Exploring the Data

08/07/2025

I had an idea to compare NIMO vs Swarm and PyIRI vs Swarm to
NIMO (+1 day) vs Swarm to PyIRI (+1 day) to Swarm respectively.
Basically, I want to see what kind of intricacies NIMO and PyIRI catch.
Ideally, the models should match Swarm of the same day better than
the models (+ 1day) match Swarm.
The reason I did this is beacuse I had a suspicion that PyIRI would
Be fairly similar no matter what day I use and NIMO would be more different
hence showing that even if NIMO does worse in some tests, maybe it does better
in a test of accuracy.
For this reason, I created a separate code
offset_codes.NIMO_SWARM_mapplot_offset
In this, you can specify the number of days to offset NIMO compared to swarm
e.g. if the offset is 1 and Swarm is at January 5, then NIMO data at January 6 is
used.
The offset can also be negatvie indicating a decrease in days and a larger number
than 1
A word of caution, save these files in a separate location from your original daily
files
because the filenames will be the same to make open_daily_files.open_daily
work.
The figures will have a different name.
Once you have new daily files, you can run SwarmPyIRI.PyIRI_NIMO_SWARM_plot
to get
the offset PyIRI files since that one is based off of the same time as NIMO
Then you can run some stats as shown below and as outlined in
Swarm_Stats_Walkthrough.

In [2]:
```
import pandas as pd
from datetime import datetime, timedelta
# Self Created Functions -------------------------
# Swarm downlaod and load functions
from download_swarm import download_and_unzip

# Plotting NIMO and Swarm together
from NIMO_Swarm_Map_Plotting import find_all_gaps, NIMO_SWARM_mapplot
from NIMO_SWARM_single import nimo_swarm_single_plot
from SwarmPyIRI import PyIRI_NIMO_SWARM_plot
```

```
from swarm_panel_ax import swarm_panel
import matplotlib.pyplot as plt

from Swarm_Stats import states_report_swarm, LSS_plot_Swarm, map_hist_
from Swarm_Stats import decision_table_sat, style_df_table, HMFC_perce
from Swarm_Stats import style_LSS_table, LSS_table_sat
from offset_codes import NIMO_SWARM_mapplot_offset
```

In [ ]:
```
# Create new NIMO files
fig_dir='~/Plots/NIMO_SWARM_offsets'
file_dir='~/Type_Files/Daily_offsets'
swarm_fdir = '~/swarm_data'
nimo_fdir='~/data/NIMO/'

mlat_val = 30
stime1 = datetime(2020, 4, 15, 0, 0)  # Starting Date

for i in range(15):  # How many days you want to make files for
    stime = stime1 + timedelta(days=i)
    print(stime)
    df = NIMO_SWARM_mapplot_offset(
        stime, swarm_fdir, nimo_fdir, offset=1, file_dir=file_dir,
        fig_on=False, fig_dir=fig_dir)
```

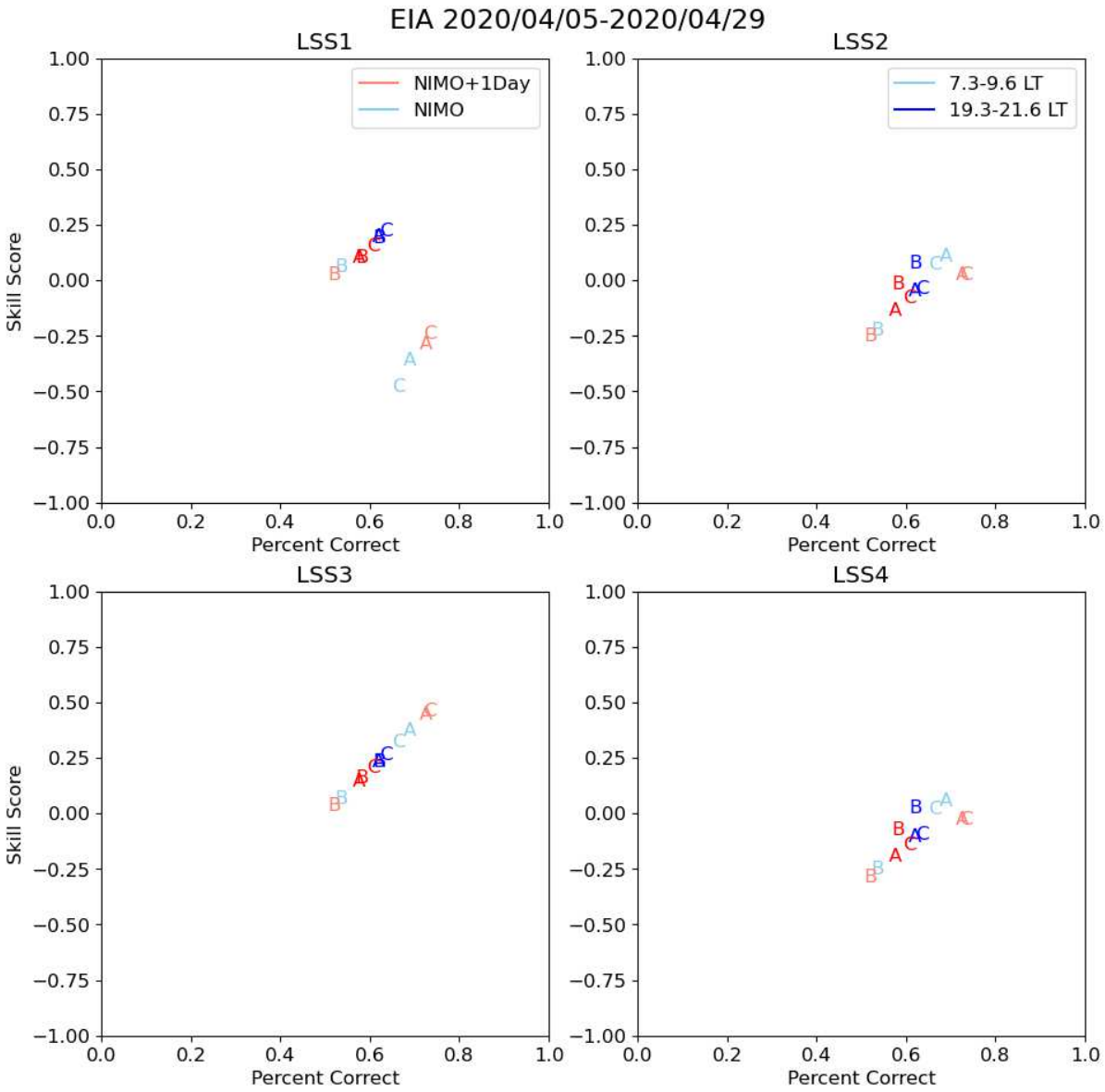
```
2020-04-15 00:00:00
```

In [ ]:
```
# Create new PyIRI files
# PyIRI files are created using NIMO info, so you don't need to specif
# Save the files in a separate folder from original Daily files
fig_dir='~/Plots/NIMO_SWARM_offsets'
daily_dir='~/Type_Files/Daily_offsets'
swarm_fdir = '~/swarm_data'

stime1 = datetime(2020, 4, 15, 0, 0)

for i in range(15):  # How many days you want to make files for
    stime = stime1 + timedelta(days=i)
    print(stime)
    pdf = PyIRI_NIMO_SWARM_plot(stime, daily_dir, swarm_fdir, fig_on=T
                                fig_save_dir=fig_dir, file_save_dir=da
```

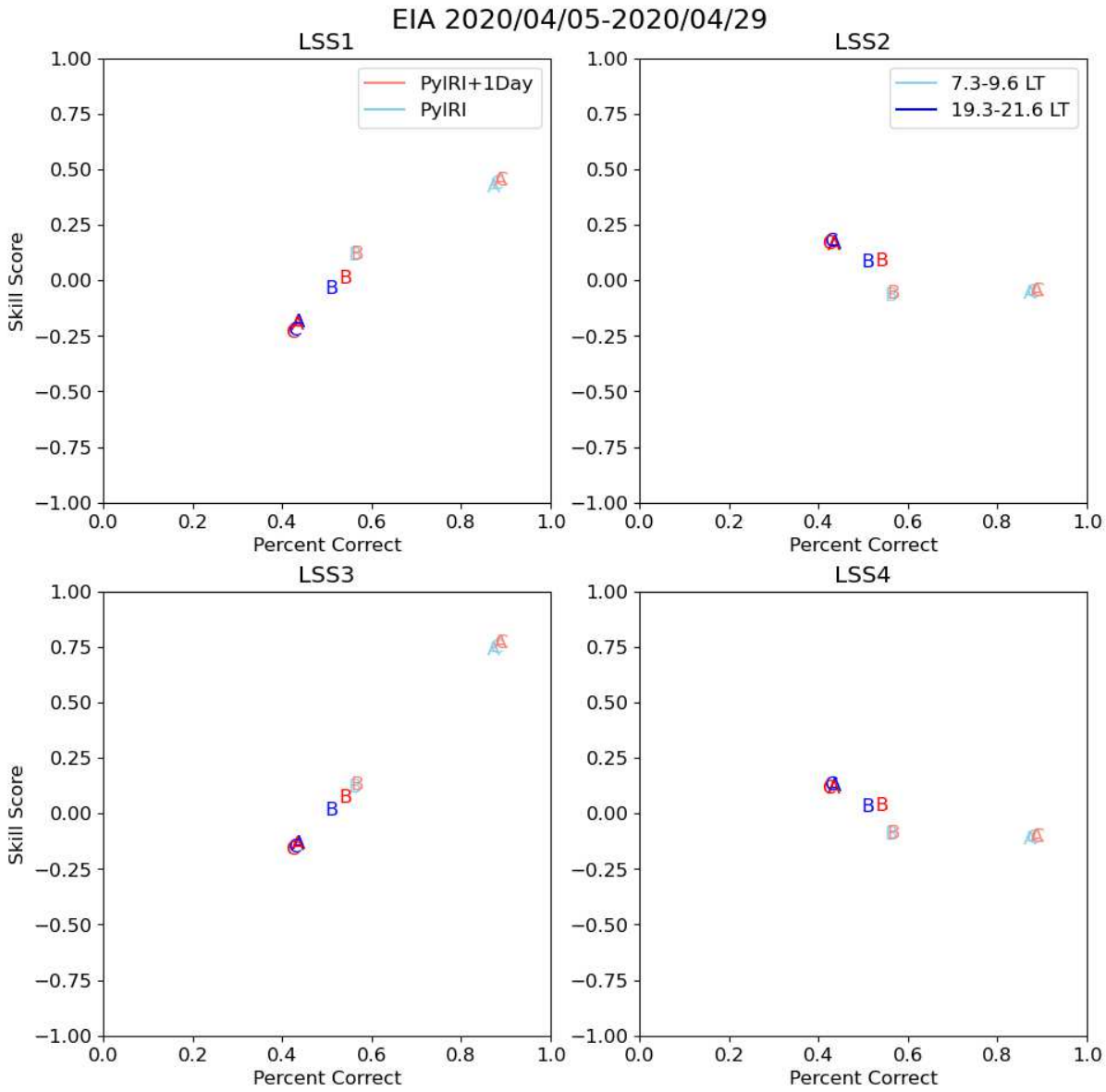
In [3]:
```
# Original data states_report
date_range = pd.date_range(start='2020-04-05',  end='2020-04-29')
daily_files = '~/Type_Files/Daily'
Nimo_og, Sw_og, PyI_og = states_report_swarm(date_range, daily_files,
```

In [4]:
```
# offset by 1 day states report
date_range = pd.date_range(start='2020-04-05',  end='2020-04-29')
daily_files = '~/Type_Files/Daily_offsets'
Nimo_1d, Sw_1d, PyI_1d = states_report_swarm(date_range, daily_files,
```

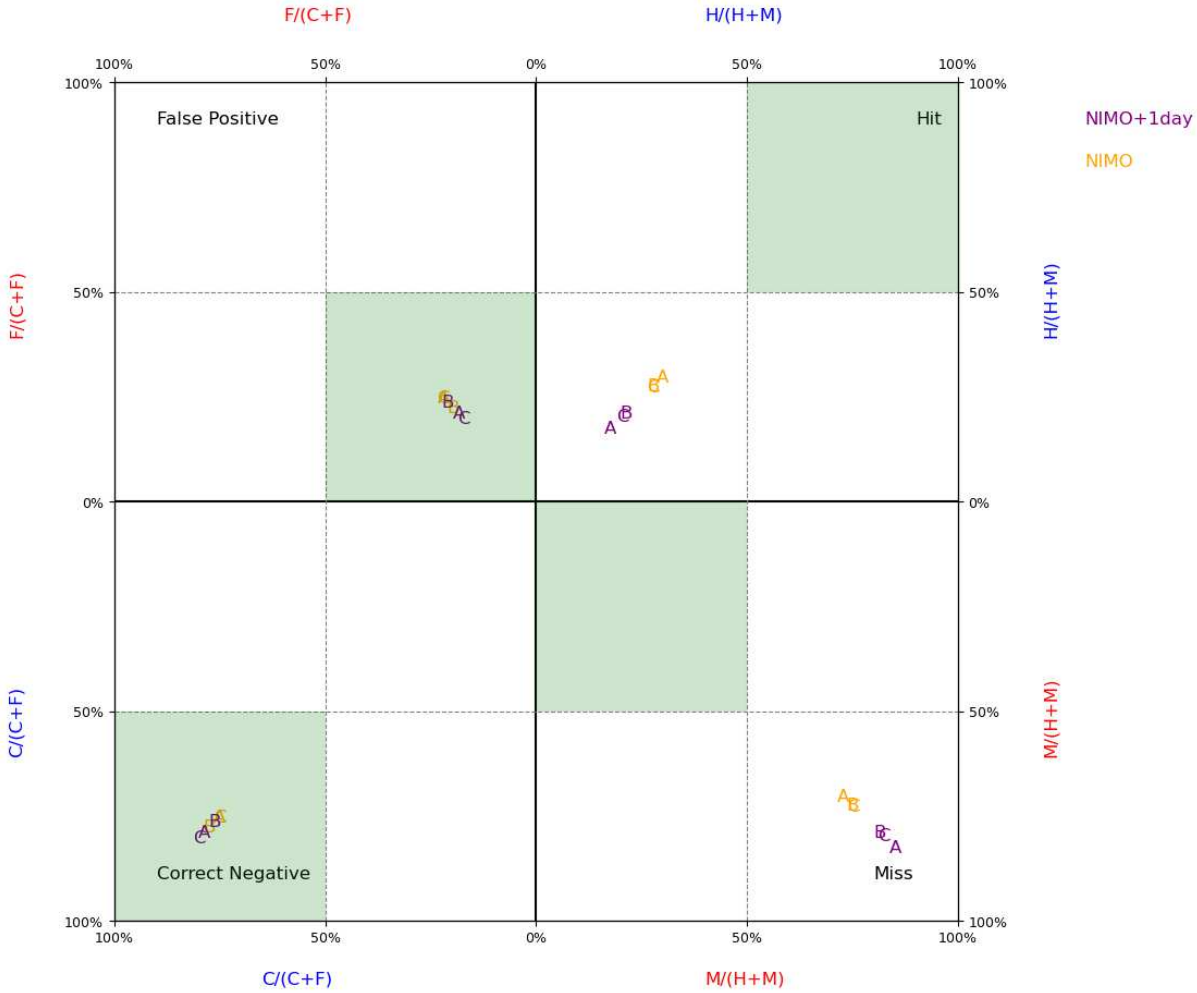
In [6]:
```
# Comparing NIMO original to NIMO 1 day offset
plt.rcParams.update({'font.size': 12})
fig = LSS_plot_Swarm(Nimo_og, Nimo_1d, 'EIA', date_range, model1_name=
```



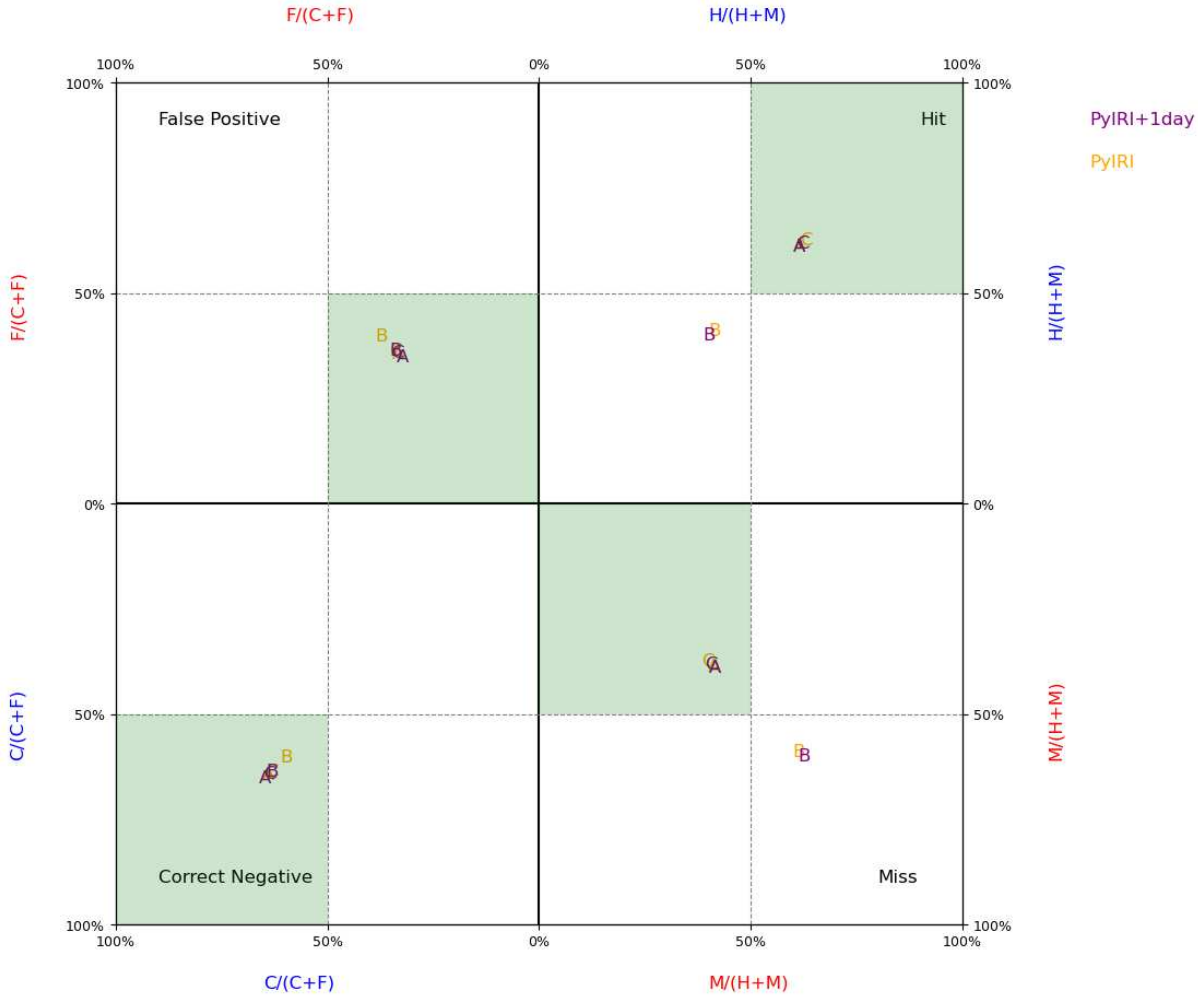
In [7]:
```
# Comparing NIMO original to NIMO 1 day offset
plt.rcParams.update({'font.size': 12})
fig = LSS_plot_Swarm(PyI_og, PyI_1d, 'EIA', date_range,
                     model1_name='PyIRI', model2_name='PyIRI+1Day',
                     PorC='PC', coin=False)
```



In [28]:
```
fig = HMFC_percent_figure(Nimo_og, Nimo_1d, 'eia', model1_name='NIMO',
```



In [29]:
```
fig = HMFC_percent_figure(PyI_og, PyI_1d, 'eia', model1_name='PyIRI',
```


F/(C+F)
H/(H+M)
False Positive
Hit
Correct Negative
Miss
PyIRI+1day
PyIRI
C/(C+F)
M/(H+M)
100%
50%
0%

In [ ]:

SWARM NIMO

In [1]:
```
import pandas as pd
from datetime import datetime, timedelta
# Self Created Functions ------------------------
# Swarm downlaod and load functions
from download_swarm import download_and_unzip

# Plotting NIMO and Swarm together
from NIMO_Swarm_Map_Plotting import find_all_gaps, NIMO_SWARM_mapplot
from NIMO_SWARM_single import nimo_swarm_single_plot
```

Example of Downloading Swarm File

Function : download_swarm.download_and_unzip
Required Parameters

> ymd : datetime
> satellite : string 'A', B', or 'C'
>
> > Any other satellite letter will return file does not exist
>
> out_dir : string directory for output

