A new Phosphorus Model Atom for the Tübingen Model-Atom Database and its Application to the Spectral Analysis of a hot sdB Star

Bachelor Thesis

for the acquisition of the academic degree
Bachelor of Science (B. Sc.)

Michael Knörzer

Eberhard Karls University Tübingen
Institute for Astronomy and Astrophysics
Kepler Center for Astro and Particle Physics
Department of Astronomy

May 2018
Hiermit versichere ich, Michael Knörzer, Matrikelnummer 3008274, gemäß §25(4) des allgemeinen Teils der Studien- und Prüfungsordnung der Universität Tübingen für den Studiengang Physik mit akademischer Abschlussprüfung Bachelor of Science (B. Sc.) diese Bachelorarbeit mit dem Titel

“A new Phosphorus Model Atom for the Tübingen Model-Atom Database and its Application to the Spectral Analysis of a hot sdB Star”

selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet zu haben. Die Arbeit war weder vollständig noch in wesentlichen Teilen Gegenstand eines Prüfungsverfahrens und wurde weder vollständig noch in wesentlichen Teilen bereits veröffentlicht. Das in Dateiform eingereichte Exemplar stimmt mit den gebundenen Exemplaren überein.

Tübingen, den 13.05.2018

Ort, Datum

Unterschrift

Cover image: Grotrian diagram for P Iv (bottom) and v (top).
Contents

1 Introduction 1

2 Spectral analysis 1
   2.1 Basics of atomic physics ........................................... 1
   2.2 Transitions and line formation ................................... 3
   2.3 Opacity and emissivity .............................................. 4
   2.4 Line broadening ...................................................... 4

3 Stellar atmospheres 6
   3.1 Assumptions ............................................................ 6
   3.2 Equations .............................................................. 7
   3.3 Tübingen NLTE Model-Atmosphere Package ..................... 9

4 Phosphorus model atom 10
   4.1 Tübingen Model-Atom Database .................................. 10
   4.2 Phosphorus atomic data ............................................. 12
   4.3 Sources of atomic data ............................................. 14

5 Stellar evolution 16
   5.1 Subdwarfs .............................................................. 19

6 Preliminary analysis of CPD-20°1123 19

7 Supplementary software 22
   7.1 nist_level_reader .................................................. 22
   7.2 nist_line_reader ................................................... 22
   7.3 fs_split_merge ...................................................... 23

8 Summary 23

References 24

Lists of Figures and Tables 25

Acknowledgements 26

A TMAD interface 27

B Grotrian diagrams for P I - XV 28

C Ionization at different effective temperatures 33

D Source codes 36
1 Introduction

The atmosphere is the only part of a star that can be observed by its emergent electromagnetic radiation (Werner 2013). Therefore, the understanding of its physics is essential to learn about a star’s interior structure and processes, and its evolution. This makes stellar atmospheres a topic of high interest for astrophysics and they are even considered as laboratories for atomic physics, hydrodynamics, plasma physics, thermodynamics, etc. (e.g., Rauch et al. 2017).

The study of the interaction between matter and electromagnetic radiation is called spectral analysis or spectroscopy. State-of-the-art spectral analysis uses sophisticated techniques, which will be described in Chapter 3, to calculate model atmospheres and compare their spectra with observations. Hence, high-resolution and high signal-to-noise (S/N) spectra, and reliable atomic data are required. The latter is only available for a limited number of chemical elements, but is necessary for the calculation of realistic models. The scope of this work was to critically review available atomic data for phosphorus (P), and to compile a revised model atom for the Tübingen Model-Atom Database (TMAD). This may be used, e.g., by the Tübingen NLTE Model-Atmosphere Package (TMAP). As an example, we apply the new P model atom to the spectral analysis of the sdB-type star CPD-20°1123 (Chapter 6).

2 Spectral analysis

2.1 Basics of atomic physics

This chapter will give a brief overview of atomic physics and explain the most common expressions used later in this work. A comprehensive introduction to atomic physics can be found in, e.g., Haken & Wolf (2005).

To quantum mechanically describe the state of an atom (it may be neutral, one or many times ionized), i.e., the state of the nucleus and its electrons, we use quantum numbers. Those refer to properties of the atom respectively its wave function $\Psi$. For our purposes, we are interested in the total energy that is associated with different electron configurations of an atom. To characterize a configuration of a single valence electron, we utilize the following quantum numbers:

principal quantum number $n$

$n$ was first used in Bohr’s theory (e.g., Demtröder 2010, p. 111) and referred to the orbit an electron occupies around a nucleus. A better mathematical description for the probability to find an electron at a given position is given by atomic orbitals which still make use of a principal quantum number. $n$ describes the electron shell and is always a positive (nonzero) integer. As $n$ increases, the electron has a higher energy and is less tightly bound to the nucleus.

orbital quantum number $l$

$l$ describes the subshell that an electron occupies and is also an integer. For a given $n$, the possible values of $l$ range from 0 to $n - 1$. The orbital angular momentum of the electron is given by

$$|\hat{l}| = \sqrt{l(l + 1)} \, \hbar, \quad \hbar = \frac{h}{2\pi},$$

(1)
with the Planck constant $h$. For historical reasons, different numbers of $l$ are associated with different letters: $0 = s(harp)$, $1 = p(rincipal)$, $2 = d(iffuse)$, $3 = f(undamental)$. For $l \geq 4$ the series continues alphabetically, i.e., $g$, $h$, $i$, etc.

**spin quantum number $s$**

The electron spin can be $\pm \frac{1}{2}h$ and is the intrinsic angular momentum of the electron.

**total angular momentum quantum number $j$**

$j$ is the absolute value of the vector addition $\vec{J} = \vec{l} + \vec{s}$. An electron in state $\Psi(n, l, s, j) \leftrightarrow \langle 2, 1, +1/2, 1/2 \rangle$ would be denoted as $2p_{1/2}$.

For a multi-(valence)-electron atom, the energy level is dependent on the total electron configuration and the coupling of angular momentum vectors, which is represented in a so-called term symbol. The commonly used term symbols assume LS coupling which is mainly determined by electrostatic interactions. The angular momentum vectors and spin vectors add up to according total vectors. A new total angular momentum vector is formed with the new vectors (Figure 1). Quantum numbers for multi-electron systems now refer to the absolute of those vectors,

$$\vec{L} = \sum_i \vec{l}_i, \quad \vec{S} = \sum_i \vec{s}_i$$

$$\vec{J} = \vec{L} + \vec{S}$$

with

$$|L - S| \leq J \leq L + S$$

For $L \geq S$, the term splits in $r = (2S + 1)$ levels with different values of $J$. This quantum number is called multiplicity and denoted as singlet ($S = 0$), doublet ($S = 1/2$), triplet
In presence of an outer field (e.g., magnetic field) the total angular momentum vector \( J \) aligns so that the component in the direction of the field \( M_J \) (its magnetic quantum number) can take up to \( 2J + 1 \) different possibilities. Without any outer field, the level is \( g_J = 2J + 1 \) times degenerate, which is called the statistical weight. The so-called parity for a configuration of several electrons is either even or odd, depending whether the sum \( \sum l_i \) is even or odd. An odd level is marked with a " + " after its term symbol where an even level is in general not denoted (partly with an " - "). The parity is important for the selection rules of different types of transitions, e.g., an allowed electronic dipole transition can only occur from an odd to an even term or vice versa. A transition between two terms can result in a group of distinct lines which in sum is called a multiplet, e.g., a \( ^2P - ^2D^o \) doublet multiplet consists of three distinct lines \( (J: 1/2 \leftrightarrow 3/2, \ 3/2 \leftrightarrow 3/2, \ 3/2 \leftrightarrow 5/2) \) according to selection rules.

### 2.2 Transitions and line formation

There are several processes of photons interacting with matter that affect the absorption and emission in a star’s photosphere.

#### Radiative transitions

A photon with frequency \( \nu \) can be absorbed by an atom/ion (more precisely by one of its shell electrons) in a state with energy \( E_i \) and excite it to a higher state with energy \( E_j \) if \( h\nu = \Delta E = |E_i - E_j| \), i.e., the photon energy equals the exact energy difference of those two states. The reverse process, i.e., the creation of a photon with energy \( \Delta E \), where a shell electron falls into a lower state, is called emission. It can happen spontaneously, due to the finite lifetime of a state, or as stimulated emission, where a photon of frequency \( \nu \) induces the atom to emit a photon of the same energy/frequency. Transitions are usually categorized as follows (the keywords bound and free refer to the electron being part of the atom/ion before and after the transition):

- **Bound-bound (RBB):** \( E_i, E_j < E_{\text{ion}} \), i.e., both energies are lower than the ionization limit. Radiative bound-bound transitions are responsible for characteristic absorption and emission lines.

- **Bound-free (RBF):** \( E_i < E_{\text{ion}}, E_j \geq E_{\text{ion}} \), i.e., the atom is ionized and an electron of energy \( m_e c^2/2 = h\nu - (E_{\text{ion}} - E_i) \) leaves the atom. The reverse process is called recombination and is responsible for ionization edges. This does not result in discrete lines but in a continuous absorption/emission for \( \nu \geq \nu_{\text{ion}} \).

- **Free-free (RFF):** \( E_i, E_j > E_{\text{ion}} \), i.e., a photon of energy \( h\nu = |E_i - E_j| \) is absorbed or emitted when a free electron is accelerated or decelerated in the coulomb field of an atom or ion. The latter is called bremsstrahlung and also has a continuous spectrum.

#### Collisional transitions

Transitions can also occur due to particle collisions, i.e., collisional bound-bound (CBB) and collisional bound-free (CBF) transitions. For the hot plasma of a typical stellar atmosphere, collisions between electrons are dominant.

#### Scattering

A photon being elastically scattered by a free charged particle (e.g., an electron), called Thomson scattering, is also contributing to the spectrum. As long as the
photon energy is much lower than the kinetic energy of the particle, the particle energy and photon frequency do not change as a result of the scattering (only the direction does).

2.3 Opacity and emissivity

The total opacity $\kappa_\nu$ and emissivity $\eta_\nu$ are made up of the emission- and absorption coefficients of all the before mentioned processes. The ratio of opacity and emissivity is called the source function

$$S_\nu = \frac{\eta_\nu}{\kappa_\nu} = B_\nu(T)$$

which is equal to the Planck function $B_\nu(T)$ in case of thermodynamical equilibrium (Kirchhoff’s law). The Planck function

$$B_\nu(T) = \frac{2 h \nu^3}{c^2} \frac{1}{e^{h\nu/k_B T} - 1},$$

with the Boltzmann constant $k_B$, and temperature $T$, describes the spectral energy distribution of a black body in thermal equilibrium at temperature $T$.

According to Einstein, we can describe RBB transition probabilities with the so-called Einstein coefficients $A_{ji}$ (for spontaneous emission), $B_{ji}$ (for stimulated emission), and $B_{ij}$ (for absorption). Einstein coefficients are atomic properties and are related as follows

$$A_{ji} = \frac{2h\nu^3}{c^2} B_{ji}, \quad g_j B_{ij} = g_i B_{ji}.$$

In spectroscopy, the commonly used quantity is the dimensionless oscillator strength (here referred as $f$-value)

$$f_{ij} = \frac{4\pi \varepsilon_0 m_e c^3}{8\pi^2 e^2 \nu^2} g_j A_{ji},$$

with the statistical weights $g_i$, $g_j$, the electric constant $\varepsilon_0$, the electron mass $m_e$, and the electron charge $e$.

2.4 Line broadening

The shape of a line profile is not infinitely sharp because several mechanisms lead to line broadening.

Natural line broadening

Due to quantum mechanical uncertainty $\Delta E \Delta t \geq \hbar$ and the finite lifetime $\tau$ of a level, photons emitted in a transition from this level will have a range of possible frequencies

$$\Delta \nu = \Delta\frac{E}{\hbar} = \frac{1}{2\pi \tau}.$$

This causes the line to take a so-called Lorentzian profile

$$L(\Delta \nu) = \frac{\gamma}{(2\pi \Delta \nu)^2 + (\gamma/2)^2}, \quad \Delta \nu = \nu - \nu_0$$
with the rest frequency $\nu_0$ and the damping constant $\gamma = \frac{1}{\tau}$.

**Doppler broadening**

Line broadening due to the Doppler effect is caused by thermal motion of electrons. Photons emitted with a frequency $\nu_0$ are observed at a shifted frequency

$$\nu = \sqrt{\frac{1 + \beta}{1 - \beta}} \nu_0, \quad \beta = \frac{v}{c} \quad ,$$

depending on the actual velocity $v$ of the emitting atom. The Maxwellian distribution of velocities leads to a so-called Gaussian profile

$$D(\Delta \nu) = \frac{1}{\sqrt{\pi \Delta \nu_D}} e^{- (\Delta \nu / \Delta \nu_D)^2} \quad ,$$

with the Doppler width

$$\Delta \nu_D = \frac{\Delta \nu_0}{c} v_0, \quad v_0 = \sqrt{\frac{2k_B T}{m}} \quad .$$

$v_0$ is the most likely velocity, $\nu_0$ the center of the line, and $m$ the mass of the involved atom. Microturbulences can have a similar effect.

**Pressure (collisional broadening)**

Collisions reduce the effective lifetime of a state, thus leading to broader lines. High pressure, e.g., due to the high gravity of a compact stellar object, causes more collisions. As a result, pressure broadening also causes a Lorentzian line profile.

**Stark effect**

In the presence of an electric field, spectral lines shift and split. This phenomenon is called the Stark effect and can be calculated with quantum mechanical perturbation theory. At high density, electrons are influenced by the electric fields of other charged particles.

**Rotational broadening**

The stellar rotation causes one side to move towards the observer, and the other one to move away. This also causes Doppler broadening in addition to the Doppler effect caused by thermal motion.

All the effects combined result in a so-called Voigt profile (Figure 2), which is the convolution of a Gaussian and Lorentzian profile

$$V(\Delta \nu) = D(\Delta \nu) \otimes L(\Delta \nu) = \int_{-\infty}^{+\infty} D(\Delta \nu') L(\Delta \nu - \Delta \nu') d(\Delta \nu') \quad .$$

The resulting, normalized function is conventionally introduced as

$$V(x) = \frac{\alpha_L}{\pi^{3/2}} \int_{-\infty}^{+\infty} \frac{e^{-t^2}}{(x - t)^2 + \left(\frac{\alpha_L}{\alpha_D}\right)^2} dt \quad ,$$

with the half-width for pressure broadening $\alpha_L$, the half-width for Doppler broadening $\alpha_D$, and the weighted distance to the line center $x = \frac{\nu - \nu_0}{\alpha_D}$. 


3 Stellar atmospheres

This chapter will give a brief overview of the basic concept to compute a stellar atmosphere. The pretense is, that it is both realistic enough for scientific purposes and computation time is reasonable. An extensive review of the theory of stellar atmospheres is given in Hubeny & Mihalas (2014).

3.1 Assumptions

For a “normal” star like the Sun, the depth of its photosphere and chromosphere is about 3000 km which is more than two dex thinner than its radius (≈ 700 000 km). The same is true for a white-dwarf star (100 km/6000 km ≪ 1). Thus, we can assume plane-parallel geometry is a sufficient approximation that makes a model atmosphere less computational exhausting than, e.g., spherical-symmetrical geometry.

The characteristics of a stellar atmosphere depend only on the following parameters: the effective temperature \( T_{\text{eff}} \) (in K), the surface gravity \( g \) (cm/s\(^2\)), and the chemical composition, i.e., the abundances of all chemical species.

A stellar atmosphere is an open system (it radiates into space) and therefore cannot be in thermodynamical equilibrium and described by a single temperature. However, we assume that for a small volume of the atmosphere, there is a so-called “Local Thermodynamic Equilibrium” (LTE). Thus, the atomic/ionic occupation numbers depend on the Saha-Boltzmann equations

\[
\frac{n_j}{n_i} = \frac{g_j}{g_i} e^{-\frac{(E_j - E_i)}{k_B T}} \tag{15}
\]

\[
\frac{n_k}{n_i} = \frac{g_k}{g_i} \frac{2}{n_e} \left( \frac{2\pi m_e k_B T}{\hbar^2} \right)^{3/2} e^{-\frac{(E_k - E_i)}{k_B T}} \tag{16}
\]

with the occupation numbers \( n_i \) and \( n_j \) for levels of the same ionization stage, occupation number \( n_k \) of a different ionization stage, and statistical weights \( g_i, g_j, \) and \( g_k \). The LTE assumption holds for optically thick atmospheres (low temperature, high density, i.e., spectral type B or later) where collisional rates dominate radiative rates, and the mean free path of a photon is small. For more optically thin atmospheres, where radiative processes dominate, the Saha-Boltzmann equations have to be replaced by detailed non-LTE (NLTE) rate equations. NLTE calculations are necessary for atmospheres with high temperatures and low densities.

We assume chemical homogeneity, i.e., that elements are distributed evenly within the atmosphere, and convection and heat-conduction effects are negligible.

### 3.2 Equations

The atmosphere is mathematically described with the following equations. More detailed explanations to each point can be found in Werner et al. (2003).

