```python
# pip install pyasl

from pathlib import Path
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
import pandas as pd
import lightkurve as lk
from astropy import units as u
from specutils.spectra import Spectrum
from specutils import Spectrum1D
from specutils.fitting.continuum import fit_continuum
from astropy.io import ascii
from PyAstronomy import pyasl
from scipy.signal import correlate
import juliet
from astropy.modeling import models
import csv
from scipy.optimize import curve_fit

DATA = Path.home() / "Experimental" / "Exp_3" / "data"
example = sorted(DATA.glob("ADP*.fits"))[0]
print("Archivo:", example.name)

with fits.open(example) as hdul:
    hdul.info()
    for i, hdu in enumerate(hdul):
        print(f"\n[HDU {i}] EXTNAME = {hdu.header.get('EXTNAME', '(sin nombre)')}")
        for key in list(hdu.header.keys())[:15]:
            print(f"  {key:<20s} {hdu.header[key]}")
```

```
Archivo: ADP.2018-03-28T01:03:35.906.fits
Filename:
/home/2025/AST0421-1/svtroncoso/Experimental/Exp_3/data/ADP.2018-03-28T01:03:35.906.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU    3074   ()
  1  SPECTRUM      1 BinTableHDU     46   1R x 3C   [313115D, 313115E, 313115E]

[HDU 0] EXTNAME = (sin nombre)
  SIMPLE               True
  BITPIX               -32
  NAXIS                0
  EXTEND               True
  COMMENT                FITS (Flexible Image Transport System) format is defined in 'Astronomy
  and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H
```

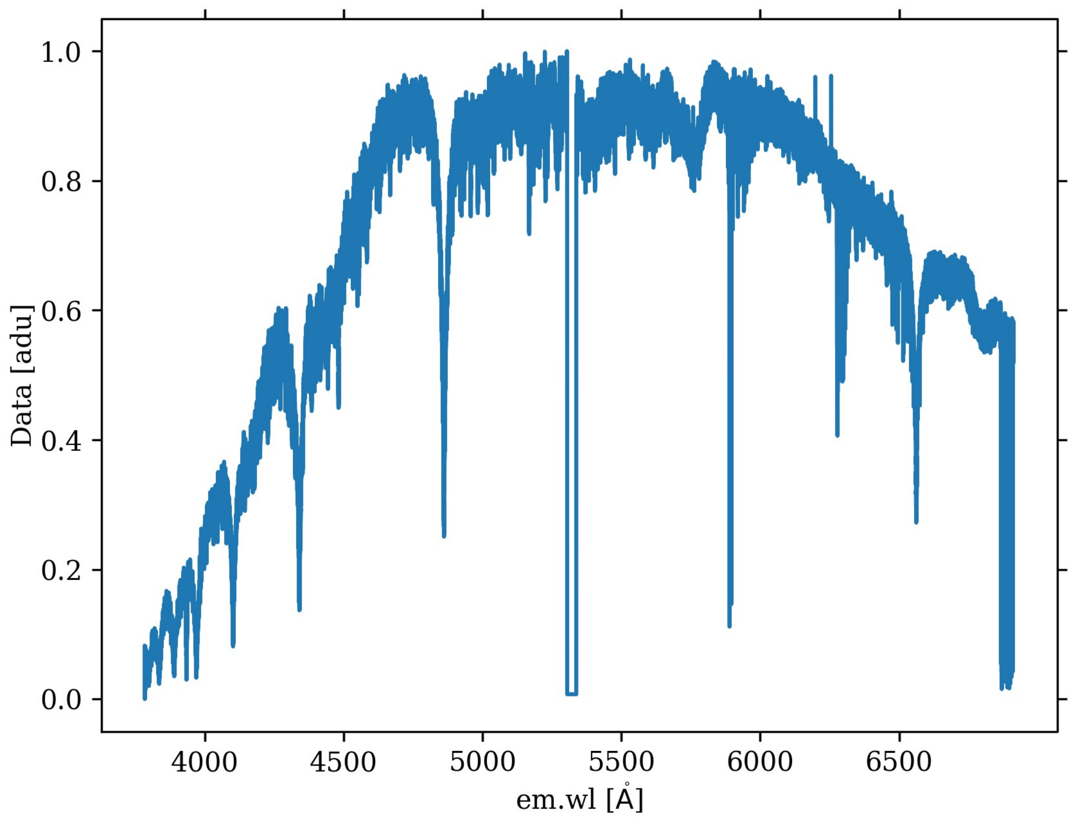
```
  COMMENT                FITS (Flexible Image Transport System) format
is defined in 'Astronomy
  and Astrophysics', volume 376, page 359; bibcode:
2001A&A...376..359H
  DATE                   2018-03-27T04:08:02.174
  INSTRUME               HARPS
  RA                     225.683782
  DEC                    -3.03271
  EQUINOX                2000.0
  RADECSYS               FK5
  EXPTIME                599.9983
  MJD-OBS                58204.16528109
  DATE-OBS               2018-03-27T03:58:00.286

[HDU 1] EXTNAME = SPECTRUM
  XTENSION               BINTABLE
  BITPIX                 8
  NAXIS                  2
  NAXIS1                 5009840
  NAXIS2                 1
  PCOUNT                 0
  GCOUNT                 1
  TFIELDS                3
  TTYPE1                 WAVE
  TFORM1                 313115D
  TTYPE2                 FLUX
  TFORM2                 313115E
  TTYPE3                 ERR
  TFORM3                 313115E
  VOCLASS                SPECTRUM v1.0
```

Lineas de Absorción

```
file = "ADP.2023-05-05T08:25:34.948.fits"

with fits.open(DATA / file, memmap=True) as hdul:
    tab = hdul["SPECTRUM"].data
    wave = np.asarray(tab["WAVE"][0], dtype=float)
    flux = np.asarray(tab["FLUX"][0], dtype=float)

if np.nanmax(wave) < 1000:
    # wave = wave * 10.0tab
    wave, flux = wave[::5], flux[::5]
    plt.figure(figsize=(6,3.5))
    plt.plot(wave, flux, lw=0.6)
    plt.xlabel("Wavelength (Å)")
    plt.ylabel("Flux")
    plt.title(file)
```

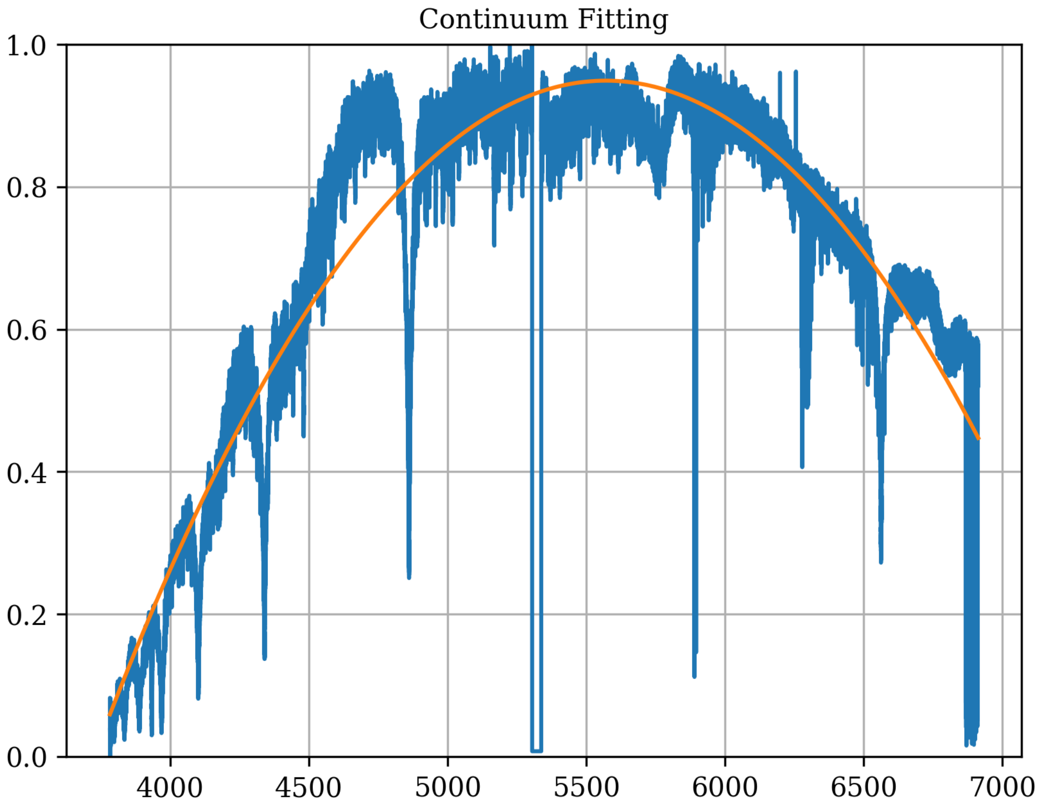
```
    plt.tight_layout()
    plt.show()

flux_n = (flux - np.nanmin(flux)) / (np.nanmax(flux) - 
np.nanmin(flux))
spec = Spectrum(spectral_axis=wave*u.AA, flux=flux_n*u.adu)

spec.plot()
```

```
<WCSAxes: ylabel='Data [adu]'>
```



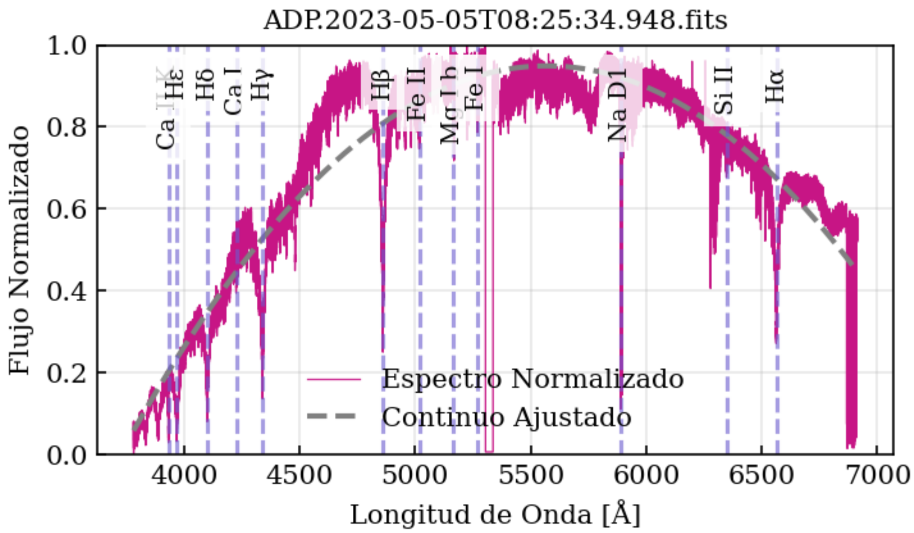

```
region = [(3782.63 * u.AA, 5300 * u.AA), (5350 * u.AA, 6912.88 * 
u.AA)]
# region = [(4000 * u.AA, 5000 * u.AA)]
fitted_continuum = fit_continuum(spec, window=region, 
model=models.Polynomial1D(degree=2))
y_continuum_fitted = fitted_continuum(spec.spectral_axis)

f, ax = plt.subplots()
ax.plot(spec.spectral_axis, spec.flux)
ax.plot(spec.spectral_axis, y_continuum_fitted)
ax.set_title("Continuum Fitting")
ax.grid(True)
ax.set_ylim(0,1)
```

```
(0.0, 1.0)
```




```
LINES = [
    ("Ca II K", 3933.66),
    ("Hε", 3970.07),
    ("Hδ", 4101.74),
    ("Ca I", 4226.73),
    ("Hγ", 4340.47),
    ("Hβ", 4861.33),
    ("Fe II", 5018.44),
    ("Mg I b", 5167.32),
    ("Fe I", 5270.00),
    ("Na D1", 5889.95),
    ("Si II", 6347.10),
    ("Hα", 6562.80),
]

plt.rcParams.update({
    "font.family": "serif",
    "font.size": 10,
    "axes.labelsize": 10,
    "axes.titlesize": 10,
    "xtick.labelsize": 10,
    "ytick.labelsize": 10,
```