Key Word Arguments

> bse_url : URL where data can be found
>
> > Defualt 'https://swarm-diss.eo.esa.int/?do=download&file=swarm%2FLevel'
> > The base URL to use can be found by going to https://swarm-diss.eo.esa.int/#
> > and navigating to desired file, right clicking and choosing "Copy Link Address"
> > Use string before the level is specified
>
> level : product level
>
> > Default '1b' can also use '2daily'
>
> baseline : product baseline
>
> > 'Latest_baselines' is recommended, has not been tested for 'Entire Mission Data'

> instrument: Instrument type
>
> > Default 'EFI' (Electric Field Instruments
>
> instrument2: Speficic instrument
>
> > Default 'LP' (Langmuir Probe)
>
> f_end : file ending
>
> > Default '0602_MDR_EFI_LP'
> > 0602 represents the file version
> > MDR_EFI_LP represents the Record Type
>
> T1 string starting time
>
> > Default '000000' (midnight)
> > MOST swarm files will follow this format but NOT ALL
>
> T2 string ending time
>
> > Default '235959' (1 minute before midnight)
> > MOST swarm files will follow this format but NOT ALL
>
> num_days : number of days that will be downloaded after initial file
>
> > Default is 0

File will be downloaded if it does not already exist in out_dir
File will not be downlaoded if that filename dows not exist on Swarm website
NOTE: often if a file does not exist, it is because T1 is not '000000' and
T2 is not '235959'
If that is the case, check the Swarm Data Website to find the proper times

In [3]:
```
out_dir = '~/swarm_data/'
fdate = datetime(2020, 12, 29)
download_and_unzip(fdate, 'A', out_dir)
```

```
Downloading: SW_OPER_EFIA_LP_1B_20201229T000000_20201229T235959_0602.CD
F.ZIP
Extracted to: /Users/aotoole/Documents/Python_Code/EFI/Sat_A/2020/20201
229
```

Single NIMO/Swarm Plot Example

Function NIMO_SWARM_single.nimo_swarm_single_plot

This makes a single plot for Swarm and NIMO as close to the provided time as possible
If Swarm file is not found then it will attempt to download

Required Parameters

> stime : datetime to plot
> satelltie : string 'A', B', or 'C' for Swarm
> swarm_file_dir : file directory for Swarm data
> nimo_file_dir : file directory for NIMO data

Key Word Arguments

> MLat : Magnetic Latitude cutoff
>
> > 30° Default
>
> swarm_filt : filter for swarm data
>
> > Default is 'barrel_average'
>
> swarm_interpolate : linear interpolation parameter
>
> > the number of data points will incrase by swarm_interpolate
> > Default is 1 (no interpolation)
>
> swarm_envelope : boolean
>
> > determines if an envelope is used if barrel is in filter
> > Default is True
>
> swarm_barrel : double deteriming magnetic latitude radius of barrel
>
> > Default is 3°
>
> swarm_window : double determing magnetic latitude moving average window size
>
> > Default is 2°
>
> nimo_filt : filter for nimo data
>
> > Default '' (no filter)
>
> nimo_interpolate : linear interpolation parameter

> the number of data points will incrase by swarm_interpolate
> Default is 2 (doubles number of points)

nimo_envelope : boolean

> determines if an envelope is used if barrel is in filter
> Default is False (no envelope)

nimo_barrel : double deteriming magnetic latitude radius of barrel

> Default is 3°

nimo_window : double determing magnetic latitude moving average window size

> Default is 3°

fosi : int for plot font size

> Default 18
> Exceptions:
>
> > Super Title (fosi + 10)
> > legends (fosi - 3)

out_dir : string of output directory

> if it is left empty ('' default), then cwd will be used

nimo_name_format : string specifying nimo filename before '.nc'

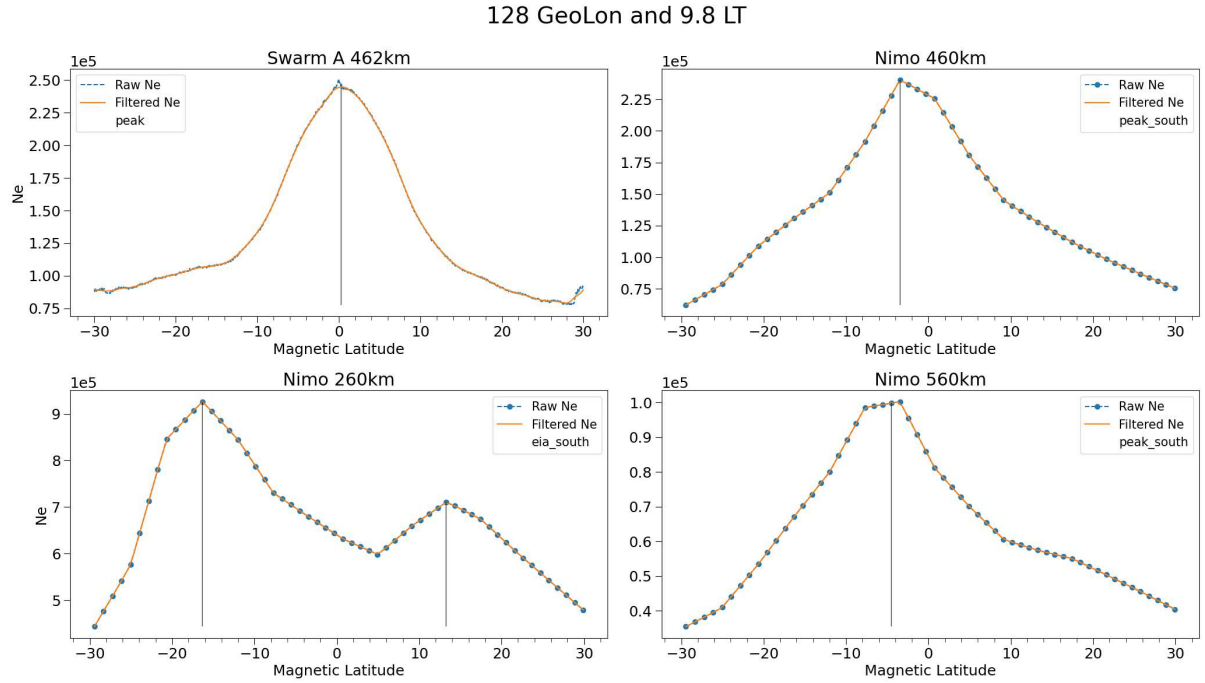
> Default is 'NIMO_AQ_%Y%j'

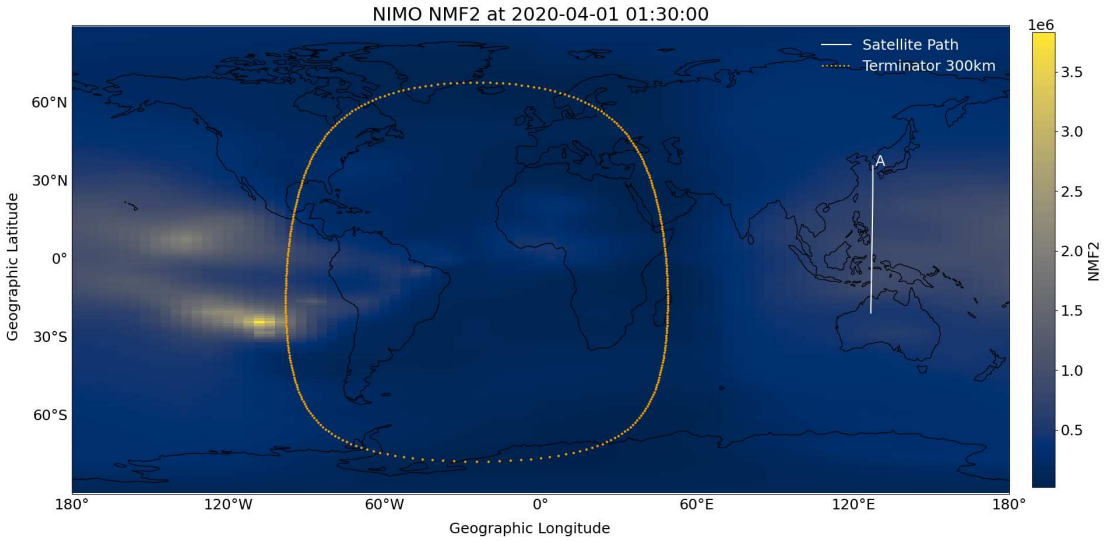
*_var : str of variable names for NIMO

> variable names to be opened in the NIMO file
> * ne, lon, lat, alt, hr, min, tec, hmf2, nmf2
> Defaults
>
> > electron density - 'dene'
> > geo longitude - 'lon'
> > geo latitude - 'lat'
> > altitude - 'alt'
> > hour - 'hour'
> > minute - 'minute'

> > > TEC - 'tec'
> > > hmf2 - 'hmf2'
> > > nmf2 - 'nmf2'
>
> nimo_cadence: int
>
> > time cadence of NIMO data in minutes
> > default is 15 minutes
>
> max_tdif : double
>
> > maximum time distance (in minutes) between a NIMO
> > and Swarm
> > conjunction allowed (default 15)

In [2]:
```
swarm_fdir = '~/swarm_data'
nimo_fdir = '~/NIMO/*'
stime = datetime(2020, 4, 1,1,17)
satellite = 'A'
nimo_swarm_single_plot(stime, satellite, swarm_fdir, nimo_fdir);
```





Creating Daily Figs and Daily Files

Function NIMO_Swarm_Map_Plotting.NIMO_SWARM_mapplot
This function creates daily files and figures for SWARM (all_satellites) and NIMO conjunctions
Returns adataframe of the information that goes into the daily file

Required Parameters

> start_day : datetime (day to create file for)
> swarm_file_dir : file directory for Swarm data
> nimo_file_dir : file directory for NIMO data

Key Word Arguments

MLat : Magnetic Latitude cutoff

30° Default

file_dir: string of output directory for file

if it is left empty ('' default), then cwd will be used

fig_dir: string of output directory for figures

if it is left empty ('' default), then cwd will be used

fig_on : boolean specifying whether or not to make the file Default True
swarm_filt : filter for swarm data

Default is 'barrel_average'

swarm_interpolate : linear interpolation parameter

the number of data points will incrase by swarm_interpolate
Default is 1 (no interpolation)

swarm_envelope : boolean

determines if an envelope is used if barrel is in filter
Default is True

swarm_barrel : double deteriming magnetic latitude radius of barrel

Default is 3°

swarm_window : double determing magnetic latitude moving average window size

Default is 2°

nimo_filt : filter for nimo data

Default '' (no filter)

nimo_interpolate : linear interpolation parameter

the number of data points will incrase by swarm_interpolate

> Default is 2 (doubles number of points)

nimo_envelope : boolean

> determines if an envelope is used if barrel is in filter
> Default is False (no envelope)

nimo_barrel : double deteriming magnetic latitude radius of barrel

> Default is 3°

nimo_window : double determing magnetic latitude moving average window size

> Default is 3°

fosi : int for plot font size

> Default 18
> Exceptions:
>
> > Super Title (fosi + 10)
> > legends (fosi - 3)

nimo_name_format : string specifying nimo filename before '.nc'

> Default is 'NIMO_AQ_%Y%j'

*_var : str of variable names for NIMO

> variable names to be opened in the NIMO file
> * ne, lon, lat, alt, hr, min, tec, hmf2, nmf2
> Defaults
>
> > electron density - 'dene'
> > geo longitude - 'lon'
> > geo latitude - 'lat'
> > altitude - 'alt'
> > hour - 'hour'
> > minute - 'minute'
> > TEC - 'tec'
> > hmf2 - 'hmf2'
> > nmf2 - 'nmf2'

nimo_cadence: int

> time cadence of NIMO data in minutes
> default is 15 minutes

max_tdif : double

> maximum time distance (in minutes) between a NIMO and Swarm
> conjunction allowed (default 15)

In [3]:
```
fig_dir='~/Plots/NIMO_SWARM'
file_dir='~/Type_Files/Daily'
swarm_fdir = '~/swarm_data'
nimo_fdir='~/Python_Code/data/NIMO/'
mlat_val = 30
stime1 = datetime(2020, 4,1,0,0)  # Starting Date
for i in range(30):  # How many days you want to make files for
    stime = stime1 + timedelta(days=i)
    print(stime)
    df_big = NIMO_SWARM_mapplot(stime,swarm_fdir, nimo_fdir, MLat=mlat
                                file_dir=file_dir, fig_on=True,
                                fig_dir=fig_dir)
```

```
2020-04-01 00:00:00
2020-04-02 00:00:00
2020-04-03 00:00:00
2020-04-04 00:00:00
2020-04-05 00:00:00
Odd Orbit longitude span > 5 degrees: Skipping Pass
2020-04-06 00:00:00
Odd Orbit longitude span > 5 degrees: Skipping Pass
2020-04-07 00:00:00
2020-04-08 00:00:00
2020-04-09 00:00:00
Odd Orbit longitude span > 5 degrees: Skipping Pass
2020-04-10 00:00:00
2020-04-11 00:00:00
2020-04-12 00:00:00
2020-04-13 00:00:00
2020-04-14 00:00:00
Odd Orbit longitude span > 5 degrees: Skipping Pass
2020-04-15 00:00:00
2020-04-16 00:00:00
2020-04-17 00:00:00
Odd Orbit longitude span > 5 degrees: Skipping Pass
2020-04-18 00:00:00
2020-04-19 00:00:00
2020-04-20 00:00:00
2020-04-21 00:00:00
2020-04-22 00:00:00
Odd Orbit longitude span > 5 degrees: Skipping Pass
2020-04-23 00:00:00
2020-04-24 00:00:00
2020-04-25 00:00:00
Odd Orbit longitude span > 5 degrees: Skipping Pass
Odd Orbit longitude span > 5 degrees: Skipping Pass
2020-04-26 00:00:00
2020-04-27 00:00:00
2020-04-28 00:00:00
2020-04-29 00:00:00
2020-04-30 00:00:00
```

Swarm vs PyIRI at NIMO Conjunctions

In [5]:
```
from datetime import datetime, timedelta
# Self Created Function
# PyIRI daily files and plotting function
from SwarmPyIRI import PyIRI_NIMO_SWARM_plot
```

PyIRI daily Files at Nimo Conjunctions

Function SwarmPyIRI.PyIRI_NIMO_SWARM_plot

This function creates daily plots and a daily file

based on the NIMO-Swarm conjunctions found in the
NIMO Swarm daily files

Required parameters

> sday: datetime
>
> > (day starting at 0,0)
>
> daily_dir : str
>
> > directory of daily files made by
> > NIMO_Swarm_Map_Plotting.NIMO_SWARM_mapplot
>
> swarm_dir : str
>
> > Swarm data directory to which data will be
> > downloaded into an
> > appropriate date/satellite directory structure

Key Word Arguments

> file_save_dir : str kwarg
>
> > directory where file should be saved, default cwd
>
> fig_on : kwarg bool
>
> > set to true, plot will be made, if false, plot will not be
> > made
>
> fig_save_dir : str kwarg
>
> > directory where figure should be saved, default cwd
>
> pyiri_filt : str kwarg
>
> > Desired Filter for nimo data (no filter default)
>
> pyiri_interpolate : int kwarg
>
> > int that determines the number of data points in
> > interpolation
> > new length will be len(density)xinterpolate
> > default is 2

> pyiri_envelope : bool kwarg
>
> > if True, barrel roll will include points inside an envelope, if false (default), no envelope will be used
>
> pyiri_barrel : double
>
> > latitudinal radius of barrel for swarm (default: 3 degrees maglat)
>
> pyiri_window : double kwarg
>
> > latitudinal width of moving window (default: 3 degrees maglat)
>
> fosi : int kwarg
>
> > fontsize for plot (default is 18)
> > Exceptions:
> >
> > > Super Title (fosi + 10)
> > > legends (fosi - 3)

The returns include daily files, figures (if fig_on),
and a dataframe with what is contained in the daily files

In [8]:
```
daily_files = '~/Type_Files/Daily'
pyiri_fig_dir = '~/Plots/NIMO_SWARM/'
pyiri_file_dir = '~/Type_Files/Daily/'

stime1 = datetime(2020, 4,1,0,0) # Starting Date
for i in range(30):  # How many days you want to make files for
    stime = stime1 + timedelta(days=i)
    print(stime)
    iridf,dailydf = PyIRI_NIMO_SWARM_plot(stime, daily_files,swarm_fdi
                          file_save_dir=pyiri_file_dir)
```

```
2020-04-01 00:00:00
2020-04-02 00:00:00
2020-04-03 00:00:00
2020-04-04 00:00:00
2020-04-05 00:00:00
2020-04-06 00:00:00
2020-04-07 00:00:00
2020-04-08 00:00:00
2020-04-09 00:00:00
2020-04-10 00:00:00
2020-04-11 00:00:00
2020-04-12 00:00:00
2020-04-13 00:00:00
2020-04-14 00:00:00
2020-04-15 00:00:00
2020-04-16 00:00:00
2020-04-17 00:00:00
2020-04-18 00:00:00
2020-04-19 00:00:00
2020-04-20 00:00:00
2020-04-21 00:00:00
2020-04-22 00:00:00
2020-04-23 00:00:00
2020-04-24 00:00:00
2020-04-25 00:00:00
2020-04-26 00:00:00
2020-04-27 00:00:00
2020-04-28 00:00:00
2020-04-29 00:00:00
2020-04-30 00:00:00
```

What still needs to be done?

Create separate PyIRI plots with maps

In [ ]:

Statistical Plots

Created by Alanah Cardenas-O'Toole
Summer 2025
Latest update: 08/07/2025
Email alanahco@umich.edu
Walkthrough of how to use functions from Swarm_Stats.py

In [1]:
```
import numpy as np
import pandas as pd
from datetime import datetime,timedelta


# Statistical codes
from Swarm_Stats import states_report_swarm, LSS_plot_Swarm, map_hist_
from Swarm_Stats import plot_hist_quad_maps, Liemohn_Skill_Scores
from Swarm_Stats import decision_table_sat, style_df_table, HMFC_perce
from Swarm_Stats import style_LSS_table, LSS_table_sat, one_model_LSS_
```

Getting dataframes that include the basic state and H, M, F, C

Swarm_stats.states_report_swarm This code requires that both NIMO and PyIRI daily files have been created
and returns 3 dataframes
that will be used for future statistics
Note: if you just want H, M, F ,C for one model,
state_check(obs_type, mod_type, state='eia') is useful

Requred Parameters

> date_range : pandas daterange
>
> > Date range of desired states files
>
> daily_dir : str
>
> > directory of daily files

Key Word Arguments

> typ: str
>
> > desired type to check against

> for state orientations
> 'eia'(default), 'peak', 'flat', 'trough'
> for direction orientations
> 'north', 'south', 'neither'

NIMO_alt: str

> specifies which altitude to use
> 'swarm'(default),'hmf2','100'

Returns

> NiSw : DataFrame
>
> > NIMO states, directions, and types (original full name)
> > also includes longitude, local times, and sat list
>
> Sw : DataFrame
>
> > Swarm States, direction, and types
> > also includes longitude, local times, and sat list
>
> Py : DataFrame
>
> > PyIRI states, directions, and types
> > also includes longitude, local times, and sat list

In [2]:
```
date_range = pd.date_range(start='2020-04-01',  end='2020-04-30')
daily_files = '~/Type_Files/Daily'
NiSw, Sw, PyI = states_report_swarm(date_range, daily_files, typ='eia'
print(NiSw) # Nimo Swarm comparison
```

```
     state direction                    type   GLon         LT Sat skill
0     peak     north              peak_north  -40.0  21.950833   A     M
1     peak     south              peak_south  128.0   9.901974   A     C
2     peak     north              peak_north  -64.0  21.903889   A     C
3     peak   neither                    peak  104.0   9.897939   A     M
4      eia     south  eia_saddle_peak_south  -88.0  21.890833   A     H
...    ...       ...                    ...    ...        ...  ..   ...
2735   eia     north  eia_saddle_peak_north  160.0   7.292343   C     F
2736  peak   neither                    peak  -32.0  19.345556   C     M
2737   eia     north  eia_saddle_peak_north  136.0   7.284213   C     F
2738  peak     north              peak_north  -56.0  19.299444   C     M
2739  peak   neither                    peak  112.0   7.281541   C     C

[2740 rows x 7 columns]
```

Creating Liemohn Skill Score plots

Swarm_Stats.LSS_plot_Swarm
Created using Liemohn Skill Scores 1-4 from
"Leaving Heidke behind: Defining an independent reference model
for event detection skill scores" Liemohn et al. (in preparation 2025)
This requires 2 models for compasion becuase LSS is valuable as a comparison tool.
If you only want 1, then use Swarm_Stats.one_model_LSS_plot_Swarm
NOTE: LSS can range outside of +/-1

