Atmosphere Plotter Notebook

This notebook read the spectral datacubes generated by `PolarVortice/AtmosphereGenerator.py`.

The datacubes are located at `PolarVortice/output/`. In this script, I configured it to read only `production` datacubes.

1. **Step 1**: Read Spectral Monitoring Data
2. **Step 2**: Generate Synthetic Spectra Cube with Sonora

Step 0: Initialization

In [654…

```
import astropy.constants as c
import astropy.units as u
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import os
import re
import h5py
from astropy.convolution import convolve, Box1DKernel
from collections import defaultdict
import matplotlib.pyplot as plt
```

In [656…

```
# function to print levels of nested dicts
def print_nested_dict(d, indent=0):
    for key, value in d.items():
        print("  " * indent + str(key) + ":")
        if isinstance(value, dict):
            print_nested_dict(value, indent + 1)
        else:
            print("  " * (indent + 1) + str(value))

# lamda function to create nested_dict
nested_dict = lambda: defaultdict(nested_dict)

# function to bring array value to one
def bring_to_one(array):
    shift = (array.max() + array.min())/2
    return 1 + array - shift

# itertools to generated random marker
import itertools
markerRandomList = itertools.cycle((',', '+', '.', 'o', '*'))
linestyleRandomList = itertools.cycle(('--', '-', '-.'))
### Usage
# for n in y:
#     plt.plot(x,n, marker = next(marker), linestyle='')
```

Step 1: Read Spectral Monitoring Data Cube

Initialization

- Choose class of models to read.
- model_class, model_id, inclination, starting_time, end_time, frame_numbers
- data structure:

dict:static|dynamic|nopolar| => dict:inclination_value:'string' => dict:time_value:'string' => numpy:gray_array:2d_image

1A) Read files into 1 dictionary contain 3 classes: Polar_static, Polar_dynamic, NoPolar

In [561…

```
directory = "/Users/nguyendat/Documents/GitHub/polar_vortice/PolarVortice/c
production0_list = []
other_list = []

productionKey = 'production0'
# modelclassKey = ['polarStatic', 'polarDynamic', 'noPolar']
# modelclassKey = ['polarStatic', 'noPolar']

print('Production key: "%s"'%productionKey)
print('\nThe production datacubes includes:')
for foldername in os.listdir(directory):
    if productionKey in foldername and 'polarDynamic' not in foldername:
        print(foldername)
        production0_list.append(directory+foldername+'/')
    else:
        other_list.append(directory+foldername+'/')
print('\n')
# ================================================================
# Function to categorize file names
# ================================================================
def categorize_filenames(directory_list):
    # create empty, pre-nested dicts
    datacubes = nested_dict()
    metadatas = nested_dict()

    for directory in directory_list:
        # Iterate over files in the directory
        for filename in os.listdir(directory):
            # print(filename)
            # Check if the file is a H5 file
            if filename.endswith(".h5"):
                parts = re.findall("\[(.*?)\]", filename)
                model_id = str(parts[1])
                class_name = str(parts[0])
                incli_value = str(parts[2])
                t1t2FrameNo_value = parts[3] + "-" + parts[4] + "-" + parts

                # Load numpy array from H5 file
                with h5py.File(os.path.join(directory, filename), 'r') as :
                    data = f['dataset'][:]
                    datacubes[model_id][class_name][incli_value][t1t2FrameI
```

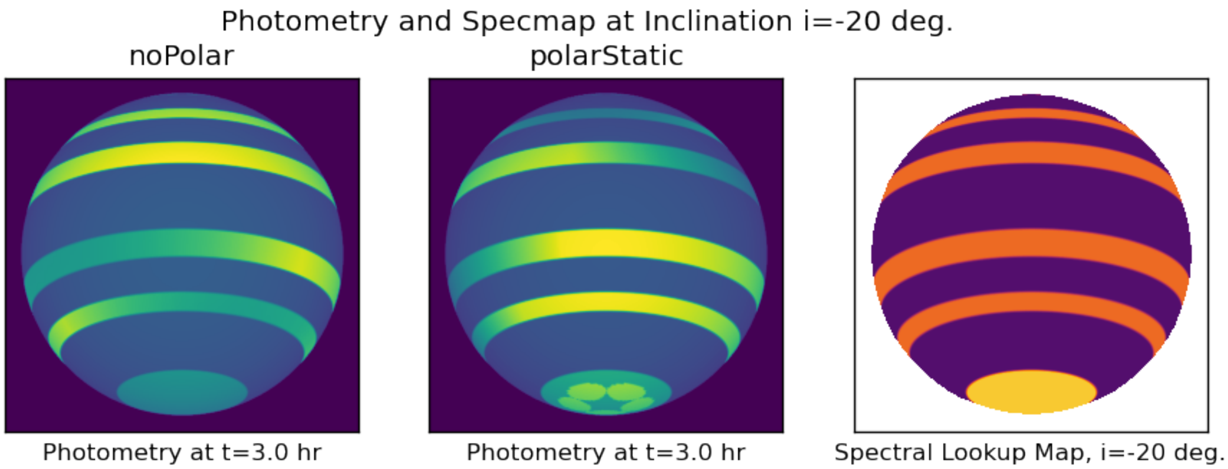
```
            # Check if the file is a pickle file
            if filename.endswith(".pkl"):
                parts = re.findall("\[(.*?)\]", filename)
                model_id = parts[2]
                class_name = parts[1]
                incli_value = parts[3]
                t1t2FrameNo_value = parts[4] + "-" + parts[5] + "-" + parts

                # Load metadata from pickle file
                with open(os.path.join(directory, filename), 'rb') as f:
                    data = pickle.load(f)
                    metadatas[model_id][class_name][incli_value][t1t2FrameN

    return datacubes, metadatas

# read the production datacube
datacubes, metadatas = categorize_filenames(production0_list)

# ============================================================
# Print bookkeeping information
# ============================================================
modelclasses = []
modelclasses = list(datacubes['production0'].keys())

print('============================================================')
print('Datacubes contains these model classes:', modelclasses)

for model in modelclasses:
    incli = list(datacubes['production0'][model].keys())
    incli = sorted([int(x) for x in incli])
    incli = [str(x) for x in incli]

    photo_config = list(datacubes['production0'][model][incli[0]].keys())

    # =======================================================
    # Choose which photometry configuration to use:
    # Currently photo_config
    # =======================================================
    photo_config0 = photo_config[0]

    print('\n[%s] contains these inclination'%model, sorted(incli))
    print('for [t0]-[t1]-[FrameNumber]:', photo_config)

    metakeys = list(metadatas['production0'][model][incli[0]]['0-30-30'].ke
    no_frame = datacubes['production0'][model][iang][photo_config0].shape[
    t0, t1 = int(photo_config0.split('-')[0]), int(photo_config0.split('-'

    time_array = np.linspace(t0, t1, no_frame)

    print('Total no. of frames:', no_frame, ', cadence: %.1f min'%((t1-t0)/

print('\nThe metadata contains these attributes: \n', metakeys)
print('============================================================')
```

