OPEN ACCESS

The Astrophysical Journal, 917:14 (17pp), 2021 August 10 https://doi.org/10.3847/1538-4357/ac02be


# The *Lightweaver* Framework for Nonlocal Thermal Equilibrium Radiative Transfer in Python

Christopher M. J. Osborne[1] and Ivan Milić[2,3,4]
[1] SUPA School of Physics and Astronomy, University of Glasgow, Glasgow G12 8QQ, UK; c.osborne.1@research.gla.ac.uk
[2] Department of Physics, University of Colorado, Boulder, CO 80309, USA
[3] Laboratory for Atmospheric and Space Physics, University of Colorado, Boulder, CO 80303, USA
[4] National Solar Observatory, Boulder, CO 80303, USA
*Received 2021 April 27; revised 2021 May 14; accepted 2021 May 17; published 2021 August 9*

## Abstract

Tools for computing detailed optically thick spectral line profiles out of local thermodynamic equilibrium have always been focused on speed, due to the large computational effort involved. With the *Lightweaver* framework, we have produced a more flexible, modular toolkit for building custom tools in a high-level language, Python, without sacrificing speed against the current state of the art. The goal of providing a more flexible method for constructing these complex simulations is to decrease the barrier to entry and allow more rapid exploration of the field. In this paper we present an overview of the theory of optically thick nonlocal thermodynamic equilibrium radiative transfer, the numerical methods implemented in *Lightweaver* including the problems of time-dependent populations and charge-conservation, as well as an overview of the components most users will interact with, to demonstrate their flexibility.

*Unified Astronomy Thesaurus concepts:* Radiative transfer (1335); Radiative transfer simulations (1967); Computational methods (1965); Solar physics (1476); Stellar physics (1621)

## 1. Introduction

Optically thick nonlocal thermodynamic equilibrium (NLTE) radiative transfer (RT) is one of the most computationally intensive problems in modern solar and stellar physics. It consists of taking a model atmosphere and computing self-consistent atomic populations while taking into account the fact that radiation originating from these atomic transitions may also affect their states elsewhere in the atmosphere. The high numerical cost of this problem is due in part to the high dimensionality of the the intensity, as it varies with wavelength and direction in addition to the spatial and temporal variation of most other quantities considered, and also the possibly large number of contributors at each wavelength. The NLTE problem can be extended to take into account the problem of finding an electron density consistent with the atomic populations, and this will also be discussed.

In recent times there has been a rise of flexible high-performance frameworks available in high-level languages such as Python. One domain where these have demonstrated their power is machine learning, where the building blocks provided by the frameworks allow researchers to rapidly prototype new systems with little loss in performance over a hand-tuned highly specific low-level implementation. The goal of *Lightweaver* is to provide a similar set of tools for plane-parallel optically thick RT. To this end it consists of an extensible Python frontend with a clean high-performance C++ backend. During development the code has been extensively tested against both RH (Uitenbroek 2001; Pereira & Uitenbroek 2015) and SNAPI (Milić & van Noort 2018), to ensure agreement between all three on a range of problems. While most RT tools are designed specifically for a single task, there is much commonality between the operations performed (especially the most costly operations, such as the formal solution of the RT equation (RTE)). It is therefore reasonable to abstract out these common building blocks in a way that allows a user to quickly build what amounts to a specialized tool with very little code, in a high-level, memory-safe language that is widely supported in the scientific computing community.

This paper describes in detail the components of the *Lightweaver* framework, including the numerical methods used. In Section 2 we provide an overview of NLTE RT and describe the numerical methods and their implementations. Then in Section 3 the structure of the framework is described to demonstrate how modularity is achieved.

*Lightweaver* can be installed by an end-user through the standard Python package manager `pip` without need for particular compilers to be installed. The code is freely available under the permissive MIT license[5] and is available on GitHub[6] with archival on Zenodo (Osborne 2021a). *Lightweaver* is in constant development and suggestions and enhancements are welcomed by contacting the authors or through the software's repository. All examples in this paper were tested against the most recent release of *Lightweaver*, v0.5.0. These examples are available on Zenodo (Osborne 2021b).

## 2. Numerical NLTE RT

In this section we first present a brief overview of NLTE RT; for a much more in-depth introduction see Hubený & Mihalas (2014). We also explain how most terms are implemented in *Lightweaver*, especially those that are less apparent.

