PACS photometer map-making with MADmap

PACS-401

for HIPE 12.1 user release

Babar Ali (NHSC)

Introduction

- This tutorial provides a walk-through of Level 1 to Level 2.5 processing using the MADmap branch of the PACS photometer pipeline.
- The tutorial follows the ***ipipe*** script:

  L25_scanMapMadMap_iPipe_beta.py
- At the end of the tutorial, you will have created a PACS map from individual bolometer readouts using the optimal map-making 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.

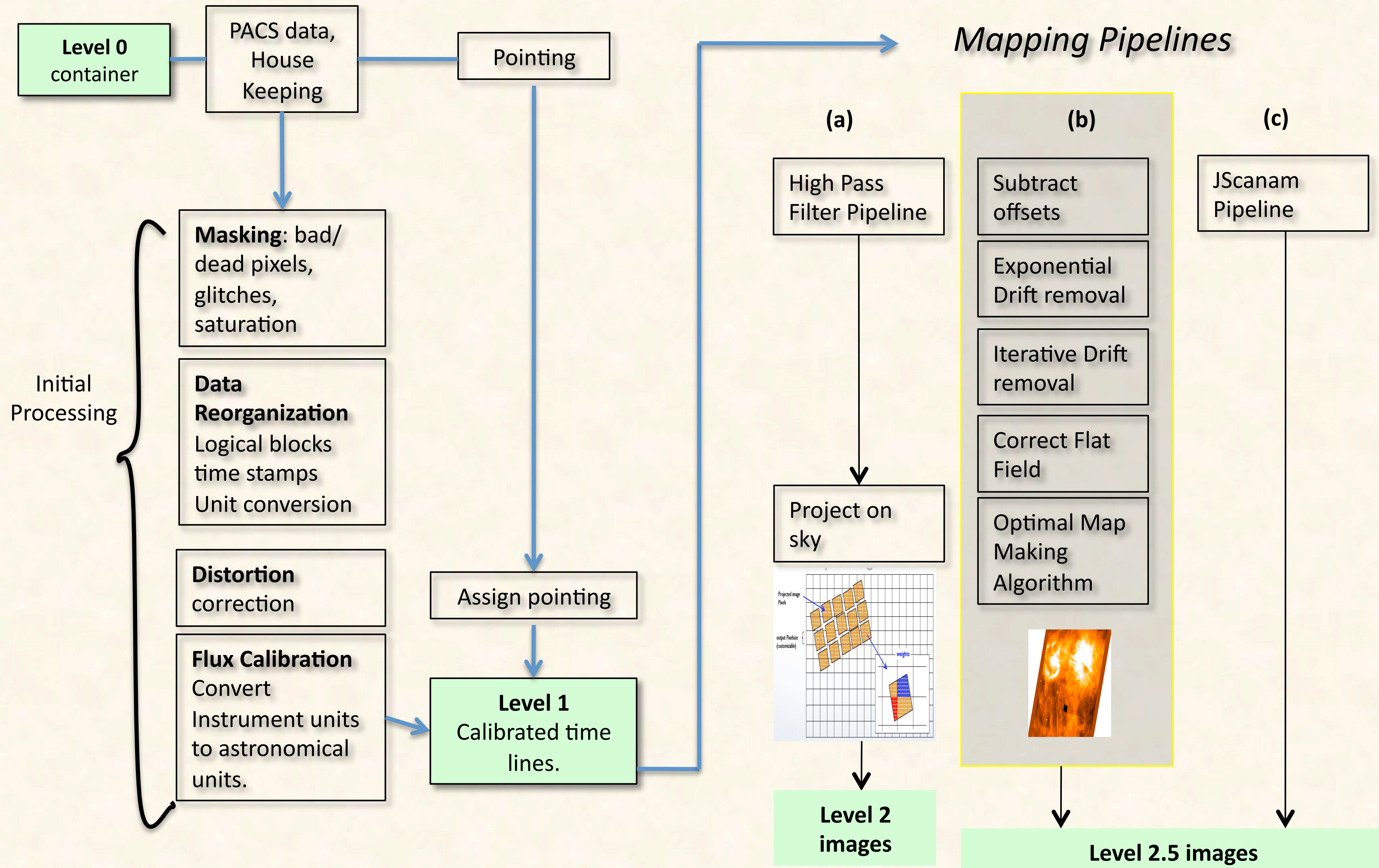
When should you use MADmap?

- 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



Documentation Reference

- PACS data reduction guide for HIPE 12
- **PACS-101**: Introduction to PACS tutorials
- **PACS-103**: Accessing & Storing PACS data
- **PACS-104**: Using iPipe scripts

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 12.1 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 MADmap ipipe script

Step 3

Pre-amble and dataset identification

Step 4

Processing parameters

Step 5

Refining the Level 1 product for MADmap

Processing (Cont.)

Step 6

MADmap pre-processing (post Level 1)