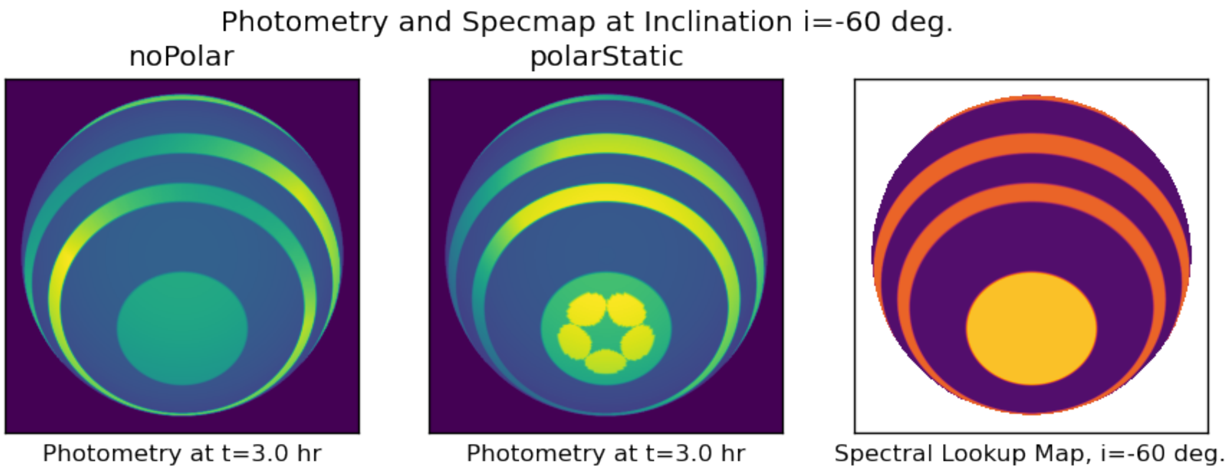
```
Production key: "production0"

The production datacubes includes:
dataCube[production0][noPolar][i][0][60][60]
dataCube[production0][polarStatic][i][0][30][30]
dataCube[production0][polarStatic][i][0][60][60]
dataCube[production0][noPolar][i][0][30][30]


============================================================
Datacubes contains these model classes: ['noPolar', 'polarStatic']

[noPolar] contains these inclination ['-10', '-20', '-30', '-40', '-50', '-
60', '-70', '-80', '-90', '0']
for [t0]-[t1]-[FrameNumber]: ['0-60-60', '0-30-30']
Total no. of frames: 60 , cadence: 60.0 min

[polarStatic] contains these inclination ['-10', '-20', '-30', '-40', '-50'
, '-60', '-70', '-80', '-90', '0']
for [t0]-[t1]-[FrameNumber]: ['0-30-30', '0-60-60']
Total no. of frames: 30 , cadence: 60.0 min

The metadata contains these attributes:
 ['modu_config', 'modelname', 'inclin', 'Fband', 'Fambient', 'Pband', 'Ppol
', 'config_columns', 'config', 'specmap', 'speckey', 'cond_is_amb', 'cond_i
s_band', 'cond_is_pol']
============================================================
```

1B) Photometry and specmaps

In [633…

```
# ============================================================
# [[Plot]]
# Plot the first photometry-frame, spectra-map for one inclin
# of each model class
# ============================================================
%matplotlib inline
plt.close('all')
for iang in ['-20', '-60']:
    fig, axs = plt.subplots(1, 3, dpi=120, figsize=(9,3))
    cadence = (t1-t0)/no_frame
    fig.suptitle('Photometry and Specmap at Inclination i=%s deg.'%iang)
    for i, model in enumerate(modelclasses):
        photo_config0 = photo_config[0]
        if model == 'polarStatic': timepoint = 3
        else: timepoint = 2
        gray = datacubes['production0'][model][iang][photo_config0][timepo
        axs[i].imshow(gray)
        axs[i].set_title(model)
        axs[i].set_xticks([]), axs[i].set_yticks([])
        axs[i].set_xticks([]), axs[i].set_yticks([])

    specmap = metadatas['production0'][model][iang][photo_config0]['specma
    axs[2].imshow(specmap, cmap='inferno')
    axs[2].set_xticks([]), axs[2].set_yticks([])

    axs[0].set_xlabel('Photometry at t=%.1f hr'%(timepoint*cadence))
    axs[1].set_xlabel('Photometry at t=%.1f hr'%(timepoint*cadence))
    axs[2].set_xlabel('Spectral Lookup Map, i=%s deg.'%iang)
```





1C) Creating flux dictionaries

This section will assume one is using one identical time-array for 3 model classes.