Solving the NLTE RT problem consists primarily of two coupled sub-problems.

1. Solving the RTE to obtain the specific intensity at each frequency, point, and direction in the discretized computational domain for a given set of atomic populations and a given atmospheric model. This step is known as the formal solution of the RTE.

[5] https://opensource.org/licenses/MIT
[6] https://github.com/Goobley/Lightweaver

2. Updating the populations based on the radiative rates obtained from the formal solution.

These two problems are solved iteratively; a formal solution is first computed from an initial guess of the atomic populations, which is then used to construct a linear operator applied in conjunction with the current populations to update these until convergence. Note that to compute the population update it is necessary to compute the intensity from transitions that do not overlap the spectral range of interest, due to their effect on the balance of transitions between the atomic population levels.

In the following we will expand on the solution of these two problems, first describing the terms that enter the equations, the construction of a linear operator, and the formal solution to the RTE.

### 2.1. Basic Definitions

The most basic quantity to consider in the study of radiation and radiation transport is specific intensity. This is commonly denoted $I(\nu, \boldsymbol{d})$ at some frequency $\nu$ and direction $\boldsymbol{d}$ and has (SI) units $\mathrm{J\,m^{-2}\,s^{-1}\,Hz^{-1}\,sr^{-1}}$. Specific intensity (and its projections in the Stokes vector) is the quantity where our observations and simulations meet, and the only vector by which spectroscopic (and polarimetric) information arrives from the observed object.

A ray traveling through a medium, such as neutral gas or a plasma, gains a certain amount of energy per unit length due to emission processes in the plasma, and loses another amount due to absorption ($\chi_{\mathrm{abs}}$) and scattering ($\chi_{\mathrm{scatt}}$) processes. These gain and loss terms are called emissivity and opacity, typically denoted $\eta$ and $\chi = \chi_{\mathrm{abs}} + \chi_{\mathrm{scatt}}$ respectively, and will depend both on the frequency considered, as well as the location and direction of the ray.

Considering the case of a ray traveling through plasma made up of neutral and ionized atoms, the emissivity and opacity will depend on the number of atoms, the frequency-dependent cross-sections of these atoms, and the quantum mechanical processes coupling the photons and the plasma.

For a bound–bound process we then arrive at the following expression for the radiative rates for a transition from level $i$ to level $j$ ($j > i$):

$$R_{ij} = \oint \int B_{ij}\phi(\nu, \boldsymbol{d}) I(\nu, \boldsymbol{d}) d\nu \, d\Omega, \tag{1}$$

$$R_{ji} = \oint \int [(A_{ji} + B_{ji} I(\nu, \boldsymbol{d}))\psi(\nu, \boldsymbol{d})] \, d\nu \, d\Omega, \tag{2}$$

where $\phi$ is the line absorption profile, $\psi$ is the line emission profile, and $A$ and $B$ are the Einstein coefficients for the transition. The radiative rates have units $\mathrm{s^{-1}}$, and locally describe the number of atomic transitions ($i \to j$ and $j \to i$ respectively) per unit time. For bound–free transitions the radiative rates are given by

$$R_{ij} = \oint \int \alpha_{ij}(\nu) I(\nu, \boldsymbol{d}) d\nu \, d\Omega, \tag{3}$$

$$R_{ji} = \oint \int \left[ \left( I(\nu, \boldsymbol{d}) + \frac{2h\nu^3}{c^2} \right) \cdot \alpha_{ij}(\nu) n_e \Phi_{ij}(T) e^{-h\nu/k_{\mathrm{B}}T} \right] d\nu \, d\Omega, \tag{4}$$

where $h$ is Planck's constant, $c$ is the speed of light, $\alpha_{ij}$ is the photoionization cross-section, $k_{\mathrm{B}}$ is Boltzmann's constant, and $n_e$ is the electron number density. $\Phi$ is the Saha–Boltzmann equation defined such that

$$n_e \Phi_{ij}(T) = \frac{n_i^*}{n_j^*} = \frac{g_i}{2g_j} \left( \frac{h^2}{2\pi m_e k_{\mathrm{B}} T} \right)^{3/2} \exp\left( \frac{\Delta E_{ji}}{k_{\mathrm{B}} T} \right), \tag{5}$$

where $n^*$ is the population of the species in LTE, $m_e$ is the electron mass, $\Delta E_{ji}$ is the energy difference between levels $j$ and $i$, and $g_i$ is the statistical weight of level $i$. The Saha–Boltzmann equation is obtained by combining the Saha ionization equation and the Boltzmann excitation equation, and describes the distribution of the total atomic population across its possible states, at a given electron density, and under the assumption of LTE.