#### Radiation transfer

Ignoring the two above mentioned effects, we assume that energy transport in a stellar atmosphere is done only by photons. It depends on the emissivity and opacity (emission and absorption) of the matter, e.g., the processes described in Chapter 2.2. To describe the radiation field, one commonly uses the radiation intensity, or specific intensity \( I_\nu \), which is defined as the rate of energy flowing at a given point, per unit area, per unit time, per unit frequency interval, and per unit solid angle. In plane-parallel geometry, the azimuthal angle dependency vanishes and we get

\[
I_\nu = \frac{dE}{\mu dA dt d\nu d\Omega} \left[ \frac{\text{erg}}{\text{cm}^2 \text{s} \text{Hz} \text{sr}} \right] \quad (17)
\]

with

\[
\mu = \cos(\vartheta), \quad \mu \in [0, 1] \quad ,
\]

where \( \vartheta \) is the angle between the direction of the radiation beam and the normal vector. We use the mean intensity

\[
J_\nu = \frac{1}{2} \int_{-1}^{+1} I_\nu d\mu, \quad d\mu = -\sin(\vartheta) d\vartheta \quad ,
\]

(19)

to describe quantities that are photon dominated but independent of the angular distribution of the radiation (e.g., photoionization and photo excitation rates). Instead of the geometrical depth \( z \), one uses the dimensionless optical depth

\[
\tau_\nu = -\int_{0}^{z} \kappa_\nu dz' \quad ,
\]

(20)

so that the radiation-transfer equation

\[
\pm \mu \frac{\partial I_\nu(\pm\mu)}{\partial \tau_\nu} = S_\nu - I_\nu(\pm\mu) \quad (21)
\]

describes the energy transport of photons at a certain depth point and angle of incidence \( \vartheta \).
Hydrostatic equilibrium

We assume that all matter interacting with photons is at rest, which occurs when gravity and a pressure gradient force are in balance. The hydrostatic equation

\[
\frac{dP}{dm} = g ,
\]

where

\[
m = \int_a^b \rho(z) \, dz, \quad [m] = \frac{g}{\text{cm}^2}
\]

is the so-called column-mass density, measured from the outermost part of the model atmosphere, which depends on the geometrical depth \( z \). The pressure

\[
P = Nk_B T + \frac{4\pi}{c} \int_0^\infty f_\nu J_\nu \, d\nu + \frac{1}{2} \rho v^2_{\text{turb}}
\]

with the the variable Eddington factor \( f_\nu \) and the turbulent velocity \( v_{\text{turb}} \) consists of gas, radiation, and turbulent pressures. Solving the hydrostatic-equilibrium equation determines the total particle density \( N \).

Radiative equilibrium

Radiative equilibrium means that the emitted and absorbed energy per unit time and volume element is equal, i.e., they cancel out, and it is

\[
\int_0^\infty \kappa_\nu (S_\nu - J_\nu) \, d\nu = 0 .
\]

Solving the radiative-equilibrium equation derives the temperature \( T \).

Statistical equilibrium

Each chemical element is considered with different ionization stages and different level energies belonging to that ionization stage. The change of occupation numbers \( n_i \) of level \( i \) (interacting with another level \( j \)) is described by the rate equation. In statistical equilibrium,

\[
\sum_{i \neq j} n_i P_{ij} - \sum_{j \neq i} n_j P_{ji} = 0
\]

the upward and downward rates cancel out. The rate coefficients

\[
P_{ij} = R_{ij} + C_{ij}
\]

include radiative and collisional components.

Particle conservation

The total particle density \( N \) equals the electron density and the population of all levels, it is
\[ N = n_e + \sum_{k=1}^{\text{NATOM}} \sum_{l=1}^{\text{NION}(k)} \left( \sum_{i=1}^{\text{N-NLTE}(l)} n_{kli} + \sum_{i=1}^{\text{N-LTE}(l)} n_{kli}^* \right). \]  

(28)

NATOM is the number of atoms, NION(k) the number of ions of element k, N-NLTE(l) the number of NLTE levels for the ion l, N-LTE(l) the number of LTE levels for the ion l, and \( n_{kli} \) and \( n_{kli}^* \) the population numbers of level i of ion l and atom k. Solving the particle-conservation equation yields the electron density \( n_e \).

### 3.3 Tübingen NLTE Model-Atmosphere Package

To solve the stellar-atmosphere problem, the previously shown equations have to be solved simultaneously for each depth point \( d \) in a grid of \( ND \) (typically 90) depth points describing the complete atmosphere. These equations are non-linear and highly coupled, i.e., no single equation determines uniquely a single quantity, but instead depend on each other.

The straight-forward approach would be to completely linearize the equations with respect to all variables, which is described by [Auer & Mihalas (1969)](http://astro.uni-tuebingen.de/~TMAP/UserGuide/UserGuide.pdf). TMAP uses another approach [Werner et al. (2003, 2012)](http://astro.uni-tuebingen.de/~TMAP/UserGuide/UserGuide.pdf), which proved to be much faster. It eliminates the explicit appearance of mean intensity \( J_d \) from the set of equations by expressing it with yet-to-be determined occupation numbers and temperature. This is done with an iteration procedure called Accelerated Lambda-Iteration Method [Werner & Husfeld (1985; Werner (1986)], that can be written as

\[ J^n = \Lambda^* S^n + (\Lambda - \Lambda^*) S^{n-1}, \]

(29)

where \( \Lambda^* \) is an approximate lambda operator and \( n \) the iteration step. The second part of the sum is a correction term that includes the exact lambda operator \( \Lambda \), which guarantees convergence to the exact solution \( J = \Lambda S \) once \( S^n = S^{n-1} \). The resulting set of equations is then solved by a so-called Kantorovich iteration [Kantorovich (1949)](http://astro.uni-tuebingen.de/~TMAP/UserGuide/UserGuide.pdf) implemented as shown in [Hubeny & Lanz (1992)](http://astro.uni-tuebingen.de/~TMAP/UserGuide/UserGuide.pdf).

The complete software package consists of five programs and several more utility programs. A detailed user’s guide is available on the homepage of the astronomical institute Tübingen. Here, a brief overview of the software used in this work is given.

**ATOMS2** checks and prepares the atomic data files for further use. It can detect inconsistencies and errors of the atomic data.

**SETF2** creates a frequency grid which is used by the following three programs. It requires the ATOMS2 output data and an effective temperature \( T_{\text{eff}} \) as input parameters.

**LTE2** computes a model atmosphere in so-called gray approximation, that is, opacity and emissivity do not depend on the frequency, hence there are no absorption lines. A model is computed for given elemental abundances, \( T_{\text{eff}} \), and \( \log g \), and is then used as start model for PRO2. LTE2 requires the ATOMS2 atomic data output and the frequency grid computed by SETF2.

Table 1: Statistics of P ions.

<table>
<thead>
<tr>
<th>P ion</th>
<th>NLTE levels</th>
<th>LTE levels</th>
<th>line transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>51</td>
<td>33</td>
<td>20</td>
</tr>
<tr>
<td>II</td>
<td>36</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>III</td>
<td>33</td>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td>IV</td>
<td>36</td>
<td>71</td>
<td>28</td>
</tr>
<tr>
<td>V</td>
<td>18</td>
<td>19</td>
<td>49</td>
</tr>
<tr>
<td>VI</td>
<td>15</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>VII</td>
<td>17</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>VIII</td>
<td>10</td>
<td>23</td>
<td>3</td>
</tr>
<tr>
<td>IX</td>
<td>8</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>X</td>
<td>11</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>XI</td>
<td>21</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>XII</td>
<td>19</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>XIII</td>
<td>11</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>XIV</td>
<td>41</td>
<td>16</td>
<td>234</td>
</tr>
<tr>
<td>XV</td>
<td>15</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>XVI</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Total</td>
<td>343</td>
<td>239</td>
<td>434</td>
</tr>
</tbody>
</table>

PRO2 performs the NLTE model calculations with some approximations, e.g., fine-structure splitting of the atomic level and tabulated data for Stark line broadening are not considered. It uses the output data of the previous three programs as input data. The output model includes opacities of all elements considered in the calculations.

LINE1_PROF uses the PRO2 output model, atomic data including fine-structure splitting from ATOMS2, and frequency grid to compute a complete synthetic spectrum that can be compared to observational data.

4 Phosphorus model atom

4.1 Tübingen Model-Atom Database

The Tübingen Model-Atom Database (TMAD\textsuperscript{4}, Rauch & Deetjen \textsuperscript{2003}) was created on the framework of the Tübingen project with in the German Astrophysical Virtual Observatory (GAVO\textsuperscript{5}) initiative. It provides model atoms with level energies and oscillator strengths in TMAP format. In general, the model atoms can be used by any model-atmosphere code using a suitable adapter. TMAD hosts atomic data for over 30 species and can be accessed publicly \textsuperscript{[Figure 12].}

An atom starts with the keyword ATOM in a single line, then followed by its chemical element code, the charge of the first stage, and its atomic weight in AMU separated with spaces. A special keyword starts a section with specific data (e.g., L for NLTE levels, LTE for LTE levels, RBB for radiative bound-bound transitions, etc.) and a zero character ends that section. A complete atom consist of several ionization stages (can be as many

\textsuperscript{4}http://astro.uni-tuebingen.de/~TMAD
\textsuperscript{5}http://g-vo.org
Figure 3: Grotrian diagram of P\textsc{v}.

Table 2: Parameters for our TMAP test calculations.

<table>
<thead>
<tr>
<th>( T_{\text{eff}} / \text{K} )</th>
<th>( \log(g / \text{cm/s}^2) )</th>
<th>considered P ions</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000</td>
<td>7</td>
<td>I - IV</td>
</tr>
<tr>
<td>30000</td>
<td>7</td>
<td>II - VI</td>
</tr>
<tr>
<td>40000</td>
<td>7</td>
<td>II - VI</td>
</tr>
<tr>
<td>50000</td>
<td>7</td>
<td>III - VII</td>
</tr>
<tr>
<td>60000</td>
<td>7</td>
<td>III - VII</td>
</tr>
<tr>
<td>70000</td>
<td>7</td>
<td>III - VII</td>
</tr>
<tr>
<td>100000</td>
<td>7</td>
<td>IV - VIII</td>
</tr>
<tr>
<td>200000</td>
<td>7</td>
<td>VI - X</td>
</tr>
<tr>
<td>300000</td>
<td>7</td>
<td>X - XIII</td>
</tr>
<tr>
<td>400000</td>
<td>8</td>
<td>X - XIV</td>
</tr>
<tr>
<td>500000</td>
<td>8</td>
<td>XI - XV</td>
</tr>
</tbody>
</table>

as number of electrons of that element) where each stage has its own NLTE, LTE, RBB, etc. section. A dot at the beginning of a line marks this line as a comment.

Level energies are found in the NLTE and LTE level sections, depending on whether they should be considered for NLTE or not. For historical reasons, the format utilizes a ten character string (in FORTRAN, a so-called A10 string) to describe a level including electron configuration and term symbol. Due to the limited space, the chemical code and ionization stage are coded as the first three characters (e.g., P11 for ten times ionized P), the next three characters code the electron configuration where usually the last part of the electron configuration is used to identify the level (e.g., 2P for 2s\(^2\)2p\(^2\) or 2P\(^2\) for 2s2p\(^2\)), and characters eight to ten represent the term symbol without the total angular momentum where odd levels are marked with a capital O and even levels have a space at the end (e.g., 2P0 for 2P\(^0\)). The seventh character is used for the total angular momentum in case of fine-structure splitting, or, if the latter is omitted, mark a one or more times excited level by special character like “‘”, “''”, etc. (in the latter case, this character can sometimes be found at the sixth place if the electron configuration is short enough). A complete level consists of two subsequent A10 strings for the level and its parent level (for BF transitions) in the next ionization stage, the energy to the ionization limit in Hz, and the statistical weight of that level. If there is no parent level (either the highest ionization stage of the element or the highest ionization stage in a restricted
model atom), the second A10 string becomes **NONE______** (i.e., “NONE” and six spaces) and the statistical weight 1.0. Everything in this line after character 80 is automatically considered a comment. A typical level could look like this

```
P112P 2POP12S2 1S 1.157343987430E+17 6
```

A transition (found in the according RBB, RBF, etc. sections) also consists of two subsequent A10 strings of the involved levels and after that the formula number for the calculation of the cross-section, the number of arguments as an integer, and the arguments itself, if there are any. For the different formula numbers and allowed number of arguments, please see the appendix A of the TMAP guide. An example transition is

```
P112P 2POP113D 2D 3 3 6.5000E-01 1.2194E+12 0.5306
```

### 4.2 Phosphorus atomic data

The newly compiled P model atom consists of all 15 ionization stages and a total of 343 NLTE and 239 LTE levels with 434 line transitions (Table 1).

Grotrian diagrams were created for all ionization stages. Each diagram (e.g., Figure 3) shows levels as thick horizontal lines and line transitions as thin connecting lines between the levels. The energy of each level (in cm\(^{-1}\)) is indicated by the position on the ordinate, electron configuration is shown in red, and the corresponding term symbol is displayed by the placement on abscissa. The ionization limit is denoted with a dashed line. Grotrian diagrams for all ionization stages can be found in Appendix B.

As seen in Table 1, sufficient level energies were available for all ionization stages. However, for several ions, the number of line transitions with known \(f\)-values was sparse. For only eight out of 15 model ions, we could find more than ten unique line transitions in databases and literature. The number of allowed electronic dipole transitions according to selection rules is much higher (indicated in Figure 5). NLTE model-atmospheres could therefore greatly benefit from extensive theoretical calculations or laboratory measurements of \(f\)-values.
Table 3: Most prominent P lines at different $T_{\text{eff}}$ and $\log g$ \cite{Table 2}.

<table>
<thead>
<tr>
<th>$T_{\text{eff}}$ / K</th>
<th>$\lambda^0 / \AA$</th>
<th>Ion</th>
<th>Configuration</th>
<th>$\lambda^0 / \AA$</th>
<th>Ion</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000</td>
<td>1301.87</td>
<td>II</td>
<td>$3p^2 3P_0 - 3p^4 3P^o_0$</td>
<td>1300.49</td>
<td>II</td>
<td>$3p^2 3P_1 - 3p^4 3P^o_0$</td>
</tr>
<tr>
<td></td>
<td>1304.68</td>
<td>II</td>
<td>$3p^2 3P_1 - 3p^4 3P^o_0$</td>
<td>1305.50</td>
<td>II</td>
<td>$3p^2 3P_1 - 3p^4 3P^o_0$</td>
</tr>
<tr>
<td></td>
<td>1309.87</td>
<td>II</td>
<td>$3p^2 3P_2 - 3p^4 3P^o_1$</td>
<td>1310.70</td>
<td>II</td>
<td>$3p^2 3P_2 - 3p^4 3P^o_2$</td>
</tr>
<tr>
<td></td>
<td>1334.81</td>
<td>III</td>
<td>$2p^2 3P_{1/2} - 3p^2 2P_{3/2}$</td>
<td>1344.33</td>
<td>III</td>
<td>$2p^2 3P_{3/2} - 3p^2 2D_{5/2}$</td>
</tr>
<tr>
<td></td>
<td>1344.85</td>
<td>III</td>
<td>$2p^2 3P_{3/2} - 3p^2 2D_{3/2}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30000</td>
<td>1334.81</td>
<td>III</td>
<td>$2p^2 3P_{1/2} - 3p^2 2D_{3/2}$</td>
<td>1334.81</td>
<td>III</td>
<td>$2p^2 3P_{3/2} - 3p^2 2D_{5/2}$</td>
</tr>
<tr>
<td></td>
<td>1118.55</td>
<td>IV</td>
<td>$2p^1 3P^o_1 - 3p^2 1S_0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1117.98</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_0$</td>
<td>1128.01</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{1/2}$</td>
</tr>
<tr>
<td>40000</td>
<td>1028.09</td>
<td>IV</td>
<td>$3p^2 3P_0 - 3p^4 3P_1$</td>
<td>1030.51</td>
<td>IV</td>
<td>$3p^2 3P_1 - 3p^4 3P_1$</td>
</tr>
<tr>
<td></td>
<td>1030.51</td>
<td>IV</td>
<td>$3p^2 3P_2 - 3p^4 3P_2$</td>
<td>1033.11</td>
<td>IV</td>
<td>$3p^2 3P_1 - 3p^4 3P_0$</td>
</tr>
<tr>
<td></td>
<td>1035.52</td>
<td>IV</td>
<td>$3p^2 3P_2 - 3p^4 3P_2$</td>
<td>1118.55</td>
<td>IV</td>
<td>$3p^2 3P_1 - 3p^4 3P_0$</td>
</tr>
<tr>
<td></td>
<td>1117.98</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_0$</td>
<td>1128.01</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{1/2}$</td>
</tr>
<tr>
<td>50000</td>
<td>1028.09</td>
<td>IV</td>
<td>$3p^2 3P_0 - 3p^4 3P_1$</td>
<td>1030.51</td>
<td>IV</td>
<td>$3p^2 3P_1 - 3p^4 3P_1$</td>
</tr>
<tr>
<td></td>
<td>1030.51</td>
<td>IV</td>
<td>$3p^2 3P_2 - 3p^4 3P_2$</td>
<td>1033.11</td>
<td>IV</td>
<td>$3p^2 3P_1 - 3p^4 3P_0$</td>
</tr>
<tr>
<td></td>
<td>1035.52</td>
<td>IV</td>
<td>$3p^2 3P_2 - 3p^4 3P_2$</td>
<td>1118.55</td>
<td>IV</td>
<td>$3p^2 3P_1 - 3p^4 3P_0$</td>
</tr>
<tr>
<td></td>
<td>328.47</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{3/2}$</td>
<td>328.77</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{1/2}$</td>
</tr>
<tr>
<td>60000</td>
<td>1117.98</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{3/2}$</td>
<td>1128.01</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{1/2}$</td>
</tr>
<tr>
<td></td>
<td>74.95</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^1 1P^o_1$</td>
<td>75.64</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^3 3D^o_1$</td>
</tr>
<tr>
<td></td>
<td>76.53</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^3 3P^o_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70000</td>
<td>328.47</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{3/2}$</td>
<td>328.77</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{1/2}$</td>
</tr>
<tr>
<td></td>
<td>1117.98</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{3/2}$</td>
<td>1128.01</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{1/2}$</td>
</tr>
<tr>
<td></td>
<td>74.95</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^1 1P^o_1$</td>
<td>75.64</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^3 3D^o_1$</td>
</tr>
<tr>
<td></td>
<td>76.53</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^3 3P^o_1$</td>
<td>90.65</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3s^1 1P^o_1$</td>
</tr>
<tr>
<td></td>
<td>91.47</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3s^3 3P^o_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>1117.98</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{3/2}$</td>
<td>1128.01</td>
<td>V</td>
<td>$3s^2 2S_{1/2} - 3p^2 2P^o_{1/2}$</td>
</tr>
<tr>
<td></td>
<td>74.95</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^1 1P^o_1$</td>
<td>75.64</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^3 3D^o_1$</td>
</tr>
<tr>
<td></td>
<td>76.53</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3d^3 3P^o_1$</td>
<td>90.65</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3s^1 1P^o_1$</td>
</tr>
<tr>
<td></td>
<td>91.47</td>
<td>VI</td>
<td>$2p^6 1S_0 - 3s^3 3P^o_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>219.91</td>
<td>VII</td>
<td>$2p^5 2P^o_{3/2} - 2p^6 2S_{1/2}$</td>
<td>223.49</td>
<td>VII</td>
<td>$2p^5 2P^o_{1/2} - 2p^6 2S_{1/2}$</td>
</tr>
<tr>
<td>200000</td>
<td>219.91</td>
<td>VII</td>
<td>$2p^5 2P^o_{3/2} - 2p^6 2S_{1/2}$</td>
<td>223.49</td>
<td>VII</td>
<td>$2p^5 2P^o_{1/2} - 2p^6 2S_{1/2}$</td>
</tr>
<tr>
<td></td>
<td>196.76</td>
<td>VIII</td>
<td>$2p^4 1D_2 - 2p^5 1P^o_1$</td>
<td>222.36</td>
<td>VIII</td>
<td>$2p^4 1S_0 - 2p^5 1P^o_1$</td>
</tr>
<tr>
<td></td>
<td>244.55</td>
<td>VIII</td>
<td>$2p^4 3P_2 - 2p^5 3P^o_1$</td>
<td>246.32</td>
<td>VIII</td>
<td>$2p^4 3P_1 - 2p^5 3P^o_2$</td>
</tr>
<tr>
<td></td>
<td>247.65</td>
<td>VIII</td>
<td>$2p^4 3P_2 - 2p^5 3P^o_2$</td>
<td>248.05</td>
<td>VIII</td>
<td>$2p^4 3P_1 - 2p^5 3P^o_1$</td>
</tr>
<tr>
<td></td>
<td>249.32</td>
<td>VIII</td>
<td>$2p^4 3P_0 - 2p^5 3P^o_2$</td>
<td>251.23</td>
<td>VIII</td>
<td>$2p^4 3P_1 - 2p^5 3P^o_2$</td>
</tr>
</tbody>
</table>

Notes. (a) vacuum wavelength.