Plot LSS vs CSI or PC 4 panels (one for each LSS) Required Parameters

> model1 : dataframe
>
> > first model dataframe built by states_report_swarm
>
> model2 : dataframe
>
> > second model dataframe built by
> > states_report_swarm
>
> eia_type : str
>
> > desired eia type for fig title
>
> date_range : datetime range
>
> > For plotting title purposes

Key Word Arguments

> model1_name : str kwarg
>
> > first model name for labelling purposes
>
> model2_name : str kwarg
>
> > second model name for labelling purposes
>
> PorC : str kwarg
>
> > Percent correct or Critical success index for x axes
>
> DayNight : bool kwarg

> > True (default) if panels should have separate markers
> > for day and night
>
> otherwise (false) all are plotted together

LT_range : list kwarg

> Range of day night local time, Default is 7 LT to 19 LT for day and
> 19 LT to 7 LT for Night

coin : bool kwarg

> If True, coin LSS will be plotted for comparison (default)
> if false, coin LSS will not be plotted

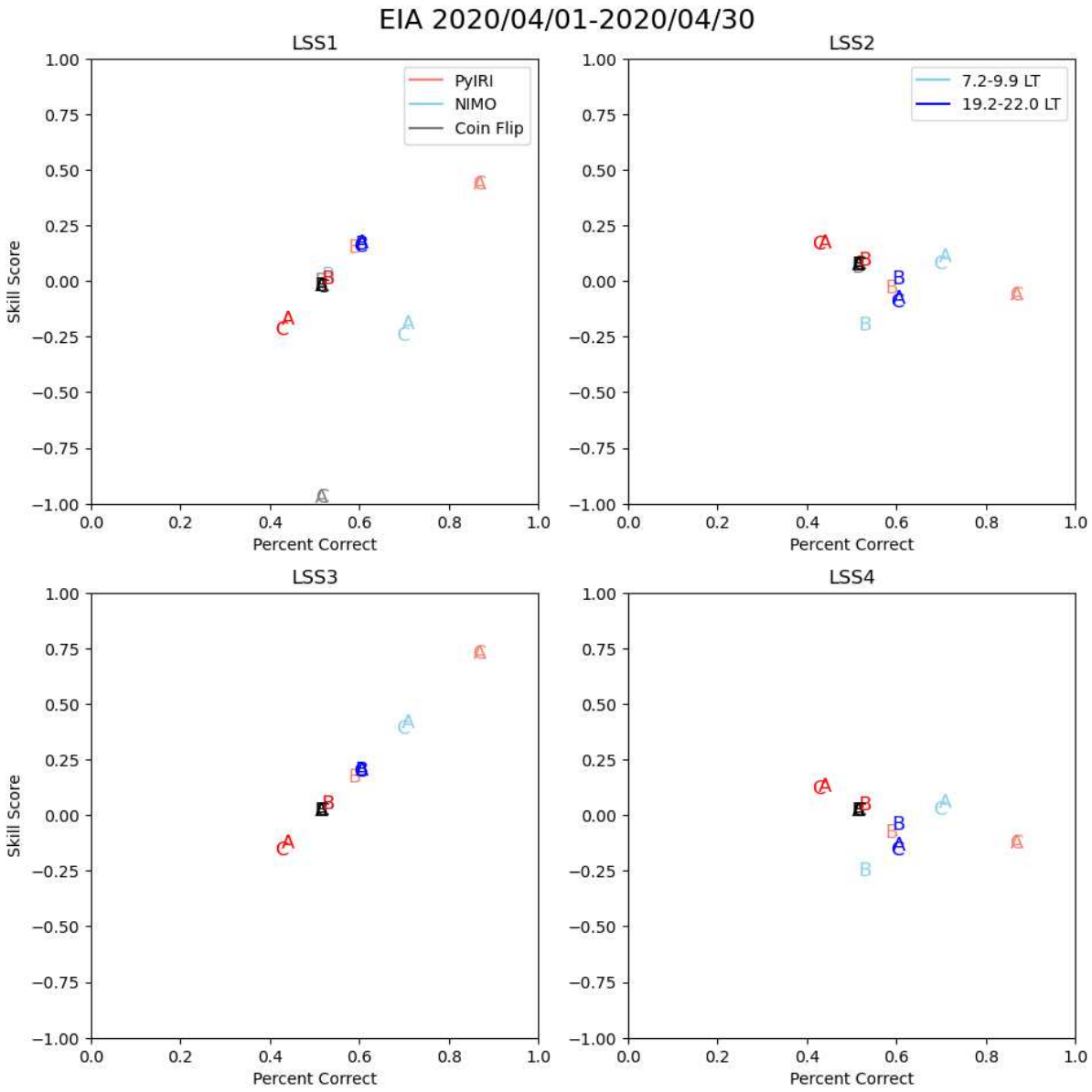
Returns

> fig : figure handle
>
> > 4 panel figure that includes LSS for the 2 models
> > and a coin toss if coin

Note: Since we care about Correct Negatives,
Percent Correct is more useful than Critical Success Index

> PC = (H + C)/T
> CSI = H/(H + M + F)
>
> > according to Liemohn et al. pg 8

In [3]:
```
fig = LSS_plot_Swarm(NiSw, PyI, 'EIA', date_range, model1_name='NIMO',
```



Creating Liemohn Skill Score plots continued

Swarm_Stats.one_model_LSS_plot_Swarm
Created using Liemohn Skill Scores 1-4 from
"Leaving Heidke behind: Defining an independent reference model
for event detection skill scores" Liemohn et al. (in preparation 2025)
If you want to compare 2 models, then use Swarm_Stats.LSS_plot_Swarm

Plot LSS vs CSI or PC 4 panels (one for each LSS) Required Parameters

> model1 : dataframe
>
> > first model dataframe built by states_report_swarm
>
> eia_type : str

> > desired eia type for fig title
>
> date_range : datetime range
>
> > For plotting title purposes

Key Word Arguments

> model_name : str kwarg
>
> > first model name for labelling purposes
>
> PorC : str kwarg
>
> > Percent correct or Critical success index for x axes
>
> DayNight : bool kwarg
>
> > True (default) if panels should have separate markers
> > for day and night
>
> otherwise (false) all are plotted together

LT_range : list kwarg

> Range of day night local time, Default is 7 LT to 19 LT for day and
> 19 LT to 7 LT for Night

coin : bool kwarg

> If True, coin LSS will be plotted for comparison (default)
> if false, coin LSS will not be plotted

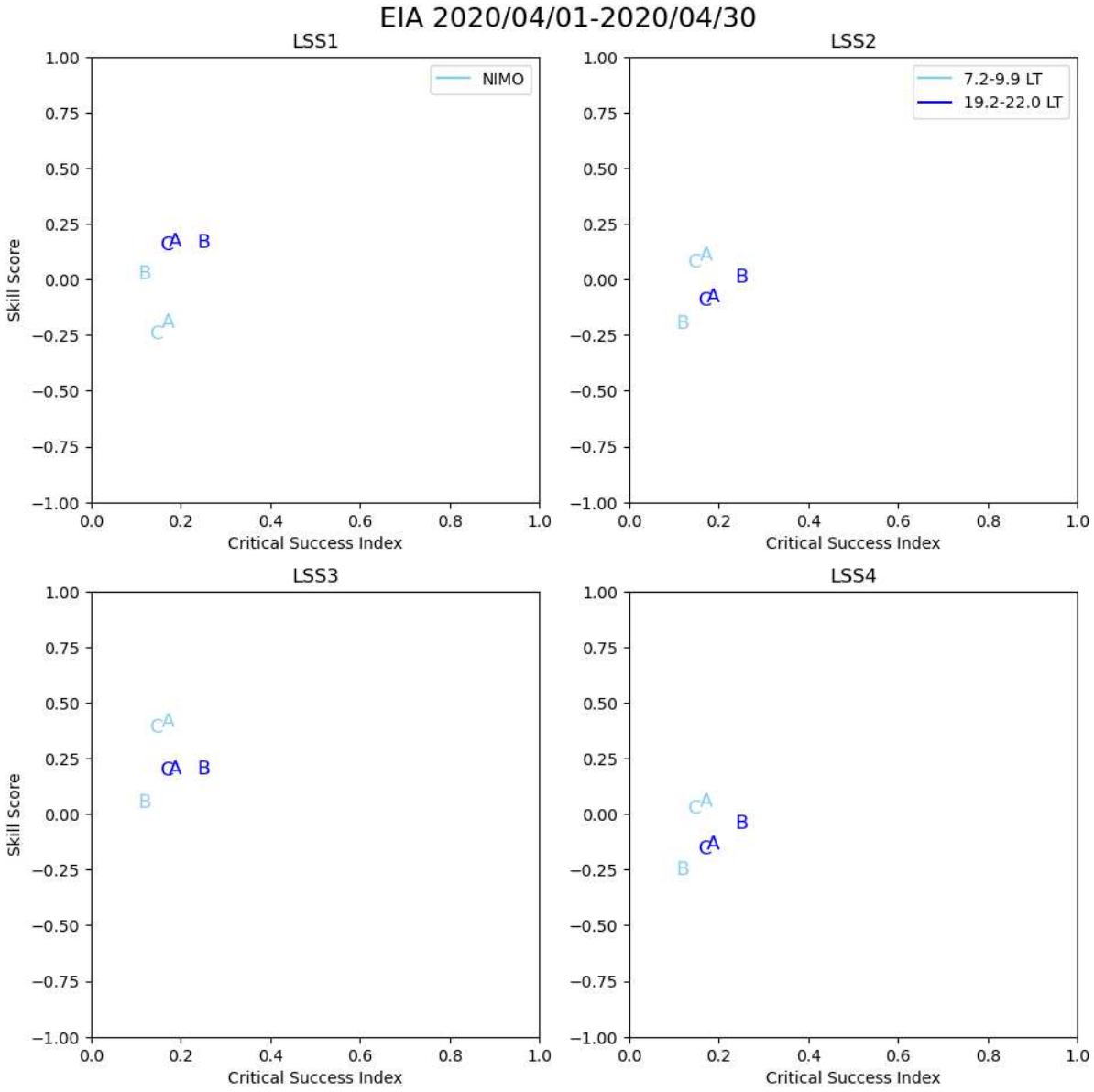
Returns

> fig : figure handle
>
> > 4 panel figure that includes LSS for the 2 models
> > and a coin toss if coin

Note: Warning will be printed if coin is specified as False

In [4]:
```
fig = one_model_LSS_plot_Swarm(NiSw, 'EIA', date_range, model_name='NI
                               LT_range=[7, 19], coin=False)
```

```
/Users/aotoole/Documents/Python_Code/EIA_Update/Swarm_Stats.py:575: Use
rWarning: Warning: Coin is False! LSS is a comparison tool!
  warnings.warn("Warning: Coin is False! LSS is a comparison tool!")
```



Plotting Histogram Maps

Function Swarm_Stats.plot_hist_quad_maps
plot histogram maps on a 4 panel figure for each score: Hit, Miss,
False positive, and Correct Negative
This function calls
Swarm_Stats.map_hist_panel(ax, model, bin_lons=37, DayNight=True, LT_range=[7, 19])
Which will make just 1 panel

Required Parameters

> model_states : dataframe
>
> > dataframe of model data including skill and local
> > times
> > built by states_report_swarm
>
> sat : str
>
> > swarm satellite 'A', 'B', or 'C'
>
> eia_type : str
>
> > eia state e.g. EIA, Peak, etc.
> > depending on what is considered a hit
>
> date_range : pandas daterange
>
> > range of dates for title purposes

Key Word Arguments

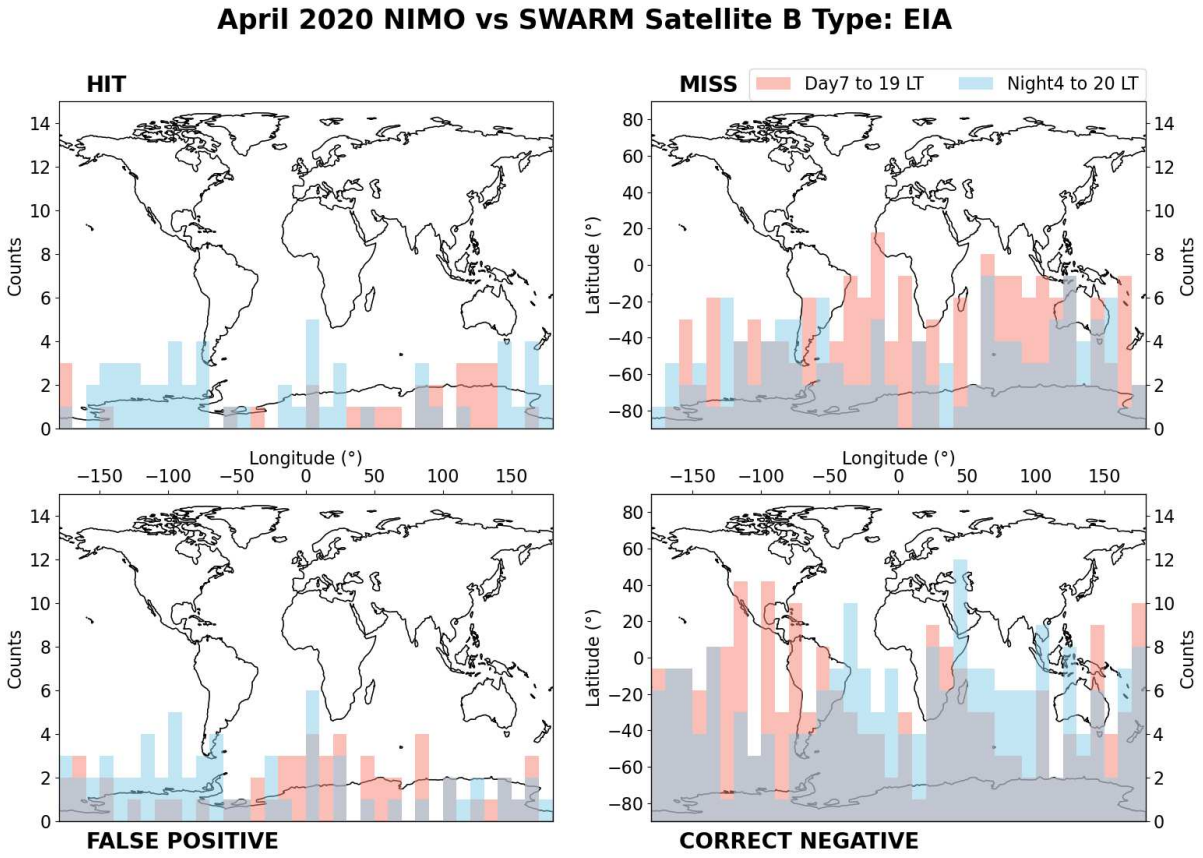
> bin_lons : int kwarg
>
> > number of bins between -180 and 180 deg geo lon
> > np.linspace(-180, 180, bin_lons)
> > default 37
>
> model_name : str kwarg
>
> > name of model for title purposes
> > default 'Model'
>
> fosi : int kwarg
>
> > font size for plot
> > default 16
>
> hist_ylim : list kwarg
>
> > y range (counts) for hist plot
> > default [0,15]
>
> LT_range : list kwarg
>
> > Range of day night local time
> > Default is 7 LT to 19 LT for day and
> > 19 LT to 7 LT for Night

Returns

> fig : figure handle
>
> > fig with 4 panels of hist maps

Note: A side thought is to have only 2 panels:
one with HIT and total in state (H + M)
and another with Correct Negatives and total out of state(C + F)

In [5]:
```
fig = plot_hist_quad_maps(NiSw, 'B', 'eia', date_range, bin_lons=37, m
```



Making Decision Tables

Swarm_Stats.decision_table_sat

Takes in dataframe created by Swarm_Stats.states_report_swarm
Neat decision table summing up the hits, misses,
correct negatives, and false positives per satellite

Required Parameters

> states: dataframe

> > dataframe of model data including skill and local
> > times
> > built by states_report_swarm
>
> eia_type : str
>
> > eia state e.g. EIA, Peak, etc. depending on what is
> > considered a hit

Key Word Arguments

> sats : list of strings kwarg
>
> > swarm satellites 'A', 'B', and 'C' as default
> > can specify just 1 or 2
>
> model_name : str kwarg
>
> > Model name for decision table label
> > default 'Model'

Returns

> df : dataframe
>
> > dataframe in table format separated by satellite
> > and event state (state, non-state)
> > index using
> > df.loc[(f'Swarm {satellite}', eia_type), (model_name,
> > eia_type)]

Swarm_Stats.style_df_table

This function styles the table created by Swarm_Stats.decision_table_sat
This will only be for all satellites because I spent too much time
Trying to figure out how to make it more general.
The issue is from 941 where I specify the colors

Required Parameters:

> df_table : dataframe
>
> > dataframe created by decision_table_sat
>
> eia_type : str

> > string designating which eia type is being reported

Returns

> Styled dataframe with colors indicating successes and failures and table spearators by satelltie

In [6]:
```
df_table = decision_table_sat(NiSw)
df_table
```

Out[6]:

		Model	
		eia	Non-eia
Swarm A	**eia**	65.0	184.0
	Non-eia	142.0	522.0
Swarm B	**eia**	83.0	287.0
	Non-eia	118.0	416.0
Swarm C	**eia**	56.0	191.0
	Non-eia	146.0	530.0

In [7]:
```
styled_table = style_df_table(df_table, 'eia')
styled_table
```

Out[7]:

		Model	
		eia	Non-eia
Swarm A	**eia**	65	184
	Non-eia	142	522
Swarm B	**eia**	83	287
	Non-eia	118	416
Swarm C	**eia**	56	191
	Non-eia	146	530

Making Liemohn Skill Score Tables

Swarm_Stats.LSS_table_sat

Neat table including the Liemohn Skill Scores 1-4 separated by satellite

Required Parameters

model1: dataframe

> dataframe of 1st model data including skill and local times built by states_report_swarm

model2 : dataframe

> dataframe of 2nd model data including skill and local times built by states_report_swarm

Key Word Arugments

> model1_name : str kwarg
>
> > string of name of model1
>
> model2_name : str kwarg
>
> > string of name for model2
>
> sats : list of strings kwarg
>
> > swarm satellites 'A', 'B', and 'C' as default can specify just 1 or 2

Returns

> LSS_df : dataframe
>
> > dataframe in table format separated by satellite and Liemohn skill score

Swarm_Stats.style_LSS_table

This function styles LSS_df by adding lines in between each satellite
All satellites are not required for this one

Required Parameters

> LSS_df : dataframe
>
> > dataframe created by LSS_table_sat

Key word Arguments

> sat_list: list of strings kwarg
>
> > satellite list for LSS_df

Returns

> LSS table with dividers between satellites
> This can be further edited in pyhton and
> by copying and pasting it to a document

In [8]:
```
LSS_df = LSS_table_sat(NiSw, PyI, model1_name='NIMO', model2_name='PyI
LSS_df
```

Out[8]:

		NIMO	PyIRI
Swarm A	**LSS1**	0.099900	0.091616
	LSS2	0.009910	0.179228
	LSS3	0.285871	0.279299
	LSS4	-0.061149	0.120321
Swarm B	**LSS1**	0.073489	0.057475
	LSS2	-0.117436	0.014682
	LSS3	0.103982	0.088496
	LSS4	-0.169597	-0.031311
Swarm C	**LSS1**	0.068553	0.063026
	LSS2	-0.014166	0.173335
	LSS3	0.269772	0.265439
	LSS4	-0.086980	0.113983

In [9]:
```
styled_df = style_LSS_table(LSS_df)
styled_df
```

Out[9]:

		NIMO	PyIRI
Swarm A	**LSS1**	0.099900	0.091616
	LSS2	0.009910	0.179228
	LSS3	0.285871	0.279299
	LSS4	-0.061149	0.120321
Swarm B	**LSS1**	0.073489	0.057475
	LSS2	-0.117436	0.014682
	LSS3	0.103982	0.088496
	LSS4	-0.169597	-0.031311
Swarm C	**LSS1**	0.068553	0.063026
	LSS2	-0.014166	0.173335
	LSS3	0.269772	0.265439
	LSS4	-0.086980	0.113983

Plotting HM percents and FC percents

Plot full figure using HMFC_percent_panel
2 Models required e.g. Py IRI and NIMO

> This figure has a lot going on. When you look at it, think of each quadrant as a separate plot defined by Hit, Miss, Correct Negative, and False Positive as labelled. The percentages are the percent the model got correct or incorrect based on event states
> For example, for Hits, ther percentage is Hit/(Hit + Miss) where Hit+Miss
> is the total in the event states, the panel below that Miss/(Hit+Miss) is
> equivalent to 100% - Hit/(Hit + Miss), so those sectors are conjugate to
> each other
> For quick viewing, there are 4 shaded regions. These represent when a
> model is doing better than a coin toss. Ideally, False positives and Misses
> would have a low % and Hits and Correct Negatives have a higher

