PACS photometer map-making with MADmap

PACS-401

for HIPE 11 user release Version

Babar Ali (NHSC)

NASA Herschel Science Center

NHSC PACS Web Tutorial

Introduction

- This tutorial provides a walk-through from Level 0 to 2.5 processing using the MADmap branch of the PACS photometer pipeline.
- The tutorial follows the ***ipipe*** script:

  L25_scanMapMadMap.py

- At the end of the tutorial, you will have created a PACS map from individual bolometer readouts using the optimal map-mapping algorithm MADmap.

What is MADmap?

- Mapping code written by the Berkeley CMB group to remove 1/f noise from bolometers.
  http://newscenter.lbl.gov/feature-stories/2010/02/03/madmap/

- MADmap was ported to Java for use in HIPE.
- MADmap offers the so-called optimal map-making to convert time ordered readouts to a final map.
  – Uses maximum likelihood (given a noise/probability model) to determine the optimal sky value.

When should you use MADmap?

- When spatially extended emission is present in the data. E.g. Galactic star-formation fields.
  – MADmap preserves spatially extended emission.
- When the source itself is extended. E.g. large galaxies.
- When extended structure is present around a compact source. E.g. extended halos or nebulosity.

Caveats & Warnings

- MADmap assumes that bolometer time-lines are calibrated.
  - Primarily that pixel-to-pixel instrument variations are already removed.
- MADmap assumes that 1/f noise is uncorrelated amongst pixels.
- All correlated noise (signal drift) must be removed prior to running MADmap.

Pipeline section covered here is shown in gray

NHSC PACS Web Tutorial

Level 0 container

PACS data, House Keeping

Pointing

Initial Processing

Masking: bad/ dead pixels, glitches, saturation

Data Reorganization
Logical blocks
time stamps
Unit conversion

Distortion correction

Chopper Processing
Assign offsets on sky
Flag “moving” frames

Assign pointing

Level 0.5
Minimally processed uncalibrated data cubes

Chopped AOTs

Combine & Subtract ON-OFF beams

Correct Flat Field

Flux Calibrate

Level 1 cube

Combine individual beams

Level 2 images

Mapping AOTs

(a)

High Pass Filter

Correct Flat Field

Calibrate

(b)

Subtract offsets

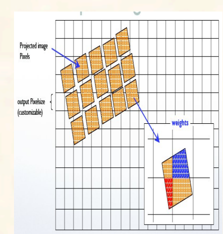
Correct Flat Field

Calibrate

Level 1 cubes

Project on sky

Optimal Map Making Algorithm



Level 2 images

Documentation Reference

- PACS data reduction guide, chapter 9
- **PACS-101**: Introduction to PACS tutorials
- **PACS-103**: Accessing & Storing PACS data
- **PACS-104**: Using iPipe scripts
- **PACS-201**: Level 0 to 1 processing of PACS photometer data

Pre-requisites:

1. You should have completed the following tutorials:
   - ***PACS-101****: How to use these tutorials.*
   - ***PACS-104****: How to access and use ipipe data reduction scripts.*
   - ***PACS-201****: Level 0 to Level 1 processing*
2. HIPE 11 user-release
3. The example dataset for RCW 120 on local disk or obtained via the HSA during the execution of the script.

Processing Overview

Step 1

Check script and software pre-requisites

Step 2

Loading ipipe script L25_scanMapMadMap.py

Step 3

Pre-amble and dataset identification

Step 4

Processing parameters

Step 5

Making sense of the main processing loop

Processing (Cont.)

Step 6

MADmap pre-processing (post Level 1)