Test calculations were performed to determine the strongest P lines for different effective temperatures $T_{\text{eff}}$. Each model did consist of hydrogen and P at solar abundance ($n(P)/n(H) = 2.570 \times 10^{-7}$, \cite{Asplund et al 2009}). \cite{Table 2} shows the different parameters of $T_{\text{eff}}$ and $\log g$ that were used.

Ionization-fraction plots show the dominant ionization stages for each calculation. The
plot abscissa shows column-mass density, which indicates the depth in the atmosphere, the ordinate shows the ionization fractions of selected ions. An example is shown in Figure 4, all plots can be found in Appendix C.

To determine the most prominent lines, additional synthetic spectra with reduced P abundance (\(n(P)/n(H) = 1.0 \times 10^{-20}\)) were calculated and subtracted. The most prominent lines at each temperature are listed in Table 3.

### 4.3 Sources of atomic data

The main source of atomic data (level energies and transition probabilities) was the American National Institute of Standards and Technology (NIST) Atomic Spectra Database. Transitions for IV, IX, and XII were taken from the CHIANTI database, transitions for XIII - XV from the atomic line list of the University of Kentucky.
Figure 5: Grotrian diagrams P II - IV. Connecting black lines represent transitions with known $f$-values. Additional possible line transitions according to selection rules (including intercombination lines) with unknown $f$-values are shown in gray.
5 Stellar evolution

This chapter gives a brief overview of stellar evolution. More detailed explanations can be found in, e.g., Unsöld & Baschek (2005), Weigert et al. (2011), or Kippenhahn et al. (2012).

Stars are often categorized by their spectral class and their position in the Hertzsprung-Russell diagram (HRD, Figure 6). It shows the effective temperature $T_{\text{eff}}$ on the abscissa, and the absolute luminosity $L$ on the ordinate. The Margan-Keenan (MK) spectral classification system uses the letters O, B, A, F, G, K, and M, where O-type stars are the hottest and M-type stars are the coolest. With increasing $T_{\text{eff}}$, the maximum in their

---

Figure 6: Hertzsprung-Russell diagram showing positions of the main sequence, giants, supergiants, and white dwarfs. Image taken from ESO, modified.

[https://www.eso.org/public/images/eso0728c](https://www.eso.org/public/images/eso0728c)
Figure 7: Evolutionary tracks off the main sequence for stars with $M_{\text{ini}} = 1, 5, \text{ and} 10 \, M\odot$. Image taken from the Australia Telescope National Facility.

spectral energy distribution is located at lower wavelengths/higher frequency. Stars of different classes show different characteristic absorption lines, a fact on which the MK classification was historically based on, namely, the strength of hydrogen (H) lines.

Stars start their life in the HRD on the zero-age main sequence (ZAMS) but with different starting positions and evolutionary tracks, depending on their initial mass $M$. The Stefan-Boltzmann law shows that the absolute luminosity of a star is depending on both, its radius $R$ and $T_{\text{eff}}$,

$$L = 4\pi\sigma R^2 T_{\text{eff}}^4,$$

with the Stefan-Boltzmann constant $\sigma$. Since more massive stars are bigger, have a higher fusion rate and thus a higher $T_{\text{eff}}$, they are much brighter than their less massive counterparts. Hence, more massive stars are positioned more on the top/left part of the MS and less massive ones on the bottom/right. The empirical relationship between the luminosity and the mass of MS stars is given by

$$\log \frac{L}{L\odot} = 3.8 \log \frac{M}{M\odot} + 0.08.$$

As mentioned above, stars start their core H burning on the ZAMS, and from there on move to the top/right direction as they age. If a star has an initial mass of $0.8 \, M\odot \leq M_{\text{ini}} \leq 8 \, M\odot$, it travels a very peculiar path on the HRD during its lifetime [Figure 7]. As an example, we follow the path of a star with $M_{\text{ini}} \leq 2.5 \, M\odot$. After spending its time on

the main sequence and fusing its H supply, a helium (He) core has formed which cannot be fused any more, because the temperature at the center is not high enough. Outside of the core, H-shell burning starts, but the core contracts under its own gravitational pull, and is stabilized only by electron degeneracy pressure. The H-shell burning drives the outer layers of the star further out, increasing the total radius to several $r_\odot$. While the star’s surface increases, the overall energy output does not, and so the star effectively cools down, now being located on the red giant branch.

H-shell burning deposits He on the core, which will heat up to about $8 \times 10^7$ K. At this point, He is ignited at the core which results in the explosive He flash, and the star can lose up to 50 per cent of its mass as a result. With central He burning, the star shrinks, increases in temperature, and moves towards the horizontal branch (HB) in the HRD. He burning in the core terminates when the He is completely processed to carbon (C) and oxygen (O). The star remains with a CO core, while the He burning continues in a concentric shell surrounding the exhausted core. This new double-shell burning is
responsible for an expansion of the star up to several hundred times of its original radius. The star is now a supergiant, located on the asymptotic giant branch (AGB). He shell burning is an unstable process, which happens in a series of thermal pulses, which occur in intervals of about 1000 years. These are responsible for more mass loss and the creation of a planetary nebula around the central star. After the burning of He and H shells ceases, a white dwarf remains, if the mass of the remaining core is less than $\approx 1.4 M_\odot$ (Chandrasekhar limit), otherwise a neutron star is born.

5.1 Subdwarfs

Subdwarfs (sd) are located below the MS in the HRD and consist of stars with lower luminosity than their MS counterparts. They are generally categorized as either cool sds of spectral type G to M, or hot sds of spectral type O or B. Most B-type subdwarfs (sdB) are situated at the blue end of the HB, the so-called Extreme Horizontal Branch (Heber 1986), and thus are (assumed to be) stripped cores of red giant stars which burn He (Heber 2016). Compared to HB stars, their H envelope is too thin to sustain H shell burning (Heber 2016). Their mass is assumed to be roughly half a solar mass. The majority of hot subdwarf stars is He-poor (about 90 per cent, Naslim et al. 2012), whereas the optical spectra of the remaining He-rich are dominated by strong He i (He-sdB) or He ii (He-sdO) lines.

6 Preliminary analysis of CPD-20°1123

CPD-20°1123 is a bright sdB star with a V magnitude of $11.75 \pm 0.07$ (Vennes et al. 2007). Its optical spectrum shows strong He i lines along with the Balmer series (Vennes et al. 2007). A recent analysis showed, that it is a so-called intermediate He-rich sdB star (Naslim et al. 2012), with a He abundance of $n(\text{He}) = 0.2089 \pm 0.003$ (number fraction). It is different from normal sdB stars, which are He poor, and He rich sdB stars, which have more than 80 per cent He. CPD-20°1123 was chosen as an object of interest, because P lines were previously detected by Naslim et al. (2012). Observations were made at the Australian Astronomical Telescope with the University College London Echelle Spectrograph in January 2010, which covered a wavelength range of $\lambda = 3820 - 5230$ Å. A series of six optical spectra with a total exposure time of 9000 s were taken, with in a
Calculations with a simple model including H, He, and P (considering ionization stages P \textsc{iii} - iv) were performed for $T_{\text{eff}}/K \in [24500, 25000, 25500, 26500]$ and $\log g \in [4.9, 5.0, 5.1]$. Starting abundances were adopted from Naslim et al. (2012). The dominant ionization stage in the line-forming region ($-4 \leq \log m \leq 0.5$) is P \textsc{iii} (Figure 9). Prominent P \textsc{iii} lines $\lambda\lambda$ 4059.31, 4222.12, 4246.72 Å were well reproduced (Figure 10). P \textsc{ii} lines were too weak to be unambiguously identified, P \textsc{iv} lines do not lie in the observed spectrum. The model at $T_{\text{eff}} = 25500$ K and $\log g = 4.9$ with an abundance of P = $(1.42 \pm 0.1) \times 10^{-5}$ (mass fraction) was found to be in best agreement to the observation. The best-fitting model, overplotted with the observation, is shown in Figure 11. All synthetic spectra were convolved with a Gaussian with FWHM of 0.09 Å to match the resolving power of the instrument.
Figure 11: Spectrum of CPD-20°1123 along with our best fit model ($T_{\text{eff}} = 25500$ K, \(\log g = 4.9\)). The most prominent lines are marked. The green, dashed line shows the continuum flux level.
7 Supplementary software

Supplementary software was created for the purpose of automating the process of obtaining and converting the atomic data from the NIST database into TMAP format. To retrieve suitable input data from the NIST homepage, format output in the query mask has to be set to ASCII (plain text) and, in case of line transitions, the checkbox for line transitions has to be activated.

The full source code of each program is available in Appendix D.

7.1 nist_level_reader

The program nist_level_reader parses ASCII data taken from the NIST database (level configuration, energy) and outputs different files that can be used to get a TMAD atom file. The raw data looks like this

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Term</th>
<th>J</th>
<th>Level (cm⁻¹)</th>
<th>Uncertainty (cm⁻¹)</th>
<th>Leading percentages</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2p6.3s2</td>
<td>1S</td>
<td>0</td>
<td>0.00</td>
<td></td>
<td>95 : 4 3p2 1S</td>
<td>L7147</td>
</tr>
<tr>
<td>3s.3p</td>
<td>3P</td>
<td>0</td>
<td>67918.03</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>68146.48</td>
<td></td>
<td></td>
<td></td>
<td>L7147</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>68615.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

nist_level_reader reads the electron configuration, term symbol, total angular momentum, and the corresponding energies. It computes the statistical weights and converts the energy into the TMAP format, i.e., frequency (in Hz) to the ionization limit. It creates two output files, one with fine-structure (FS) splitting as given in the NIST data, and one without FS splitting which is used for PRO2, and a warning/log file, which tracks the symbols that were used for excited configurations, if there were any. The program tries to automatically recognize excited configurations and uses default symbols to mark them in the TMAP format. An interactive mode allows the user to assign symbols at runtime, which is useful if there are more different excited configurations than the number of default symbols.

7.2 nist_line_reader

The program nist_line_reader parses line transitions from the NIST database. The process is very similar to the last program – it reads the configuration, term, and total angular momentum of both involved levels and the corresponding oscillator strength \( f_{ik} \). It creates an output file that can be used for a TMAD atom file with fine-structure splitting (i.e., used by LINE1_PROF). To get an atomic data file suitable for PRO2, one can use fs_split_merge, which splits or merges multiplets according to the multiplet line strength splitting. Sample input data for nist_line_reader is
<table>
<thead>
<tr>
<th>Observed Wavelength (Vac (Å))</th>
<th>Ritz Wavelength (Vac (Å))</th>
<th>Rel. Int.</th>
<th>(s⁻¹)</th>
<th>fik</th>
<th>Acc. (cm⁻¹)</th>
<th>Ei (cm⁻¹)</th>
<th>Lower level</th>
<th>Upper level</th>
<th>Conf.</th>
<th>Term</th>
<th>J</th>
<th>Conf.</th>
<th>Term</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>371.299</td>
<td>371.295</td>
<td>120</td>
<td>1.1e+09</td>
<td>6.8e-02</td>
<td>D</td>
<td>67918.03 - 337245.64</td>
<td>3s.3p</td>
<td>3P*</td>
<td>0</td>
<td>3s.5d</td>
<td>3D</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>371.504</td>
<td>371.501</td>
<td>150</td>
<td>1.5e+09</td>
<td>5.2e-02</td>
<td>D</td>
<td>68146.48 - 337245.64</td>
<td>3s.3p</td>
<td>3P*</td>
<td>1</td>
<td>3s.5d</td>
<td>3D</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>372.001</td>
<td>372.000</td>
<td>200</td>
<td>2.0e+09</td>
<td>5.8e-02</td>
<td>D</td>
<td>68615.17 - 337432.26</td>
<td>3s.3p</td>
<td>3P*</td>
<td>2</td>
<td>3s.5d</td>
<td>3D</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>372.149</td>
<td>372.149</td>
<td>5.1e+08</td>
<td>1.1e-02</td>
<td>D</td>
<td>68615.17 - 33724.91</td>
<td>3s.3p</td>
<td>3P*</td>
<td>2</td>
<td>3s.5d</td>
<td>3D</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 7.3 fs_split_merge

This program utilizes the multiplet line strength splitting provided by TMAP program multiplet (Rauch 2008). It can either split the total oscillator strength of an entire multiplet in fitting splitted oscillator strengths, or it can make the inverse calculation and propose a total oscillator strength from multiple splitted oscillator strengths. Input data is expected to be in TMAP format without any section beginning / leading characters or trailing zeros.

### 8 Summary

We compiled a revised, extensive P model atom. It is publicly accessible via TMAD which is part of the GAVO project. As sources for the atomic data, we mainly used the NIST Atomic Spectra Database, the CHIANTI database, and the atomic line list provided by the University of Kentucky (Chapter 3). The new model atoms can be used for the calculation of synthetic spectra and precise abundance measurements. Grotrian diagrams of all ionization stages can be found in Appendix B and in TMAD.

Supplementary software was created to transform the standard output of NIST and other databases to TMAP format. Source codes are available in Appendix D.

Test calculations with the new P model atom were performed to determine the dominant ionization stages (Figure 14) and characteristic P lines for different effective temperatures (Table 3). Ionization fractions from those models can be found in Appendix C.

A preliminary analysis of the hot sdB-type star CPD-20°1123 was performed. Starting parameters were adopted from Naslim et al. (2012) and models for $T_{\text{eff}}/K \in [24500, 25000, 25500, 26500]$ and $\log g \in [4.9, 5.0, 5.1]$ were calculated. The prominent P III lines $\lambda \lambda 4059.31, 4222.12, 4246.72$ Å were well reproduced. We determined an abundance of $n(P)/n(H) = (7.1 \pm 1.0) \times 10^{-7}$ (approximately three times solar) which is within error limits well in agreement with the measurement of Naslim et al. (2012). $n(P)/n(H) = (6.8 \pm 1.3) \times 10^{-7}$. [2012].
References


Demtröder, W. 2010, Experimentalphysik 3: Atome, Moleküle und Festkörper, 4th edn., Experimentalphysik / Wolfgang Demtröder (Berlin: Springer), XXI, 650

Haken, H. & Wolf, H. 2005, The Physics of Atoms and Quanta, 7th edn. (Berlin: Springer-Verlag), XX, 520


Kantorovich, L. 1949, Trudy Mat. Inst. Steklov., vol. 28, 104

Kastberg, A. 2016, Lecture Notes Atomic Physics (Nice: Sophia Antipolis University Nice)


Unsöld, A. & Baschek, B. 2005, Der neue Kosmos: Einführung in die Astronomie und Astrophysik, 7th edn. (Berlin: Springer-Verlag)


Werner, K. 1986, Astronomy & Astrophysics, 161, 177

Werner, K. 2013, Vorlesungsskript: Physik der Sternatmosphären I WS 2013/14 (Tübingen: Universität Tübingen)


Lists of Figures and Tables

List of Figures

1. LS coupling ................................................................. 2
2. Voigt profile ................................................................. 6
3. Grotrian diagram of PⅤ .................................................. 11
4. P ionization fractions for $T_{\text{eff}} = 20\,000$ K and log $g = 7.0$ ......... 12
5. Grotrian diagrams PⅡ - IV with intercombination lines ............................... 15
6. Hertzsprung-Russell diagram ............................................... 16
7. Evolutionary tracks off the main sequence ......................................... 17
8. Extreme Horizontal Branch .................................................... 18
9. P ionization fractions in CPD-20°1123 ....................................... 19
10. PⅢ lines in the model spectrum of CPD-20°1123 .............................. 20
11. Spectrum and model of CPD-20°1123 ........................................ 21
12. TMAD homepage .................................................................. 27
13. Grotrian diagrams PⅠ - XV ................................................... 28
14. Ionization fractions for $T_{\text{eff}} = 20 \text{ kK} - 500 \text{ kK}$ ............................... 33

List of Tables

1. Statistics of P ions .............................................................. 10
2. Parameters for our TMAP test calculations ...................................... 11
3. Most prominent P lines at different $T_{\text{eff}}$ and log $g$ .......................... 13
Acknowledgements

I thank all the people who supported me during my bachelor thesis. Especially I thank

- **Prof. Dr. Klaus Werner**
  for the possibility to write this thesis at the IAAT and his interesting lectures.

