PyFANT Documentation

Release 17.12.21

Julio Trevisan

Dec 21, 2017

CONTENTS

Welcome!

PyFANT is a Python interface to the PFANT stellar spectral synthesis code, with a varied set of extra tools.

CHAPTER ONE

INTRODUCTION

PyFANT is a Python interface to PFANT, a stellar spectral synthesis code for Astronomy originally written in Fortran. PyFANT provides different ways to run spectral synthesis: command-line or graphical interface; single or batch mode. It also includes many additional tools to open, edit, visualize, convert and manipulate data files and spectra, and an API (application programming interfce) to create your own spectral synthesis applications in Python.

This documentation addresses coding with the PyFANT API, and provides a reference to the scripts included in the `pyfant` package.

For a tutorial on spectral synthesis using PFANT and PyFANT, please visit the PFANT project website: https://trevisanj.github.io/PFANT.

1.1 Acknowledgement

The project started in 2015 at IAG-USP (Institute of Astronomy, Geophysics and Atmospheric Sciences at University of São Paulo, Brazil).

Funded by FAPESP - Research Support Foundation of the State of São Paulo, Brazil (2015-2017).

1.2 Contact

For bugs reports, questions, suggestions, etc., please open an issue at the project site on GitHub: http://github.com/trevisanj/pyfant.

CHAPTER

TWO

INSTALLATION

If you have both **Python 3** and **PFANT** installed, then simply type:

```
pip install pyfant
```

2.1 Pre-requisites

2.1.1 PFANT

The PFANT spectral synthesis software installation instructions can be found at http://trevisanj.github.io/PFANT/install.html.

2.1.2 Python 3

If you need to set up your Python 3 environment, one option is to visit project F311 installation instructions at http://trevisanj.github.io/f311/install.html. That page also provides a troubleshooting section that applies.

2.2 Installing PyFANT in developer mode

This is an alternative to the "pip" at the beginning of this section. Use this option if you would like to download and modify the Python source code.

First clone the "pyfant" GitHub repository:

```
git clone ssh://git@github.com/trevisanj/pyfant.git
```

or

```
git clone http://github.com/trevisanj/pyfant
```

Then, install PyFANT in **developer** mode:

```
cd pyfant
python setup.py develop
```

2.3 Upgrade `pyfant`

Package `pyfant` can be upgraded to a new version by typing:

```
pip install pyfant --upgrade
```

CHAPTER

THREE

CODING USING THE API

This section contains a series of examples on how to use the PFANT Fortran executables from a Python script. These "bindings" to the Fortran binaries, together with the ability to load, manipulate and save PFANT data files allow for complex batch operations.

3.1 Spectral synthesis

The following code generates Figure 3.1.

```python
"""Runs synthesis over short wavelength range, then plots normalized and convolved spectrum"""

import f311.pyfant as pf
import f311.explorer as ex
import matplotlib.pyplot as plt


if __name__ == "__main__":
    # Copies files main.dat and abonds.dat to local directory (for given star)
    pf.copy_star(starname="sun-grevesse-1996")
    # Creates symbolic links to all non-star-specific files, such as atomic & molecular lines,
    # partition functions, etc.
    pf.link_to_data()

    # # First run
    # Creates object that will run the four Fortran executables (innewmarcs, hydro2, pfant, nulbad)
    obj = pf.Combo()
    # synthesis interval start (angstrom)
    obj.conf.opt.llzero = 6530
    # synthesis interval end (angstrom)
    obj.conf.opt.llfin = 6535

    # Runs Fortrans and hangs until done
    obj.run()

    # Loads result files into memory. obj.result is a dictionary containing elements ...
    obj.load_result()
    print("obj.result = {}".format(obj.result))
    res = obj.result
    plt.figure()
    ex.draw_spectra_overlapped([res["norm"], res["convolved"]])
    plt.savefig("norm-convolved.png")
    plt.show()
```

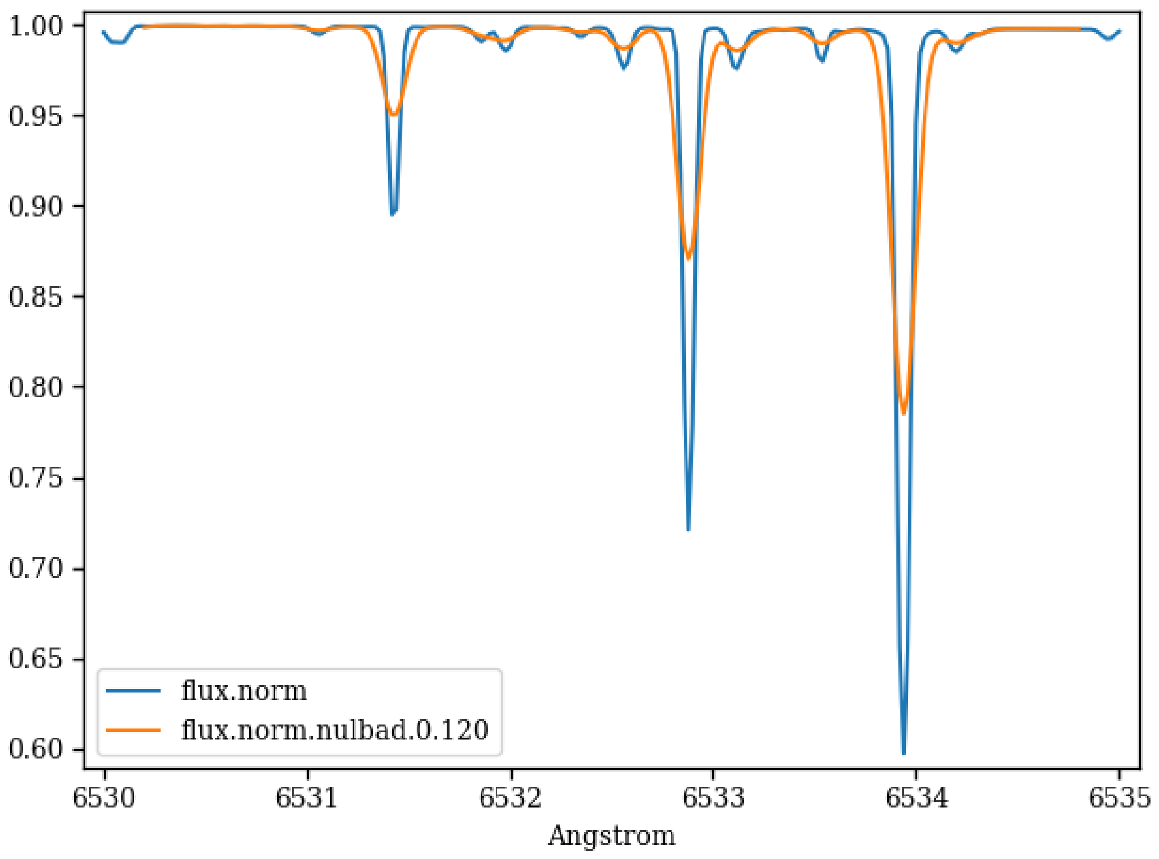

Figure 3.1: – `pfant` (file "flux.norm") and `nulbad` outputs.

