VIMS_alpOri271I_analysis_v2

October 10, 2023

0.1 VIMS_alpOri271I_analysis_v2.ipynb

Richard G. French, Wellesley College

This Jupyter notebook contains an analysis of the Cassini VIMS observations of the alpOri271 occultation, using data provided by An Foster and Phil Nicholson.

The following steps are performed:

1) Perform model fits to the observations: isothermal fits isothermal + absorption model based on Goody random band model
2) Overplot Phil Nicholson's model predictions
3) Perform numerical inversion of observations to derive T(P) profile
4) Compare retrieved T(P) with CIRS observations
5) Construct synthetic lightcurve based on CIRS T(P) and compare with observations
6) Perform end-to-end test of inverting the synthetic lightcurve to show that it matches input T(P)

This is a non-interactive notebook - simply run all cells All required Python packages and data files are defined in the first code cell All individual functions and procedures are in separate cells, with a description of the purpose and method of each. The final cells perform the numbered steps above.

Revisions:

v2:

```
2023 Oct 10 - rfrench - Move all routines to top, retain tests for documentation, but start ne
```

Liens:

1. g(r) not implemented - assumed to be constant over range of inversion (easy to fix)

[1]:
```
# Load Python packages (ALL required dependencies are collected here)

from astropy import units as u
import astropy.constants as const
import csv
from lmfit import Parameters,minimize, fit_report
import matplotlib.pyplot as plt
import numpy as np
import os
from scipy.integrate import odeint
```

```python
from scipy.interpolate import interp1d
from scipy.io import readsav # for IDL savefiles
from scipy.optimize import fsolve
from scipy.special import erfc # erfc for isothermal cap
import spiceypy as spice

# specify paths to required data files, and define VIMS dictionary with event␣
↪information

path2inputfiles = './' # modify this to point to directory containing datafiles
path2kernels = './' # modify this to point to directory containing leapseconds␣
↪kernel file
path2outputfiles = './output/' # modify this to point to directory containing␣
↪datafiles
if not os.path.exists(path2outputfiles): # create the output directory if it␣
↪doesn't exist
   os.makedirs(path2outputfiles)

VIMS_observed_lightcurve = 'lightcurvewithgoodbaseline.csv' # from An Foster
VIMS_isothermal_model = 'nicholso-isothermal-alpori271.out' # Phil Nicholson␣
↪isothermal model
VIMS_isothermal_metadata = 'nicholso-isothermal-metadata-alpori271.out' # ...␣
↪and metadata for documentation
tlsfile = 'naif0012.tls'

CIRS_TPprofiles = 'globaltemp.sav' # IDL savefile containing CIRS T(P) profiles␣
↪as function of JD and latitude
# (Obtained from PDS Atmos node? not sure of origin)

# define VIMS dictionary containing essential event geometry and information

VIMS = {
    'path2inputfiles':path2inputfiles,
    'path2outputfiles':path2outputfiles,
    'path2kernels':path2kernels,
    'VIMS_observed_lightcurve':VIMS_observed_lightcurve,
    'dtsec':1.68, # from An Foster
    'VIMS_isothermal_model':VIMS_isothermal_model,
    'VIMS_isothermal_metadata':VIMS_isothermal_metadata,
    'tlsfile':tlsfile,
    'CIRS_TPprofiles':CIRS_TPprofiles,
    'event':'VIMS alpOri271I',
    'UTC':"2017-116T21:20", #approximate only - used only to determine JD of␣
↪event for CIRS T(P)
    'g_ms2':11.90*u.m/u.s**2, # from PDN
    'H_km':44.*u.km, # from An
```

```
    'half-light':930, # frame number, from metadata file...
    't_cube':1.68*u.s,
    't_pixel':0.021*u.s,
    'rc':54914.3*u.km,
    'vperp':-2.311*u.km/u.s,
    'lat_c':-74.44*u.deg,
    'lat_g':-77.23*u.deg,
    'time':1566.6*u.s,
    'range':6.8726e+05*u.km,
    'alpha':-66.36*u.deg,
    'r_curv':66043.5*u.km,
    'mu':2.2* u.g/u.mol, # from PDN
    'RSTP': 129.e-6 # refractivity at STP
}
```

1 All defs loaded first

[2]:

```
# non-dimensional bending angle (theta_hat = 1 at half-light)
def ftheta_hat(theta_hat,x):
    return np.log(theta_hat)+theta_hat - 1. - x
```

[3]:

```
# normalized flux as function of non-dimensional time \
# (that_vals in scale-heights in observer plane) for isothermal atmosphere
def get_phi_vals(that_vals,xtol=1e-10):
    phi_vals = np.zeros(len(that_vals))
    for i,xval in enumerate(that_vals):
        if xval <= 0:
            theta_hat0 = np.exp(xval)
        elif xval <= 4:
            theta_hat0 = 2.
        elif xval <= 20:
            theta_hat0 = 20.
        else:
            theta_hat0 = 1/xval
        theta_hat_val = fsolve(ftheta_hat,theta_hat0,xval,xtol=xtol)
        phi_iso = 1./(1.+theta_hat_val)
        phi_vals[i] = phi_iso
    return phi_vals
```

[4]:

```
# normalized flux as function of non-dimensional time \
# (that_vals in scale-heights in observer plane)
# for isothermal atmosphere with optional random band model opacity
# tau_hl = half-light slant-path optical depth
# tau_gamma = random band model strength parameter
```

```python
def get_phi_tau_vals(that_vals,tau_hl,tau_gamma,IsothermalOnly=False,xtol=1e-10):
    phi_tau_vals = np.zeros(len(that_vals))
    for i,xval in enumerate(that_vals):
        if xval <= 0:
            theta_hat0 = np.exp(xval)
        elif xval <= 4:
            theta_hat0 = 2.
        elif xval <= 20:
            theta_hat0 = 20.
        else:
            theta_hat0 = 1/xval
        theta_hat_val = fsolve(ftheta_hat,theta_hat0,xval,xtol=xtol)
        phi_iso = 1./(1.+theta_hat_val)
        if IsothermalOnly:
            phi_tau_vals[i] = phi_iso
        else:
# Random band model
            tau_gamma = max([0,tau_gamma]) # to avoid negative square root
            tau_exponent = -tau_hl * np.sqrt((1+tau_gamma)/(1+tau_gamma*theta_hat_val)) * theta_hat_val
            # suppress exponent overflow message
            tau_exponent = np.clip(tau_exponent, -709.78, 709.78)
            tau_factor = np.exp(tau_exponent)
            refrac_factor = 1
            phi_tau_vals[i] = phi_iso * tau_factor * refrac_factor
    return phi_tau_vals
```

[5]:

```python
# compute isothermal model and residuals to observations, used by lmfit

def fit_iso_lmfit(params,tsec,v_perp_kms,data,\
    returnModel=False,Verbose=False):
    Hkm = params['H_km']
    t_hl = params['t_hl']
    that_vals = -v_perp_kms * (tsec - t_hl)/Hkm
    neg_resid = []
    y_bkg = params['y_bkg']
    y_scale = params['y_scale']
    phi_vals = get_phi_vals(that_vals,xtol=1e-10)
    model = y_bkg + y_scale*phi_vals
    if returnModel:
        this_neg_resid = model
    else:
        this_neg_resid = model - data
    if len(neg_resid) == 0:
        neg_resid = this_neg_resid
    else:
```

```
        neg_resid = np.concatenate((neg_resid,this_neg_resid))
    return neg_resid
```

[6]:
```
# compute isothermal + opacity model and residuals to observations, used by␣
↪lmfit

def fit_iso_tau_lmfit(params,tsec,v_perp_kms,tau_hl,tau_gamma,data,\
    returnModel=False,Verbose=False,IsothermalOnly=False):
    Hkm = params['H_km']
    t_hl = params['t_hl']
    that_vals = -v_perp_kms * (tsec - t_hl)/Hkm

    y_bkg = params['y_bkg']
    y_scale = params['y_scale']
    tau_hl = params['tau_hl']
    tau_gamma = params['tau_gamma']

    phi_vals = get_phi_tau_vals(that_vals,tau_hl,tau_gamma,\
            IsothermalOnly=IsothermalOnly,xtol=1e-10)

    neg_resid = []
    model = y_bkg + y_scale*phi_vals
    if returnModel:
        this_neg_resid = model
    else:
        this_neg_resid = model - data
    if len(neg_resid) == 0:
        neg_resid = this_neg_resid
    else:
        neg_resid = np.concatenate((neg_resid,this_neg_resid))
    return neg_resid
```

[7]:
```
# perform isothermal fit to raw lightcurve, using minimize()

def fit_iso_to_observations(tsec,v_perp_kms,data,params,\
    max_nfev=50,fit_xtol=1.e-8,Verbose=False):
    keywords_dict = {"returnModel":False}
    fitted_params = minimize(fit_iso_lmfit,params,\
            args=(tsec,v_perp_kms,data),kws=keywords_dict,\
            method='least_squares',xtol=fit_xtol,max_nfev =␣
↪max_nfev,nan_policy='omit')
    yfit = fit_iso_lmfit(fitted_params.
↪params,tsec,v_perp_kms,data,returnModel=True,Verbose=False)
    return yfit,fitted_params
```

[8]:

```
# perform isothermal fit + opacity to raw lightcurve, using minimize()

def fit_iso_tau_to_observations(tsec,v_perp_kms,tau_hl,tau_gamma,data,params,\
    max_nfev=50,fit_xtol=1.e-8,Verbose=False):
    keywords_dict = {"returnModel":False}
    fitted_params = minimize(fit_iso_tau_lmfit,params,\
            args=(tsec,v_perp_kms,tau_hl,tau_gamma,data),kws=keywords_dict,\
            method='least_squares',xtol=fit_xtol,max_nfev =␣
 ↪max_nfev,nan_policy='omit')
    yfit = fit_iso_tau_lmfit(fitted_params.
 ↪params,tsec,v_perp_kms,tau_hl,tau_gamma,data,\
            returnModel=True,Verbose=False)
    return yfit,fitted_params
```

[9]:

```
# Read and plot the data...

def plot_VIMS_data(infile,figfile_rawdata):
    with open(infile,newline='') as csvfile:
        reader = csv.reader(csvfile)
        counts = np.array(list(reader),dtype='float')[:,0]

# three-panel plot of raw data, saved as figfile_rawdata
# log scale
# linear scale
# zoomed linear cale

    fig1=plt.figure(1,figsize=(12,14)) # open up figure
    plt.rcParams.update({'font.size': 18})

    plt.subplot(3,1,1)
    plt.plot(counts)
    #plt.xlabel('Frame number')
    plt.ylabel('DN')
    plt.ylim(0.1,max(counts)*1.5)
    #plt.ylim(max(min([counts,.1]),max(counts)*1.5))
    plt.yscale('log')
    plt.title(VIMS['event']+' raw observations')

    plt.subplot(3,1,2)

    plt.plot(counts)
    #plt.xlabel('Frame number')
    plt.ylabel('DN')
    plt.ylim(0.0,200)
    plt.yscale('linear')

    plt.subplot(3,1,3)
```

```python
    plt.plot(counts)
    plt.xlabel('Frame number')
    plt.ylabel('DN')
    plt.xlim(800,1450)
    plt.ylim(-10,200)
    plt.yscale('linear')

    plt.show()
    plt.savefig(figfile_rawdata)
    print("Saved",figfile_rawdata,"\n")

    return counts # for later use
```