Step 7

Remove correlated signal drifts

Step 8

Create MADmap ToD product

Step 9

Create the “naive” and optimal maps

Step 10

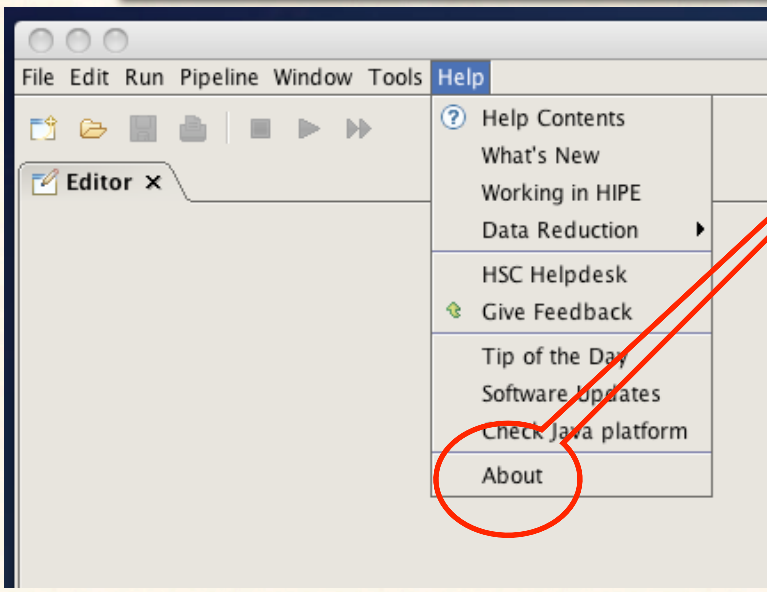
Point-Source artifact correction

Step 1

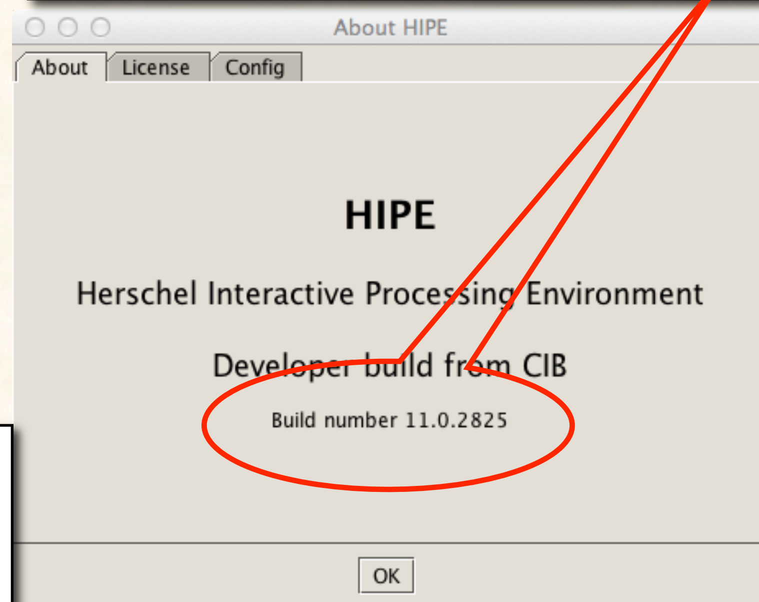
Check your software version

Check # 1: HIPE 11.0 build ….

From the "Help" menu, select "About"



Which shows the HIPE version



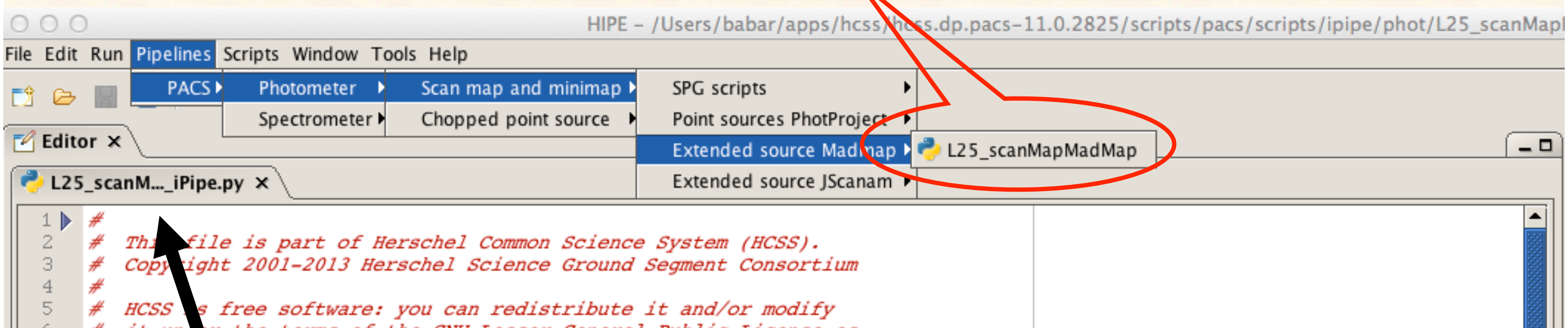
If your Build number does not start with 11, stop and upgrade.

Step 2

Load ipipe script "L25_scanMapMadMap.py"

Step 2: Load ipipe script

From the pipeline menu, make the selections as shown to get to “L25_scanMapMadMap”



If you successfully loaded the script, it’ll appear as a folder tab under the Editor window.

Step 2: Warnings

- You should always save the template ipipe script under a new name before making changes. HIPE will not allow you to overwrite the original source template.

- See **PACS-104** for details.

Step 3

Pre-amble and Identify the data (scan and cross-scan) for processing.

The preamble

Highlight and execute the block of import and definition statements with the single green arrow.

Editor ×

L25_scanM..._iPipe.py ×

```
84  """
85  from herschel.pacs.spg.pipeline   import *
86  from herschel.pacs.signal.context import *
87  from herschel.pacs.spg.all        import *
88  from herschel.ia.numeric          import *
89  from herschel.pacs.cal.util       import *
90  from herschel.ia.dataset          import LongParameter
91  from java.lang                    import Boolean
92  from java.util                    import ArrayList
93
94  from herschel.pacs.spg.phot.madmapUtils import *
95
```

The initial lines (84 in the case of example shown) are comments and may be skipped.