> percentage

Required Parameters

> model1 : dataframe
>
> > first model dataframe built by states_report_swarm
>
> model2 : dataframe
>
> > second model dataframe built by
> > states_report_swarm
>
> eia_type : str
>
> > desired eia type for fig title

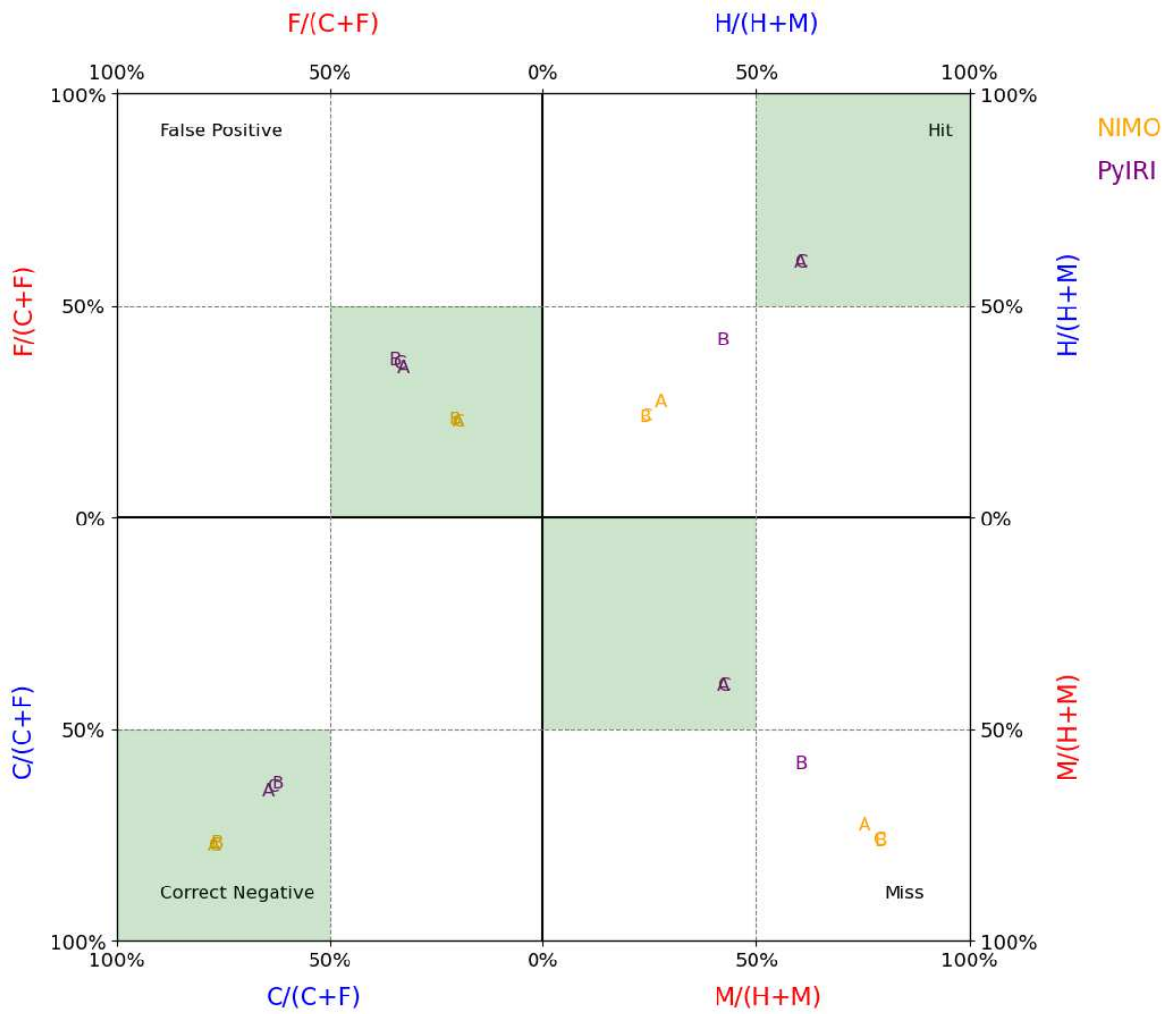
Key Word Arguments

> model1_name : str kwarg
>
> > first model name for labelling purposes
> > default Model1
>
> model2_name : str kwarg
>
> > second model name for labelling purposes
> > default Model2
>
> col1 : str
>
> > plotting color for Model1
> > defualt orange
>
> col2 : str
>
> > plotting color for Model 2
> > default purple
>
> fosi : int
>
> > font size for plot

Returns

> fig : figure handle as desribed above

In [10]:
```
fig = HMFC_percent_figure(NiSw, PyI, 'eia', model1_name='NIMO', model2
```



In [ ]:
```

```

MADRIGAL

In [1]:
```
from datetime import datetime, timedelta
# Self Created Functions -------------------------
# Plot NIMO and Madrigal TEC together
from Madrigal_NIMO2 import NIMO_MAD_DailyFile, load_madrigal, mad_nimo
from Load_NIMO2 import load_nimo
import pysat
import pysatMadrigal as pymad
```

Download Madrigal data that coincides with model data.

Madrigal NIMO single Plot

Function: Madrgial_NIMO2.NIMO_MAD_DailyFile
Plot up to 12 panels including Madrigal TEC with
standard devation as error bars,filtered Madrigal TEC,
and NIMO TEC along with the corresponding EIA Types

Required Parameters

> mad_dc : dictionary of madrigal data from load_madrigal function
> nimo_dc : dictionary of nimo data from load_nimo function

Key Word arguments

> lon_start : starting longitude for plot. i.e. -90
>
> > Plot will range between -90 to -60 as a Default
>
> stime : datetime for plot
> mlat_val : int magnetic latitude cutoff
> max_nan : double of Maximum acceptable percent nan values in a
> pass fosi : int font size

load_madrigal

Function: Madrigal_NIMO2.load_madrigal
Loads madrgial data into a dictionary

Required Parameters

> stime: datetime Universal time for the desired madrigal output

> fdir : str directory where file is located

Returns

> mad_dc : dictionary object
>
> > dictionary of the madrigal data including:
> > tec, geographic latitude, geographic longitude,
> > dtec, timestamp, date (datetime format),
> > magnetic latitude, magnetic longitude

Notes

> This takes in madrgial files of format gps%y%m%dg.002.netCDF4
> 5 minute cadence

load_nimo

Function : Load_NIMO2.load_nimo
Loads Nimo file inot a dictionary

Required Parameters:

> stime : datetime of desired Nimo data

Key Word Arguments

> fdir : directory of NIMO file
> name_format : string
>
> > format of NIMO filename including date format before
> > .nc
> > Default: 'NIMO_AQ_%Y%j'
>
> *_var : str of variable names for NIMO*
>
> > *variable names to be opened in the NIMO file*
>
> ne, lon, lat, alt, hr, min, tec, hmf2, nmf2
> Defaults
>
> > electron density - 'dene'
> > geo longitude - 'lon'
> > geo latitude - 'lat'
> > altitude - 'alt'

> > hour - 'hour'
> > minute - 'minute'
> > TEC - 'tec'
> > hmf2 - 'hmf2'
> > nmf2 - 'nmf2'

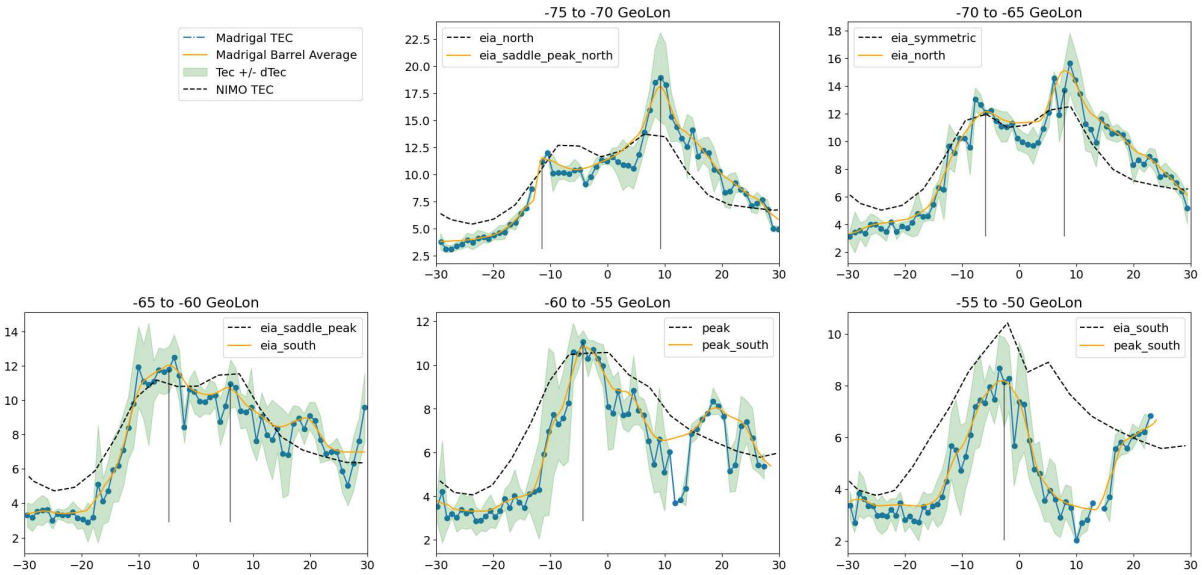
nimo_cadence: int

> time cadence of NIMO data in minutes
> default is 15 minutes

Retunrs

> nimo_dc : dictionary
>
> > dictionary with variables:
> > dene,lon,lat,alt,hour,minute,date, tec,hmf2

In [2]:
```
stime = datetime(2020, 4,1,0,30)
load_file = '~/data/gnss/tec/vtec/'
nim_dir = '~/NIMO/'
mad_dc = load_madrigal(stime, load_file)
nimo_dc = load_nimo(stime, fdir=nim_dir)
lon_start = -90
mlat_val = 30
mad_nimo_single_plot(mad_dc, nimo_dc, lon_start, stime,
                     mlat_val);

# TO make a map use
#-------------------
# where nimo_map comes from:
#nimo_df, nimo_map = nimo_mad_conjunction(nimo_dc, mlat_val,
#                                         glon_val, stime)
# madrigal_nimo_world_maps(stime, mad_dc, nimo_map)
```



Madrigal NIMO Daily Files and plots

Function: Madrgial_NIMO2.NIMO_MAD_DailyFile
Daily Files include information about the conjunctions, peak locations, and EIA type
Plot up to 12 panels including Madrigal TEC with
standard devation as error bars,filtered Madrigal TEC,
and NIMO TEC along with the corresponding EIA Types
Includes a separate map plot for the TEC

Required Parameters

> start_day : datetime for daily file
> mad_file_dir : string file directory of Madrigal File
> nimo_file_dir : string file directory of NIMO File

Key Word Arguments

> MLat : Magnetic Latitude cutoff
>
> > 30° Default
>
> lon_start : starting longitude for plot. i.e. -90
>
> > Plot will range between -90 to -30 as a Default
> > Another Recommended Region is 60 to 120
>
> file_save_dir: string of output directory for file

> > if it is left empty ('' default), then cwd will be used

fig_on : boolean specifying whether or not to make the file Default True
fig_save_dir: string of output directory for figures

> > if it is left empty ('' default), then cwd will be used

max_nan : double specifying the maximum %nan is acceptable in a pass
mad_filt : str Desired Filter for madrigal data (default barrel_average)
mad_interpolate : int

> > int that determines the number of data points in interpolation
> > new length will be len(density)xinterpolate
> > default is 2 indicating double number of points

mad_envelope : bool

> > if True, barrel roll will include points inside an br envelope, if False (default), no envelope will be used

mad_barrel : double latitudinal radius of barrel for madrigal (default: 3 degrees maglat)
mad_window : double latitudinal width of moving window (default: 3 degrees maglat)
nimo_filt : filter for nimo data

> > Default '' (no filter)

nimo_interpolate : linear interpolation parameter

> > the number of data points will incrase by swarm_interpolate
> > Default is 2 (doubles number of points)

nimo_envelope : boolean

> > determines if an envelope is used if barrel is in filter
> > Default is False (no envelope)

nimo_barrel : double deteriming magnetic latitude radius of barrel

> > Default is 3°

nimo_window : double determing magnetic latitude moving average window size

> Default is 3°

fosi : int for plot font size

> Default 18
> Exceptions:
>
> > Super Title (fosi + 10)
> > legends (fosi - 3)

nimo_name_format : string specifying nimo filename before '.nc'

> Default is 'NIMO_AQ_%Y%j'

*_var : str of variable names for NIMO

> variable names to be opened in the NIMO file
> * ne, lon, lat, alt, hr, min, tec, hmf2, nmf2
> Defaults
>
> > electron density - 'dene'
> > geo longitude - 'lon'
> > geo latitude - 'lat'
> > altitude - 'alt'
> > hour - 'hour'
> > minute - 'minute'
> > TEC - 'tec'
> > hmf2 - 'hmf2'
> > nmf2 - 'nmf2'

nimo_cadence: int

> time cadence of NIMO data in minutes
> default is 15 minutes

max_tdif : double

> maximum time distance (in minutes) between a NIMO and Swarm
> conjunction allowed (default 15)

In [3]:
```
stime = datetime(2020, 4,1,0,0)
fig_save = '~/Plots/NIMO_MADRIGAL/'
file_save = '~/Type_Files/Daily/'
mad_load_file = '~/data/gnss/tec/vtec/'
nim_dir = '~/NIMO/'
mad_df = NIMO_MAD_DailyFile(stime, mad_load_file, nim_dir,
                            mlat_val=30, lon_start=-90,
                            file_save_dir=file_save,
                            fig_on=True, fig_save_dir=fig_save)
```

In [8]:
```
stime1 = datetime(2014, 1,1,0,0) # Starting Date
for i in range(31):  # How many days you want to make files for
    stime = stime1 + timedelta(days=i)
    print(stime)
    mad_df = NIMO_MAD_DailyFile(stime, mad_load_file, nim_dir,
                            mlat_val=30, lon_start=60,
                            file_save_dir=file_save,
                            fig_on=True, fig_save_dir=fig_save)
```

```
2014-01-01 00:00:00
2014-01-02 00:00:00
2014-01-03 00:00:00
2014-01-04 00:00:00
2014-01-05 00:00:00
2014-01-06 00:00:00
2014-01-07 00:00:00
2014-01-08 00:00:00
2014-01-09 00:00:00
2014-01-10 00:00:00
2014-01-11 00:00:00
2014-01-12 00:00:00
2014-01-13 00:00:00
2014-01-14 00:00:00
2014-01-15 00:00:00
2014-01-16 00:00:00
2014-01-17 00:00:00
2014-01-18 00:00:00
2014-01-19 00:00:00
2014-01-20 00:00:00
2014-01-21 00:00:00
2014-01-22 00:00:00
2014-01-23 00:00:00
2014-01-24 00:00:00
2014-01-25 00:00:00
2014-01-26 00:00:00
2014-01-27 00:00:00
2014-01-28 00:00:00
2014-01-29 00:00:00
2014-01-30 00:00:00
2014-01-31 00:00:00
```

In [ ]:

Madrigal Stats

Mad_Stats.py is the start of statistically analyzing Madrigal TEC EIA results
It works very similar to Swarm_Stats.py, but it is less comprehensive right now

In [4]:
```
import pandas as pd
from Mad_Stats import states_report_mad,  Mad_LSS_plot
```

Mad_Stats.states_report_mad

Report States for date range for Swarm comparison,
need to make one for Madrigal comparison

Required Parameters

> date_range : pandas daterange
>
> > Date range of desired states files
>
> daily_dir : str
>
> > directory of daily files

Key Word Arguemnts

> typ: str
>
> > desired type to check against
> > for orientation of 'state'
> > 'eia'(default), 'peak', 'flat', 'trough'
> > for orientation of 'direction'
> > 'north', 'south', 'neither'
>
> mad_lon : int
>
> > starting longitude for Madrigal Daily Files

Returns

Ni : DataFrame

> NIMO states, directions, and types
> also includes longitude and local times

Mad : DataFrame

> Madrigal States, direction, and types
> also includes longitude and local times

In [9]:
```
date_range = pd.date_range(start='2014-01-01',  end='2014-01-31')

date_array = date_range.to_pydatetime()
sday = date_array[0]
daily_files = '~/Type_Files/Daily'

NiMad, Mad = states_report_mad(date_range, daily_files, typ='eia', mad
NiMad
```

Out[9]:

	state	direction	type	GLon	LT	skill
0	peak	neither	peak	-76.0	18.666667	M
1	peak	neither	peak	-72.0	19.000000	C
2	peak	neither	peak	-68.0	19.333333	C
3	peak	south	peak_south	-64.0	19.666667	M
4	eia	south	eia_saddle_peak_south	-56.0	20.000000	F
...	...	...	...	...	...	...
16760	eia	neither	eia_saddle_peak	-76.0	18.416667	F
16761	eia	south	eia_saddle_peak_south	-72.0	18.750000	F
16762	eia	south	eia_saddle_peak_south	-68.0	19.083333	H
16763	eia	south	eia_saddle_peak_south	-64.0	19.416667	F
16764	eia	south	eia_south	-52.0	20.083333	H

16765 rows × 6 columns

Mad_Stats.Mad_LSS_plot

Plot LSS vs CSI or PC 4 panels (one for each LSS) for 1 model alone
NOTE: LSS is only useful in comparison to another model; therefore,
coin set to True is highly recommended!

Required Parameters

> model1 : dataframe

> > model dataframe built by states_report_swarm
>
> eia_type : str
>
> > desired eia type for fig title
>
> date_range : datetime range
>
> > For plotting title purposes

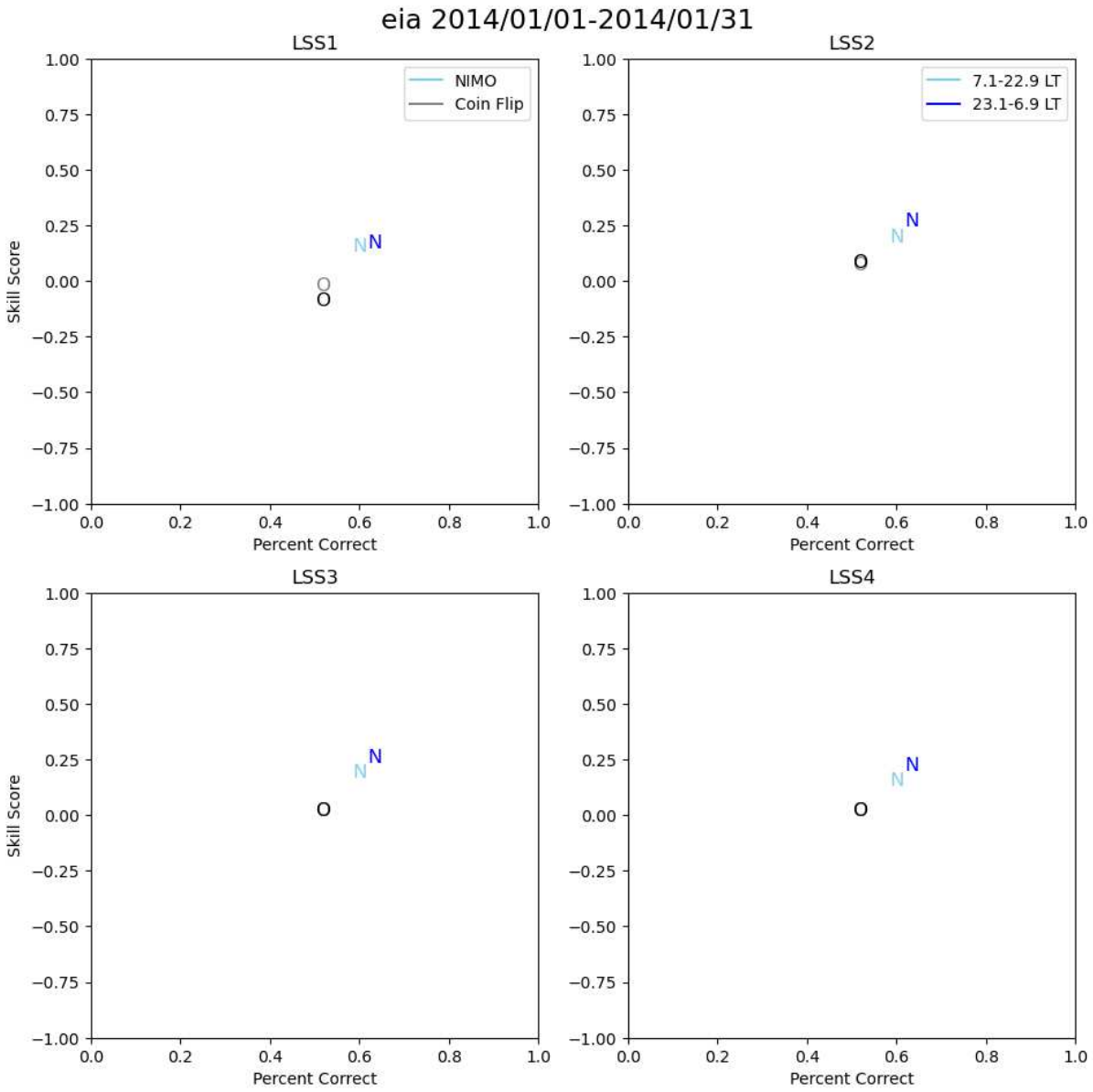
Key Word Arguments

> model_name : str kwarg
>
> > first model name for labelling purposes
>
> PorC : str kwarg
>
> > Percent correct or Critical success index for x axes
>
> DayNight : bool kwarg
>
> > True (default) if panels should have separate markers
> > for day and night
> > otherwise (false) all are plotted together
>
> LT_range : list kwarg
>
> > Range of day night local time, Default is 7 LT to 19 LT
> > for day and
> > 19 LT to 7 LT for Night
>
> coin : bool kwarg
>
> > If True, coin LSS will be plotted for comparison
> > (default)
> > if false, coin LSS will not be plotted

