Sweet-Cat – Spectroscopic characterization of stars with exoplanets

Report - PEEC 2019/2020

Bárbara M. T. B. Soares[1]

Supervisor: Sérgio G. Sousa[2]
Co-supervisor: Solène Ulmer-Moll[2]

[1] Departamento de Física e Astronomia, Faculdade de Ciências, Universidade do Porto, Rua do Campo Alegre, 4169-007 Porto, Portugal
[2] Centro de Astrofísica, Universidade do Porto, Rua das Estrelas, 4150-762 Porto, Portugal

July, 2020

ABSTRACT

To better understand planetary properties, such as their internal structure and their atmosphere, it is necessary to know the characteristics of the planet host star. For this purpose, it was created the SWEET-Cat, a catalogue compiling stellar parameters for stars with planets, built from literature data. The aim of this work was to connect the stellar database SWEET-Cat to the planetary databases Extrasolar Planets Encyclopaedia (EU) and NASA Exoplanet Archive (NASA), enabling a way to work with the information from the three sources at once. This was done using the programming language pyhton3. Downloading the NASA database, loading it in a pandas dataframe and making use of the module `pyexoplaneteu` to do the same for the EU database, it was possible to work with all the data in the same environment. In order to test the code, we tried to reproduce a Period-Metallicity diagram in a paper already published, using the same conditions. The resulting plot was very similar to the original one, with minor differences, possibly due to data changes in the EU database, as it is continuously being updated.

Key words. exoplanets – planetary systems

1. Introduction

Since Mayor and Queloz did the first detection of a giant planet orbiting a solar-like star in 1995, extra-solar planets (for short, exoplanets) have continued to be discovered. These exoplanets range from smaller, Earth-size ones to Jupiter-like planets, with many Neptune-like in between. The development of new technology, such as the ESO high-resolution HARPS spectrograph, has made possible these continuous discoveries, allowing obtaining high-quality, high S/N and high-resolution data (Sousa et al. 2008).

The increasing number of known planetary systems enables the possibility to explore the planetary properties, such as their internal structure and their atmospheres. This is done specially through the characterisation of the planet host stars, as stellar parameters are needed to fully characterise the planetary properties. For example, if we want to measure precise values for the radius of a transiting planet, we need precise stellar radii values. In turn, the value of stellar radii will depend on the quality of the derived stellar parameters, such as the effective temperature. On the other hand, the chemical composition of the molecular cloud that originated the proto-star, reflected on the composition of the stellar atmosphere, is probably related to the interior and atmospheric chemical composition of a planet.

With this in mind, it was created the SWEET-Cat (Stars With ExoplanETs Catalog), a catalog of stellar parameters for stars with planets listed in the Extrasolar Planets Encyclopaedia and in NASA Exoplanet Archive (henceforward referred as EU database and NASA database, respectively). It compiles sets of atmospheric parameters previously published in the literature, such as Teff, logg, and [Fe/H] and derived using the same uniform methodology, whenever possible. This catalog, described in Santos et al. (2013) is built from literature data, either published or to be published soon, and therefore is continuously being updated as new planets are announced and new stellar parameters derived.

This report is divided in the following sections: in Section 2 there will be a description of the code developed; in Section 3 there will be an example of an application of the code, with a comparison with a figure from a published paper; Section 4 concludes with the location of the repository for the code (the GitHub repository) and what can be done as future work to improve the code.

2. Code description

The code was developed using python3 and divided in different files, stored in a GitHub repository. The repository is composed of nine python files, three encased within two folders and the rest outside them.

For the folders, we have:

1. **Plots:** `plots.py` and `plot2.py` are two different ways to plot the planet's period as a function of the stellar metallicity, for planets with mass M $<$ 30 $M_{\oplus}$ and whose measurements were done with a precision above 20%.

2. **Older versions codes:** `funcoes.py` is an older version of `functions.py`, added with some unused functions.

For the remaining files, we have:

- `dictio.py`: a python file which corresponds EU database parameters (column names) to NASA database parameters (column names), whenever possible.
- `functions.py`: this is the main file, a python file with functions to extract planetary information from SWEET-Cat, NASA and EU databases.
- `nasa.py`: python file responsible for the downloading and storing of NASA database in a dictionary.
- `new_nasa.py`: python file modifying the NASA data base, adding 3 columns for the parameters *Msini* and its upper and lower uncertainties.
- `README.md`: README file describing the repository, its functions and explaining how to use them with a tutorial.
- `WEBSITE_online_EU-NASA_full_database_clean_06-04-2020.rdb`: file with data from the SWEET-Cat database.