The rest frequency $\nu_{ij}$ of a transition is given by

$$\nu_{ij} = \frac{h}{\Delta E_{ji}}. \tag{6}$$

In *Lightweaver*, due to existing convention in other RT codes, the energy of each level relative to the ground level of the model atom is supplied in $\mathrm{cm^{-1}}$.

The Einstein coefficients are related to each other and to the oscillator strength (a dimensionless quantity describing absorption probability) by

$$A_{ji} = \frac{2\pi e^2 \nu_{ij}^2}{\epsilon_0 m_e c^3} f_{ij}, \tag{7}$$

$$B_{ji} = \frac{c^2}{2h\nu_{ij}^3} A_{ji}, \tag{8}$$

$$B_{ij} = \frac{g_j}{g_i} B_{ji}, \tag{9}$$

where $f_{ij}$ is the oscillator strength, $e$ is the charge of an electron, and $\epsilon_0$ is the vacuum permittivity. As the oscillator strength can be used to compute the Einstein coefficients for a transition, and again for consistency with other codes, *Lightweaver* requires $f_{ij}$ for each spectral line.

### 2.2. Line Broadening

A transition between well-defined energy states in a bulk motionless plasma is not infinitely narrow, but instead broadened by a number of factors, including natural radiative broadening, Doppler broadening, and collisional broadening, to give the absorption profile $\phi_{ij}$. By default we follow the standard assumption of a Voigt absorption profile, and allow for a combination of different damping terms. There are described in detail in Appendix A.

The design of *Lightweaver* also supports non-Voigt line profiles, such as the more complex model for electric pressure broadening discussed in Kowalski et al. (2017), while only modifying the Python code in the model atom object; however, the standard code-path for the Voigt profile is more optimized. The example presented in Section 3.7 shows how a Doppler line profile could be implemented.

### 2.3. MALI

In the following we present a brief review of the numerical techniques implemented in *Lightweaver*. Much of the content follows Uitenbroek (2001) and the RH code described therein.

Given an atomic species where the population of excitation level $i$ is given by $n_i$ the general form of the kinetic equilibrium equation is given by

$$\frac{\partial n_i}{\partial t} + \nabla \cdot (n_i \boldsymbol{v}) = \sum_{j \neq i} n_j P_{ji} - n_i \sum_{j \neq i} P_{ij}, \tag{10}$$

where $\boldsymbol{v}$ is the bulk macroscopic velocity of the particle distribution, $P_{ij}$ is the total transition rate between atomic states $i$ and $j$ and is given by

$$P_{ij} = R_{ij} + C_{ij}, \tag{11}$$

where $R_{ij}$ is the radiative rate, due to interaction with photons or spontaneous emission, and $C_{ij}$ is the collisional rate, due to interaction with other particles. In NLTE studies it is normally assumed that the collisional rate can be known a priori from the model atmosphere definition.

A common simplification of (10) is to assume that the atmosphere is in a steady state (i.e., $\partial n_i / \partial t = 0$), and therefore the advective term can also be ignored. This sets the left-hand side of (10) to 0 and we obtain the statistical equilibrium equation

$$\sum_{j \neq i} n_j P_{ji} - n_i \sum_{j \neq i} P_{ij} = 0. \tag{12}$$

For both (10) and (12), the system must be solved simultaneously for all levels of an atomic species, the latter requiring a constraint equation to avoid degeneracies.

*Lightweaver* adopts the same Rybicki–Hummer full preconditioning approach (Rybicki & Hummer 1992) as used in RH (Uitenbroek 2001), although the implementation is slightly different. Following these authors, we write the emissivity $\eta$ and opacity $\chi$ of a transition between atomic levels $i$ and $j$, at frequency $\nu$, along a ray of direction $\boldsymbol{d}$ as

$$\eta_{ij} = n_j U_{ji}(\nu, \boldsymbol{d}), \tag{13}$$

$$\chi_{ij} = n_i V_{ij}(\nu, \boldsymbol{d}) - n_j V_{ji}(\nu, \boldsymbol{d}), \tag{14}$$

where $n_i$ is the population density level $i$. We assume here that $j > i$ and then, by convention, $\chi_{ji} = -\chi_{ij}$.