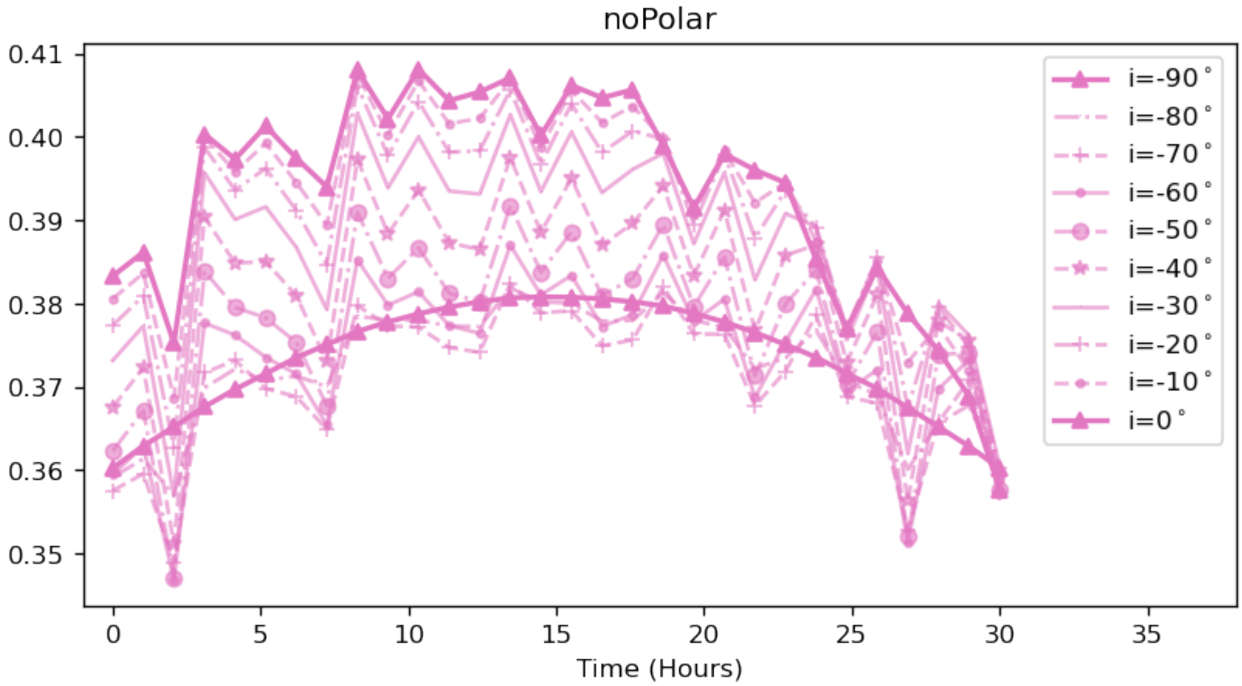
In [661…

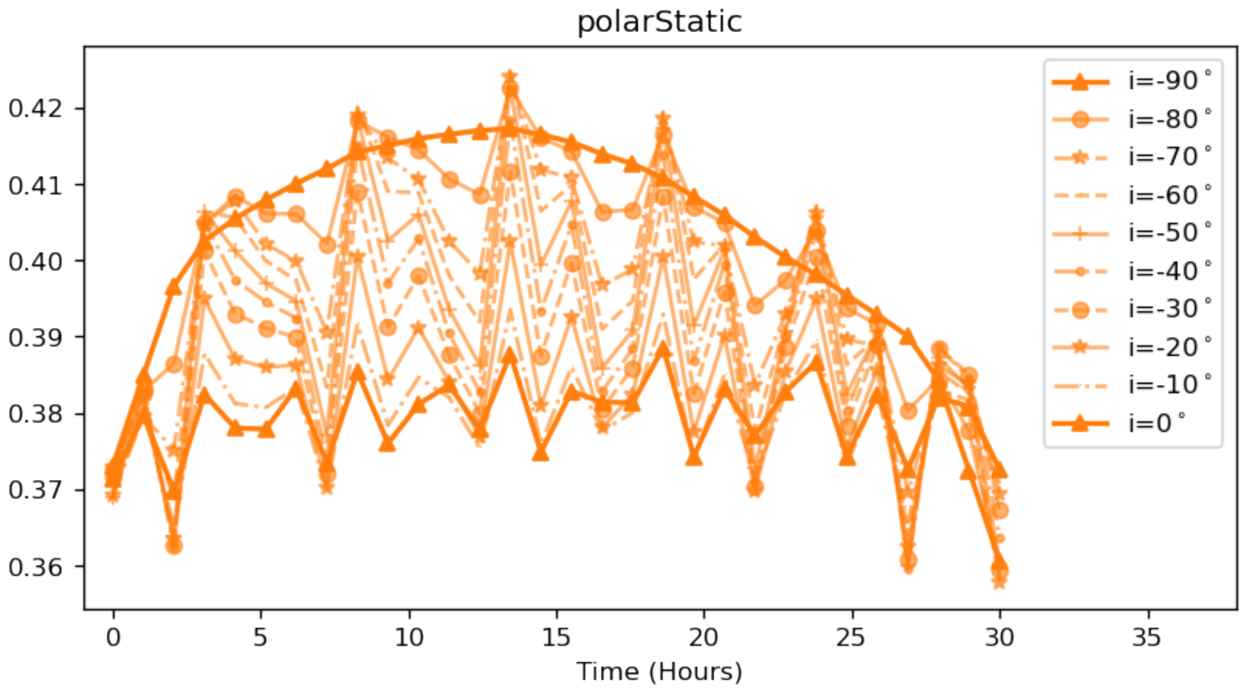
```
# ==============================================================
# Read photometry cubes, mask out region by feature-class
# and save the *fluxes*, as well as *masked images*
# and the masked specmaps *specmasks*
# ==============================================================

fluxes = nested_dict() # format: normflux, flux_bytype, fraction_bytype
images = nested_dict() # format: [frameim, ambim, bandim, polim]
specmasks = nested_dict() # format: [is_amb, is_band, is_pol]
photo_config0 = photo_config[0]

do_gaussian_filter = False

gaussian_sigma = 0

for model in modelclasses:
    for iang in incli:
        meta_iang = metadatas['production0'][model][iang][photo_config0]
        specmap = meta_iang['specmap']
        con_amb = meta_iang['cond_is_amb']
        is_amb = ((specmap >= con_amb[0]) & (specmap < con_amb[1])).astype
        con_band = meta_iang['cond_is_band']
        is_band = ((specmap >= con_band[0]) & (specmap < con_band[1])).asty
        con_pol = meta_iang['cond_is_pol']
        is_pol = ((specmap >= con_pol[0]) & (specmap < con_pol[1])).astype
```

```
        flux = []
        fluxtyp = []

        # Prepare a dictionary of specmap masks
        specmap_total = metadatas['production0'][model][iang][photo_config(
        specmap_amb = specmap*is_amb
        specmap_band = specmap*is_band
        specmap_pol = specmap*is_pol

        specmap_total[specmap_total == 0] = np.nan
        specmap_amb[specmap_amb == 0] = np.nan
        specmap_band[specmap_band == 0] = np.nan
        specmap_pol[specmap_pol == 0] = np.nan

        frac_amb = np.nansum(specmap_amb)/np.nansum(specmap_total)
        frac_band = np.nansum(specmap_band)/np.nansum(specmap_total)
        frac_pol = np.nansum(specmap_pol)/np.nansum(specmap_total)

        specmasks[model][iang] = [specmap_total, specmap_amb, specmap_band,

        for t in range(no_frame):
            gray = datacubes['production0'][model][iang][photo_config0][t]
            if do_gaussian_filter:
                frameim=gaussian_filter(gray, sigma=gaussian_sigma)
            else: frameim = np.copy(gray)

            ## mask-out band and pole and calculate their
            ## respective flux contribution at each frame
            ambim, polim, bandim = frameim * is_amb, frameim * is_pol, fram

            ambim[ambim == 0] = np.nan
            bandim[bandim == 0] = np.nan
            polim[polim == 0] = np.nan
            frameim[frameim == 0] = np.nan

            fluxtyp.append([np.nanmean(ambim), np.nanmean(bandim), np.nanme
            flux.append(np.nanmean(frameim))

            # prepare dictionary of photometries
            images[model][iang][t] = [frameim, ambim, bandim, polim]

        # prepare dictionary of fluxes
        flux = np.array(flux)
        fluxtyp = np.array(fluxtyp)

        fluxes[model][iang]['norm'] = flux

        fluxes[model][iang]['bytype']['amb'] = np.transpose(fluxtyp)[0] #
        fluxes[model][iang]['bytype']['band'] = np.transpose(fluxtyp)[1]
        fluxes[model][iang]['bytype']['pol'] = np.transpose(fluxtyp)[2]

        fluxes[model][iang]['fraction']['amb'] = frac_amb
        fluxes[model][iang]['fraction']['band'] = frac_band
        fluxes[model][iang]['fraction']['pol'] = frac_pol
```

In [666…

```
# ==============================================================
# [[Plot]]
# Plot all norm_flux with time for each inclinations
# in each model classes
# ==============================================================
plt.close()
colorList = {'polarStatic':'tab:orange', 'noPolar':'tab:pink', 'polarDynami
for model in modelclasses:
    fig, ax1 = plt.subplots(dpi=120, figsize=(8,4))
    for iang in incli:
        y_flux = fluxes[model][iang]['norm']
        if iang == '-90' or iang =='0':
            line, = ax1.plot(time_array, y_flux, c=colorList[model],ls='-',
        else:
            line, = ax1.plot(time_array, y_flux, c=colorList[model],ls=next
        ax1.set_xlim(-1, 38), ax1.set_xlabel('Time (Hours)'), ax2.set_ylabe
        ax1.set_title(model)
        ax1.legend()
```

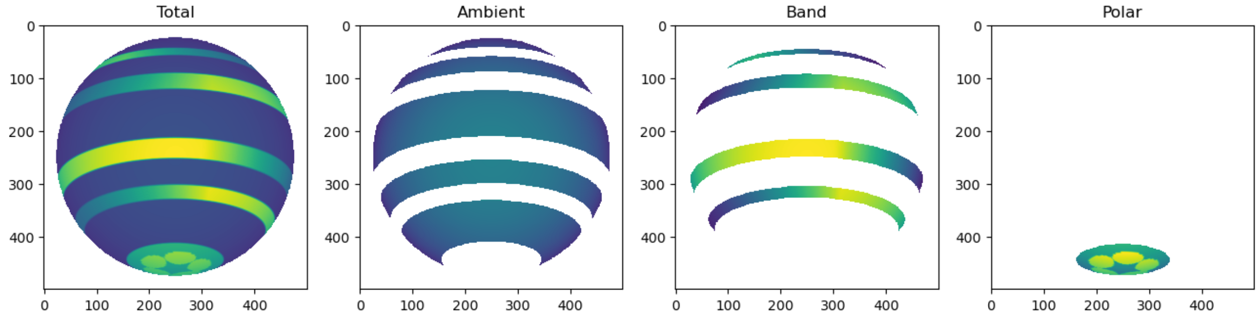


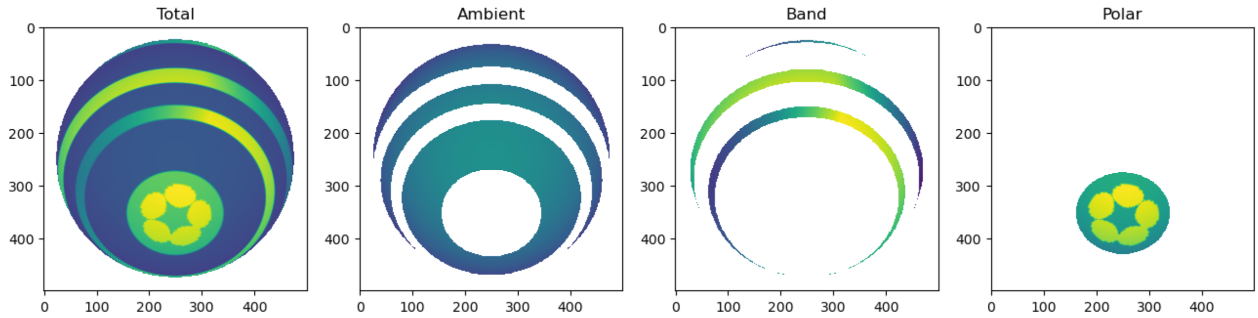


1D) Plot photometry by each feature-class region

In [731…

```python
# ============================================================
# [[Plot]]
# Plot the total photometry, and each feature-class:
# ambient, band, pole
# ============================================================
plot = True
if plot:
    plt.close()
    for iang in ['-20', '-60']:
    # inclination angle iang
        fig, axs = plt.subplots(1,4, figsize=(16, 4), dpi=100)
        labelList = ['Total', 'Ambient', 'Band', 'Polar']
        for i, array in enumerate(images['polarStatic'][iang][5]):
            axs[i].imshow(array)
            axs[i].set_title(labelList[i])
```