- **Dr. Thomas Rauch**
  for the supervision, discussing details of my work and proofreading my thesis.

- **Lisa Löbling and Denny Hoyer**
  for answering all my questions and being very supportive and engaging colleagues.

- **My parents and my family**
  for their continuous support during my studies.
A TMAD interface

Figure 12: TMAD homepage, overview of model atoms.
Figure 13: Grotrian diagrams for PⅠ - XV. Energy levels are shown as horizontal lines, the corresponding electron configuration (partly shortened) is shown in red, the term symbol is indicated at the top. Connecting black lines represent line transitions with known $f$-values.
Figure 13: Continued.
Figure 13: Continued.
ionization energy 3 423 000 cm$^{-1}$
ionization energy 3 866 950 cm$^{-1}$
ionization energy 4 521 700 cm$^{-1}$

Figure 13: Continued.
Figure 13: Continued.
C  Ionization at different effective temperatures

Figure 14: Ionization fractions for $T_{\text{eff}} = 20 \text{ kK} - 500 \text{ kK}$. 

![Plot showing ionization fractions for different effective temperatures]
Figure 14: Continued.
Figure 14: Continued.
D Source codes

Listing 1: nist_level_reader.cpp

```cpp
// ==============================================================
// Name : nist_level_reader
// Author : Michael Knoerzer
// Version : 1.1 (2017-07-04)
// Copyright : Copyright (c) 2017 Michael Knoerzer
// Description : reads in level energies from NIST (ascii output) and
// returns TMAD data with and without fine structure splitting
// ==============================================================

#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <time.h>
#include <algorithm>
#include <iomanip>
using namespace std;

// struct for excited state
struct exc_t
{
    // part of the full string to find it, char for the A10 char
    string str;
    char c;
};

// struct for fs part of level state
struct level_fs
{
    string J;
    string energy;
};

// struct for whole level / multiplet
struct multiplet
{
    string config, term;
    string parent_a10;
    string parent_a10_fs;
    string exc_str;
    vector<string> vec_warnings;
    vector<level_fs> levels_fs;
};

// struct to sort levels and check for duplicate energies
struct level_simple
{
    string name;
    double energy;
};
```
// globals
bool interactive = false;
int warnings;
int errors;
char exc_default[17] = {
    ' ','\','"','^','#','*','~','%','&','(','/','$','!','+','\'\'};
vector<exc_t> vec_exc;

// functions
void do_stuff(string filename);
inline string to_upper(string s);
inline int roman(const string & str);
inline string int_to_string(const int i);
string make_a10(string atom, string config, string term, string exc_string);
inline string make_a10_fs(string atom, string config, string term, string J, string exc_string);
double J_to_double(string J);
inline string J_to_c(string J);
inline string a10_term(string s);
double str_to_double(string s);

// read, compute, write //
void do_stuff(string filename)
{
    // arrays of multiplets, reset excited states
    vector<multiplet> multis;
    vec_exc.clear();
    // read in stream, line buffer
    ifstream input;
    ofstream out_l, out_lfs, out_log;
    // stuff for parsing
    string line;
    string atom = "";
    string parent, parent_a10, parent_a10_fs;
    string e_ion;
    // number of warnings, errors
    warnings = 0;
    errors = 0;
    int linecounter = 0;
    // open file and read line by line
    cout << "attempting to open file: " << filename << endl;
    input.open(filename.c_str());
    if(input.is_open())
    {
        // try to find "limit" to read in parent, ionization energy and own name
        while(getline(input,line))
        {
            stringstream s(line);
            string s1, s2, s3, s4, s5, s6, s7, s8, s9, s10;
            s >> s1 >> s2 >> s3 >> s4 >> s5 >> s6 >> s7 >> s8 >> s9 >> s10;
            if("Limit" == s6)
            {
                atom = to_upper(s1) + int_to_string(roman(s2) - 1);
parent = s1 + s2 + s3 + s4;

// parse s4... can look like 1S<0> or 4P*<3/2>
size_t found, found2;
found = s4.find("<");
found2 = s4.find(">");
parent_a10 = make_a10(s1 + int_to_string(roman(s2)), s3.substr(1), s4.substr(0, found), "");
parent_a10_fs = make_a10_fs(s1 + int_to_string(roman(s2)), s3.substr(1), s4.substr(0, found), s4.substr(found + 1, found2 - (found + 1)), "");
e_ion = s10;

cout << "atom: " << atom << " || e_ion: " << e_ion << " || J_parent: " << s4.substr(found + 1, found2 - (found + 1)) << endl;
break;
}

// check if we found atom, energy, etc
if(0 == atom.size())
{
++errors;
cout << "ERROR: couldn't find atom name and e_ion in file: " << filename << endl;
return;
}

// reset, now read in data
input.clear();
input.seekg(0, std::ios::beg);

// read in stuff
while(getline(input, line))
{
    // check if we find "Limit" -> stop reading
    size_t found;
    found = line.find("Limit");
    if(found != string::npos)
        break;

    linecounter ++;

    // get line size
    int size_line = line.size();

    // skip blank and too-short lines
    if(size_line <= 1)
        continue;

    // read in stuff
    // if the output format should change in the future
    // one has to adjust the input variables
    string s1, s2, s3, s4, s5, s6, s7, s8;
    stringstream s(line);
    s >> s1 >> s2 >> s3 >> s4 >> s5 >> s6 >> s7 >> s8;

    // check if -------------------
    if("-----" == s1.substr(0, 5))
        continue;

    // check if new multiplet or just J + energy
    if("|" == s1 && "|" == s2)
    {

// check if more empty fields exist
if ("|" == s3 && "|" == s4) {
    // J + level also empty -> blank line
    continue;
} else if ("|" == s4 && "|" == s6) {
    // looking good, s3+s5 is data, there should be at least one multiplet
    if (multis.empty()) {
        // this shouldnt happen
        cout << "ERROR: no config/term info for data found:" << endl;
        << line << endl;
        continue;
    }
    // else get most recent multiplet and add data
    multiplet & m = multis.back();

    level_fs fs;
    fs.J = s3;
    fs.energy = s5;

    // add to m
    m.levels_fs.push_back(fs);
} else {
    // unknown input format
    ++errors;
    cout << "ERROR: unknown input format 2: " << endl;
    << line << endl;
    continue;
}

else if ("|" == s2 && "|" == s4) {
    // skip table header
    if ("Configuration" == s1)
        continue;

    // new multiplet
    multiplet m;
    m.config = s1;
    m.term = s3;
    m.parent_a10 = parent_a10;
    m.parent_a10_fs = parent_a10_fs;

    // check if we have J + energy
    if ("|" == s5 && "|" == s6)
        {
// no J + energy
++warnings;
m. vec_warnings . push_back("WARNING: multiplet may be incomplete: no J and energy found:");
m. vec_warnings . push_back(line);
cout << "WARNING: multiplet may be incomplete: no J and energy found:" << endl << line << endl;
}

else if("|" == s6 && "|" == s8) {

  // grab data
  level_fs fs;
  fs.J = s5;
  fs.energy = s7;

  // check energy for ?s
  do
  { found = fs.energy.find('?');
    if(found != string::npos)
    {
      // warning
      ++warnings;
      m. vec_warnings . push_back("WARNING: found '?' in energy... level may not be real: " + m.config + " " + m.term + " --> " + fs.energy);
      m. vec_warnings . push_back(line);
      cout << "WARNING: found '?' in energy... level may not be real: " << m.config << " " << m.term << " --> " << fs.energy << endl;
    }
    fs.energy.replace(found,1,"");  
  }while(found != string::npos);

  // add to current multiplet
  m.levels_fs.push_back(fs);
}
else
{
  // unknown input
  ++errors;
  cout << "ERROR: unknown input format 3: " << endl << line << endl;
  continue;
}

// check if config term has "?" or excited states i.e. (2P*), etc
size_t found1,found2;
found1 = m.config.find('(');
found2 = m.config.find(')');

if(found1 != string::npos)
{
  // looks like we found an excited state
  if(found2 != string::npos)
  {
    // we got one
    string exc = m.config.substr(found1+1,found2-(found1+1));
    m.exc_str = exc;
  }
// check if we already have it in our array
bool found = false;

for(vector<exc_t>::iterator it = vec_exc.begin(); it != vec_exc.end(); it++)
{
    if(exc == it->str)
    {
        // found it
        found = true;
        break;
    }
}

// if not found, check if interactive, else take from default array
if( ! found)
{
if(interactive)
{
    // ask user for input
    cout << "found excited state " << exc << " in the following line" << endl << line << endl;
    cout << "please enter the matching char (blank, ','," << endl;
    string s;
do
    {
        getline(cin,s);
    } while(s.size() != 1);

    // add to vector
    exc_t e;
    e.str = exc;
    e.c = s[0];
    vec_exc.push_back(e);
}
else
{
    // first time info about interactive mode
    if(0 == vec_exc.size())
    {
        m.vec_warnings.push_back("INFO: you can manually edit excited states in interactive mode ("-i" parameter)");
        cout << "INFO: you can manually edit excited states in interactive mode ("-i" parameter)" << endl;
    }
}

    // take from default
    exc_t e;
    e.str = exc;
    e.c = exc_default[vec_exc.size()];
    vec_exc.push_back(e);

    // warning
    ++warnings;
    m.vec_warnings.push_back("WARNING: added default excited state: " + e.str + " -> '" + e.c + "'");
    m.vec_warnings.push_back(line);
    cout << "WARNING: added default excited state: " << e.str << " -> '" << e.c << "'" << endl;
}
else
{
    // '(' but no ')' -> malformed data ?
    ++errors;
    cout << "ERROR: unknown input format in config (excited state) : " << endl << line << endl;
    continue;
}
} // end search for brackets / exc state

// check for ?s in config....
do
{
    found1 = m.config.find('?');
    if(found1 != string::npos)
    {
        // warning
        ++warnings;
        m.vec_warnings.push_back("WARNING: found '?' in configuration... level may not be real: " + m.config);
        m.vec_warnings.push_back(line);
        cout << "WARNING: found '?' in configuration... level may not be real: " << endl;
        m.config.replace(found1,1,"");
    }
}while(found1 != string::npos);

// add to multiplet vector
multis.push_back(m);
}
else
{
    // unknown input
    ++errors;
    cout << "ERROR: unknown input format 1: " << endl << line << endl;
}
} // end getline

// check if we got any useful data
if(multis.empty())
{
    ++errors;
    cout << "ERROR: no data was parsed ! file: " << filename << endl;
    return;
}

// open output file
out_l.open((filename+"_output.level").c_str());
out_lfs.open((filename+"_output.level_fs").c_str());
out_log.open((filename+"_output.warnings").c_str());

// make a copy of each level (fs splitted)
vector<level_simple> levels_check;

// for each multiplet calculate
for(vector<multiplet>::iterator it = multis.begin(); it != multis.end(); it++)
{

double e_i = str_to_double(e_ion);

// speed of light
const double c = 2.99792458e10;  // cms/s 3*10^-10

// added energy of all splitted levels
double e_all = 0.0;
// stat. weights of all splitted levels
double w_all = 0.0;

// check for warnings first
if(it->vec_warnings.size() > 0)
{
    // output warnings in log
    for(vector<string>::iterator jt = it->vec_warnings.begin(); jt != it->vec_warnings.end(); jt++)
    {
        out_log << *jt << endl;
    }
}

// for each splitted level
for(vector<level_fs>::iterator jt = it->levels_fs.begin(); jt != it->levels_fs.end(); jt++)
{
    // name with J
    string name_fs = make_a10_fs(atom,it->config,it->term,jt->J,it->exc_str);
    // energy
    double e_fs = str_to_double(jt->energy);
    // stat weight
    double w_fs = J_to_double(jt->J)*2 + 1;

    e_all += e_fs*w_fs;
    w_all += w_fs;

    // output in SYN file
    out_lfs << name_fs << it->parent_a10_fs << " " << scientific << setprecision(12) << uppercase << (e_i - e_fs)*c << nouppercase << fixed << setprecision(1) << setw(9) << w_fs << string(32, ' ') << " NIST" << endl;
}

// output in MOD file
out_l << make_a10(atom,it->config,it->term,it->exc_str) << it->parent_a10 << " " << scientific << setprecision(12) << uppercase << (e_i - e_all/w_all)*c << nouppercase << fixed << setprecision(1) << setw(9) << w_all << string(32, ' ') << " NIST" << endl;

// check for duplicates in level_check
cout << scientific << setprecision(12);
out_log << scientific << setprecision(12);
string last_name = "";
double last_energy = 0.0;
for(vector<level_simple>::iterator it = levels_check.begin(); it != levels_check.end(); it++)
{
    if(it->energy == last_energy)
    {
        ++warnings;
    }
out_log << "WARNING: duplicate level energy found: " << uppercase
   << last_energy << nouppercase << " levels: " << it->name << " -> " << last_name << endl;

cout << "WARNING: duplicate level energy found: " << uppercase <<
   last_energy << nouppercase << " levels: " << it->name << " <-> 
" << last_name << endl;
}

last_name = it->name;
last_energy = it->energy;
}
}

else
{
  ++ errors;
  cout << "ERROR: couldn’t open file: " << filename << endl << endl;
}

// info
cout << "End parsing file: " << filename << " lines parsed: " << linecounter
   << endl << warnings << " warning(s) and " << errors << " error(s) 
occurred" << endl << endl;
}

int main(int argc, char* argv[])
{
  if(2 > argc)
  {
    cout << "NIST level reader" << endl;
    cout << "input: NIST level energies (ascii format)" << endl;
    cout << "output: <filename>_output.level - TMAD data without fine 
structure splitting" << endl;
    cout << " <filename>_output.level_fs - TMAD data with fine 
structure splitting" << endl;
    cout << "usage: \"nist_level_reader [-i] <file 1> <file 2> ... <file n>\" 
  " << endl;
    cout << "parameter \
\"-i\": interactive mode (labeling excited states, 
malformatted data)" << endl;
    cout << " if not set no user input is expected at all" << endl;
    return 0;
  }

  // performance check
  clock_t t1,t2;
  t1 = clock();

  // for each actual argument run parse subroutine
  for(int i = 1; i < argc; i++)
  {
    string s(argv[i]);

    // check if interactive mode
    if("-i" == s)
    {
      interactive = true;
      continue;
    }

    do_stuff(s);
  }

  // precision for clock/time
cout << std::setprecision(4) << scientific;

// performance check
t2 = clock();
float diff = ((float)t2 - (float)t1) / CLOCKS_PER_SEC;
cout << "time to run the program: " << fixed << diff << "s" << endl;

return 0;

// return uppercase string
inline string to_upper(string s)
{
    std::transform(s.begin(), s.end(), s.begin(), ::toupper);
    return s;
}

// converts romann numerals to arabic
// returns 0 on invalid input
inline int roman(const string & str)
{
    int big = 0;
    int sum = 0;
    for (int i = (str.size() - 1); i >= 0; i--)
    {
        int cur = -1;
        switch(str[i])
        {
            case 'M':
                cur = 1000;
                break;
            case 'D':
                cur = 500;
                break;
            case 'C':
                cur = 100;
                break;
            case 'L':
                cur = 50;
                break;
            case 'X':
                cur = 10;
                break;
            case 'V':
                cur = 5;
                break;
            case 'I':
                cur = 1;
                break;
        }
        if (cur < 0)
        {
            // invalid input
            return 0;
        }
        if (cur >= big)
        {
            big = cur;
```cpp
595    sum += cur;
596  }
597  else
598    sum -= cur;
599  }
600  return sum;
601 }
602
603 // convert int to string c++98
604 inline string int_to_string(const int i)
605 {
606    stringstream ss;
607    ss << i;
608    return ss.str();
609 }
610
611 // make a10 string
612 string make_a10(string atom, string config, string term, const string exc_string)
613 {
614    // check atom size
615    size_t size_atom = atom.size();
616    if(size_atom <= 1 || size_atom >= 4)
617    {
618        // too short / long
619        ++errors;
620        cout << "ERROR: invalid atom name: " << atom << endl;
621        return string("1234567890");
622    }
623  }
624  else if(2 == size_atom)
625  {
626      // if only len 2 (i.e. P3) pad one zero (i.e. P30)
627      atom += "0";
628  }
629  
630  // get only most outer shell of configs like 1s2.2s2.2p6.3s etc
631  size_t found = config.find_last_of(".");
632  string conf;
633  if(found != string::npos)
634      conf = config.substr(found+1);
635  else
636    conf = config;
637  
638  size_t size_conf = conf.size();
639  if(size_conf <= 1 || size_conf >= 5)
640  {
641    // too short / long
642    ++errors;
643    cout << "ERROR: invalid atom configuration: " << config << " ( " << atom
644    << config << " " << term << " )" << endl;
645    return string("1234567890");
646  }
647  else if(4 == size_conf)
648  {
649    // warning, might be too long
650    ++warnings;
651    cout << "WARNING: configuration size too big: " << config << " ( " << atom
652    << config << " " << term << " )" << endl;
653  }
654  
655  string atom_conf = atom + conf;
656  while(atom_conf.size() < 7)
657    atom_conf += " ";
46
```
atom_conf = to_upper(atom_conf + a10_term(term));