[10]:

```python
#isothermal fits to the data
def plot_isofits(VIMS,counts,figfile_isofits):
    params = Parameters()
# specify initial guesses to fitted parameters
    Hkm = VIMS['H_km'].value
    t_hl = VIMS['half-light']*VIMS['t_cube'].value
    y_bkg = 0.
    y_scale = max(counts)

    vary_H_km = True
    vary_t_hl = True
    vary_y_bkg = False
    vary_y_scale = True

    params.add('H_km',value=Hkm ,vary=vary_H_km)
    params.add('t_hl',value=t_hl,vary=vary_t_hl)
    params.add('y_bkg',value=y_bkg,vary=vary_y_bkg)
    params.add('y_scale',value=y_scale,vary=vary_y_scale)

    frame = np.linspace(0,len(counts)-1,len(counts))

    L = np.where((frame >= 800) & (frame <= 1200)) # zoom specification

    tsec = frame[L]*VIMS['t_cube'] # time in sec from start of data
    data = counts[L] # the observed lightcurve

    yfit,fitted_params = fit_iso_to_observations(tsec.value,VIMS['vperp'].
 ↪value,data,params,\
        max_nfev=50,fit_xtol=1.e-6,Verbose=True)

    print('--------------Isothermal model------------------')
    print('Parameter                   Value          Stderr')
```

```
for name, param in fitted_params.params.items():
    try:
        print('{:18s} {:14.4f} {:14.4f}'.format(name, param.value, param.
↪stderr))
    except:
        pass

model_iso = fit_iso_lmfit(fitted_params.params,tsec.value,VIMS['vperp'].
↪value,data,\
    returnModel=True,Verbose=False)

# Plot isothermal fit, isothermal + absorption fits, and PDN model

fig2=plt.figure(2,figsize=(12,8)); # open up figure
plt.rcParams.update({'font.size': 18})

plt.plot(tsec,data,label='Obs')

# overplot PDN model

result = np.genfromtxt( VIMS['path2inputfiles'] +␣
↪VIMS['VIMS_isothermal_model'],
                        dtype='float',skip_header =␣
↪12,usecols=[6,7],names=['tsec','phi'])
dt = result['tsec']+fitted_params.params['t_hl']
phi = result['phi']
label='H = '+f'{VIMS["H_km"].value:.2f}'+' km (PDN)'
Lfit = np.where((dt >0 ) & (dt < max(tsec.value)))
plt.plot(dt[Lfit],fitted_params.params['y_scale']*phi[Lfit],label=label)

# plot isothermal fit

label='H = '+f'{fitted_params.params["H_km"].value:.2f}'+' km (fit)'
plt.plot(tsec,model_iso,label=label)

# now perform fit with band model

params = Parameters()

# specify initial guesses

Hkm = VIMS['H_km'].value
t_hl = VIMS['half-light']*VIMS['t_cube'].value
y_bkg = 0.
y_scale = max(counts)
tau_hl = 0.2
```

```
# specify set of fixed values of tau_gamma
tau_gammas = [0,100]

vary_t_hl = True
vary_y_bkg = False
vary_H_km = True
vary_y_scale = True
vary_tau_hl = True
vary_tau_gamma = False

for imodel,tau_gamma in enumerate(tau_gammas):
    params.add('H_km',value=Hkm ,vary=vary_H_km)
    params.add('t_hl',value=t_hl,vary=vary_t_hl)
    params.add('y_bkg',value=y_bkg,vary=vary_y_bkg)
    params.add('y_scale',value=y_scale,vary=vary_y_scale)
    params.add('tau_hl',value=tau_hl,vary=vary_tau_hl)
    params.add('tau_gamma',value=tau_gamma,vary=vary_tau_gamma)

    yfit,fitted_params_tau = fit_iso_tau_to_observations(tsec.
↪value,VIMS['vperp'].value,
        tau_hl,tau_gamma,data,params,max_nfev=50,fit_xtol=1.
↪e-6,Verbose=True)

    print('-------------------Band model␣
↪'+str(imodel+1)+'-----------------')
    print('Parameter                  Value          Stderr')

    for name, param in fitted_params_tau.params.items():
        try:
            print('{:18s} {:14.4f} {:14.4f}'.format(name, param.value,␣
↪param.stderr))
        except:
            pass

    model_tau = fit_iso_tau_lmfit(fitted_params_tau.params,tsec.
↪value,VIMS['vperp'].value,
        tau_hl,tau_gamma,data,returnModel=True,Verbose=False)

    label='H = '+f'{fitted_params_tau.params["H_km"].value:.2f}'+' km '+\
        r'$\tau_{1/2}=$' +f'{fitted_params_tau.params["tau_hl"].value:.2f}'␣
↪+\
        ', '+r"$\gamma=$" + str(tau_gamma)
    plt.plot(tsec.value,model_tau,label=label)

plt.xlabel('t (sec)')
plt.ylabel('DN')
```

```
    plt.legend()
    plt.title(VIMS['event'])
    plt.show();
    plt.savefig(figfile_isofits);
    print("Saved",figfile_isofits,"\n")

    return tsec,data,model_iso, fitted_params, model_tau, fitted_params_tau
```

[11]:
```
#isothermal fits with CIRS synthetic lightcurve overplotted
def plot_isofits_CIRS(VIMS,tsec_CIRS,flux_CIRS,counts,figfile_isofits_CIRS):
    params = Parameters()
# specify initial guesses to fitted parameters
    Hkm = VIMS['H_km'].value
    t_hl = VIMS['half-light']*VIMS['t_cube'].value
    y_bkg = 0.
    y_scale = max(counts)

    vary_H_km = True
    vary_t_hl = True
    vary_y_bkg = False
    vary_y_scale = True

    params.add('H_km',value=Hkm ,vary=vary_H_km)
    params.add('t_hl',value=t_hl,vary=vary_t_hl)
    params.add('y_bkg',value=y_bkg,vary=vary_y_bkg)
    params.add('y_scale',value=y_scale,vary=vary_y_scale)

    frame = np.linspace(0,len(counts)-1,len(counts))

    L = np.where((frame >= 800) & (frame <= 1200)) # zoom specification

    tsec = frame[L]*VIMS['t_cube'] # time in sec from start of data
    data = counts[L] # the observed lightcurve

    yfit,fitted_params = fit_iso_to_observations(tsec.value,VIMS['vperp'].
 ↪value,data,params,\
        max_nfev=50,fit_xtol=1.e-6,Verbose=True)

    print('--------------Isothermal model------------------')
    print('Parameter                  Value          Stderr')

    for name, param in fitted_params.params.items():
        try:
            print('{:18s} {:14.4f} {:14.4f}'.format(name, param.value, param.
 ↪stderr))
        except:
            pass
```

```
model_iso = fit_iso_lmfit(fitted_params.params,tsec.value,VIMS['vperp'].
↪value,data,\
    returnModel=True,Verbose=False)

# Plot isothermal fit, isothermal + absorption fits, and PDN model

fig2=plt.figure(2,figsize=(12,8)); # open up figure
plt.rcParams.update({'font.size': 18})

plt.plot(tsec,data,label='Obs')

# overplot PDN model

result = np.genfromtxt( VIMS['path2inputfiles'] +␣
↪VIMS['VIMS_isothermal_model'],
                         dtype='float',skip_header =␣
↪12,usecols=[6,7],names=['tsec','phi'])
dt = result['tsec']+fitted_params.params['t_hl']
phi = result['phi']
label='H = '+f'{VIMS["H_km"].value:.2f}'+' km (PDN)'
Lfit = np.where((dt >0 ) & (dt < max(tsec.value)))
plt.plot(dt[Lfit],fitted_params.params['y_scale']*phi[Lfit],label=label)

# plot isothermal fit

label='H = '+f'{fitted_params.params["H_km"].value:.2f}'+' km (fit)'
plt.plot(tsec,model_iso,label=label)

# now perform fit with band model

params = Parameters()

# specify initial guesses

Hkm = VIMS['H_km'].value
t_hl = VIMS['half-light']*VIMS['t_cube'].value
y_bkg = 0.
y_scale = max(counts)
tau_hl = 0.2

# specify set of fixed values of tau_gamma
tau_gammas = [0,100]

vary_t_hl = True
vary_y_bkg = False
vary_H_km = True
```

```
    vary_y_scale = True
    vary_tau_hl = True
    vary_tau_gamma = False

    for imodel,tau_gamma in enumerate(tau_gammas):
        params.add('H_km',value=Hkm ,vary=vary_H_km)
        params.add('t_hl',value=t_hl,vary=vary_t_hl)
        params.add('y_bkg',value=y_bkg,vary=vary_y_bkg)
        params.add('y_scale',value=y_scale,vary=vary_y_scale)
        params.add('tau_hl',value=tau_hl,vary=vary_tau_hl)
        params.add('tau_gamma',value=tau_gamma,vary=vary_tau_gamma)

        yfit,fitted_params_tau = fit_iso_tau_to_observations(tsec.
↪value,VIMS['vperp'].value,
            tau_hl,tau_gamma,data,params,max_nfev=50,fit_xtol=1.
↪e-6,Verbose=True)

        print('-------------------Band model␣
↪'+str(imodel+1)+'-----------------')
        print('Parameter                  Value          Stderr')

        for name, param in fitted_params_tau.params.items():
            try:
                print('{:18s} {:14.4f} {:14.4f}'.format(name, param.value,␣
↪param.stderr))
            except:
                pass

        model_tau = fit_iso_tau_lmfit(fitted_params_tau.params,tsec.
↪value,VIMS['vperp'].value,
            tau_hl,tau_gamma,data,returnModel=True,Verbose=False)

        label='H = '+f'{fitted_params_tau.params["H_km"].value:.2f}'+' km '+\
            r'$\tau_{1/2}=$' +f'{fitted_params_tau.params["tau_hl"].value:.2f}'␣
↪+\
            ', '+r"$\gamma=$" + str(tau_gamma)
        plt.plot(tsec.value,model_tau,label=label)
    t_CIRS = tsec_CIRS.value+fitted_params.params['t_hl']
    model_CIRS = fitted_params.params['y_scale']*flux_CIRS + fitted_params.
↪params['y_bkg']
    plt.plot(t_CIRS,model_CIRS,label='CIRS T(P)',linewidth=4)
    plt.xlabel('t (sec)')
    plt.ylabel('DN')
    plt.legend()
    plt.xlim(np.min(tsec.value),np.max(tsec.value))
    plt.title(VIMS['event'])
```

```
    plt.show();
    plt.savefig(figfile_isofits_CIRS);
    print("Saved",figfile_isofits_CIRS,"\n")
    return #tsec,model_iso, fitted_params, model_tau, fitted_params_tau
```

[12]:
```
#rebin normalized input lightcurve into equal altitude bins
def alt_bin(that_vals,flux,dh_bin,ha,D,tau_hl=0,Verbose=False):

    dt_av = that_vals[1] - that_vals[0]
    if Verbose == True:
        print('alt_bin debug:')
        print('that_vals[0:5]=',that_vals[0:5])
        print('in alt_bin: dt_av=',dt_av)
        print('that_vals=',that_vals)
        print('flux:',flux)
        print('dt_av:',dt_av)
    i = 0
    j = 0
    i_tot = 0
    t_start = dt_av/2.

    MAXSTORE = 500000 # modify this at some point to use append()
    profile_flux = np.zeros(MAXSTORE)
    profile_time = np.zeros(MAXSTORE)
    profile_dtheta = np.zeros(MAXSTORE)

    t_skip = that_vals[0] - dt_av # confirmed that this and t_start give proper
                                  # alignment of profile_flux and profile_time
    t_j = t_start + dt_av

    while(j < np.size(flux)):
        if Verbose == True:
            print('\nj,np.size(flux)=',j,np.size(flux))
        dt_phi = (t_j - t_start) * flux[j]
        d_sum = dt_phi
        first = True
        if Verbose== True:
            print('first,TP1: d_sum,dt_phi,dh_bin=',first,d_sum,dt_phi,dh_bin)
        if d_sum <= dh_bin: # altitude bin not yet full
            first = False
            while d_sum < dh_bin:
                t_j += dt_av
                j+= 1
                if Verbose == True:
                    print('TP2: t_j,j=',t_j,j)
                if j >= np.size(flux):
                    break
```

```
                dt_phi = flux[j] * dt_av
                d_sum += dt_phi
        if j >= np.size(flux):
            break
        dxs = d_sum - dh_bin
        ddt = dxs/flux[j]
        if first == True:
            ddt = (t_j - t_start)*dxs/dt_phi
        t_last = t_j - ddt
        delta_t = np.abs(t_last - t_start)
        profile_flux[i] = dh_bin/delta_t
    # apply tau_hl if non-zero
        this_alt = ha-i*dh_bin - dh_bin/2 # to agree with below
        if tau_hl != 0:
            this_tau = tau_hl * np.exp(-this_alt)
            profile_flux[i] *= np.exp(-this_tau)
        profile_dtheta[i] = 1/profile_flux[i] - 1
        t_start = t_last
        t_now = t_skip + t_last
        profile_time[i] = t_now
        i_tot = i
        if Verbose == True:# and flux[j] < 0.99:
            print('output␣
 ↪i,profile_flux[i],profile_time[i]',i,profile_flux[i],profile_time[i])
            print('input  j,flux[j]        ',j,flux[j])
        i+=1
        if i > MAXSTORE-1:
            break
    profile_flux = profile_flux[0:i_tot]
    profile_time = profile_time[0:i_tot]
    profile_dtheta = profile_dtheta[0:i_tot]/D # note the 1/D factor!
    profile_theta = np.cumsum(profile_dtheta)*dh_bin
    profile_alt = ha - np.linspace(0,i_tot-1,i_tot)*dh_bin - dh_bin/2
# ??? not sure what this was supposed to be...
    profile_alt_theta = ha - np.linspace(0,i_tot-1,i_tot)*dh_bin - dh_bin
    return profile_flux,profile_time,profile_alt,profile_alt_theta,\
        profile_dtheta,profile_theta,i_tot
```

[13]:
```
def␣
 ↪alt_bin_dimensional(that_vals_sec,flux,dh_bin_km,hakm,vperp_kms,Dkm,tau_hl=0,Verbose=False)
 ↪

    dt_av = that_vals_sec[1] - that_vals_sec[0]
    i = 0
    j = 0
    i_tot = 0
    t_start = dt_av/2.
```

```
    MAXSTORE = 500000 # modify this at some point to use append()
    profile_flux = np.zeros(MAXSTORE)
    profile_time = np.zeros(MAXSTORE)*u.s
    profile_dtheta = np.zeros(MAXSTORE)

    t_skip = that_vals_sec[0] - dt_av # confirmed that this and t_start give␣
↪proper
                                      # alignment of profile_flux and profile_tim
    t_j = t_start + dt_av

    while(j < np.size(flux)):
        if Verbose == True:
            print('j,np.size(flux)=',j,np.size(flux))
        dt_phi = (t_j - t_start) * flux[j]
        d_sum = dt_phi * abs(vperp_kms)
        first = True
        if Verbose== True:
                  print('TP1: d_sum,dt_phi,dh_bin=',d_sum,dt_phi,dh_bin)
        if d_sum <= dh_bin_km: # altitude bin not yet full
            first = False
            while d_sum < dh_bin_km:
                t_j += dt_av
                j+= 1
                #print('TP2: t_j,j=',t_j,j)
                if j >= np.size(flux):
                    break
                dt_phi = flux[j] * dt_av
                d_sum += dt_phi * abs(vperp_kms)
        if j >= np.size(flux):
            break
        dxs = (d_sum - dh_bin_km)/abs(vperp_kms)
        ddt = dxs/flux[j]
#        print('ddt should have units of time:',ddt)
        if first == True:
            ddt = (t_j - t_start)*dxs/dt_phi
        t_last = t_j - ddt
        delta_t = np.abs(t_last - t_start)
        profile_flux[i] = dh_bin_km/(abs(vperp_kms)*delta_t)
    # apply tau_hl if non-zero
        this_alt = hakm-i*dh_bin_km - dh_bin_km/2 # to agree with below
#          if tau_hl != 0:
#              this_tau = tau_hl * np.exp(-this_alt)
#              profile_flux[i] *= np.exp(-this_tau)
        profile_dtheta[i] = 1/profile_flux[i] - 1
        t_start = t_last
        t_now = t_skip + t_last
```

```
        profile_time[i] = t_now
        i_tot = i
        i+=1
    profile_flux = profile_flux[0:i_tot]
    profile_time = profile_time[0:i_tot]
    profile_dtheta = profile_dtheta[0:i_tot]/Dkm # note the 1/D factor!
    profile_theta = np.cumsum(profile_dtheta)*dh_bin_km
#   print('check units of profile_theta',profile_theta)
    profile_alt = hakm - np.linspace(0,i_tot-1,i_tot)*dh_bin_km - dh_bin_km/2
# ??? not sure what this was supposed to be...
    profile_alt_theta = hakm - np.linspace(0,i_tot-1,i_tot)*dh_bin_km -␣
↪dh_bin_km
    return profile_flux,profile_time,profile_alt,profile_alt_theta,\
        profile_dtheta,profile_theta,i_tot
```