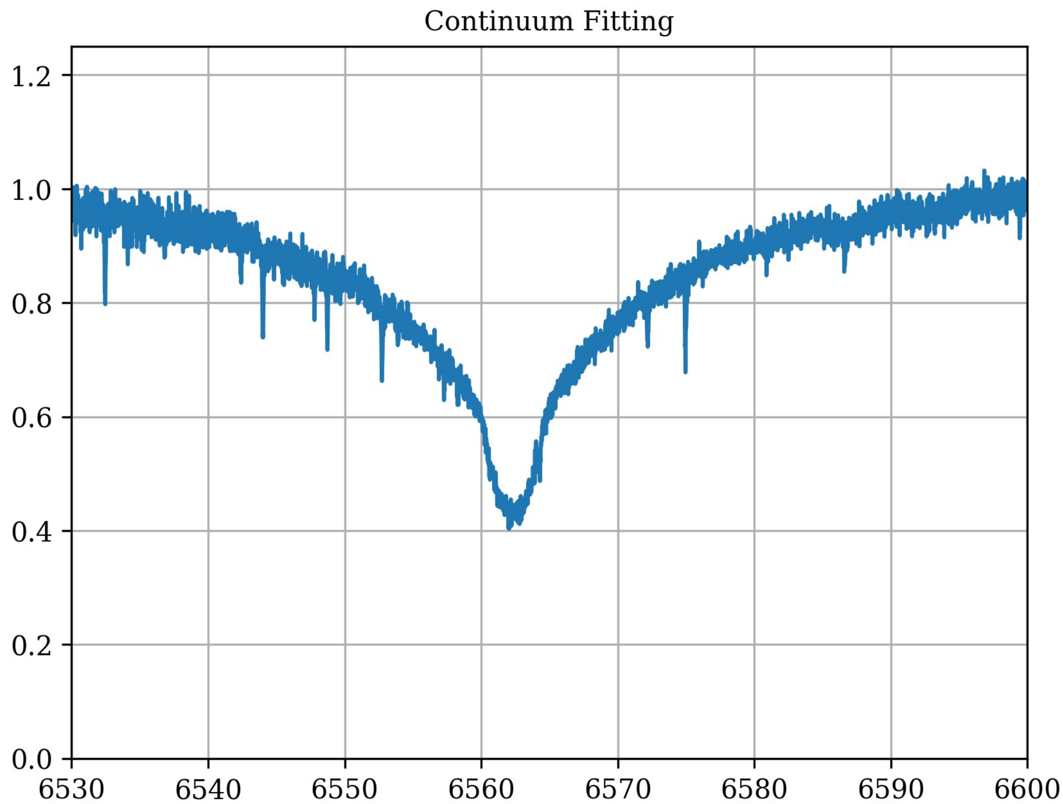
```python
    "legend.fontsize": 10,
    "figure.dpi": 300,
    "axes.linewidth": 0.8,
    "xtick.major.width": 0.8,
    "ytick.major.width": 0.8
})

f, ax = plt.subplots(figsize=(5, 3), dpi=150)

ax.plot(spec.spectral_axis, spec.flux, lw=0.6,
color="mediumvioletred", label="Espectro Normalizado")
ax.plot(spec.spectral_axis, y_continuum_fitted, color="gray", ls='--',
lw=2, label="Continuo Ajustado")

yflux = spec.flux.value if hasattr(spec.flux, "value") else spec.flux
spec_axis_vals = spec.spectral_axis.value if
hasattr(spec.spectral_axis, "value") else spec.spectral_axis
ymax = np.nanpercentile(yflux, 99)

for name, wl in LINES:
    wl_val = float(wl)
    if spec_axis_vals.min() <= wl_val <= spec_axis_vals.max():
        ax.axvline(wl_val, color='slateblue', ls='--', alpha=0.6)
        ax.text(
            wl_val, ymax, name,
            rotation=90, va="top", ha="center", fontsize=9,
            bbox=dict(facecolor="white", edgecolor="none", alpha=0.8)
        )

file = "ADP.2023-05-05T08:25:34.948.fits"
ax.set_title(file)
ax.set_xlabel("Longitud de Onda [Å]")
ax.set_ylabel("Flujo Normalizado")
ax.set_ylim(0, 1)
ax.grid(True, alpha=0.3)
ax.legend(frameon=False)

plt.tight_layout()
plt.show()
```



Velocidades Radiales

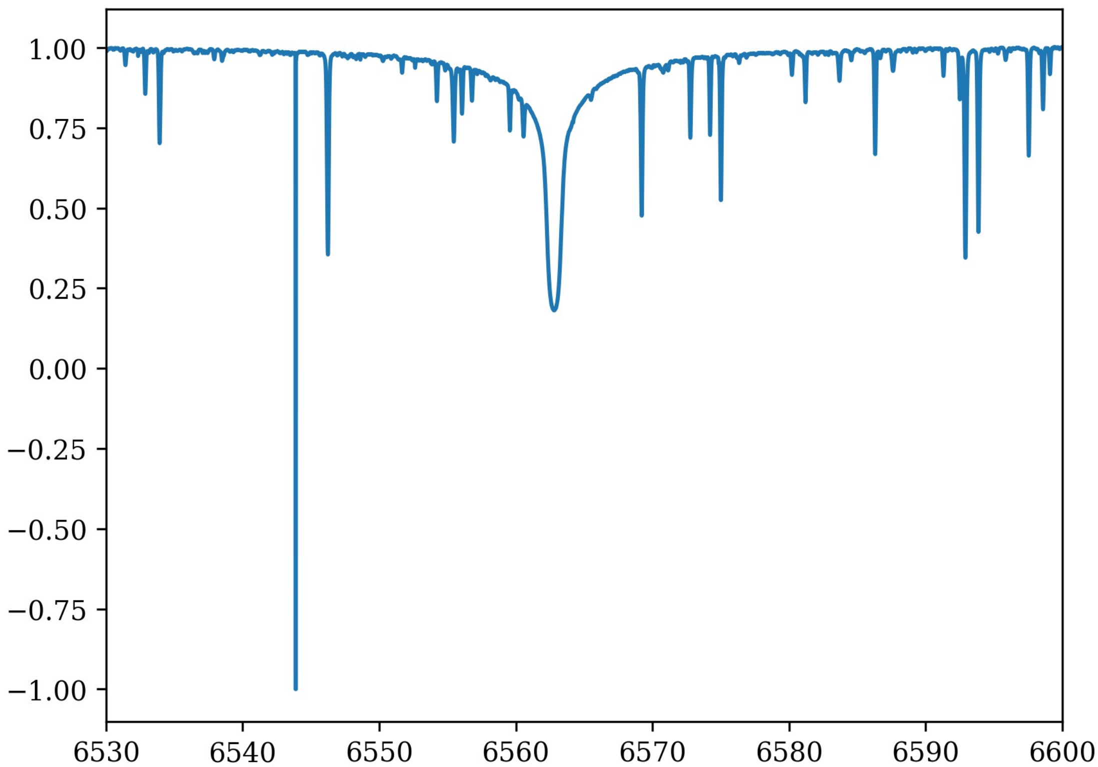
```
spec_normalized = spec.flux / y_continuum_fitted
spec_normalized[spec_normalized>2] = 2
spec_normalized[spec_normalized<0] = 0

f, ax = plt.subplots()
# ax.plot(spec.spectral_axis, spec.flux)
ax.plot(spec.spectral_axis, spec_normalized)
ax.set_title("Continuum Fitting")
ax.grid(True)
plt.xlim(6530,6600)
plt.ylim(0,1.25)
```

```
(0.0, 1.25)
```




```
template = ascii.read("template_sintetico.txt")
# template

plt.plot(template['waveobs']*10,template['flux'])
plt.xlim(6530,6600)
```

```
(6530.0, 6600.0)
```

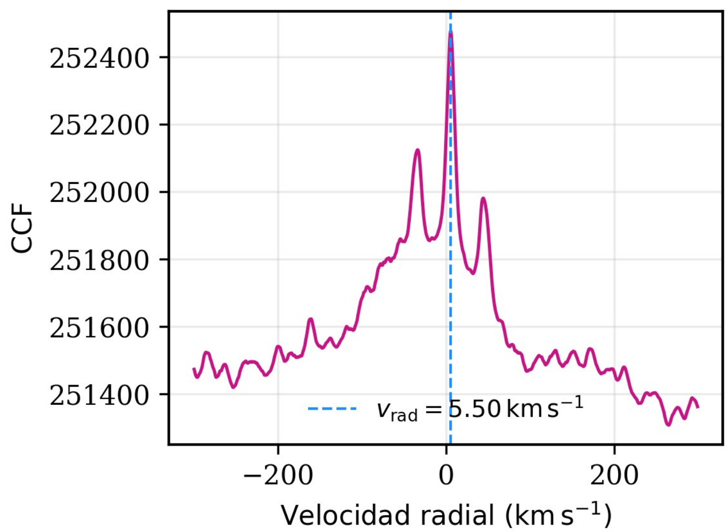



```
wave_obs = spec.spectral_axis.to(u.AA).value
flux_obs = spec.flux.value / np.median(spec.flux.value)
wave_tpl = template['waveobs'] * 10
flux_tpl = template['flux'] / np.median(template['flux'])


rv_range = [-300., 300.]
step = 0.5
rv, ccf = pyasl.crosscorrRV(
    wave_obs, flux_obs, wave_tpl, flux_tpl,
    rv_range[0], rv_range[1], step, skipedge=20
)

rv_peak = rv[np.argmax(ccf)]
ccf_max = np.max(ccf)
print(f"Velocidad radial (pico CCF): {rv_peak:.2f} km/s")
```

```
Velocidad radial (pico CCF): 5.50 km/s
```

```
plt.rcParams.update({
    "font.family": "serif",
    "font.size": 10,
    "axes.labelsize": 10,
    "axes.titlesize": 10,
    "xtick.labelsize": 10,
    "ytick.labelsize": 10,
```

```
    "axes.linewidth": 1.0,
})

fig, ax = plt.subplots(figsize=(4, 3))
ax.plot(rv, ccf, lw=1.2, color="mediumvioletred")
ax.axvline(rv_peak, color="dodgerblue", lw=1.0, ls="--", label=fr"$v_\
mathrm{{rad}}={rv_peak:.2f}\,\mathrm{{km\,s^{{-1}}}}$")
ax.set_xlabel(r"$\mathrm{Velocidad\ radial\ (km\,s^{-1})}$")
ax.set_ylabel(r"$\mathrm{CCF}$")
ax.legend(frameon=False, fontsize=9)
ax.grid(alpha=0.25)
plt.tight_layout()
plt.show()
```




```
# # Capetas entrada y salida
# source_folder = 'Exp_3/data'
# output_folder = 'norm_spectra_2'
# snr_threshold = 80
# filtered_fits_names = []

# if os.path.exists(output_folder):
#     # 2. Si existe, la borra completamente
#     print(f"Eliminando carpeta existente: '{output_folder}'...")
#     shutil.rmtree(output_folder)
# # 3. Vuelve a crear la carpeta, ahora limpia y vacía
# print(f"Creando carpeta limpia: '{output_folder}'...")
# os.makedirs(output_folder)


# for nombre_archivo in os.listdir('Exp_3/data'):
#     if nombre_archivo.endswith('.fits'):
#         # Abrimos el .fits
```

```
#         source_path = os.path.join(source_folder, nombre_archivo)
#         output_path = os.path.join(output_folder, nombre_archivo)

#         with fits.open(source_path) as hdul:
#             snr = hdul[0].header.get('SNR')

#             if snr >= snr_threshold:
#                 # Abrimos los fits, para normalizar los espectros
#                 spec = Spectrum1D(spectral_axis=hdul[1].data['WAVE']
[0]*u.AA,
#                               flux=hdul[1].data['FLUX'][0]*u.adu)

#                 flux = spec.flux.value
#                 wave = spec.spectral_axis.value
#                 flux_err = np.sqrt(flux)

#                 # Hay espectros con flujos muy bajos, estos hay que
quitarlos
#                 if np.nanmedian(flux) > 10:

#                     region = [(3500 * u.AA, 5000 * u.AA), (5500 *
u.AA, 7000 * u.AA)]
#                     fitted_continuum = fit_continuum(spec,
window=region)
#                     y_continuum_fitted =
fitted_continuum(spec.spectral_axis)

#                     # Vemos en donde el continuo es cero o negativo
y enmascaramos:
#                     posiciones = [indice for indice, numero in
enumerate(y_continuum_fitted.value) if numero <= 0]

                    # # Si el continuo nunca fue <= 0, no enmascaramos
nada
                    # if len(posiciones) > 0:
                    #     mask = wave > wave[posiciones[-1]]
                    # else:
                    #     mask = np.ones_like(wave, dtype=bool)  # no
se aplica corte

                    # wave = wave[mask]
                    # flux = flux[mask]
                    # y_continuum_fitted = y_continuum_fitted[mask]
                    # flux_err = flux_err[mask]


                    # norm_flux = (flux / y_continuum_fitted).value
                    # norm_flux_err = flux_err / y_continuum_fitted

                    # norm_flux_clipped = np.clip(norm_flux, 0, 2)
```

```
                    # # Creacion nuevos fits
                    # col_wave = fits.Column(name='WAVE', format='E',
array=wave, unit='Angstrom')
                    # col_flux = fits.Column(name='NORM_FLUX',
format='E', array=norm_flux_clipped)
                    # col_err = fits.Column(name='NORM_FLUX_ERR',
format='E', array=norm_flux_err)
                    # # Creamos el objeto HDU (Header/Data Unit) de la
tabla binaria
                    # normalized_hdu =
fits.BinTableHDU.from_columns([col_wave, col_flux, col_err])
                    # normalized_hdu.name = 'NORMALIZED_DATA' # Le
damos un nombre para identificara
                    # # --- C. Ensamblaje y guardado del nuevo archivo
FITS ---
                    # # Creamos una nueva lista de HDUs: el primario
original + la nueva tabla
                    # # (Podrías añadir también hdul[1] si quieres
conservar la tabla original)
                    # new_hdul = fits.HDUList([hdul[0],
normalized_hdu])
                    # # Guardamos el nuevo archivo en la carpeta de
salida
                    # new_hdul.writeto(output_path, overwrite=True)
                    # # Guardamos el nombre del archivo en nuestra
lista de seguimiento
                    # filtered_fits_names.append(nombre_archivo)


# snr_fits = []
# mjd_fits = []
# for nombre_archivo in os.listdir('norm_spectra_2'):
#     if nombre_archivo.endswith('.fits'):
#         # Abrimos el .fits
#         path = 'norm_spectra_2/' + nombre_archivo
#         with fits.open(path) as hdul:
#             # Guardamos la SNR de los .fits
#             snr_fits.append(hdul[0].header['SNR'])
#             # Guardamos la MJD-OBS de los fits:
#             mjd_fits.append(hdul[0].header['MJD-OBS'])

# snr_fits = np.array(snr_fits)
# mjd_fits = np.array(mjd_fits)

# # Lisa de radial velocities obtenidas de la cc
# rv_mean = []
# rv_median = []
# rv_err = []
# files_to_process = [fit for fit in os.listdir('norm_spectra_2') if
```

```
fit.endswith('.fits')]

# for nombre_archivo in tqdm(files_to_process, desc="Procesando
archivos FITS"):

#     if nombre_archivo.endswith('.fits'):
#         # Abrimos el .fits
#         try:
#             path = os.path.join('norm_spectra_2/', nombre_archivo)
#             with fits.open(path) as hdul:
#                 # Extraemos los datos de espectros
#                 datos_normalizados = hdul[1].data

#                 wave = datos_normalizados['WAVE'] # Angstrom
#                 norm_flux = datos_normalizados['NORM_FLUX']
#                 norm_flux_err = datos_normalizados['NORM_FLUX_ERR']

#                 # Inicio montecarlo
------------------------------------
#                 rvs_simulations     = []
#                 number_simulations = 50

#                 # Simulacion del mismo espectro con error extra
gaussiano
#                 for i in tqdm(range(number_simulations), desc=f"MC
para {nombre_archivo[:10]}...", leave=False):
#                     gaussian_noise = np.random.normal(0, 1,
size=len(norm_flux))

#                     new_flux = norm_flux + (gaussian_noise *
norm_flux_err)
#                     new_flux_clip = np.clip(new_flux, 0, 2)

#                     # Calcular las RV de nuevo espectro
#                     rv, cc = pyasl.crosscorrRV(wave, new_flux_clip,
#                                                            template['wave_A'],
template['flux'],
#                                                        -40., -5., 20./100.,
skipedge=10)

#                     # 1. Encuentra el índice del píxel con el valor
más alto
#                     max_ind = np.argmax(cc)

                    # 2. Comprobación de seguridad: asegúrate de que
el pico no esté en el borde
        #             if max_ind > 1 and max_ind < len(rv) - 2:
        #                 # 3. Selecciona los puntos alrededor del
pico (ej. 5 puror ocurre si el array datos_normalizados['WAVE'] tiene
```