Returns

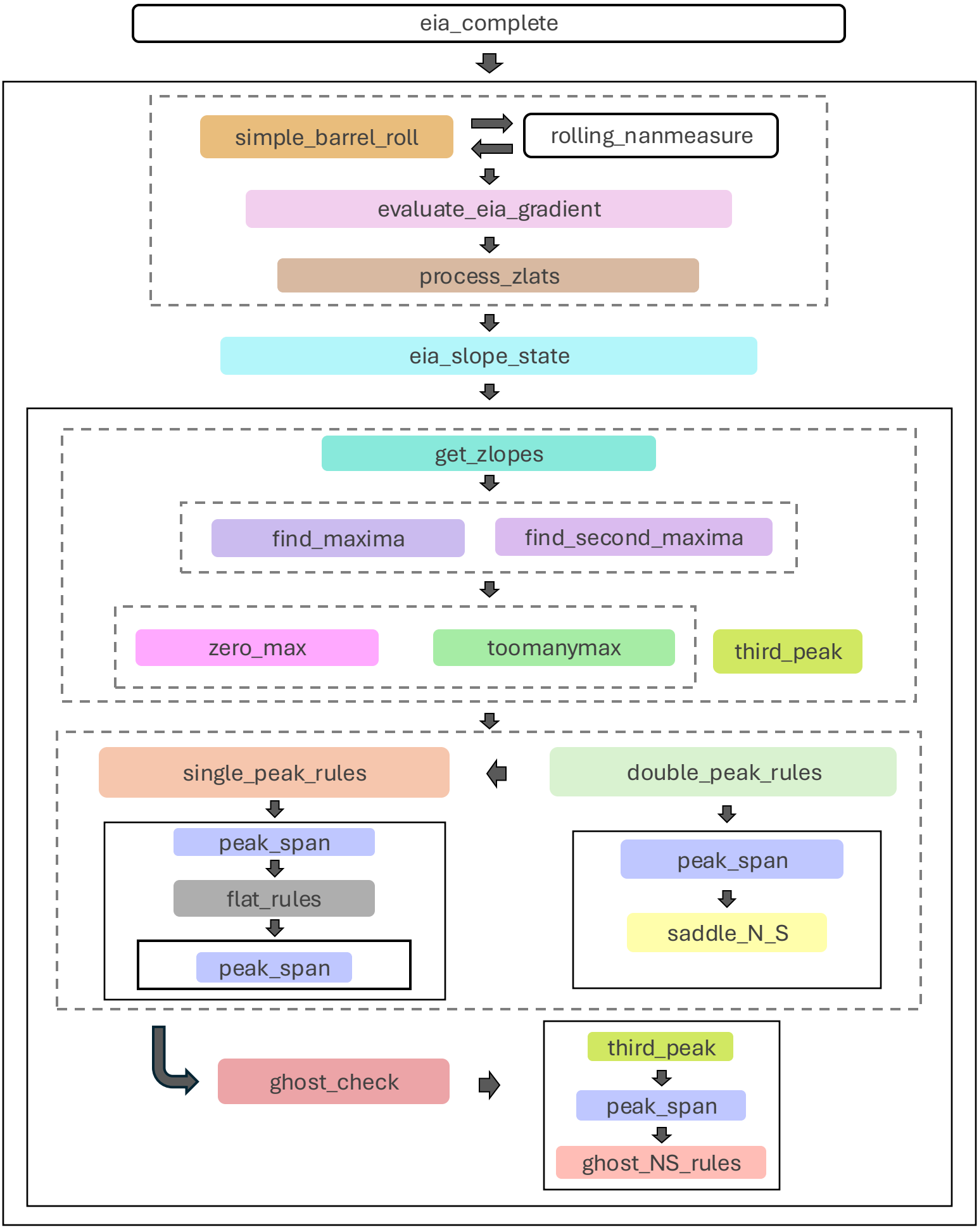
> fig : fig handle
>
> > 4 panel figure (one for each LSS)

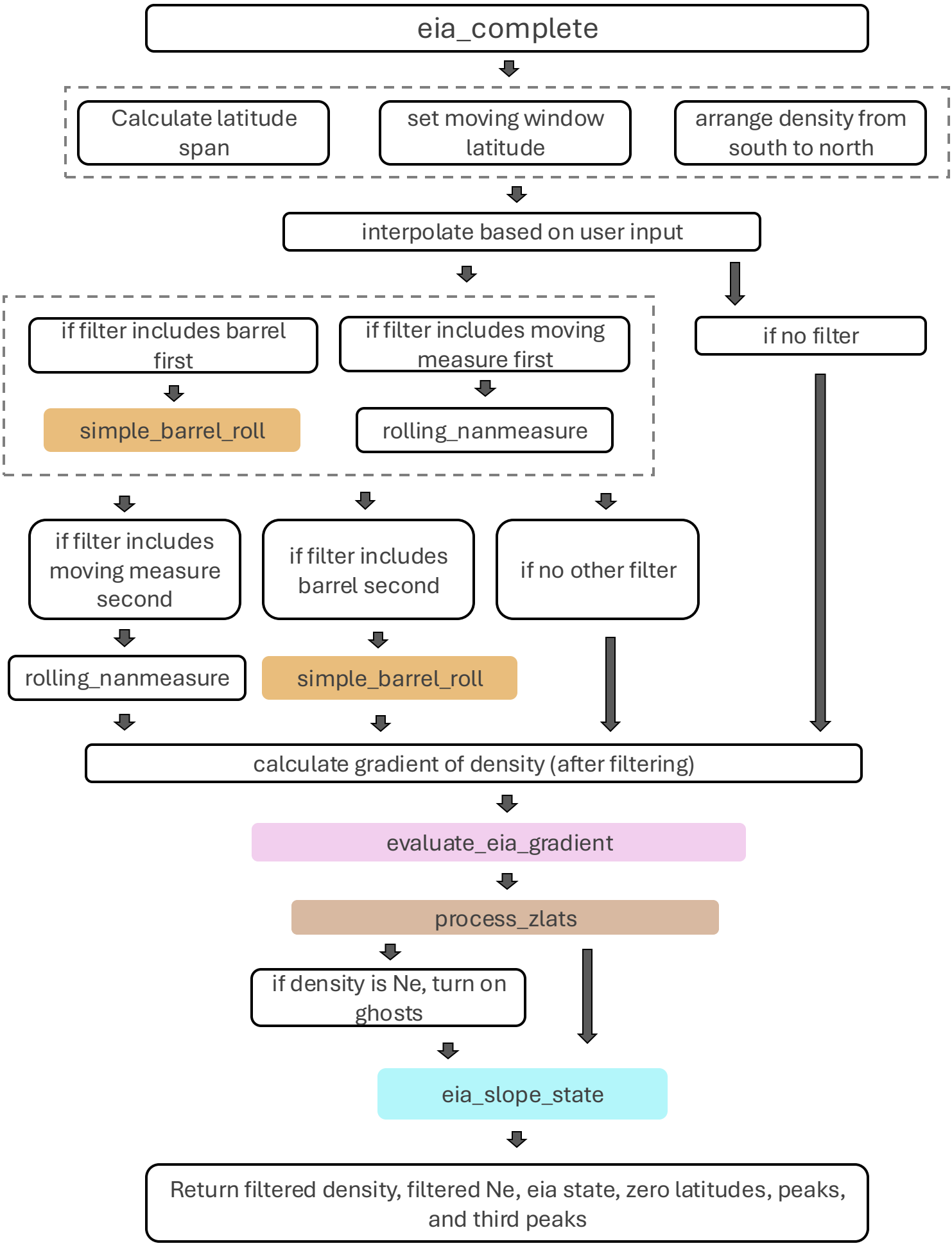
In [10]:
```
fig = Mad_LSS_plot(NiMad, 'eia', date_range, model_name='NIMO',
                                PorC='PC', DayNight=True, LT_range=[7, 23
                                coin=True)
```


eia 2014/01/01-2014/01/31
LSS1
LSS2
LSS3
LSS4
NIMO
Coin Flip
7.1-22.9 LT
23.1-6.9 LT
Skill Score
Percent Correct
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]:

EIA Detection Flow Charts


eia_complete
simple_barrel_roll
rolling_nanmeasure
evaluate_eia_gradient
process_zlats
eia_slope_state
get_zlopes
find_maxima
find_second_maxima
zero_max
toomanymax
third_peak
single_peak_rules
double_peak_rules
peak_span
flat_rules
peak_span
peak_span
saddle_N_S
ghost_check
third_peak
peak_span
ghost_NS_rules
KEY
Related
Code Name
Grouped in same code


eia_complete
Calculate latitude span
set moving window latitude
arrange density from south to north
interpolate based on user input
if filter includes barrel first
if filter includes moving measure first
if no filter
simple_barrel_roll
rolling_nanmeasure
if filter includes moving measure second
if filter includes barrel second
if no other filter
rolling_nanmeasure
simple_barrel_roll
calculate gradient of density (after filtering)
evaluate_eia_gradient
process_zlats
if density is Ne, turn on ghosts
eia_slope_state
Return filtered density, filtered Ne, eia state, zero latitudes, peaks, and third peaks

Simple barrel roll

Set first points as starting x and y contact points

While the index is less than the last point in the array, look for contact points

Set the barrel radius from input

Establish a region of interest within barrel radius

Calculate the angular distance between current contact and region of interest points

Choose smallest angular distance as next contact point

Linearly interpolate contacts to be the same length as original array

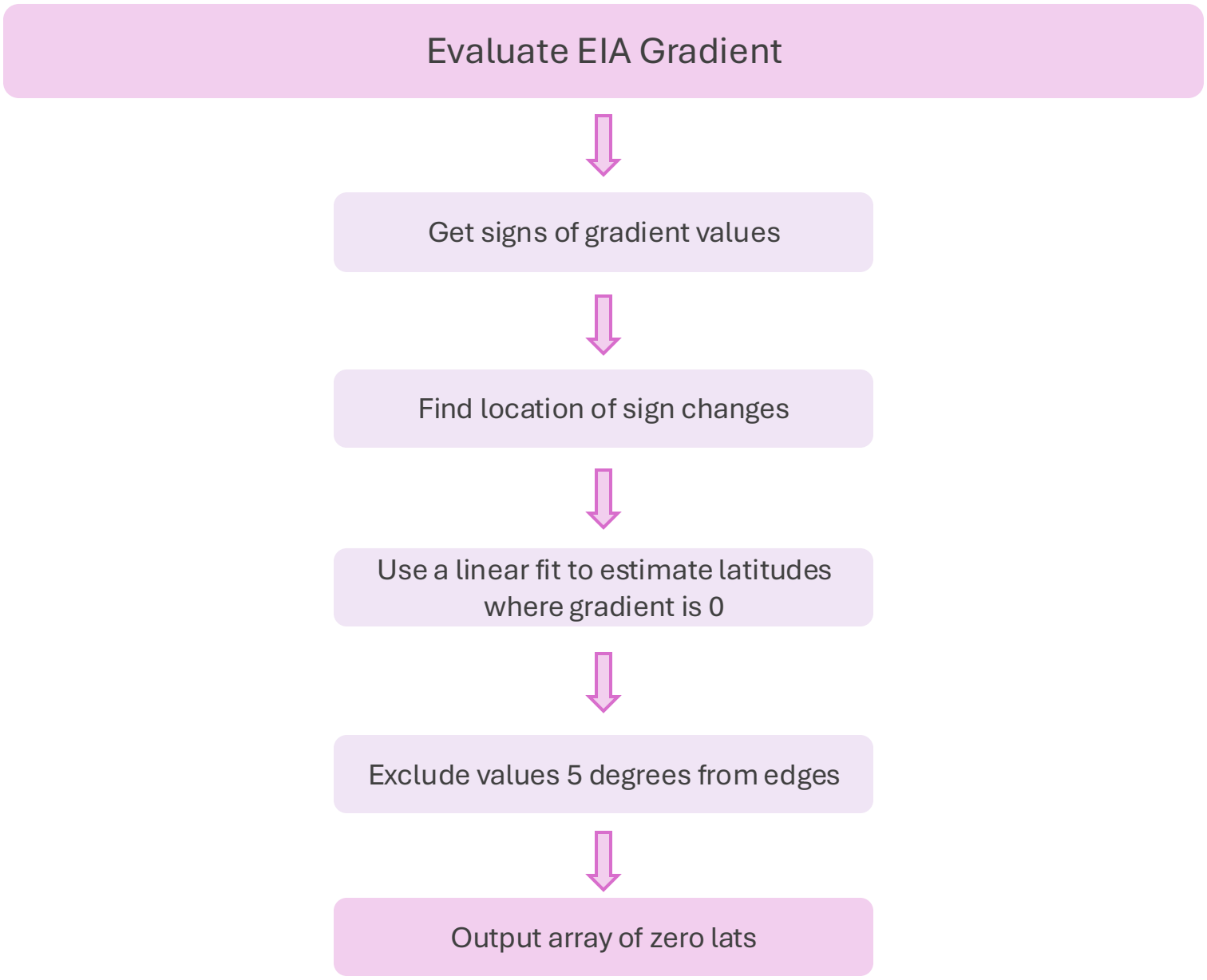
If envelope = 1

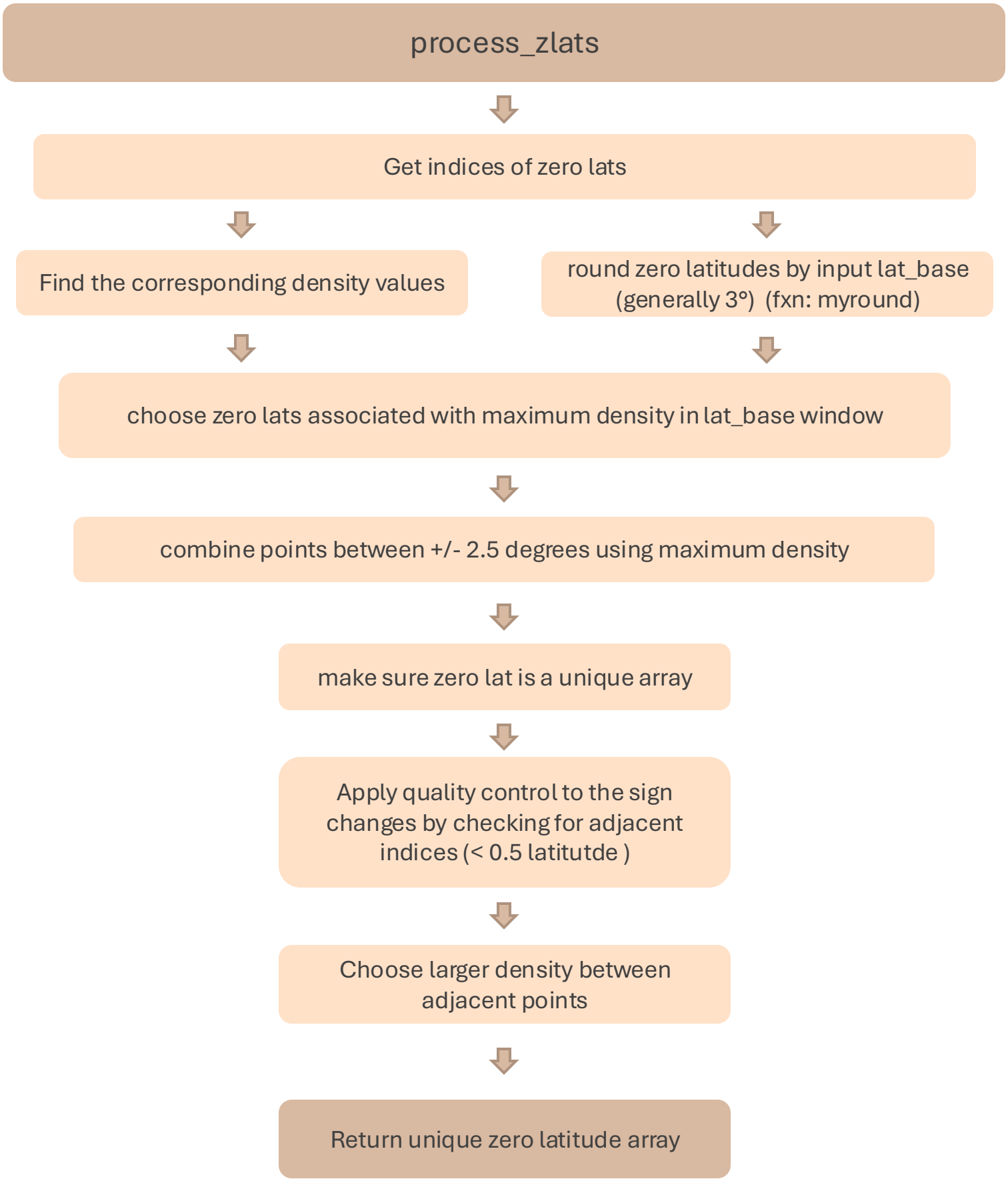
If envelope != 1

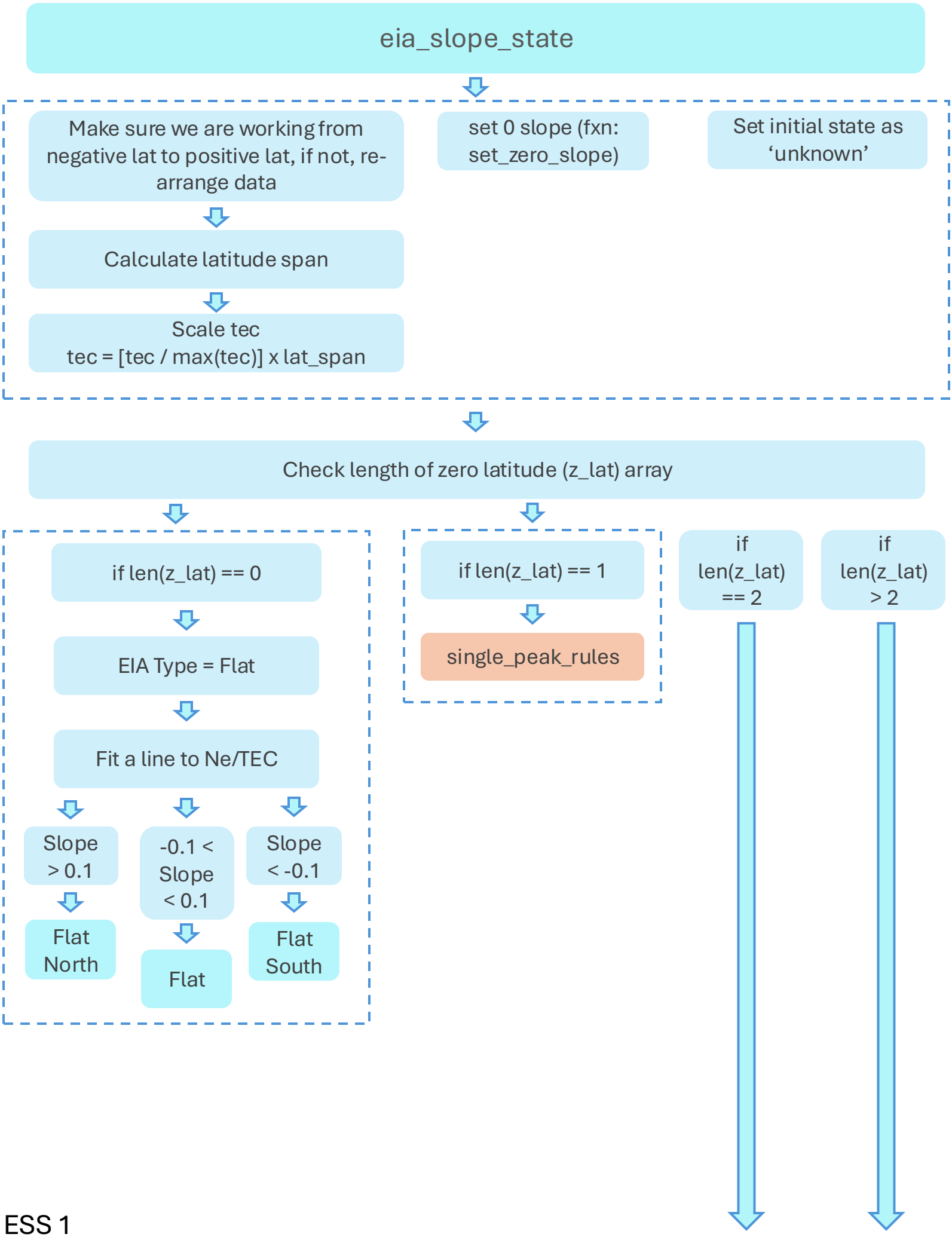
Include points between +0.02*max y and – 0.06* min y, rescale array