The $U$ and $V$ terms are defined for bound–bound and bound–free transitions as

$$U_{ji} = \begin{cases} \frac{h\nu}{4\pi} A_{ji} \psi_{ij}(\nu, \boldsymbol{d}), & \text{bound} - \text{bound} \\ n_e \Phi_{ij}(T) \left( \frac{2h\nu^3}{c^2} \right) e^{-h\nu / k_B T} \alpha_{ij}(\nu), & \text{bound} - \text{free}, \end{cases} \tag{15}$$

$$V_{ij} = \begin{cases} \frac{h\nu}{4\pi} B_{ij} \phi_{ij}(\nu, \boldsymbol{d}), & \text{bound} - \text{bound} \\ n_e \Phi_{ij}(T) e^{-h\nu / k_B T} \alpha_{ij}(\nu), & \text{bound} - \text{free}, \end{cases} \tag{16}$$

$$V_{ji} = \begin{cases} \frac{h\nu}{4\pi} B_{ji} \psi_{ij}(\nu, \boldsymbol{d}), & \text{bound} - \text{bound} \\ \alpha_{ij}(\nu), & \text{bound} - \text{free}. \end{cases} \tag{17}$$

By convention we define $U_{ij} = U_{ii} = V_{ii} = 0$.

*Lightweaver* can also treat lines necessitating partial redistribution (PRD). Following Uitenbroek (2001) we define

$$\rho_{ij}(\nu, \boldsymbol{d}) = \frac{\psi_{ij}(\nu, \boldsymbol{d})}{\phi_{ij}(\nu, \boldsymbol{d})} \tag{18}$$

and thus

$$U_{ji} = \frac{h\nu}{4\pi} A_{ji} \rho_{ij}(\nu, \boldsymbol{d}) \phi_{ij}(\nu, \boldsymbol{d}), \quad \text{bound–bound} \tag{19}$$

$$V_{ji} = \frac{h\nu}{4\pi} B_{ji} \rho_{ij}(\nu, \boldsymbol{d}) \phi_{ij}(\nu, \boldsymbol{d}), \quad \text{bound–bound.} \tag{20}$$

In the case of complete redistribution (CRD) $\rho = 1$. These terms will be discussed in detail in Section 2.5.

The total opacity and emissivity can then be found by summing over all species:

$$\eta_{\text{tot}}(\nu, \boldsymbol{d}) = \sum_{\text{species}} \left[ \sum_j \sum_{i<j} \eta_{ij}(\nu, \boldsymbol{d}) \right], \tag{21}$$

$$\chi_{\text{tot}}(\nu, \boldsymbol{d}) = \sum_{\text{species}} \left[ \sum_j \sum_{i<j} \chi_{ij}(\nu, \boldsymbol{d}) \right]. \tag{22}$$

In *Lightweaver* we split species into three categories.

1. Background: bound–free transitions are considered under the assumption of LTE. The opacity and emissivity contribution here is considered to be isotropic.
2. Detailed: all transitions (bound–bound and bound–free) are considered in detail, using either given (e.g., from a previous NLTE simulation) or LTE populations. The opacity and emissivity contribution here is considered to be angle-dependent.
3. Active: all transitions are considered in detail and terms necessary for iterating the populations are accumulated. The opacity and emissivity contribution here is considered to be angle-dependent.

The expressions for total emissivity and opacity can then be written as the summation over the emissivity and opacity in each of the three previous categories, for each frequency and direction.

The source function at a given frequency and direction is then given by

$$S(\nu, \boldsymbol{d}) = \frac{\eta_{\text{tot}}(\nu, \boldsymbol{d}) + \sigma(\nu) J(\nu)}{\chi_{\text{tot}}(\nu, \boldsymbol{d})}, \tag{23}$$

where $\sigma$ is the continuum scattering coefficient that will be discussed further in Section 2.9.5.