The import statement define java classes to be used later.

Preamble (cont.)

The next set of lines define your observation and which of the two PACS channels to use.

```
camera = 'red' # choice is 'red' or 'blue'
obsids = [ 1342185553, 1342185554 ]
obsList=[getObservation(obsids[0],useHsa=True,instrument='PACS'),\
         getObservation(obsids[1],useHsa=True,instrument='PACS')]
```

These lines are for a test dataset. Replace them with your OBSIDs, or process the test data.

Execute these statement.

See **PACS-102** for reminders

Step 4

Pre-amble and script parameters

Processing Parameters

The next segment sets run-time parameter values.

```
# ****************************************************************************
# Basic parameter which will need tuning in interactive HIPE sessions
# ****************************************************************************

ignoreFirst    = 100
model          = 1
polyOrder      = 3
binSize        = 998
verbose        = Boolean(0)
doplot         = Boolean(0)
outdir         = herschel.shar
scale          = 1.
maxRelError    = 1.e-5
maxIterations  = 500
#segmentSize = 10000
#perMatrix    = Boolean(0)
#saveData     = Boolean(0)
#showPlot     = Boolean(0)
#copy         = 0
# Point source artifacts corr
try:
  doPGLScorrection
except NameError:
  doPGLScorrection       = True
try:
  PGLS_iterations
except NameError:
  PGLS_iterations        = 3
#
```

verbose controls how much feedback is given back to you during the processing.

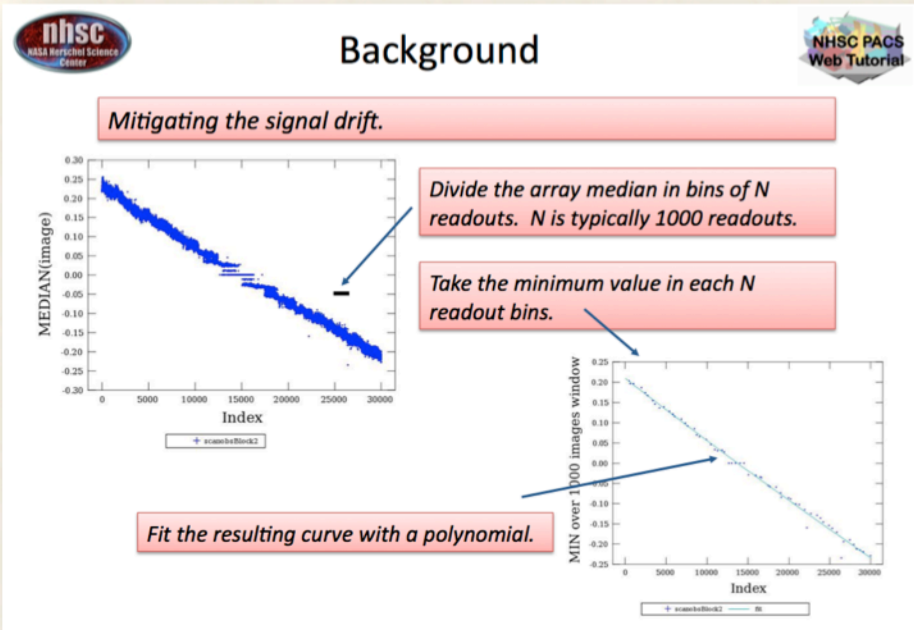
doplot allows you to view the baseline signal drift

outdir is required to store the result of doplot

Set *doplot* by replace Boolean(0) with Boolean(1)

Set *outdir* to a local directory (which must already exist).

Processing Parameters (Cont.)



Set *polyOrder* to 2 for the demo. MADmap models the drift (shown above) with a low-order polynomial.

Parameter Summary

Parameter	Description	Recommend Value
verbose	Print processing step details.	Boolean(1)
doplot	For signal drift correction, show the plot of best-fit model.	Boolean(1)
outdir	Where to save the output from 'doplot' and final maps and reduced data.	A valid directory on your system.
binSize	Size of bin for drift estimation. The minimum value in each bin is used to estimate the baseline.	998
polyOrder	Polynomial order. See step 6.	2
ignoreFirst	Mask and remove this many readouts from the start of the observation	460
scale	Scale of the output map pixels compared to PACS native pixels: 1 = use native pixel scaling of 3.2 or 6.4" per pixel for the blue and red channels, respectively. 0.3125 = 1" or 2" per pixel for the blue and red channels, respectively.	1
doPGLScorrection	Whether or not to do MADmap point-source artifact correction. See Step 11.	True, if bright point sources are present. False, otherwise.
PGLS_iterations	The number of iterations in the point-source artifact corrector algorithm. See Step 11.	