[14]:
```
def lnbarometric(lnP, h, method_TofPmbar,mu,g):
    P = np.exp(lnP)
    T = method_TofPmbar(P)*u.K
    H = (const.k_B * const.N_A *T/(mu*g)).to('km').value
    if T<0:
        print(P,T)
    deriv = -1/H # dlnP/dh
    return deriv
```

[15]:
```
def lnbarometric2(lnP, h, method_ToflogPmbar,mu,g):

    T = method_ToflogPmbar(lnP)*u.K
    H = (const.k_B * const.N_A *T/(mu*g)).to('km').value
    if T<0:
        P = np.exp(lnP)
        print(P,T)
    deriv = -1/H # dlnP/dh
    return deriv
```

[16]:
```
# altitude and bending angle to pressure, density, scale height, and temperature
# using summations, but no isothermal cap to infinity

def htheta2sclht(hvals,theta,D,denfac=1,prfac=1,Tfac=1*u.K/u.km):
    imax = np.size(theta)
    dh = np.abs(hvals[0] - hvals[1])
    hfac = 0.4 * dh
    dthet = D * np.gradient(-theta,hvals) # factor of D from Wasserman and␣
↪Ververka and C code quick_invert.c
    znum = np.zeros(imax)
    zden = np.zeros(imax)
    index = np.linspace(0,imax-1,imax)
    sqrt_index = np.sqrt(index)
```

```
    for i in range(1,imax+1):
        j_index = np.array(index[1:i],dtype=int)
        fimj_vals = i - j_index
        fimj1_vals= fimj_vals - 1
        fac1_vals = sqrt_index[fimj_vals]*fimj_vals
        fac2_vals = sqrt_index[fimj1_vals]*fimj1_vals
        fac3_vals = fimj_vals  * fac1_vals
        fac4_vals = fimj1_vals * fac2_vals
        znum[i-1]= sum(dthet[j_index-1].value*(fac3_vals-fac4_vals))
        zden[i-1]= sum(dthet[j_index-1].value*(fac1_vals-fac2_vals))
    pr  = znum * prfac
    den = zden * denfac
    zden[0] = zden[1] # to avoid divide by zero
    H = hfac*znum/zden
    T = H * Tfac
    return dthet,den,pr,H,T,znum,zden# invert the isothermal model curve
```

[17]:
```
def HtoT(G,mu,H):
    Avogadro =const.N_A # 6.022141e23
    kBoltzCGS = const.k_B.to(u.erg/u.K) #1.380658e-16
    Gcgs    = G.to(u.cm/u.s**2)
    Tfac = (mu *Gcgs /(const.N_A * const.k_B)).to(u.K/u.km)
    T = (Tfac * H).to(u.K)
    return T # in K
```

[18]:
```
def TtoH(G,mu,T):
    Avogadro =const.N_A # 6.022141e23
    kBoltzCGS = const.k_B.to(u.erg/u.K) #1.380658e-16
    Gcgs    = G.to(u.cm/u.s**2)
    Tfac = (mu *Gcgs /(const.N_A * const.k_B)).to(u.K/u.km)
    H = (T/Tfac).to(u.km)
    return H
```

[19]:
```
# get physical units profile from refractivity

def pro_refrac2profile(R,nu,Ttop,G,mu,refrac):
#     ; input
# ;       nu(R)   radial refractivity profile  (R in km, radius from center of␣
 ↪planet)
# ;       Ttop    Temperature (K) at top of profile
# ;       Gref    gravitational acceleration (m/s^2)
# ;       mu      scalar mean molecular wt
# ;       refrac  refractivity

# ; output
# ;       T(R)    Kelvin
# ;       P(R)    Pressure (Pascals)
```

```
# ;       den     density  (kg/m^3)
# ;       H(R)    scale height (km)
#     print('R=',R[0:4])
#     print('nu=',nu[0:4])
#     print('Ttop',Ttop)
#     print('G',G)
#     print('mu',mu)
#     print('refrac',refrac)


    Avogadro =const.N_A # 6.022141e23
    kBoltzCGS = const.k_B.to(u.erg/u.K) #1.380658e-16
    mAMU    = 1*u.g/const.N_A # 1.66053886e-24 # gm /mol
    Loschmidt = 2.6867811e+19 /u.cm**3 # const.amagat # 2.68684e19
    nR      = len(nu)
    ncm3    = nu*Loschmidt/refrac
    Pmbar    = np.zeros(nR,dtype=float)*u.mbar
    Gcgs    = G.to(u.cm/u.s**2)
    dencgs  = ncm3 * mu / Avogadro # gm/cm^3
    den     = dencgs * 1000.0 # kg/m^3

    Pmbar[0] = (ncm3[0] * kBoltzCGS * Ttop).to(u.mbar) # need pressure at top␣
 ↪of top shell
#     ␣
 ↪print('ncm3[0],kBoltzCGS,Ttop,Pcgs[0],refrac',ncm3[0],kBoltzCGS,Ttop,Pmbar[0],refrac)

    # integrate hydrostatic equation to get dP

    ivals = np.linspace(1,nR-1,num=nR-1,dtype=int)
    for i in ivals:
#         if i < 5:
#             print(i)
#             print('R[i]',R[i])
#             print('nu[i]',nu[i])
#             print('Pmbar[i-1]',Pmbar[i-1])
#             print('ncm3[i-1]',ncm3[i-1])
#             print('dencgs[i-1]',dencgs[i-1])
#         Pmbar[i] = Pmbar[i-1] - (dencgs[i-1] * Gcgs * (R[i] - R[i-1])).to(u.
 ↪mbar)
#  this reduces the error considerably
        Pmbar[i] = Pmbar[i-1] - ((dencgs[i]+dencgs[i-1])/2. * Gcgs * (R[i] -␣
 ↪R[i-1])).to(u.mbar)

    ncm3[-1]  = ncm3[-2] # fix off-scale endpints
    Pmbar[-1]  = Pmbar[-2]
    den[-1]  = den[-2]
```

```
    T       = (Pmbar/(kBoltzCGS * ncm3)).to(u.K)
    Tfac = (mu *Gcgs /(const.N_A * const.k_B)).to(u.K/u.km)
    H       = T/Tfac

    return ncm3,den,Pmbar,T,H
```

[20]:
```
# convert altitude/bending angle arrays to lightcurve in time, accommodating␣
↪ray crossing
def hvalstheta2tsecflux(hvals,theta,vperp,tsec_start,tsec_stop,dt_sec,D_km):
    dh_km = abs(hvals[0] - hvals[1]) # input altitude spacing
    dlc_km = dt_sec * abs(vperp) # conversion between time and distance on␣
↪lightcurve
    dh_dlc = dh_km/dlc_km # non-dimensional
    ijmax = int((tsec_stop - tsec_start)/dt_sec+0.5)
    flux = np.zeros(ijmax)
    tsec_vals = tsec_start + np.linspace(0,ijmax-1,ijmax)*dt_sec
    thetaD = theta * D_km / dlc_km
#    print('theta',theta)
#    print('thetaD -  should be dimensionless!',thetaD)
    imax = theta.size
    for i in range(1,imax):
        zji=(i-1)*dh_dlc + thetaD[i-1] - 1.5 # empirical factor
        zji1=i*dh_dlc + thetaD[i] - 1.5
        zjmin=min([zji,zji1])
        zjmax=max([zji,zji1])
        jmin=int(1+zjmin)
        jmax=int(1+zjmax)
        if jmin <= ijmax and jmax >= 1:
            jmin1=max([jmin,1])
            jmax1=min([jmax,ijmax])
            fphi=dh_dlc/(zjmax-zjmin)
            phifac=fphi
            for j in range(jmin1,jmax1+1):
                add=phifac*(min([zjmax,j])-max([zjmin,(j-1)]))
                flux[j-1] += add
    flux[0:2] = flux[2] # since first bins not filled
    return tsec_vals,flux
```

[21]:
```
def lightcurve_new(hvals,theta,tau_hl=0,pts_per_H=1000):
    dh = hvals[0] - hvals[1]
    print('in lightcurve_new: dh = ',dh)
    hhat = 1./dh
    that = pts_per_H
    dj = 1./that
    at = dj

# compute duration of lightcurve
```

```
    ha = max(hvals)
    hb = -min(hvals)
    du = ha + np.exp(hb) + hb

   # print("in lightcurve: pts_per_H,hhat,ha,hb,du = ",pts_per_H,hhat,ha,hb,du)

    ijmax = int(du/at+0.5)
    du = at*ijmax

    flux = np.zeros(ijmax)
    #print("in lightcurve: pts_per_H,du,ijmax,at = ",\
    #      pts_per_H,du,ijmax,at)

# This is the best I could come up with for alignment with variety of choices of dh and dj
# not ideal but good enough.
# using golc() run 5 for a variety of choices
   # that_vals = -max(hvals) + np.linspace(0,ijmax-1,ijmax)*dj - 1 +dh - 2*dj# so zero at half-light
   # that_vals = -max(hvals) + np.linspace(0,ijmax-1,ijmax)*dj - 1 +2*dh - 1*dj
    that_vals = -max(hvals) + np.linspace(0,ijmax-1,ijmax)*dj - 1 +dh# -2*dh #-1*dj
    #print('in lightcurve: dj,that_vals[0:4]=',dj,that_vals[0:4])
# step through entire theta array and compute contribution to this part of lightcurve

    imax = theta.size
    for i in range(1,imax):
        zji=((i-1)*dh+theta[i-1])/dj
        zji1=(i*dh+theta[i])/dj
        zjmin=min([zji,zji1])
        zjmax=max([zji,zji1])
        jmin=int(1+zjmin)
        jmax=int(1+zjmax)
        if jmin <= ijmax  and jmax >= 1:
            jmin1=max([jmin,1])
            jmax1=min([jmax,ijmax])
            fphi=dh/(dj*(zjmax-zjmin))
            phifac=fphi
            for j in range(jmin1,jmax1+1):
                add=phifac*(min([zjmax,j])-max([zjmin,(j-1)]))
                flux[j-1] += add
    if tau_hl>0:
        flux *= np.exp(-tau_hl * (1/flux-1))
    return that_vals,flux
```

[22]:
```python
# Use refractivity profile to determine bending angle

def nu2theta(dh_km,nu_in):

# integrate along line of sight through successively deeper rays...
    if nu_in[10]< nu_in[9]:
        print('Refractivity should increase with depth. Flipping...')
        nu=np.flip(nu_in)
    else:
        nu = nu_in
    imax = np.size(nu)
    theta = np.zeros(imax)
    index = np.linspace(0,imax-1,imax)
    fac2_all = np.sqrt(index)
    for i in range(imax):
        i_index= np.array(index[0:i+1],dtype=int)
        i_vals = i - i_index
        fac1_vals = -2*fac2_all[(i_vals-1).clip(min=0)]
        fac2_vals = fac2_all[i_vals]
        fac3_vals = fac2_all[(i_vals-2).clip(min=0)]
        facs = fac1_vals + fac2_vals + fac3_vals
        theta[i-1] = sum(nu[i_index] * facs)
        print(i,end='\r')
    # scale theta by R_planet/dh
    theta *= (VIMS['r_curv']/abs(dh_km)).value
    return theta
```

[23]:
```python
def pro_lcgen_v2(R_in,nu_in,D_in):
    R = R_in.value
    nu = nu_in
    D = D_in.value
    nRp1 = len(R)
    nR = nRp1 - 1
    nRm1 = nR - 1
# determine if arrays are increasing outward or not

    if R[0] < R[-1]:
        print('flipping input R array')
        R       = np.flip(R)
    if nu[0] > nu[-1]:
        print('flipping input nu array')
        nu      = np.flip(nu)
    print('First ten elements of R, nu: R should decrease, nu should increase')
    print(R[0:10])
    print(nu[0:10])
    print('len(R),len(nu)',len(R),len(nu))
    if R[-1] > R[100]:
```

```
        print('R array is not in the correct order!')
    else:
        print('R array is in the correct order')
    if nu[-1] < nu[100]:
        print('nu array is not in the correct order!')
    else:
        print('nu array is in the correct order')
#  compute coefficients without using 2-D arrays
    dr      = np.abs(R[0:nRm1+1] - R[1:nR+1]) # note that dr is always positive
    Rsq     = (R*R)

# ; compute theta array
    theta = np.zeros(nRp1,dtype=object)
    rho = np.zeros(nRp1,dtype=object)
    dnudr = np.gradient(nu,R)
    dnudr = abs(np.diff(nu)/np.diff(R))
    ivals = np.linspace(1,nR,num=nR,dtype=int)

    for i in ivals:
        print(i,end='\r')
#        Xji = np.array([np.sqrt(Rsq[0:i-1+1] - Rsq[i]),0.0],dtype=object)
        Xji = np.sqrt(Rsq[0:i-1+1] - Rsq[i])
        Xji=np.append(Xji,0.0)
        dXji = -np.diff(Xji)
#        print('i,Xji,dXji',i,Xji,dXji)
        vals = dXji[0:i] * dnudr[0:i]/R[0:i] * 2.0 * R[i]

#        print('vals[0],np.sum(vals[0])',vals[0],np.
 ↪sum(vals[0],keepdims=False))
#THIS IS VERY SLOW
#        theta[i] = np.sum(vals[0],keepdims=False)
# try this instead
        theta[i] = sum(vals)
#        print('i,theta[i],theta_compare',i,theta[i],theta_compare)
        rho[i]   = R[i] - D*theta[i]
#        print('theta[0:i+1]:',theta[0:i+1])
#         if i <= 4:
#             print('\nR[i],nu[i],dnudr[i]',R[i],nu[i],dnudr[i])
#             print('Xji',Xji)
#             print('dXji',dXji)
#             print('vals',vals)
#             print('theta[i]',theta[i])
#             print('rho[i]',rho[i])
    phi_cyl = np.zeros(nRp1)
    drho = np.abs(np.diff(rho))
#    print('drho',drho)
#    phi_cyl = dr/drho
```