Step 7

Remove exponential signal drifts

Step 8

Remove residual drift

Step 9

Create the ToD, “naive” and optimal maps

Step 10

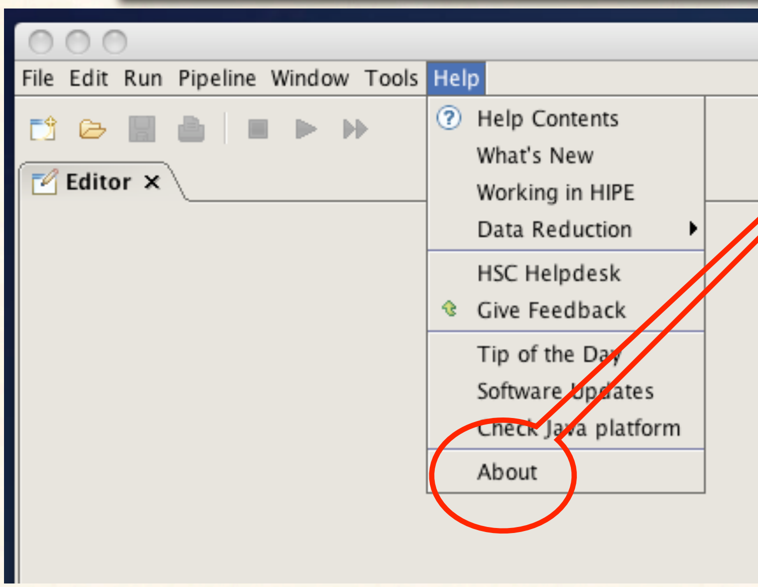
Point-Source artifact correction

Step 1

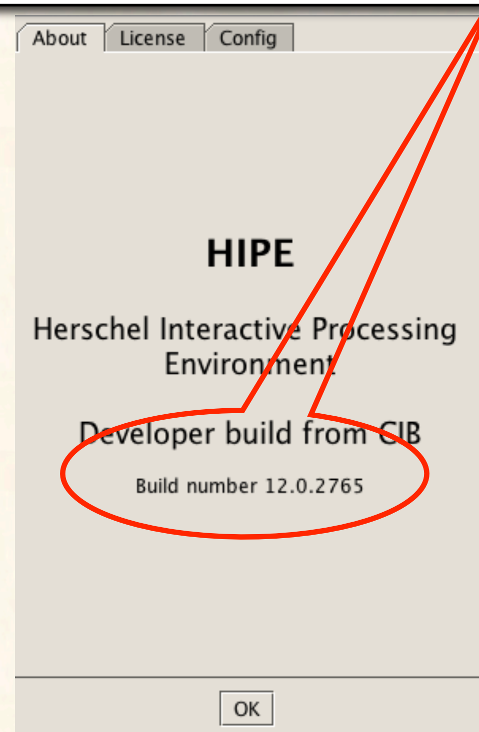
Check your software version

Check # 1: HIPE 12.1 build ....

From the "Help" menu, select "About"



Which shows the HIPE version



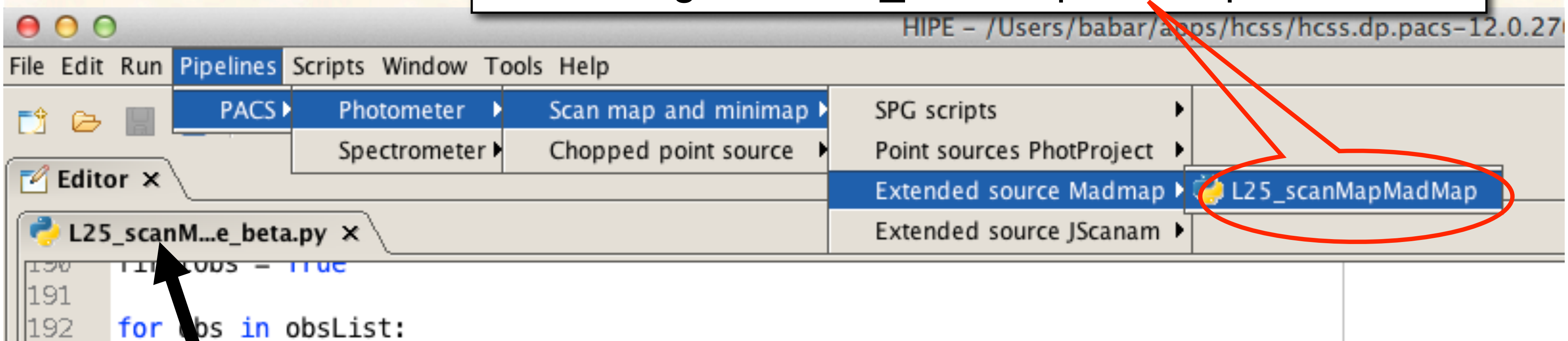
If your Build number does not start with 12, stop and upgrade.

Step 2

Load ipipe script "L25_scanMapMadMap_iPipe_beta.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.

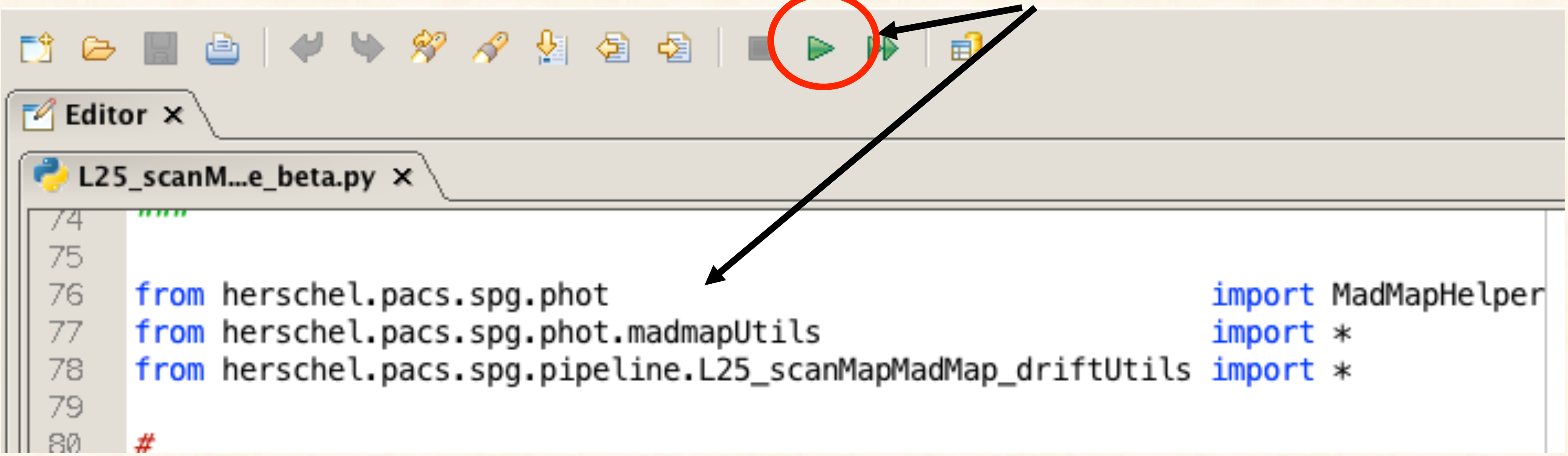
- See **PACS-104** for details.