3.2 Spectral synthesis - convolutions

The following example convolves the synthetic spectrum (file "flux.norm") with Gaussian profiles of different FWHMs (Figure 3.2).

```python
#!/usr/bin/env python

"""Runs synthesis over short wavelength range, then plots normalized and convolved spectrum"""

import f311.pyfant as pf
import f311.explorer as ex
import matplotlib.pyplot as plt
import a99

# FWHM (full width at half of maximum) of Gaussian profiles in angstrom
FWHMS = [0.03, 0.06, 0.09, 0.12, 0.15, 0.20, 0.25, 0.3, 0.5]

if __name__ == "__main__":
    # Copies files main.dat and abonds.dat to local directory (for given star)
    pf.copy_star(starname="sun-grevesse-1996")
    # Creates symbolic links to all non-star-specific files
    pf.link_to_data()

    # # 1) Spectral synthesis
    # Creates object that will run the four Fortran executables (innewmarcs, hydro2, pfant, nulbad)
    ecombo = pf.Combo()
    # synthesis interval start (angstrom)
    ecombo.conf.opt.llzero = 6530
    # synthesis interval end (angstrom)
    ecombo.conf.opt.llfin = 6535
    # Runs Fortrans and hangs until done
    ecombo.run()
    ecombo.load_result()
    # Retains un-convolved spectrum for comparison
    spectra = [ecombo.result["norm"]]

    # # 2) Convolutions
    for fwhm in FWHMS:
        enulbad = pf.Nulbad()
        enulbad.conf.opt.fwhm = fwhm
        enulbad.run()
        enulbad.load_result()
        # Appends convolved spectrum for comparison
        spectra.append(enulbad.result["convolved"])

    # # 3) Plots
    plt.figure()
    ex.draw_spectra_overlapped(spectra)
    K = 1.1
    a99.set_figure_size(plt.gcf(), 1000*K, 500*K)
    plt.tight_layout()
    plt.savefig("many-convs.png")
    plt.show()
```

3.3 Spectral synthesis - Continuum

The following code generates Figure 3.3.

```python
"""Runs synthesis over large wavelength range, then plots continuum"""

```

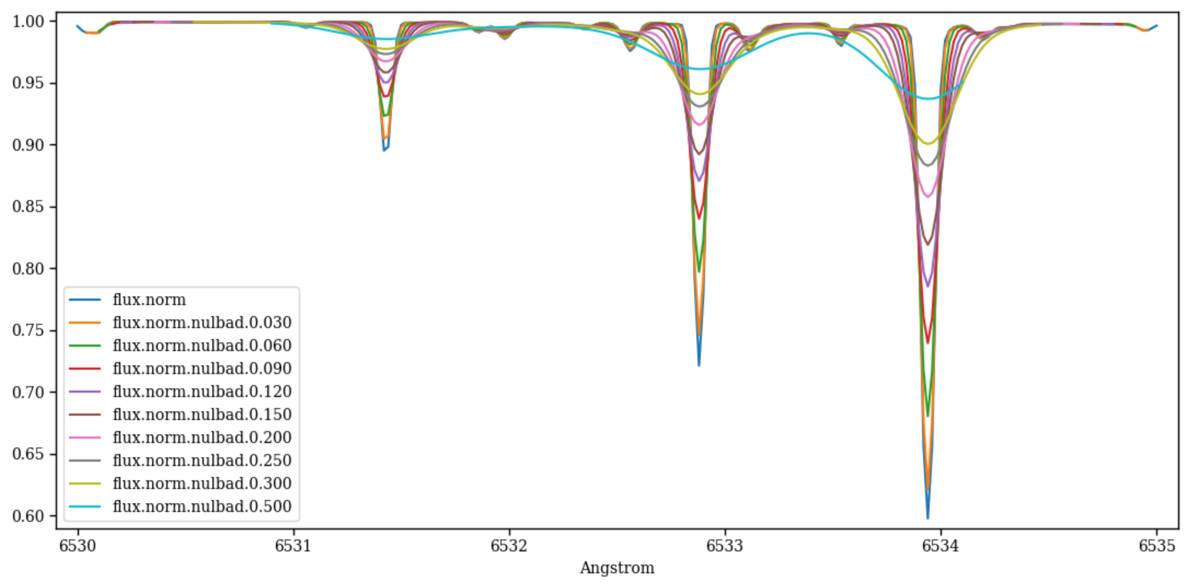

Figure 3.2: – single pfant output and several nulbad outputs.

```python
import f311.pyfant as pf
import f311.explorer as ex
import matplotlib.pyplot as plt
import a99

if __name__ == "__main__":
    # Copies files main.dat and abonds.dat to local directory (for given star)
    pf.copy_star(starname="sun-grevesse-1996")
    # Creates symbolic links to all non-star-specific files, such as atomic & molecular lines,
    # partition functions, etc.
    pf.link_to_data()

    # Creates object that will run the four Fortran executables (innewmarcs, hydro2, pfant, nulbad)
    obj = pf.Combo()
    oo = obj.conf.opt
    # synthesis interval start (angstrom)
    oo.llzero = 2500
    # synthesis interval end (angstrom)
    oo.llfin = 30000
    # savelength step (angstrom)
    oo.pas = 1.
    # Turns off hydrogen lines
    oo.no_h = True
    # Turns off atomic lines
    oo.no_atoms = True
    # Turns off molecular lines
    oo.no_molecules = True

    obj.run()
    obj.load_result()
    print("obj.result = {}".format(obj.result))
    res = obj.result
    ex.draw_spectra_stacked([res["cont"]], setup=ex.PlotSpectrumSetup(fmt_ylabel=None))
    K = .75
    a99.set_figure_size(plt.gcf(), 1300*K, 450*K)
    plt.tight_layout()
    plt.savefig("continuum.png")
    plt.show()
```

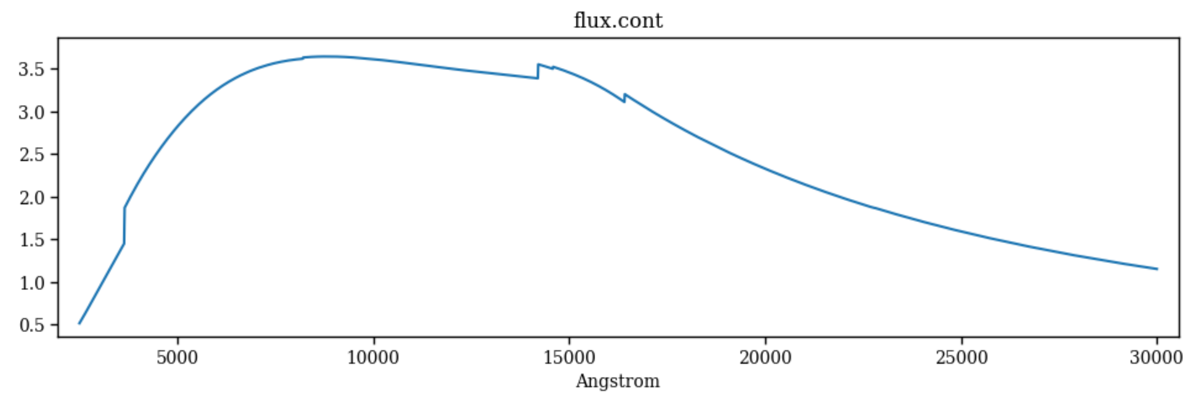

Figure 3.3: – continuum.

3.4 Spectral synthesis - Separate atomic species

PFANT atomic lines files contains wavelength, log_gf and other tabulated information for several (element, ionization level) atomic species.

The following code calculates isolated atomic spectra for a list of arbitrarily chosen atomic species (Figure 3.4).

```python
"""Runs synthesis for specified atomic species separately. No molecules or hydrogen lines."""

import f311.pyfant as pf
import f311.explorer as ex
import matplotlib.pyplot as plt
import f311.filetypes as ft
import a99

# ["<element name><ionization level>", ...] for which to draw panels
MY_SPECIES = ["Fe1", "Fe2", "Ca1", "Ca2", "Na1", "Si1"]

if __name__ == "__main__":
    pf.copy_star(starname="sun-grevesse-1996")
    pf.link_to_data()

    # Loads full atomic lines file
    fatoms = ft.FileAtoms()
    fatoms.load()

    runnables = []
    for elem_ioni in MY_SPECIES:
        atom = fatoms.find_atom(elem_ioni)

        # Creates atomic lines file object containing only one atom
        fatoms2 = ft.FileAtoms()
        fatoms2.atoms = [atom]

        ecombo = pf.Combo()
        # Overrides file "atoms.dat" with in-memory object
        ecombo.conf.file_atoms = fatoms2
        ecombo.conf.flag_output_to_dir = True
        oo = ecombo.conf.opt
        # Assigns synthesis range to match atomic lines range
        oo.llzero, oo.llfin = fatoms2.llzero, fatoms2.llfin
        # Turns off hydrogen lines
        oo.no_h = True
        # Turns off molecular lines
        oo.no_molecules = True

        runnables.append(ecombo)

    pf.run_parallel(runnables)

    # Draws figure
    f = plt.figure()
    a99.format_BLB()
    for i, (title, ecombo) in enumerate(zip(MY_SPECIES, runnables)):
        ecombo.load_result()
        plt.subplot(2, 3, i+1)
        ex.draw_spectra_overlapped([ecombo.result["spec"]],
                                   setup=ex.PlotSpectrumSetup(flag_xlabel=i/3 >= 1, flag_
↪legend=False))
        plt.title(title)

    K = 1.
    a99.set_figure_size(plt.gcf(), 1300*K, 740*K)
    plt.tight_layout()
    plt.savefig("synthesis-atoms.png")
    plt.show()
```