// check if we have exc state to add
if(exc_string.size() > 0) {
  // check if we already have it in our array
  bool found = false;
  char c;
  for(vector<exc_t>::iterator it = vec_exc.begin(); it != vec_exc.end(); it ++)
  {
    if(exc_string == it->str)
    {
      // found it
      found = true;
      c = it->c;
      break;
    }
  }
  if(! found) {
    ++errors;
    cout << "ERROR: excited state not found while making a10 string: " << exc_string << endl;
    return string("1234567890");
  }
  // check if not ground state (i.e. blank)
  if(c != ' ')
  {
    // cout << "exc state replace test " << atom_conf << " -> " ;
    // replace with exc state
    atom_conf.replace(5,1,1,c);
    //cout << atom_conf << endl;
  }
  return atom_conf;
}

// make a10 string with J
inline string make_a10_fs(string atom, string config, string term, string J, string exc_string)
{
  return make_a10(atom,config,term,exc_string).replace(6,1,J_to_c(J));
}

double J_to_double(string J)
{
  size_t size = J.size();
  if(size < 1 || size > 4) {
    // too short / long
    ++errors;
    cout << "ERROR: invalid J: " << J << endl;
    return 0.0;
  }
if (1 == size) return str_to_double(J);

double d1, d2;
size_t found = J.find("/");
if (found != string::npos) {
  std::stringstream convert(J.substr(0, found));
  convert >> d1;
  convert.clear();
  convert.str(J.substr(found + 1));
  convert >> d2;
  return (d1 / d2);
}

// else not valid
++errors;
cout << "ERROR: invalid J: " << J << endl;
return 0.0;

// make J to 1 character for a10 string
inline string J_to_c(string J) {
  size_t size = J.size();
  if (size < 1 || size > 4) {
    // too short / long
    ++errors;
    cout << "ERROR: invalid J: " << J << endl;
    return string("?");
  }

  // if size == 1, should be 0, 1, 2, 3, etc
  if (1 == size) return J;

  size_t found = J.find("/");
  if (found != string::npos) {
    // should be 1/2, 3/2, ... 11/2, 13/2, 15/2
    int i;
    istringstream convert(J.substr(0, found));
    convert >> i;
    if (i > 10) {
      switch (i) {
      case 11: return (string("E"));
      case 13: return (string("T"));
      case 15: return (string("F"));
      default:
        ++errors;
    }

    // if size == 1, should be 0, 1, 2, 3, etc
    if (1 == size) return str_to_double(J);
  }
    cout << "ERROR: invalid J: " << J << endl;
    return(string("?"));
  }
  return(int_to_string(i));
}

    // not valid
    ++errors;
    cout << "ERROR: invalid J: " << J << endl;
    return string("?");
}

    // return pad up to size 3, replace * with capital o
inline string a10_term(string s)
{
    size_t size = s.size();
    if(1 == size || size > 3)
    {
        // too short / long
        ++errors;
        cout << "ERROR: invalid term: " << s << endl;
        return string("???");
    }
    if(2 == size)
    {
        // even term like 1S, 2P, etc... pad
        return(s + " ");
    }
    // else odd term like 2P*
    return(s.substr(0,2) + "O");
}

// ==============================================================
// convert str to double and remove brackets beforehand
// ==============================================================
double str_to_double(string s)
{
    if( (s[0] == '(' || s[0] == '[') && (s[s.size()-1] == ')' || s[s.size()-1] == ']') )
    {
        // first and last are brackets
        return str_to_double(s.substr(1,s.size()-2));
    }
    size_t found1 = s.find("(");
    if(found1 != string::npos)
    {
        // found ( left bracket
        size_t found2 = s.find(""));
        if(found2 == std::string::npos)
        {
            // but no brackets, try anyway
            double d;
            stringstream convert(s);
            convert >> d;
            return d;
        }
    }
}

// remove additional stuff in brackets
return str_to_double(s.erase(found1,found2));

// check for +x
found1 = s.find("+x");
if(found1 != std::string::npos)
{
    // remove '+x' and following characters
    s.erase(found1);
}

// check for ?
found1 = s.find("?" biod
if(found1 != std::string::npos)
{
    // remove '? ' and following characters
    s.erase(found1);
}

// looking good
double d;
stringstream convert(s);
convert >> d;
return d;

// ==============================================================
// end str to double
// ==============================================================
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <time.h>
#include <algorithm>
#include <iomanip>
using namespace std;

// struct for excited state
struct exc_t
{
    // part of the full string to find it, char for the A10 char
    string str;
    char c;
};

// globals
bool interactive = true;
int warnings;
int errors;
char exc_default[17] = {'!', '(', ')', '*', '/', '$', '^', '#', '~', '=', '(', ')', '\';
vector<exc_t> vec_exc;

// functions
void do_stuff(string filename, string atom);
inline string to_upper(string s);
inline string int_to_string(const int i);
string make_a10(string atom, string config, string term, string exc_string);
inline string make_a10_fs(string atom, string config, string term, string J, string exc_string);
inline string J_to_c(string J);
inline string a10_term(string s);
inline string a10_wrapper(string atom, string config, string term, string J, string line);

void do_stuff(string filename, string atom) {
    // we use the same data format as in nist_level_reader except no parent level
    // here, two levels, which we convert to a10 strings, make a transition

vec_exc.clear();

// read in stream, line buffer
ifstream input;
ofstream out_syn;

// stuff for parsing
string line;

// number of warnings, errors
warnings = 0;
errors = 0;
int linecounter = 0;

// open file and read line by line
cout << "attempting to open file: " << filename << endl;
input.open(filename.c_str());
if(input.is_open())
{
  // open output file
  out_syn.open((filename+"_lines_fs").c_str());

  // read all the (text) lines
  while(getline(input,line))
  {
    linecounter++;
    stringstream s(line);
    string f_ik, low_conf, low_term, low_J, up_conf, up_term, up_J;
    string s1;

    // not all fields will be filled so we have to keep track, start with position 1
    int position = 1;

    // check first field
    s >> s1;

    // check if dashes or keywords -> skip
    if(0 == s1.compare("Vac") || 0 == s1.compare("Wavelength") || 0 == s1.compare("Observed") || "-----" == s1.substr(0,5))
    continue;

    // skip to pos 5 where f_ik should be
    // failsafe in case we get a random file so we don't loop infinite
    int failsafe = 0;
    while(position < 5)
    {
      failsafe ++;
      if(0 == s1.compare ("|")
      {
        position++;
        failsafe = 0;
      }
    }
    s >> s1;

    // if failsafe gets too big we got malformed data
    // -> prevent infinite loop
    // there shouldn't be more than 4 strings without a "|
    if(failsafe > 4)
    break;
  }
if(failsafe > 4)
  continue;
// 5th field should be f_ik, if not skip this line
if (0 == s1.compare("|"))
{
    // no f_ik found
    continue;
}
f_ik = s1;

// skip to position 8 where we hopefully find the rest
failsafe = 0;
while (position < 8)
{
    s >> s1;
    failsafe ++;
    if (0 == s1.compare("|"))
    {
        position ++;
        failsafe = 0;
    }
    if (failsafe > 4)
    {
        break;
    }
    if (failsafe > 4)
    {
        continue;
    }

    // rest should be in order
    s >> low_conf >> s1 >> low_term >> s1 >> low_J >> s1 >> up_conf >> s1 >> up_term >> s1 >> up_J;
    string low_10 = a10_wrapper(atom, low_conf, low_term, low_J, line);
    string up_10 = a10_wrapper(atom, up_conf, up_term, up_J, line);
    // write to output
    out_syn << low_10 << up_10 << " 1 1 " << f_ik << endl;
}
else
{
    ++errors;
    cout << "ERROR: couldn't open file: " << filename << endl << endl;
}
// info
cout << "End parsing file: " << filename << " lines parsed: " << linecounter << endl << warnings << " warning(s) and " << errors << " error(s) occurred" << endl << endl;

int main(int argc, char* argv[])
{
    // we dont know the element and ionization stage from the data so we ask the
string atom, filename;
cout << "NIST line reader" << endl;
cout << "input: NIST ASD lines data (ascii format) with f_{ik} values" << endl;
cout << "output: TMAD compatible data" << endl << endl;

// get atom name

cout << "Please enter the element code and ionization stage (two or three characters total):" << endl;
do {
    getline(cin, atom);
} while(atom.size() != 2 && atom.size() != 3);

cout << "Please enter the file name to parse:" << endl;
do {
    getline(cin, filename);
} while(filename.size() == 0);

// performance check

clock_t t1, t2;
t1 = clock();
do_stuff(filename, atom);

// precision for clock/time

cout << std::setprecision(4) << scientific;

// performance check

t2 = clock();
float diff = ((float)t2-(float)t1) / CLOCKS_PER_SEC;
cout << "time to run the program: " << fixed << diff << " s" << endl;

// done
return 0;

// return uppercase string
inline string to_upper(string s) {
    std::transform(s.begin(), s.end(), s.begin(), ::toupper);
    return s;
}

// convert int to string c++98
inline string int_to_string(const int i) {
    stringstream ss;
    ss << i;
    return ss.str();
}

// make a10 string

string make_a10(string atom, string config, string term, const string exc_string) {
    // check atom size
    size_t size_atom = atom.size();
    if(size_atom <= 1 || size_atom >= 4)
{ // too short / long
  ++errors;
  cout << "ERROR: invalid atom name: " << atom << endl;
  return string("1234567890");
} else if(2 == size_atom)
{
  // if only len 2 (i.e. P3) pad one zero (i.e. P30)
  atom += "0";
}

// get only most outer shell of configs like 1s2.2s2.2p6.3s etc
size_t found = config.find_last_of(".");
string conf;
if(found != string::npos)
  conf = config.substr(found+1);
else
  conf = config;

size_t size_conf = conf.size();
if(size_conf <= 1 || size_conf >= 5)
{
  // too short / long
  ++errors;
  cout << "ERROR: invalid atom configuration: " << conf << " ( " << atom
       << conf << " )" << endl;
  return string("1234567890");
}
else if(4 == size_conf)
{
  // warning, might be too long
  ++warnings;
  cout << "WARNING: configuration size too big: " << conf << " ( " << atom
       << conf << " )" << endl;
}

string atom_conf = atom + conf;
while(atom_conf.size() < 7)
  atom_conf += " ";

// all uppercase
atom_conf = to_upper(atom_conf + a10_term(term));

// check if we have exc state to add
if(exc_string.size() > 0)
{
  // check if we already have it in our array
  bool found = false;
  char c;
  for(vector<exc_t>::iterator it = vec_exc.begin(); it != vec_exc.end(); it++)
  {
    if(exc_string == it->str)
    {
      // found it
      found = true;
      c = it->c;
      break;
    }
  }

  // if not found -> error
  if(! found)


```cpp
306   {
307       ++errors;
308       cout << "ERROR: excited state not found while making a10 string: " <<
309           exc_string << endl;
310       return string("1234567890");
311   }
312
313   // check if not ground state (i.e. blank)
314   if(c != ' ')
315   {
316       // cout << "exc state replace test " << atom_conf << " -> " ;
317       // replace with exc state
318       atom_conf.replace(5,1,1,c);
319       // cout << atom_conf << endl;
320   }
321   return atom_conf;
322 }
323
324 }
325
326 // make a10 string with J
327 inline string make_a10_fs(string atom, string config, string term, string J,
328                         string exc_string)
329 {
330       return make_a10(atom,config,term,exc_string).replace(6,1,J_to_c(J));
331 }
332
333 // make J to 1 character for a10 string
334 inline string J_to_c(string J)
335 {
336       size_t size = J.size();
337       if(size < 1 || size > 4)
338       {
339           // too short / long
340           ++errors;
341           cout << "ERROR: invalid J: " << J << endl;
342           return string("?");
343       }
344
345       // if size == 1, should be 0,1,2,3,etc
346       if(1 == size)
347       return J;
348       size_t found = J.find("/");
349       if(found != string::npos)
350       {
351           // should be 1/2, 3/2, ... 11/2, 13/2, 15/2
352           int i;
353           istringstream convert(J.substr(0,found));
354           convert >> i;
355           if(i > 10)
356           {
357               switch(i)
358               {
359                   case 11:
360                       return(string("E"));
361                   case 13:
362                       return(string("T"));
363                   case 15:
364                       return(string("F"));
365               }
366           }
56
```
default:
++errors;
cout << "ERROR: invalid J: " << J << endl;
return(string("?"));
}

return(int_to_string(i));

// not valid
++errors;
cout << "ERROR: invalid J: " << J << endl;
return(string("?"));

// return pad up to size 3, replace * with capital o
inline string a10_term(string s)
{
    size_t size = s.size();
    if(1 == size || size > 3)
    {
        // too short / long
        ++errors;
        cout << "ERROR: invalid term: " << s << endl;
        return(string("???"));
    }
    if(2 == size)
    {
        // even term like 1S, 2P, etc... pad
        return(s + " ");
    }
    return(s.substr(0,2) + "O");
}

inline string a10_wrapper(string atom, string conf, string term, string J, string line)
{
    size_t found1, found2;
    found1 = conf.find('(');
    found2 = conf.find(')');
    string exc = "";
    if(found1 != string::npos)
    {
        // check if config term has excited states i.e. (2P*), etc
        found2 = conf.find(')');
        string exc = "";
        if(found1 != string::npos)
        {
            // looks like we found an excited state
            if(found2 != string::npos)
            {
                // we got one
                exc = conf.substr(found1+1,found2-(found1+1));
                // check if we already have it in our array
                bool found = false;
                for(vector<exc_t>::iterator it = vec_exc.begin(); it != vec_exc.end(); it++)
{ if(exc == it->str)
    // found it
    found = true;
    break;
 }

// if not found, check if interactive, else take from default array
if( ! found)
{
    if(interactive)
    {
        // ask user for input
        cout << "found excited state " << exc << " in the following line
        " << endl << line << endl;
        cout << "please enter the matching char (blank, ',',\",'-',etc)" << endl;

        string s;
do {
            getline(cin,s);
        } while(s.size() != 1);

        // add to vector
        exc_t e;
        e.str = exc;
        e.c = s[0];
        vec_exc.push_back(e);
    }
    else
    {
        // first time info about interactive mode
        if(0 == vec_exc.size())
        {
            cout << "INFO: you can manually edit excited states in
            interactive mode ("-i" parameter)" << endl;
        }

        // take from default
        exc_t e;
        e.str = exc;
        e.c = exc_default[vec_exc.size()];
        vec_exc.push_back(e);

        // info
        cout << "INFO: added default excited state: " << e.str << " -> ',
        " << e.c << "'" << endl;
    }
}

// exc string already in array, nothing more to do
}

else
{
    // '(', but no ')') -> malformed data ??
    ++errors;
    cout << "ERROR: unknown input format in config (excited state): " << endl << line << endl;
    return("0123456789");
}

//return a10 string
return(make_a10_fn(atom, conf, term, J, exc));

// ================================================= ===========================
// end a10 wrapper
// ==============================================================

# include <iostream>
# include <sstream>
# include <fstream>
# include <vector>
# include <map>
# include <string.h>
# include <time.h>
using namespace std;

// struct to store a line transition with fs splitting (J)
// a line looks like this: 1-6 config 7:J 8-10 term, same with 10+ for 2nd level
// P302P412P0P302P532D 3 3 3.2570E-01
// ^ lower ^ upper ^ f_ik
// J can be 0,1,2,... or 1/2,3/2,5/2 etc
// for the latter it is 1,3,5,...,9,E,T,F

struct transition_fs
{
    char level_lower[7], level_upper[7];
    char J_lower, J_upper;
    char term_lower[4], term_upper[4];
    double f_ik;
    char multiplet_type;
};

// struct to store a line transition without fs splitting (so no J)
// level lower/upper are now stored as A10 strings
// we may have several possible f_ik for the same transition, so
// f_total is the total number of possible values for f_ik (for each fss transition)
// f_index is the #
// arranging is done while calculating the new f_ik
struct transition_no
{
    char level_lower_a10[11], level_upper_a10[11];
    double f_ik;
    char multiplet_type;
};
unsigned int f_index;
unsigned int f_total;
};

struct new_f_value
{
    double d;
    unsigned int index;
    unsigned int total;
};

struct for new f value: conversion value, and how many duplicates we can have
f_new = f_old / d

struct new_f_value
{
    double d;
    unsigned int index;
    unsigned int total;
};

tree/hashmap with the calculation values, index and index max

first level: number as char ('1','2',etc) singlet, doublet, etc
2nd level: S -> P, P -> D, etc (pair of two chars)
final level: 1/2 -> 3/2, etc as a pair of two chars
1/2 = 1, 5/2 = 5, 11/2 = E, 2=2, 4=4, etc, 1/2 -> 3/2 = <'1','3'>

typedef map<pair<char, char>, new_f_value> multiplet_type_fs_map;
typedef map<pair<char, char>, multiplet_type_fs_map> multiplet_type_map;
typedef map<char, multiplet_type_map> multiplet_map;
// the all mighty map
multiplet_map multiplet;

terminal values

functions

// converts str to double
double str_to_double(string s);

// actual program doing stuff
void FS_merge(string s);
void FS_split(string s);

// wrapper for logging stuff
void log_transition_fs(ostream &s, transition_fs t);
void log_transition(ostream &s, transition_no t);

// calculate the new f value for a fss transition, return transition without fss
transition_no calculate_f(transition_fs t);

// check angular momentum
inline bool check_nofs_L(transition_no t_no);
inline bool check_fs_L_J(transition_fs t/fs);

// init the hashmap
inline void init_hash_map();
// sort vector by index
bool sort_by_index(const transition_no &lhs, const transition_no &rhs) {
    return lhs.f_index < rhs.f_index;
}

// sort vector by left term first
bool sort_by_name(const transition_no &lhs, const transition_no &rhs);
bool sort_by_left_term(const string &lhs, const string &rhs);
bool inline sort_by_spdf(const char l, const char r);

char inline L_value(char c);