Do not use an envelope, return array that is rescaled

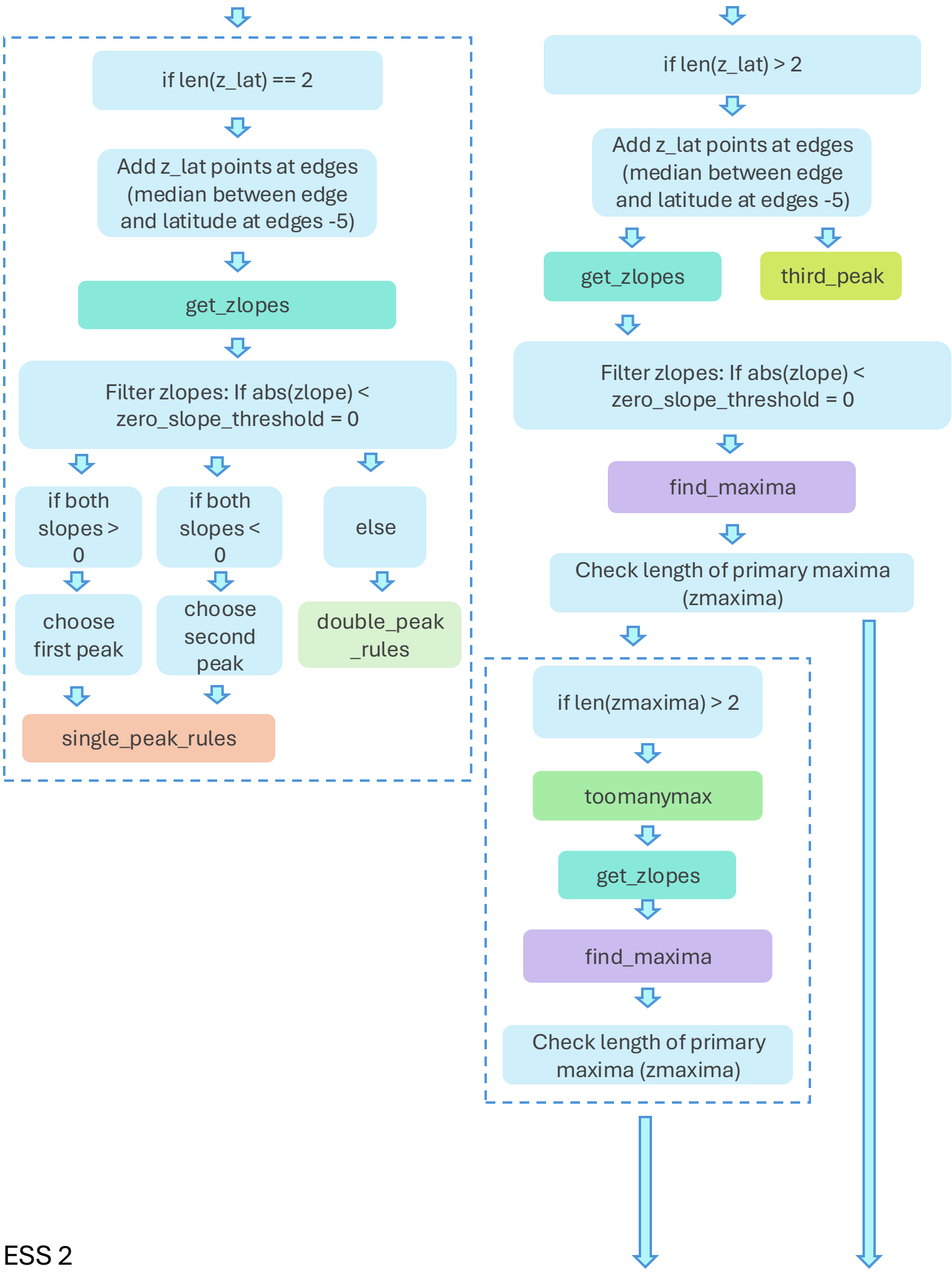
Output barrel-rolled data


Evaluate EIA Gradient
Get signs of gradient values
Find location of sign changes
Use a linear fit to estimate latitudes where gradient is 0
Exclude values 5 degrees from edges
Output array of zero lats

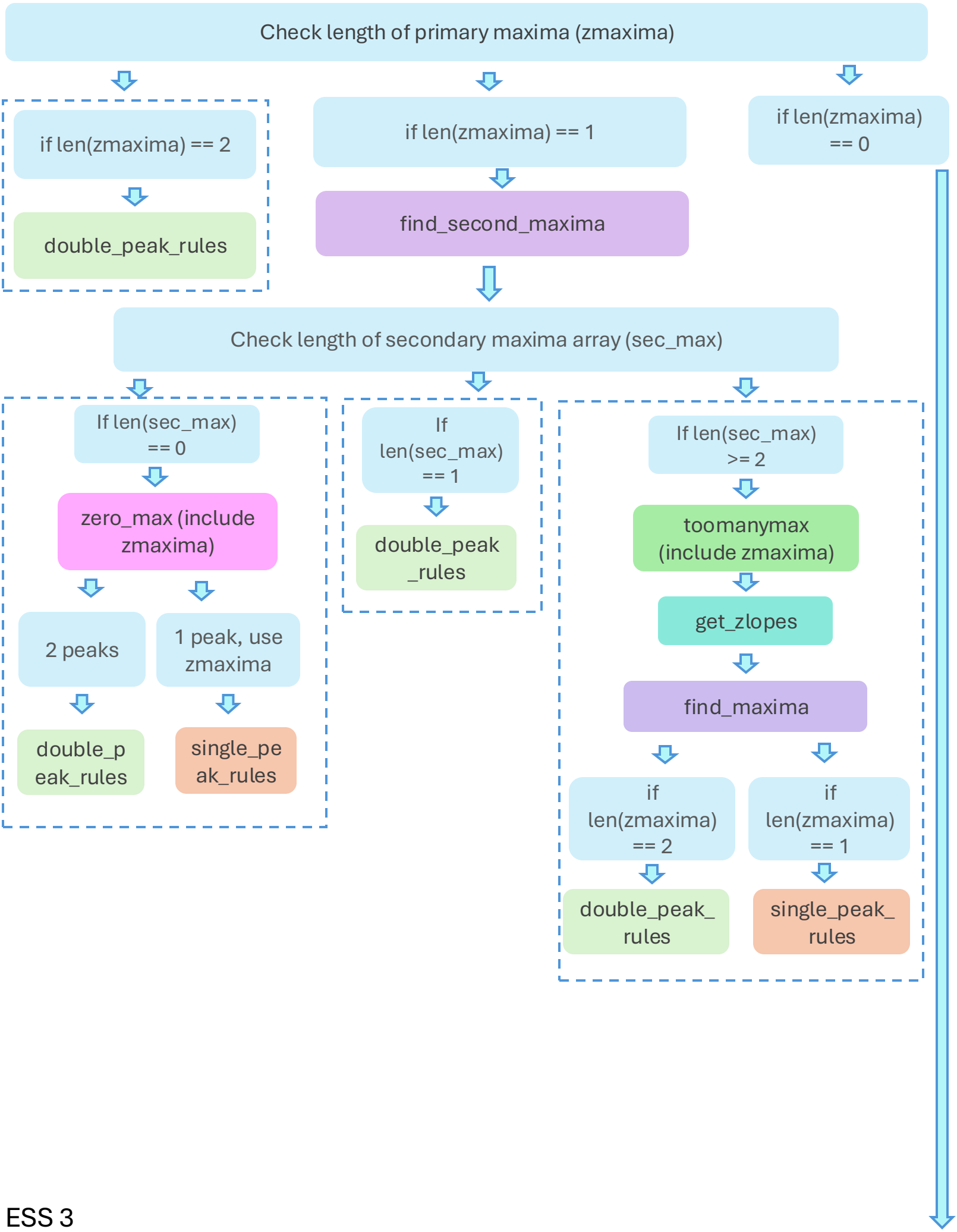

process_zlats
Get indices of zero lats
Find the corresponding density values
round zero latitudes by input lat_base (generally 3°) (fxn: myround)
choose zero lats associated with maximum density in lat_base window
combine points between +/- 2.5 degrees using maximum density
make sure zero lat is a unique array
Apply quality control to the sign changes by checking for adjacent indices (< 0.5 latitutde)
Choose larger density between adjacent points
Return unique zero latitude array


eia_slope_state
Make sure we are working from negative lat to positive lat, if not, re-arrange data
set 0 slope (fxn: set_zero_slope)
Set initial state as 'unknown'
Calculate latitude span
Scale tec
tec = [tec / max(tec)] x lat_span
Check length of zero latitude (z_lat) array
if len(z_lat) == 0
EIA Type = Flat
Fit a line to Ne/TEC
Slope > 0.1
-0.1 < Slope < 0.1
Slope < -0.1
Flat North
Flat
Flat South
if len(z_lat) == 1
single_peak_rules
if len(z_lat) == 2
if len(z_lat) > 2

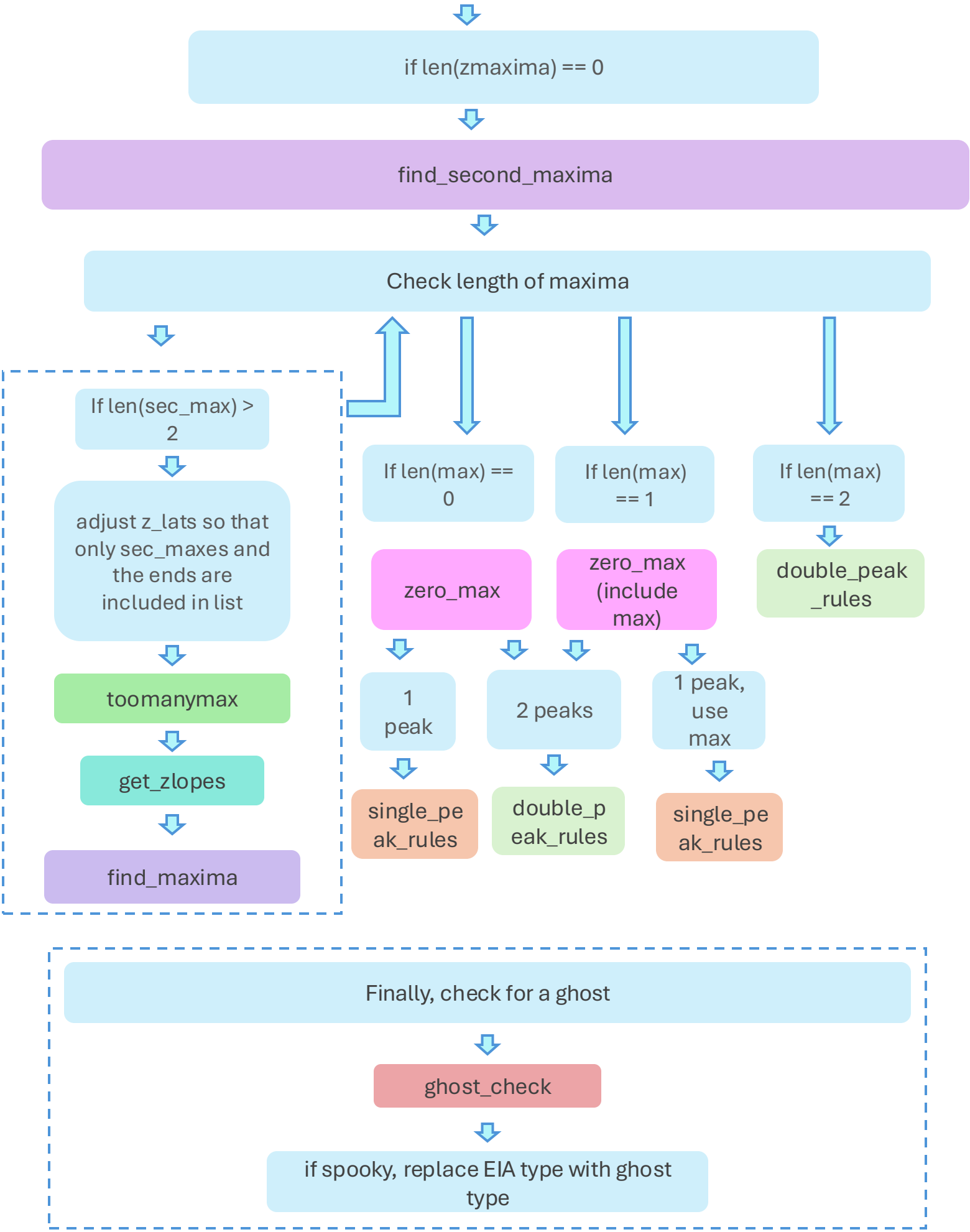
ESS 1


if len(z_lat) == 2
Add z_lat points at edges (median between edge and latitude at edges -5)
get_zlopes
Filter zlopes: If abs(zlope) < zero_slope_threshold = 0
if both slopes > 0
if both slopes < 0
else
choose first peak
choose second peak
double_peak _rules
single_peak_rules
if len(z_lat) > 2
Add z_lat points at edges (median between edge and latitude at edges -5)
get_zlopes
third_peak
Filter zlopes: If abs(zlope) < zero_slope_threshold = 0
find_maxima
Check length of primary maxima (zmaxima)
if len(zmaxima) > 2
toomanymax
get_zlopes
find_maxima
Check length of primary maxima (zmaxima)

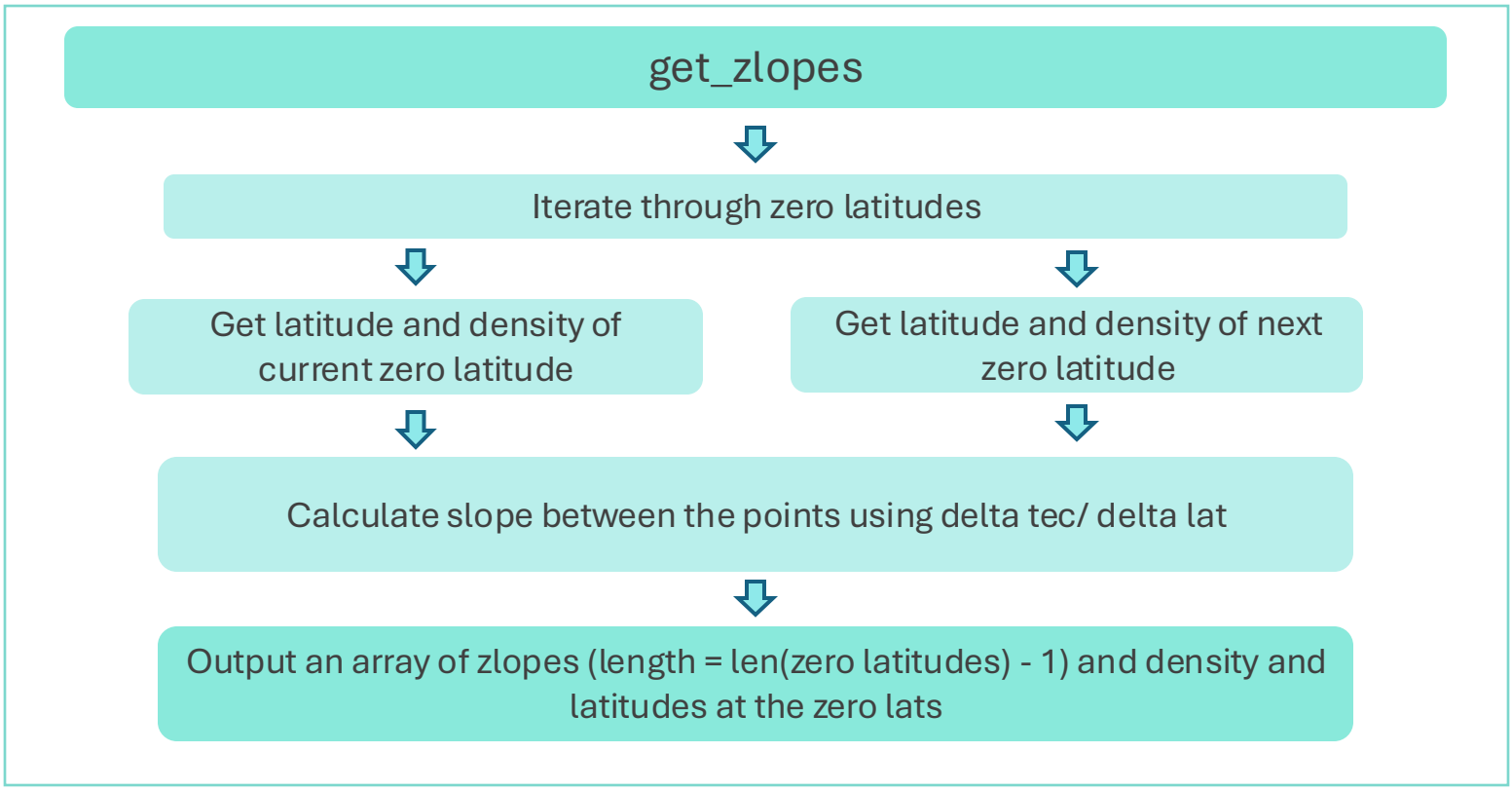
ESS 2

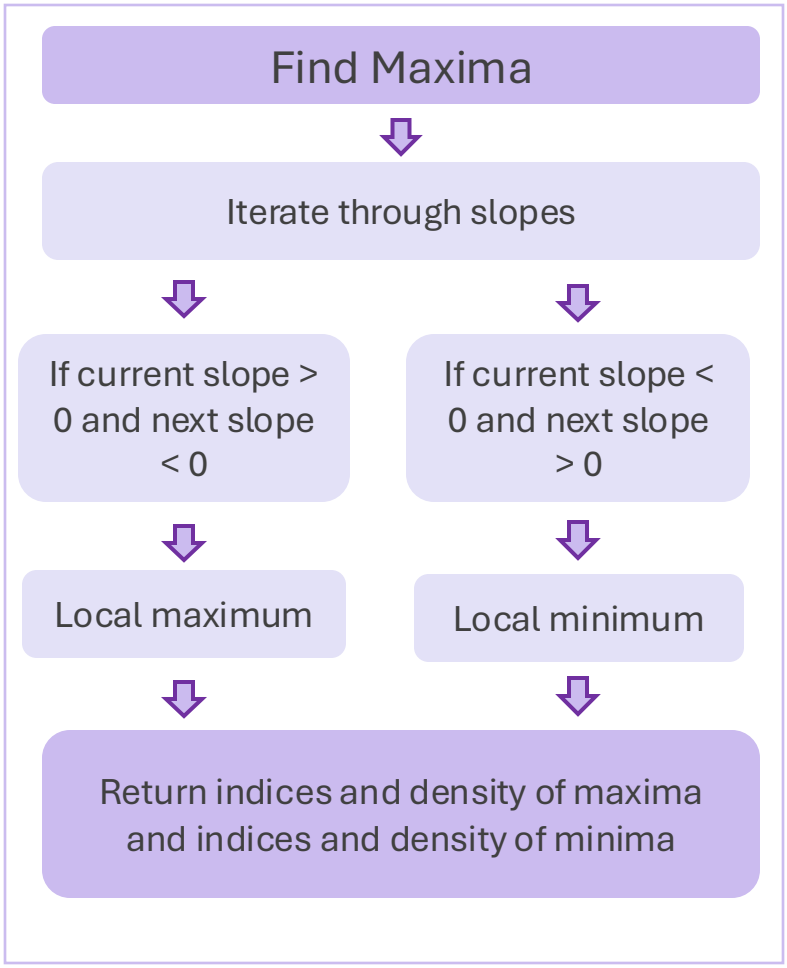

Check length of primary maxima (zmaxima)
if len(zmaxima) == 2
double_peak_rules
if len(zmaxima) == 1
find_second_maxima
if len(zmaxima) == 0
Check length of secondary maxima array (sec_max)
If len(sec_max) == 0
zero_max (include zmaxima)
2 peaks
1 peak, use zmaxima
double_peak_rules
single_peak_rules
If len(sec_max) == 1
double_peak_rules
If len(sec_max) >= 2
toomanymax (include zmaxima)
get_zlopes
find_maxima
if len(zmaxima) == 2
if len(zmaxima) == 1
double_peak_rules
single_peak_rules