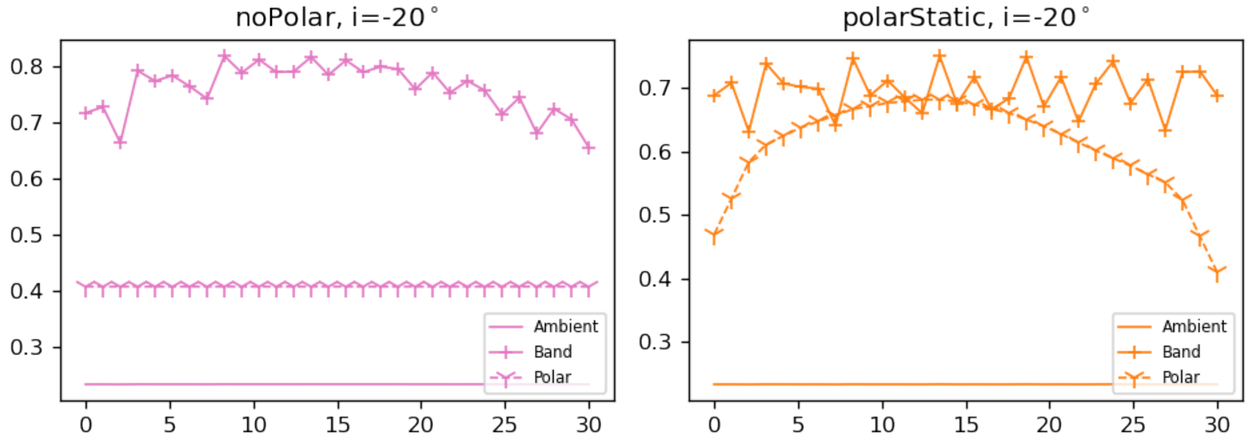


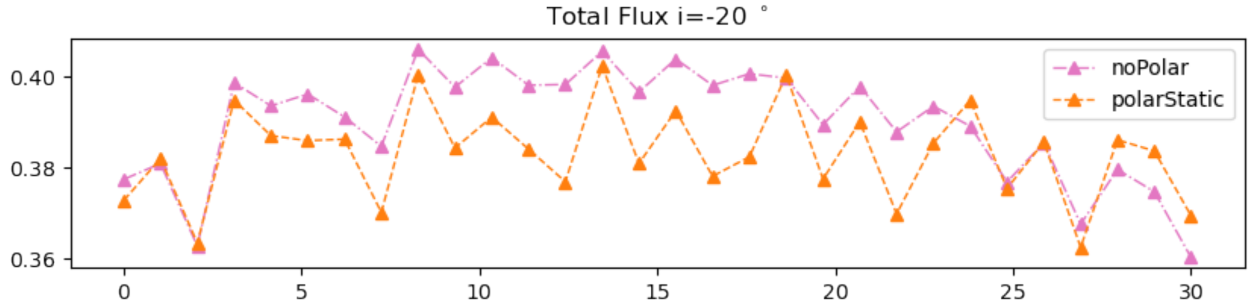


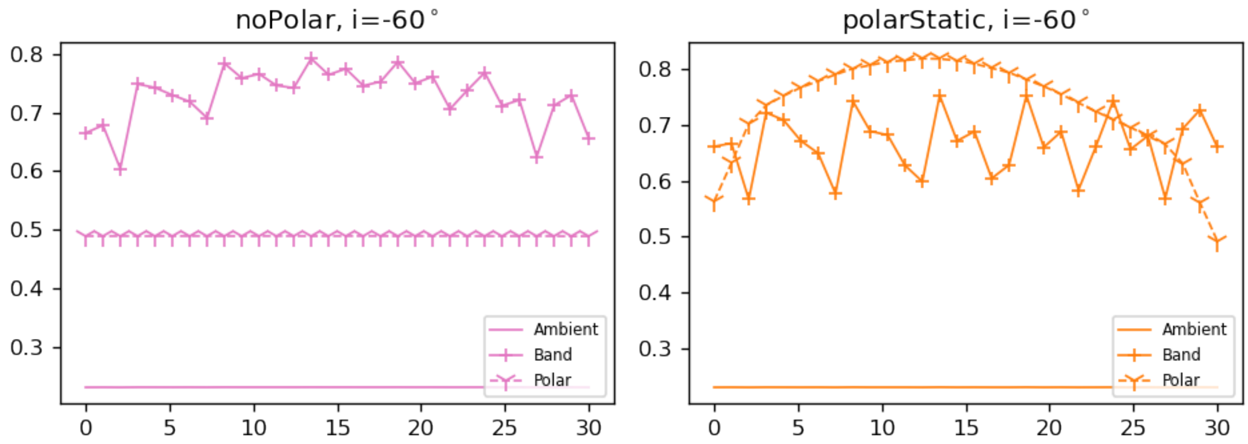
1E) Plot flux evolution by inclination, model class, and feature class

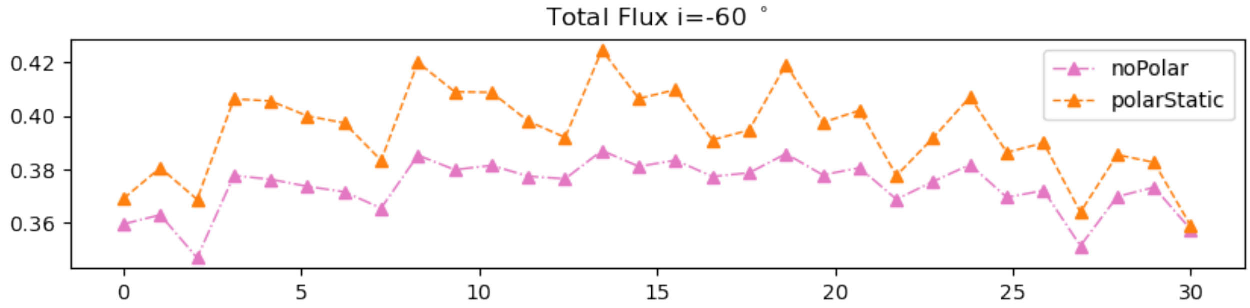
In [713…

```python
# ==============================================================
# [[Plot]]
# Plot flux by class-type (ambient, band, polar) by
# model class (polarStatic, polarDynamic, noPolar)
# for some inclination
# ==============================================================
plt.close()
colorList = {'polarStatic':'tab:orange', 'polarDynamic':'k', 'noPolar':'tal
for ia, iang in enumerate(['-20', '-60']):
    fig, axs = plt.subplots(1,2, dpi=120, figsize=(8,3))
    ticklist = {'b': '+', 'a':'', 'p':'1'}
    for i, model in enumerate(modelclasses):
        ambflux = fluxes[model][iang]['bytype']['amb']
        bandflux = fluxes[model][iang]['bytype']['band']
        polflux = fluxes[model][iang]['bytype']['pol']

        axs[i].plot(time_array, ambflux, ls='-', lw=1, c=colorList[model],
        axs[i].plot(time_array, bandflux, ls='-', lw=1, c=colorList[model]
        axs[i].plot(time_array, polflux, ls='--', lw=1, c=colorList[model]

        axs[i].set_title(model+r', i=%s$^\circ$'%iang)
        axs[i].legend(fontsize=7, loc='lower right')
        plt.tight_layout()

    plt.figure(figsize=(10, 2), dpi=100)
    plt.plot(time_array, fluxes['noPolar'][iang]['norm'], ls='-.', marker=
    plt.plot(time_array, fluxes['polarStatic'][iang]['norm'], ls='--', marl
    plt.title('Total Flux i=%s $^\circ$'%iang), plt.legend()
```