The parameters after the recommended edits.

```
ignoreFirst   = 100
model         = 1
polyOrder     = 3
binSize       = 998
verbose       = Boolean(0)
doplot        = Boolean(0)
outdir        = herschel.share.util.Configuration.g
scale         = 1.
maxRelError   = 1.e-5
maxIterations = 500
```

Highlight and execute this stanza after editing the values.

WARNING: The directory specified in 'outdir' must already exist on your system.

Point Source Artifact Correction

```
# Point source artifacts correcti
try:
  doPGLScorrection
except NameError:
  doPGLScorrection = False
try:
  PGLS_iterations
except NameError:
  PGLS_iterations  = 5
#
```

Set to ‘False’ for this demo.

Highlight & execute this stanza.

Check # 2: Preamble okay?

Execute the following line either by typing in console or editing your script.

Console

```
HIPE> print verbose, doplot, ignoreFirst, doPGLScorrection
```

History | Log | Console

```
HIPE> print verbose, doplot, ignoreFirst, doPGLScorrection
true false 100 False
HIPE>
```

The output should contain the values set in Step 3.

Step 5

MADmap pre-processing

The pre-processing loop:

A. For each observation in your scan and cross-scan pair, the following processing steps are executed:
   - Step 5: Level 0 to 1 processing.
   - Step 6: Post level 1, MADmap pre-processing.
   - Step 7: Remove correlated signal drift

B. After the processing steps, on the first pass through the loop a super frames structure is created.

C. On the next pass the cross-scan data is appended to the super frames structure

The Processing Loop

Is this the first time through?

A. Step 5

```
first        = 1
metaList     = []
#
for obs in obsList:
  level1      = PacsContext(obs.level1)
  if camera=='blue':
    tmp_frames = level1.refs["HPPAVGB"].product.refs[0].prod
  else:
    tmp_frames = level1.refs["HPPAVGR"].product.refs[0].prod
  #remove unused Noise dataset if exist (PACS-5143)
  if tmp_frames.containsKey("Noise"):
     tmp_frames.remove("Noise")
  #
  #check if frames has 2nd level deglitching mask
  hasIIdeglitchMask = tmp_frames.getMask().containsMask("2nd
  if ( not hasIIdeglitchMask):
     print "!!!2nd level glitchmask not found!  Data possibl
  #
  calTree = getCalTree(obs=obs)
  #
  nframes = tmp_frames.dimensions[2]
  print "Number of frames = ", nframes
  # Save meta data of observation
  metaList.append(obs.meta)
  #
  if (nframes >= 15000):
     ignoreFirst = 2000
  #
  tmp_frames = photMadMapIgnoreFirst(tmp_frames, ignoreFirst
  tmp_frames = photAssignRaDec(tmp_frames, calTree=calTree)
  tmp_frames = photOffsetCorr(tmp_frames)
  #
  if (nframes > 5000):
     print "do global drift correction"
     photGlobalDriftCorrection(tmp_frames, model=model, verb
         doPlot=doplot, datadir=outdir, outprefix=str(obs.ob
         order=polyOrder, binsize=binSize)
```