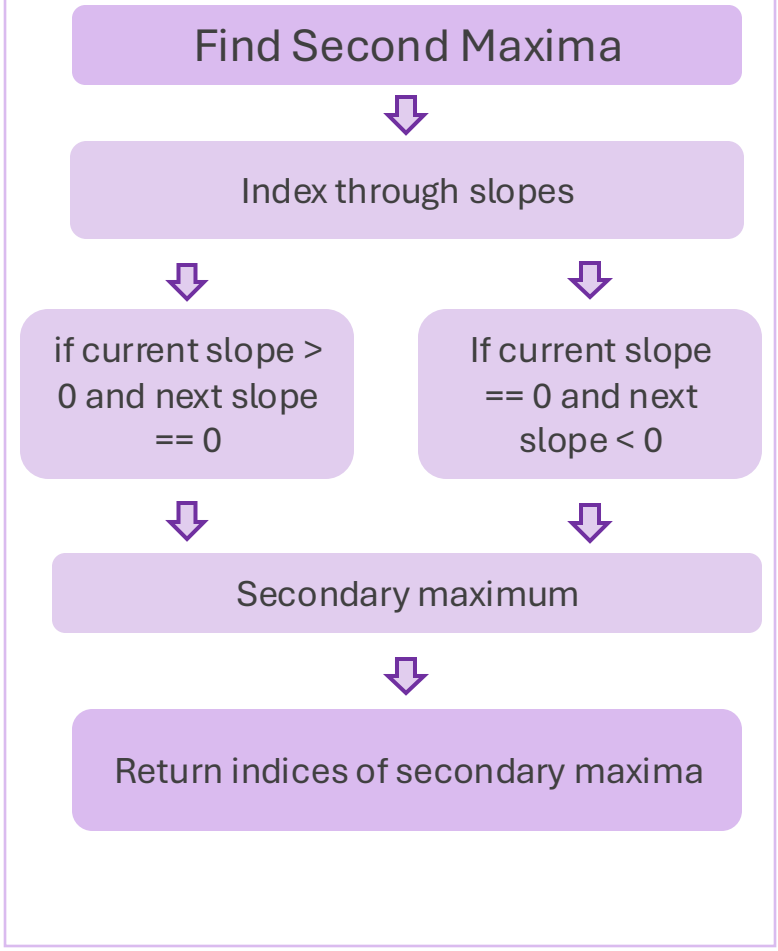
ESS 3

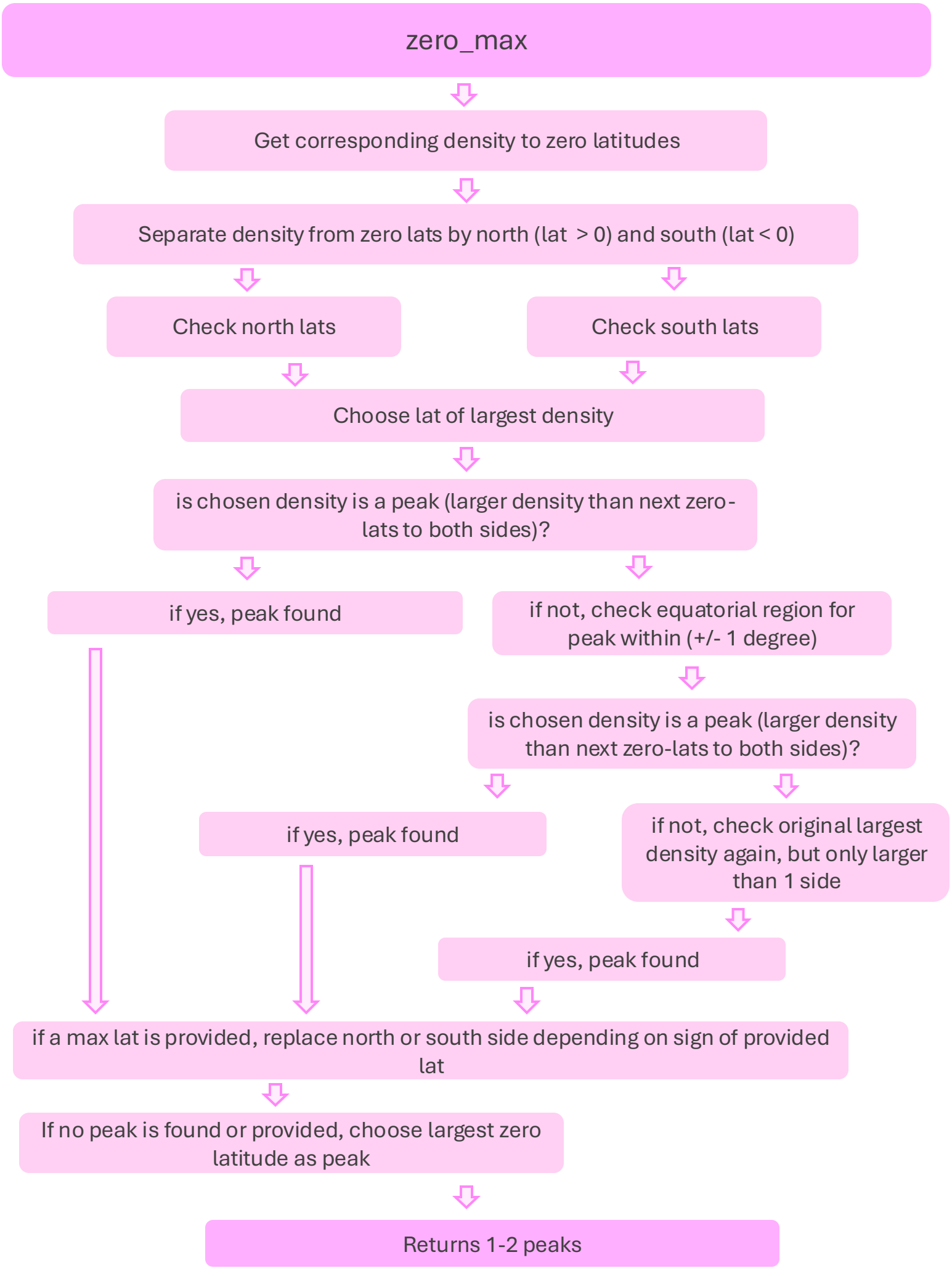

if len(zmaxima) == 0
find_second_maxima
Check length of maxima
If len(sec_max) > 2
adjust z_lats so that only sec_maxes and the ends are included in list
toomanymax
get_zlopes
find_maxima
If len(max) == 0
If len(max) == 1
If len(max) == 2
zero_max
zero_max (include max)
double_peak_rules
1 peak
2 peaks
1 peak, use max
single_peak_rules
double_peak_rules
single_peak_rules
Finally, check for a ghost
ghost_check
if spooky, replace EIA type with ghost type

ESS 4


get_zlopes
Iterate through zero latitudes
Get latitude and density of current zero latitude
Get latitude and density of next zero latitude
Calculate slope between the points using delta tec/ delta lat
Output an array of zlopes (length = len(zero latitudes) - 1) and density and latitudes at the zero lats


Find Maxima
Iterate through slopes
If current slope > 0 and next slope < 0
If current slope < 0 and next slope > 0
Local maximum
Local minimum
Return indices and density of maxima and indices and density of minima


Find Second Maxima
Index through slopes
if current slope > 0 and next slope == 0
If current slope == 0 and next slope < 0
Secondary maximum
Return indices of secondary maxima


zero_max
Get corresponding density to zero latitudes
Separate density from zero lats by north (lat > 0) and south (lat < 0)
Check north lats
Check south lats
Choose lat of largest density
is chosen density is a peak (larger density than next zero-lats to both sides)?
if yes, peak found
if not, check equatorial region for peak within (+/- 1 degree)
is chosen density is a peak (larger density than next zero-lats to both sides)?
if yes, peak found
if not, check original largest density again, but only larger than 1 side
if yes, peak found
if a max lat is provided, replace north or south side depending on sign of provided lat
If no peak is found or provided, choose largest zero latitude as peak
Returns 1-2 peaks

toomanymax

find corresponding density to zero latitudes

Separate tec from zero lats by north (between 3 and 20 mlat), south (between -3 and -20 mlat), and equator (between between +/-3)

if there are north lats

If no other lat provided or provided lat is south

Choose lat closest to equator in north lats

If north lat provided

Use provided lat

If there are equator lats

Choose lat of largest density

If there are south lats

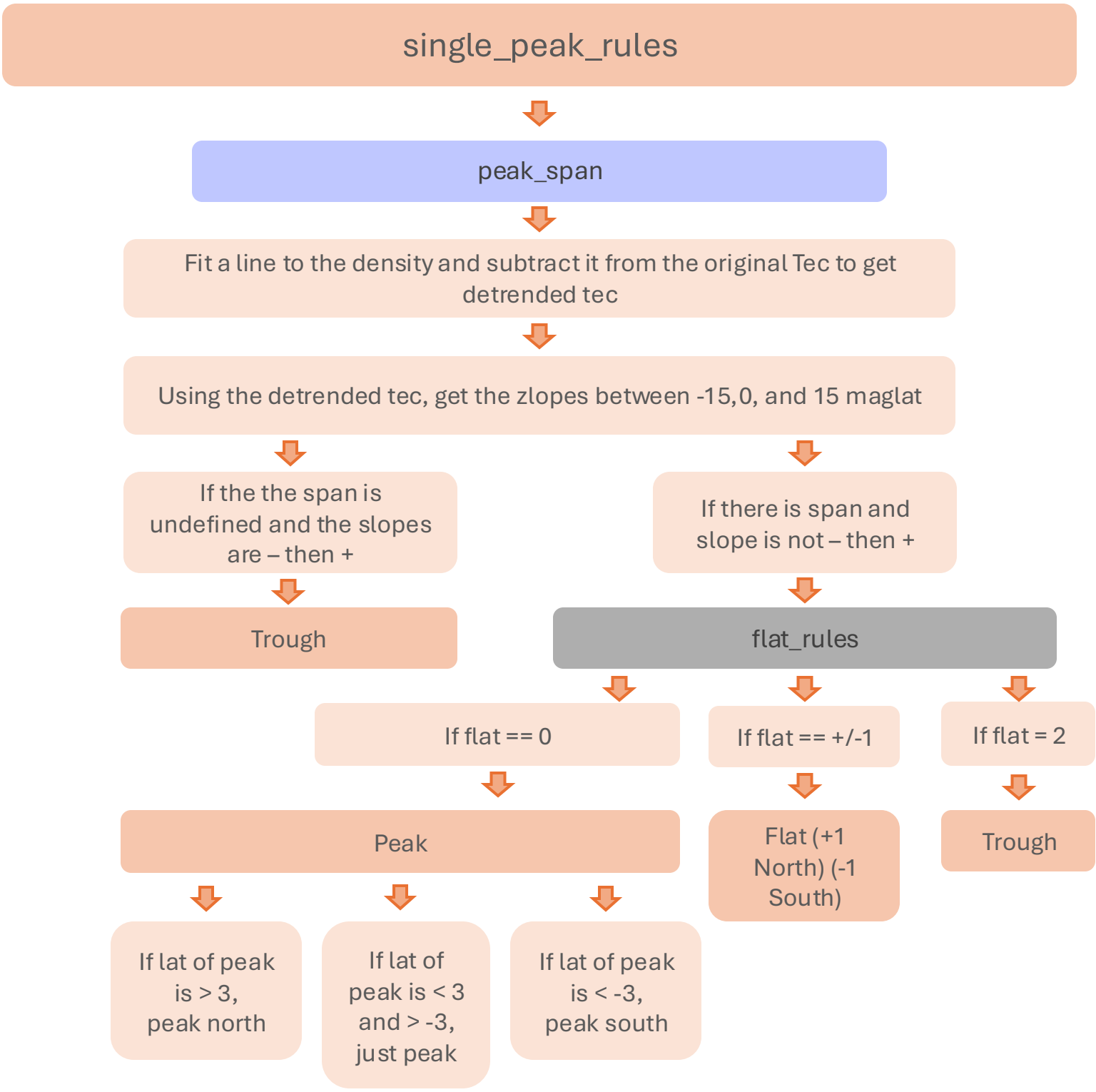
If no other lat provided or provided lat is north

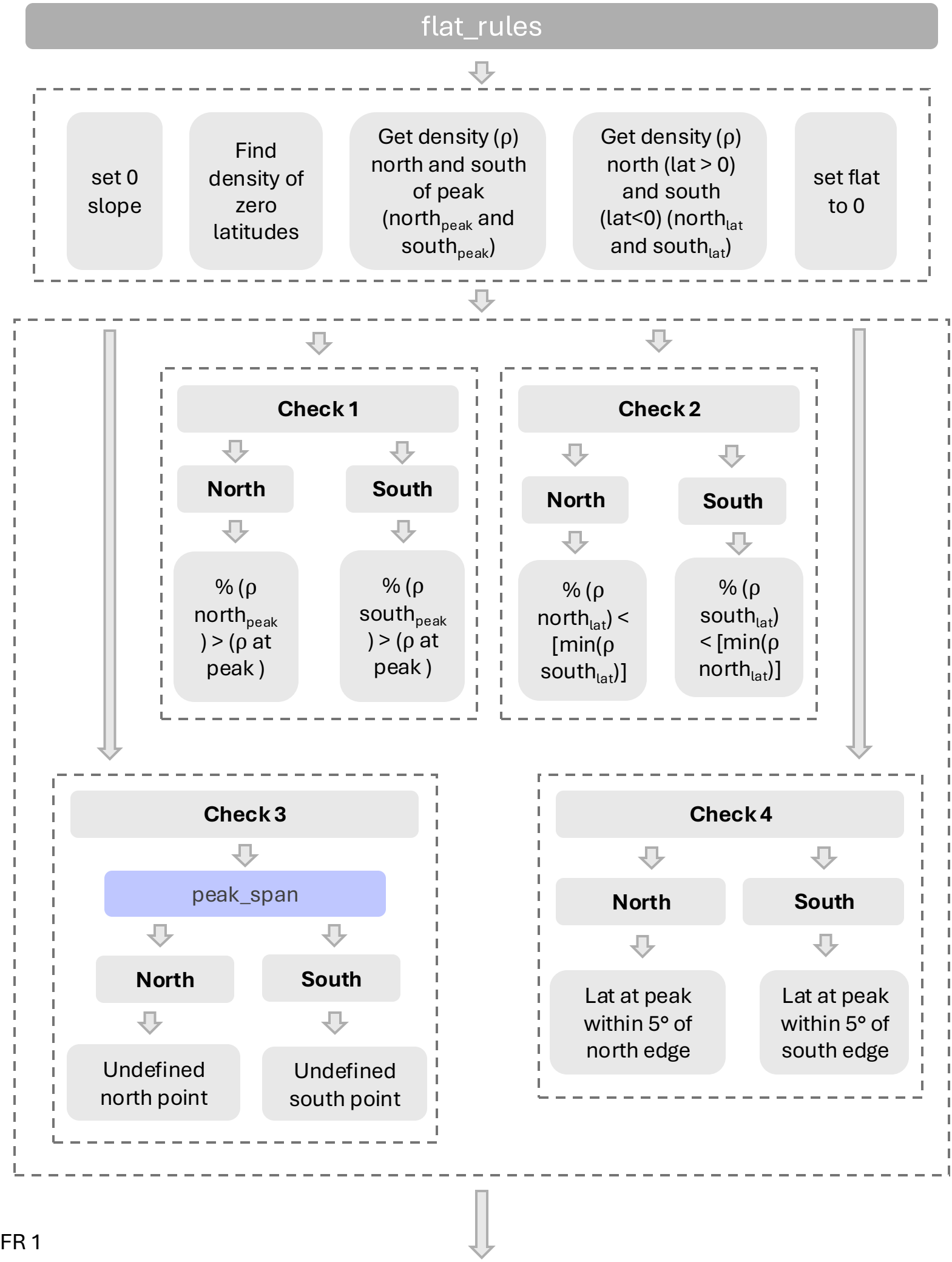
Choose lat closest to equator in south lats

If north lat provided

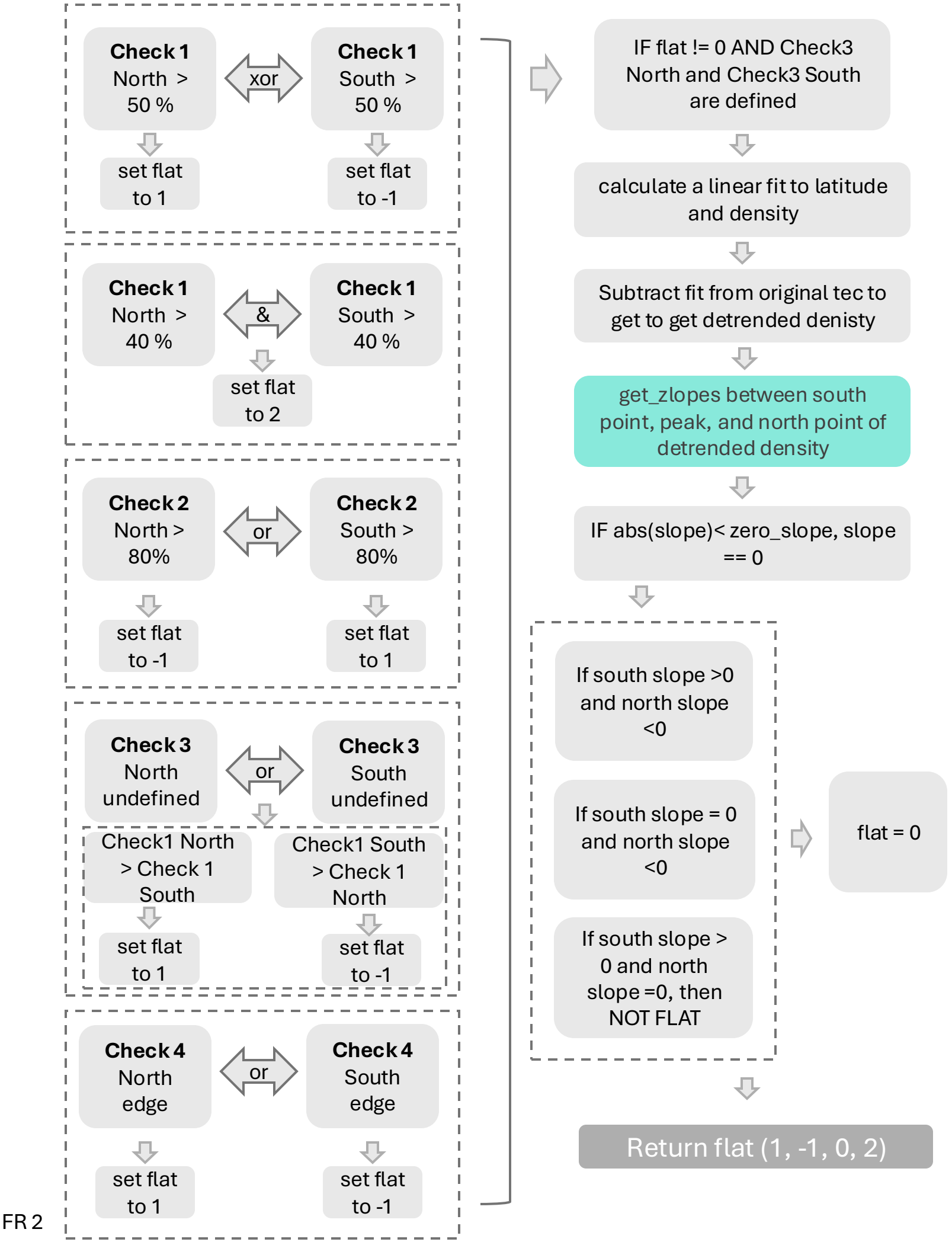
Use provided lat

Return unique new latitude array that contains a maximum of 5 values [south edge, closest south, equator max, closest north, and north edge]