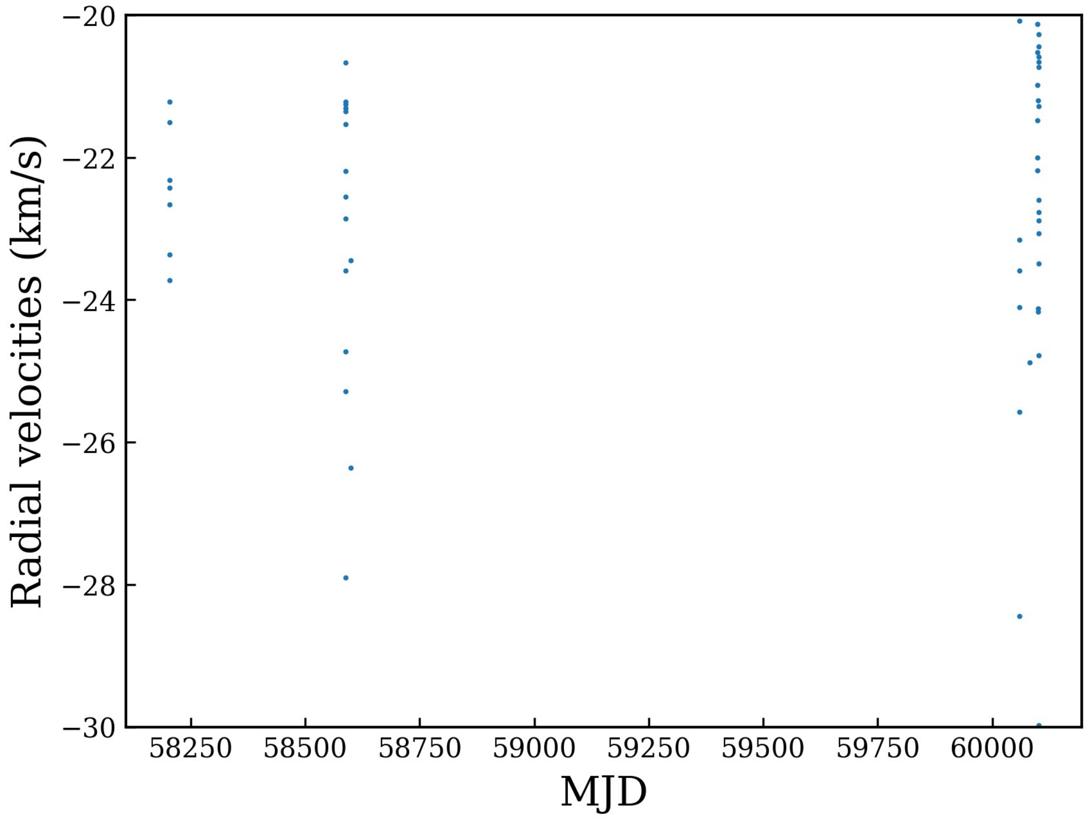
```
menos de 14,044 elementos.ntos)
        #                 rv_peak = rv[max_ind-2 : max_ind+3]
        #                 cc_peak = cc[max_ind-2 : max_ind+3]

        #                 # 4. Ajusta un polinomio de 2º grado (una
parábola)
        #                 p = np.polyfit(rv_peak, cc_peak, 2)

        #                 # 5. La posición del vértice de la parábola
(-b / 2a) es la RV de alta precisión
        #                 rv_max_simulations = -p[1] / (2 * p[0])
        #             else:
        #                 # Si el pico está en el borde, usa el método
simple como respaldo
        #                 rv_max_simulations = rv[max_ind]

        #             #
======================================================================
=
        #             # FIN: BÚSQUEDA DEL PICO
        #             #
======================================================================
=

        #             rvs_simulations.append(rv_max_simulations)

        #         # Fin montecarlo
-----------------------------------------
        #         # Calcular Resultados y guardarlos: Media/mediana y
dsv std
        #         rv_media   = np.nanmean(rvs_simulations)
        #         rv_mediana = np.nanmedian(rvs_simulations)
        #         rv_std     = np.nanstd(rvs_simulations)

        #         rv_mean.append(rv_media)
        #         rv_median.append(rv_mediana)
        #         rv_err.append(rv_std)

        # except IndexError as e:
        #     print(f"\n⚠  Error de índice en '{nombre_archivo}'.
Causa: {e}. Se guardará como NaN.")

        #     # Guardamos NaN en nuestras listas para este espectro
fallido
        #     rv_mean.append(np.nan)
        #     rv_median.append(np.nan)
        #     rv_err.append(np.nan)

# datos = {
#     'rv_mean' : rv_mean,
```

```python
#     'rv_median' : rv_median,
#     'rv_err' : rv_err,
#     'snr' : snr_fits,
#     'mjd' : mjd_fits,
#     'fit_name' : filtered_fits_names

# }
# # Crear dataframe
# df = pd.DataFrame(datos)
# # Guardar dataframe como .txt
# df.to_csv('rv_data.txt', index=False, sep=';')

def read_txt_as_df(path):
    try:
        return pd.read_csv(path, sep=None, engine='python',
comment='#', header='infer')
    except Exception:
        return pd.read_csv(path, delim_whitespace=True, comment='#',
header='infer')

# uso (solo pones el path):
rv_data = read_txt_as_df("rv_data.txt")
rv_data.head()
```

```
     rv_mean  rv_median    rv_err     snr           mjd  \
0 -16.865279 -16.315657  3.059075  108.35  60101.121344
1 -16.453570 -14.582778  5.758718  109.10  60101.118659
2 -15.659700 -15.628974  1.307610   99.60  60099.222916
3 -20.136346 -18.476785  4.560771  119.10  60101.129400
4 -18.179407 -18.203441  2.628480  126.45  60101.126715

                           fit_name
0  ADP.2023-06-07T01:04:24.663.fits
1  ADP.2023-06-07T01:04:24.661.fits
2  ADP.2023-06-05T01:02:22.459.fits
3  ADP.2023-06-07T01:04:24.669.fits
4  ADP.2023-06-07T01:04:24.667.fits
```

```python
plt.scatter(rv_data['mjd'], rv_data['rv_median'], s = 0.9)
plt.ylim(-30,-20)
# plt.xlim(60000, 60200)

plt.xlabel('MJD'                          , fontsize = 15)
plt.ylabel('Radial velocities (km/s)', fontsize = 15)
```

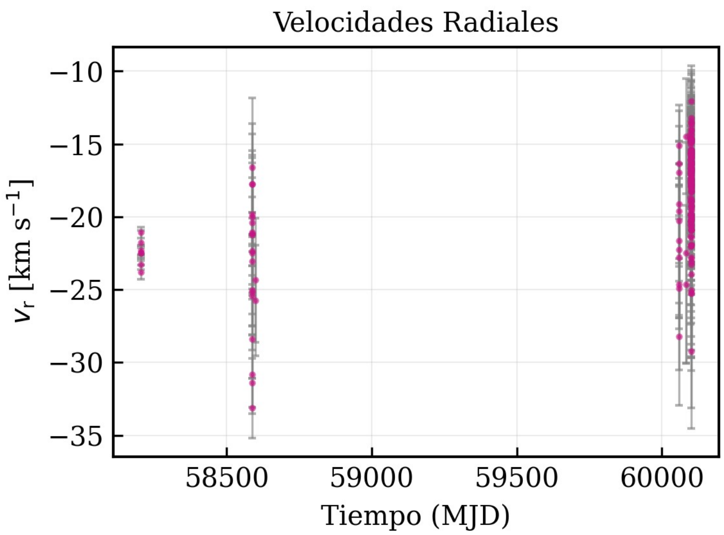
```
Text(0, 0.5, 'Radial velocities (km/s)')
```




```
vr_median = np.median(rv_data["rv_mean"])
vr_median
```

```
-17.481542760103693
```

```
fig, ax = plt.subplots(figsize=(4, 3))

ax.errorbar(rv_data["mjd"], rv_data["rv_mean"],
            yerr=rv_data["rv_err"], fmt='.',
            ms=3, lw=1.2, capsize=1.5, elinewidth=0.8,
            mfc="mediumvioletred", mec="mediumvioletred",
            ecolor="gray", alpha=0.6)

ax.set_xlabel("Tiempo (MJD)")
ax.set_ylabel(r"$v_\mathrm{r}$ [km s$^{-1}$]")
ax.set_title("Velocidades Radiales")
ax.grid(alpha=0.25, lw=0.5)

plt.tight_layout()
plt.show()
```



Curvas de Luz

```
search_result = lk.search_lightcurve('wasp-189')
```

```
search_result
```