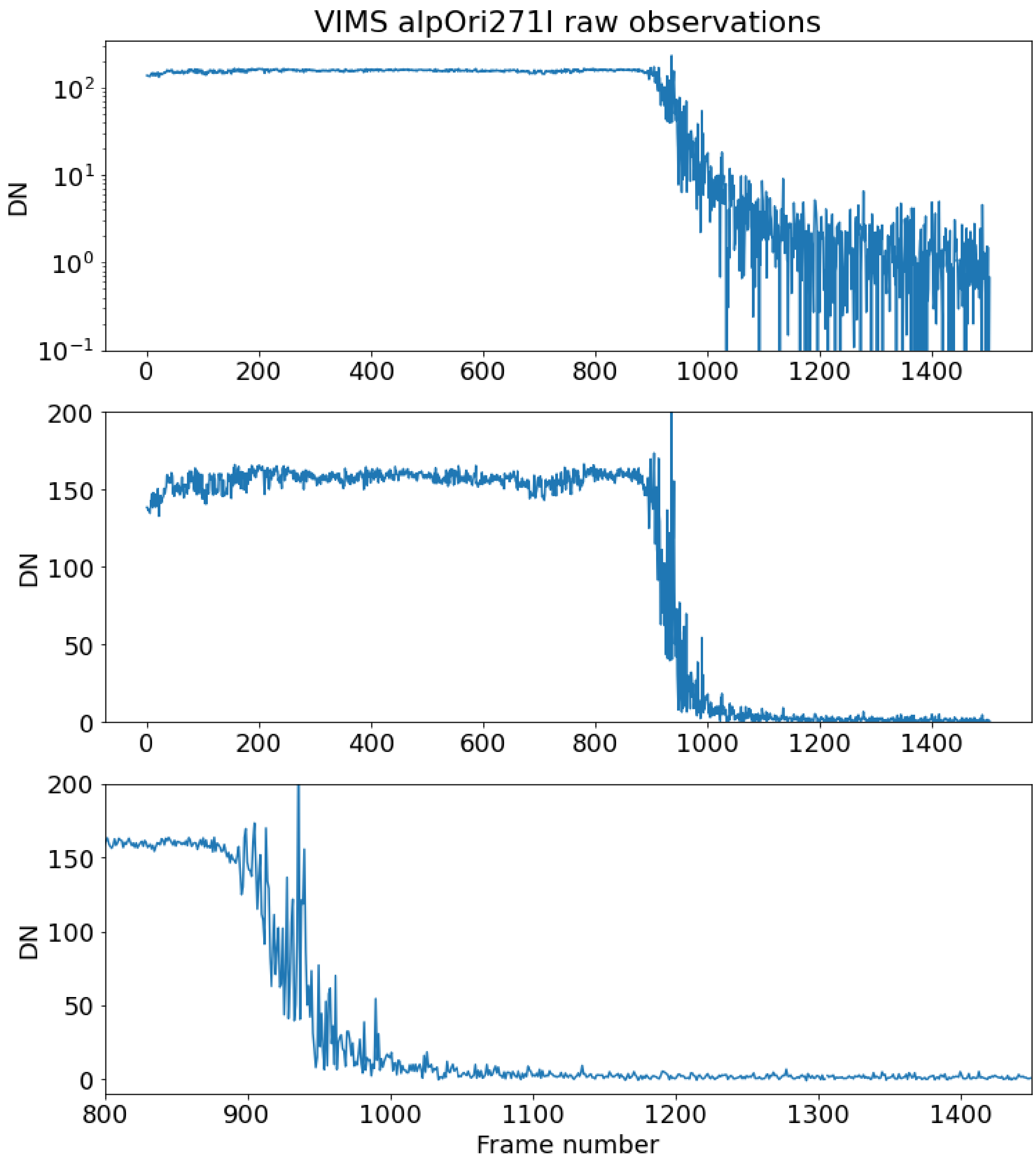
```
    for i in ivals:
        phi_cyl[i] = dr[i-1]/abs(rho[i-1] - rho[i])
    phi_cyl[0]      = 1.0 # top of light curve
    phi_cyl[1] = (phi_cyl[0] + phi_cyl[2])/2
#    print('phi_cyl[0:4]',phi_cyl[0:5])
    rho[0]=rho[1]
    return rho,theta,phi_cyl
```

[24]:
```
# determine refractivity and bending angle from numerical inversion of␣
 ↪lightcurve
def pro_lcinvert(rho_in,phi_cyl_in,nu0_in,D_in):
#    rho_in: distance in observer plane of ray from projected center of planet␣
 ↪in km
#    phi_cyl_in: normalized flux (cyl means no spherical focussing)

    phi_cyl = np.zeros(len(phi_cyl_in),dtype=float) + phi_cyl_in
#     print(phi_cyl.flags.writeable)
    D = D_in

    nRp1 = len(rho_in)
    rho = np.zeros(nRp1,dtype=float) + rho_in
    rho[0] = rho[1] + rho[1] - rho[2]

    nR = nRp1-1
    nRm1 = nR -1

    if rho_in[10] < rho_in[-1]:
        rho = np.flip(rho)
        print('reversed order of rho')
    if phi_cyl[5]<phi_cyl[-1]:
        phi_cyl = np.flip(phi_cyl)
        print('reversed order of phi_cyl')
    if phi_cyl[1] < .1:
        phi_cyl[1]  = 0.99999643
    R = np.zeros(nRp1, dtype = float)
    dr = np.zeros(nR, dtype = float)

    R[0] = rho[0] # align at top of

#     print('rho[0:4]',rho[0:5])
#     print('phi_cyl[0:4]',phi_cyl[0:5])

    ivals = np.linspace(1,nR-1,num=nR-1,dtype=int)
    for i in ivals:
        dr[i] = abs(rho[i-1]- rho[i]) * phi_cyl[i]
        if i == 1:
            dr[0] = dr[1]
```

```python
        R[i] = R[i-1] - dr[i-1]
#         if i <=5:
#             print('i,dr[i-1]',i,dr[i-1])
#             print('R[i-1],R[i]',R[i-1],R[i])
#     if R[0] == R[1]:
 #       R[0] = R[1]-(R[2]-R[1])
    Rsq     = R*R
    if dr[0] == 0:
        dr[0] = dr[1]
    theta   = (R - rho)/D
    dnu     = np.zeros(nRp1,dtype=float)
    nu      = np.zeros(nRp1,dtype=float)
    nu0     = nu0_in
    Aji_01  = 2.0*np.sqrt(Rsq[0] - Rsq[1])/dr[0]
    if Aji_01>0:
        dnu[0]  = theta[1]/Aji_01
    nu[1]   = nu[0] + dnu[0]
#     print('rho[0:4]',rho[0:4])
#     print('theta[0:4]',theta[0:4])
#     print('phi_cyl[0:4]',phi_cyl[0:4])
#     print('R[0:4]',R[0:4])
#     print('dr[0:4]',dr[0:4])
#     print('Aji_01=',Aji_01)
#     print('dr[0]',dr[0])
#     print('R[0:4]-rho[0:4]',R[0:4]-rho[0:4])
#     print('D',D)
    for i in ivals:
        Xji1    = np.sqrt(Rsq[0:i+2] - Rsq[i+1])
        Aji1    = 2. * R[i+1]/R[0:i+1] * (Xji1[0:i+1] - Xji1[1:i+2])/dr[0:i+1]
        if Aji1[i] >0:
            dnu[i]  = (theta[i+1] - sum(Aji1[0:i-1+1]*dnu[0:i-1+1]))/Aji1[i]
        nu[i+1] = nu[i] + dnu[i]
#         if i <= 4:
#             print(i)
#             print('Xji1',Xji1)
#             print('Aji1',Aji1)
#             print('dnu[i]',dnu[i])
#             print('nu[i]',nu[i])
        print(i,end='\r')
    nu[0] = nu[1]
    nu[-1]=nu[-2]
    return nu,theta,R
```

2 Load data, plot and perform isothermal and iso-tau band model fits

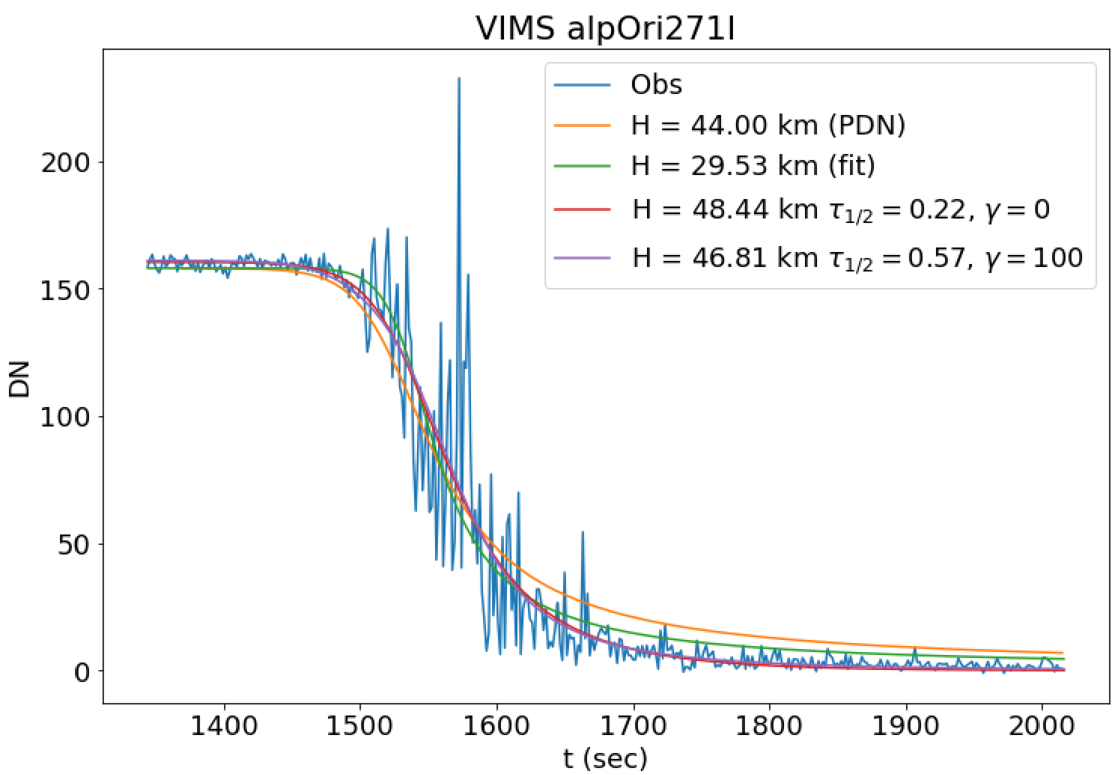
[25]:
```
infile = VIMS['path2inputfiles'] + VIMS['VIMS_observed_lightcurve']
figfile_rawdata = VIMS['path2outputfiles'] + VIMS['event']+'_rawdata.jpg'

counts = plot_VIMS_data(infile,figfile_rawdata)

figfile_isofits = VIMS['path2outputfiles'] + VIMS['event']+'_isofits.jpg'
figfile_isofits_CIRS = VIMS['path2outputfiles'] + VIMS['event']+'_isofits_CIRS.
 ↪jpg'

tsec,data,model_iso,fitted_params,model_tau,fitted_params_tau =␣
 ↪plot_isofits(VIMS,counts,figfile_isofits)
```




```
Saved ./output/VIMS alpOri271I_rawdata.jpg

--------------Isothermal model------------------
Parameter                   Value          Stderr
H_km                      29.5256          1.7366
t_hl                    1559.9046          1.8103
y_bkg                      0.0000          0.0000
y_scale                  158.0188          1.6116
--------------------Band model 1-----------------
Parameter                   Value          Stderr
```

```
H_km                      48.4420         5.6023
t_hl                    1577.7694         6.1565
y_bkg                      0.0000         0.0000
y_scale                  160.4330         1.7819
tau_hl                     0.2164         0.0821
tau_gamma                  0.0000         0.0000
-------------------Band model 2----------------
Parameter                   Value         Stderr
H_km                      46.8084         3.9394
t_hl                    1597.0443         8.8826
y_bkg                      0.0000         0.0000
y_scale                  161.0295         1.8444
tau_hl                     0.5699         0.1534
tau_gamma                100.0000         0.0000
```

```
<Figure size 432x288 with 0 Axes>
```




```
Saved ./output/VIMS alpOri271I_isofits.jpg
```

```
<Figure size 432x288 with 0 Axes>
```