The description of the codes will be done not in an alphabetical order, but as they were written.

2.1. nasa.py

This file is composed of eighteen functions, being that the main one is the `get_data()` function, which allows to load the NASA database information stored on a dictionary and work with it. This was the most difficult file to write, as it was necessary to not only download the online data with the `urllib` module, which is done in the `download_data()` function, but also store it in the computer with a directory created (done in the `_create_data_dir()`, `_check_data_dir()` and `get_data_dir()` using the `os` module) and checking if the data was recent, as the NASA database is continuously being updated (`check_data_age()` function defined that any data older than 5 days needs to be updated).

Furthermore, in order to preserve the table present online, it was necessary to save the data in a dictionary, which is done with the `DataDict` class.

However, to read the dictionary created correctly, it was necessary to take into account that although all the entries were stings, there were some that corresponded to values and hence had to be converted to float. Therefore, it was created the `read_data()` function.

As such, the `get_data()` function checks if there is already a directory created for the file (if not, creates it), checks if the data is recent enough (if not, download the data again) and stores it in a dictionary ready to be used.

CODE:
```
import pandas as pd
from nasa import get_data
# dictionary
data_dic = get_data()
# dataframe
data_df = pd.DataFrame.from_dict(get_data())
```

2.2. new_nasa.py

The EU database has columns with all *Mass* information related and columns with the *Msini* information related. Yet, the NASA database has the *Mass* and *Msini* information mixed in the same columns. In order to work with the both of them at the same time in a more uniform way, it was created the `new_na()` function in the `new_nasa.py` file, which imports the NASA database from the `nasa.py` file. This function uses the information from the *Mass provenance* column to assess if the mass information for that planet was a *Mass* ou *Msini* value. If the label was `'Msini'`, then the value and its upper and lower uncertainties were moved to new columns, specific for all *Msini* information related. This function returns the new NASA database in a pandas dataframe.

CODE:
```
from new_nasa import new_na
# dataframe with Msini columns
new_nasa_df = new_na()
```

2.3. dictio.py

This python file only contains a dictionary responsible for corresponding the EU database column names to the NASA database column names, so that the user doesn't have to memorize the different names for that same planetary parameters in both databases. It was chosen to correspond the EU parameters to the NASA ones, as there are more in EU than NASA and because the EU ones are more readable then the ones in NASA. For example, the mass parameter in the EU database is listed as `'mass'` but in the NASA database is listed as `'pl_bmassj'`.

However, not all EU parameters have a NASA correspondence (and vice versa) and in these cases, the EU parameter match remains empty (for example, `'geometric_albedo':''`).

2.4. functions.py

This python file contains five functions:

- `coor_sc2deg()`: Converts coordinates of the SWEET-Cat star to degrees.
- `match()`: Given a SWEET-Cat star, verifies if it already exists in the NASA and EU database. Returns all or specified information.
- `verify_database()`: Given a SWEET-Cat star, uses the database column of SWEET-Cat to verify which database contains information of the star and its planets.
- `get_sc()`: Given the name of a SWEET-Cat star, returns all information or the selected parameters as they are in SWEET-Cat.
- `coor2deg()`: Converts all SWEET-Cat coordinates (right ascension and declination) to degrees.

The new NASA database modified is imported from the `new_nasa.py` file, using the `new_na()` function. The EU database is imported through the `pyexoplaneteu` module, using the `get_data()` function, in a way similar to the NASA case. Also, it is needed the dictionary `dic` from the `dictio.py` file to correspond the parameter from the two databases.

In order to work with the data, the mainly used modules are the `numpy` and `pandas` modules. The `pandas` module is particularly useful as it enables the storing of data in a pandas dataframe, which works similar to a table, with rows for the planets and columns for the parameters, whether the data is in dictionary, csv file or other formats.