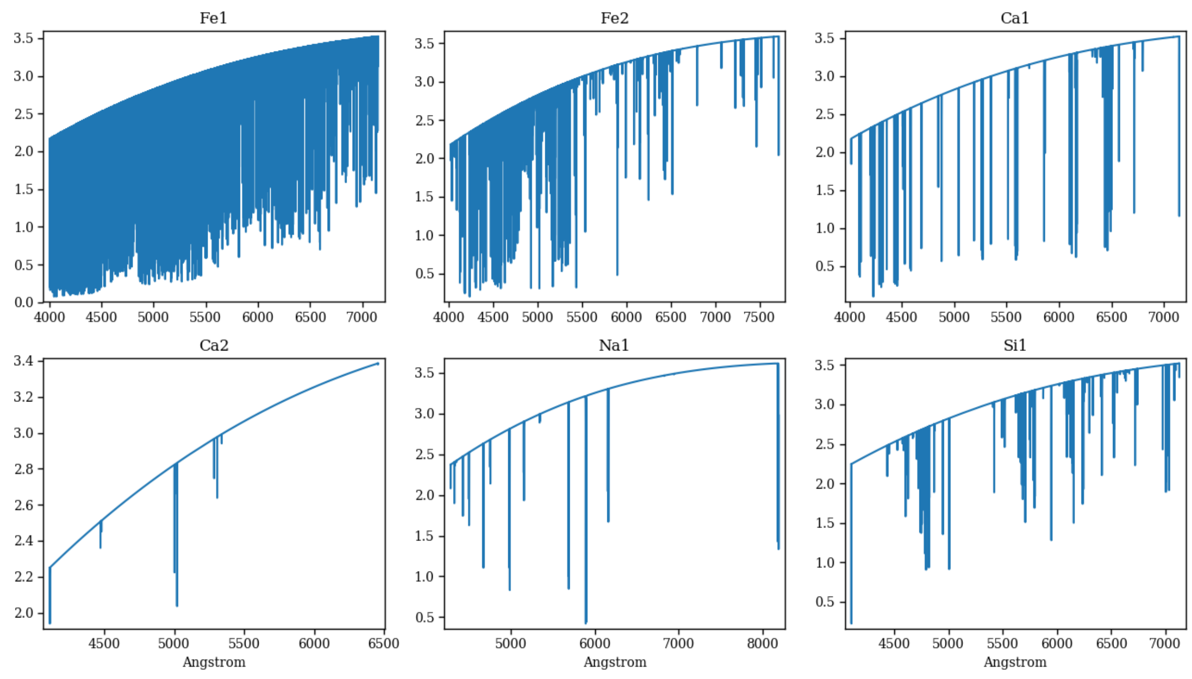

Figure 3.4: – atomic lines synthesized separately for each species.

3.5 Spectral synthesis - Separate molecules

The following code generates Figure 3.5, Figure 3.6, and additional plots not shown here.

```
"""Runs synthesis for molecular species separately. No atomic nor hydrogen lines."""

import f311.pyfant as pf
import f311.explorer as ex
import matplotlib.pyplot as plt
import f311.filetypes as ft
import a99

SUBPLOT_NUM_ROWS = 2
SUBPLOT_NUM_COLS = 2

if __name__ == "__main__":
    pf.copy_star(starname="sun-grevesse-1996")
    pf.link_to_data()

    # Loads full molecular lines file
    fmol = ft.FileMolecules()
    fmol.load()


    runnables = []
    for molecule in fmol:
        fmol2 = ft.FileMolecules()
        fmol2.molecules = [molecule]

        ecombo = pf.Combo()
        # Overrides file "molecules.dat" with in-memory object
        ecombo.conf.file_molecules = fmol2
        ecombo.conf.flag_output_to_dir = True
        oo = ecombo.conf.opt
        # Assigns synthesis range to match atomic lines range
        oo.llzero, oo.llfin = fmol2.llzero, fmol2.llfin
        # Turns off hydrogen lines
        oo.no_h = True
        # Turns off atomic lines
        oo.no_atoms = True
        # Adjusts the wavelength step according to the calculation interval
        oo.pas = max(1, round(oo.llfin*1./20000/2.5)*2.5)
        oo.aint = max(50., oo.pas)

        runnables.append(ecombo)

    pf.run_parallel(runnables)

    num_panels = SUBPLOT_NUM_COLS*SUBPLOT_NUM_ROWS
    num_molecules = len(runnables)
    ifigure = 0
    a99.format_BLB()
    for i in range(num_molecules+1):
        not_first = i > 0
        first_panel_of_figure = (i / num_panels - int(i / num_panels)) < 0.01
        is_panel = i < num_molecules

        if not_first and (not is_panel or first_panel_of_figure):
            plt.tight_layout()
            K = 1.
            a99.set_figure_size(plt.gcf(), 1500 * K, 740 * K)
            plt.tight_layout()
            filename_fig ="synthesis-molecules-{}.png".format(ifigure)
            print("Saving figure '{}'...".format(filename_fig))
            plt.savefig(filename_fig)
            plt.close()
            ifigure += 1

        if first_panel_of_figure and is_panel:
            plt.figure()

        if is_panel:
            ecombo = runnables[i]
            ecombo.load_result()

            isubplot = i % num_panels + 1
            plt.subplot(SUBPLOT_NUM_ROWS, SUBPLOT_NUM_COLS, isubplot)
            ex.draw_spectra_overlapped([ecombo.result["spec"]],
               setup=ex.PlotSpectrumSetup(flag_xlabel=i/3 >= 1, flag_legend=False))

            _title = fmol[i].description
            if "]" in _title:
                title = _title[:_title.index("]")+1]
            else:
                title = _title[:20]
            plt.title(title)
```