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.



The initial lines (74 in the case of example shown) are comments and may be skipped.

The import statement define java classes to be used later.

The preamble

Next set spgMode variable to False. This simply tells HIPE you are running the pipeline interactively.

```
# Is the script being run in SPG mode?
#
spgMode = False

#
# Use the Herschel Science Archive?
# The data are assumed to be in a local pool otherwise, specified by the
# poolDir variable.
#
useHsa = True
poolDir = herschel.share.util.Configuration.getProperty("user.home") + "/.hcss/lstore/"

```

Next select source for the data. Either use the HSA, or point the ***poolDir*** *variable to a local area on your hard drive.*

Define your observation

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

```
obsids = [1342204441, 1342204443]

if not spgMode:
  obsList = []
  for obsid in obsids:
      if useHsa:
          obsList.append(getObservation(obsid, useHsa=True, instrument='PACS'))
      else:
          obsList.append(getObservation(obsid, useHsa=False, poolLocation=poolDir,

#
# Which channel to reduce?
# Choices are 'red' or 'blue'
#
camera = "red"

try:
  camera
except NameError:
  camera = "blue"
```

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.

```
useMinMedFirstGuess     = False
perPixelExpPolyFit      = True   # Fit exponential+polynomial to each individual
resetScanMedLevels      = False
deglitch                = True
nSigmaDeglitch          = 5
globalExpFitIterations = 5
nIterations             = 5
minDuration             = 150    # Minumum observation allowed in readouts.
binSize                 = 1003   # For shorter observations, script will switch
                                 # to max(25 or 0.05*# of images in cube)
#
#-----MADmap parameters
#
pixScale       = 0.5
maxRelError    = 1.e-5
maxIterations = 500
doPGLScorrection = True
PGLS_iterations  = 5
```

See next slides for meaning and recommended values for these parameters.

Drift Correction Parameters

Parameter	Description	Recommend Value
useMinMedFirstGuess	Uses the "classic" minimum median in user-defined bins to trace the drift.	False
perPixelExpPolyFit	Fit an exponential model to the initial drift.	True
resetScanMedLevels	Legacy from prototype code. Set to False. It should be removed in the next release of the code.	False
deglitch	Deglitch the time streams again? This will ignore the deglitching applied up to Level 1 processing and perform the deglitching again using the Scanamrophos algorithm.	True
nSigmaDeglitch	Definition of a glitch.	5
globalExpFitIterations	Number of iterations used to decouple signal from exponential drift.	5
nIterations	Number of iterations to use for the iterative drift removal.	5
minDuration	This is the minimum size of the timeline on which MADmap can work.	150
binSize	binSize to use with useMinMedFirstGuess.	1003

Mapping & Runtime Parameters

Parameter	Description	Recommend Value
pixScale	pixScale * 3.2 (blue) or 6.4 (red) determines the output pixel size for the projected map.	0.5
maxRelError	Threshold for stopping the iterative solution hunting using conjugate gradient method.	1.e-5
maxIterations	Maximum allowed iterations for the conjugate gradient matrix inversion.	500
doPGLScorrection	Correct for point source artifacts? If you do not have any strong point sources, set this to False.	True
PGLS_iterations	Point source artifact correction is iterative. This is how many iterations to use.	5
saveImagesToDisk	Save maps to local disk? If True, need to supply where to save the images.	True
outDir	The local path (ending with /) where to store the final maps.	A valid path ending with '/'
doPlot	Show some diagnostic plots to mark progress.	False
testRow, testCol	Use this pixel for diagnostic plots.	Any valid pixel coordinates.

The parameters after the recommended edits.

```
pixScale       = 0.5
maxRelError    = 1.e-5
maxIterations = 500
doPGLScorrection = True
PGLS_iterations   = 5

#
#-----Reporting and saving intermediate products
#

#
# Edit and change outDir variable outside this script, or take out the
# try:/except: block and replace with a outDir="/my/output/direcotry/path/"
# The last / is important.
#
saveImagesToDisk = False

try:
    outDir
except:
    outDir = herschel.share.u
```

Highlight and execute the parameter stanza after editing the values.

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

Steps 5-7

MADmap Pre-Processing Loop

The Pre-Processing Loop:

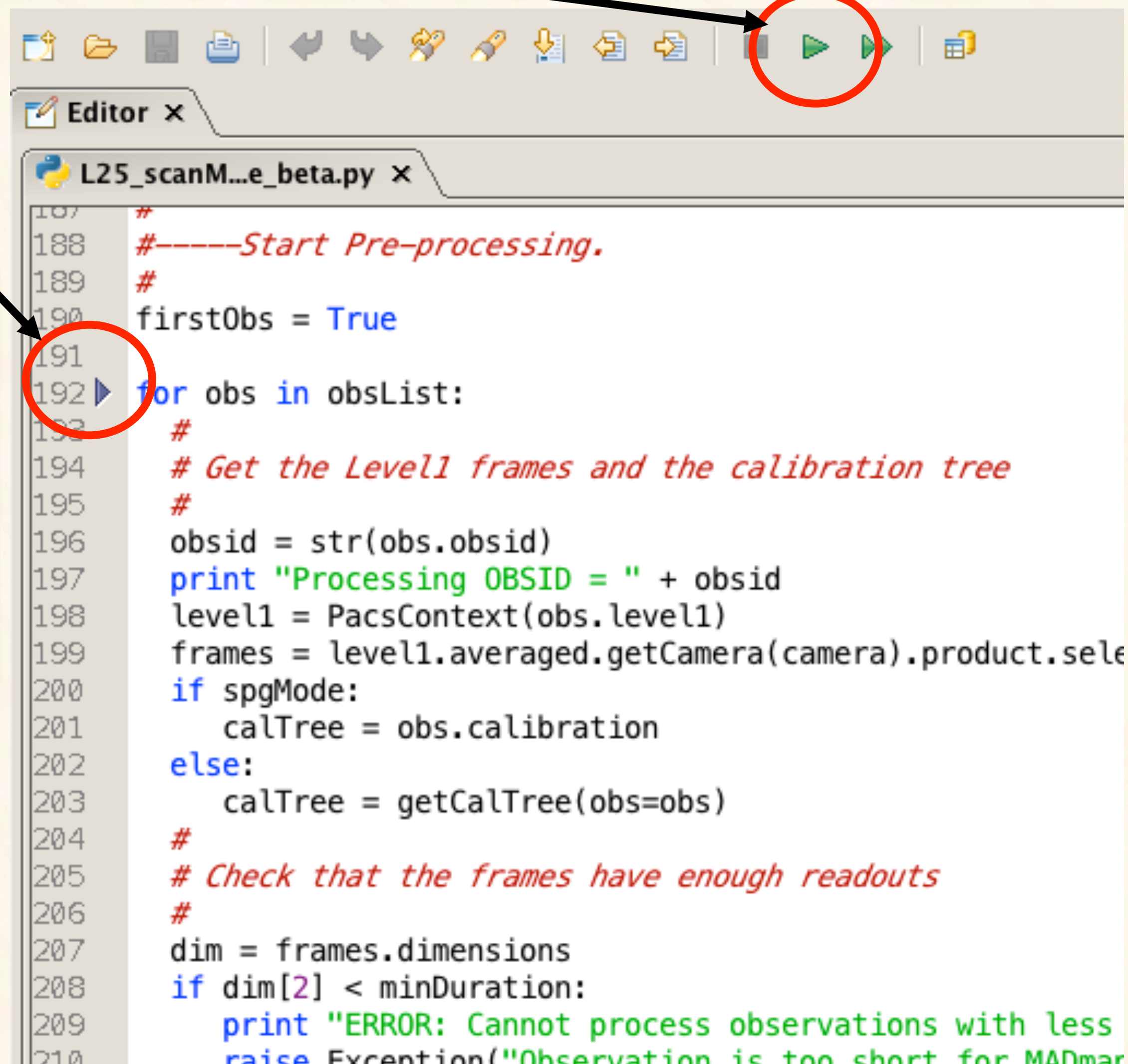
A. For each observation in your scan and cross-scan pair, the following processing steps are executed:
   - Step 5: Level 1 refinements: build pointing cube, remove glitches, and apply optical distortion correction.
   - Step 6: Remove pixel-to-pixel offsets.
   - Step 7: Apply exponential drift correction.

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

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

Executing the loop

Position then execute full loop with the single green arrow.



You are encouraged to read about the steps performed in the loop in steps 5-7 before executing the loop.

Step 5

Further refining the Level 1 product

The Pre-Processing Loop

NHSC PACS Web Tutorial

Is this the first time through?

This loop will execute Steps 5-7 identified in the previous slide for each OBSID in your list.

This is the MADmap preprocessing.

```
firstObs = True

for obs in obsList:
   #
   # Get the Level1 frames and the calibration tree
   #
   obsid = str(obs.obsid)
   print "Processing OBSID = " + obsid
   level1 = PacsContext(obs.level1)
   frames = level1.averaged.getCamera(camera).product.selectAll()
   if spgMode:
      calTree = obs.calibration
   else:
      calTree = getCalTree(obs=obs)
   #
   # Check that the frames have enough readouts
   #
   dim = frames.dimensions
   if dim[2] < minDuration:
      print "ERROR: Cannot process observations with less than " +
      raise Exception("Observation is too short for MADmap processi
   #
   # Calculate the RA and DEC datasets
   #
   print "Calculating Ra and Dec"
   frames = photAssignRaDec(frames, calTree=calTree)
   #
```

Grab Level 1 data.

This section picks up the calibration tree, and performs some checks.

Start of Step 5

The Pre-Processing Loop

More refinements at Step 5.
Remove module dropouts and apply optical distortion corrections.

```
# Flag jump/module drop outs
#
print "Detecting module jumps and drop outs"
frames = scanamorphosMaskLongTermGlitches(frames, stepAfter=20)
#
# Convert the frames signal to fixed pixel sizes
#
print "Converting the frames signal to fixed pixel sizes"
frames = convertToFixedPixelSize(frames, calTree)[0]
#
```

Check # 2: 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

Pixel-to-pixel offset correction

MADmap preprocessing

NHSC PACS Web Tutorial

Executing the loop will automatically execute this step for both OBSIDs.

```
# Remove the MEDIAN of the timeline in each pixel
#
frames.setStatus("OnTarget", Bool1d(dim[2], 1))
frames = photOffsetCorr(frames)
```

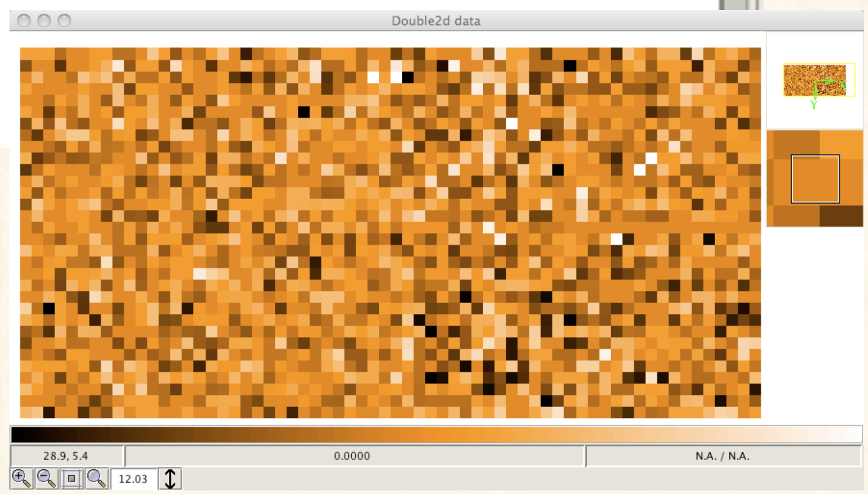
Apply pixel-to-pixel electronic offset correction.

