AMI SIM

pyami

drive_binary.py

deltaX (-dx)
deltaY (-dy)
spectralType (-st)
fluxRatio (-fr)
oversample (-O)
uptheramp (-utr)
targetMagnitude
totalElectrons
filt

generate PSF
star_array
generate star PSF
create calib star
calc dither positions
create binary
tcube{trials}{dither}.fits
sim skydata (binary)
ccube{trials}{dither}.fits
sim skydata (calib)

etc

NIRISSami_apt_calc_v3.py
generatePSF

simcode

make_binary.py

generate_starPSF
generate_starPSF

make_scene.py

simulate_scenedata
check psf and sky are same size
Check that oversampling exactly divides sky scene
Check that native size image is odd
Get the oversampled ips map (set to ones)
FFT (sky_ov, psf_ov)
Get acculated position errors due to dithering
Get jitter errors
jitter
adjustsizes

Loop around dithers + integrations

work out position + total errors
get ips weighted skyscene
get the ips map section for this dither/int position
sky_ov (ipsw)
ips_ov
apply padding (dithers)?
apply_padding_image
apply_padding_ips
apply padding (+1px)
rebin (+1px image)
rebin
sky_rb (ipsw)
cut down image (+1px image to image)
ips_rb
flat field sky-rb with ips-rb =(sky_rb / ips_rb)
create ramps
get flatfield
get_flatfield
create integration
create_integration
"reduce data" remove dark/background/flat
store in cube

Populate header
Save cube to file
{cube_name}_{filename}{trials}{dither}.fits

utils.py

driver_scene.py

filter (-f)
psf (-p)
sky (-s)
nint (-I)
ngroups (-G)
create_calibrator (-c)
countrate (-cr)
uptheramp (-utr)
Creating calibrator should be separate process

load PSF data
load sky data
get dither position sequence from user inputs
from dither sequence name read a lookup table to get dither positions
construct output filename
Should be compatible with Webb Pipeline compute headers here too?
simulate scene (data)
t_{filename}{trials}{dither}.fits
simulate scene (calib)
c_{filename}{trials}{dither}.fits

create_ramp
display counts per second
display counts per frame
nreadouts = ngroups + 1 (first readout has no scene component)

loop around iread (0 to nreadouts)

iread=0
generate readnoise_cube normal(0, readnoise, size)
rd_cube
add rd_cube[iread]

iread>0
generate photon expectation (sky_rb * tframe)
generate photon expectation + poisson noise
pn_cube
calculate background
bg_cube
calculate dark current
dk_cube
generate readnoise_cube normal(0, readnoise, size)
rd_cube
add ramp[iread-1]
add pn_cube[iread]
add dk_cube[iread]
add rd_cube[iread]

ramp
return the ramp