`coor_sc2deg()`

Given the name of a SWEET-Cat star, this function converts its coordinates in degrees. The right ascension is in hours:minutes:seconds (hh:mm:ss) and the declination is in degrees:minutes:seconds (dd:mm:ss). It is also necessary to note that the coordinates are string values and as such, need to be converted to floats. In the case of the right ascension, the conversion from hh:mm:ss to degrees is done using

$$degrees = \left(hh + \frac{mm}{60} + \frac{ss}{3600}\right) \times 15 \tag{1}$$

In case of the declination, if the degrees dd are positive, we have

$$degrees = dd + \frac{mm}{60} + \frac{ss}{3600} \tag{2}$$

and if it is negative, we have

$$degrees = dd - \frac{mm}{60} - \frac{ss}{3600} \tag{3}$$

CODE:

```
from functions import coor_sc2deg
ra, dec = coor_sc2deg('42 Dra')
```

`match()`

Given the SWEET-Cat name of a star or its coordinates, this function verifies if it already exists in the NASA and EU database. If so, returns all the stellar and planetary information or the parameters specified. The user can choose to give the star's name or its coordinates, both as they are in SWEET-Cat.

If the name is given, firstly the function verifies if that name exist in SWEET-Cat, as the names are case sensitive, and then, using the `coor_sc2deg()` function, it gets the coordinates of the star (in degrees), which are then converted to arcseconds.

CODE:

```
from functions import match
# All information
na_name, eu_name = match('42 Dra')

# Specific parameters
na_name2, eu_name2 = match('42 Dra',
list_of_parameters=
['mass','radius','orbital_period'])
```

If the coordinates are given, they must be the string values as they are in SWEET-Cat. In this case, the function verifies if these coordinates are present in SWEET-Cat. If so, then it proceeds to convert these coordinates to arcseconds. If the user only gives the right ascension or only the declination, the function will give a warning, asking for both coordinates and not only one. In case the users gives both the name and coordinate(s), the function will give a warning, asking for only the name or only the coordinates.

CODE:

```
from functions import match
# All information
na_coor, eu_coor = match(r_asc='18 25 59.13',
                         declin='+65 33 48.52')

#Specific parameters
na_coor2, eu_coor2 = match(r_asc='18 25 59.13',
declin='+65 33 48.52', list_of_parameters=
['mass','radius','orbital_period'])
```

At this point, independently of the input given, we have the star coordinates in arcseconds. Next, the function will check the NASA and EU databases for a match with the star given. This will be done firstly comparing the star's coordinates with the coordinates in the databases (the coordinates in EU are string values, so it is necessary to convert them to float values), within a certain limit, as different databases may have slightly different values of coordinates. Initially, the coordinates were only in degrees and the limit was defined as 0.09 degrees. However, after start working with arcseconds, this was a value too much high (0.09 degrees = 324 arcseconds) and therefore, the limit was redefined to 5 arcseconds, by default. As this is one of the inputs, the user may change the value of this limit, not forgetting it is in arcseconds. If the matching by coordinates fails, the function will try a matching by name.

If no list of parameters is specified, the function returns all the information it founds. In case the list of parameters is specified, the parameters should be given as they are in EU. Therefore, we need the dictionary in `dictio.py` to make the correspondence of the same parameters for the two databases. If one or more parameters given are exclusive to the EU database, in the NASA dataframe will appear only those present in the NASA database, and the function will give a message saying which parameters do not exist in NASA database.

This procedure is repeated for both NASA and EU databases, with the exception that when searching for a match in NASA, the function needs to resort to the dictionary in case there are parameters specified.

`verify_database()`

This function's main purpose is to check which databases (EU and NASA) have information related to planets in this star's systems. Making use of the `'database'` column, the user provides the name of a SWEET-Cat star and the function says if there is information in EU database, in NASA database or in both of them. This way, it is possible to verify if the match function is returning all the information required.

CODE:

```
from functions import verify_database
db = verify_database('42 Dra')
```

`get_sc()`

Similar to the `verify_database()` function, the user provides the name of a SWEET-Cat star to see all information the catalogue has about this star. Additionally, it is possible to provide a list of specific parameters in case the user doesn't want to see all information, but only some properties.

CODE:

```
from functions import get_sc
# All information
get_sc('42 Dra')

# Specific parameters
get_sc('42 Dra', list_of_parameters=['Teff','logg','feh'])
```

`coor2deg()`

This function simply converts all the SWEET-Cat stars' coordinates to degrees, returning two arrays: one for the right ascension in degrees and the other to declination in degrees.

CODE:

```
from functions import coor2deg
RA_sc, DEC_sc = coor2deg()
```

3. Application

With the purpose of showing how to apply the code, a reproduction of the Period–Metallicity diagram for Low Mass Planets (LMP) will be done, which is Figure 2 in Sousa et al. (2019), with a step by step process explaining the code. In the end, the obtained plot will be compared with the original plot.

The authors chose to work only with LMP for this diagram, in other words, with planets with minimum masses $M \leq 30 M_{\oplus}$, derived at least with 20% precision. All the planets and its properties were taken from the EU database.