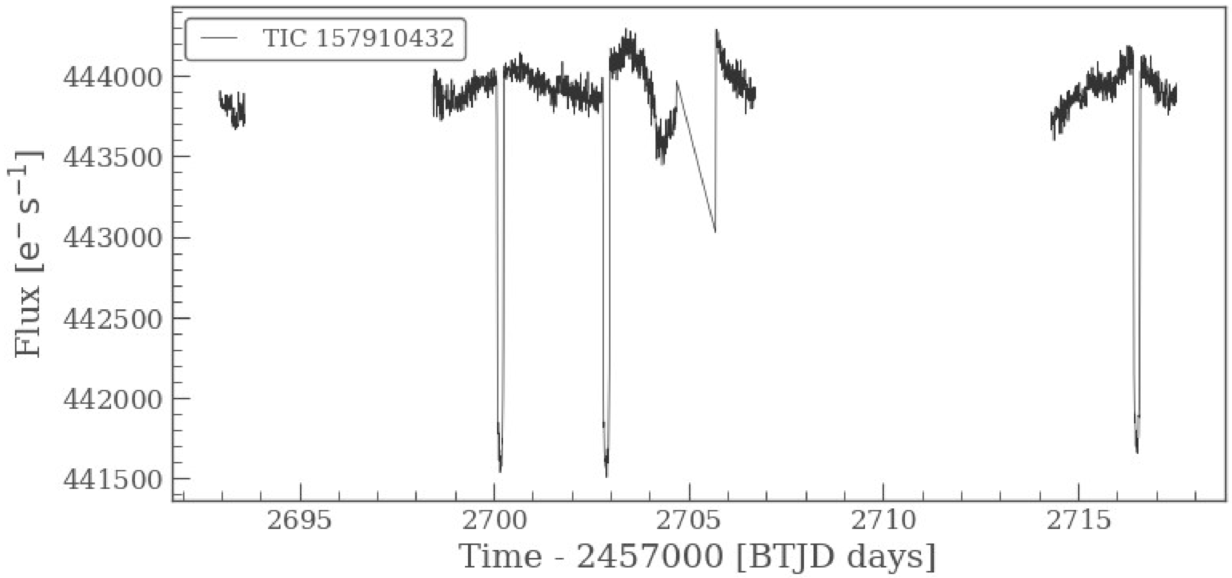
```
SearchResult containing 3 data products.

 #     mission     year   author   exptime target_name distance
                                      s                 arcsec
--- -------------- ---- --------- ------- ----------- --------
  0 TESS Sector 51 2022      SPOC     120   157910432      0.0
  1 TESS Sector 51 2022 TESS-SPOC     600   157910432      0.0
  2 TESS Sector 51 2022       QLP     600   157910432      0.0
```

```
lc = search_result[1].download()
```

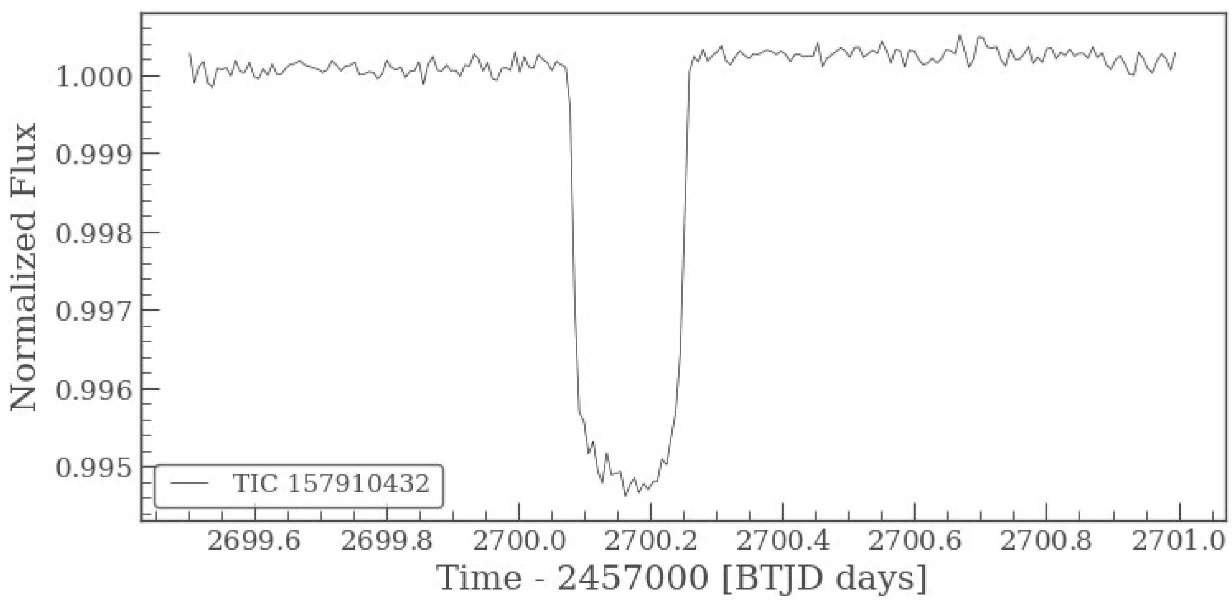
```
Warning: 27% (928/3399) of the cadences will be ignored due to the
quality mask (quality_bitmask=17087).
```

```
lc.plot()
```

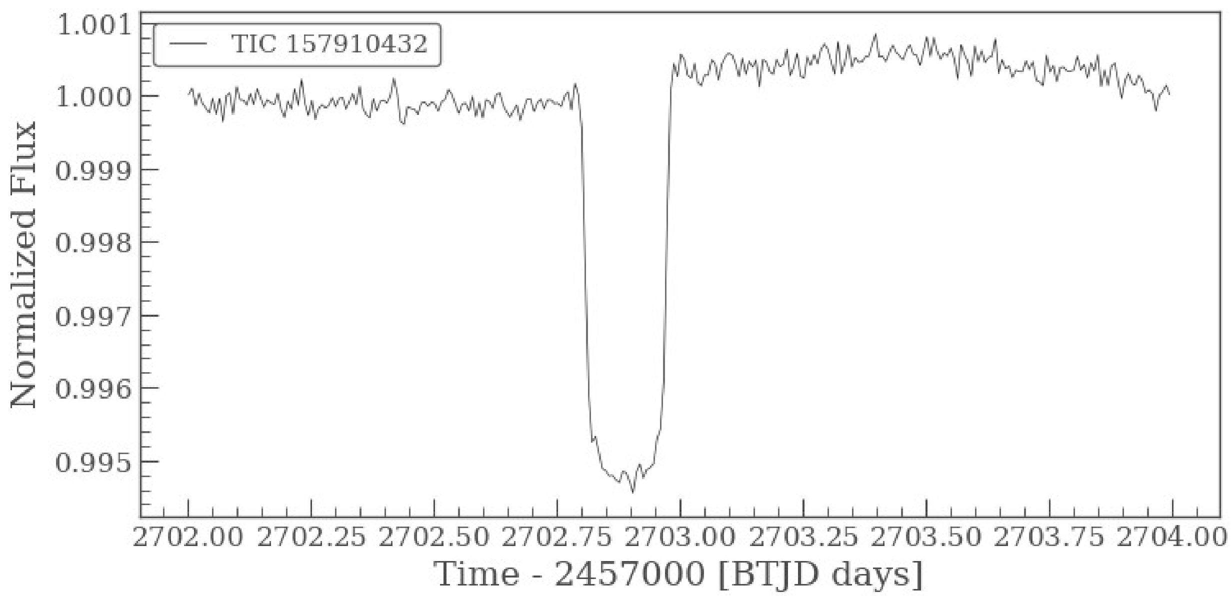
```
<Axes: xlabel='Time - 2457000 [BTJD days]', ylabel='Flux [$\\
mathrm{e^{-}\\,s^{-1}}$]'>
```



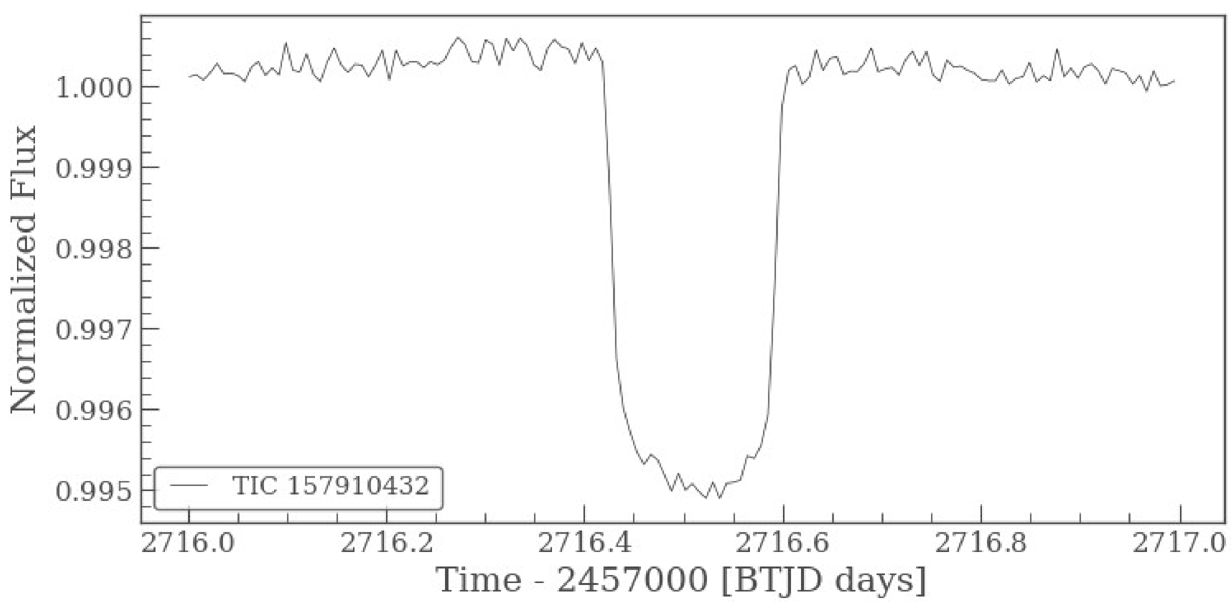

```
lc.normalize().truncate(2699.5,2701).plot() #1
plt.show()
```




```
lc.normalize().truncate(2702, 2704).plot() #2
plt.show()
```




```
lc.normalize().truncate(2716, 2717).plot() #3
plt.show()
```



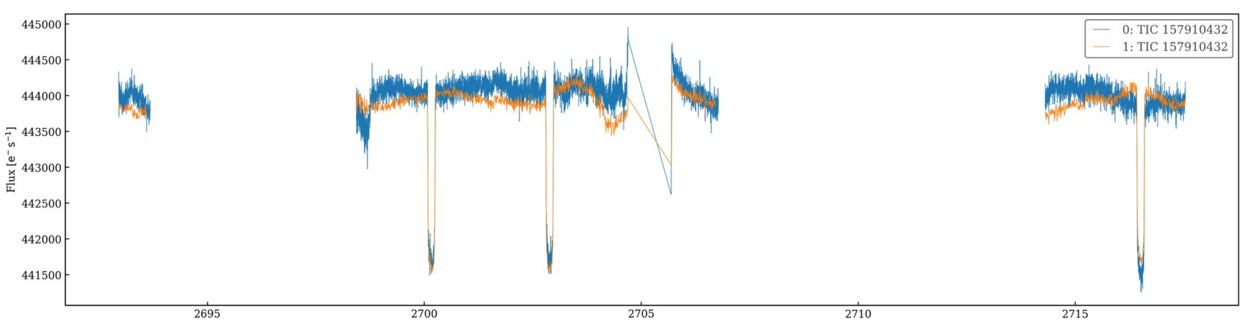

```
lc_1 = lc.normalize().truncate(2699.5, 2701)
lc_1

<TessLightCurve length=215 LABEL="TIC 157910432" SECTOR=51
AUTHOR=TESS-SPOC FLUX_ORIGIN=pdcsap_flux>
       time             flux       ...   pos_corr1      pos_corr2
                                   ...      pix            pix
       Time           float32      ...    float32        float32
------------------ -------------- ... -------------- --------------
```

```
2699.5021574476364  1.0002649e+00 ... -6.4754248e-02  7.6476760e-02
2699.5091019657407  9.9989229e-01 ... -6.4245030e-02  7.7144787e-02
 2699.516046483844  1.0000969e+00 ... -6.2825188e-02  7.6338612e-02
 2699.522991001949  1.0001659e+00 ... -6.4412437e-02  7.5792171e-02
 2699.529935519587  9.9988604e-01 ... -6.4355113e-02  7.7376850e-02
2699.5368800376905  9.9983931e-01 ... -6.3696086e-02  7.4951410e-02
 2699.543824555794  1.0000809e+00 ... -6.4481422e-02  7.6697752e-02
2699.5507690734325  1.0000664e+00 ... -6.4201601e-02  7.7072777e-02
 2699.557713591071  1.0000873e+00 ... -6.3515633e-02  7.4295737e-02
               ...            ... ...            ...            ...
2700.9396716348424  1.0002862e+00 ... -5.4408737e-02  6.9424495e-02
2700.9466161427013  1.0002085e+00 ... -5.5566389e-02  6.9606759e-02
2700.9535606505606  1.0000764e+00 ... -5.6059908e-02  6.9006465e-02
2700.9605051579542  1.0000252e+00 ... -5.5285640e-02  6.9091402e-02
2700.9674496653474  1.0001743e+00 ... -5.4640129e-02  6.7254581e-02
 2700.974394173207  1.0002661e+00 ... -5.5420782e-02  6.7602776e-02
2700.9813386806013  1.0001987e+00 ... -5.5837434e-02  6.8498068e-02
2700.9882831879954  1.0000554e+00 ... -5.5407129e-02  6.9554284e-02
 2700.995227695388  1.0002754e+00 ... -5.5329349e-02  6.8305239e-02
```

```
lc_collection = search_result[[0,1]].download_all()
```

```
Warning: 36% (6058/16999) of the cadences will be ignored due to the
quality mask (quality_bitmask=17087).
Warning: 27% (928/3399) of the cadences will be ignored due to the
quality mask (quality_bitmask=17087).
```

```
# Create a larger figure for clarity
fig, ax = plt.subplots(figsize=(20,5))
# Plot the light curve collection
lc_collection.plot(ax=ax);
plt.show()
```