Step 2: Generate Synthetic Spectra Cube with Sonora

2A) Binning down Sonora Cloudless & Cloudy Model

In [718…

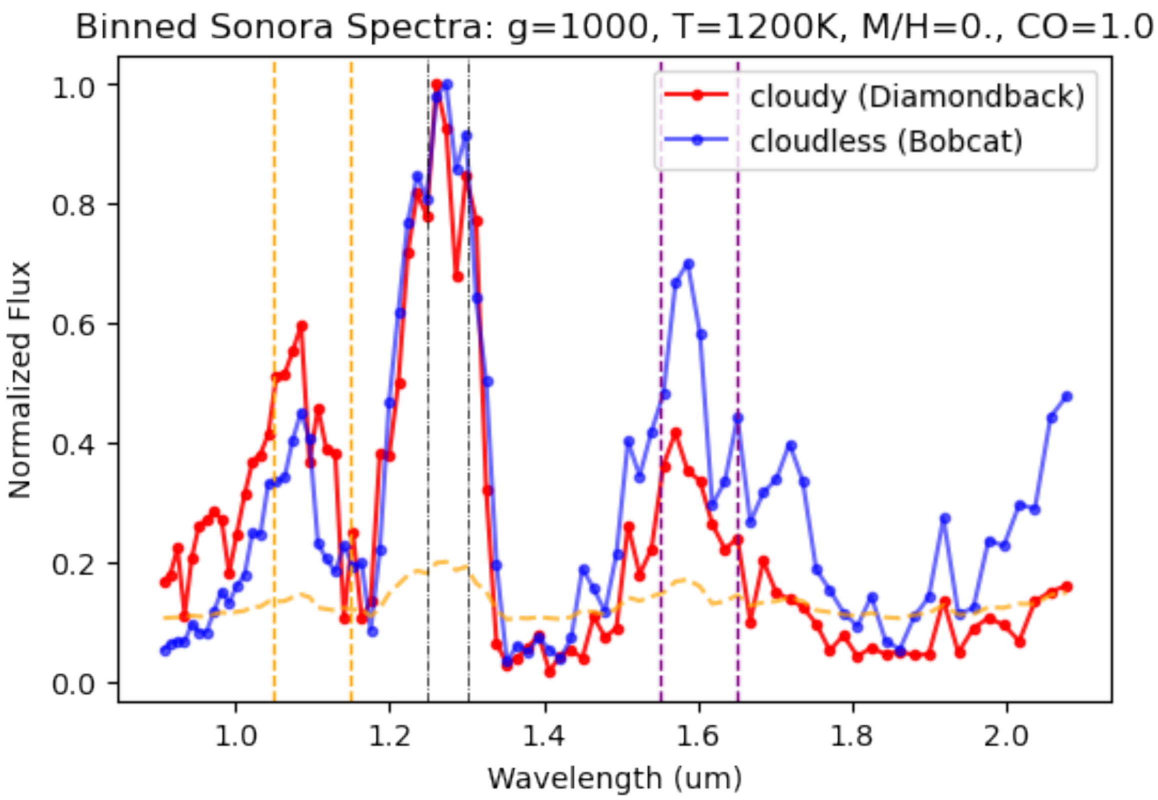
```
cloudlessPath ='~/Documents/GitHub/polar_vortice/data/spectras/bobcatCloudl
cloudyPath = '~/Documents/GitHub/polar_vortice/data/spectras/diamondbackClo

cloudlessSpec = pd.read_csv(cloudlessPath, sep='\s+', names = ['wave', 'flu
cloudlessSpec = cloudlessSpec.query(' 0.0 <= wave <= 10')
cloudlessSpec.sort_values('wave', inplace=True)

cloudySpec = pd.read_csv(cloudyPath, sep='\s+', names = ['wave', 'flux'], s
cloudySpec = cloudySpec.query(' 0.0 <= wave <= 10')
cloudySpec.sort_values('wave', inplace=True)

cloudlessSpec['norm'] = cloudlessSpec.flux / cloudlessSpec.flux.max()
cloudySpec['norm'] = cloudySpec.flux / cloudySpec.flux.max()

# set up color index center wavelength
lamJ, lamH = 1.10, 1.60
lamContinuum = 1.275
lamwidth = 0.05
```

In [855…

```
%matplotlib inline
# ===========================================================================
# [[Plot]]
# Plot the binned flux, and output file for cloudy AND cloudless spectra
# idea: try searching a bin-reserving flux
# idea: try NIRSPEC resolution, R=2000
# ===========================================================================
### Options
save = False ## output spectra files
# save = True
plot = False ## perform convolution of box-car smooth and plot spectras
# plot = True
read = not plot ## reading existing, smoothed spectra

# ===========================================================================
# Boxcar convolution for spectra binning
# ===========================================================================
if plot:
    plt.close(), plt.figure(figsize=(8,5))
    # print(cloudySpec.norm.shape)
    # plt.plot(cloudySpec.wave, cloudySpec.norm)
    binresolution = 2000
    convolve_res = 50

    wavebin1 = convolve(cloudySpec.wave, Box1DKernel(convolve_res))[::binre
    normbin1 = convolve(cloudySpec.norm, Box1DKernel(convolve_res))[::binre
    # print(wavebin1.shape)

    wavebin2 = convolve(cloudlessSpec.wave, Box1DKernel(convolve_res))[::bi
    normbin2 = convolve(cloudlessSpec.norm, Box1DKernel(convolve_res))[::bi
    # print(wavebin2.shape)
    normbin2_interp = np.interp(x=wavebin1, xp=wavebin2, fp=normbin2)

    plt.plot(wavebin1, normbin1, ls='-', marker='.', c='r', label='cloudy
    plt.plot(wavebin1, normbin2_interp, ls='-', marker='.', c='b', label='c
    plt.plot(wavebin1, 0.1 + 0.1*(normbin2_interp+normbin1)/2, ls='--', mar

    R=1.5/np.diff(wavebin1).mean()
    print('R=lamda/d_lamda=', R)
```

```
    # center wavelength near J and H band
    plt.axvline(lamJ-lamwidth, c='k', ls='--', lw=0.5), plt.axvline(lamJ+la
    plt.axvline(lamH-lamwidth, c='k', ls='-.', lw=0.5), plt.axvline(lamH+la
    plt.axvline(lamContinuum-lamwidth/2, c='k', ls='-.', lw=0.5), plt.axvli

    plt.xlim(0.90,2.090)
    plt.title('Binned Sonora Spectra: g=1000, T=1200K, M/H=0., CO=1.0')
    plt.xlabel('Wavelength (um)')
    plt.ylabel('Normalized Flux')
    plt.legend()

    # ==========================================================================
    ### Output binned spectras into text files
    # ==========================================================================

    pathCloudyOut = '~/Documents/GitHub/polar_vortice/data/spectras/binR=%
    cloudyOut = pd.DataFrame(np.transpose(np.array([wavebin1, normbin1])),
    cloudyOut = cloudyOut.query('0.90 <= wave <= 2.09')
    cloudyOut.flux = cloudyOut.flux/cloudyOut.flux.max()
    if save: cloudyOut.to_csv(pathCloudyOut, index=False)

    pathCloudlessOut = '~/Documents/GitHub/polar_vortice/data/spectras/binI
    cloudlessOut = pd.DataFrame(np.transpose(np.array([wavebin1, normbin2_
    cloudlessOut = cloudlessOut.query('0.90 <= wave <= 2.09')
    cloudlessOut.flux = cloudlessOut.flux/cloudlessOut.flux.max()
    if save: cloudlessOut.to_csv(pathCloudlessOut, index=False)

# ==============================================================================
# read from existing saved spectras
# ==============================================================================
if read:
    header = ['wave', 'flux']
    pathCloudyOutput = '~/Documents/GitHub/polar_vortice/data/spectras/binI
    cloudy = pd.read_csv(pathCloudyOutput, names=header, skiprows=1)

    pathCloudlessOutput = '~/Documents/GitHub/polar_vortice/data/spectras/I
    cloudless = pd.read_csv(pathCloudlessOutput, names=header, skiprows=1)

    plt.close(), plt.figure(figsize=(6,4), dpi=100)
    # print(cloudy.flux.shape)

    bandspec = cloudy.flux
    polarspec = cloudless.flux
    basespec = 0.1 + 0.1*cloudless.flux
    lam = cloudy.wave

    plt.plot(lam, bandspec, ls='-', marker='.', c='r', label='cloudy (Diam
    plt.plot(lam, polarspec, ls='-', marker='.', c='b', label='cloudless (I
    plt.plot(lam, basespec, ls='--', marker='', c='orange', alpha=0.7)

    R=1.5/np.diff(cloudy.wave).mean()
    print('R=lamda/d_lamda=', R)
    # center wavelength near J and H band
    plt.axvline(lamJ-lamwidth, c='orange', ls='--', lw=1), plt.axvline(lamJ
    plt.axvline(lamH-lamwidth, c='purple', ls='--', lw=1), plt.axvline(lamI
    plt.axvline(lamContinuum-lamwidth/2, c='k', ls='-.', lw=0.5), plt.axvli

    plt.title('Binned Sonora Spectra: g=1000, T=1200K, M/H=0., CO=1.0')
    plt.xlabel('Wavelength (um)')
    plt.ylabel('Normalized Flux')
    plt.legend()
```

```
R=lamda/d_lamda= 106.21354832727827
```