(sprintf(// globals

// output streams: lines (for mod file), redundant stuff (for completeness check), log (for errors/warnings)
// these could be inside the main procedure but this way they can also be used in other functions if needed
ofstream out_lines, out_red, out_log;
int warnings;
int errors;

// ///////////////////////////////////////////////// ////////////////////////////
// program start
// ///////////////////////////////////////////////// ////////////////////////////
int main(int argc, char* argv[])
{
    // check if arguments were given
    if(argc < 2)
    {
        cout << "FS split / merge " << endl;
        cout << "Usage: \"fs_split_merge [-s] <file 1> [-s] <file 2> ... [-s] <file N>\" " << endl;
        return 0;
    }
    // performance check
clock_t t1, t2;
t1 = clock();

    // initialize the multiplet map
    init_hash_map();

    // precision for cout
    cout << std::setprecision(4) << scientific;

    bool split = false;
    // for each real argument run parse subroutine
    for(int i = 1; i < argc; i++)
    {
        string s(argv[i]);

        // check if split, if yes try to split next file
        if("-s" == s)
        {
            split = true;
        }
186          continue;
187       }
188       // if split flag is set
189       if (split)
190          {
191             split = false;
192             cout << "splitting now: " << s << endl;
193             FS_split(s);
194             continue;
195          }
196       // else merge
197       FS_merge(s);
198   }
199   // performance check
200   t2 = clock();
201   float diff = ((float)t2-(float)t1) / CLOCKS_PER_SEC;
202   cout << "time to run the program: " << fixed << diff << " s" << endl;
203   // done
204   return 0;
205 }
206 // ==============================================================
207 // parses file, input should be a SYN file
208 // ==============================================================
209 void FS_split(string filename)
210 {
211     // start
212     cout << "Attempting to parse file: " << filename << endl;
213     // read in stream, line buffer
214     ifstream input;
215     string line;
216     transition_no t_no;
217     // number of warnings, errors
218     warnings = 0;
219     errors = 0;
220     // number of transitions parsed
221     int counter = 0;
222     // vector with all the lines
223     vector<transition_no> v_t_no;
224     // open file and read line by line
225     input.open(filename.c_str());
226     if (input.is_open())
227        {
228          // open output/log files
229          out_lines.open((filename+"_out.fs").c_str());
230          out_log.open((filename+"_out.fsinfo").c_str());
231          // set precision / scientific
232          out_lines << std::setprecision(4) << scientific;
233          out_log << std::setprecision(4) << scientific;
234          // log
235          out_log << "Attempting to parse file: " << filename << endl;
236          // read in stuff
237          }
while(getline(input,line))
{
    // get line size
    int size_line = line.size();

    // skip blank lines (=size 0)
    if(0 == size_line)
        continue;

    // skip comments (dots)
    if(line[0] == '.')
        continue;

    // skip short lines (should be 20 chars for lower+upper then " 1 1 " + another like 7 for the f-value)
    if(size_line < 25)
    {
        // log short lines
        warnings++;
        out_log << "WARNING: line too short: " << size_line << endl << line << endl;

        // skip to next line
        continue;
    }

    // read in stuff
    strcpy(t_no.level_lower_a10,line.substr(0,10).c_str());
    strcpy(t_no.level_upper_a10,line.substr(10,10).c_str());

    // TODO: check some syn files if 25+13 is ok with most files
    t_no.f_ik = str_to_double(line.substr(25,13));

    t_no.f_index = 0;
    t_no.f_total = 0;

    // check if both are of the same multiplet
    if(t_no.level_lower_a10[7] != t_no.level_upper_a10[7])
    {
        // lower and upper term are not of the same multiplet
        errors++;

        cout << "ERROR: lower and upper term are not of the same multiplet: " << endl;
        log_transition(cout,t_no);

        out_log << "ERROR: lower and upper term are not of the same multiplet:" << endl;
        log_transition(out_log,t_no);

        // skip to next line
        continue;
    }

    // are the same, so save the multi type
    t_no.multiplet_type = t_no.level_lower_a10[7];

    // check angular momentum, J
    if(! check_nofs_L(t_no))
    {
        errors++;

        cout << "ERROR with angular momentum or J" << endl;
        log_transition(cout,t_no);

        out_log << "ERROR with angular momentum or J" << endl;
    }
log_transition(out_log,t_no);

else
{
    // everything checks out, add to vector
    v_t_no.push_back(t_no);
}

// end while getline(input,line)

// close input
input.close();

// sort the vector by name, so the output looks nice
std::sort(v_t_no.begin(), v_t_no.end(), sort_by_name);

// loop through and split
for(vector<transition_no>::iterator it = v_t_no.begin(); it != v_t_no.end(); it++)
{
    counter++;

    // lower / upper A10 str
    string s_lo(it->level_lower_a10);
    string s_up(it->level_upper_a10);

    // get lower and upper L
    char lo = it->level_lower_a10[8];
    char up = it->level_upper_a10[8];

    // check if we have a singlet
    if('1' == it->multiplet_type)
    {
        // J = L
        s_lo[6] = L_value(lo);
        s_up[6] = L_value(up);

        // cout << s_lo << s_up << " 1 1 " << it->f_ik << endl;
        out_lines << s_lo << s_up << " 1 1 " << it->f_ik << endl;
        continue;
    }

    // else get stuff from hash map
    multiplet_type_fs_map fs_map = multiplet[it->multiplet_type][make_pair(lo,up)];

    for(multiplet_type_fs_map::iterator fs_it = fs_map.begin(); fs_it != fs_map.end(); fs_it++)
    {
        // get J
        s_lo[6] = fs_it->first.first;
        s_up[6] = fs_it->first.second;

        // cout << s_lo << s_up << " 1 1 " << (it->f_ik * fs_it->second.d) << endl;
        out_lines << s_lo << s_up << " 1 1 " << (it->f_ik * fs_it->second.d) << endl;
    }
}

// end parsing file, show # warnings/errors
cout << counter << " transitions / lines were successfully parsed!" << endl;
out_log << counter << " transitions / lines were successfully parsed!"
cout << "End parsing file: " << filename << endl;  

out_log << "End parsing file: " << filename << endl;  

// close all output streams  
out_lines.close();  
out_log.close();  
return;
}

// file could not be opened  
cout << "File not found: " << filename << endl;  
// return so code below wont be executed  
return;
}

// done  
cOut << "done splitting" << endl;  
return;

} //===================================================================================  
// end FS_merge  
} //===================================================================================  

// parses file, input should be a SYN file  
void FS_merge(string filename)
{
  // start  
cOut << "Attempting to parse file: " << filename << endl;

  // read in stream, line buffer  
  ifstream input;  
  string line;  
  transition_fs t_fs;  

  // number of warnings, errors  
  warnings = 0;  
  errors = 0;  

  // number of transitions parsed  
  int counter = 0;  

  // vector with all the lines and all the newly calculated stuff  
  vector<transition_fs> v_t_fs;  
  vector<transition_no> v_t;  

  // open file and read line by line  
  input.open(filename.c_str());  
  if(input.is_open())
  {
    // open output/log files  
    out_lines.open((filename+"_out.l").c_str());  
    out_red.open((filename+"_out.all").c_str());  
    out_log.open((filename+"_out.info").c_str());  

    // set precision / scientific  
    out_lines << std::setprecision(4) << scientific;  
  }
out_red << std::setprecision(4) << scientific;
out_log << std::setprecision(4) << scientific;

// log
out_log << "Attempting to parse file: " << filename << endl;

// read in stuff
while(getline(input,line))
{
    // get line size
    int size_line = line.size();
    // skip blank lines (= size 0)
    if(0 == size_line)
        continue;
    // skip comments (dots)
    if(line[0] == '.')
        continue;
    // skip short lines (should be 20 chars for lower+upper then " 3 3 " + another like 7 for the f-value)
    if(size_line < 25)
    {
        // log short lines
        warnings++;
        out_log << "WARNING: line too short: " << size_line << endl << line;
        // skip to next line
        continue;
    }
    // read in stuff
    strcpy(t_fs.level_lower,line.substr(0,6).c_str());
    strcpy(t_fs.level_upper,line.substr(10,6).c_str());
    t_fs.J_lower = line[6];
    t_fs.J_upper = line[16];
    strcpy(t_fs.term_lower,line.substr(7,3).c_str());
    strcpy(t_fs.term_upper,line.substr(17,3).c_str());

    // TODO: check some syn files if 25 is ok with most files
    t_fs.f_ik = str_to_double(line.substr(25));

    // check if both are of the same multiplet
    if(t_fs.term_lower[0] != t_fs.term_upper[0])
    {
        // lower and upper term are not of the same multiplet
        errors++;
        cout << "ERROR: lower and upper term are not of the same multiplet:"
             << endl;
        log_transition_fs(cout,t_fs);
        out_log << "ERROR: lower and upper term are not of the same multiplet:" << endl;
        log_transition_fs(out_log,t_fs);
        // skip to next line
        continue;
    }
    // are the same, so save the multi type
    t_fs.multiplet_type = t_fs.term_lower[0];

    // check angular momentum, J
if(! check_fs_L_J(t_fs))
{
    errors++;
    cout << "ERROR with angular momentum or J" << endl;
    log_transition_fs(cout, t_fs);
    out_log << "ERROR with angular momentum or J" << endl;
    log_transition_fs(out_log, t_fs);
}
else
{
    // everything checks out, add to vector
    v_t_fs.push_back(t_fs);
}
// end while getline(input, line)

// close input
input.close();

// all lines are read in, compute the new f_ik and write output
for(vector<transition_fs>::iterator it = v_t_fs.begin(); it != v_t_fs.end(); it++)
{
    // calculate new f value and add to vector
    v_t.push_back(calculate_f(*it));
}

// sort the vector by name, so the output looks nice
std::sort(v_t.begin(), v_t.end(), sort_by_name);

// temp vector to store, sort
vector<transition_no> v_t_temp;

// loop through v_t until everything is processed
while(v_t.begin() != v_t.end())
{
    // get first item from the vector
    transition_no t_temp = v_t.front();
    v_t.erase(v_t.begin());
    // add it to temporary vector
    v_t_temp.push_back(t_temp);
    // get all the same transitions
    for(vector<transition_no>::iterator it = v_t.begin(); it != v_t.end(); it++)
    {
        // check if a match
        if(! strcmp(t_temp.level_lower_a10, it->level_lower_a10) && ! strcmp(t_temp.level_upper_a10, it->level_upper_a10))
        {
            // if yes, append to temp
            v_t_temp.push_back(*it);
            // delete from old vec, yields new iterator
            it = v_t.erase(it);
        }
    }
    else
    {
        // no match, next item
        it++;
    }
}
if(v_t_temp.size() != v_t_temp.front().f_total)
{
    // number of fs lines does not match
    warnings++;

    cout << "WARNING: number of possible duplicate lines does not match
to multiplet (is " << v_t_temp.size() << ", should be " << v_t_temp.front().f_total << ")" << endl;
    out_log << "WARNING: number of possible duplicate lines does not
match to multiplet (is " << v_t_temp.size() << ", should be " << v_t_temp.front().f_total << ")" << endl;

    for(vector<transition_no>::iterator it = v_t_temp.begin(); it !=
    v_t_temp.end(); it++)
    {
        log_transition(cout,*it);
        log_transition(out_log,*it);
    }
}

// sort temp vector by index
std::sort(v_t_temp.begin(), v_t_temp.end(), sort_by_index);

// write output
bool out_lines_only_once = true;
for(vector<transition_no>::iterator it = v_t_temp.begin(); it !=
    v_t_temp.end(); it++)
{
    counter++;

    // only write the first possible new f value in the actual output
    file
    if(out_lines_only_once)
    {
        out_lines_only_once = false;
        out_lines << it->level_lower_a10 << it->level_upper_a10 << " 1 1
" << it->f_ik << endl;
    }

    // info in red file
    out_red << "............................................" << 

    out_red << ". multiplet(" << it->multiplet_type << "),
calculated f values: " << v_t_temp.size() << "/" << it->
f_total << endl;
    out_red << ". some statistics to be implemented" << endl;
}

// write everything in redundant file
out_red << it->level_lower_a10 << it->level_upper_a10 << " 1 1 "
    << it->f_ik << endl;
}

// clear temp
v_t_temp.clear();

// end parsing file, show # warnings/errors
cout << counter << " transitions / lines were successfully parsed!" << 
    endl;
out_log << counter << " transitions / lines were successfully parsed!" 
    << endl;

cout << "End parsing file: " << filename << endl << warnings << " warning

(s) and " << errors << " error(s) occurred" << endl << endl;
out_log << "End parsing file: " << filename << endl << warnings << " warning(s) and " << errors << " error(s) occurred" << endl << endl;

// close all output streams
out_lines.close();
out_red.close();
out_log.close();
return;
else
{
    // file could not be opened
cout << "File not found: " << filename << endl;
    // return so code below wont be executed
    return;
}
//===============================================================================
// end FS_merge
//===============================================================================
// convert str to double
//===============================================================================
double str_to_double(string s)
{
    // remove brackets if necessary
    //if( (s[0] == '(' || s[0] == '[') && (s[s.size()-1] == ')' || s[s.size()-1] == ']')
    //{
    // // first and last are brackets
    // return str_to_double(s.substr(1,s.size()-2));
    //}
    // actual conversion
    double d;
    istringstream convert(s);
    convert >> d;
    return d;
} //===============================================================================
// end str to double
//===============================================================================
// write all the values of t_fs in on of the output streams
//===============================================================================
void log_transition_fs(ostream& s, transition_fs t)
{
    s << t.level_lower << t.J_lower << t.term_lower << "||" << t.level_upper << t.J_upper << t.term_upper << "||" << t.f_ik << "||" << t.multiplet_type << endl;
    return;
}
void log_transition(ostream& s, transition_no t) {
    s << t.level_lower_a10 << "|| " << t.level_upper_a10 << "|| " << t.f_ik << "|| " << t.multiplet_type << "|| " << t.f_index << "/" << t.f_total << endl;
    return;
}

bool check_nofs_L(transition_no t_no) {
    if ('1' == t_no.multiplet_type) return true;
    char lo = t_no.level_lower_a10[8];
    char up = t_no.level_upper_a10[8];
    if (multiplet.count(t_no.multiplet_type))
        if (multiplet[t_no.multiplet_type].count(make_pair(lo, up)))
            return true;
    return false;
}

bool check_fs_L_J(transition_fs t_fs) {
    if ('1' == t_fs.multiplet_type) return true;
}

bool check_fs_L_J(transition_fs t_fs) {
    // check if FS is ok
    // ==============================================================
    inline bool check_fs_L_J(transition_fs t_fs) {
        // check if we have a singlet
        if ('1' == t_fs.multiplet_type) return true;
        // ==============================================================
// get lower and upper L
char lo = t_fs.term_lower[1];
char up = t_fs.term_upper[1];

// return true if in hashmap
if(multiplet.count(t_fs.multiplet_type))
  if(multiplet[t_fs.multiplet_type].count(make_pair(lo,up)))
    if(multiplet[t_fs.multiplet_type][make_pair(lo,up)].count(make_pair(t_fs.J_lower,t_fs.J_upper)))
      return true;

// else return false
return false;

// ==============================================================
// end check
// ==============================================================

// calculate the new f value accordingly
// ==============================================================
transition_no calculate_f(transition_fs t)
{
  transition_no t_new;

  // make A10 string and new multiplet type
  strcpy(t_new.level_lower_a10,(string(t.level_lower) + " " + t.term_lower).c_str());
  strcpy(t_new.level_upper_a10,(string(t.level_upper) + " " + t.term_upper).c_str());
  t_new.multiplet_type = t.multiplet_type;

  // check if we have a singlet
  if('1' == t.multiplet_type)
  {
    t_new.f_ik = t.f_ik;
    t_new.f_index = 1;
    t_new.f_total = 1;
    return t_new;
  }

  // else use the multiplet map to get the stuff
  char lo = t.term_lower[1];
  char up = t.term_upper[1];
  new_f_value ff = multiplet[t.multiplet_type][make_pair(lo,up)][make_pair(t.J_lower,t.J_upper)];

  // calculate new f value and return max number of duplicates and index
  t_new.f_ik = (t.f_ik / ff.d);
  t_new.f_index = ff.index;
  t_new.f_total = ff.total;

  // return it
  return t_new;
}

// ==============================================================
// end calculate_f
// ==============================================================
```cpp
// init the hash map

// values taken from "multiplet" (Rauch 2008)

doublet F -> ?
multiplet ['2'][make_pair ('S', 'P')][make_pair ('1', '1')] = 3.3333E-01,1,2;
multiplet ['2'][make_pair ('S', 'P')][make_pair ('1', '3')] = 6.6667E-01,2,2;

// doublet P -> ?
multiplet ['2'][make_pair ('P', 'S')][make_pair ('1', '1')] = 1.0000E+00,1,2;
multiplet ['2'][make_pair ('P', 'S')][make_pair ('3', '3')] = 3.3333E-01,2,2;
multiplet ['2'][make_pair ('P', 'P')][make_pair ('1', '1')] = 6.6667E-01,2,2;
multiplet ['2'][make_pair ('P', 'P')][make_pair ('3', '3')] = 8.3333E-01,4,4;
multiplet ['2'][make_pair ('P', 'D')][make_pair ('1', '1')] = 9.3333E-01,1,3;
multiplet ['2'][make_pair ('P', 'D')][make_pair ('3', '3')] = 3.5675E-02,3,4;
multiplet ['2'][make_pair ('P', 'D')][make_pair ('5', '5')] = 9.3266E-01,4,4;
multiplet ['2'][make_pair ('P', 'F')][make_pair ('5', '5')] = 8.9982E-01,3,3;

// doublet D -> ?
multiplet ['2'][make_pair ('D', 'D')][make_pair ('3', '1')] = 8.3383E-01,1,3;
multiplet ['2'][make_pair ('D', 'D')][make_pair ('3', '3')] = 1.6666E-01,2,2;
multiplet ['2'][make_pair ('D', 'D')][make_pair ('5', '3')] = 6.7152E-02,3,4;
multiplet ['2'][make_pair ('D', 'D')][make_pair ('5', '5')] = 9.3266E-01,4,4;
multiplet ['2'][make_pair ('D', 'F')][make_pair ('5', '5')] = 1.0000E+00,1,3;
multiplet ['2'][make_pair ('D', 'F')][make_pair ('5', '7')] = 4.7619E-02,2,3;
multiplet ['2'][make_pair ('D', 'F')][make_pair ('7', '7')] = 9.5238E-01,3,3;

// doublet S -> ?
multiplet ['2'][make_pair ('S', 'P')][make_pair ('1', '1')] = 9.9955E-01,1,2;
multiplet ['2'][make_pair ('S', 'P')][make_pair ('1', '3')] = 8.3383E-01,1,3;
multiplet ['2'][make_pair ('S', 'P')][make_pair ('3', '3')] = 6.6667E-01,2,2;
multiplet ['2'][make_pair ('S', 'P')][make_pair ('5', '5')] = 9.6419E-01,4,4;
multiplet ['2'][make_pair ('S', 'D')][make_pair ('3', '3')] = 9.9944E-01,1,3;
multiplet ['2'][make_pair ('S', 'D')][make_pair ('7', '7')] = 2.8194E-02,2,3;
multiplet ['2'][make_pair ('S', 'D')][make_pair ('7', '9')] = 9.7222E-01,3,3;
```