It is common to define an operator, $\Lambda$, used to obtain the monochromatic radiation field in a particular direction from the source function:

$$I(\nu, \boldsymbol{d}) = \Lambda_{\nu, \boldsymbol{d}}[S(\nu, \boldsymbol{d})]. \tag{24}$$

In essence, this is our formal solver, discussed in Section 2.4.1. Rybicki & Hummer (1992) introduce an additional operator, $\Psi$, that aids in the construction of a linear preconditioned iterative scheme for solving (12) such that

$$\Psi_{\nu, \boldsymbol{d}}[\eta_{\text{tot}}(\nu, \boldsymbol{d})] = \Lambda_{\nu, \boldsymbol{d}} \left[ \frac{\eta_{\text{tot}}(\nu, \boldsymbol{d})}{\chi^{\dagger}_{\text{tot}}(\nu, \boldsymbol{d})} \right], \tag{25}$$

where $\chi^{\dagger}_{\rm tot}$ is the opacity evaluated with the populations from the previous iteration. For the converged solution, these two operators are equivalent, as $\chi^{\dagger} = \chi$.

Taking the $\boldsymbol{n}$ as the vector of level populations $\{n_1, n_2,\ldots,n_N\}$ at a location in the atmosphere, we can write our iterative scheme for (10) as

$$\frac{\partial n_i}{\partial t} + \nabla \cdot (n_i \boldsymbol{v}) = \Gamma_i \boldsymbol{n}, \tag{26}$$

where $\Gamma_i$ is a row vector from the matrix $\Gamma = \Gamma^C + \Gamma^R$, which is evaluated using the previous population estimate. $\Gamma^C$ and $\Gamma^R$ represent the preconditioned collisional and radiative rate equations respectively. We will address the construction of $\Gamma^C$ later. From Rybicki & Hummer (1992) and Uitenbroek (2001) we can write

$$\Gamma^R_{ll'} = \oint\int \frac{1}{h\nu}(U^{\dagger}_{l'l} + V^{\dagger}_{l'l} I^{\rm eff}_{\nu,\boldsymbol{d}} - \left(\textstyle\sum_{m\neq l} \chi^{\dagger}_{lm}\right)\Psi^*_{\nu,\boldsymbol{d}}\left[\textstyle\sum_p U^{\dagger}_{l'p}\right]\Big) d\nu\, d\Omega \tag{27}$$

for $l \neq l'$, and all $\dagger$ terms evaluated with the current level population and $\rho$ estimates. The term

$$I^{\rm eff}_{\nu,\boldsymbol{d}} = I^{\dagger}(\nu, \boldsymbol{d}) - \Psi^*_{\nu,\boldsymbol{d}}\left[\sum_{i,j} \eta^{\dagger}_{ij}\right], \tag{28}$$

when assuming a diagonal $\Psi^*$ operator, describes the nonlocal contribution to the radiation field from the atom in question, and the local contribution from other species. It is often separated, as it remains constant for all transitions in an atom. The diagonal terms of $\Gamma$ are computed using the conservation property that requires, for the sake of total number conservation, that the sum of each column of $\Gamma$ be zero (Rybicki & Hummer 1992). Thus,

$$\Gamma_{ll} = -\sum_{m\neq l} \Gamma_{ml}. \tag{29}$$

Now that $\Gamma$ has been constructed it can be used in (26). In the case of statistical equilibrium, we must solve the matrix-vector equation $\Gamma\boldsymbol{n} = \boldsymbol{0}$. A constraint equation is also needed, to avoid the trivial solution ($\boldsymbol{n} = \boldsymbol{0}$), typically a constraint on the total number density of the species. In effect, this amounts to replacing one of the equations with a sum over the level populations, i.e., replacing one of the *rows* of $\Gamma$ with ones, and the associated entry in right-hand side with the total population number density.

The discretization of the time-dependent form of the kinetic equilibrium equation is discussed in the following section as it involves extra complexities strongly coupled to the numerical methods applied.

#### 2.3.1. Numerical Implementation

Most of the integration terms proceed similarly to those of the RH code, but as those have not been presented in a single document, we describe the numerical implementation in detail here.