Check # 3: Offsets are removed

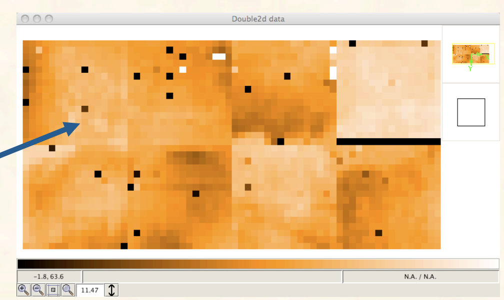
Type this command

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.



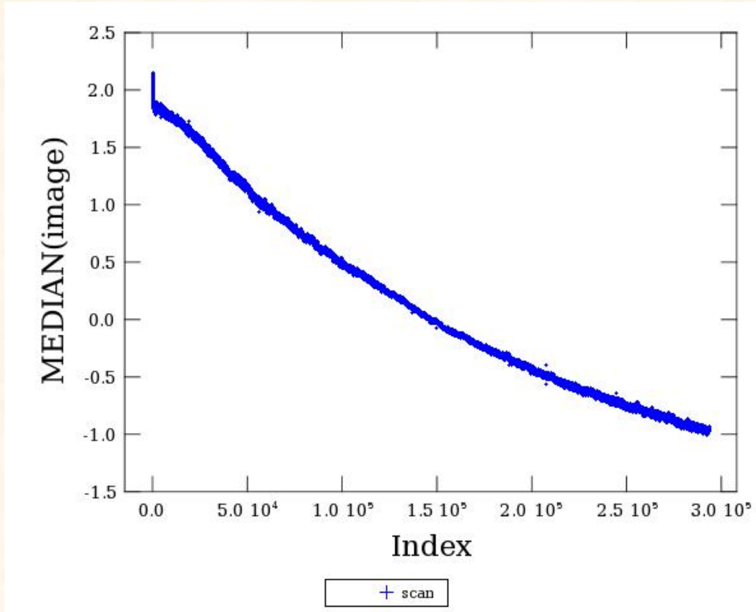
An example of improper or no pixel-to-pixel offset correction.

Step 7

Remove Exponential Signal Drift

Background

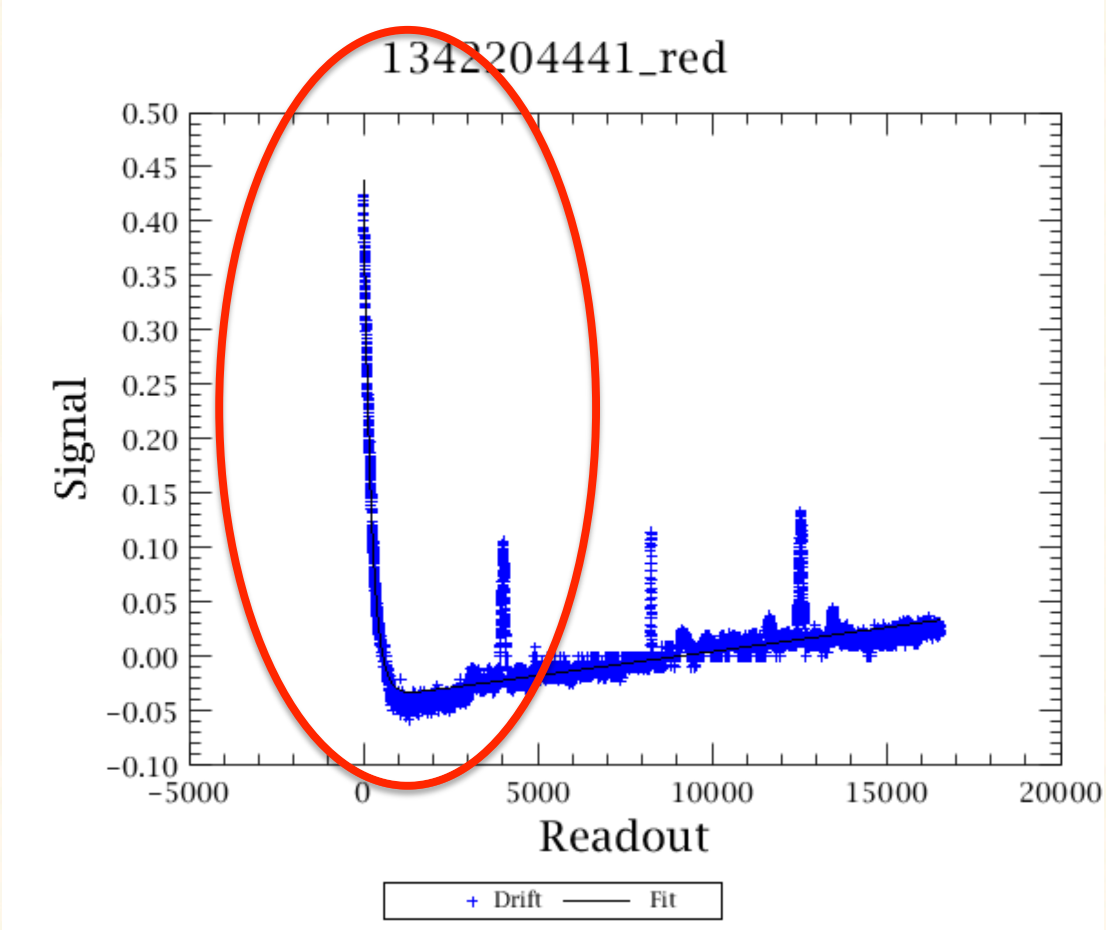
PACS' correlated signal drift.