First of all, using the `match()` function, we run through all the stars in SWEET-Cat, finding all their information present in the NASA database and the EU database. All the NASA matches indexes will be saved in a list, together with their SWEET-Cat index (`sc2nasa`), and the same for the EU matches, in another list (`sc2eu`), together with their SWEET-Cat index. Naturally, if the star has more than one planet, the SWEET-Cat index will appear as many times as there are planets in that system.

Next, we apply the condition for the mass (*Mass* and *Msini*) and precision, for the NASA database. Planets which satisfy the requirements for *Mass* will be saved through their NASA and respective SWEET-Cat indexes in a list (`nasa_mass`) and those that meet the requirements for the *Msini* parameter will be saved in another list (`nasa_msini`), together with their SWEET-Cat index. We then separate the information: the SWEET-Cat indexes are listed on `SC_NA` and the planetary indexes on NASA are listed on `NA_ind` (be it *Mass* ou *Msini*).

```
import numpy as np
import pandas as pd
from functions import *
import matplotlib.pyplot as plt

indi = list(sc.index)  # Index for running through SWEET Cat

sc2nasa=[]  # NASA
sc2eu=[]  # EU

for i in indi:
    # match all the SC stars with stars on NASA and EU
    NA, EU = match(sc_name=sc['name'][i])
    if len(NA)!=0:
    # If a star has more than on planet, the star index will appear repeated
        if len(NA)>1:
            for k in NA.index:
                sc2nasa.append([i,k])
        else:      # Star only has one planet
            sc2nasa.append([i,NA.index[0]])

    if len(EU)!=0:
     # If a star has more than on planet, the star ind will appear repeated
        if len(EU)>1:
            for j in EU.index:
                sc2eu.append([i,j])
        else:      # Star only has one planet
            sc2eu.append([i,EU.index[0]])

# From list to array
sc2nasa = np.array(sc2nasa)
sc2eu = np.array(sc2eu)

nasa_mass=[]
nasa_msini=[]
for i in range(len(sc2nasa)): # Apply condition for mass and precision in NASA
    if (na['pl_bmassj'][sc2nasa[i,1]]<0.094399) &
        ((na['pl_bmassjerr1'][sc2nasa[i,1]]<0.2*na['pl_bmassj'][sc2nasa[i,1]]) &
        (na['pl_bmassjerr2'][sc2nasa[i,1]]<0.2*na['pl_bmassj'][sc2nasa[i,1]])):

        # list[SC_index, NASA_index]
        nasa_mass.append([sc2nasa[i,0],sc2nasa[i,1]])

    if (na['pl_bmsinij'][sc2nasa[i,1]]<0.094399) &
        ((na['pl_bmsinijerr1'][sc2nasa[i,1]]<0.2*na['pl_bmsinij'][sc2nasa[i,1]]) &
        (na['pl_bmsinijerr2'][sc2nasa[i,1]]<0.2*na['pl_bmsinij'][sc2nasa[i,1]])):

        # list[SC_index, NASA_index]
        nasa_msini.append([sc2nasa[i,0],sc2nasa[i,1]])

nasa_mass = np.array(nasa_mass)
nasa_msini = np.array(nasa_msini)
SC_NA = np.concatenate((nasa_mass[:,0],nasa_msini[:,0]))
NA_ind = np.concatenate((nasa_mass[:,1],nasa_msini[:,1]))
```

When applying for the EU database, the process is very similar: we apply the condition for the mass and precision on the data (for the *Mass* and *Msini* parameter) and separate the information on two lists, `eu_mass` and `eu_msini`. However, the main difference lies in that there are some planets that have both the *Mass* and *Msini* value. In that case, we choose the mass value, keeping in mind that a few of them might still be minimum masses. Therefore, we check which indexes are common in both `eu_mass` and `eu_msini`, removing the repeated indexes from the `eu_msini` set. To not lose information about the order of the indexes, we reorder them following the original set, guarantying that the correspondence to the SWEET-Cat indexes is not lost. After removing the repeated indexes, we must removed their corresponding index in the SWEET-Cat list, otherwise we would get more stars than actual planets.

Next, we will do the same as in the NASA case, separating the EU indexes (be it related to *Mass* or *Msini*) on the `EU_ind` array and the SWEET-Cat indexes on `SC_EU`. All the SWEET-Cat indexes are stored in the `sc_ind` array.