2B) Generate the spectral datacube

Method of generate spectral cube:

- Obtain flux by feature-class type (`ambient` , `band` , `polar`): These types will be a function of:
  - Latitudinal distribution of each feature-class
  - and the inclination angle which affect the projected area of each feature-class
- Generate spectras by feature-class type (`ambient` , `band` , `polar`):
  - Use a constant 0.2 flux for `ambient` .
  - Use Sonora-Bobcat Cloudless for `band` .
  - Use Sonora-Diamondback Cloudy for `polar` .
- Final expression: `Total` = norm_mean($\Sigma[$ $\mathrm{Frac.\ area}(i) \times (1 + \mathrm{Flux}(i)) \times \mathrm{Spectra}(i)]$)

In [856…

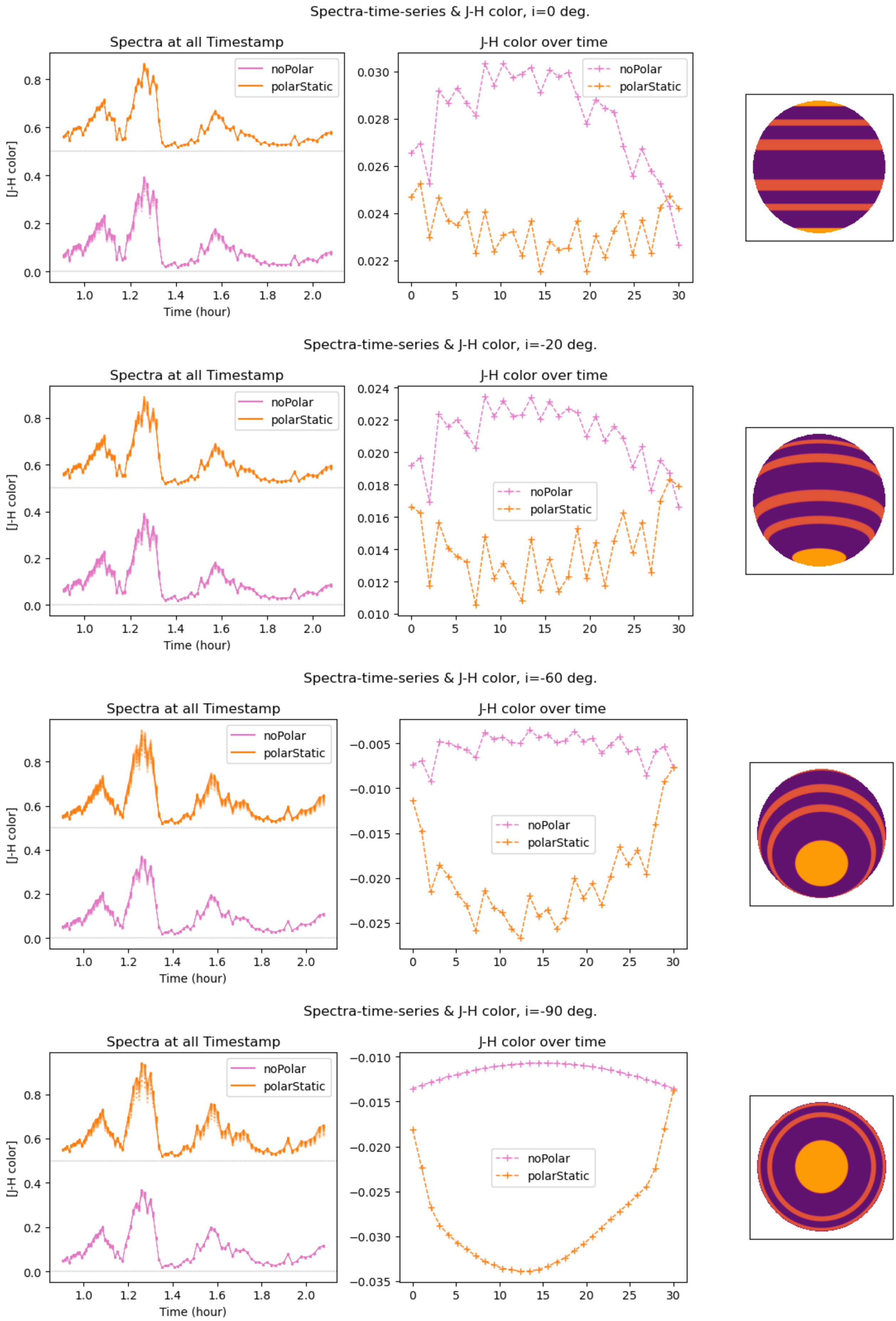
```
# ===========================================================================
# Define spectras of ambient (basespec),
# band (bandspec) and polar region (polarspec).
# ===========================================================================
#### Output options
save = False
# save = True
testPlot = True
# testPlot = False

bandspec = cloudy.flux
polarspec = cloudless.flux
basespec = 0.1 + 0.1*cloudless.flux
lam = cloudy.wave

# outline the wavelength band to calculate color
Jband_index = np.where(np.logical_and((lamJ-lamwidth)<=lam, lam<=(lamJ+lamw
Hband_index = np.where(np.logical_and((lamH-lamwidth)<=lam, lam<=(lamH+lamw
continuum_index = np.where(np.logical_and((lamContinuum-lamwidth/2)<=lam, l
# print(lam[Jband_index], lam[Hband_index])

# ===========================================================================
# Spectra cube configs
# ===========================================================================

spectral_cube = nested_dict()

colorList = {'polarStatic':'tab:orange', 'polarDynamic':'k', 'noPolar':'tab

for i, model in enumerate(modelclasses):
    for iang in incli:
        ambflux = fluxes[model][iang]['bytype']['amb']
        bandflux = fluxes[model][iang]['bytype']['band']
        polflux = fluxes[model][iang]['bytype']['pol']

        frac_amb = fluxes[model][iang]['fraction']['amb']
        frac_band = fluxes[model][iang]['fraction']['band']
        frac_pol = fluxes[model][iang]['fraction']['pol']

        # prepare spectral cube
        spectra_array, JHcolor_array = [], []
        continuum_array = []

        for i,t in enumerate(time_array):
            spectra_at_t = ambflux[i]*frac_amb*basespec + bandflux[i]*frac_
            color_at_t = np.mean(spectra_at_t[Jband_index]) - np.mean(spect
            continuumRatio = color_at_t/np.mean(spectra_at_t[continuum_inde

            spectra_array.append(spectra_at_t)
            JHcolor_array.append(color_at_t)
            continuum_array.append(continuumRatio)

        spectral_cube[model][iang]['spectra'] = spectra_array
        spectral_cube[model][iang]['JH_color'] = JHcolor_array
        spectral_cube[model][iang]['continuumRatio'] = continuum_array
```