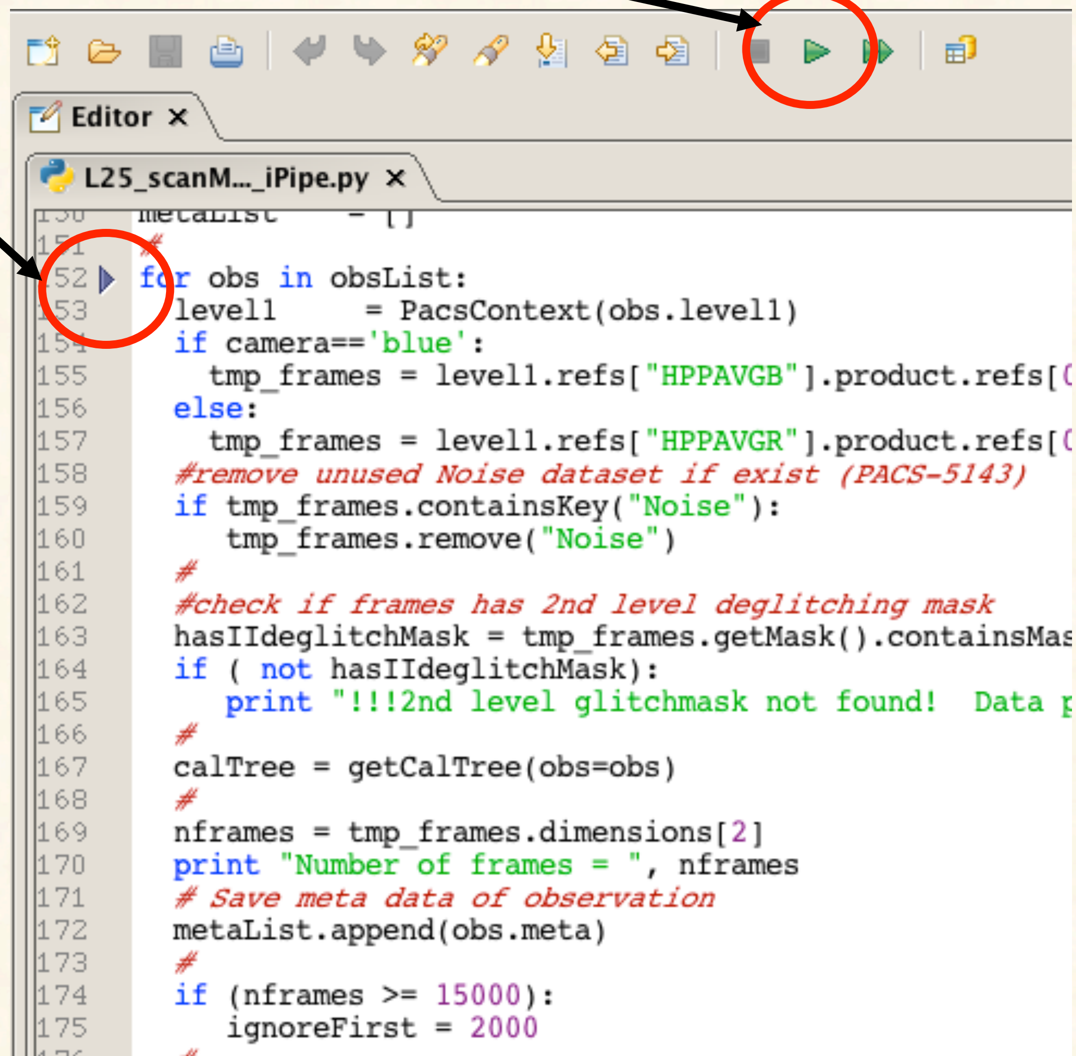
This loop will execute Step 5 identified in the previous slide for each OBSID in your list.

This is the MADmap preprocessing.

Executing the loop

Position then execute with the single green arrow.

You are encouraged to read about the steps performed in the loop in step 6 before executing the loop.



Check # 4: Position cubes exist

Issue this command in the console window

Console ×

```
HIPE>
HIPE> print frames
{description="Frames", meta=[type, creator, creationDate, description, instrument,
modelName, startDate, endDate, formatVersion, detRow, detCol, camName, relTimeOffset, Apid,
subType, compVersion, algoNumber, algorithm, compNumber, compMode, dxid,
qflag_pacs_phot_red_FailedSPUBuffer, qflag_pacs_phot_blue_FailedSPUBuffer, RemovedSetTime,
blue, chopAvoidFrom, chopAvoidTo, dec, dither, fluxExtBlu, fluxExtRed, fluxPntBlu,
fluxPntRed, lineStep, m, mapRasterAngleRef, mapRasterConstrFrom, mapRasterConstrTo,
mapScanAngle, mapScanAngleRef, mapScanConstrFrom, mapScanConstrTo, mapScanCrossScan,
mapScanHomCoverage, mapScanLegLength, mapScanNumLegs, mapScanSpeed, mapScanSquare, n,
naifid, obsOverhead, pointStep, ra, repFactor, source, fileName, obsid, obsType, obsCount,
aorLabel, aot, cusMode, equinox, missionConfig, naifId, object, obsMode, odNumber, origin,
raDeSys, telescope, level, isInterlaced], datasets=[Signal, Status, Mask, BlockTable,
History, Ra, Dec, Noise], history=Available}
HIPE>
```

Look for "dataset"s Ra and Dec

Your output will likely look slightly different but you should NOT get an error message and the important "ra" and "dec" datasets exist in your "frames" object.

Step 6

Post level 1 MADmap pre-processing

MADmap preprocessing

Executing the loop will automatically execute these steps for both OBSIDs.

Cleanup unstable frames at the start of observation.

```
#
frames = photMadMapIgnoreFirst(frames, ignoreFirst)
#
frames = photAssignRaDec(frames, calTree=calTree)
#
frames = photOffsetCorr(frames)
#
```