As we are considering LMP planets with minimum mass $M < 30M_{\oplus}$, we will be using only the *Msini* parameter. The results are shown in Figure 1.

Although both plots have some different points, they are both samples with 127 planets. The different points can be explained as the databases are continuously being updated and recent measurements may have demoted some observations that were thought as being planets but might not be. For example, the planetary system with four planets that appears in the upper panel, with a metallicity around $[Fe/H] \sim -0.5$, is absent in the lower panels and may be one of those cases. However, there are some systems that remain common between the two panels, as is the case of the three planetary system with metallicity $[Fe/H] \sim -0.6$.

(The lower panel is obtained using the `plots2.py` code, present in the folder `Plots`, in the repository).

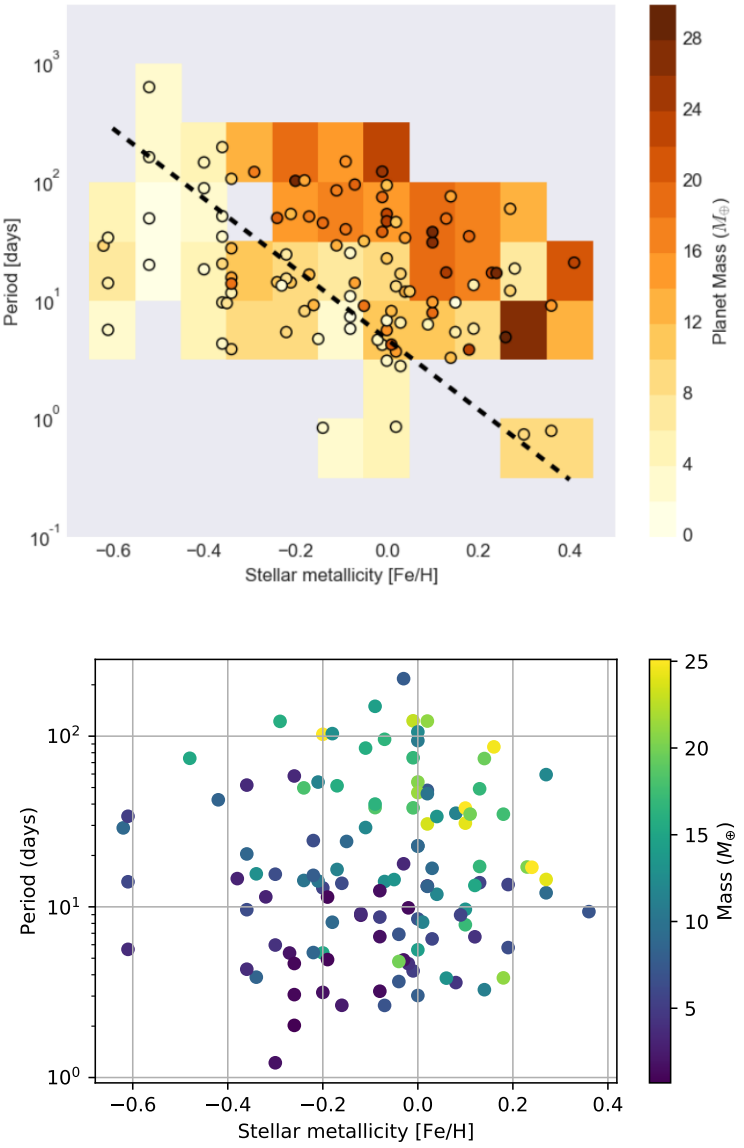

Fig. 1. Period-Metallicity diagram for LMP, with homogeneous parameters in SWEET-Cat. The color scheme represents the mass in $M_{\oplus}$. The upper panel shows the original plot, taken from Figure 2 in Sousa et al. (2019) and the lower panel represents the plot obtained using the `match` function.

```
eu_mass = []  # Mass for EU
eu_msini = []   # Msini for EU

for j in range(len(sc2eu)):
    # Apply condition for mass and precision in EU
    if (eu['mass'][sc2eu[j,1]]<0.094399)&
        ((eu['mass_error_max'][sc2eu[j,1]]<0.2*eu['mass'][sc2eu[j,1]])&
        (eu['mass_error_min'][sc2eu[j,1]]<0.2*eu['mass'][sc2eu[j,1]])):

        eu_mass.append([sc2eu[j,0],sc2eu[j,1]])

    if (eu['mass_sini'][sc2eu[j,1]]<0.094399)&
        ((eu['mass_sini_error_max'][sc2eu[j,1]]<0.2*eu['mass_sini'][sc2eu[j,1]])&
        (eu['mass_sini_error_min'][sc2eu[j,1]]<0.2*eu['mass_sini'][sc2eu[j,1]])):

        eu_msini.append([sc2eu[j,0],sc2eu[j,1]])
eu_mass = np.array(eu_mass)
eu_msini = np.array(eu_msini)

# Remove repeated indexes
idx_eu = list(set(eu_mass[:,1])^set(eu_msini[:,1]))

# Indexes of mass for EU
# Choose mass and not msini but few of them are still minimum masses in EU
# Remove msini values
msini_eu = np.intersect1d(eu_msini[:,1],idx_eu)

# Sort msini_eu as eu_msini[:,1] was, to not lose information
m2 = np.array(sorted(list(msini_eu), key=list(eu_msini[:,1]).index))

# Remove the stars for which some mass indexes were removed
df = pd.DataFrame(eu_msini)
df2 = df.loc[df[1].isin(m2)]
eu_msini2 = np.array(df2)

EU_ind = np.concatenate((eu_mass[:,1],eu_msini2[:,1]))  # Indexes for EU
SC_EU = np.concatenate((eu_mass[:,0],eu_msini2[:,0]))

# First indexes for NASA, second for EU
sc_ind = np.concatenate((SC_NA,SC_EU))  # Indexes for SWEET Cat

    'Metallicity'
feh = sc['feh'][eu_msini2[:,0]]
    'Period'
per = pd.concat([eu.loc[eu_msini2[:,1],'orbital_period']])