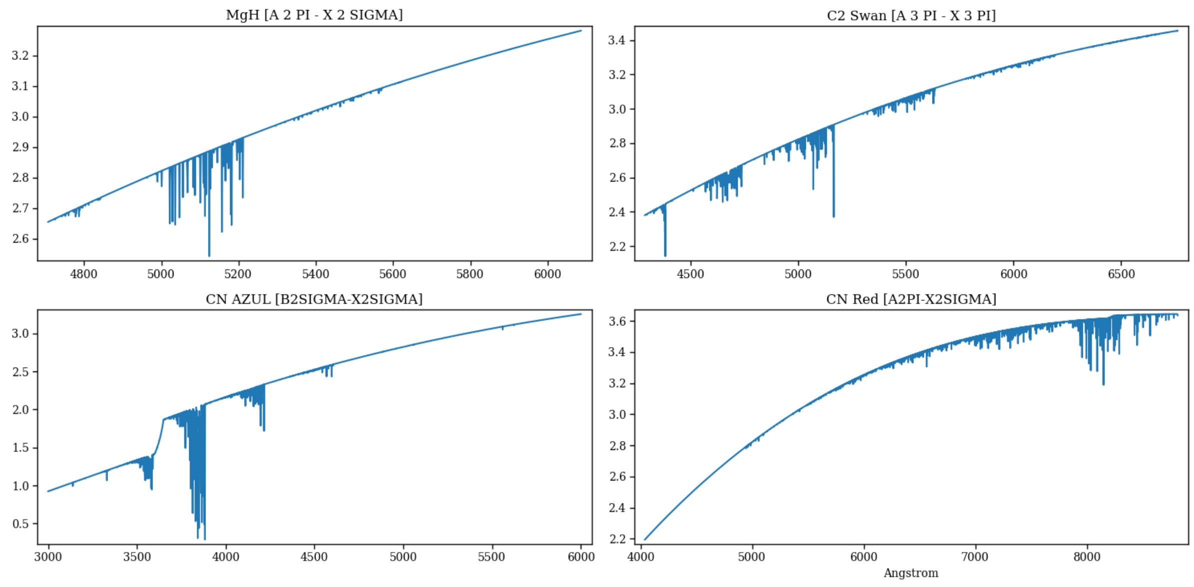

Figure 3.5: – molecular lines synthesized separately for each species

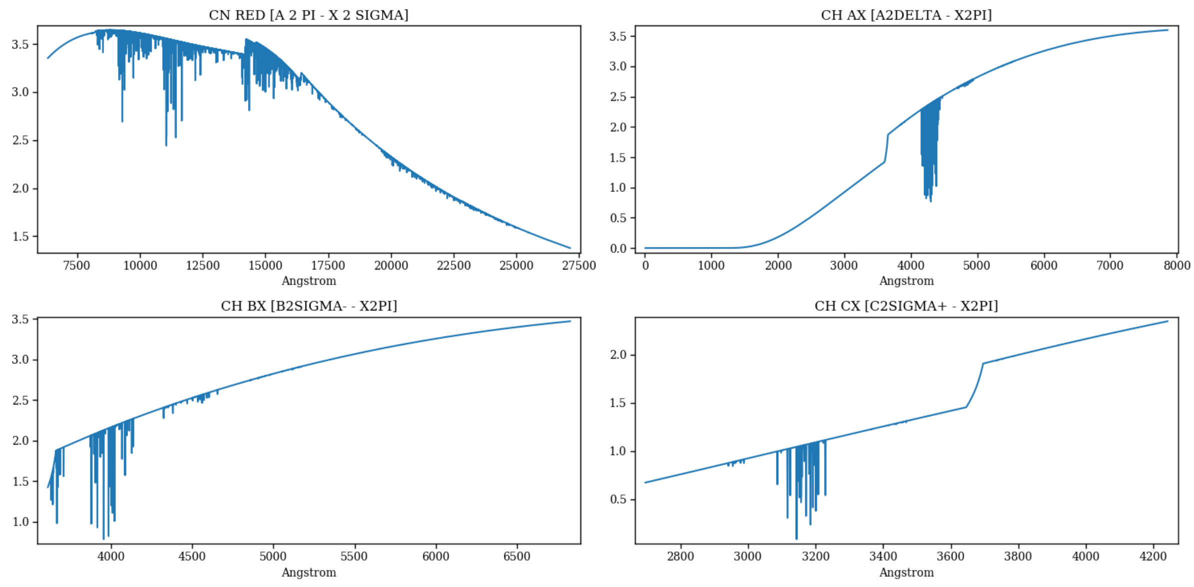

Figure 3.6: – molecular lines synthesized separately for each species.

3.6 Gaussian profiles as nulbad outputs

nulbad is one of the Fortran executables of the PFANT package. It is the one that convolves the synthetic spectrum calculated by pfant with a Gaussian profile specified by a “fwhm” parameter (Figure 3.7).

```
"""
Nulbad's "impulse response"

Saves "impulse" spectrum (just a spike at lambda=5000 angstrom) as "flux.norm",
then runs `nulbad` repeatedly to get a range of Gaussian profiles.

"""

import f311.pyfant as pf
import f311.explorer as ex
import matplotlib.pyplot as plt
import a99
import f311.filetypes as ft
import numpy as np


# FWHM (full width at half of maximum) of Gaussian profiles in angstrom
FWHMS = [0.03, 0.06, 0.09, 0.12, 0.15, 0.20, 0.25, 0.3]

if __name__ == "__main__":
    # Copies files main.dat and abonds.dat to local directory (for given star)
    pf.copy_star(starname="sun-grevesse-1996")
    # Creates symbolic links to all non-star-specific files
    pf.link_to_data()

    # # 1) Creates "impulse" spectrum
    fsp = ft.FileSpectrumPfant()
    sp = ft.Spectrum()
    N = 2001
    sp.x = (np.arange(0, N, dtype=float)-(N-1)/2)*0.001+5000
    sp.y = np.zeros((N,), dtype=float)
    sp.y[int((N-1)/2)] = 1.

    fsp.spectrum = sp
    fsp.save_as("flux.norm")

    # # 2) Convolutions
    spectra = []
    for fwhm in FWHMS:
        enulbad = pf.Nulbad()
        enulbad.conf.opt.fwhm = fwhm
        enulbad.run()
        enulbad.load_result()
        enulbad.clean()
        # Appends convolved spectrum for comparison
        spectra.append(enulbad.result["convolved"])

    # # 3) Plots
    f = plt.figure()
    ex.draw_spectra_overlapped(spectra)
    K = 0.7
    a99.set_figure_size(plt.gcf(), 1300*K, 500*K)
    plt.tight_layout()
    plt.savefig("gaussian-profiles.png")
    plt.show()
```

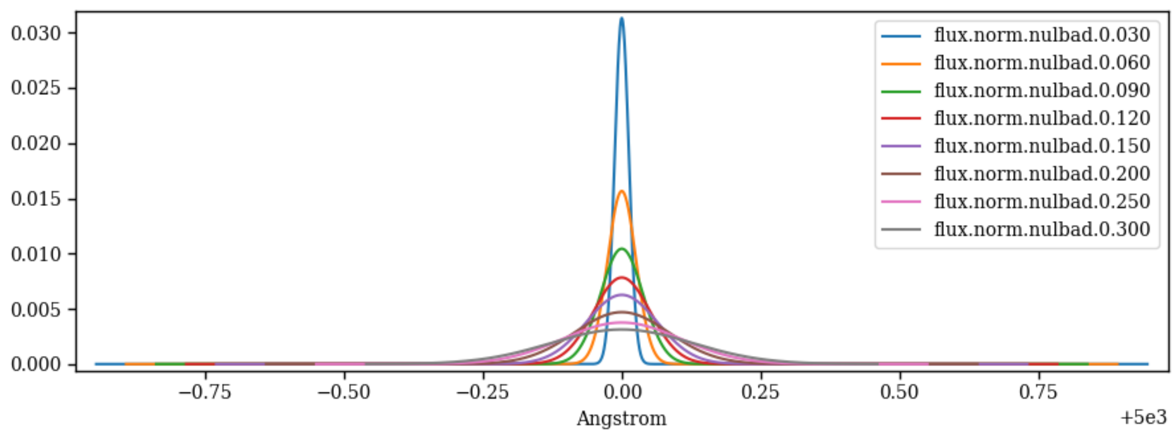

Figure 3.7: – Gaussian profiles illustrated for different FWHMs.

3.7 Plot hydrogen profiles

The following code generates Figure 3.8.

```
"""
Calculates hydrogen lines profiles, then plots them in several 3D subplots
"""

import pyfant
import a99
import os
import shutil
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # yes, required (see below)

def mylog(*args):
    print("^^ {}".format(", ".join(args)))


def main(flag_cleanup=True):
    tmpdir = a99.new_filename("hydrogen-profiles")

    # Saves current directory
    pwd = os.getcwd()
    mylog("Creating directory '{}'...".format(tmpdir))
    os.mkdir(tmpdir)
    try:
        pyfant.link_to_data()
        _main()
    finally:
        # Restores current directory
        os.chdir(pwd)
        # Removes temporary directory
        if flag_cleanup:
            mylog("Removing directory '{}'...".format(tmpdir))
            shutil.rmtree(tmpdir)
        else:
            mylog("Not cleaning up.")


def _main():
    fm = pyfant.FileMain()
    fm.init_default()
    fm.llzero, fm.llfin = 1000., 200000.  # spectral synthesis range in Angstrom

    ei = pyfant.Innewmarcs()
    ei.conf.file_main = fm
    ei.run()
    ei.clean()

    eh = pyfant.Hydro2()
    eh.conf.file_main = fm
    eh.run()
    eh.load_result()
    eh.clean()

    _plot_profiles(eh.result["profiles"])


def _plot_profiles(profiles):
    fig = plt.figure()
    i = 0
    for filename, ftoh in profiles.items():
        if ftoh is not None:
            mylog("Drawing '{}'...".format(filename))
            # ax = plt.subplot(2, 3, i+1)
            ax = fig.add_subplot(2, 3, i+1, projection='3d')
            ax.set_title(filename)
            pyfant.draw_toh(ftoh, ax)
            i += 1

    plt.tight_layout()
    plt.savefig("hydrogen-profiles.png")
    plt.show()


if __name__ == "__main__":
    main(flag_cleanup=True)
```

