20150725' 8254 \
    --patch='4,4' --filter='HSC-I'
```

- The above command will search the input Visits to one patch, and output is like:

```
# Input visits for Tract=8524 Filter=HSC-I Patch=4,4

 # Input CCDs includes 12 Visits

7288^7310^7350^7352^7364^7378^7394^7402^14144^14164^14166^14178
```

- For `HSC-R` : `11426^11462^11496^11498^11522^11524`
- For `HSC-G` : `9840^9880^9882^9892^9908^9916^9918^11634^11672`
- For `HSC-Z` : `9696^9718^9742^9744^9754^9766^9782^9790^13284^15078^15142^15154`
- For `HSC-Y` : `6466^6488^6490^6536^6538^6558^13144^13146^13194^13212^16098^16106`

- It also works for the entire Tract:

```
tractFindVisits.py 'yasuda/SSP3.8.5_20150725' 9347
```

- To visualize the locations of these Visits to the Tract, you can use the `showTractVisit.py` script in the `test` folder. The basic usage is:

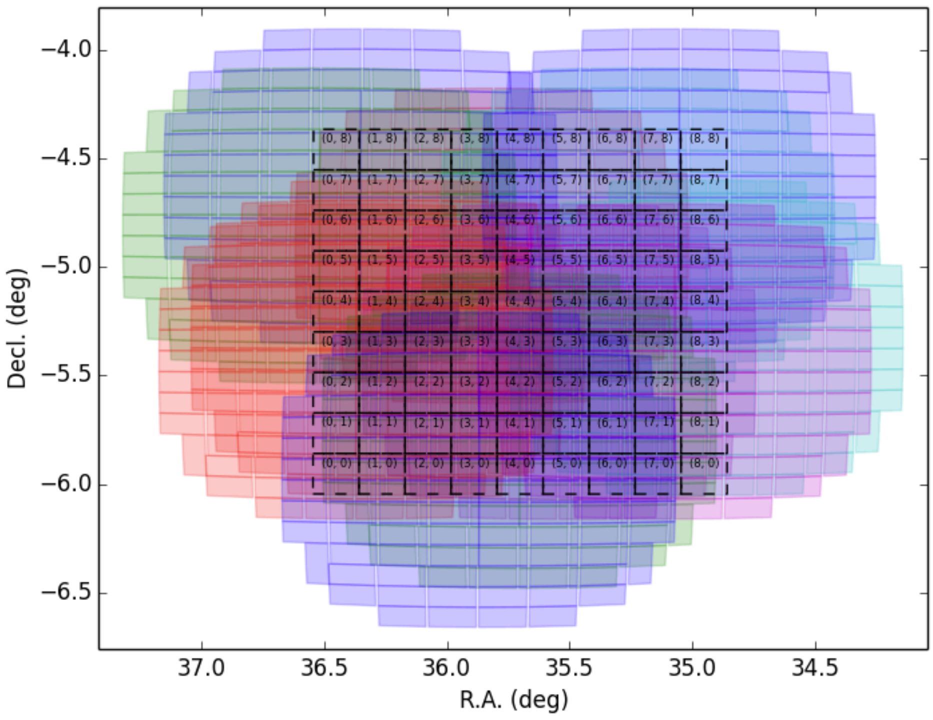
```
python showTractVisit.py RERUN TRACT VISITS -p
```

- `RERUN` , `TRACT` , `VISITS` : Define the datasets, the target `Tract` , and the selected `Visits`
- `-p` : Overplot the boundaries of the patches within this `Tract`

- For example:

```
python showTractVisit.py \
    /lustre/Subaru/SSP/rerun/yasuda/SSP3.8.5_20150725 8524
    7288^7310^7350^7352^7364^7378^7394^7402^14144^14164^14166^14178 -p
```

- Will draw a figure like this:



- It clearly shows that for a typical `WIDE` field `Tract` , the number of `Visits` we need to consider during the fake tests can be much larger than 3.

[2] Prepare the config file for runAddFakes.py

- For each band, one needs to prepare one configuration file for `runAddFakes.py` ; Example config file, `addfake_i.config` , contains:

```
from fakes import positionGalSimFakes
root.fakes.retarget(positionGalSimFakes.PositionGalSimFakesTask)
root.fakes.galList = INPUT_CATALOG
root.fakes.galType = 'sersic'
root.fakes.maxMargin = 150
root.fakes.addShear = False
```

- In case your input catalog includes the g1, g2 shears, set `addShear = True`

[3] Add fake galaxies to images by running runAddFakes.py

- Now, you can run `runAddFakes.py` to add fake galaxies to the images. You need to manually run this for each band. The basic usage is:

```
runAddFakes.py DATA_DIR --rerun OLD_RERUN:NEW_RERUN --id visit=VISIT_LIST \
    --clobber-config -C CONFIG_FILE --do-exec --queue small --job JOB_NAME \
    --nodes 4 --procs 12
```

- `DATA_DIR` : location of the data, `/lustre/Subaru/SSP/`
- `OLD_RERUN` : The rerun contains the input images
- `NEW_RERUN` : Output rerun for images with fake galaxies added
- `VISIT_LIST` : List of input visits to go through, `7288^7310^7350^7352^7364^7378^7394^7402^14144^14164^14166^14178`
- `CONFIG_FILE` : Configuration file from the above section, `addfake_i.config`
- `JOB_NAME` : Name of the qsub job

- For example:

```
runAddFakes.py /lustre/Subaru/SSP/ \
    --rerun yasuda/SSP3.8.5_20150725:song/fake/multi_test1 \
    --id visit=7288^7310^7350^7352^7364^7378^7394^7402^14144^14164^14166^14178 \
    --clobber-config -C addfake_i.config \
    --queue small --job addfake_i_1 --nodes 8 --procs 12
```

- Will creat a new rerun folder called `song/fake/multi_test1` ; You can check the job using `qstat` ; And, you should start to see many log files from qsub. When the job finishes, you should check the job summary file to make sure there is no error. Also, you can check two output files:
    - `runAddFake.skipped` : For different reasons, the `addFakes.py` sometimes have to skip certain fake galaxies. In this file, you can see the id of the fake and the reason why it was skipped: `199053 , bboxEdge` ; `bboxEdge` means that the (RA, Dec) is in the gap between CCDs, which is Ok; In case you see the reason involves `GalSim` , it indicates that `GalSim` fails to generate the image of fake galaxy for some reason. You should check the model parameter of those fake models to make sure the parameters are reasonable.
    - `runAddFake.missingCCD` : The `addFakes.py` will go through all the CCDs in a Visit to check whether we should put any fake on it. However, in some rare occasions, FITS files for certain CCDs could be problematic (Often indicated by the abnormal file size of the CORR-XXXXX.fits file). There is nothing we can do here, so we simply skip them. You can check this file for information of these "missing" CCDs.

[4] Visually exam the fake objects on single CCD image (optional)

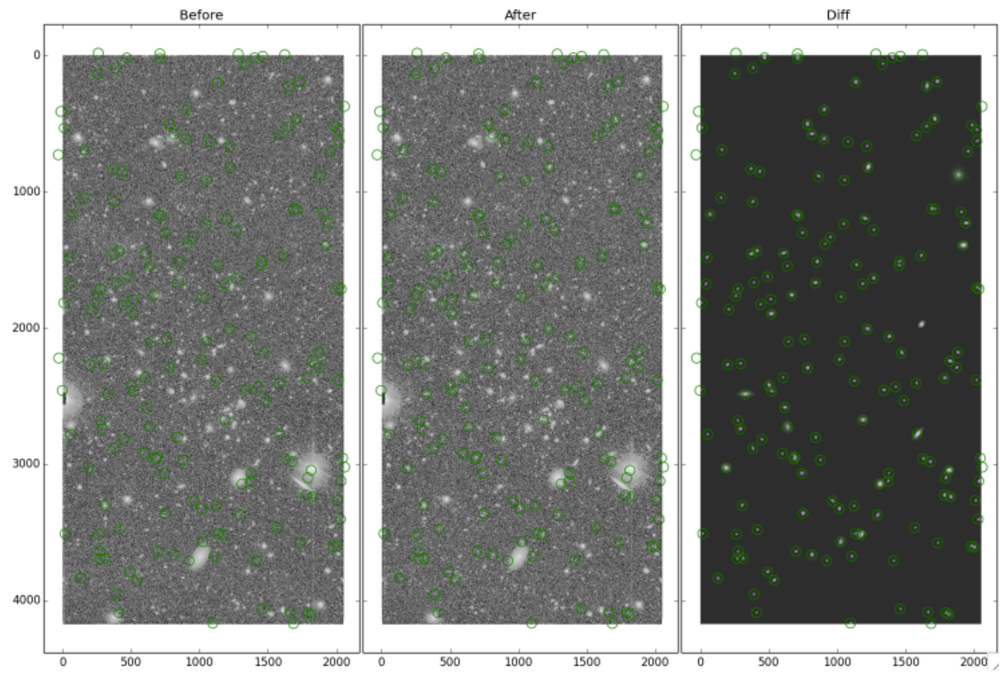
- At this point, you may want to check the fake galaxies on the single CCD image. In the `test` folder of fakePipe, a short script, `compFakeGalaxy.py` , can be used for this purpose. It generates a three-panel figure. Left is the original image, middle is the one with fake objects on it, and the right one is the difference between them, so you should be able to see the fake galaxies on this one clearly. The usage is:

```
python compFakeGalaxy.py OLD_RERUN NEW_RERUN VISIT CCD
```

- For example:

```
python compFakeGalaxy.py yasuda/SSP3.8.5_20150725 song/fake/multi_test1 7388 40
```

- Here, we assume the rootDir for data is: `/lustre/Subaru/SSP` ; If your data are stored somewhere else, you can easily change the script to point it to somewhere else. This also applies to the `compFakeCoadd.py` script in the next section.



- `OLD_RERUN` : Rerun for the original data
- `NEW_RERUN` : Rerun for data that have fake galaxies on them
- `VISIT` , `CCD` : DataID for the CCD to check
- Note that not all CCDs of a Visit should have fakes on them. For a Visit, it is possible that only a few CCDs that really contribute to the Tract. So, it is normal that you find no fake for certain CCD. This tool is meant to be a simple visualization.

(d). Generating coadd images in each band

[1] Prepare the configuration file:

- Now we need to run `stack.py` to generate the coadd images in each band. We also need a configure file to overwrite some default parameters of `stack.py` :
    - Since we only care about the fake objects, we can use `detectOnlyFakes.py` to only measure the objects close to fake footprint:

```
import fakes.detectOnlyFakes
root.processCoadd.detectCoaddSources.detection.retarget(fakes.detectOnlyFakes.OnlyFakesDetectionTask)
```

    - Since the pipeline is using the `multiband.py` now. We don't really care about the photometric measurements in each band. So, we can turn-off the the most time-consuming cModel photometry by replacing it with Gaussian photometry:

```
root.processCoadd.calibrate.measurement.algorithms.names=['flux.psf', 'flags.pixel', 'flux.gaussian', 'flux.aperture'
, 'flux.naive', 'centroid.naive', 'flux.sinc', 'shape.sdss', 'shape.hsm.moments', 'multishapelet.psf', 'correctfluxes
', 'classification.extendedness', 'skycoord', 'shape.hsm.psfMoments']

root.processCoadd.measurement.algorithms.names=['flux.psf', 'flags.pixel', 'shape.hsm.moments', 'flux.aperture', 'flu
x.naive', 'focalplane', 'flux.gaussian', 'centroid.naive', 'flux.sinc', 'shape.sdss', 'jacobian', 'shape.hsm.regauss'
, 'flux.kron', 'correctfluxes', 'classification.extendedness', 'skycoord', 'shape.hsm.psfMoments']

root.processCoadd.calibrate.measurement.slots.modelFlux='flux.gaussian'
root.processCoadd.measurement.slots.modelFlux='flux.gaussian'
```

[2] Run `stack.py` to generate coadd images:

- Save the above content in a config file, e.g. `stack_i.config` , we can use `stack.py` to generate the coadd images; Example of command for `stack.py` is like:

```
stack.py /lustre/Subaru/SSP/ --rerun=song/fake/multi_test1 \
    --id tract=8524 filter=HSC-I patch='4,4' \
    --selectId visit=7394^7402^14144^14164^14166^14178 \
    --queue small --nodes 6 --procs 12 --job stack_i_1 \
    --clobber-config -C stack_i.config \
    --config doOverwriteOutput=True doOverwriteCoadd=True \
             makeCoaddTempExp.doOverwrite=True
```

- In the above example, we will only generate coadd for one `Patch` ; Without the `patch=XXX` , the process will generate all Patches for this Tract.
- `--selectId` indicates the specific Visits you want to use for this coadd.

- The last three options are needed if you added the fake sources to already processed data, otherwise the pipeline will skip making the coadd and use the coadd without any fake sources.

- After the process, you should check the job output file to ensure that nothing werid happens. And, you should be able to find the coadd image in your rerun: e.g. for above run, it should be: `..../song/fake/multi_test1/deepCoadd/HSC-I/8524/4,4.fits` .

[3] Check the coadd images (optional):

- You can use the `showInDs9.py` script in the `test` folder of the fakePipe to visualize the FAKE mask plane on the coadd image. The basic usage is:

```
python showInDs9 NEW_RERUN TRACT PATCH --filter FILTER
```

- `NEW_RERUN` : Rerun for data that have fake galaxies on them
- `TRACT` , `PATCH` , `FILTER` : DataID for the Patch to check; Note that a filter must be supplied by the user, or the script will try to interpret the `TRACT` , `PATCH` as `VISIT` , `CCD` for single visit exposure.
- This will open up a DS9 window, and display the coadd image with information from the mask plane. The **Blue** pixels belong to detected objects; **Green** regions are bad data (interpolated pixels); **Red** squares are the regions used for the fake objects (the `FAKE` mask plane); any object (both real and fake one) whose fake pipeline overlaps the `FAKE` mask plane will be included in the measurement process (highlighted by tiny **red** circles; Noticed that some of the red circles can be well outside the `FAKE` plane, they are included because their footprints still touch one of the `FAKE` mask region (due to large size or problematic deblending process).

- You can also use the simple script, `compFakeCoadd.py` , in the `test` folder to compare the coadd image before and after the fakes are added. The usage is very simiar to the `compFakeGalaxy.py` for single exposure:

```
python compFakeCoadd.py OLD_RERUN NEW_RERUN TRACT PATCH FILTER
```

- For example:

```
python compFakeCoadd.py yasuda/SSP3.8.5_20150725 song/fake/multi_test1 8524 '4,4' 'HSC-I'
```

- Will generate a 3-column figure that shows the original coadd, the coadd with fakes on it, and their differences.
- Notice that if you did NOT select all the Visits that go into this Patch, most of the differences are actually caused by the difference of the image depth.

- If your tests only reguire the coadd results in single band. You have finished the parts that require running the pipeline, and can jump to step (f) to match the results.

(e). Multiband processing

- Now the HSC pipeline has switched to the multi-band processing as the default method. For the multi-band process, please see the following links for more details:
  - http://hsca.ipmu.jp/hscsphinx/pipeline/multiband.html
  - http://hscsurvey.pbworks.com/w/page/87953929/Coadd%20Multi-Band%20Processing

[1] Prepare the configuration file:

- As usual, a configuration file is necessary for this process. The file should contain the following information:

```
root.measureCoaddSources.propagateFlags.flags={}
root.clobberMergedDetections = True
root.clobberMeasurements = True
root.clobberMergedMeasurements = True
root.clobberForcedPhotometry = True
```

- And, we can also only do measurements for objects that are close to our fake footprints. This involves the `mergeOnlyFakes.py` function, which is still UNDER CONSTRCUTION [!!]. This should be done by including following lines into the configuration file:

```
import fakes.mergeOnlyFakes
#root.measureCoaddSources.retarget(fakes.mergeOnlyFakes.OnlyFakesMergeTask)
# NOT AVAILABLE YET
```

[2] Running `multiBand.py` for multiband processing:

- Once the configuration file, e.g. `multiband.config` , is prepared, we can start run `multiBand.py` . The basic usage is something like:

```
multiBand.py /lustre/Subaru/SSP --rerun=song/fake/multi_test1  \
    --id tract=8524 filter=HSC-I^HSC-R patch='4,4' \
    --queue small --nodes 1 --procs 2 --job multiband_test1 \
    --clobber-config -C multiband.config \
    --mpiexec='-bind-to socket' --time 2000
```

- The basic idea is very similar to the `stack.py`, so we just show this example without giving further explanation.
- Also, without giving specific `patch=xxx`, `multiBand.py` will work on the entire `Tract`
- After the job is finished, you should check the job summary to make sure nothing weird happened. And, you should be able to find multiband outputs in the `..../multi_test1/deepCoadd-results` folder. The above example applies to two filters, so you should find three folders here: `HSC-R`, `HSC-I`, and `merged`. In the folder for each filter, you should find files like: `det-HSC-I-8524-4,4.fits`, `meas-HSC-I-8524-4,4.fits`, `srcMatch-HSC-I-8524-4,4.fits`, and `forced_src-HSC-I-8524-4,4.fits`. And, in the `merged` folder, you should find `mergeDet-8524-4,4.fits`, `ref-8524-4,4.fits` under `8524/4,4/`.
- By now, you have finished all the parts that involves using HSC pipeline, and can proceed to check the results. One can easily use the pipeline tool to load in the catalogs, and do their own match with the input catalog. The `fakePipe` provides a naive code to match the results with the input too.

(f). Match input fake galaxies with output catalogs

- By using the `runMatchFakes.py` command, one can provide basic matching with the input catalog using the (RA, DEC) information at both single exposure and coadd image level. The basic usage is:

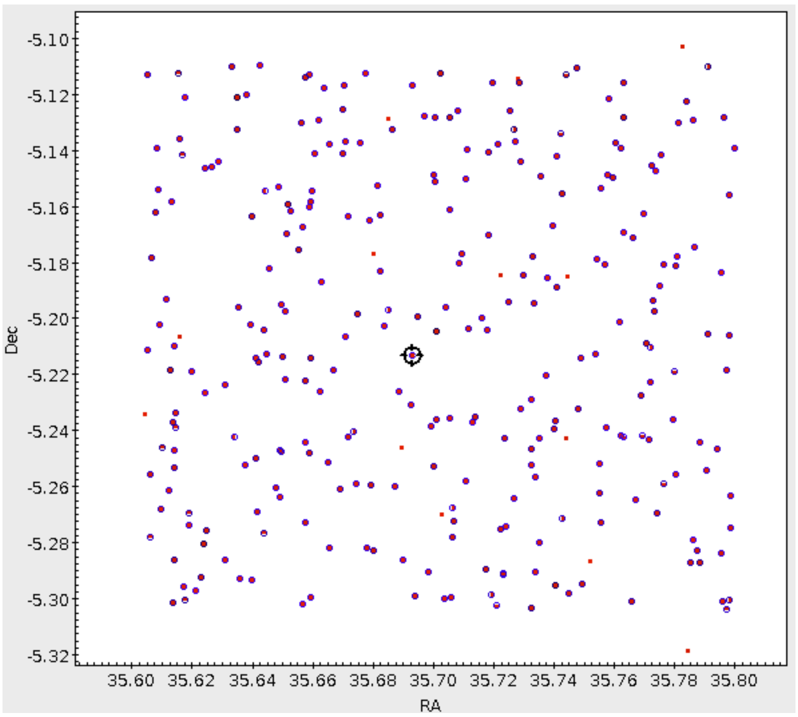
```
runMatchFakes.py RERUN_DIR VISIT_OR_TRACT --ccd CCD_OR_PATCH \
    -f FILTER -o OUTPUT_FITS -c INPUT_FAKES -w -m -t TOLERANCE
```

- `RERUN_DIR`: Location of the rerun that contains the fakePipe test
- `VISIT_OR_TRACT`, `CCD_OR_PATCH`, `FILTER`: Controls the data to be matched. When no `FILTER` is provided, the code treats the dataId as `VISIT`, `CCD`; And, when `FILTER` is provided, it considers the dataId to be `TRACT`, `PATCH`; Multiple `CCDs` and `PATCHes` can be provided too. But, each time, only one filter can be provided (For now, you have to run match on multiband results separately, and deal the results manually).
- `OUTPUT_FITS`: Name of the output FITS catalog
- `INPUT_FAKES`: The input catalog of fake galaxies. If not present, the code will try to use the information stored in the metadata (e.g. `FAKEXXX X Y`), instead of the (RA, Dec) information in the catalog. This only works for the single exposure images, and will fail for coadd products.
- `-w`: Whether over-write the output, default is `False`
- `-m`: Whether match with the multiband output (`deepCoadd_meas`), instead of the single band catalog from `stack.py` (`deepCoadd_src`). For multiband results, there are other useful catalogs (e.g. `deepCoadd_ref`, `deepCoadd_forced_src`). To use them, one has to change the `matchFakes.py` [TODO]
- `-t TOLERANCE`: The matching radius in PIXELS. Default value is 1.0.

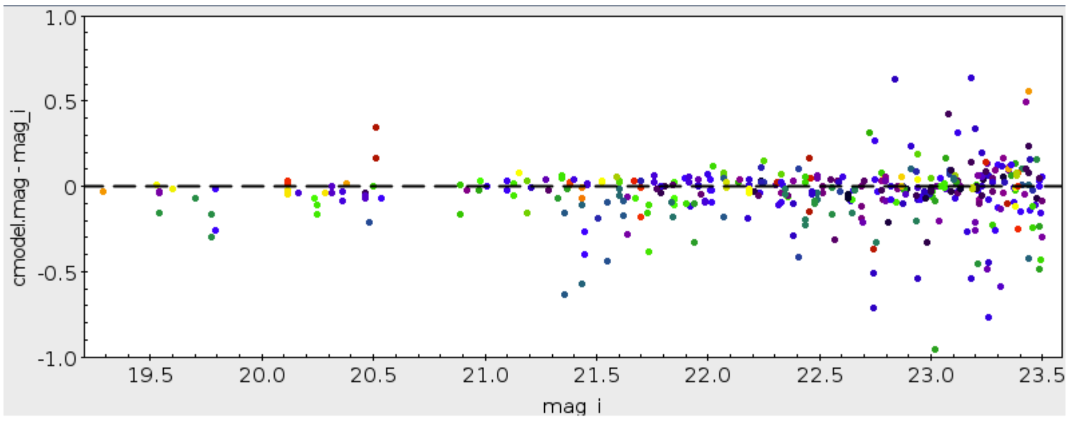
- Example usage of `runMatchFakes.py` for the above `multiBand.py` example is:

```
runMatchFakes.py /lustre/Subaru/SSP/rerun/song/fake/multi_test1 8524 \
    --ccd '4,4' -f HSC-R -c src_8524_radec_R.fits \
    -o src_8254_R_multiband_test1 \
    -w -t 1.5 -m
```

- This generate a FITS catalog named: `src_8254_I_multiband_test1.fits`
- The matched catalog contains all the information from the pipeline output catalog, and the information from the input catalog. In the meanwhile, the code also put some extra information in the matched catalog:
    - `coord_ra`, `coord_dec` that can be used to compare with the input `RA`, `Dec` of fake objects.
    - `fakeId`, `fakeOffset`: ID of the fake galaxy, and the pixel separation between its match from the pipeline. [TODO: Right now, there is a bug in the `runMatchFakes.py` due to the repeat fakeId in the catalog -- the same fake model can be assigned to different coordinates]
    - `zeropoint`, `pixelScale`: Photometric zeropoint (Now it should always be 27.0); and the pixel scale that convert pixel into arcsec.
    - Convert all flux and flux error into magnitude and magnitude error, e.g.: `cmodel.mag`, `cmodel.mag.err`, `mag.kron`, `mag.kron.err`.
- Not all items in the catalog are useful, you can filter the catalog by:
    - `id > 0` should easily remove all the unmatched objects.
    - `deblend.nchild == 0 && parent == 0` can be used to select "isolated" objects to test basic photometry.
- Example: **Red dot**: Input fake objects on `Tract=8524` and `Patch='4,4'` in `HSC-I` band from the above example, using `RA`, `Dec` in the catalog; **Blue circle**: Matched objects from the pipeline, using `coord_ra`, `coord_dec`.
- Noticed that some input fake galaxies are without any match. This could due to low S/N of faint fake galaxy; or deblending process; or other complications.



- Example: You can already start using the results for some basic test. e.g. Input magnitude `mag_i` versus the magnitude difference between the pipeline output cModel magnitude and the input (`cmode.mag - mag_i`), color-coded by the input Sersic index of the fake model (`sersic_n` ; Blue--Red: Low--High)



2. Single Band Fake Star Tests

[1] Random stars with the same magnitude: `randomStarFakes.py`

- This was designed to test the completeness of the point source detection in HSC pipeline. Each time, certain numbers of the stars with the **same magnitude** will be added to the single exposure images using the PSF esimated by the HSC pipeline.

- The only difference with the above example of fake galaxy test is, during the `runAddFakes.py` step, you need a different configuration file, e.g:

```
import fakes.randomStarFakes as randomStarFakes
root.fakes.retarget(randomStarFakes.RandomStarFakeSourcesTask)
root.fakes.nStars = 100
root.fakes.magnitude = 22
```

- `nStars` : The number of the stars you want to put on a single CCD
- `magnitude` : The magnitude of these stars

- After saving this into a configuration file, e.g. `random_star_test.config` , you can add fake stars by running something like:

```
runAddFakes.py /lustre/Subaru/SSP/ \
    --rerun yasuda/SSP3.8.5_20150725:song/fake/rstar_test \
    --id visit=7288\
    --clobber-config -C random_star_test.config \
    --queue small --job addRandomStar_i_1 --nodes 1 --procs 1
```

- After this step, the rest should be very similar to the fake galaxy test.

[2] Stars from input magnitude: `positionStarFakes.py`

- This is very similar to the `positionGalSimFakes.py` procedure except the input catalog only contains four columns: `ID` , `RA` , `Dec` , `mag` . One can use the `makeSourceList.py`

Debugging:

If you want to add check that the fake source adding is working without going through all the measurements, use `debugFakes` , which takes a calibrated exposure from a completed rerun (rerun1) and writes the exposure with fakes added to rerun2.

```
debugFakes.py /to/data/ --rerun=rerun1:rerun2 --id visit=XXX ccd=YY -C config_debug
```

```
- Note that this will fail if rerun1 doesn't have the visit/ccd you are trying to process already in it. This code do
esn't do any measurements, it simply adds the fake sources to the image, which you can then open in DS9.
```

Log of Example

- The results can be found at : `/lustre/Subaru/SSP/rerun/song/fake/multi_test1`
- This 2-filter, 6-visit, 1-patch test results in **103 GB** of extra data.

Make Sources

```
makeSourceList.py /lustre/Subaru/SSP \
     --rerun=yasuda/SSP3.8.5_20150725 \
     --id tract=8524 filter='HSC-I' patch='4,4' \
     -c inputCat='cosmos_23.5_multiband.fits' \
     rejMask='ssp385_wide_8524_HSC-I_nodata_big.wkb' \
     rhoFakes=300
```

Select Visits

```
tractFindVisits.py 'yasuda/SSP3.8.5_20150725' 8254 \
    --patch='4,4' --filter='HSC-I'
```

- `7288^7310^7350^7352^7364^7378^7394^7402^14144^14164^14166^14178`