3 Load CIRS data and construct T(P) profile

[26]:
```
Isothermal = False

Hhat = 50 # points per scale height
#Hhat = 100 # points per scale height
#Hhat = 200 # points per scale height
#Hhat = 300 # points per scale height
#Hhat = 500 # points per scale height
Ha = 15.
Hb = 7.05
dh_bin_sec = 0.01*u.s
# dh_bin_sec = 0.02*u.s
# dh_bin_sec = 0.1*u.s
dh_bin_km = dh_bin_sec * abs(VIMS['vperp'])

bin_km = dh_bin_km

# get CIRS profile

IDL = readsav(VIMS['path2inputfiles']+VIMS['CIRS_TPprofiles'])
CIRS_Tarray = IDL['finaltemp']
CIRS_pmbar = IDL['press']*1e3 * u.mbar
CIRS_lat_list = IDL['newlat'] * u.deg
CIRS_JD_list = IDL['newdays']

VIMSlat_deg = VIMS['lat_c']
spice.furnsh(VIMS['path2kernels']+VIMS['tlsfile'])
VIMSUTC = VIMS['UTC']
VIMSETsec = spice.str2et(VIMSUTC)
VIMSJDocc = spice.j2000() + VIMSETsec/spice.spd()

JD = VIMSJDocc

dlat_deg = abs(VIMSlat_deg-CIRS_lat_list)
index_lat = np.where(dlat_deg == min(dlat_deg))
index_lat = int(index_lat[0])

dJD = abs(JD - CIRS_JD_list)
index_JD = np.where(dJD== min(dJD))
index_JD = int(index_JD[0])

CIRS_T = CIRS_Tarray[index_JD,index_lat,:]*u.K

index = np.where(CIRS_T > 50*u.K)
CIRS_Tindex = CIRS_T[index]
CIRS_pmbarindex = CIRS_pmbar[index]
```

```
fill_Thigh_P= CIRS_Tindex[0]
fill_Tlow_P = CIRS_Tindex[-1]
fill_Tbelow = CIRS_Tindex[0]
fill_Tabove = CIRS_Tindex[-1]

if Isothermal == True:
    H_km = VIMS['H_km']
    T_VIMS = HtoT(VIMS['g_ms2'],VIMS['mu'],H_km)
    method_TofPmbar=␣
 ↪interp1d(CIRS_pmbarindex,CIRS_Tindex*0+T_VIMS,'cubic',bounds_error=False,
        fill_value = (T_VIMS,T_VIMS))
    method_ToflogPmbar= interp1d(np.log(CIRS_pmbarindex.value),CIRS_Tindex*0␣
 ↪+T_VIMS,'cubic',bounds_error=False,\
        fill_value=(T_VIMS,T_VIMS))
else:
    H_km = TtoH(VIMS['g_ms2'],VIMS['mu'],fill_Tlow_P) # in km already
    method_TofPmbar=␣
 ↪interp1d(CIRS_pmbarindex,CIRS_Tindex,'cubic',bounds_error=False,
       fill_value=(fill_Tlow_P,fill_Thigh_P))
    method_ToflogPmbar= interp1d(np.log(CIRS_pmbarindex.
 ↪value),CIRS_Tindex,'cubic',bounds_error=False,\
       fill_value=(fill_Tlow_P,fill_Thigh_P))

Haphb = Ha + Hb
NH = int(Hhat*Haphb)
imax = NH
dH = 1.0/Hhat
print('dH=',dH,' in scale heights')
Hvals = -Hb + np.linspace(0,NH-1,NH)*dH # this is positive upward

Hkm = H_km.value
hvalskm = Hvals * H_km # array of altitudes in km
hvalskm_ = hvalskm.value

g = VIMS['g_ms2'].to('km/s**2')
Losch = (2.6867811e25/u.m**3).to('1/km**3')
RSTP = VIMS['RSTP']
mu = VIMS['mu']
R_curv = VIMS['r_curv']
N_A = const.N_A
D_km = VIMS['range']

denfac0 = 2*Losch/(np.pi*np.sqrt(2*R_curv)*RSTP)
denfac = denfac0 * bin_km**1.5 * 2/(3*D_km) # number density /km^3
denfac_cgs = denfac.to(1/u.cm**3)
den_halflight = (denfac0*H_km**1.5/D_km).to(1/u.cm**3)*np.sqrt(np.pi)/2
```

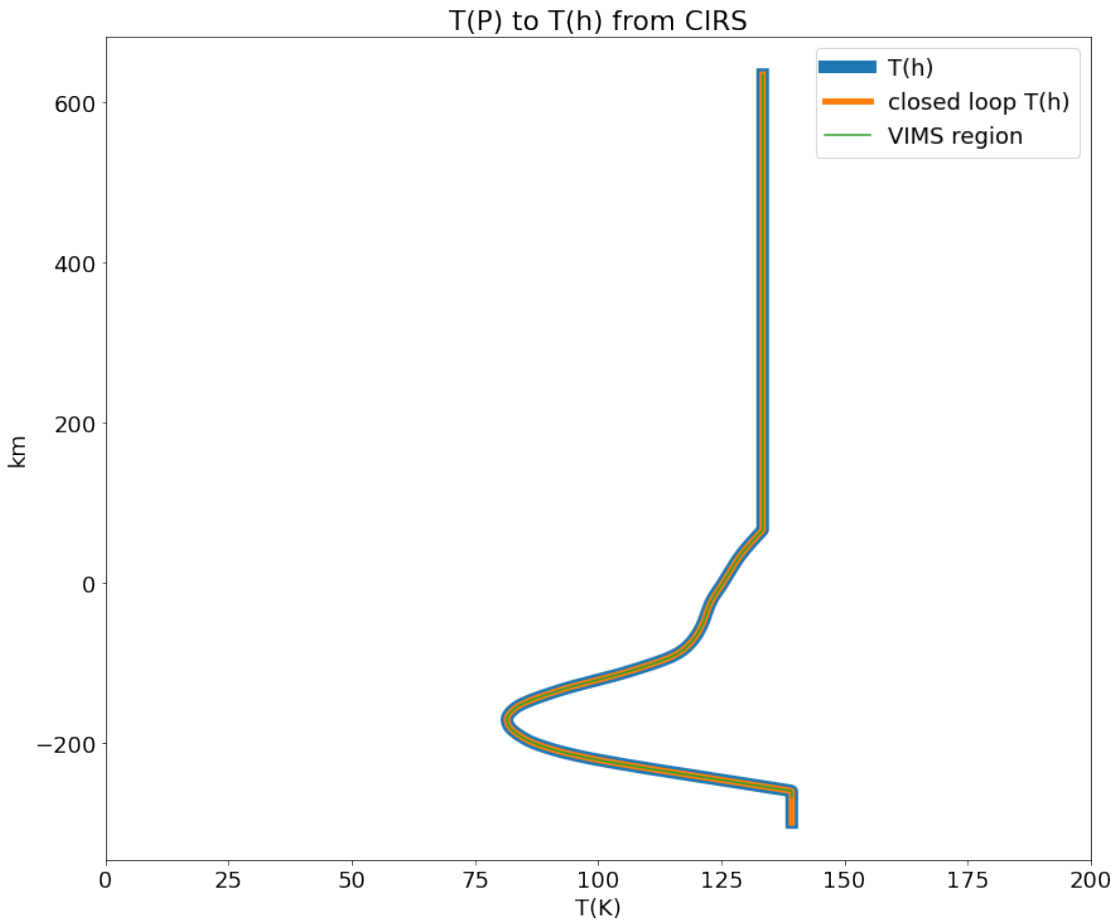
```
prfac0 = 4*g*mu*Losch/(3*np.pi*np.sqrt(2*R_curv)*RSTP*N_A)
prfac  = prfac0 * bin_km**2.5 * 2/(5*D_km)
prfac_mbar = prfac.to(u.mbar)
phalf_mbar = (prfac0 * H_km**2.5/D_km * 3 * np.sqrt(np.pi)/4).to(u.mbar)

dhkm = hvalskm[1]-hvalskm[0]

#specify initial condition (boundary condition )

Pmbar_initial = phalf_mbar.value * np.exp(Hb)
P0_mb = Pmbar_initial

print('Deepest pressure level P0_mb=',P0_mb)
g_kms2 = VIMS['g_ms2'].to('km/s**2')
lnPmbarinitial = np.log(Pmbar_initial)

lnPsolution = odeint(lnbarometric, lnPmbarinitial,␣
 ↪hvalskm,args=(method_TofPmbar,mu,g_kms2),rtol=1e-12)
lnPodeint = lnPsolution[:,0] # Pressure as a function of height
Podeint = np.exp(lnPodeint)

lnPsolution2 = odeint(lnbarometric2, lnPmbarinitial,␣
 ↪hvalskm_,args=(method_ToflogPmbar,mu,g_kms2),rtol=1e-12)
lnPodeint2 = lnPsolution2[:,0] # Pressure as a function of height
Podeint2 = np.exp(lnPodeint2)

Tofh = method_TofPmbar(Podeint)
Pofh = Podeint

Tofh2 = method_ToflogPmbar(lnPodeint2)
Pofh2 = Podeint2

if Isothermal == True:
    fill_value=(T_VIMS,T_VIMS)
else:
    fill_value=(fill_Tlow_P,fill_Thigh_P)

method_Tofhkm = interp1d(hvalskm_,Tofh,bounds_error=False,
    fill_value=fill_value)

method_Pofhkm= interp1d(hvalskm_,Podeint,bounds_error=False)

method_lnPofhkm= interp1d(hvalskm_,lnPodeint,bounds_error=False)

method_hkmofPmbar= interp1d(Podeint,hvalskm_,bounds_error=False)
```

```python
method_hkmoflnPmbar= interp1d(lnPodeint,hvalskm_,bounds_error=False)

testTofh = method_TofPmbar(method_Pofhkm(hvalskm_))

testTofh2 = method_ToflogPmbar(method_lnPofhkm(hvalskm_))

# confirm that starting Podeint is our intial value
# print(Pofh[0],Pmbar_initial)

# compute number density and then refractivy
nofh = (Pofh*u.mbar/(const.k_B * Tofh*u.K)).to("1/cm**3")
# print("min,max nofh=",min(nofh),max(nofh))
# print("min,max Pofh=",min(Pofh),max(Pofh))
# print("min,max Tofh=",min(Tofh),max(Tofh))
Nofh = nofh *RSTP/Losch.to('1/cm**3')

T_new = method_TofPmbar(Pmbar_initial)*u.K
P_new = np.exp(method_lnPofhkm(0))*u.mbar

print("\nPmbar at 0 altitude from method_lnPofhkm:",P_new)
print("Compared to phalf_mbar.value",phalf_mbar.value,'\n')

n_new = (P_new/(const.k_B * T_new)).to("1/cm**3")
print('Compute half-light nden: ',n_new,den_halflight)

nofh_new = (Pofh*u.mbar/(const.k_B * Tofh*u.K)).to("1/cm**3")
dnofh = abs(nofh_new - den_halflight)
index = np.where(dnofh == min(dnofh))[0]

h0 = hvalskm[index]
print('index,half-light altitude from number density =',index,h0)
#determine altitude where Nofh is closest to half light pred value
print(min(Nofh),max(Nofh))
Nofh_new = nofh_new * RSTP / Losch.to('1/cm**3')

H_km_VIMS = H_km

fig1=plt.figure(figsize=(14,12)) # open up figure

nhalf_pred = (Losch * H_km_VIMS**1.5/(np.sqrt(2*np.pi*R_curv)*D_km*RSTP)).to(1/
 ↪u.cm**3)
Nhalf_pred = (nhalf_pred * VIMS['RSTP']/Losch.to('1/cm**3')).value
phalf_pred = (nhalf_pred * H_km_VIMS * mu * g_kms2 * 1/N_A).to(u.mbar)

print(min(Nofh_new),max(Nofh_new))
dNofh = abs(Nofh_new.value-Nhalf_pred)
index = np.where(dNofh == min(dNofh))[0]
```

```
h0 = hvalskm[index]
print('index,half-light altitude from refractivity =',index,h0)
Phalf_odeint = Pofh[index]
print("from odeint, pred: Phalf = ",Phalf_odeint,phalf_pred)

# refractivity at hvalskm of 0
l=np.where(np.abs(hvalskm) == min(np.abs(hvalskm)))
Nofh_new_0km = Nofh_new[l]
print('Refractivity at zero altitude',Nofh_new_0km)

VIMS_Pmbar_max = Pmbar_initial*.5*u.mbar # placeholdeer
index = np.where(hvalskm_ > method_hkmofPmbar(VIMS_Pmbar_max))
hvals_ = hvalskm[index]
xvals = method_Tofhkm(hvals_.value)
#print('xvals=',xvals)
#print('hvals=',hvals)
fig1=plt.figure(1,figsize=(10,6)) # open up figure
plt.rcParams.update({'font.size': 18})
plt.plot(Tofh,hvalskm_,linewidth=10,label='T(h)')
plt.plot(testTofh,hvalskm_,linewidth=5,label='closed loop T(h)')

plt.plot(xvals,hvals_,label='VIMS region')
plt.xlim(0,200)
plt.xlabel('T(K)')
plt.ylabel('km')
#plt.ylim(200,600)
plt.title('T(P) to T(h) from CIRS')
plt.legend()

# now try predicting flux level at top of CIRS data
phi_upper = 1/(1+min(CIRS_pmbarindex)/phalf_pred)
phi_lower = 1/(1+VIMS_Pmbar_max/phalf_pred)
print('predicted flux at upper range of CIRS data=',phi_upper)
print('predicted flux at lower range of VIMS data=',phi_lower)
```

```
dH= 0.02  in scale heights
Deepest pressure level P0_mb= 2752.9373374100487

Pmbar at 0 altitude from method_lnPofhkm: 0.4332912618002866 mbar
Compared to phalf_mbar.value 2.3879225053743665

Compute half-light nden:  2.2533064737770332e+16 1 / cm3 1.2968433102469112e+17
1 / cm3
index,half-light altitude from number density = [279] [-62.26318281] km
3.374104069928733e-14 0.0006873755583793674
3.374104069928733e-14 0.0006873755583793674
index,half-light altitude from refractivity = [279] [-62.26318281] km
```

```
from odeint, pred: Phalf =  [2.14633839] 2.387922505374367 mbar
Refractivity at zero altitude [1.21589141e-07]
predicted flux at upper range of CIRS data= 0.965154682781798
predicted flux at lower range of VIMS data= 0.0017318135334713684
```