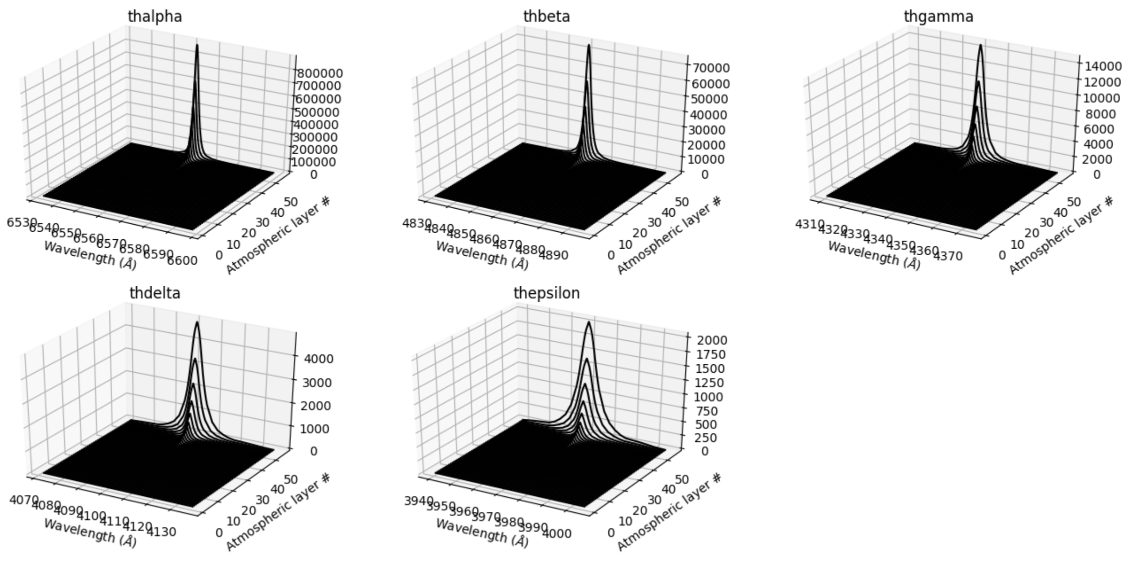

Figure 3.8: – hydrogen lines profiles as calculated by hydro2.

3.8 Import Kurucz' molecular linelist file

```
"""
This example loads file "c2dabrookek.asc" and prints a memory representation of its first line.

This file can be obtained at http://kurucz.harvard.edu/molecules/c2/. First lines of file:

```
  287.7558-14.533 23.0 2354.082 24.0 -37095.578 6063a00e1 3d10e3 12 677 34741.495
  287.7564-14.955 22.0 2282.704 23.0 -37024.124 6063a00f1 3d10f3 12 677 34741.419
  287.7582-14.490 21.0 2214.696 22.0 -36955.900 6063a00e1 3d10e3 12 677 34741.205
  287.7613-15.004 24.0 2428.453 25.0 -37169.280 6063a00f1 3d10f3 12 677 34740.828
  287.7650-14.899 20.0 2149.765 21.0 -36890.147 6063a00f1 3d10f3 12 677 34740.382
```
"""

import f311.filetypes as ft

f = ft.load_any_file("c2dabrookek.asc")

print(repr(f[0]).replace(", ", ",\n              "))
```

This code should print the following:

```
KuruczMolLine(lambda_=2877.558,
              loggf=-14.533,
              J2l=23.0,
              E2l=2354.082,
              Jl=24.0,
              El=37095.578,
              atomn0=6,
              atomn1=6,
              state2l='a',
              v2l=0,
              lambda_doubling2l='e',
              spin2l=1,
              statel='d',
              vl=10,
              lambda_doublingl='e',
              spinl=3,
              iso=12)
```

CHAPTER

FOUR

INDEX OF APPLICATIONS (SCRIPTS)

This chapter is a reference to all scripts in project PyFANT

4.1 Script `copy-star.py`

```
usage: copy-star.py [-h] [-l] [-p] [directory]

Copies stellar data files (such as main.dat, abonds.dat, dissoc.dat) to local directory

examples of usage:
  > copy-star.py
  (displays menu)

  > copy-star.py arcturus
  ("arcturus" is the name of a subdirectory of PFANT/data)

  > copy-star.py -p /home/user/pfant-common-data
  (use option "-p" to specify path)

  > copy-star.py -l
  (lists subdirectories of PFANT/data , doesn't copy anything)

positional arguments:
  directory   name of directory (either a subdirectory of PFANT/data or the
              path to a valid system directory (see modes of operation)
              (default: None)

optional arguments:
  -h, --help  show this help message and exit
  -l, --list  lists subdirectories of
              /home/j/Documents/projects/astro/github/PFANT/code/data
              (default: False)
  -p, --path  system path mode (default: False)
```

This script belongs to package *pyfant*

4.2 Script `create-grid.py`

```
usage: create-grid.py [-h] [--pattern [PATTERN]] [-m [{opa,modtxt,modbin}]]
                      [fn_output]

Merges several atmospheric models into a single file (_i.e._, the "grid")

"Collects" several files in current directory and creates a single file
containing atmospheric model grid.
```

```

Working modes (option "-m"):
 "opa" (default mode): looks for MARCS[1] ".mod" and ".opa" text file pairs and
                       creates a *big* binary file containing *all* model
                       information including opacities.
                       Output will be in ".moo" format.

 "modtxt": looks for MARCS ".mod" text files only. Resulting grid will not contain
           opacity information.
           Output will be in binary ".mod" format.

 "modbin": looks for binary-format ".mod" files. Resulting grid will not contain
           opacity information.
           Output will be in binary ".mod" format.

References:
  [1] http://marcs.astro.uu.se/

.
.
.

positional arguments:
  fn_output             output file name (default: "grid.moo" or "grid.mod",
                        depending on mode)

optional arguments:
  -h, --help            show this help message and exit
  --pattern [PATTERN]   file name pattern (with wildcards) (default: *.mod)
  -m [{opa,modtxt,modbin}], --mode [{opa,modtxt,modbin}]
                        working mode (see description above) (default: opa)
```

This script belongs to package *pyfant*

4.3 Script `cut-atoms.py`

```
usage: cut-atoms.py [-h] llzero llfin fn_input fn_output

Cuts atomic lines file to wavelength interval specified

The interval is [llzero, llfin]

positional arguments:
  llzero      lower wavelength boundary (angstrom)
  llfin       upper wavelength boundary (angstrom)
  fn_input    input file name
  fn_output   output file name

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.4 Script `cut-molecules.py`

```
usage: cut-molecules.py [-h] llzero llfin fn_input fn_output

Cuts molecular lines file to wavelength interval specified
```

```

The interval is [llzero, llfin]

positional arguments:
  llzero      lower wavelength boundary (angstrom)
  llfin       upper wavelength boundary (angstrom)
  fn_input    input file name
  fn_output   output file name

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.5 Script `hitran-scraper.py`

```
usage: hitran-scraper.py [-h] [-t T] [M] [I] [llzero] [llfin]

Retrieves molecular lines from the HITRAN database [Gordon2016]

This script uses web scraping and the HAPI to save locally molecular lines from the HITRAN database.

While the HAPI provides the downloading facility, web scraping is used to get the lists of molecules
and isotopologues from the HITRAN webpages and get the IDs required to run the HAPI query.

The script is typically invoked several times, each time with an additional argument.

References:

[Gordon2016] I.E. Gordon, L.S. Rothman, C. Hill, R.V. Kochanov, Y. Tan, P.F. Bernath, M. Birk,
    V. Boudon, A. Campargue, K.V. Chance, B.J. Drouin, J.-M. Flaud, R.R. Gamache, J.T. Hodges,
    D. Jacquemart, V.I. Perevalov, A. Perrin, K.P. Shine, M.-A.H. Smith, J. Tennyson, G.C. Toon,
    H. Tran, V.G. Tyuterev, A. Barbe, A.G. Császár, V.M. Devi, T. Furtenbacher, J.J. Harrison,
    J.-M. Hartmann, A. Jolly, T.J. Johnson, T. Karman, I. Kleiner, A.A. Kyuberis, J. Loos,
    O.M. Lyulin, S.T. Massie, S.N. Mikhailenko, N. Moazzen-Ahmadi, H.S.P. Müller, O.V. Naumenko,
    A.V. Nikitin, O.L. Polyansky, M. Rey, M. Rotger, S.W. Sharpe, K. Sung, E. Starikova,
    S.A. Tashkun, J. Vander Auwera, G. Wagner, J. Wilzewski, P. Wcisło, S. Yu, E.J. Zak,
    The HITRAN2016 Molecular Spectroscopic Database, J. Quant. Spectrosc. Radiat. Transf. (2017).
    doi:10.1016/j.jqsrt.2017.06.038.

positional arguments:
  M           HITRAN molecule number (default: (lists molecules))
  I           HITRAN isotopologue number (not unique, starts over at each
              molecule) (default: (lists isotopologues))
  llzero      Initial wavelength (Angstrom) (default: None)
  llfin       Final wavelength (Angstrom) (default: None)

optional arguments:
  -h, --help  show this help message and exit
  -t T        Table Name (default: (molecular formula))
```