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

The so-called fast or exponential transient

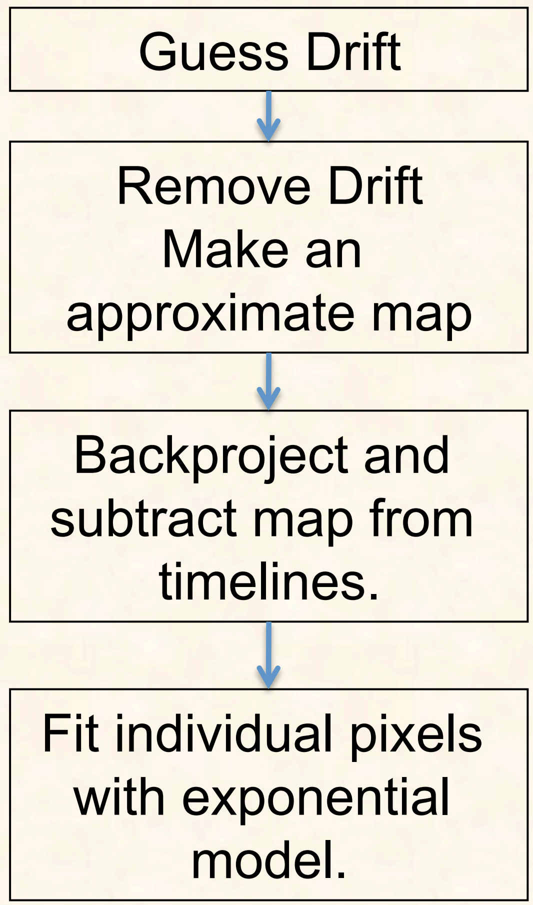


NASA Herschel Science Center

Background

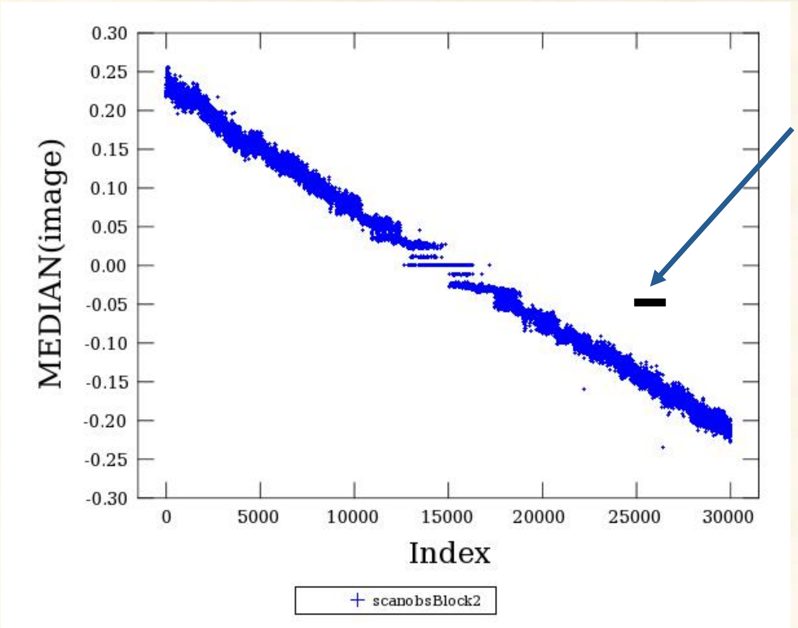
NHSC PACS Web Tutorial

Step 7: Procedure for removing fast-transient.



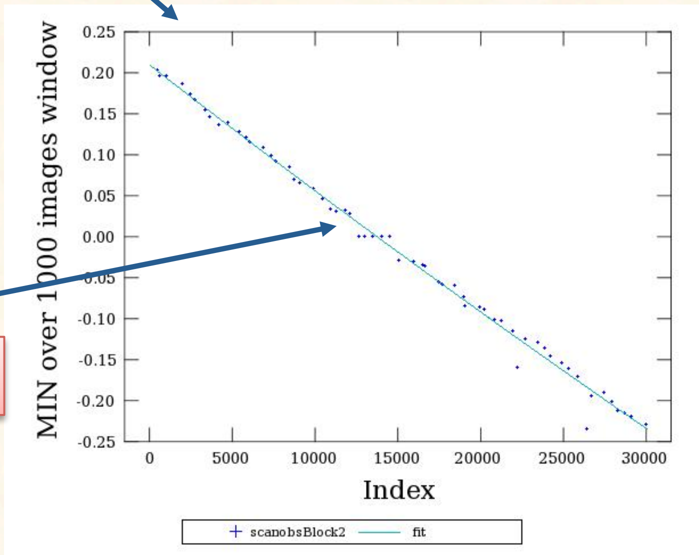
Background

Guess drift with **useMinMedFirstGuess**



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

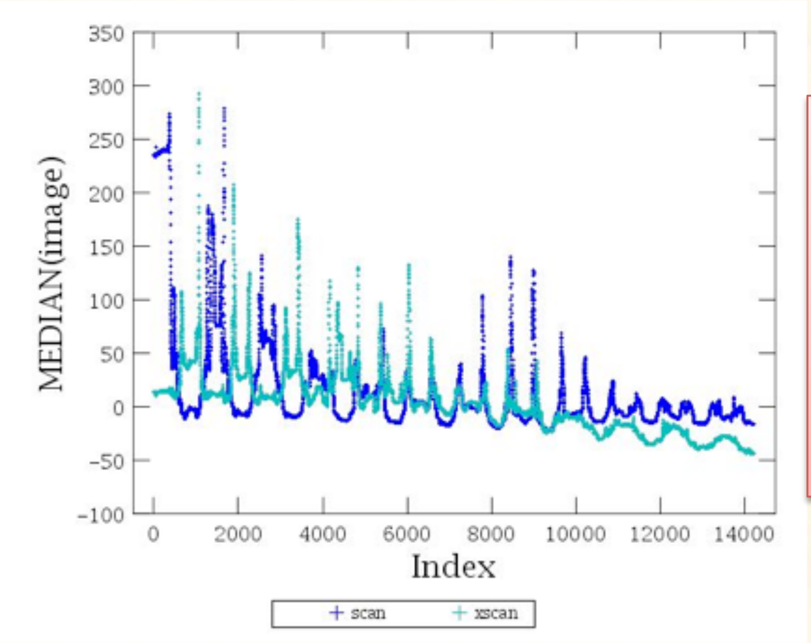
Take the minimum value in each N readout bins.