4 Construct synthetic lightcurve based on CIRS T(P) and refractivity

[27]:

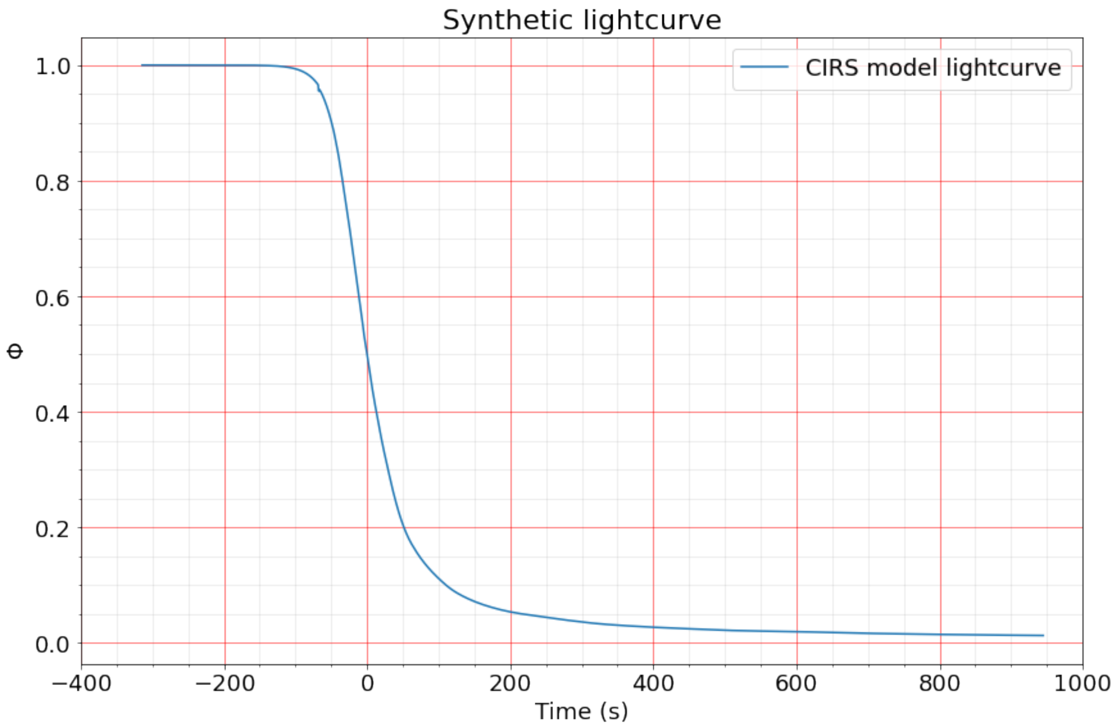
```python
print('Computing lightcurve...')
Rt = hvalskm + VIMS['r_curv']
R_in = Rt
nu_in = Nofh_new
rho2,theta2,phi_cyl2 = pro_lcgen_v2(R_in,nu_in,VIMS['range'])
print('Finished computing lightcurve')
```

```
Computing lightcurve…
flipping input R array
flipping input nu array
```

```
First ten elements of R, nu: R should decrease, nu should increase
[66677.56792294 66676.7208048  66675.87368667 66675.02656854
 66674.1794504  66673.33233227 66672.48521414 66671.638096
 66670.79097787 66669.94385974]
[3.37410407e-14 3.44226549e-14 3.51180387e-14 3.58274701e-14
 3.65512330e-14 3.72896169e-14 3.80429172e-14 3.88114351e-14
 3.95954781e-14 4.03953598e-14]
len(R),len(nu) 1102 1102
R array is in the correct order
nu array is in the correct order
Finished computing lightcurve
```

[28]:

```
offset2 = 0*u.s
xvals2 = -(-np.array(rho2,dtype=float)*u.km+VIMS['r_curv'])/VIMS['vperp'] +␣
 ↪offset2
xvals2[0] = xvals2[1] - (xvals2[2]-xvals2[1])
tscale = xvals2
L = np.where(tscale < 1000*u.s)
tscale = tscale[L]
flux = phi_cyl2[L]
dt_interp = 0.1*u.s

start = np.ceil(tscale[0])
stop = np.floor(tscale[-1] )
Npts = int((stop-start)/dt_interp +1)

t_interpol = np.linspace(start,stop,Npts,endpoint=True)
f_interpol = interp1d(tscale,flux,kind='linear')
flux_interpol = f_interpol(t_interpol)

L = np.min(np.where(flux_interpol <= 0.5))
t_half = t_interpol[L]

fig,ax=plt.subplots(figsize=(14,9)) # open up figure

plt.plot(t_interpol-t_half,flux_interpol,label='CIRS model lightcurve')
plt.xlabel('Time (s)')
plt.ylabel(r'$\Phi$')
plt.title('Synthetic lightcurve')
plt.legend()
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.minorticks_on()
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.xlim(-400,1000)
plt.show()
```



5 Invert synthetic lightcurve to confirm retrieval agrees with input T(P)

[29]:
```
D_in = VIMS['range']
nu_in0 = 0.
nu2,theta2,R2 = pro_lcinvert(rho2,phi_cyl2,nu_in0,D_in)


ncm3_2a,den_2a,Pmbar_2a,T2a,H2a = \
    pro_refrac2profile(R2*u.km,nu2,Tofh[-1]*u.
 ↪K,VIMS['g_ms2'],VIMS['mu'],VIMS['RSTP'])
```

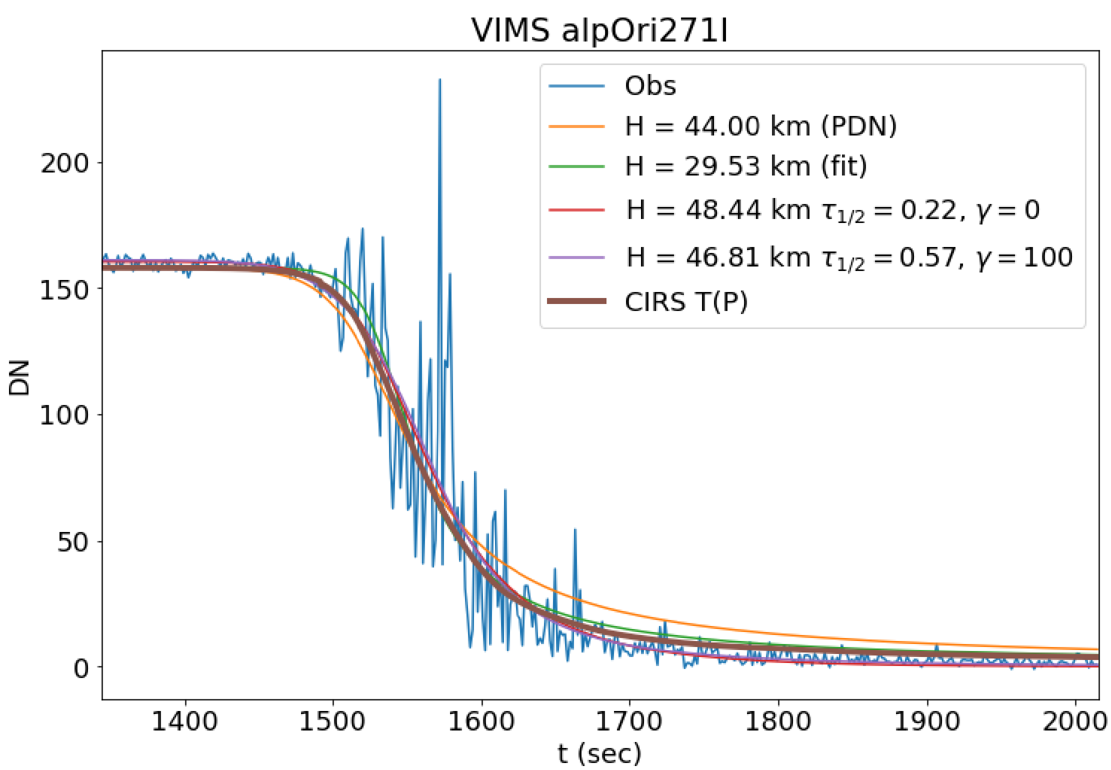
```
1100
```

6 Overplot CIRS model lightcurve on isofit/isotau VIMS observations

[30]:
```
tsec_CIRS = t_interpol-t_half
flux_CIRS = flux_interpol
plot_isofits_CIRS(VIMS,tsec_CIRS,flux_CIRS,counts,figfile_isofits_CIRS)
```

```
--------------Isothermal model------------------
```

```
Parameter                   Value          Stderr
H_km                      29.5256          1.7366
t_hl                    1559.9046          1.8103
y_bkg                      0.0000          0.0000
y_scale                  158.0188          1.6116
-------------------Band model 1-----------------
Parameter                   Value          Stderr
H_km                      48.4420          5.6023
t_hl                    1577.7694          6.1565
y_bkg                      0.0000          0.0000
y_scale                  160.4330          1.7819
tau_hl                     0.2164          0.0821
tau_gamma                  0.0000          0.0000
-------------------Band model 2-----------------
Parameter                   Value          Stderr
H_km                      46.8084          3.9394
t_hl                    1597.0443          8.8826
y_bkg                      0.0000          0.0000
y_scale                  161.0295          1.8444
tau_hl                     0.5699          0.1534
tau_gamma                100.0000          0.0000
```




```
Saved ./output/VIMS alpOri271I_isofits_CIRS.jpg
```

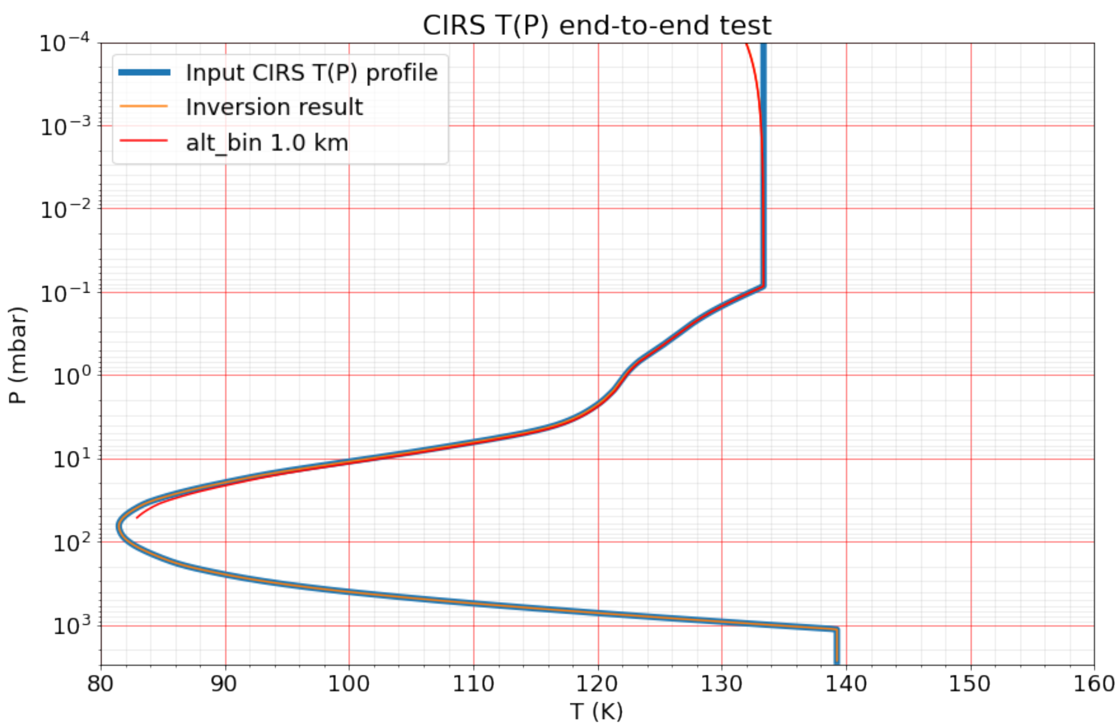
<Figure size 432x288 with 0 Axes>

7 Invert CIRS interpolated lightcurve to confirm it matches input T(P)

[31]:
```
vperp_kms = abs(VIMS['vperp'])
Dkm = VIMS['range']
tau_hl = 0.
dh_bin_km = 1*u.km # desired altitude resolution
hakm = 400*u.km # not sure what this should be

#lightcurve to altitude-binned values

profile_flux,profile_time,profile_alt,profile_alt_theta,\
    profile_dtheta,profile_theta,i_tot = \
   ␣
 ↪alt_bin_dimensional(tsec_CIRS,flux_CIRS,dh_bin_km,hakm,vperp_kms,Dkm,tau_hl=tau_hl,Verbose=


# inversion to retrieve refractivity and bending angle

rho_in = (profile_time*VIMS['vperp']+VIMS['r_curv']).value
phi_cyl_in = profile_flux
nu0_in = 0.
D_in = Dkm.value

nu,theta,R = pro_lcinvert(rho_in,phi_cyl_in,nu0_in,D_in)

# conversion to physical units for vertical profile

ncm3_,den_,Pmbar_,T_,H_ = \
    pro_refrac2profile(R*u.km,nu,Tofh[-1]*u.
 ↪K,VIMS['g_ms2'],VIMS['mu'],VIMS['RSTP'])

# plot results

fig,ax=plt.subplots(figsize=(14,9)) # open up figure
plt.plot(Tofh,Pofh,label='Input CIRS T(P) profile',linewidth=5)

#plt.plot(T2,Pmbar_2,label='Inversion result')
plt.plot(T2a,Pmbar_2a,label='Inversion result')
plt.plot(T_,Pmbar_,color='r',label='alt_bin '+str(dh_bin_km))

plt.yscale('log')
plt.ylim(3000,.0001)
```

```
plt.xlim(80,160)
plt.ylabel('P (mbar)')
plt.xlabel('T (K)')
plt.legend()
plt.title('CIRS T(P) end-to-end test')
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.minorticks_on()
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)

plt.show();
```