Assign pointing to each and every pixel and readout

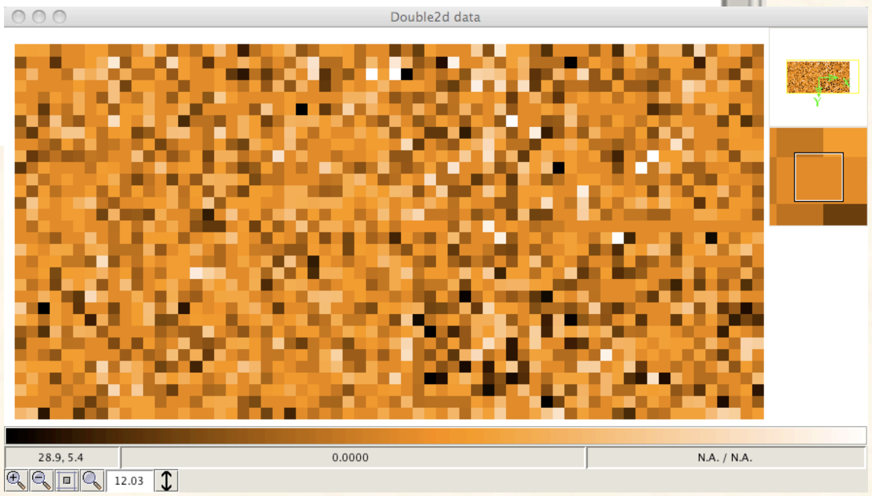
Apply pixel-to-pixel electronic offset correction.

Check # 5: Offsets are removed

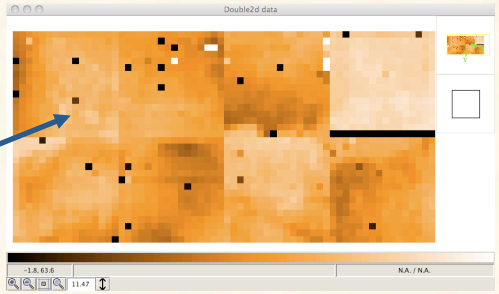
Type this command

Console

HIPE> Display(frames.signal[:,:,100])



Expected Output:
With proper offset removal, the image shows a relatively constant signal. Note: extremely bright sources may also be observed on a single image.

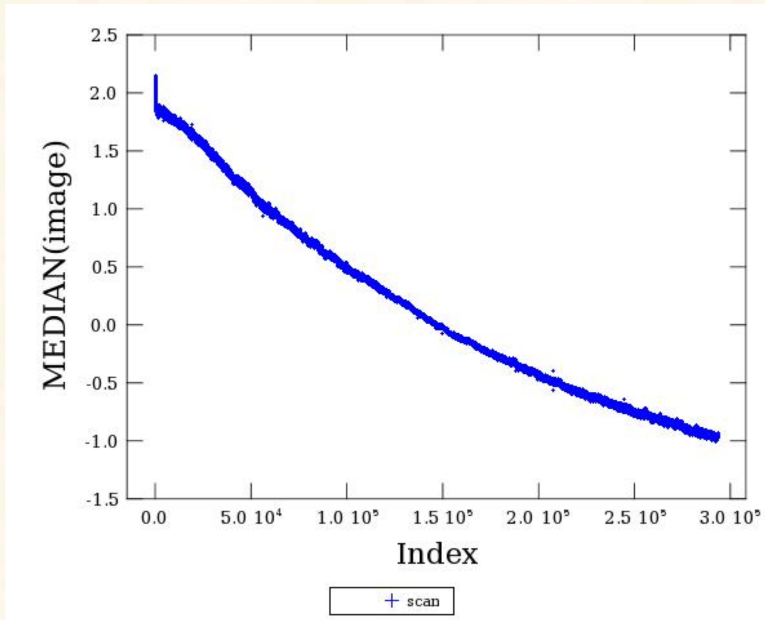


An example of improper or no pixel-to-pixel offset correction.

Step 7

Remove Correlated Signal Drifts

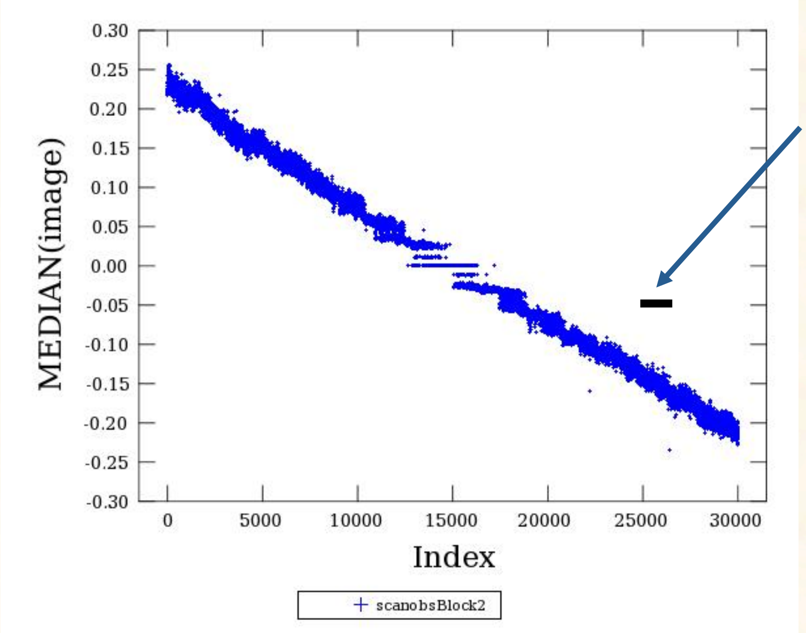
PACS' correlated signal drift.