This script belongs to package *pyfant*

4.5.1 Usage examples

```
$ hitran-scraper.py

List of all HITRAN molecules
```

```
============================

  ID  Formula    Name
----  ---------  --------------------
   1  H2O        Water
   2  CO2        Carbon Dioxide
   3  O3         Ozone
   4  N2O        Nitrous Oxide
   5  CO         Carbon Monoxide
   6  CH4        Methane
   7  O2         Molecular Oxygen
   8  NO         Nitric Oxide
   9  SO2        Sulfur Dioxide
  10  NO2        Nitrogen Dioxide
  11  NH3        Ammonia
  12  HNO3       Nitric Acid
  13  OH         Hydroxyl Radical
  14  HF         Hydrogen Fluoride
  15  HCl        Hydrogen Chloride
  16  HBr        Hydrogen Bromide
  17  HI         Hydrogen Iodide
  18  ClO        Chlorine Monoxide
  19  OCS        Carbonyl Sulfide
  20  H2CO       Formaldehyde
  21  HOCl       Hypochlorous Acid
  22  N2         Molecular Nitrogen
  23  HCN        Hydrogen Cyanide
  24  CH3Cl      Methyl Chloride
  25  H2O2       Hydrogen Peroxide
  26  C2H2       Acetylene
  27  C2H6       Ethane
  28  PH3        Phosphine
  29  COF2       Carbonyl Fluoride
  31  H2S        Hydrogen Sulfide
  32  HCOOH      Formic Acid
  33  HO2        Hydroperoxyl Radical
  34  O          Oxygen Atom
  36  NO+        Nitric Oxide Cation
  37  HOBr       Hypobromous Acid
  38  C2H4       Ethylene
  39  CH3OH      Methanol
  40  CH3Br      Methyl Bromide
  41  CH3CN      Methyl Cyanide
  43  C4H2       Diacetylene
  44  HC3N       Cyanoacetylene
  45  H2         Molecular Hydrogen
  46  CS         Carbon Monosulfide
  47  SO3        Sulfur trioxide

Now, to list isotopologues for a given molecule, please type:

    hitran-scraper.py <molecule ID>

where <molecule ID> is one of the IDs listed above.
```

Now suppose we want the molecule OH molecule:

```
$ hitran-scraper.py 13

List of all isotopologues for molecule 'OH' (Hydroxyl Radical)
==============================================================

m_formula      ID    ID_molecule  Formula      AFGL_Code  Abundance
```

```
-----------  ----  -------------  ---------  -----------  ---------------
OH              1             13  (16)OH              61  0.997473
OH              2             13  (18)OH              81  0.002
OH              3             13  (16)OD              62  1.553710 × 10-4


Now, to download lines, please type:

    hitran-scraper.py 13 <isotopologue ID> <llzero> <llfin>

where <isotopologue ID> is one the numbers from the 'ID' column above,

and [<llzero>, <llfin>] defines the wavelength interval in Angstrom.
```

Now selecting the first isotopologue and specifying the visible wavelength range:

```
$ hitran-scraper.py 13 1 3000 7000

Isotopologue selected:
======================

Field name    Value
------------  --------
m_formula     OH
ID            1
ID_molecule   13
Formula       (16)OH
AFGL_Code     61
Abundance     0.997473

Wavelength interval (air): [3000.0, 7000.0] Angstrom
Wavenumber interval (vacuum): [14289.61969369552, 33342.42546386186] cm**-1
Table name: '(16)OH'

Fetching data...
===
=== BEGIN messages from HITRAN API ===
===
BEGIN DOWNLOAD: (16)OH
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
  65536 bytes written to ./(16)OH.data
Header written to ./(16)OH.header
END DOWNLOAD
                     Lines parsed: 3855
PROCESSED
===
=== END messages from HITRAN API ===
===
...done
Please check files '(16)OH.header', '(16)OH.data'
```

4.5.2 Quick note on the HITRAN API

The files created ('(16)OH.header', '(16)OH.data') can be opened using the HAPI. They are also accessed by the application `convmol.py`.

The HAPI can be downloaded, but one version is also included with the f311 package. The following is an example of how the HITRAN data could be accessed from the Python console:

```
>>> from f311 import hapi
>>> hapi.loadCache()
Using .
(16)OH
                     Lines parsed: 3855
>>> oh_data = hapi.LOCAL_TABLE_CACHE["(16)OH"]
>>> oh_data.keys()
dict_keys(['data', 'header'])
>>> oh_data["data"].keys()
dict_keys(['ierr', 'gpp', 'molec_id', 'global_lower_quanta', 'sw', 'gamma_self', 'n_air', 'elower',
↪'line_mixing_flag', 'local_lower_quanta', 'gp', 'global_upper_quanta', 'gamma_air', 'local_upper_
↪quanta', 'iref', 'local_iso_id', 'delta_air', 'nu', 'a'])
```

To work properly with these data in your code, you may have a look at the HAPI source code and manual, as this library is superbly documented.

Within f311, the code in `f311.convmol.conv_hitran.hitran_to_sols()` contains a usage example of HITRAN data.

4.6 Script `link.py`

```
usage: link.py [-h] [-l] [-p] [-y] [directory]

Creates symbolic links to PFANT data files as an alternative to copying these (sometimes large)␣
↪files into local directory

A star is specified by three data files whose typical names are:
main.dat, abonds.dat, and dissoc.dat .

The other data files (atomic/molecular lines, partition function, etc.)
are star-independent, and this script is a proposed solution to keep you from
copying these files for every new case.

How it works: link.py will look inside a given directory and create
symbolic links to files *.dat and *.mod.

The following files will be skipped:
  - main files, e.g. "main.dat"
  - dissoc files, e.g., "dissoc.dat"
  - abonds files, e.g., "abonds.dat"
  - .mod files with a single model inside, e.g., "modeles.mod"
  - hydrogen lines files, e.g., "thalpha", "thbeta"

This script works in two different modes:

a) default mode: looks for files in a subdirectory of PFANT/data
   > link.py common
   (will create links to filess inside PFANT/data/common)

b) "-l" option: lists subdirectories of PFANT/data

c) "-p" option: looks for files in a directory specified.
   Examples:
```

```
   > link.py -p /home/user/pfant-common-data
   > link.py -p ../../pfant-common-data

Note: in Windows, this script must be run as administrator.

positional arguments:
  directory   name of directory (either a subdirectory of PFANT/data or the
              path to a valid system directory (see modes of operation)
              (default: common)

optional arguments:
  -h, --help  show this help message and exit
  -l, --list  lists subdirectories of
              /home/j/Documents/projects/astro/github/PFANT/code/data
              (default: False)
  -p, --path  system path mode (default: False)
  -y, --yes   Automatically answers 'yes' to eventual question (default:
              False)
```

This script belongs to package *pyfant*

4.7 Script `merge-molecules.py`

```
usage: merge-molecules.py [-h] [-o [FN_OUTPUT]] files [files ...]

Merges several PFANT molecular lines file into a single one

positional arguments:
  files                 files specification: list of files, wildcards allowed

optional arguments:
  -h, --help            show this help message and exit
  -o [FN_OUTPUT], --fn_output [FN_OUTPUT]
                        output filename. If not specified, creates file such
                        as 'molecules-merged-.0000.dat' (default: (automatic))
```