798



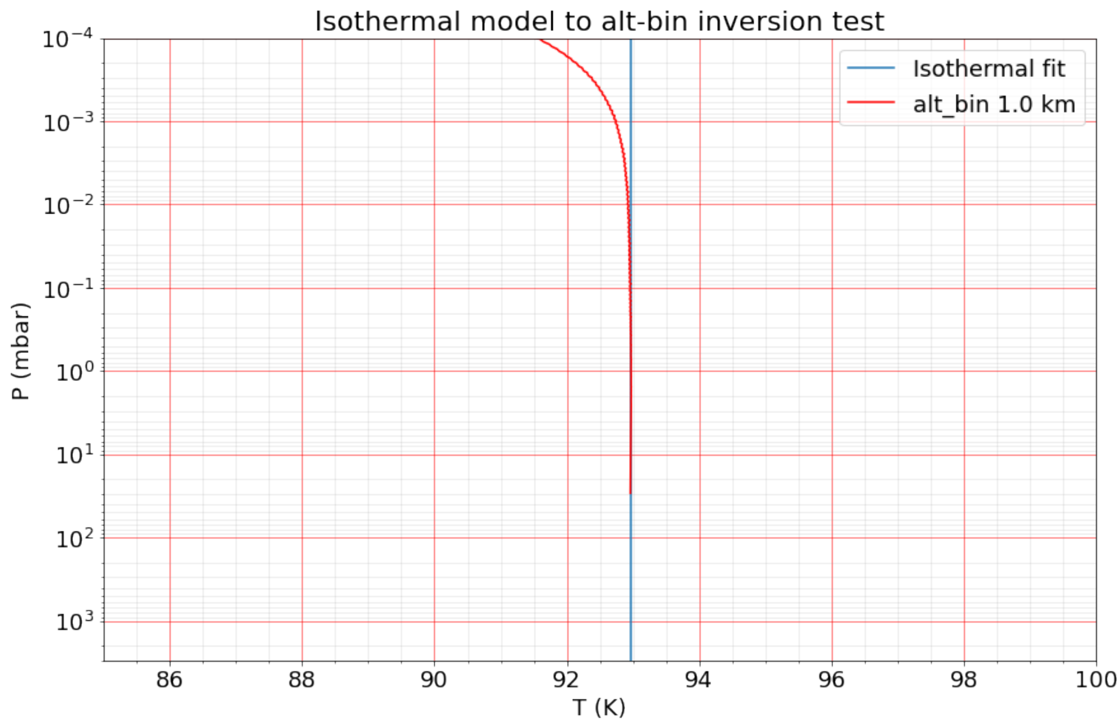
8 Invert isothermal model lightcurve as end-to-end test of alt-bin

[32]:
```
# Normalized flux and halflight-relative time:
t_iso = tsec-fitted_params.params['t_hl']*u.s
flux_iso = (model_iso-fitted_params.params['y_bkg'])/fitted_params.
 ↪params['y_scale']

vperp_kms = abs(VIMS['vperp'])
Dkm = VIMS['range']
tau_hl = 0.
```

```
dh_bin_km = 1*u.km # desired altitude resolution
hakm = 400*u.km # not sure what this should be

#lightcurve to altitude-binned values

profile_flux,profile_time,profile_alt,profile_alt_theta,\
    profile_dtheta,profile_theta,i_tot = \
    ␣
  ↪alt_bin_dimensional(t_iso,flux_iso,dh_bin_km,hakm,vperp_kms,Dkm,tau_hl=tau_hl,Verbose=False

# plt.plot(t_iso,flux_iso)
# plt.plot(profile_time,profile_flux)

# inversion to retrieve refractivity and bending angle

rho_in = (profile_time*VIMS['vperp']+VIMS['r_curv']).value
phi_cyl_in = profile_flux
nu0_in = 0.
D_in = Dkm.value

nu,theta,R = pro_lcinvert(rho_in,phi_cyl_in,nu0_in,D_in)
# conversion to physical units for vertical profile

ncm3_,den_,Pmbar_,T_,H_ = \
    pro_refrac2profile(R*u.km,nu,Tofh[-1]*u.
  ↪K,VIMS['g_ms2'],VIMS['mu'],VIMS['RSTP'])

# plot results

fig,ax=plt.subplots(figsize=(14,9)) # open up figure

T_iso = HtoT(g,mu,fitted_params.params['H_km']*u.km)
plt.axvline(T_iso.value,label='Isothermal fit')

# #plt.plot(T2,Pmbar_2,label='Inversion result')
# plt.plot(T2a,Pmbar_2a,label='Inversion result')
plt.plot(T_,Pmbar_,color='r',label='alt_bin '+str(dh_bin_km))

plt.yscale('log')
plt.ylim(3000,.0001)
plt.xlim(85,100)
plt.ylabel('P (mbar)')
plt.xlabel('T (K)')
plt.legend()
plt.title('Isothermal model to alt-bin inversion test')
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.minorticks_on()
```

```
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)

plt.show();
```

571

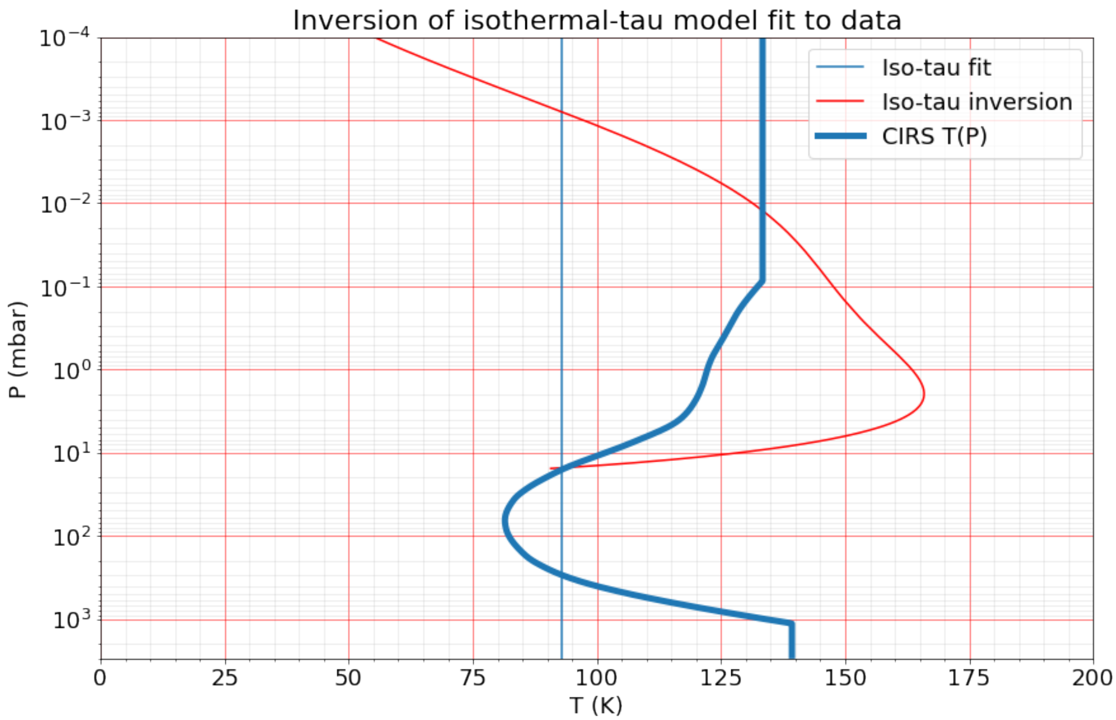


9 Invert isothermal tau model and compare to CIRS T(P)

[33]:
```
# Normalized flux and halflight-relative time:
t_isotau = tsec-fitted_params_tau.params['t_hl']*u.s
flux_isotau = (model_tau-fitted_params_tau.params['y_bkg'])/fitted_params_tau.
 ↪params['y_scale']

vperp_kms = abs(VIMS['vperp'])
Dkm = VIMS['range']
tau_hl = 0.
dh_bin_km = 1*u.km # desired altitude resolution
hakm = 400*u.km # not sure what this should be

#lightcurve to altitude-binned values

profile_flux,profile_time,profile_alt,profile_alt_theta,\
    profile_dtheta,profile_theta,i_tot = \
```

```
    ␣
  ↪alt_bin_dimensional(t_isotau,flux_isotau,dh_bin_km,hakm,vperp_kms,Dkm,tau_hl=tau_hl,Verbose

# plt.plot(t_iso,flux_iso)
# plt.plot(profile_time,profile_flux)

# inversion to retrieve refractivity and bending angle

rho_in = (profile_time*VIMS['vperp']+VIMS['r_curv']).value
phi_cyl_in = profile_flux
nu0_in = 0.
D_in = Dkm.value

nu,theta,R = pro_lcinvert(rho_in,phi_cyl_in,nu0_in,D_in)
# conversion to physical units for vertical profile

ncm3_,den_,Pmbar_,T_,H_ = \
    pro_refrac2profile(R*u.km,nu,Tofh[-1]*u.
  ↪K,VIMS['g_ms2'],VIMS['mu'],VIMS['RSTP'])

# plot results

fig,ax=plt.subplots(figsize=(14,9)) # open up figure

T_isotau = HtoT(g,mu,fitted_params_tau.params['H_km']*u.km)
plt.axvline(T_iso.value,label='Iso-tau fit')

# #plt.plot(T2,Pmbar_2,label='Inversion result')
# plt.plot(T2a,Pmbar_2a,label='Inversion result')
plt.plot(T_,Pmbar_,color='r',label='Iso-tau inversion')
plt.plot(Tofh,Pofh,label='CIRS T(P)',linewidth=5)

plt.yscale('log')
plt.ylim(3000,.0001)
plt.xlim(0,200)
plt.ylabel('P (mbar)')
plt.xlabel('T (K)')
plt.legend()
plt.title('Inversion of isothermal-tau model fit to data')
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.minorticks_on()
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)

plt.show();
```

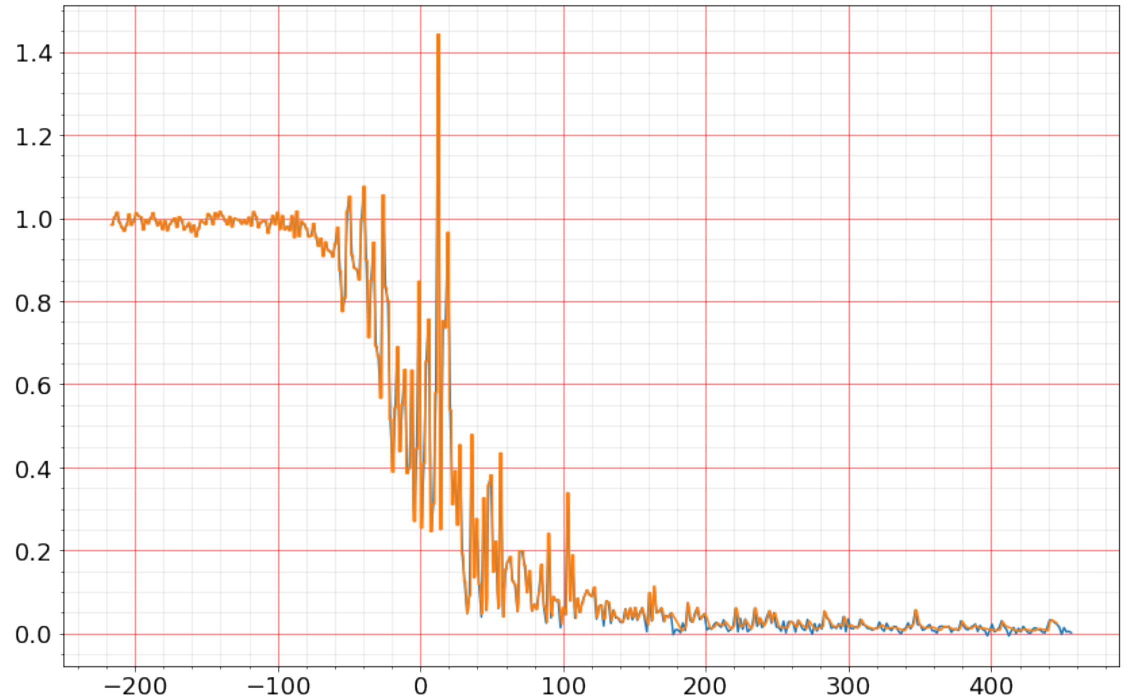
537



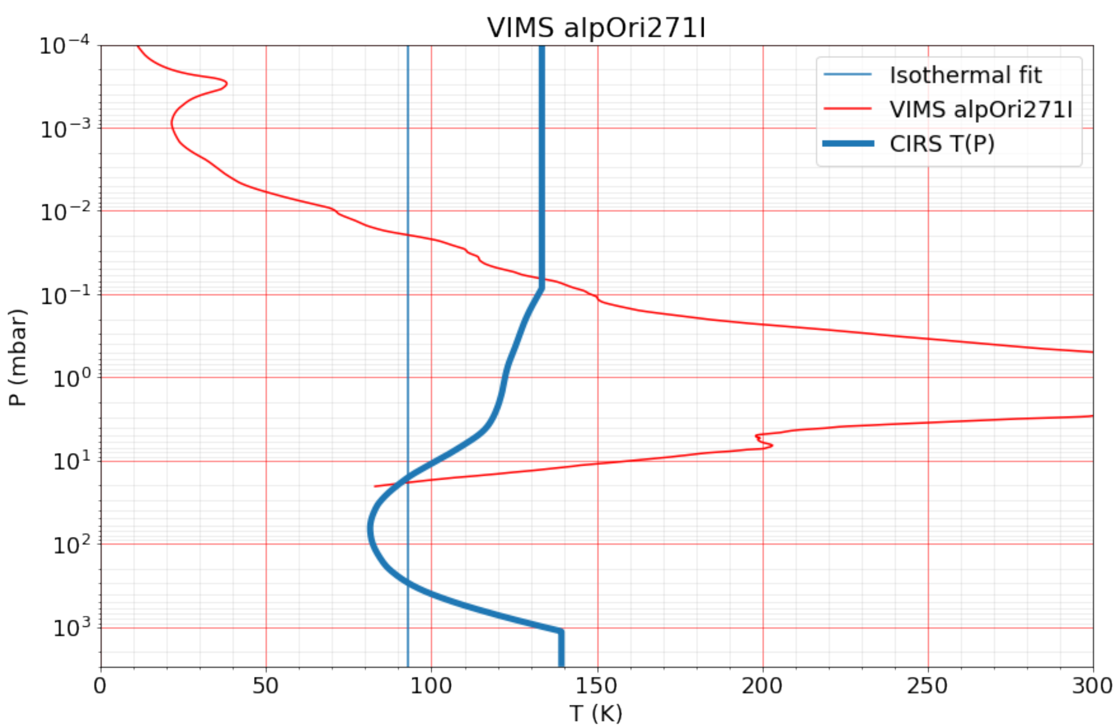
10 Inversion of VIMS lightcurve - no isothermal cap yet