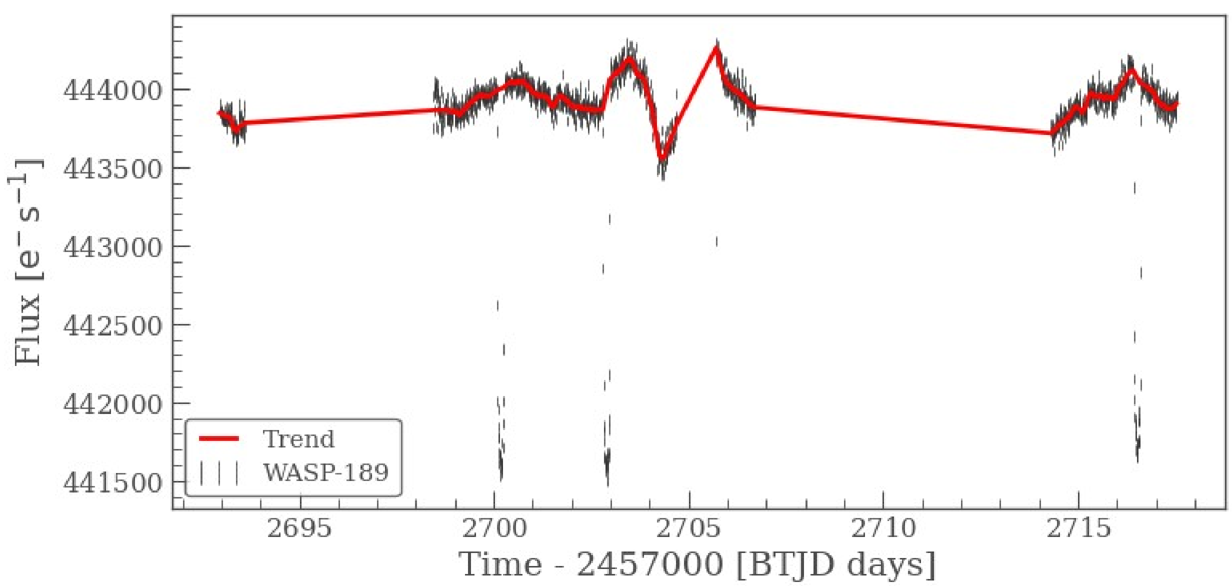



```
# import lightkurve as lk
# import numpy as np
# import astropy.units as u

# target = "TIC 157910432"   # o tu TIC/TOI
# coll = lk.search_lightcurve(target, author="lc1-SPOC").download_all(
```

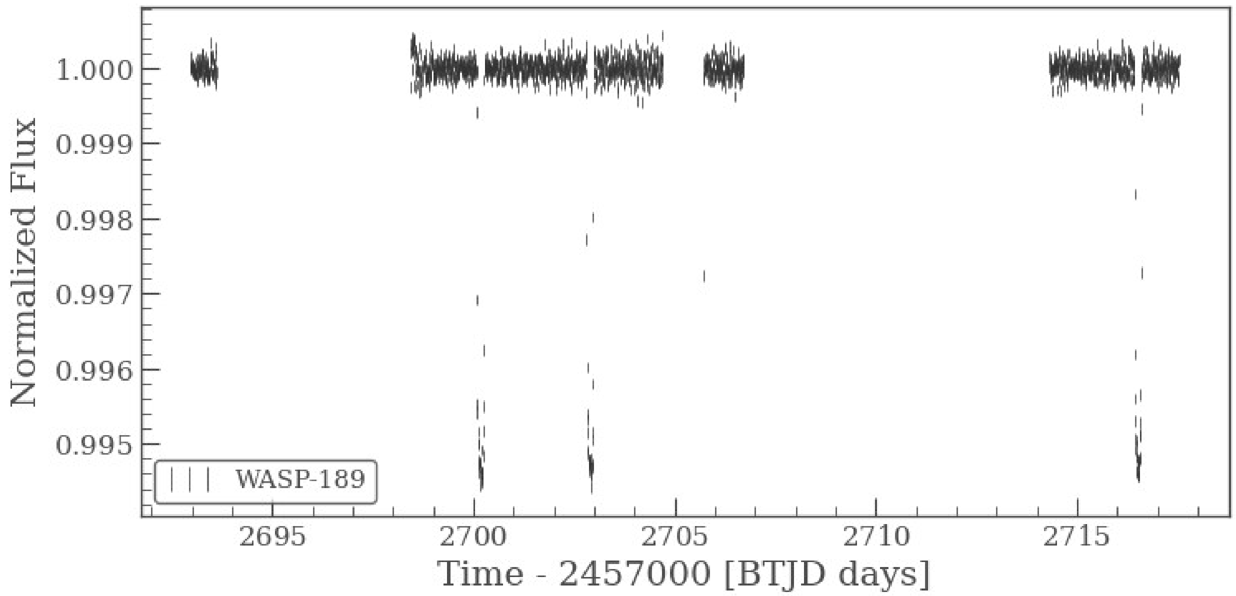
```
#     flux_column="pdcsap_flux"   # <- fuerza PDCSAP
# )
# lc = coll.stitch().remove_nans()
# lc

# cadencia en minutos (el tiempo está en días BTJD)
cad_min = np.nanmedian(np.diff(lc.time.value))*24*60

# queremos ~8 horas de ventana:
ventana_horas = 8
window_length = int(round((ventana_horas*60) / cad_min))

# debe ser impar y >= 3
if window_length % 2 == 0:
    window_length += 1
window_length = max(window_length, 3)

window_length
```

```
49
```

```
# lc_stitched = lc_collection.stitch()

flat, trend = lc.flatten(window_length=49, return_trend=True)

ax = lc.errorbar(label="WASP-189")                     # plot() returns
a matplotlib axes ...
trend.plot(ax=ax, color='red', lw=2, label='Trend');  # which we can
pass to the next plot() to use the same axes
plt.show()
```



```
flat.errorbar(label="WASP-189")
plt.show()
```




```
flat
```

```
<TessLightCurve length=2471 LABEL="TIC 157910432" SECTOR=51
AUTHOR=TESS-SPOC FLUX_ORIGIN=pdcsap_flux>
       time              flux       ...   pos_corr1      pos_corr2
                                    ...      pix            pix
       Time            float64      ...    float32        float32
------------------ -------------- ... -------------- --------------
2692.9604063741494  1.0000071e+00 ... -6.9402784e-02  6.3055903e-02
 2692.967350910415  1.0001523e+00 ... -6.9297031e-02  6.2495850e-02
2692.9742954466806  1.0000189e+00 ... -7.1495153e-02  6.4111710e-02
2692.9812399829457  1.0000406e+00 ... -7.1231581e-02  6.4287752e-02
 2692.988184519211  9.9999551e-01 ... -7.1345359e-02  6.1678298e-02
 2692.995129055476  9.9993417e-01 ... -7.3348403e-02  6.5978348e-02
2693.0020735917406  1.0000345e+00 ... -7.2299123e-02  6.1374273e-02
 2693.009018128005  1.0000123e+00 ... -7.3061965e-02  6.3946456e-02
2693.0159626642703  1.0000618e+00 ... -7.2574303e-02  6.1784334e-02
               ...            ... ...            ...            ...
 2717.481236303237  1.0000469e+00 ... -2.5651811e-02 -8.0795579e-02
 2717.488180627159  9.9997218e-01 ... -2.4394184e-02 -8.1052974e-02
 2717.495124950616  9.9983492e-01 ... -2.3944166e-02 -8.0902517e-02
 2717.502069273607  1.0000961e+00 ... -2.6194107e-02 -8.0844231e-02
 2717.509013596599  9.9991424e-01 ... -2.4427896e-02 -8.1940465e-02
2717.5159579191245  1.0000810e+00 ... -2.2886200e-02 -8.1285439e-02
  2717.52290224165  9.9985548e-01 ... -2.3216257e-02 -8.2897201e-02
 2717.529846564176  1.0000946e+00 ... -2.3270033e-02 -8.2283966e-02
 2717.536790886236  1.0001189e+00 ... -2.2792827e-02 -8.3633140e-02
```

```
lc_time = flat["time"].mjd
flat["time"] = flat["time"].mjd
len(lc_time)
```

```
2471
```

```
flat.to_fits(path='WASP189.fits', overwrite=True)
```

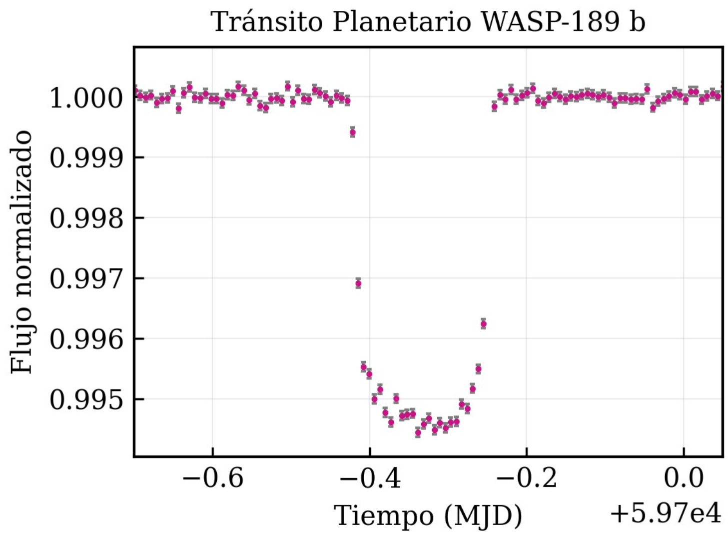
```
tab = fits.open('WASP189.fits')[1].data
tab
```

```
FITS_rec([(59692.46040637, 1.000007  , 7.3913587e-05, 211031,
1995.01925749, 1357.2774774 , 0),
          (59692.46735091, 1.0001522 , 7.3932526e-05, 211032,
1995.01939752, 1357.27704759, 0),
          (59692.47429545, 1.0000188 , 7.3944328e-05, 211033,
1995.01743782, 1357.27895773, 0),
          ...,
          (59717.02290224, 0.99985546, 7.3283089e-05, 214568,
1995.0588015 , 1357.14133689, 0),
          (59717.02984656, 1.0000945 , 7.3293639e-05, 214569,
1995.05939421, 1357.14205925, 0),
          (59717.03679089, 1.0001189 , 7.3294184e-05, 214570,
1995.05978023, 1357.14086664, 0)],
         dtype=(numpy.record, [('TIME', '>f8'), ('FLUX', '>f4'),
('FLUX_ERR', '>f4'), ('CADENCENO', '>i4'), ('MOM_CENTR1', '>f8'),
('MOM_CENTR2', '>f8'), ('QUALITY', '>i4')]))
```

```
tab['TIME']
```

```
array([59692.46040637, 59692.46735091, 59692.47429545, ...,
       59717.02290224, 59717.02984656, 59717.03679089], dtype='>f8')
```

```
tab = fits.open("WASP189.fits")[1].data
time = tab["TIME"]
flux = tab["FLUX"]
ferr = tab["FLUX_ERR"]

fig, ax = plt.subplots(figsize=(4, 3))
ax.errorbar(time, flux, yerr=ferr, fmt='.', ms=2.5, alpha=1,
             color="mediumvioletred", ecolor="gray", elinewidth=1,
capsize=1.0)

ax.set_xlabel("Tiempo (MJD)")
ax.set_ylabel("Flujo normalizado")
ax.set_title("Tránsito Planetario WASP-189 b")
ax.grid(alpha=0.25, lw=0.5)
ax.set_xlim(59699.3, 59700.05)
plt.tight_layout()
plt.show()
```