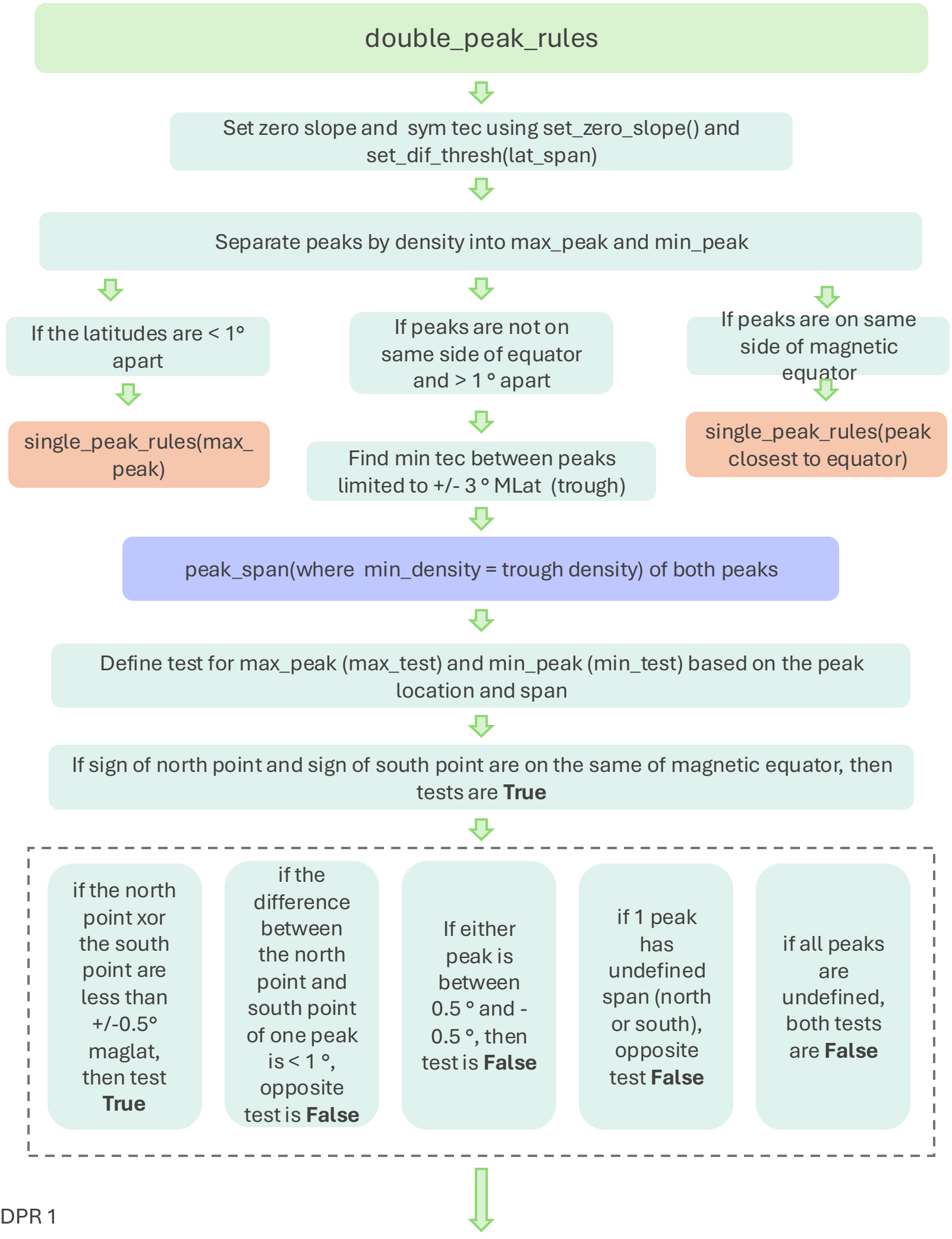

single_peak_rules
peak_span
Fit a line to the density and subtract it from the original Tec to get detrended tec
Using the detrended tec, get the zlopes between -15,0, and 15 maglat
If the the span is undefined and the slopes are – then +
If there is span and slope is not – then +
Trough
flat_rules
If flat == 0
If flat == +/-1
If flat = 2
Peak
Flat (+1 North) (-1 South)
Trough
If lat of peak is > 3, peak north
If lat of peak is < 3 and > -3, just peak
If lat of peak is < -3, peak south

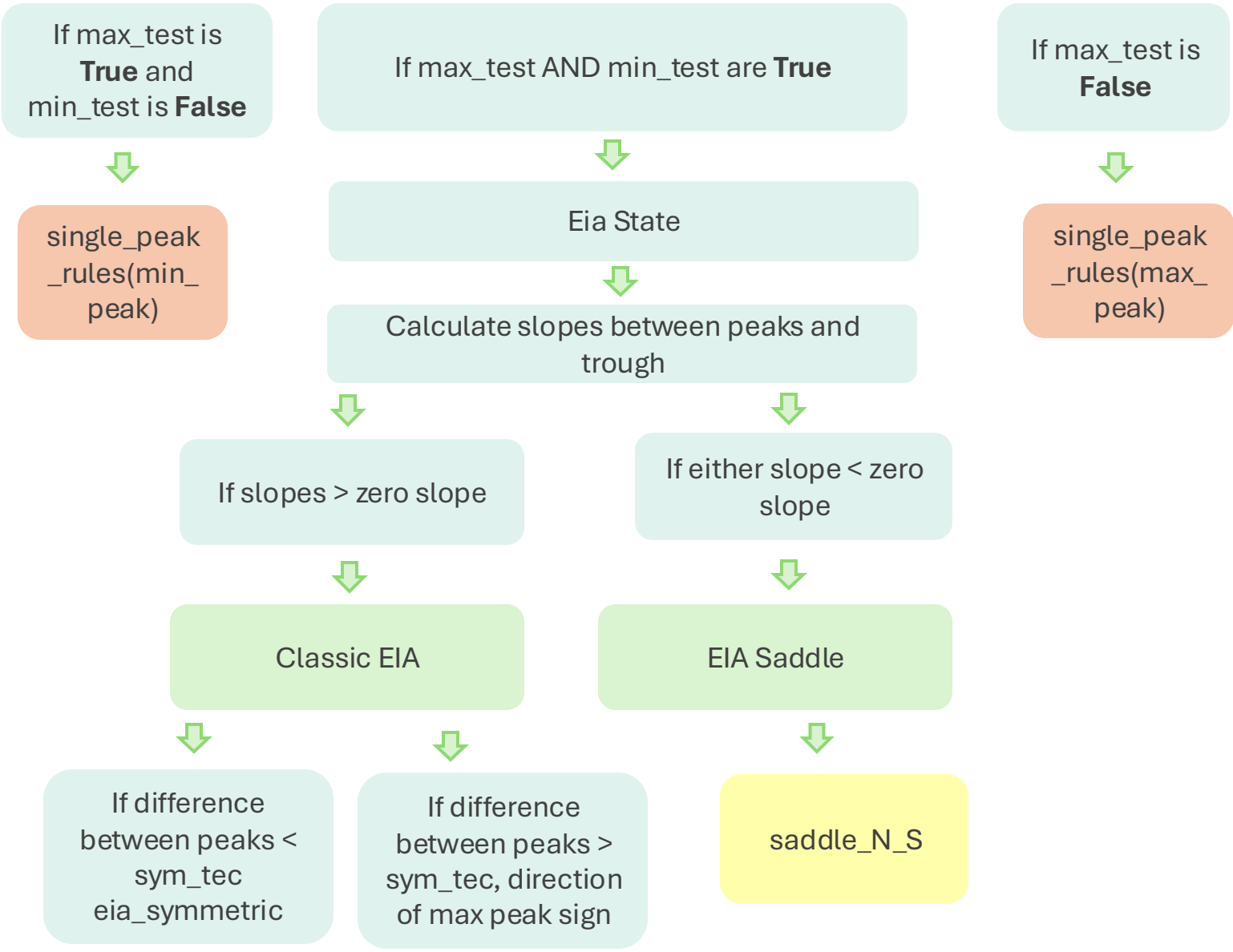

flat_rules
set 0 slope
Find density of zero latitudes
Get density (ρ) north and south of peak (north$_{peak}$ and south$_{peak}$)
Get density (ρ) north (lat > 0) and south (lat<0) (north$_{lat}$ and south$_{lat}$)
set flat to 0
Check 1
North
South
% (ρ north$_{peak}$) > (ρ at peak)
% (ρ south$_{peak}$) > (ρ at peak)
Check 2
North
South
% (ρ north$_{lat}$) < [min(ρ south$_{lat}$)]
% (ρ south$_{lat}$) < [min(ρ north$_{lat}$)]
Check 3
peak_span
North
South
Undefined north point
Undefined south point
Check 4
North
South
Lat at peak within 5° of north edge
Lat at peak within 5° of south edge

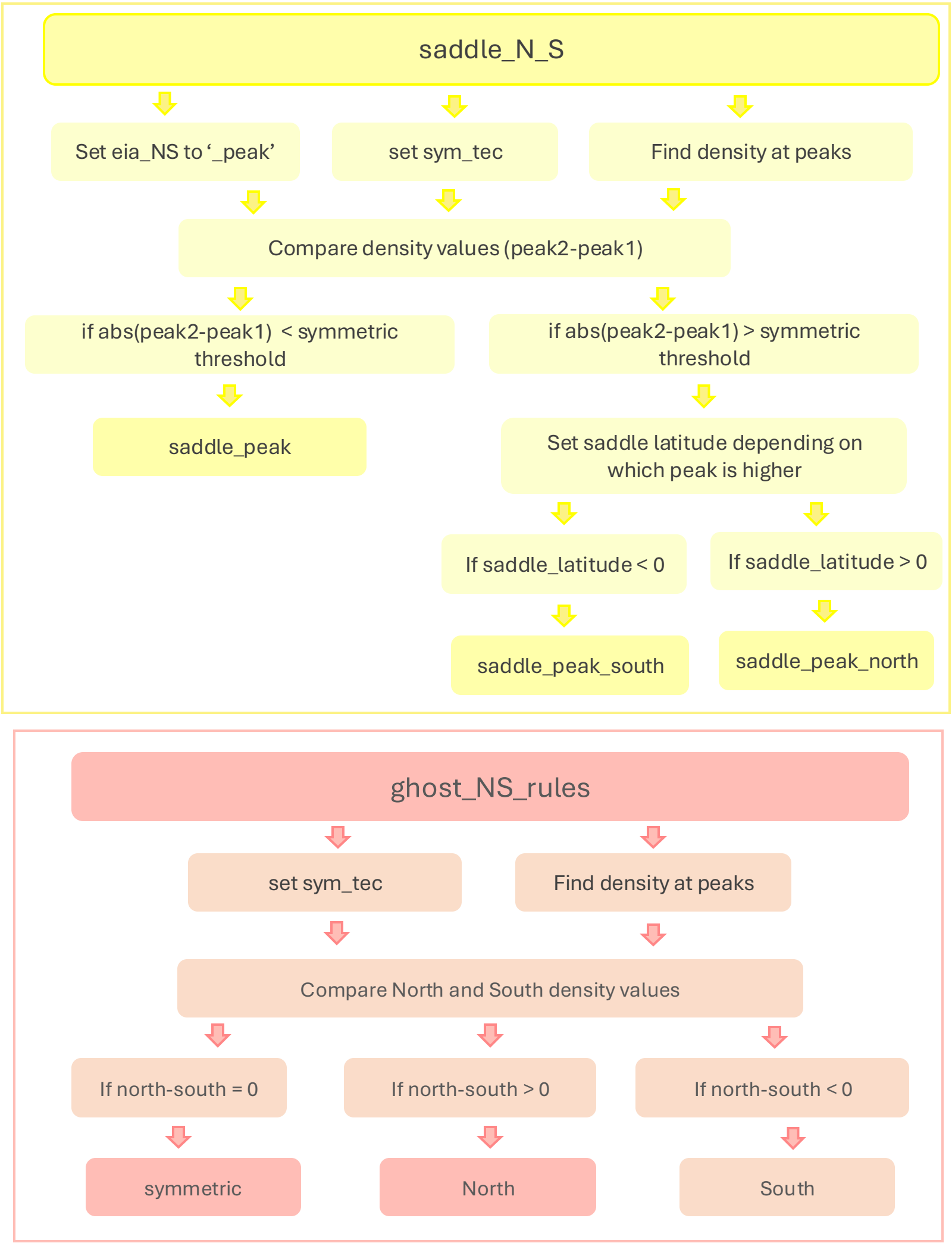
FR 1


Check 1 North > 50 %
xor
Check 1 South > 50 %
set flat to 1
set flat to -1
Check 1 North > 40 %
&
Check 1 South > 40 %
set flat to 2
Check 2 North > 80%
or
Check 2 South > 80%
set flat to -1
set flat to 1
Check 3 North undefined
or
Check 3 South undefined
Check1 North > Check 1 South
Check1 South > Check 1 North
set flat to 1
set flat to -1
Check 4 North edge
or
Check 4 South edge
set flat to 1
set flat to -1
IF flat != 0 AND Check3 North and Check3 South are defined
calculate a linear fit to latitude and density
Subtract fit from original tec to get to get detrended denisty
get_zlopes between south point, peak, and north point of detrended density
IF abs(slope)< zero_slope, slope == 0
If south slope >0 and north slope <0
If south slope = 0 and north slope <0
If south slope > 0 and north slope =0, then NOT FLAT
flat = 0
Return flat (1, -1, 0, 2)

FR 2

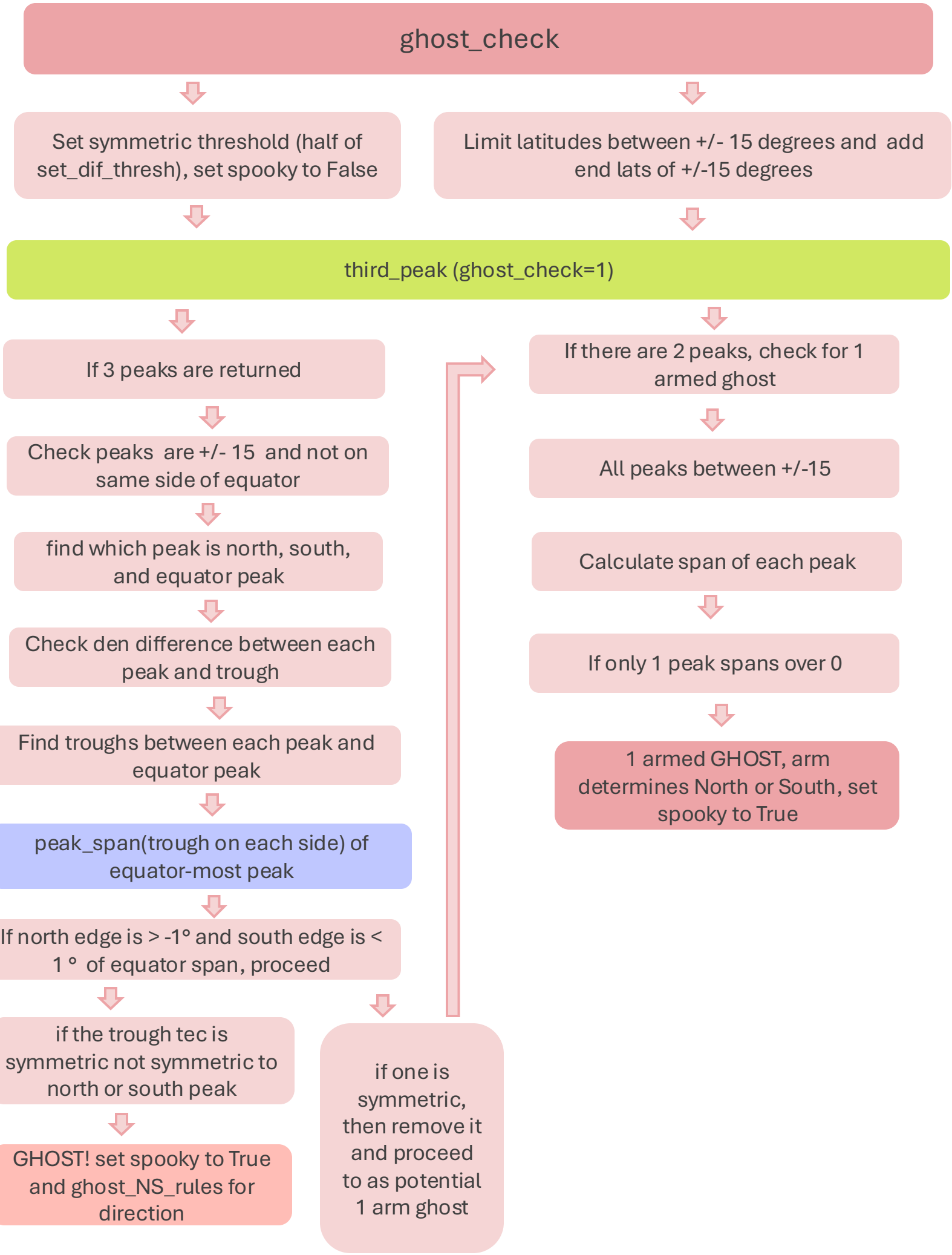

double_peak_rules
Set zero slope and sym tec using set_zero_slope() and set_dif_thresh(lat_span)
Separate peaks by density into max_peak and min_peak
If the latitudes are < 1° apart
single_peak_rules(max_peak)
If peaks are not on same side of equator and > 1 ° apart
Find min tec between peaks limited to +/- 3 ° MLat (trough)
If peaks are on same side of magnetic equator
single_peak_rules(peak closest to equator)
peak_span(where min_density = trough density) of both peaks
Define test for max_peak (max_test) and min_peak (min_test) based on the peak location and span
If sign of north point and sign of south point are on the same of magnetic equator, then tests are True
if the north point xor the south point are less than +/-0.5° maglat, then test True
if the difference between the north point and south point of one peak is < 1 °, opposite test is False
If either peak is between 0.5 ° and -0.5 °, then test is False
if 1 peak has undefined span (north or south), opposite test False
if all peaks are undefined, both tests are False


If max_test is **True** and min_test is **False**
single_peak _rules(min_ peak)
If max_test AND min_test are **True**
Eia State
Calculate slopes between peaks and trough
If slopes > zero slope
If either slope < zero slope
Classic EIA
EIA Saddle
If difference between peaks < sym_tec eia_symmetric
If difference between peaks > sym_tec, direction of max peak sign
saddle_N_S
If max_test is **False**
single_peak _rules(max_ peak)


saddle_N_S
Set eia_NS to '_peak'
set sym_tec
Find density at peaks
Compare density values (peak2-peak1)
if abs(peak2-peak1) < symmetric threshold
if abs(peak2-peak1) > symmetric threshold
saddle_peak
Set saddle latitude depending on which peak is higher
If saddle_latitude < 0
If saddle_latitude > 0
saddle_peak_south
saddle_peak_north
ghost_NS_rules
set sym_tec
Find density at peaks
Compare North and South density values
If north-south = 0
If north-south > 0
If north-south < 0
symmetric
North
South


Peak Span
Find tec at peak lats
Find tec on north and south of the peak
If a trough tec is defined
If a trough tec is not defined
Set t_base to fraction of (peak-trough) + trough
Set t_base to fraction of peak tec
Check for north (south) tec below t_base
Start from peak and search north (south) until a value drops below t_base
Keep track of indices just before tec dropped below t_base, if none found append -99
Loop back until t_base = trough_tec or 1/32 of peak tec
Remove -99 from both north and south indices
If there are non -99 points
If no points are left
Report north and south points of len(points)*div (halfway point if div = 0.5)
Only remove north -99 from north and south -99 from south instead of both
Report defined north and south points using div, else report -99


third_peak
get_zlopes
find_maxima
If ghost_check = 1 AND there are 2 maxima (one arm ghost?)
Get maxima closest and farthest from equator
Check hemisphere of the value farthest from the equator
North hemisphere
South hemisphere
Look for third peak between closest point to equator and -15 mlat
Look for third peak between closest point to equator and +15 mlat
If there are z Locs in the range, choose point farthest from equator
Add a minimum between equator peak and new peak
get_zlopes of new peak
If it is not peak, keep 2 peaks
If it is a peak, add it in to make 3 peaks
If there are is more than 1 max
Return latitudes of maxima
ELSE
If ghost_check = 1
If ghost_check != 1
If there are 3 maxima
Return latitudes of maxima
If there are not 3 maxima
Return empty array


ghost_check
Set symmetric threshold (half of set_dif_thresh), set spooky to False
Limit latitudes between +/- 15 degrees and add end lats of +/-15 degrees
third_peak (ghost_check=1)
If 3 peaks are returned
Check peaks are +/- 15 and not on same side of equator
find which peak is north, south, and equator peak
Check den difference between each peak and trough
Find troughs between each peak and equator peak
peak_span(trough on each side) of equator-most peak
If north edge is > -1° and south edge is < 1 ° of equator span, proceed
if the trough tec is symmetric not symmetric to north or south peak
GHOST! set spooky to True and ghost_NS_rules for direction
if one is symmetric, then remove it and proceed to as potential 1 arm ghost
If there are 2 peaks, check for 1 armed ghost
All peaks between +/-15
Calculate span of each peak
If only 1 peak spans over 0
1 armed GHOST, arm determines North or South, set spooky to True