```
tractFindVisits.py 'yasuda/SSP3.8.5_20150725' 8254 \
    --patch='4,4' --filter='HSC-R'
```

- `11426^11462^11496^11498^11522^11524`

Add Fakes

- addfake_i.confg

```
from fakes import positionGalSimFakes
root.fakes.retarget(positionGalSimFakes.PositionGalSimFakesTask)
root.fakes.galList = src_8524_radec_I.fits
root.fakes.galType = 'sersic'
root.fakes.maxMargin = 150
root.fakes.addShear = False
```

```
runAddFakes.py /lustre/Subaru/SSP/ \
    --rerun yasuda/SSP3.8.5_20150725:song/fake/multi_test1 \
    --id visit=7288^7310^7350^7352^7364^7378 \
    --clobber-config -C addfake_i.config \
    --queue small --job addfake_i_1 --nodes 6 --procs 12
```

- addfake_r.confg

```
from fakes import positionGalSimFakes
root.fakes.retarget(positionGalSimFakes.PositionGalSimFakesTask)
root.fakes.galList = src_8524_radec_R.fits
root.fakes.galType = 'sersic'
root.fakes.maxMargin = 150
root.fakes.addShear = False
```

```
runAddFakes.py /lustre/Subaru/SSP/ \
    --rerun yasuda/SSP3.8.5_20150725:song/fake/multi_test1 \
    --id visit=11426^11462^11496^11498^11522^11524 \
    --clobber-config -C addfake_r.config \
    --queue small --job addfake_r_1 --nodes 6 --procs 12
```

Stack Images

- stack_i.config (Did not turn off cModel)

```
import fakes.detectOnlyFakes
root.processCoadd.detectCoaddSources.detection.retarget(fakes.detectOnlyFakes.OnlyFakesDetectionTask)
```

```
stack.py /lustre/Subaru/SSP/ --rerun=song/fake/multi_test1 --id tract=8524 filter=HSC-I patch='4,4' --selectId visit=
7288^7310^7350^7352^7364^7378 --queue small --nodes 6 --procs 12 --job stack_i_1 --clobber-config -C stack_i.config -
-config doOverwriteOutput=True doOverwriteCoadd=True makeCoaddTempExp.doOverwrite=True
```

- stack_r.config (Did not turn off cModel)

```
import fakes.detectOnlyFakes
root.processCoadd.detectCoaddSources.detection.retarget(fakes.detectOnlyFakes.OnlyFakesDetectionTask)
```

```
stack.py /lustre/Subaru/SSP/ --rerun=song/fake/multi_test1 --id tract=8524 filter=HSC-R patch='4,4' --selectId visit=
11426^11462^11496^11498^11522^11524 --queue small --nodes 6 --procs 12 --job stack_r_1 --clobber-config -C stack_r.co
nfig --config doOverwriteOutput=True doOverwriteCoadd=True makeCoaddTempExp.doOverwrite=True
```

Multiband Processing

- multiband.config

```
root.measureCoaddSources.propagateFlags.flags={}
root.clobberMergedDetections = True
root.clobberMeasurements = True
root.clobberMergedMeasurements = True
root.clobberForcedPhotometry = True
```

```
multiBand.py /lustre/Subaru/SSP --rerun=song/fake/multi_test1  \
    --id tract=8524 filter=HSC-I^HSC-R patch='4,4' \
    --queue small --nodes 1 --procs 2 --job multiband_test1 \
    --clobber-config -C multiband.config \
    --mpiexec='-bind-to socket' --time 2000
```

Match with Input

```
runMatchFakes.py /lustre/Subaru/SSP/rerun/song/fake/multi_test1 8524 \
    --ccd '4,4' -f HSC-R -c src_8524_radec_R.fits \
    -o src_8254_R_multiband_test1 \
    -w -t 1.5 -m
```

```
runMatchFakes.py /lustre/Subaru/SSP/rerun/song/fake/multi_test1 8524 \
    --ccd '4,4' -f HSC-I -c src_8524_radec_I.fits \
    -o src_8254_I_multiband_test1 \
    -w -t 1.5 -m
```