// doublet G -> ?
multiplet ['^2'] [make_pair ('G', 'F')] [make_pair ('7', '5')] = {3.3333 E -01 ,2 ,3};
multiplet ['^2'] [make_pair ('G', 'F')] [make_pair ('7', '7')] = {3.6250 E -02 ,2, 3};
multiplet ['^2'] [make_pair ('G', 'F')] [make_pair ('9', '7')] = {1.0000E+00, 3, 3};
multiplet ['^2'] [make_pair ('G', 'G')] [make_pair ('7', '7')] = {9.7162E-01, 1, 1, 4};
multiplet ['^2'] [make_pair ('G', 'G')] [make_pair ('7', '9')] = {2.8110E-02, 2, 4};
multiplet ['^2'] [make_pair ('G', 'G')] [make_pair ('9', '7')] = {2.2488E-02, 3, 4};
multiplet ['^2'] [make_pair ('G', 'G')] [make_pair ('9', '9')] = {9.7733E-01, 1, 1, 4};
multiplet ['^2'] [make_pair ('G', 'H')] [make_pair ('7', '9')] = {9.9964E-01, 1, 1, 3};
multiplet ['^2'] [make_pair ('G', 'H')] [make_pair ('9', '9')] = {1.8648E-02, 2, 3};
multiplet ['^2'] [make_pair ('G', 'H')] [make_pair ('9', 'E')] = {9.8146E-01, 1, 3};

// doublet H -> ?
multiplet ['^2'] [make_pair ('H', 'G')] [make_pair ('9', '7')] = {9.7764E-01, 1, 1, 3};
multiplet ['^2'] [make_pair ('H', 'G')] [make_pair ('9', '9')] = {9.9964E-01, 1, 3};
multiplet ['^2'] [make_pair ('H', 'H')] [make_pair ('9', '9')] = {9.8498E-01, 1, 1, 4};
multiplet ['^2'] [make_pair ('H', 'H')] [make_pair ('9', 'E')] = {1.5401E-02, 2, 3};
multiplet ['^2'] [make_pair ('H', 'H')] [make_pair ('E', '9')] = {1.2834E-02, 3, 4};
multiplet ['^2'] [make_pair ('H', 'H')] [make_pair ('E', 'E')] = {9.8726E-01, 1, 4, 4};
multiplet ['^2'] [make_pair ('H', 'T')] [make_pair ('9', 'E')] = {9.9896E-01, 1, 1, 3};
multiplet ['^2'] [make_pair ('H', 'T')] [make_pair ('E', '9')] = {1.2834E-02, 2, 3};
multiplet ['^2'] [make_pair ('H', 'T')] [make_pair ('E', 'E')] = {9.8726E-01, 1, 3, 3};

// doublet I -> ?
multiplet ['^2'] [make_pair ('I', 'H')] [make_pair ('E', '9')] = {9.8474E-01, 1, 1, 3};
multiplet ['^2'] [make_pair ('I', 'H')] [make_pair ('E', 'E')] = {1.5168E-02, 2, 3};
multiplet ['^2'] [make_pair ('I', 'H')] [make_pair ('T', 'E')] = {1.0001E+00, 3, 3};
multiplet ['^2'] [make_pair ('I', 'I')] [make_pair ('E', '9')] = {9.8758E-01, 1, 1, 4};
multiplet ['^2'] [make_pair ('I', 'I')] [make_pair ('E', 'E')] = {1.2691E-02, 2, 4};
multiplet ['^2'] [make_pair ('I', 'I')] [make_pair ('T', 'E')] = {1.0878E-02, 3, 4};
multiplet ['^2'] [make_pair ('I', 'I')] [make_pair ('T', 'T')] = {9.8889E-01, 1, 4, 4};
multiplet ['^2'] [make_pair ('I', 'K')] [make_pair ('E', 'T')] = {1.0000E+00, 1, 3};
multiplet ['^2'] [make_pair ('I', 'K')] [make_pair ('T', 'E')] = {9.5086E-03, 2, 3};
multiplet ['^2'] [make_pair ('I', 'K')] [make_pair ('T', 'T')] = {9.9048E-01, 3, 3};

// doublet K -> ?
multiplet ['^2'] [make_pair ('K', 'I')] [make_pair ('E', '9')] = {9.8903E-01, 1, 1, 3};
multiplet ['^2'] [make_pair ('K', 'I')] [make_pair ('E', 'E')] = {1.0971E-02, 2, 3};
multiplet ['^2'] [make_pair ('K', 'I')] [make_pair ('F', 'T')] = {1.0000E+00, 3, 3};

// triplets

// triplet S -> ?
multiplet ['^3'] [make_pair ('S', 'P')] [make_pair ('I', '0')] = {1.1111E-01, 1, 1, 3};
multiplet ['^3'] [make_pair ('S', 'P')] [make_pair ('I', '1')] = {3.3333E-01, 2, 3};
multiplet ['^3'] [make_pair ('S', 'P')] [make_pair ('I', '2')] = {5.5556E-01, 3, 3};

// triplet P -> ?
multiplet ['^3'] [make_pair ('P', 'S')] [make_pair ('0', '1')] = {1.0000E+00, 1, 3};
multiplet ['3'] [make_pair ('P', 'S')] [make_pair ('1', '1')] = {1.0000E+00, 2, 3};
multiplet ['3'] [make_pair ('P', 'S')] [make_pair ('2', '1')] = {1.0000E+00, 3, 3};

multiplet ['3'] [make_pair ('P', 'P')] [make_pair ('0', '0')] = {0.0000E+00, 1, 1};
multiplet ['3'] [make_pair ('P', 'P')] [make_pair ('0', '1')] = {1.0013E+00, 2, 2};
multiplet ['3'] [make_pair ('P', 'P')] [make_pair ('1', '0')] = {3.3375E-01, 3, 3};
multiplet ['3'] [make_pair ('P', 'P')] [make_pair ('1', '1')] = {2.5000E-01, 4, 4};
multiplet ['3'] [make_pair ('P', 'P')] [make_pair ('2', '0')] = {4.1625E-01, 5, 5};
multiplet ['3'] [make_pair ('P', 'P')] [make_pair ('2', '1')] = {2.9475E-01, 6, 6};
multiplet ['3'] [make_pair ('P', 'P')] [make_pair ('2', '2')] = {7.5000E-01, 7, 7};

multiplet ['3'] [make_pair ('P', 'D')] [make_pair ('0', '1')] = {9.9907E-01, 1, 6};
multiplet ['3'] [make_pair ('P', 'D')] [make_pair ('1', '1')] = {2.5047E-01, 2, 6};
multiplet ['3'] [make_pair ('P', 'D')] [make_pair ('1', '2')] = {4.1301E-01, 3, 6};
multiplet ['3'] [make_pair ('P', 'D')] [make_pair ('2', '1')] = {2.9475E-01, 4, 6};
multiplet ['3'] [make_pair ('P', 'D')] [make_pair ('2', '2')] = {7.5000E-01, 5, 6};
multiplet ['3'] [make_pair ('P', 'D')] [make_pair ('2', '3')] = {8.3955E-01, 6, 6};

// triplet D -> ?
multiplet ['3'] [make_pair ('D', 'P')] [make_pair ('1', '0')] = {5.5504E-01, 1, 6};
multiplet ['3'] [make_pair ('D', 'P')] [make_pair ('1', '1')] = {4.1744E-01, 2, 6};
multiplet ['3'] [make_pair ('D', 'P')] [make_pair ('1', '2')] = {7.5000E-01, 3, 6};
multiplet ['3'] [make_pair ('D', 'P')] [make_pair ('2', '1')] = {1.6791E-02, 4, 6};
multiplet ['3'] [make_pair ('D', 'P')] [make_pair ('2', '2')] = {1.7890E-01, 5, 6};
multiplet ['3'] [make_pair ('D', 'P')] [make_pair ('2', '3')] = {7.7736E-01, 6, 6};
multiplet ['3'] [make_pair ('D', 'P')] [make_pair ('3', '2')] = {9.9957E-01, 7, 7};

multiplet ['3'] [make_pair ('D', 'D')] [make_pair ('1', '1')] = {7.5041E-01, 1, 7};
multiplet ['3'] [make_pair ('D', 'D')] [make_pair ('1', '2')] = {2.5083E-01, 2, 7};
multiplet ['3'] [make_pair ('D', 'D')] [make_pair ('2', '1')] = {1.5050E-01, 3, 7};
multiplet ['3'] [make_pair ('D', 'D')] [make_pair ('2', '2')] = {6.9403E-01, 4, 7};
multiplet ['3'] [make_pair ('D', 'D')] [make_pair ('2', '3')] = {1.5647E-01, 5, 7};
multiplet ['3'] [make_pair ('D', 'D')] [make_pair ('3', '2')] = {1.1056E-01, 6, 7};
multiplet ['3'] [make_pair ('D', 'D')] [make_pair ('3', '3')] = {8.8842E-01, 7, 7};

multiplet ['3'] [make_pair ('D', 'F')] [make_pair ('1', '2')] = {9.9957E-01, 1, 6};
multiplet ['3'] [make_pair ('D', 'F')] [make_pair ('2', '2')] = {1.3013E-01, 2, 6};
multiplet ['3'] [make_pair ('D', 'F')] [make_pair ('2', '3')] = {8.8741E-01, 3, 6};
multiplet ['3'] [make_pair ('D', 'F')] [make_pair ('3', '2')] = {1.8346E-03, 4, 6};
multiplet ['3'] [make_pair ('D', 'F')] [make_pair ('3', '3')] = {8.0724E-02, 5, 6};
multiplet ['3'] [make_pair ('D', 'F')] [make_pair ('3', '4')] = {9.1732E-01, 6, 6};

// triplet F -> ?
multiplet ['3'] [make_pair ('F', 'D')] [make_pair ('2', '1')] = {8.3964E-01, 1, 6};
multiplet ['3'] [make_pair ('F', 'D')] [make_pair ('2', '2')] = {1.5882E-01, 2, 6};
multiplet ['3'] [make_pair ('F', 'D')] [make_pair ('2', '3')] = {3.5959E-03, 3, 6};
multiplet ['3'] [make_pair ('F', 'D')] [make_pair ('3', '2')] = {6.8741E-01, 4, 6};
multiplet ['3'] [make_pair ('F', 'D')] [make_pair ('3', '3')] = {1.3013E-01, 5, 6};
multiplet ['3'] [make_pair ('F', 'D')] [make_pair ('4', '3')] = {9.9886E-01, 6, 6};
multiplet ['3'] [make_pair ('F', 'F')] [make_pair ('2', '2')] = {8.8830E-01, 1, 7};
multiplet ['3'] [make_pair ('F', 'F')] [make_pair ('2', '3')] = {1.0977E-01, 2, 7};
multiplet ['3'] [make_pair ('F', 'F')] [make_pair ('3', '2')] = {7.9614E-02, 3, 7};
multiplet ['3'] [make_pair ('F', 'F')] [make_pair ('3', '3')] = {8.3957E-01, 4, 7};
multiplet ['3'] [make_pair ('F', 'F')] [make_pair ('3', '4')] = {8.0820E-02, 5, 7};
multiplet ['3'] [make_pair ('F', 'F')] [make_pair ('4', '3')] = {6.2860E-02, 6, 7};
multiplet ['3'] [make_pair ('F', 'F')] [make_pair ('4', '4')] = {9.3821E-01, 7, 7};

multiplet ['3'] [make_pair ('F', 'G')] [make_pair ('2', '3')] = {9.9951E-01, 1, 6};
multiplet ['3'] [make_pair ('F', 'G')] [make_pair ('3', '3')] = {6.2347E-02, 2, 6};
multiplet ['3'] [make_pair ('F', 'G')] [make_pair ('3', '4')] = {9.3765E-01, 3, 6};
multiplet ['3'] [make_pair ('F', 'G')] [make_pair ('4', '3')] = {9.5083E-04, 4, 6};
multiplet ['3'] [make_pair ('F', 'G')] [make_pair ('4', '4')] = {4.8492E-02, 5, 6};
multiplet ['3'] [make_pair ('F', 'G')] [make_pair ('4', '5')] = {9.5083E-01, 6, 6};
// triplet G -> 
multiplet[3'] [make_pair ('G', 'F')] [make_pair ('3', '2')] = (9.1792E-01, 1, 6);
multiplet[3'] [make_pair ('G', 'F')] [make_pair ('3', '3')] = (8.0161E-02, 2, 6);
multiplet[3'] [make_pair ('G', 'F')] [make_pair ('3', '4')] = (1.5718E-03, 3, 6);
multiplet[3'] [make_pair ('G', 'F')] [make_pair ('4', '3')] = (9.3765E-01, 1, 6);
multiplet[3'] [make_pair ('G', 'F')] [make_pair ('4', '4')] = (6.2347E-02, 2, 6);
multiplet[3'] [make_pair ('G', 'F')] [make_pair ('5', '4')] = (1.0002E+00, 6, 6);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('3', '3')] = (9.3712E-01, 1, 7);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('3', '4')] = (4.8122E-02, 2, 7);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('3', '5')] = (9.0258E-01, 1, 7);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('4', '3')] = (4.9296E-02, 2, 5);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('4', '4')] = (4.0336E-02, 2, 6);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('4', '5')] = (9.6031E-01, 1, 7);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('5', '4')] = (3.2878E-02, 2, 7);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('5', '5')] = (9.9988E-01, 1, 6);
multiplet[3'] [make_pair ('G', 'G')] [make_pair ('5', '6')] = (9.9992E-01, 6, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('3', '3')] = (9.7222E-01, 7, 7);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('3', '4')] = (3.9321E-02, 2, 7);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('3', '5')] = (9.3765E-01, 4, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('4', '3')] = (9.3412E-01, 4, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('4', '4')] = (9.9992E-01, 6, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('4', '5')] = (9.7262E-01, 4, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('5', '4')] = (9.9992E-01, 6, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('5', '5')] = (9.7262E-01, 4, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('5', '6')] = (9.7262E-01, 4, 6);
multiplet[3'] [make_pair ('G', 'I')] [make_pair ('5', '7')] = (9.7602E-01, 6, 6);

// triplet H -> 
multiplet[3'] [make_pair ('H', 'G')] [make_pair ('4', '3')] = (9.9988E-01, 1, 6);
multiplet[3'] [make_pair ('H', 'G')] [make_pair ('4', '4')] = (9.0148E-02, 2, 6);
multiplet[3'] [make_pair ('H', 'G')] [make_pair ('4', '5')] = (9.5970E-01, 1, 3, 6);
multiplet[3'] [make_pair ('H', 'G')] [make_pair ('5', '4')] = (2.9010E-04, 4, 6);
multiplet[3'] [make_pair ('H', 'G')] [make_pair ('5', '5')] = (3.2878E-02, 5, 6);
multiplet[3'] [make_pair ('H', 'G')] [make_pair ('5', '6')] = (9.6700E-01, 6, 6);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('3', '4')] = (9.6056E-01, 1, 7);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('3', '5')] = (9.3321E-02, 2, 7);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('3', '6')] = (9.5970E-01, 1, 4, 6);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('4', '5')] = (9.0148E-02, 2, 6);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('4', '6')] = (1.0001E+00, 6, 6);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('5', '5')] = (9.6056E-01, 1, 7);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('5', '6')] = (9.3321E-02, 2, 7);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('6', '5')] = (9.3321E-02, 2, 5, 7);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('6', '6')] = (2.8194E-02, 6, 7);
multiplet[3'] [make_pair ('H', 'H')] [make_pair ('6', '7')] = (9.7222E-01, 7, 7);
multiplet[3'] [make_pair ('H', 'I')] [make_pair ('4', '5')] = (9.9979E-01, 1, 6);
multiplet[3'] [make_pair ('H', 'I')] [make_pair ('5', '5')] = (2.7690E-02, 2, 6);
multiplet[3'] [make_pair ('H', 'I')] [make_pair ('5', '6')] = (9.7262E-01, 3, 6);
multiplet[3'] [make_pair ('H', 'I')] [make_pair ('6', '5')] = (2.3075E-04, 4, 6);
multiplet[3'] [make_pair ('H', 'I')] [make_pair ('6', '6')] = (2.3430E-02, 5, 6);
multiplet[3'] [make_pair ('H', 'I')] [make_pair ('6', '7')] = (9.7626E-01, 6, 6);