2C) Spectra cube plot: all spectras through time, and J-H color by model class

- J color: $\text{mean}(\lambda = 1.05 - 1.15\mu m)$
- H color: $\text{mean}(\lambda = 1.55 - 1.65\mu m)$
- Continuum: $\text{mean}(\lambda = 1.250 - 1.300\mu m)$

In [857…

```
# =============================================================================
# [[Plot]]
# Spectra cube test plots for two inclination, by model class
# 1) spectras at all timestamps, two model classes: blue and red
# 2) j-h color evolution by time, two model classes: blue and red
# =============================================================================
plt.close()
for iang in ['0', '-20', '-60', '-90']:
    f, axs = plt.subplots(1, 3, figsize=(11,4), dpi=100, gridspec_kw={'widt
    cadence = (t1-t0)/no_frame
    f.suptitle('Spectra-time-series & J-H color, i=%s deg.'%iang)

    axs[0].set_title('Spectra at all Timestamp')
    axs[1].set_title('J-H color over time')

    offset = [0., 0.5]
    for i, model in enumerate(modelclasses):
        linestyle = next(linestyleRandomList)
        for t in range(len(time_array)):
            specnum = spectral_cube[model][iang]['spectra'][t]
            axs[0].plot(lam, specnum +offset[i], ls=linestyle, lw=0.2, marl

        axs[0].plot([], c=colorList[model], label=model)
        axs[0].legend()
        axs[0].set_xlabel('Wavelength (um)')
        axs[0].set_ylabel('Intensity')
        axs[0].axhline(offset[i], ls='--', lw=0.2, c='k')

        colorArray = spectral_cube[model][iang]['JH_color']
        axs[1].plot(time_array, colorArray, c=colorList[model], ls='--', lv
        axs[1].legend()
        axs[0].set_ylabel('[J-H color]')
        axs[0].set_xlabel('Time (hour)')

        axs[2].imshow(specmasks[model][iang][0], cmap='inferno', vmax=0.85
        axs[2].set_xticks([]), axs[2].set_yticks([])

    plt.tight_layout()
```

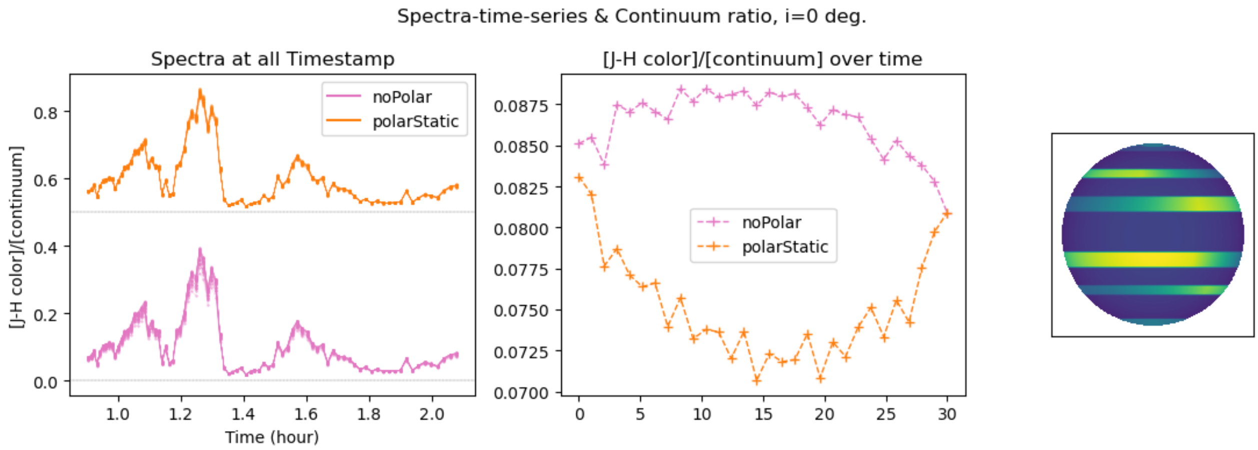


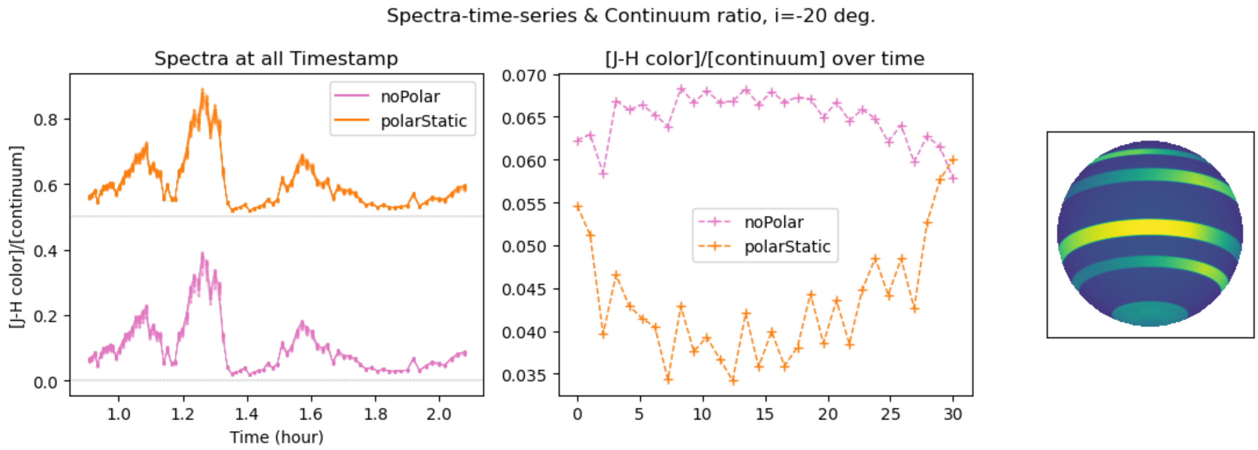
2D) Spectra cube plot: all spectras through time, and [J-H]/continuum by model class

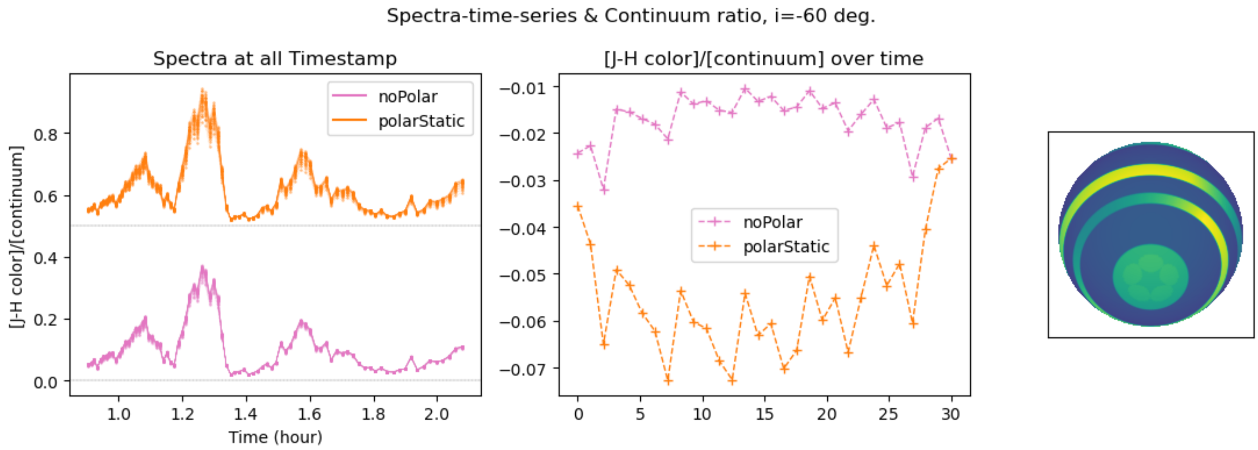
- J color: $\text{mean}(\lambda = 1.05 - 1.15\mu m)$
- H color: $\text{mean}(\lambda = 1.55 - 1.65\mu m)$
- Continuum: $\text{mean}(\lambda = 1.250 - 1.300\mu m)$