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.

Step 7

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

There are no tunable parameters for the exponential fitter.

The only choice is whether to use a first guess with **useMinMedFirstGuess** *or ignore the first guess.*

Step 7: details

```
frames = MadMapHelper.subtractDrift(frames, driftFit)
del driftFit
#
# Fit an exponential law + polynomial to each pixel timeline
#
if perPixelExpPolyFit:
   #
   # Project the frames corrected from the average drift
   #
   print "Projecting the source map"
   tod = makeTodArray(frames, calTree=calTree, scale=1)
   naivemap = runMadMap(tod, calTree, maxRelError, maxIterations, 1)
   if doPlot:
      Display(naivemap, title=obsid + ": source map")
   #
   # Back-project the source map and subtract it from the original frames
   #
   print "Subtracting the sources"
   smi = tod.getMapIndex()
   frames.setSignal(signalCopy)
   frames.signalRef.subtract(image2SkyCube(naivemap, smi)["skycube"].data)
   del tod, naivemap, smi
   System.gc()
   #
   # Pixel-to-pixel exponential drift mitigation. This time, the sources (at
   # least the bright ones) are removed from the data by the subtraction of
   # the naive map.
   #
   print "Fitting a exponential+polynomial model to the clean timelines"
   result = fitScansExpPoly(frames)
   if doPlot:
      xd = Double1d.range(dim[2])
      p = testPlot(frames, testRow, testCol,\
          title=obsid + "_row_" + str(testRow) + "_col_" + str(testCol) + "_" + camera
      testPlotAddLayer(p, xd, result["perPixelFittedCube"].data[testRow, testCol, :],\
                 java.awt.Color.black, name="Exponential+polynomial fit")
      del xd, p
   #
   # Remove the exponential + Polynomial pixel-to-pixel drift from the original frames
   #
   print "Subtracting the fits to the frames"
   frames.setSignal(signalCopy)
   frames.signalRef.subtract(result["perPixelFittedCube"].data)
   del signalCopy, result
   System.gc()
#
```

Subtract the guess fit.

Create a map.

Subtract map from timelines. What remains is drift and 1/f noise.

Fit and subtract exponential model.

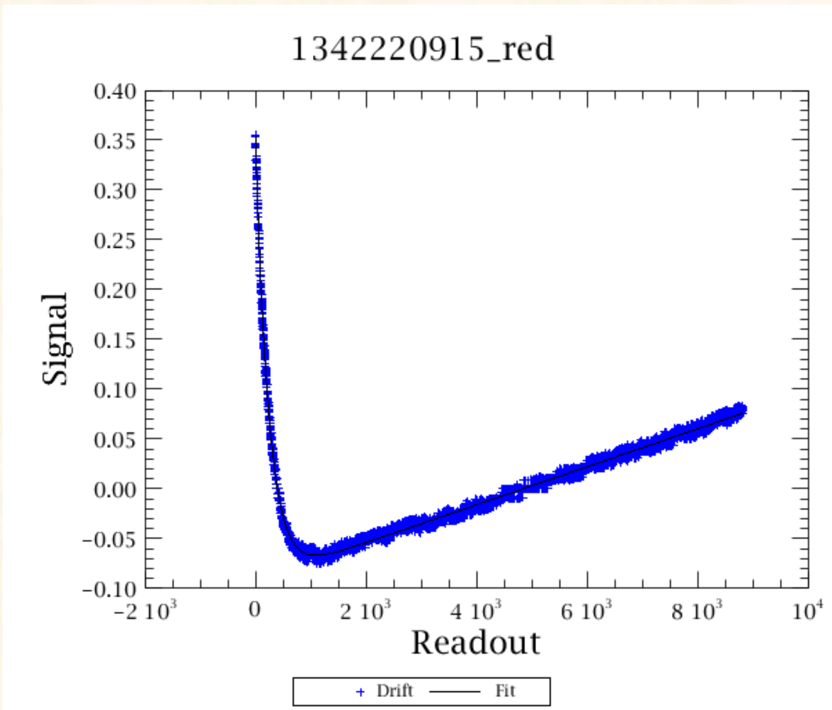
If doPlot flag is set, then diagnostic plots are made.

Step 8

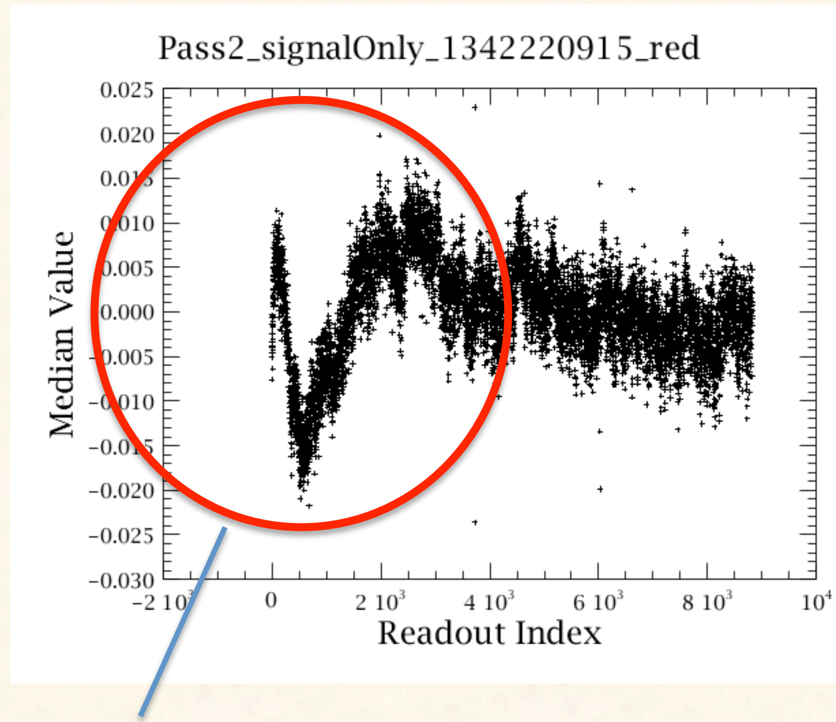
Iterative Drift Correction

Background

Exponential Drift



After Exponential Drift



Residuals in the fit. Note the scale difference in the Y-axis.