This Figure illustrates what is meant by both correlated and drift for PACS signal. The Figure shows the median value of the bolometer array as a function of readout index. The monotonic signal showing a decay in intensity is commonly observed in PACS' image cubes, and is thought to be related to focal plan temperature drifts.

Background

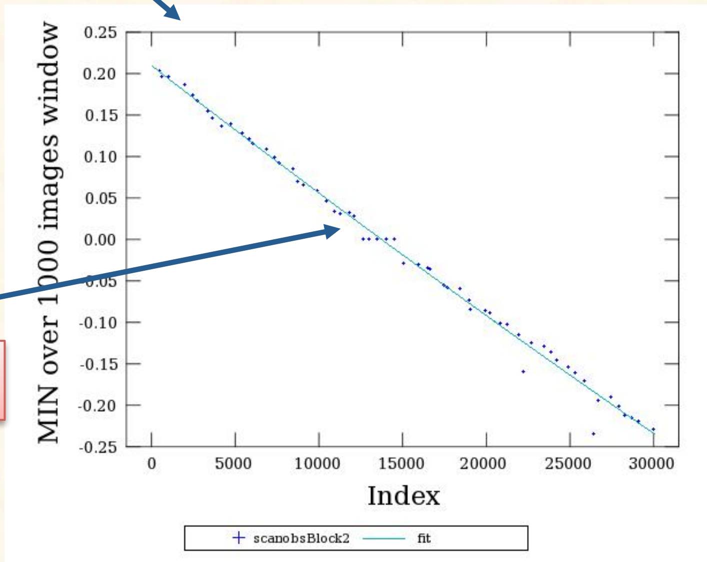
Mitigating the signal drift.



Divide the array median in bins of N readouts. N is typically 1000 readouts.

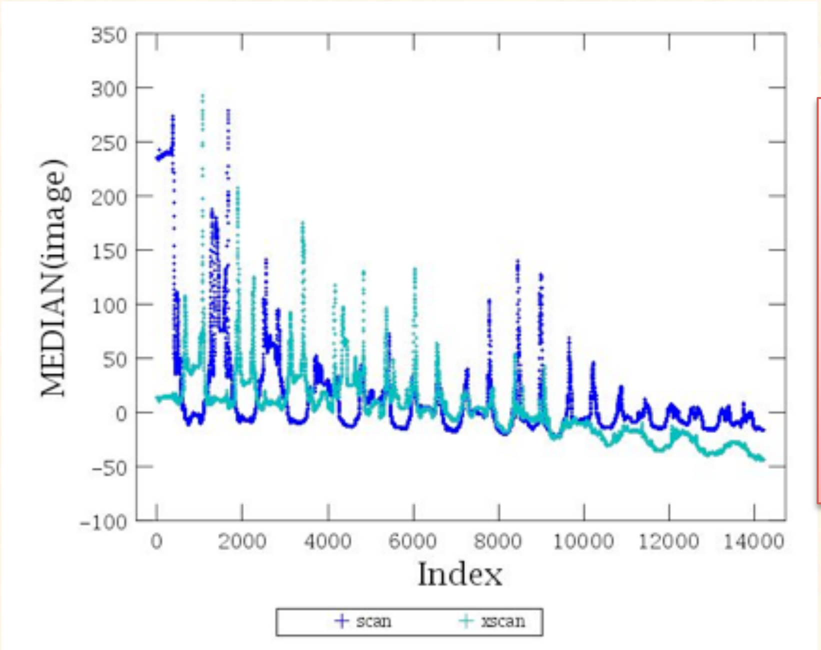
Take the minimum value in each N readout bins.

Fit the resulting curve with a polynomial.



Background

If the sources are weak (i.e. do not produce significant signal in a single image) it may be sufficient to fit the median values directly. However, for strong sources, the minimum approach becomes necessary.



An observation with strong sources. The minimum values still manage to trace the overall drift fairly accurately.

Documentation

- PACS data reduction guide, chapter 9

Step 7

The drift correction is automatically applied to the data when the main loop is exectued.

*The **photGlobalDriftCorrection** module allows several options for fitting and removing the drift.*
See: PDRG chapter 7
Or type
Print photGlobalDriftCorrection
In the HIPE window.

The most important parameters

model=1

This is the default and uses the minimum of the bins as discussed above.

polyOrder=2

Sets the order of the fitted polynomial

binSize=998

Sets the size of the bin from which the minimum value is determined.