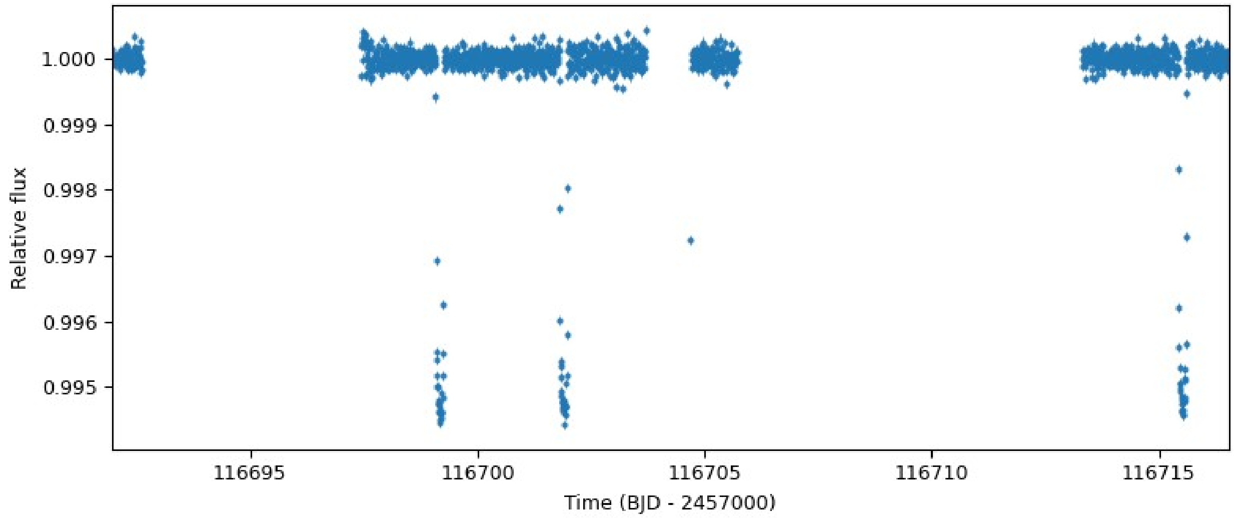
Juliet

```
t = lc_time
f = tab['FLUX']
ferr = tab['FLUX_ERR']

t = np.array(t, dtype=float)
f = np.array(f, dtype=float)
ferr = np.array(ferr, dtype=float)

mask = np.isfinite(f) & np.isfinite(ferr)

t = t[mask]
f = f[mask]
ferr= ferr[mask]

print(np.any(np.isnan(t)))
print(np.any(np.isnan(f)))
print(np.any(np.isnan(ferr)))
```

```
False
False
False
```

```
plt.figure(figsize=(8, 3.5))
plt.errorbar(t, f, yerr=ferr, fmt='.', markersize=4, elinewidth=0.6,
alpha=0.8)
plt.xlim(np.min(t), np.max(t))
# plt.ylim([0.999, 1.001])  # descomenta si quieres forzar un zoom en
el flujo relativo
plt.xlabel('Time (BJD - 2457000)')
plt.ylabel('Relative flux')
```

```
plt.tight_layout()
plt.show()
```




```
# # Create dictionaries:
# times, fluxes, fluxes_error = {},{},{}
# # Save data into those dictionaries:
# times['TESS'], fluxes['TESS'], fluxes_error['TESS'] = t,f,ferr
# # If you had data from other instruments you would do, e.g.,
# # times['K2'], fluxes['K2'], fluxes_error['K2'] = t_k2,f_k2,ferr_k2

rho = 0.158 * 1408
rho

222.464

priors = {}

# Name of the parameters to be fit:
params =
['P_p1','t0_p1','r1_p1','r2_p1','q1_lc1','q2_lc1','ecc_p1','omega_p1',
\
              'rho', 'mdilution_lc1', 'mflux_lc1', 'sigma_w_lc1', \
              'mu_HARPS', 'K_p1', 'sigma_w_HARPS'] # rv

# Distribution for each of the parameters:
dists =
['normal','normal','uniform','uniform','uniform','uniform','fixed','fi
xed',\
                 'loguniform', 'fixed', 'normal', 'loguniform', \
        'uniform', 'uniform', 'loguniform']

# Hyperparameters of the distributions (mean and standard-deviation
for normal
```

```
# distributions, lower and upper limits for uniform and loguniform
distributions, and
# fixed values for fixed "distributions", which assume the parameter
is fixed)
hyperps = [[2.7, 0.1], [8926,50], [0.,1], [0.,1.], [0., 1.], [0., 1.],
0.0, 90.,\
                   [100., 3000.], 1.0, [0.,0.1], [0.1, 1000.], \
          [-30, -20], [0.18, 0.3], [1e-3, 1]]
# Populate the priors dictionary:s
for param, dist, hyperp in zip(params, dists, hyperps):
    priors[param] = {}
    priors[param]['distribution'], priors[param]['hyperparameters'] =
dist, hyperp

# dataset = juliet.load(priors = priors,
#                       t_lc={   'lc1': t},
#                       y_lc={   'lc1': f},
#                       yerr_lc={'lc1': ferr},
#                       t_rv={'HARPS': rv_data['mjd']},
#                       y_rv={'HARPS':    rv_data['rv_mean']},
#                       yerr_rv={'HARPS': rv_data['rv_err']},
#                       out_folder = 'join_fit_1')

# results_f = dataset.fit(n_live_points = 300)
```

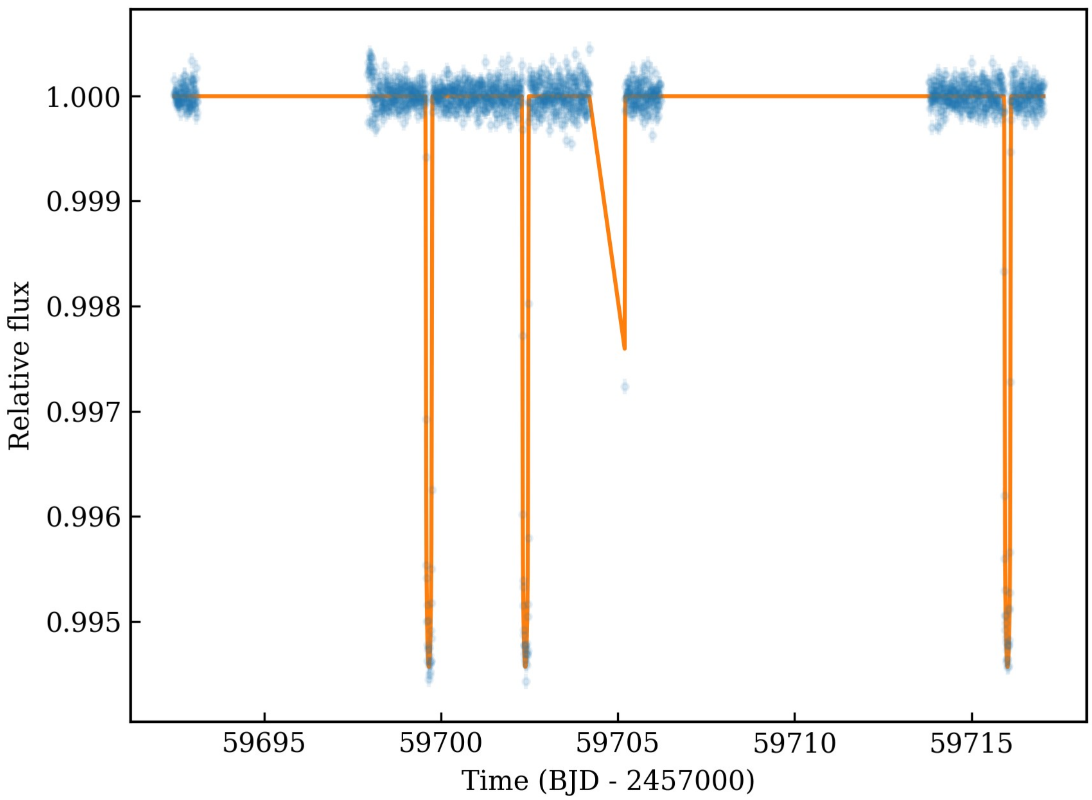
```
PyMultinest installation not detected. Forcing dynesty as the sampler.
```

```
43665it [42:05, 17.29it/s, +300 | bound: 229 | nc: 1 | ncall: 1373551
| eff(%):  3.202 | loglstar:   -inf < 10594.003 <    inf | logz:
10449.787 +/-  0.689 | dlogz:  0.001 >  0.309]
```

```
# Load already saved dataset with juliet:
dataset = juliet.load(input_folder = 'join_fit_1', out_folder =
'join_fit_1')

# Load results (the data.fit call will recognize the juliet output
files in
# the toi141_fit folder generated when we ran the code for the first
time):
results = dataset.fit()

import matplotlib.pyplot as plt

# Plot the data:
plt.errorbar(dataset.times_lc['lc1'], dataset.data_lc['lc1'], \
             yerr = dataset.errors_lc['lc1'], fmt = '.', alpha = 0.1)

# Plot the model:
plt.plot(dataset.times_lc['lc1'], results.lc.evaluate('lc1'))

# Plot portion of the lightcurve, axes, etc.:
```

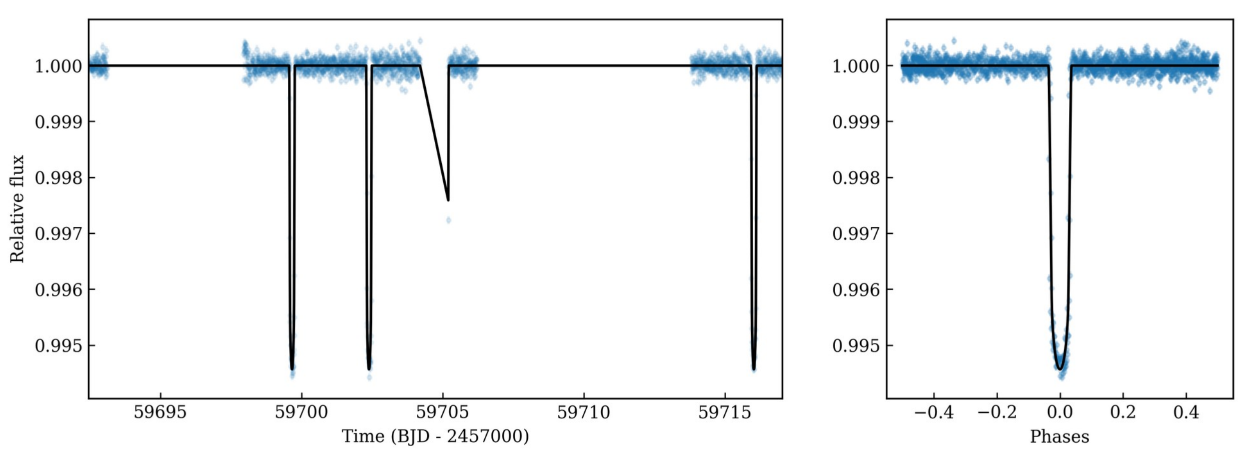
```python
# plt.xlim([1326,1332])
# plt.ylim([0.999,1.001])
plt.xlabel('Time (BJD - 2457000)')
plt.ylabel('Relative flux')
plt.show()
```

```
PyMultinest installation not detected. Forcing dynesty as the sampler.
Detected dynesty sampler output files --- extracting from
join_fit_1/_dynesty_NS_posteriors.pkl
```