This script belongs to package *pyfant*

4.8 Script `nist-scraper.py`

```
usage: nist-scraper.py [-h] [-u] formula

Retrieves and prints a table of molecular constants from the NIST Chemistry Web Book [NISTRef]

To do so, it uses web scraping to navigate through several pages and parse the desired information
from the book web pages.

It does not provide a way to list the molecules yet, but will give an error if the molecule is not
found in the NIST web book.

Example:

    print-nist.py OH

**Note** This script was designed to work with **diatomic molecules** and may not work with other
         molecules.

**Warning** The source material online was known to contain mistakes (such as an underscore instead
            of a minus signal to indicate a negative number). We have identified a few of these,
            and build some workarounds. However, we recommend a close look at the information parsed
            before use.

**Disclaimer** This script may stop working if the NIST people update the Chemistry Web Book.

References:

[NISTRef] http://webbook.nist.gov/chemistry/

positional arguments:
  formula        NIST formula

optional arguments:
  -h, --help     show this help message and exit
  -u, --unicode  Unicode output (default is to contain only ASCII characters)
                 (default: False)
```

This script belongs to package *pyfant*

4.8.1 Usage examples

Usage examples:

```
nist-scraper.py TiO
```

will print

```
*** titanium oxide ***

State       T_e           omega_e    omega_ex_e    omega_ey_e      B_e    alpha_e    gamma_e      ␣
↪D_e     beta_e       r_e  Trans.        nu_00   A
----------  -----------  ---------  ------------  ------------  -------  ---------  ---------  -----
↪---  --------  -------  ---------  --------  ---
D           31920.0        1040                                                                    ␣
↪                         D <-> X    31940
e 1Sigma+   a + 26598.1     845.2        4.2                    0.4892     0.0023              4.7e-
↪07          1.695     e <-> d R  24297.5
f 1Delta    (a + 19132)     890                                 0.50221                        6.4e-
↪07          1.67292  f <-> a R  19068.9
c 1Phi      a + 17890.2     909.6        4.19                   0.523      0.00313             3.9e-
↪07          1.6393   c <-> a R  17840.6
C 3Delta_r  19617.0         838.26       4.76          0.047    0.48989    0.00306   -3e-05    6.7e-
↪07          1.69383  C <-> X R  19334
B 3Pi_r     16331.3         875          5                      0.50617                        6.
↪86e-07         1.66636  B <-> X R  16066.7
b 1Pi       a+11322.0_3     911.2        3.72                   0.51337    0.00291             6.1e-
↪07          1.65464  b <-> d R   9054.02
A 3Phi_r    14431.0         867.78       3.942                  0.50739    0.00315   -1e-05    6.
↪92e-07     2e-09  1.66436  A <-> X R  14163
E 3Pi       12025.0         924.2        5.1                                                       ␣
↪                         E <-> X    11871
d 1Sigma+   a + 2215.6     1014.6        4.64                   0.54922    0.00337             6e-
↪07            1.59972
a 1Delta    a              1009.3        3.93                   0.5376     0.00298             5.9e-
↪07          1.61692
X 3Delta_r  197.5          1009.02       4.498        -0.0107   0.53541    0.00301   -1.1e-05  6.
↪03e-07     3e-09  1.62022
```

4.9 Script `run-multi.py`

```
usage: run-multi.py [-h] [--abs ABS] [--absoru ABSORU] [--aint AINT]
                    [--allow ALLOW] [--amores AMORES] [--convol CONVOL]
                    [--explain EXPLAIN] [--flam FLAM] [--flprefix FLPREFIX]
                    [--fn_abonds FN_ABONDS] [--fn_absoru2 FN_ABSORU2]
                    [--fn_atoms FN_ATOMS] [--fn_cv FN_CV]
                    [--fn_dissoc FN_DISSOC] [--fn_flux FN_FLUX]
                    [--fn_hmap FN_HMAP] [--fn_lines FN_LINES]
                    [--fn_log FN_LOG] [--fn_logging FN_LOGGING]
                    [--fn_main FN_MAIN] [--fn_modeles FN_MODELES]
                    [--fn_modgrid FN_MODGRID] [--fn_molecules FN_MOLECULES]
                    [--fn_moo FN_MOO] [--fn_opa FN_OPA]
                    [--fn_partit FN_PARTIT] [--fn_progress FN_PROGRESS]
                    [--fwhm FWHM] [--interp INTERP] [--kik KIK] [--kq KQ]
                    [--llfin LLFIN] [--llzero LLZERO]
                    [--logging_console LOGGING_CONSOLE]
                    [--logging_file LOGGING_FILE]
                    [--logging_level LOGGING_LEVEL] [--no_atoms NO_ATOMS]
                    [--no_h NO_H] [--no_molecules NO_MOLECULES] [--norm NORM]
                    [--opa OPA] [--pas PAS] [--pat PAT] [--play PLAY]
                    [--sca SCA] [--zinf ZINF] [--zph ZPH] [-f FN_ABXFWHM]
                    [-s CUSTOM_SESSION_ID]

Runs pfant and nulbad in "multi mode" (equivalent to Tab 4 in ``x.py``) (several abundances X FWHM
↪'s)

optional arguments:
  -h, --help            show this help message and exit
  --abs ABS
  --absoru ABSORU
  --aint AINT
  --allow ALLOW
  --amores AMORES
  --convol CONVOL
  --explain EXPLAIN
  --flam FLAM
  --flprefix FLPREFIX
  --fn_abonds FN_ABONDS
  --fn_absoru2 FN_ABSORU2
  --fn_atoms FN_ATOMS
  --fn_cv FN_CV
  --fn_dissoc FN_DISSOC
  --fn_flux FN_FLUX
  --fn_hmap FN_HMAP
  --fn_lines FN_LINES
  --fn_log FN_LOG
  --fn_logging FN_LOGGING
  --fn_main FN_MAIN
  --fn_modeles FN_MODELES
  --fn_modgrid FN_MODGRID
  --fn_molecules FN_MOLECULES
  --fn_moo FN_MOO
  --fn_opa FN_OPA
  --fn_partit FN_PARTIT
  --fn_progress FN_PROGRESS
  --fwhm FWHM
  --interp INTERP
  --kik KIK
  --kq KQ
  --llfin LLFIN
  --llzero LLZERO
  --logging_console LOGGING_CONSOLE
  --logging_file LOGGING_FILE
  --logging_level LOGGING_LEVEL
  --no_atoms NO_ATOMS
  --no_h NO_H
  --no_molecules NO_MOLECULES
  --norm NORM
  --opa OPA
  --pas PAS
  --pat PAT
  --play PLAY
  --sca SCA
  --zinf ZINF
  --zph ZPH
  -f FN_ABXFWHM, --fn_abxfwhm FN_ABXFWHM
                        Name of file specifying different abundances and
                        FWHM's (default: abxfwhm.py)
  -s CUSTOM_SESSION_ID, --custom_session_id CUSTOM_SESSION_ID
                        Name of directory where output files will be saved
                        (default: multi-session-<i>)
```