Step 8

Create MADmap ToD product

Step 8

NHSC PACS Web Tutorial

Execute the single line

```
#
#
# Make the Time Ordered Data array
tod = makeTodArray(joinframes, calTree=calTree, scale=scale)
# To scale output pixle size or to rotate the final map:
# tod = makeTodArray(joinframes,calTree=calTree,scale=myscale,crota.
#
```

The ToD stands for Time-ordered-Data and is the internal format used by MADmap.
In fact, makeTodArray will create a binary file in your temporary area that has the rearranged PACS signal in the proper format.

- *The scale parameter selects the size of the output sky grid relative to the nominal PACS pixel sizes. E.g. scale=0.5 for PACS blue channel will result in final pixel sampling of 1.6”/pixel.*

Step 9

Create Naive and Optimal Maps, and Error planes.

Step 9

Select and execute this block of commands

```
maxRelError    = 1.e-5
maxIterations = 500

naivemap = runMadMap(tod,calTree,maxRelError,maxIterations,True)
madmap=runMadMap(tod,calTree,maxRelError,maxIterations,False)
```

Documentation Reference:

PDRG Chapter 9

Both the naive and optimal maps are created with the same call. The last parameter is set to 'True' for naive map and 'False' for optimal map. MADmap uses maximum likelihood and conjugate gradient solvers to find the optimal solution. The parameters maxRelError and maxIterations control both the convergence tolerance and the number of iterations in finding the optimal solution. See the above reference for details.

Check # 8: Output map

NHSC PACS Web Tutorial

Issue this command in the console window

Console

```
HIPE> print madmap
```

Expected output:
The output from madmap or naivemap making is a simpleImage product class with several datasets.

```
HIPE> print madmap
{description="MadMap", meta=[type, creator, creationDate, description, instrument, modelName, startDate,
endDate, formatVersion, wavelength], datasets=[image, error, exposure, History], history=Available}
HIPE>
```

Step 9

Select and execute this command.

```
# Add error map
photMadmapErrors(madmap,frames,tod,method="hspot")
#
```

Documentation Reference:

PDRG Chapter 9

The error map is generated from the coverage map using the same algorithm used to generate sensitivity estimate in HSpot. See the above reference for details.

Step 10

Correct the final map for point source artifacts

Step 10

NHSC PACS Web Tutorial

Execute the block of lines

```
#
# Do point source artifacts correction if requested
if doPGLScorrection:
   print "do point source artifacts correction"
   correctedmap = photCorrMadmapArtifacts(joinframes, tod, madmap, PGLS_iterations)
   #Display(correctedmap)
#
```

The doPGLScorrection flag is set at the beginning of the script. If set, the correctedmap variable will contain the artifact free map.
See PDRG Section 9.5 for details.
The number of interations for the PGLS algorithm are set in the PGLS_iterations variable (at the start of the script).

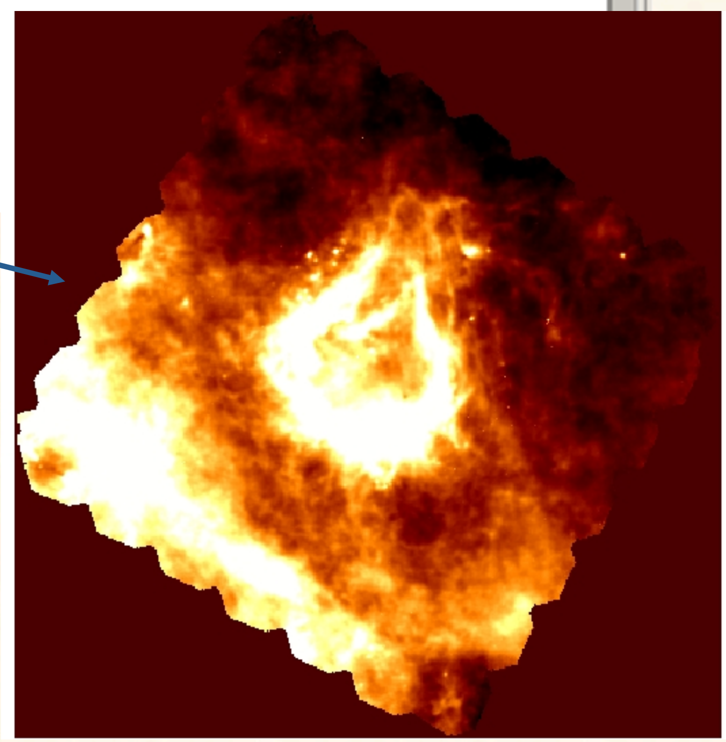
Check # 9: Display the final map

Issue this command in the console window

Console

```
HIPE> Display(madmap)
```

Expected output:
A mosiac of all images in your PACS data cube.



See PACS-202 for how to manipulate Display to show different planes in the simpleImage.