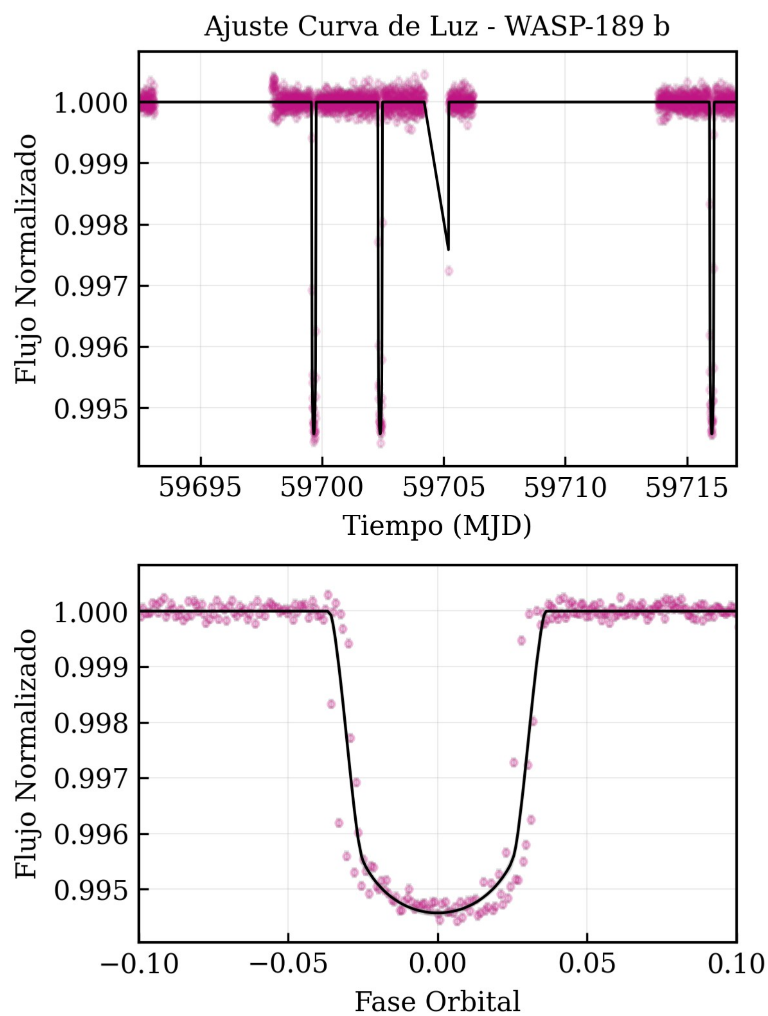



```python
# Extract median model and the ones that cover the 68% credibility
band around it:
transit_model, transit_up68, transit_low68  =
results.lc.evaluate('lc1', return_err=True)

# To plot the phased lighcurve we need the median period and time-of-
transit center:
P, t0 = np.median(results.posteriors['posterior_samples']['P_p1']),\
        np.median(results.posteriors['posterior_samples']['t0_p1'])

# Get phases:
phases = juliet.get_phases(dataset.times_lc['lc1'], P, t0)
```

```
import matplotlib.gridspec as gridspec

# Plot the data. First, time versus flux --- plot only the median
model here:
fig = plt.figure(figsize=(12,4))
gs = gridspec.GridSpec(1, 2, width_ratios=[2,1])
ax1 = plt.subplot(gs[0])

ax1.errorbar(dataset.times_lc['lc1'], dataset.data_lc['lc1'], \
             yerr = dataset.errors_lc['lc1'], fmt = '.' , alpha = 0.1)

# Plot the median model:
ax1.plot(dataset.times_lc['lc1'], transit_model,
color='black',zorder=10)

# Plot portion of the lightcurve, axes, etc.:
ax1.set_xlim([np.min(dataset.times_lc['lc1']),np.max(dataset.times_lc[
'lc1'])])
# ax1.set_ylim([0.96,1.04])
ax1.set_xlabel('Time (BJD - 2457000)')
ax1.set_ylabel('Relative flux')

# Now plot phased model; plot the error band of the best-fit model
here:
ax2 = plt.subplot(gs[1])
ax2.errorbar(phases, dataset.data_lc['lc1'], \
             yerr = dataset.errors_lc['lc1'], fmt = '.', alpha = 0.2)
idx = np.argsort(phases)
ax2.plot(phases[idx],transit_model[idx], color='black',zorder=10)
ax2.fill_between(phases[idx],transit_up68[idx],transit_low68[idx],\
                 color='white',alpha=0.5,zorder=5)
ax2.set_xlabel('Phases')
# ax2.set_xlim([-0.1,0.1])
# ax2.set_ylim([0.9985,1.0015])

Text(0.5, 0, 'Phases')
```



```python
transit_model, transit_up68, transit_low68  =
results.lc.evaluate('lc1', return_err=True)
P  = np.median(results.posteriors['posterior_samples']['P_p1'])
t0 = np.median(results.posteriors['posterior_samples']['t0_p1'])
phases = juliet.get_phases(dataset.times_lc['lc1'], P, t0)

fig = plt.figure(figsize=(4, 6))
gs = gridspec.GridSpec(2, 1, height_ratios=[1.1, 1], hspace=0.25)

ax1 = plt.subplot(gs[0])
ax1.errorbar(dataset.times_lc['lc1'], dataset.data_lc['lc1'],
             yerr=dataset.errors_lc['lc1'], fmt='.', alpha=0.15,
             color="mediumvioletred", ecolor="gray", elinewidth=0.6,
capsize=1.0)
ax1.plot(dataset.times_lc['lc1'], transit_model, color='black',
lw=1.0, zorder=10)

ax1.set_xlim(np.min(dataset.times_lc['lc1']),
np.max(dataset.times_lc['lc1']))
ax1.set_xlabel("Tiempo (MJD)")
ax1.set_ylabel("Flujo Normalizado")
ax1.set_title("Ajuste Curva de Luz - WASP-189 b")
ax1.grid(alpha=0.25, lw=0.5)

ax2 = plt.subplot(gs[1])
ax2.errorbar(phases, dataset.data_lc['lc1'],
             yerr=dataset.errors_lc['lc1'], fmt='.', alpha=0.25,
             color="mediumvioletred", ecolor="gray", elinewidth=0.6,
capsize=1.0)
idx = np.argsort(phases)
ax2.plot(phases[idx], transit_model[idx], color='black', lw=1.0,
zorder=10)
ax2.fill_between(phases[idx], transit_up68[idx], transit_low68[idx],
                 color='gray', alpha=0.3, zorder=5)

ax2.set_xlabel("Fase Orbital")
ax2.set_ylabel("Flujo Normalizado")
ax2.set_xlim(-0.1, 0.1)
# ax2.set_ylim(0.9985, 1.0015)
ax2.grid(alpha=0.25, lw=0.5)

plt.tight_layout()
plt.show()
```

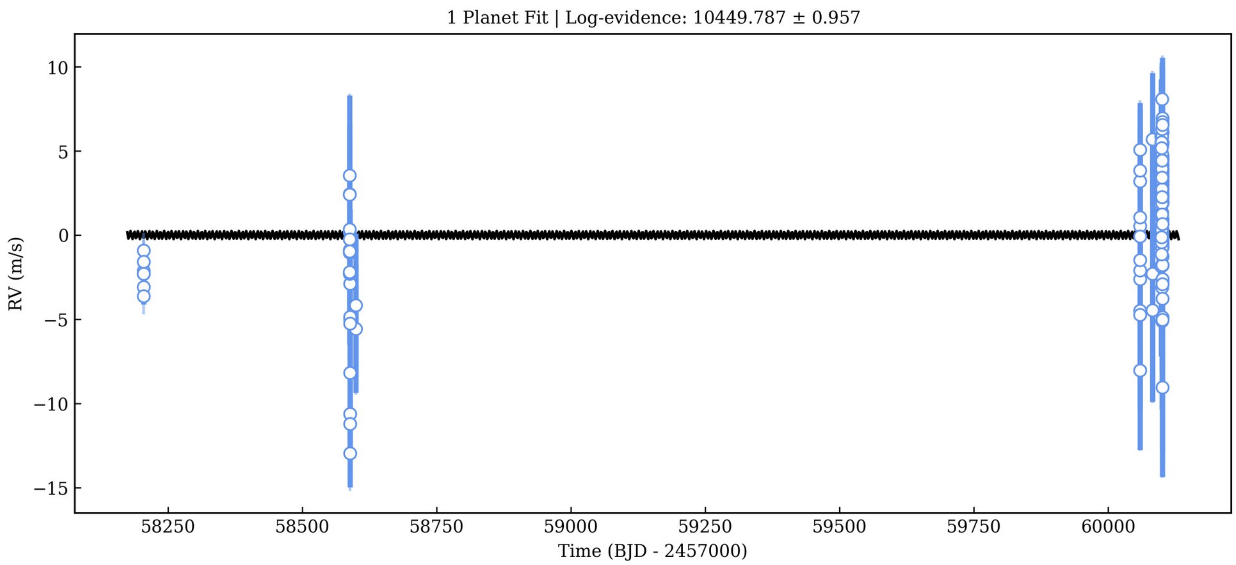



```
LC = fits.open('WASP189.fits')

# #2457000
# times, fluxes, fluxes_error = {},{},{}
# times['TESS'], fluxes['TESS'], fluxes_error['TESS'] =
LC[1].data['TIME'],LC[1].data['FLUX'],LC[1].data['FLUX_ERR']
# fluxes['TESS'] = fluxes['TESS'][times['TESS']<1400]
# fluxes_error['TESS'] = fluxes_error['TESS'][times['TESS']<1400]
# times['TESS'] = times['TESS'][times['TESS']<1400]

# plt.errorbar(times['TESS'], fluxes['TESS'],
yerr=fluxes_error['TESS'], fmt='.')
# plt.xlim([np.min(times['TESS']),np.max(times['TESS'])])

import numpy as np
import matplotlib.pyplot as plt

# Plot HARPS and FEROS datasets in the same panel. For this, first
select any
```

```
# of the two and substract the systematic velocity to get the
Keplerian signal.
# Let's do it with FEROS. First generate times on which to evaluate
the model:
min_time, max_time = np.min(dataset.times_rv['HARPS'])-30,\
                     np.max(dataset.times_rv['HARPS'])+30

model_times = np.linspace(min_time,max_time,1000)

# Now evaluate the model in those times, and substract the systemic-
velocity to
# get the Keplerian signal:
keplerian = results.rv.evaluate('HARPS', t = model_times) - \
            np.median(results.posteriors['posterior_samples']
['mu_HARPS'])

# Now plot the (systematic-velocity corrected) RVs:
fig = plt.figure(figsize=(12,5))
instruments = ['HARPS']
colors = ['cornflowerblue','orangered']
for i in range(len(instruments)):
    instrument = instruments[i]
    # Evaluate the median jitter for the instrument:
    jitter = np.median(results.posteriors['posterior_samples']
['sigma_w_'+instrument])
    # Evaluate the median systemic-velocity:
    mu = np.median(results.posteriors['posterior_samples']
['mu_'+instrument])
    # Plot original data with original errorbars:

plt.errorbar(dataset.times_rv[instrument],dataset.data_rv[instrument]-
mu,\
                 yerr = dataset.errors_rv[instrument],fmt='o',\
                 mec=colors[i], ecolor=colors[i], elinewidth=3, mfc =
'white', \
                 ms = 7, label=instrument, zorder=10)

    # Plot original errorbars + jitter (added in quadrature):

plt.errorbar(dataset.times_rv[instrument],dataset.data_rv[instrument]-
mu,\
                 yerr =
np.sqrt(dataset.errors_rv[instrument]**2+jitter**2),fmt='o',\
                 mec=colors[i], ecolor=colors[i], mfc = 'white',
label=instrument,\
                 alpha = 0.5, zorder=5)

# Plot Keplerian model:
plt.plot(model_times, keplerian,color='black',zorder=1)
plt.ylabel('RV (m/s)')
```