    'Mass'
m = pd.concat([eu['mass_sini'][eu_msini2[:,1]]*317.8])

'''PLOT'''
x2 = feh
y2 = per
t2 = m
fig, (ax1) = plt.subplots(1)
map1 = ax1.scatter(x2, y2, c=t2, cmap='viridis')
fig.colorbar(map1, ax=ax1, label = r'Mass (M_{\oplus})')
plt.xlabel('Stellar metallicity [Fe/H]')
plt.ylabel('Period (days)')
plt.yscale('log')
plt.savefig('period metal mass.pdf',format='pdf')
plt.show()
```

4. Conclusions

This codes allows us to analyse and extract planetary parameters not only from the EU database but also from the NASA database. The online repository of the codes can be found on GitHub, in the link `https://github.com/B-Soares/NASA_EU_databases`.

Despite all this, there is still work to be done. For example, in the dictionary, we covered the case where EU parameters did not have any corresponding parameters in NASA. We did not, however, take into account for the opposite case. What would happen if the parameter requested was exclusive only to the NASA database? Most likely the code would give an error, as this case was not contemplated.

When working with information from both databases simultaneously, it must be taken into account that there might be planets common to both databases. Hence, when analysing the information, for example a Period-Metallicity diagram but for planets from both NASA and EU databases, it would be necessary to add a function responsible for, in these cases, choosing which database to use. One criterion could be, mass precision, for example: the function would opt for the one which had a mass value measured with better precision. However, depending on what is the goal, there are many other criterions which probably could better apply.

References

Mayor, M. and Queloz, D. (1995). A Jupiter-mass companion to a solar-type star. *Nature*, 378(6555):355–359.

Santos, N. C., Sousa, S. G., Mortier, A., Neves, V., Adibekyan, V., Tsantaki, M., Delgado Mena, E., Bonfils, X., Israelian, G., Mayor, M., and Udry, S. (2013). SWEET-Cat: A catalogue of parameters for Stars With ExoplanETs. I. New atmospheric parameters and masses for 48 stars with planets. *A&A*, 556:A150.

Sousa, S. G., Adibekyan, V., Santos, N. C., Mortier, A., Barros, S. C. C., Delgado-Mena, E., Demangeon, O., Israelian, G., Faria, J. P., Figueira, P., Rojas-Ayala, B., Tsantaki, M., Andreasen, D. T., Brandão, I., Ferreira, A. C. S., Montalto, M., and Santerne, A. r. (2019). The metallicity-period-mass diagram of low-mass exoplanets. *MNRAS*, 485(3):3981–3990.

Sousa, S. G., Santos, N. C., Mayor, M., Udry, S., Casagrande, L., Israelian, G., Pepe, F., Queloz, D., and Monteiro, M. J. P. F. G. (2008). Spectroscopic parameters for 451 stars in the HARPS GTO planet search program. Stellar [Fe/H] and the frequency of exo-Neptunes. *A&A*, 487(1):373–381.