This script belongs to package *pyfant*

4.10 Script run4.py

```
usage: run4.py [-h] [--abs ABS] [--absoru ABSORU] [--aint AINT]
               [--allow ALLOW] [--amores AMORES] [--convol CONVOL]
               [--explain EXPLAIN] [--flam FLAM] [--flprefix FLPREFIX]
               [--fn_abonds FN_ABONDS] [--fn_absoru2 FN_ABSORU2]
               [--fn_atoms FN_ATOMS] [--fn_cv FN_CV] [--fn_dissoc FN_DISSOC]
               [--fn_flux FN_FLUX] [--fn_hmap FN_HMAP] [--fn_lines FN_LINES]
               [--fn_log FN_LOG] [--fn_logging FN_LOGGING] [--fn_main FN_MAIN]
               [--fn_modeles FN_MODELES] [--fn_modgrid FN_MODGRID]
               [--fn_molecules FN_MOLECULES] [--fn_moo FN_MOO]
               [--fn_opa FN_OPA] [--fn_partit FN_PARTIT]
               [--fn_progress FN_PROGRESS] [--fwhm FWHM] [--interp INTERP]
               [--kik KIK] [--kq KQ] [--llfin LLFIN] [--llzero LLZERO]
               [--logging_console LOGGING_CONSOLE]
               [--logging_file LOGGING_FILE] [--logging_level LOGGING_LEVEL]
               [--no_atoms NO_ATOMS] [--no_h NO_H]
               [--no_molecules NO_MOLECULES] [--norm NORM] [--opa OPA]
               [--pas PAS] [--pat PAT] [--play PLAY] [--sca SCA] [--zinf ZINF]
               [--zph ZPH]

Runs the four Fortran binaries in sequence: `innewmarcs`, `hydro2`, `pfant`, `nulbad`

Check session directory "session-<number>" for log files.

optional arguments:
  -h, --help            show this help message and exit
  --abs ABS
  --absoru ABSORU
  --aint AINT
  --allow ALLOW
  --amores AMORES
  --convol CONVOL
  --explain EXPLAIN
  --flam FLAM
  --flprefix FLPREFIX
  --fn_abonds FN_ABONDS
  --fn_absoru2 FN_ABSORU2
  --fn_atoms FN_ATOMS
  --fn_cv FN_CV
  --fn_dissoc FN_DISSOC
  --fn_flux FN_FLUX
  --fn_hmap FN_HMAP
  --fn_lines FN_LINES
  --fn_log FN_LOG
  --fn_logging FN_LOGGING
  --fn_main FN_MAIN
  --fn_modeles FN_MODELES
  --fn_modgrid FN_MODGRID
  --fn_molecules FN_MOLECULES
  --fn_moo FN_MOO
  --fn_opa FN_OPA
  --fn_partit FN_PARTIT
  --fn_progress FN_PROGRESS
  --fwhm FWHM
  --interp INTERP
  --kik KIK
  --kq KQ
  --llfin LLFIN
  --llzero LLZERO
  --logging_console LOGGING_CONSOLE
  --logging_file LOGGING_FILE
  --logging_level LOGGING_LEVEL
  --no_atoms NO_ATOMS
  --no_h NO_H
  --no_molecules NO_MOLECULES
  --norm NORM
  --opa OPA
  --pas PAS
  --pat PAT
  --play PLAY
  --sca SCA
  --zinf ZINF
  --zph ZPH
```

This script belongs to package *pyfant*

4.11 Script `vald3-to-atoms.py`

```
usage: vald3-to-atoms.py [-h] [--min_algf [MIN_ALGF]] [--max_kiex [MAX_KIEX]]
                         [-s [SKIP]]
                         fn_input [fn_output]

Converts VALD3 atomic/molecular lines file to PFANT atomic lines file.

Molecular lines and certain elements are skipped.

Usage examples:

    To skip only H and He:
    > vald3-to-atoms --skip "H, He"

positional arguments:
  fn_input              input file name
  fn_output             output file name (default: atoms-untuned-<fn_input>)

optional arguments:
  -h, --help            show this help message and exit
  --min_algf [MIN_ALGF]
                        minimum algf (log gf) (default: -7)
  --max_kiex [MAX_KIEX]
                        maximum kiex (default: 15)
  -s [SKIP], --skip [SKIP]
                        list of elements to skip (use quotes and separate
                        elements by commas) (default: H, He, F, Ne, P, Ar, Cl,
                        As, Br, Kr, Xe)
```

This script belongs to package *pyfant*

4.12 Script `abed.py`

```
usage: abed.py [-h] [fn]

Abundances file editor

positional arguments:
  fn          abundances file name (default: abonds.dat)

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.13 Script `ated.py`

```
usage: ated.py [-h] [fn]

Atomic lines file editor

positional arguments:
  fn          atoms file name (default: atoms.dat)

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.14 Script `convmol.py`

```
usage: convmol.py [-h] [--fn_moldb [FN_MOLDB]] [--fn_molconsts [FN_MOLCONSTS]]
                  [--fn_config [FN_CONFIG]]

Conversion of molecular lines data to PFANT format

optional arguments:
  -h, --help            show this help message and exit
  --fn_moldb [FN_MOLDB]
                        File name for Database of Molecular Constants
                        (default: moldb.sqlite)
  --fn_molconsts [FN_MOLCONSTS]
                        File name for Molecular constants config file (Python
                        code) (default: configmolconsts.py)
  --fn_config [FN_CONFIG]
                        File name for Configuration file for molecular lines
                        conversion GUI (Python code) (default:
                        configconvmol.py)
```

This script belongs to package *pyfant*

4.15 Script `mained.py`

```
usage: mained.py [-h] [fn]

Main configuration file editor.

positional arguments:
  fn          main configuration file name (default: main.dat)

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.16 Script `mced.py`

```
usage: mced.py [-h] [fn]

Editor for molecular constants file

This application can edit files of class FileMolConsts.

positional arguments:
  fn          Molecular constants file name (default: configmolconsts.py)

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.17 Script `mled.py`

```
usage: mled.py [-h] [fn]

Molecular lines file editor.

positional arguments:
  fn          molecules file name (default: molecules.dat)

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.18 Script moldbed.py

```
usage: moldbed.py [-h] [fn]

Editor for molecules SQLite database

This application can edit files of class FileMolDB.

positional arguments:
  fn          Molecules database file name (default: moldb.sqlite)

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.19 Script optionsed.py

```
usage: optionsed.py [-h] [fn]

PFANT command-line options file editor.

positional arguments:
  fn          PFANT Command-line Options file name (default: options.py)

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*

4.20 Script tune-zinf.py

```
usage: tune-zinf.py [-h] [--min [MIN]] [--max [MAX]] [--inflate [INFLATE]]
                    [--ge_current] [--no_clean]
                    fn_input [fn_output]

Tunes the "zinf" parameter for each atomic line in atomic lines file

The "zinf" parameter is a distance in angstrom from the centre of an atomic
line. It specifies the calculation range for the line:
[centre-zinf, centre+zinf].

This script runs pfant for each atomic line to determine the width of each
atomic line and thus zinf.

Note: pfant is run using most of its default settings and will require the
following files to exist in the current directory:
  - main.dat
  - dissoc.dat
  - abonds.dat
  - modeles.mod
  - partit.dat
  - absoru2.dat

Note: the precision in the zinf found depends on the calculation step ("pas")
specified in main.dat. A higher "pas" means lower precision and a tendency to
get higher zinf's. This is really not critical. pas=0.02 or pas=0.04 should do.
```

```

positional arguments:
  fn_input              input file name
  fn_output             output file name (default: <made-up filename>)

optional arguments:
  -h, --help            show this help message and exit
  --min [MIN]           minimum zinf. If zinf found for a particular line is
                        smaller than this value, this value will be used
                        instead (default: 0.1)
  --max [MAX]           maximum zinf. If zinf found for a particular line is
                        greater than this value, this value will be used
                        instead (default: 50.0)
  --inflate [INFLATE]   Multiplicative constant to apply a "safety margin".
                        Each zinf found will be multiplied by this value. For
                        example a value of INFLATE=1.1 means that all the
                        zinf's saved will be 10 percent larger than those
                        calculated (default: 1.1)
  --ge_current          "Greater or Equal to current": If this option is set,
                        the current zinf in the atomic lines file is used as a
                        lower boundary. (default: False)
  --no_clean            If set, will not remove the session directories.
                        (default: False)
```

This script belongs to package *pyfant*

4.21 Script x.py

```
usage: x.py [-h]

PFANT Launcher -- Graphical Interface for Spectral Synthesis

Single and multi modes.

Multi mode
----------

Runs pfant for different abundances for each element, then run nulbad for each
pfant result for different FWHMs.

The configuration is read from a .py file.

The user must specify a list of FWHM values for nulbad convolutions, and
a dictionary containing element symbols and respective list containing n_abdif
differential abundances to be used for each element.

pfant will be run n_abdif times, each time adding to each element in ab the i-th
value in the vector for the corresponding element.

nulbad will run n_abdif*n_fwhms times, where n_fwhms is the number of different
FWHMs specified.

The result will be
- several spectra saved as  "<star name><pfant name or counter>.sp"
- several "spectra list" files saved as "cv_<FWHM>.spl". As the file indicates,
  each ".spl" file will have the names of the spectrum files for a specific FWHM.
  .spl files are subject to input for lineplot.py by E.Cantelli
---------

optional arguments:
  -h, --help  show this help message and exit
```

This script belongs to package *pyfant*