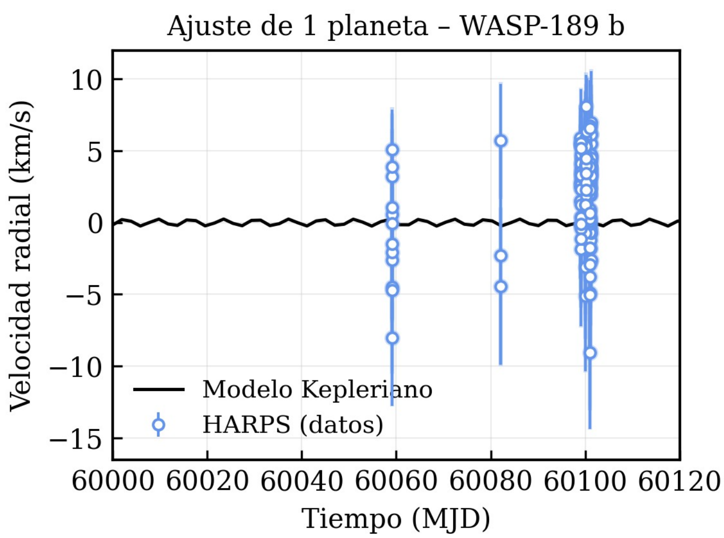
```
plt.xlabel('Time (BJD - 2457000)')
plt.title('1 Planet Fit | Log-evidence: {0:.3f} $\pm$
{1:.3f}'.format(results.posteriors['lnZ'],\
      results.posteriors['lnZerr']))
# plt.ylim([-20,20])
# plt.xlim([58500,58750])
```

```
Text(0.5, 1.0, '1 Planet Fit | Log-evidence: 10449.787 $\\pm$ 0.957')
```




```
min_time, max_time = np.min(dataset.times_rv['HARPS']) - 30, \
                     np.max(dataset.times_rv['HARPS']) + 30
model_times = np.linspace(min_time, max_time, 1000)

keplerian = results.rv.evaluate('HARPS', t=model_times) - \
            np.median(results.posteriors['posterior_samples']
['mu_HARPS'])

fig, ax = plt.subplots(figsize=(4, 3))

instruments = ['HARPS']
colors = ['cornflowerblue']

for i, instrument in enumerate(instruments):
    jitter = np.median(results.posteriors['posterior_samples']
[f'sigma_w_{instrument}'])
    mu = np.median(results.posteriors['posterior_samples']
[f'mu_{instrument}'])

    ax.errorbar(dataset.times_rv[instrument],
dataset.data_rv[instrument] - mu,
                yerr=dataset.errors_rv[instrument],
                fmt='o', mec=colors[i], ecolor=colors[i],
```

```
elinewidth=1.0,
                mfc='white', ms=4, label=f"{instrument} (datos)",
zorder=10)
    ax.errorbar(dataset.times_rv[instrument],
dataset.data_rv[instrument] - mu,
                yerr=np.sqrt(dataset.errors_rv[instrument]**2 +
jitter**2),
                fmt='o', mec=colors[i], ecolor=colors[i], mfc='white',
                alpha=0.3, zorder=5)

ax.plot(model_times, keplerian, color='black', lw=1.2, zorder=2,
label="Modelo Kepleriano")

ax.set_xlabel("Tiempo (MJD)")
ax.set_ylabel("Velocidad radial (km/s)")
ax.set_title("Ajuste Velocidad Radial WASP-189 b")
ax.grid(alpha=0.25, lw=0.5)
ax.legend(frameon=False)
ax.set_xlim([60000, 60120])
# ax.set_ylim([-20, 20])
plt.tight_layout()
plt.show()
```

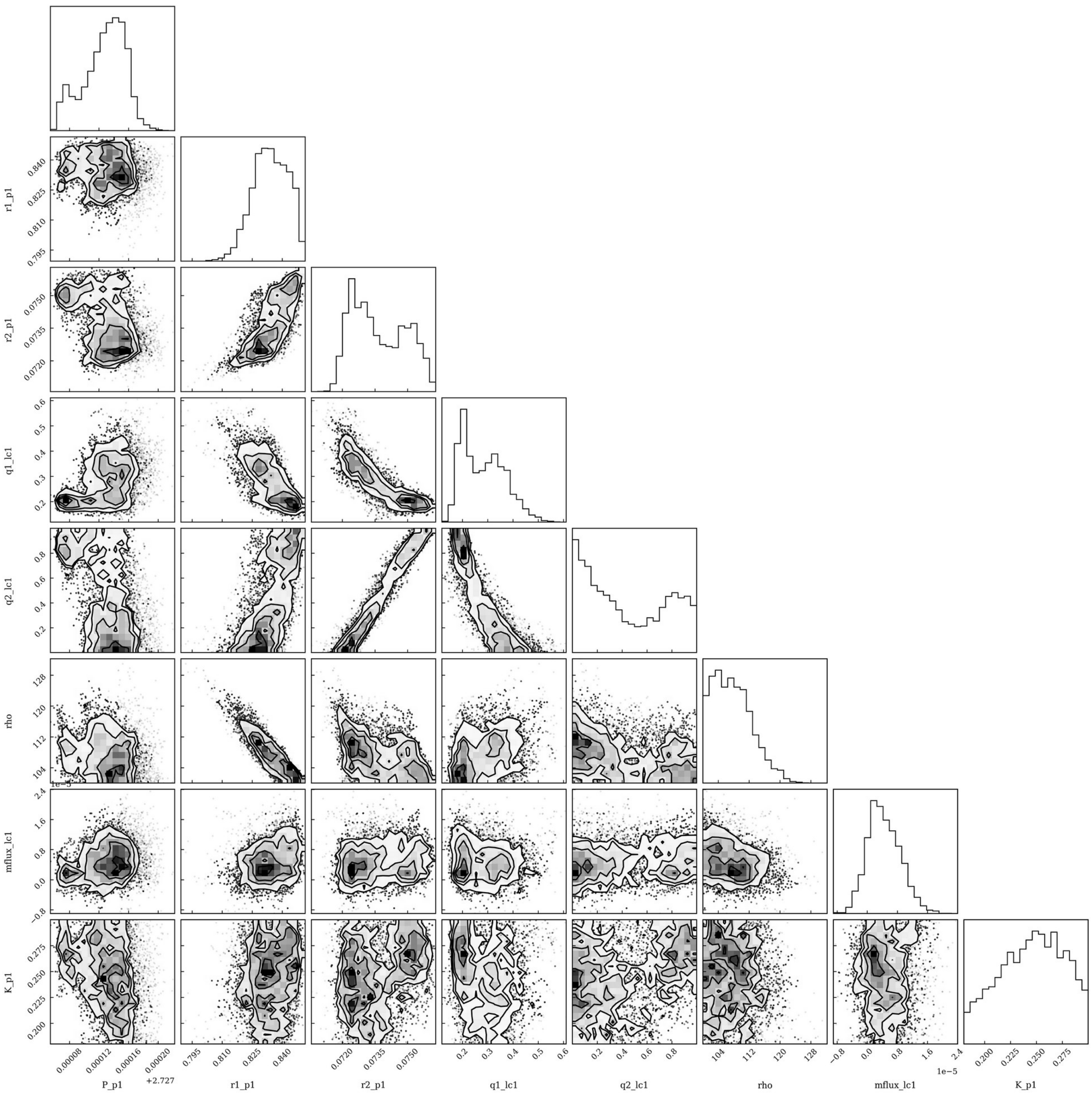



```
post = pd.read_csv('join_fit_1/posteriors.dat', sep = '\t \t')
post
```

	# Parameter Name	Median	Upper 68 CI	Lower 68 CI
0	P_p1	2.727132	0.000023	0.000035
1	a_p1	3.480244	0.058910	0.051508
2	t0_p1	8961.376259	0.651632	0.432036
3	r1_p1	0.834208	0.009199	0.008921
4	r2_p1	0.073429	0.001746	0.001058

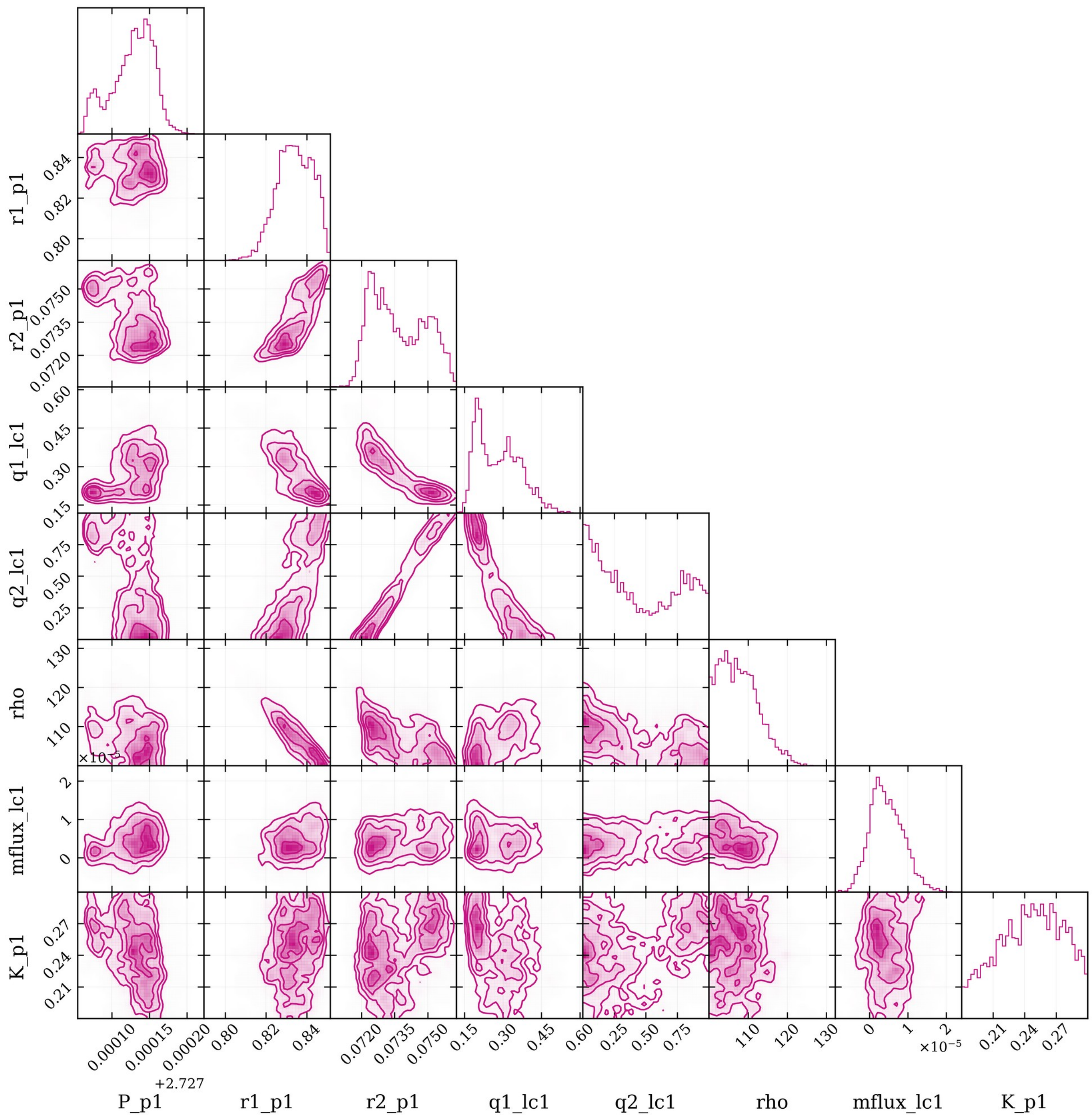
5	p_p1	0.073429	0.001746	0.001058
6	b_p1	0.751312	0.013799	0.013382
7	inc_p1	77.530276	0.423005	0.409802
8	q1_lc1	0.275896	0.097036	0.080683
9	q2_lc1	0.355939	0.484947	0.277347
10	rho	107.217457	5.540559	4.687572
11	mflux_lc1	0.000004	0.000005	0.000004
12	sigma_w_lc1	197.029658	4.966916	5.330894
13	mu_HARPS	-20.178757	0.120207	0.125494
14	K_p1	0.247576	0.030911	0.035308
15	sigma_w_HARPS	0.966082	0.022871	0.034080

```
import corner

posterior_names = [r"$K_1$ (m/s)", r"$P_1$ (days)"]
first_time = True
posterior_names2 = []
for i in range(len(params)):
    if dists[i] != 'fixed' and 't0' not in params[i] and \
    params[i][0:2] != 'mu' and params[i][0:5] != 'sigma':
        if first_time:
            posterior_data = results.posteriors['posterior_samples']
[params[i]]
            first_time = False
            posterior_names2.append(params[i])
        else:
            posterior_data  = np.vstack((posterior_data,
results.posteriors['posterior_samples'][params[i]]))
            posterior_names2.append(params[i])
posterior_data = posterior_data.T
# figure = corner.corner(posterior_data, labels = posterior_names)
figure = corner.corner(posterior_data, labels = posterior_names2)
```




```
import corner

fig = plt.figure(figsize=(15, 15))
figure = corner.corner(
    posterior_data,
    labels=posterior_names2,
    color="mediumvioletred",
    bins=40,
    smooth=1.0,
    hist_kwargs={"density": True, "color": "mediumvioletred"},
    label_kwargs={"fontsize": 18},
    plot_contours=True,
```

```
    contourf_kwargs={"alpha": 0.6},
    use_math_text=True,
    plot_datapoints=False,
    max_n_ticks=4,
    fig=fig,
)

for ax in figure.get_axes():
    ax.tick_params(direction='in', top=True, right=True,
                   width=1.1, length=6, labelsize=14)
    ax.grid(alpha=0.2, lw=0.5)

for ax in figure.get_axes():
    if ax.get_xlabel():
        ax.xaxis.set_label_coords(0.5, -0.6)
    if ax.get_ylabel():
        ax.yaxis.set_label_coords(-0.4, 0.5)

plt.subplots_adjust(
    left=0.08, right=0.97, bottom=0.08, top=0.97,
    wspace=0.0, hspace=0.0
)
plt.show()
```



r1_p1
r2_p1
q1_lc1
q2_lc1
rho
mflux_lc1
K_p1
P_p1
+2.727
×10⁻⁵