[34]:
```
# Normalized flux and halflight-relative time:
t_obs = tsec-fitted_params.params['t_hl']*u.s
flux_obs = 0.98*(data-fitted_params_tau.params['y_bkg'])/fitted_params.
 ↪params['y_scale']

vperp_kms = abs(VIMS['vperp'])
Dkm = VIMS['range']
tau_hl = 0.
dh_bin_km = 0.1*u.km # desired altitude resolution
hakm = 200*u.km

#lightcurve to altitude-binned values

profile_flux,profile_time,profile_alt,profile_alt_theta,\
    profile_dtheta,profile_theta,i_tot = \
   ␣
 ↪alt_bin_dimensional(t_obs,flux_obs,dh_bin_km,hakm,vperp_kms,Dkm,tau_hl=tau_hl,Verbose=False

fig,ax=plt.subplots(figsize=(14,9)) # open up figure

plt.plot(t_obs,flux_obs)
```

```
plt.plot(profile_time,profile_flux)
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.minorticks_on()
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.show()
# inversion to retrieve refractivity and bending angle

rho_in = (profile_time*VIMS['vperp']+VIMS['r_curv']).value
phi_cyl_in = profile_flux
nu0_in = 0.
D_in = Dkm.value

nu,theta,R = pro_lcinvert(rho_in,phi_cyl_in,nu0_in,D_in)
# conversion to physical units for vertical profile

ncm3_obs,den_obs,Pmbar_obs,T_obs,H_obs = \
    pro_refrac2profile(R*u.km,nu,Tofh[-1]*u.
 ↪K,VIMS['g_ms2'],VIMS['mu'],VIMS['RSTP'])

# plot results

fig,ax=plt.subplots(figsize=(14,9)) # open up figure

T_iso = HtoT(g,mu,fitted_params.params['H_km']*u.km)
plt.axvline(T_iso.value,label='Isothermal fit')


plt.plot(T_obs,Pmbar_obs,color='r',label=VIMS['event'])
plt.plot(Tofh,Pofh,label='CIRS T(P)',linewidth=5)

plt.yscale('log')
plt.ylim(3000,.0001)
plt.xlim(0,300)
plt.ylabel('P (mbar)')
plt.xlabel('T (K)')
plt.legend()
plt.title(VIMS['event'])
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.minorticks_on()
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)

plt.show();
```


1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−200
−100
0
100
200
300
400

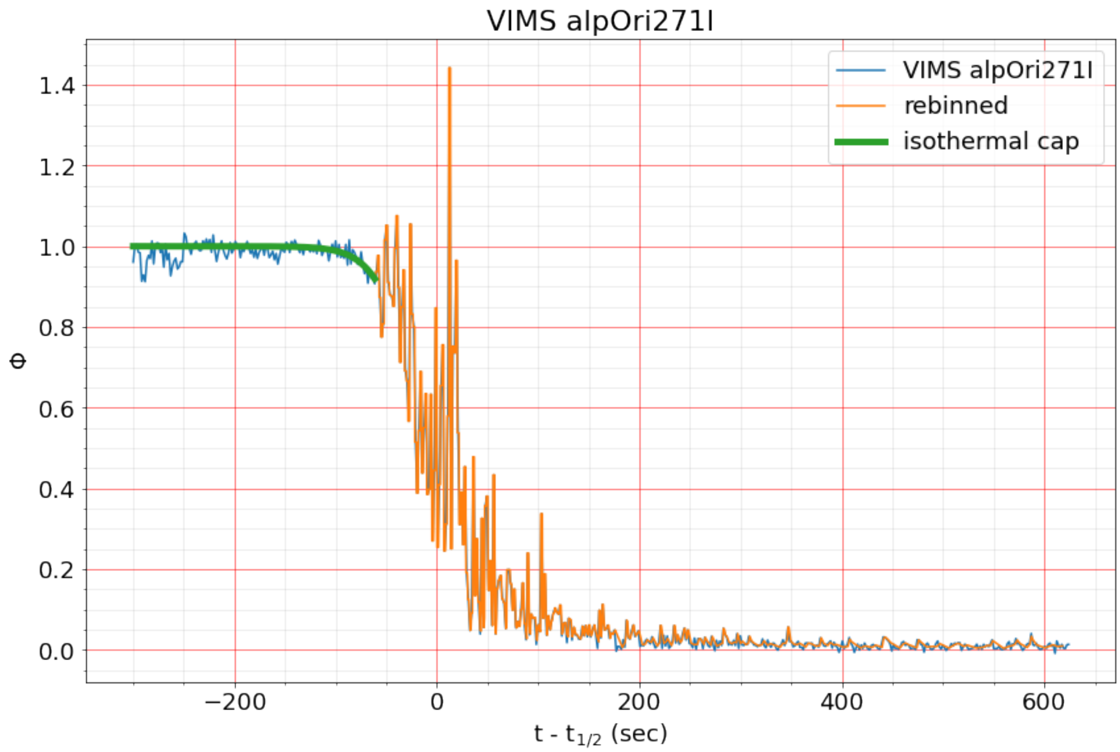
5421


VIMS alpOri271I
Isothermal fit
VIMS alpOri271I
CIRS T(P)
P (mbar)
T (K)
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
0
50
100
150
200
250
300

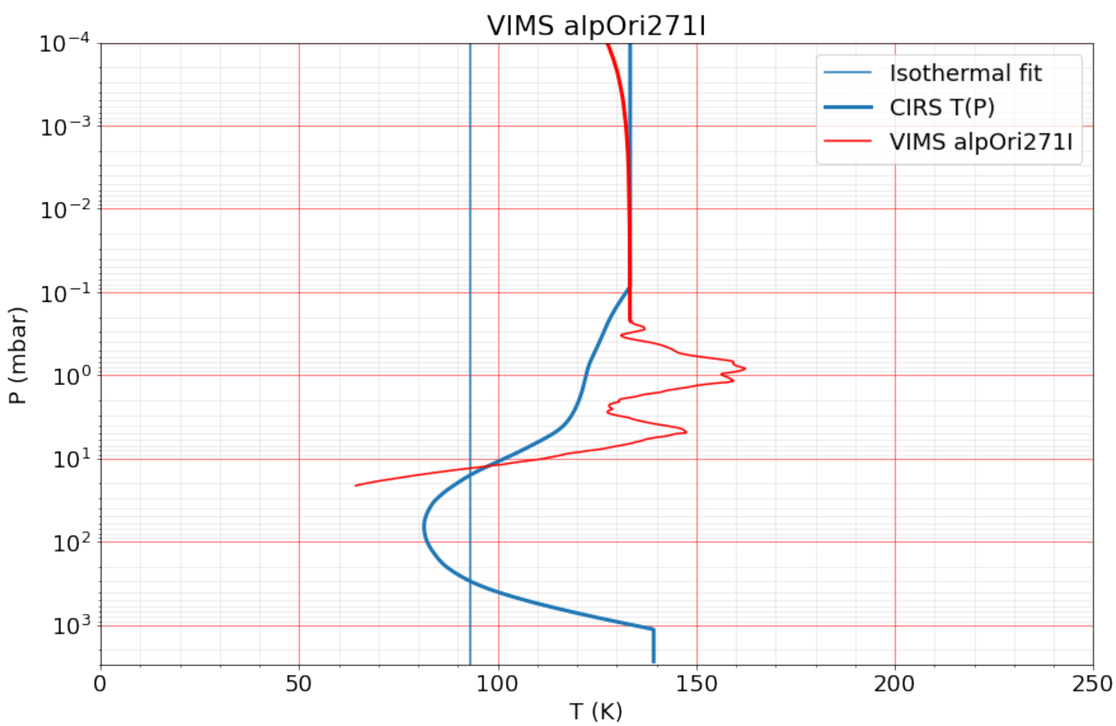
11 isothermal cap

[35]:
```
frame = np.linspace(0,len(counts)-1,len(counts))
L = np.where((frame >= 700) & (frame <= 1400)) # zoom specification
L = np.where((frame >= 750) & (frame <= 1300)) # zoom specification
tsec_all = frame[L]*VIMS['t_cube'] # time in sec from start of data
data_all = counts[L] # the observed lightcurve

# Normalized flux and halflight-relative time:
t_obs = tsec_all-fitted_params.params['t_hl']*u.s
flux_obs = 0.98*(data_all-fitted_params_tau.params['y_bkg'])/fitted_params.
 ↪params['y_scale']
temp = 0.98*(data_all-fitted_params_tau.params['y_bkg'])/fitted_params.
 ↪params['y_scale']

L = np.where(t_obs > (-40)*u.s)
L = np.where(t_obs > (-60)*u.s)
tL = t_obs[L[0]]


H_extrap = TtoH(g,mu,Tofh[-1]*u.K)
thatextrap = t_obs[0:L[0][0]]*abs(VIMS['vperp'])/H_extrap

flux_cap = flux_obs
flux_cap[0:L[0][0]] = get_phi_vals(thatextrap)

vperp_kms = abs(VIMS['vperp'])
Dkm = VIMS['range']
tau_hl = 0.
dh_bin_km = 0.1*u.km # desired altitude resolution
hakm = 600*u.km

#lightcurve to altitude-binned values

profile_flux,profile_time,profile_alt,profile_alt_theta,\
    profile_dtheta,profile_theta,i_tot = \
   ␣
 ↪alt_bin_dimensional(t_obs,flux_cap,dh_bin_km,hakm,vperp_kms,Dkm,tau_hl=tau_hl,Verbose=False

fig,ax=plt.subplots(figsize=(14,9)) # open up figure

plt.plot(t_obs,temp,label=VIMS['event'])

plt.plot(profile_time,profile_flux,label='rebinned')
plt.plot(t_obs[0:L[0][0]],flux_cap[0:L[0][0]],linewidth=5,label='isothermal␣
 ↪cap')
ax.grid(linestyle='-', linewidth='0.5', color='red')
```

```
ax.minorticks_on()
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
plt.legend()
plt.xlabel(r't - t$_{1/2}$ (sec)')
plt.ylabel(r'$\Phi$')
plt.title(VIMS['event'])
plt.show()
# inversion to retrieve refractivity and bending angle

rho_in = (profile_time*VIMS['vperp']+VIMS['r_curv']).value
phi_cyl_in = profile_flux
nu0_in = 0.
D_in = Dkm.value

nu,theta,R = pro_lcinvert(rho_in,phi_cyl_in,nu0_in,D_in)
# conversion to physical units for vertical profile

ncm3_obs,den_obs,Pmbar_obs,T_obs,H_obs = \
    pro_refrac2profile(R*u.km,nu,Tofh[-1]*u.
 ↪K,VIMS['g_ms2'],VIMS['mu'],VIMS['RSTP'])

# plot results

fig,ax=plt.subplots(figsize=(14,9)) # open up figure

T_iso = HtoT(g,mu,fitted_params.params['H_km']*u.km)
plt.axvline(T_iso.value,label='Isothermal fit')
plt.plot(Tofh,Pofh,label='CIRS T(P)',linewidth=3)
plt.plot(T_obs,Pmbar_obs,color='r',label=VIMS['event'])

L = np.where(profile_time < (-60)*u.s)
plt.plot(T_obs[L],Pmbar_obs[L],color='r',linewidth=3) # thick line for␣
 ↪isothermal cap

plt.yscale('log')
plt.ylim(3000,.0001)
plt.xlim(0,250)
plt.ylabel('P (mbar)')
plt.xlabel('T (K)')
plt.legend()
plt.title(VIMS['event'])
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.minorticks_on()
ax.grid(visible=True, which='minor', color='#999999', linestyle='-', alpha=0.2)

plt.show();
```


VIMS alpOri271I
VIMS alpOri271I
rebinned
isothermal cap
Φ
t - t1/2 (sec)

7421


VIMS alpOri271I
Isothermal fit
CIRS T(P)
VIMS alpOri271I
P (mbar)
T (K)

[36]:

```
print('All done!')
```

```
All done!
```