Background

- Remaining drift is mitigated by iteratively fitting a baseline (polynomial of order 1) to each scan.
- At the start of iteration, the current estimate of map is subtracted from the time line.
- The remaining drift is fit and subtracted.
- A new map is created.
- The procedure is repeated for N iterations. Where N is user-selectable parameter.

Step 8: Details

NHSC PACS Web Tutorial

```
for i in range(nIterations):
  #
  # Back-project the source map and subtract it from the original frames
  #
  mergedFrames.setSignal(signalCopy)
  mergedFrames.signalRef.subtract(image2SkyCube(sourceMap, smi)["skycube"].data)
  if doPlot:
     plotBaselineDrift(mergedFrames, perMatrix=True, saveData=saveImagesToDisk,\
          outputPrefix="Drift_iteration_%i" % (i), datadir=outDir, useMean=True,
          ytitle="MEAN (Array) drift")
  #
  # Fit a linear model to each scanleg and each pixel
  #
  result = photFitScanLines(mergedFrames, mode="perPixel")
  if doPlot:
     x = Double1d.range(mergedFrames.dimensions[2])
     title = "DriftInPixel_" + str(testRow) + "_" + str(testCol) + "_At_i=" + str
     p = testPlot(mergedFrames, testRow, testCol, title=title)
     p = testPlotAddLayer(p, x, result["fittedCube"].data[testRow, testCol, :],\
           java.awt.Color.black, name="PolyFit per scan", points=False)
     if saveImagesToDisk:
        p.saveAsPNG(outDir + title + ".png")
     del x, title, p
  #
  # Subtract the fits to the original signal
  #
  mergedFrames.setSignal(signalCopy)
  mergedFrames.signalRef.subtract(result["fittedCube"].data)
  del result
  #
  # Ensure the median of each scan is zero
  #
  if resetScanMedLevels:
     print "WARNING: Running photResetScansToZeroMedian"
     mergedFrames = photResetScansToZeroMedian(mergedFrames)
  #
  # Update the tod to the new signal and create a new source map
  #
  tod.deleteTodFile()
  makeTodArray.rewriteTod(tod, mergedFrames.signalRef)
  sourceMap = runMadMap(tod, calTree, maxRelError, maxIterations, 1)
  if doPlot:
    Display(sourceMap, title="Source map after iteration " + str(i))
  #
```

Start iterations.

Fit a linear baseline to each scan.

Set doPlot = True to see diagnostic plots.

Subtract baselines.

Create new map and repeat.

Step 9

Create ToD, Naive and Optimal Maps, and Error planes.

Step 9: Improve Deglitching

At this point, drifts have been removed.
Therefore, it is much easier to find glitches, especially lower magnitude glitches.
If the deglitch parameter is set, the Scanamorphos deglitcher is run again, prior to map-making.

```
if deglitch:
  print "Deglitching the frames"
  mergedFrames = scanamorphosDeglitch(mergedFrames, nSigma=nSigmaDeglitc
  try:
     #
     # Combine the two glitch mask
     #
     glitchMasks = String1d([mergedFrames.meta["Glitchmask"].value, "Sca
     mergedFrames.mergeMasks(glitchMasks, "Combined glitch masks")
     mergedFrames.meta.set("Glitchmask", StringParameter("Combined glitc
     del glitchMasks
  except:
     mergedFrames.meta.set("Glitchmask", StringParameter("Scanamorphos_G
```

Step 9: Create ToD

Execute the line

```
#
print "Projecting the naive map"
tod = makeTodArray(mergedFrames, calTree=calTree, scale=pixScale)
naivemap = runMadMap(tod, calTree, maxRelError, maxIterations, 1)
if doPlot:
  Display(naivemap, title="Naive map")
if saveImagesToDisk:
  FitsArchive().save(outDir + "Naivemap_" + camera + ".fits", naivemap)
```

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 Naivemap

Execute the line

```
#
print "Projecting the naive map"
tod = makeTodArray(mergedFrames, calTree=calTree, scale=pixScale)
naivemap = runMadMap(tod, calTree, maxRelError, maxIterations, 1)
if doPlot:
  Display(naivemap, title="Naive map")
if saveImagesToDisk:
  FitsArchive().save(outDir + "Naivemap_" + camera + ".fits", naivemap)
```

The Naivemap is a simple projection of the timelines onto a map container. The WCS is determined by the ToD (see above).
The Naivemap does not correct for the 1/f noise and will show the effects of the 1/f noise as stripes or checkered pattern in the final image.
The Naivemap is useful for checking whether the timelines themselves contain artifacts that have not been cleaned up.

Step 9: Optimal Map-making

Select and execute this block of commands

```
print "Running MadMap"
madmap = runMadMap(tod, calTree, maxRelError, maxIterations, 0)
photMadmapErrors(madmap, mergedFrames, tod, method="hspot")
if doPlot:
  Display(madmap, title="MadMap map")
if saveImagesToDisk:
  FitsArchive().save(outDir + "MADmap_" + camera + ".fits", madmap)

```

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..

Check # 4: Output map

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: Errors for MADmap

NHSC PACS Web Tutorial

Select and execute this command.

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

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

Execute the block of lines

```
if doPGLScorrection:
  print "Running point source artifacts correction"
  correctedmap = photCorrMadmapArtifacts(mergedFrames, calTree, tod, madmap, PGLS_it
  if doPlot:
    Display(correctedmap, title="PGLS corrected map")
  if saveImagesToDisk:
    FitsArchive().save(outDir + "correctedmap_" + camera + ".fits", 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 for details.
The number of interations for the PGLS algorithm are set in the PGLS_iterations variable (at the start of the script).

Check # 5: Display the final map

Issue this command in the console window

Console

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

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