// triplet I -> 
multiplet[3'] [make_pair ('I', 'H')] [make_pair ('5', '4')] = (9.6674E-01, 1, 6);
multiplet[3'] [make_pair ('I', 'H')] [make_pair ('5', '5')] = (3.2725E-02, 2, 6);
multiplet[3'] [make_pair ('I', 'H')] [make_pair ('5', '6')] = (2.7271E-04, 3, 6);
multiplet[3'] [make_pair ('I', 'H')] [make_pair ('6', '5')] = (9.7262E-01, 4, 6);
multiplet[3'] [make_pair ('I', 'H')] [make_pair ('6', '6')] = (2.7690E-02, 5, 6);
multiplet[3'] [make_pair ('I', 'H')] [make_pair ('7', '6')] = (9.9992E-01, 6, 6);
multiplet[3'] [make_pair ('I', 'I')] [make_pair ('5', '5')] = (9.7216E-01, 1, 7);
multiplet[3'] [make_pair ('I', 'I')] [make_pair ('5', '6')] = (2.8043E-02, 2, 7);
multiplet[3'] [make_pair ('I', 'I')] [make_pair ('6', '5')] = (2.3729E-02, 3, 7);
multiplet[3'] [make_pair ('I', 'I')] [make_pair ('6', '6')] = (9.5254E-01, 4, 7);
multiplet[3'] [make_pair ('I', 'I')] [make_pair ('7', '6')] = (2.3729E-02, 5, 7);
multiplet[3'] [make_pair ('I', 'I')] [make_pair ('7', '7')] = (2.0565E-02, 6, 7);
multiplet[3'] [make_pair ('I', 'I')] [make_pair ('7', '7')] = (9.7928E-01, 7, 7);
// quartets

// quartet P -> ?

multiplet[4][make_pair('P', 'P')][make_pair('1', '1')] = {1.6585E-01, 1, 1, 7};
multiplet[4][make_pair('P', 'P')][make_pair('1', '3')] = {4.1672E-01, 1, 2, 7};
multiplet[4][make_pair('P', 'P')][make_pair('3', '1')] = {8.3345E-01, 1, 3, 7};
multiplet[4][make_pair('P', 'P')][make_pair('3', '3')] = {1.3331E-01, 1, 4, 7};
multiplet[4][make_pair('P', 'P')][make_pair('3', '5')] = {3.0021E-01, 1, 5, 7};
multiplet[4][make_pair('P', 'P')][make_pair('5', '3')] = {4.5031E-01, 1, 6, 7};
multiplet[4][make_pair('P', 'P')][make_pair('5', '5')] = {6.9979E-01, 1, 7, 7};

multiplet[4][make_pair('P', 'D')][make_pair('1', '1')] = {2.3328E-01, 6, 1, 10};
multiplet[4][make_pair('P', 'D')][make_pair('3', '1')] = {1.1433E+00, 6, 2, 9};
multiplet[4][make_pair('P', 'D')][make_pair('3', '3')] = {8.0000E-01, 6, 3, 8};
multiplet[4][make_pair('P', 'D')][make_pair('3', '5')] = {9.5238E-03, 6, 4, 9};
multiplet[4][make_pair('P', 'D')][make_pair('5', '3')] = {5.3400E-01, 6, 5, 8};
multiplet[4][make_pair('P', 'D')][make_pair('5', '5')] = {5.0086E-01, 6, 6, 10};

// quartet D -> ?

multiplet[4][make_pair('D', 'P')][make_pair('1', '1')] = {2.4333E-01, 5, 1, 9};
multiplet[4][make_pair('D', 'P')][make_pair('3', '1')] = {3.2040E-01, 5, 2, 8};
multiplet[4][make_pair('D', 'P')][make_pair('3', '3')] = {1.4237E-01, 5, 3, 7};
multiplet[4][make_pair('D', 'P')][make_pair('3', '5')] = {5.7633E-01, 5, 4, 9};
multiplet[4][make_pair('D', 'P')][make_pair('5', '3')] = {5.0086E-01, 5, 5, 8};
multiplet[4][make_pair('D', 'P')][make_pair('5', '5')] = {5.3571E-03, 5, 6, 9};

multiplet[4][make_pair('D', 'F')][make_pair('1', '1')] = {5.0086E-01, 7, 1, 9};
multiplet[4][make_pair('D', 'F')][make_pair('3', '1')] = {4.8000E-02, 7, 2, 8};
multiplet[4][make_pair('D', 'F')][make_pair('3', '3')] = {2.8000E-01, 7, 3, 7};
multiplet[4][make_pair('D', 'F')][make_pair('3', '5')] = {1.3333E+00, 7, 4, 8};
multiplet[4][make_pair('D', 'F')][make_pair('5', '3')] = {2.4333E-01, 7, 5, 8};
multiplet[4][make_pair('D', 'F')][make_pair('5', '5')] = {1.1433E+00, 7, 6, 9};
multiplet [\{'F', 'D'\}] [\{make_pair('7', '7')\}] = {1.4250E-01, 8, 9};
multiplet [\{'F', 'D'\}] [\{make_pair('9', '7')\}] = {1.2500E+00, 9, 9};
multiplet [\{'F', 'F'\}] [\{make_pair('3', '3')\}] = {7.9993E-01, 11, 10};
multiplet [\{'F', 'F'\}] [\{make_pair('5', '3')\}] = {1.3294E-01, 3, 10};
multiplet [\{'F', 'F'\}] [\{make_pair('5', '7')\}] = {1.7878E-01, 5, 10};
multiplet [\{'F', 'F'\}] [\{make_pair('7', '7')\}] = {1.3409E-01, 6, 10};
multiplet [\{'F', 'F'\}] [\{make_pair('7', '9')\}] = {7.6212E-01, 7, 10};
multiplet [\{'F', 'F'\}] [\{make_pair('9', '7')\}] = {8.3432E-02, 9, 10};
multiplet [\{'F', 'F'\}] [\{make_pair('9', '9')\}] = {9.1683E-01, 10, 10};
multiplet [\{'F', 'G'\}] [\{make_pair('3', '7')\}] = {1.0007E+00, 1, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('5', '7')\}] = {8.9259E-01, 3, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('7', '7')\}] = {3.4988E-03, 4, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('7', '9')\}] = {6.8762E-01, 4, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('9', '7')\}] = {7.0995E-02, 5, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('9', '9')\}] = {9.5423E-01, 9, 9};
multiplet [\{'F', 'H'\}] [\{make_pair('7', '7')\}] = {1.3795E-01, 2, 9};
multiplet [\{'F', 'H'\}] [\{make_pair('9', '7')\}] = {8.1264E-01, 4, 10};
multiplet [\{'F', 'H'\}] [\{make_pair('9', '9')\}] = {6.8762E-01, 4, 10};
multiplet [\{'F', 'H'\}] [\{make_pair('9', 'E')\}] = {6.4379E-02, 8, 9};
multiplet [\{'F', 'H'\}] [\{make_pair('9', 'E')\}] = {9.3302E-01, 10, 10};
multiplet [\{'F', 'G'\}] [\{make_pair('5', '7')\}] = {1.3409E-01, 6, 10};
multiplet [\{'F', 'G'\}] [\{make_pair('5', '7')\}] = {1.3995E-03, 3, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('9', '7')\}] = {8.9259E-01, 3, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('9', '7')\}] = {1.0429E-01, 8, 9};
multiplet [\{'F', 'G'\}] [\{make_pair('9', '9')\}] = {8.3432E-02, 9, 10};
multiplet [\{'F', 'G'\}] [\{make_pair('9', '9')\}] = {9.1683E-01, 10, 10};

// quartet G -> ?
multiplet [\{'G', 'F'\}] [\{make_pair('5', '7')\}] = {1.2866E+00, 1, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('5', '7')\}] = {1.3795E-01, 2, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('7', '7')\}] = {4.4988E-03, 3, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('7', '7')\}] = {1.1476E+00, 4, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('7', '7')\}] = {1.3645E-01, 5, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('7', '9')\}] = {2.3992E-03, 6, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('9', '7')\}] = {1.1456E+00, 7, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('9', '9')\}] = {6.4379E-02, 8, 9};
multiplet [\{'G', 'F'\}] [\{make_pair('9', 'E')\}] = {9.3302E-01, 9, 9};
multiplet [\{'G', 'G'\}] [\{make_pair('5', '5')\}] = {8.9253E-01, 11, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('5', '5')\}] = {1.0778E-01, 2, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('7', '7')\}] = {8.0838E-02, 3, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('7', '7')\}] = {8.1264E-01, 4, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('7', '9')\}] = {1.0637E-01, 5, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('9', '7')\}] = {8.5093E-02, 6, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('9', '9')\}] = {8.4866E-01, 7, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('9', 'E')\}] = {5.4383E-02, 9, 10};
multiplet [\{'G', 'G'\}] [\{make_pair('E', 'E')\}] = {9.4548E-01, 10, 10};
multiplet [\{'G', 'H'\}] [\{make_pair('5', '7')\}] = {1.0000E+00, 1, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('7', '7')\}] = {6.7273E-02, 2, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('7', '9')\}] = {9.3324E-01, 3, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('9', '7')\}] = {1.1451E-03, 4, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('9', '9')\}] = {7.0995E-02, 5, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('9', 'E')\}] = {9.2751E-01, 6, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('E', 'E')\}] = {8.5881E-04, 7, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('E', 'E')\}] = {4.4894E-02, 8, 9};
multiplet [\{'G', 'H'\}] [\{make_pair('E', 'T')\}] = {9.5423E-01, 9, 9};

// quartet H -> ?
multiplet [\{'H', 'G'\}] [\{make_pair('7', '5')\}] = {1.2223E+00, 1, 9};
multiplet [\{'H', 'G'\}] [\{make_pair('7', '7')\}] = {8.2223E-02, 2, 9};
multiplet [\{'H', 'G'\}] [\{make_pair('7', '9')\}] = {1.3995E-03, 3, 9};
multiplet [\{'H', 'G'\}] [\{make_pair('9', '7')\}] = {1.1406E+00, 4, 9};
multiplet [\{'H', 'G'\}] [\{make_pair('9', '9')\}] = {8.6771E-02, 5, 9};
multiplet [\{'H', 'G'\}] [\{make_pair('9', 'E')\}] = {1.0497E-03, 6, 9};
multiplet['4'][make_pair('H', 'G')][make_pair('E', '9')] = {1.1336E+00, 7, 9};
multiplet['4'][make_pair('H', 'G')][make_pair('E', 'E')] = {5.4815E-02, 0, 9};
multiplet['4'][make_pair('H', 'G')][make_pair('T', 'E')] = {1.1663E+00, 0, 9};
multiplet['4'][make_pair('H', 'H')][make_pair('T', 'T')] = {1.1428E+00, 1, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('T', '9')] = {8.2366E-02, 2, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('9', '7')] = {6.5893E-02, 0, 3, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('9', '9')] = {8.4013E-02, 0, 4, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('9', 'E')] = {8.7308E-02, 0, 5, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('E', '9')] = {7.2757E-02, 0, 6, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('E', 'E')] = {1.0968E+00, 0, 7};
multiplet['4'][make_pair('H', 'H')][make_pair('E', 'T')] = {5.4911E-02, 0, 8, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('T', 'E')] = {4.7066E-02, 0, 9, 10};
multiplet['4'][make_pair('H', 'H')][make_pair('T', 'T')] = {1.1767E+00, 0, 10};
multiplet['4'][make_pair('H', 'I')][make_pair('7', '9')] = {9.9988E-01, 1, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('9', '9')] = {4.6018E-02, 2, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('9', 'E')] = {5.9420E-01, 0, 3, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('9', 'E')] = {5.6395E-04, 0, 4, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('E', 'E')] = {5.0758E-02, 0, 5, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('E', 'T')] = {9.4865E-01, 0, 6, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('T', 'E')] = {3.8671E-04, 0, 7, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('T', 'T')] = {2.8570E-02, 0, 8, 9};
multiplet['4'][make_pair('H', 'I')][make_pair('T', 'F')] = {9.6677E-01, 0, 9};

// quartet I -> ?

multiplet['4'][make_pair('I', 'H')][make_pair('9', '7')] = {1.1817E+00, 1, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('9', '9')] = {5.4385E-02, 2, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('9', 'E')] = {6.6648E-04, 0, 3, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('9', 'E')] = {1.1277E+00, 0, 4, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('E', 'E')] = {5.9983E-02, 0, 5, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('E', 'T')] = {4.5702E-02, 0, 6, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('T', 'E')] = {1.1210E+00, 0, 7, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('T', 'T')] = {3.8846E-02, 0, 8, 9};
multiplet['4'][make_pair('I', 'H')][make_pair('T', 'F')] = {9.6677E-01, 0, 9};
multiplet['4'][make_pair('I', 'I')][make_pair('9', '9')] = {9.4783E-01, 1, 10};
multiplet['4'][make_pair('I', 'I')][make_pair('9', 'E')] = {4.5096E-02, 2, 10};
multiplet['4'][make_pair('I', 'I')][make_pair('9', 'E')] = {3.7580E-02, 0, 3, 10};
multiplet['4'][make_pair('I', 'I')][make_pair('E', 'E')] = {9.1228E-01, 1, 10};
multiplet['4'][make_pair('I', 'I')][make_pair('E', 'T')] = {9.4865E-01, 0, 6, 9};
multiplet['4'][make_pair('I', 'I')][make_pair('T', 'E')] = {9.6677E-01, 0, 9};
multiplet['4'][make_pair('I', 'I')][make_pair('T', 'F')] = {4.5013E-02, 1, 10};
// quintet D -> ?

multiplet [5'] [make_pair ('P', 'D')] [make_pair ('1', '4')]

// quintet F -> ?

multiplet [5'] [make_pair ('F', 'D')] [make_pair ('1', '0')]

// multiplet did not yield more quintet values

// end init_hash_map
// sort vector by name, for chars [3]-[5] and [8] sort by S,P,D,F,G,H,...

// ================================================= ===========================

bool sort_by_name(const transition_no &lhs, const transition_no &rhs) {
    // make 20 char strings from lhs/rhs
    string l = string() + lhs.level_lower_a10 + lhs.level_upper_a10;
    string r = string() + rhs.level_lower_a10 + rhs.level_upper_a10;

    return (sort_by_left_term(l,r));
}

// end sort_by_name

// ================================================= ===========================

bool inline sort_by_spdf(const char l, const char r) {
    // return S,P,D,F,G,H,I,K,L order
    switch(r) {
    case 'S':
        return false;
    case 'P':
        if ('S'== l)
            return true;
        return false;
    case 'D':
        if ('S'== l || 'P'== l)
            return true;
        return false;
    case 'F':
        if ('S'== l || 'P'== l || 'D'== l)
            return true;
        return false;
    case 'G':
        if ('S'== l || 'P'== l || 'D'== l || 'F'== l)
            return true;
        return false;
    case 'H':
        if ('S'== l || 'P'== l || 'D'== l || 'F'== l || 'G'== l)
            return true;
        return false;
    case 'I':
        if ('S'== l || 'P'== l || 'D'== l || 'F'== l || 'G'== l || 'H'== l)
            return true;
        return false;
    case 'K':
        if ('S'== l || 'P'== l || 'D'== l || 'F'== l || 'G'== l || 'H'== l
            return true;
        return false;
    }

81
1375    return true;
1376    return false;
1377
1378    case 'L':
1379        if ('S' == l || 'P' == l || 'D' == l || 'F' == l || 'G' == l || 'H' == l
        || 'I' == l || 'K' == l)
        return true;
1380    return false;
1381    default:
1382        return true;
1383    }
1384
1385    // sort array of line transitions (string of size 20)
1386    bool sort_by_left_term(const string &lhs, const string &rhs)
1387    {
1388        int order[14] = {7,8,9,17,18,19,3,4,5,6,13,14,15,16};
1389        for(int i = 0; i < 13; i++)
1390        {
1391            if(lhs[order[i]] == rhs[order[i]])
1392                continue;
1393            if(4 == order[i] || 8 == order[i] || 14 == order[i] || 18 == order[i])
1394                return (sort_by_spdf(lhs[order[i]], rhs[order[i]]));
1395            return (lhs[order[i]] < rhs[order[i]]);
1396        }
1397        return false;
1398    }
1399
1400    // S = 0, P = 1, D = 2, etc
1401    char inline L_value(char c)
1402    {
1403        switch(c)
1404        {
1405            case 'S':
1406                return '0';
1407            case 'P':
1408                return '1';
1409            case 'D':
1410                return '2';
1411            case 'F':
1412                return '3';
1413            case 'G':
1414                return '4';
1415            case 'H':
1416                return '5';
1417            case 'I':
1418                return '6';
1419            case 'K':
1420                return '7';
1421            case 'L':
1422                return '8';
1423            default:
1424                return '?';
1425        }
Listing 4: voigt.py (Python script used to create Figure 2)

```python
import numpy as np
from scipy.special import wofz
import pylab

def G(x, alpha):
    """Return Gaussian line shape at x with HWHM alpha"""
    return np.sqrt(np.log(2) / np.pi) / alpha
        * np.exp(-((x / alpha)**2 * np.log(2))

def L(x, gamma):
    """Return Lorentzian line shape at x with HWHM gamma"""
    return gamma / np.pi / ((x**2 + gamma**2)

def V(x, alpha, gamma):
    """Return the Voigt line shape at x with Lorentzian component HWHM gamma
    and Gaussian component HWHM alpha.
    ""
    sigma = alpha / np.sqrt(2 * np.log(2))

    return np.real(wofz((x + 1j*gamma)/sigma/np.sqrt(2))) / sigma
        / np.sqrt(2*np.pi)

alpha, gamma = 1.0, 1.0
x = np.linspace(-6.0, 6.0, 1000)
pylab.plot(x, G(x, alpha), color='green', ls=':', c='k', label='Gaussian')
pylab.plot(x, L(x, gamma), color='blue', ls='--', c='k', label='Lorentzian')
pylab.plot(x, V(x, alpha, gamma), color='red', c='k', label='Voigt')
pylab.xlim((-6.0, 6.0))
pylab.xlabel('$(\nu - \nu_0) / \alpha_D$')
pylab.ylabel('intensity')
#pylab.show()
pylab.savefig('voigt.eps', format='eps')
```