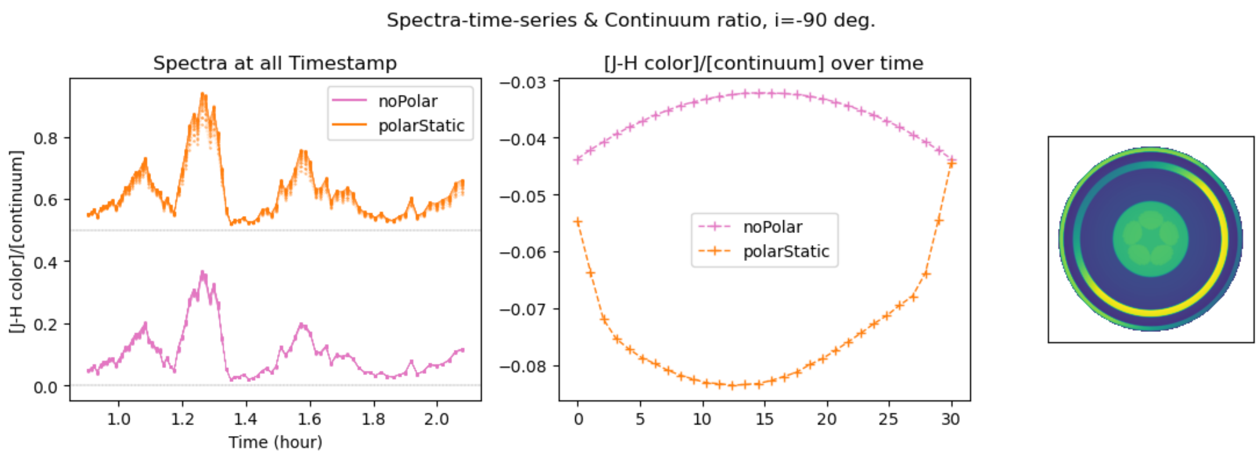
In [858…

```
# =========================================================================
# [[Plot]]
# Similar to the above plot, but use [j-h color]/[continuum] (continuum at
# Spectra cube test plots for two inclination, by model class
# 1) spectras at all timestamps, two model classes: blue and red
# 2) [j-h color]/[continuum] evolution by time, two model classes: blue and
# =========================================================================
plt.close()
for iang in ['0', '-20', '-60', '-90']:
    f, axs = plt.subplots(1, 3, figsize=(11,4), dpi=100, gridspec_kw={'widt
    cadence = (t1-t0)/no_frame
    f.suptitle('Spectra-time-series & Continuum ratio, i=%s deg.'%iang)

    axs[0].set_title('Spectra at all Timestamp')
    axs[1].set_title('[J-H color]/[continuum] over time')

    offset = [0., 0.5]
    for i, model in enumerate(modelclasses):
        linestyle = next(linestyleRandomList)
        for t in range(len(time_array)):
            specnum = spectral_cube[model][iang]['spectra'][t]
            axs[0].plot(lam, specnum +offset[i], ls=linestyle, lw=0.2, marl

        axs[0].plot([], c=colorList[model], label=model)
        axs[0].legend()
        axs[0].set_xlabel('Wavelength (um)')
        axs[0].set_ylabel('Intensity')
        axs[0].axhline(offset[i], ls='--', lw=0.2, c='k')

        continuumArray = spectral_cube[model][iang]['continuumRatio']
        axs[1].plot(time_array, continuumArray, c=colorList[model], ls='--
        axs[1].legend()
        axs[0].set_ylabel('[J-H color]/[continuum]')
        axs[0].set_xlabel('Time (hour)')

        axs[2].imshow(images[model][iang][0][0])
        axs[2].set_xticks([]), axs[2].set_yticks([])

    plt.tight_layout()
```







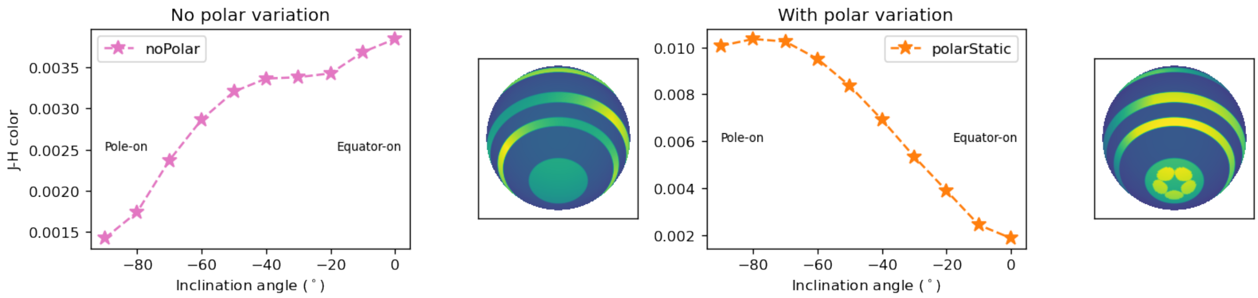


2E) Plot amplitude of J-H color time-variation as function of inclination for model class

Definition of variation amplitude: $\mathrm{Amp}(x) = (\max(x) - \min(x))/2$

In [859…

```
# ==========================================================================
# [[Plot]]
# Quantify the variability amplitude of (1) and (2) as a function of incli
# by model class
# (1) J-H color, (2) [J-H color]/[continuum]
# ==========================================================================
amplitudes = nested_dict()
for model in modelclasses:
    emp1, emp2 = [], []
    # print(model)
    for iang in incli:
        continuumArray = np.array(spectral_cube[model][iang]['continuumRat
        colorArray = np.array(spectral_cube[model][iang]['JH_color'])

        ampColor = (colorArray.max()-colorArray.min())/2
        # print('i=', iang, colorArray.max(), colorArray.min(), ampColor)
        ampContinuumRatio = (continuumArray.max()-continuumArray.min())/2

        emp1.append([int(iang), ampColor])
        emp2.append([int(iang), ampContinuumRatio])

    amplitudes[model]['JH_color'] = np.array(emp1)

    amplitudes[model]['continuumRatio'] = np.array(emp2)
    # print('================')
    # print(amplitudes[model]['JH_color'])

# ===========================================================
## The plot: J-H color
# ===========================================================
plt.close()
titles = ['No polar variation', 'With polar variation']
# for typename in ['JH_color', 'continuumRatio']:
for typename in ['JH_color']:
    fig, axs = plt.subplots(1,4, figsize=(12,3), dpi=120, gridspec_kw={'wi
    # fig.suptitle('%s with inclination i'%typename)
    for i, model in enumerate(modelclasses):
        x = amplitudes[model][typename][:,0]
        y = amplitudes[model][typename][:,1]
        axs[2*i].plot(x, y, ls='--', marker='*', c=colorList[model], ms=10
        axs[2*i].legend()
        axs[2*i].set_xlabel(r'Inclination angle ($^\circ$)')
        axs[2*i].set_title(titles[i])
    axs[0].text(-90, 0.0025, 'Pole-on', fontsize=8), axs[0].text(-18, 0.002
    axs[2].text(-90, 0.006, 'Pole-on', fontsize=8), axs[2].text(-18, 0.006
    axs[0].set_ylabel('J-H color')

    time1 = 2
    axs[1].imshow(images['noPolar']['-50'][time1][0])
    axs[1].set_xticks([]), axs[1].set_yticks([])
    time2 = 3
    axs[3].imshow(images['polarStatic']['-50'][time2][0])
    axs[3].set_xticks([]), axs[3].set_yticks([])
    plt.tight_layout()
```



No polar variation
noPolar
J-H color
0.0035
0.0030
0.0025
0.0020
0.0015
Pole-on
Equator-on
−80
−60
−40
−20
0
Inclination angle (°)
With polar variation
polarStatic
0.010
0.008
0.006
0.004
0.002
Pole-on
Equator-on
−80
−60
−40
−20
0
Inclination angle (°)