When considering a one-dimensional plane parallel atmosphere, as is done in *Lightweaver*, it is efficient to discretize the integration over solid-angle using Gauss–Legendre quadrature over the cosine of the angle between the ray and the normal to atmospheric slabs, commonly denoted $\mu$. These integrations are then implemented as weighted summations of the integrand at the Gauss–Legendre nodes i.e., the angle-averaged intensity

$$J(\nu) = \frac{1}{4\pi}\oint I(\nu, \boldsymbol{d}) d\Omega \tag{30}$$

at a point in the atmosphere can be calculated from

$$J(\nu) = \sum_{\mu} I(\nu, \boldsymbol{d}) w_{\mu}. \tag{31}$$

The number of angle points is user defined, as it depends on the problem (the anisotropy of the radiation field): for static atmospheres three angle samplings are normally sufficient, whereas five is more reliable in dynamic atmospheres.

As *Lightweaver* handles overlapping transitions, there needs to be a common wavelength grid that covers all transitions for the problem in question. Each transition provides a set of wavelengths that need to be taken into account to reliably solve the RT problem (e.g., lines are typically densely sampled in the line core and sparse in the wings). All of these individual wavelength grids are combined to produce the global wavelength grid, and a new grid is created for each transition which contains all of the original points, as well as the wavelength points from all other transitions that overlap.

These wavelength grids also define the basis of a numerical quadrature that is described in Appendix B. Therein we also describe the specific accumulation terms used in the construction of the fully preconditioned $\Gamma$.

In the case of the time-dependent kinetic equilibrium, there is no "one-size-fits-all" approach to this equation and *Lightweaver* provides the following tools. The advective term in (26) is ignored, as this requires a more complete treatment including consideration of hydrodynamics. We can discretize $\partial\boldsymbol{n}/\partial t = \Gamma\boldsymbol{n}$ using a theta method,

$$\frac{\boldsymbol{n}^{t+1} - \boldsymbol{n}^t}{\Delta t} = \theta\Gamma^{t+1}\boldsymbol{n}^{t+1} + (1 - \theta)\Gamma^t\boldsymbol{n}^t, \tag{32}$$

where the superscripts $t$ and $t+1$ indicate the start and end of the timestep being integrated over, $\Delta t$ the duration of the timestep, and $\theta$ the degree of implicitness. $\theta = 0.5$ represents the Crank–Nicolson scheme, $\theta = 1$ the backward Euler scheme, and $\theta = 0.55$ is commonly used as it is often found to cope better with stiff systems (e.g., Viallet et al. 2011). This system is solved by storing $\Gamma^t$ at the start of the process, and then updating $\Gamma^{t+1}$ using revised updates of the populations $\boldsymbol{n}^{t+1}$ with each iteration. The process of obtaining a new estimate for $\boldsymbol{n}^{t+1}$ can be found by rearranging (32) into the form

$$(\mathbb{I} - \theta\Delta t\Gamma^{t+1})\boldsymbol{n}^{t+1} = (1 - \theta)\Delta t\Gamma^t\boldsymbol{n}^t + \boldsymbol{n}^t, \tag{33}$$

where $\mathbb{I}$ is the identity matrix. As the right-hand side is known a priori, it can be evaluated directly, and (33) is a matrix-vector system that can be solved equivalently to the statistical equilibrium case, albeit without the need for a constraint equation. Currently only the fully implicit $\theta = 1$ case is supported, as during testing the differences were found to be insignificant; however, we plan to include support for other $\theta$ in the future, and this has already been implemented in separate packages that use *Lightweaver*, but without modifying the base framework.

### 2.4. RTE

To obtain the intensity terms in the radiative rates, as well as the outgoing intensity, we need to solve the monochromatic RTE, which for a one-dimensional plane-parallel atmosphere stratified along the $z$-axis is expressed as

$$\mu \frac{\partial I(\nu, \boldsymbol{d})}{\partial z} = \eta(\nu, \boldsymbol{d}) - \chi(\nu, \boldsymbol{d}) I(\nu, \boldsymbol{d}), \tag{34}$$

or along an optical-depth stratification as

$$\mu \frac{\partial I(\nu, \boldsymbol{d})}{\partial \tau(\nu)} = I(\nu, \boldsymbol{d}) - S(\nu, \boldsymbol{d}), \tag{35}$$

for the optical depth defined as $d\,\tau(\nu) = -\chi(\nu)\,dz$.

Solving this equation for multiple projected angles $\mu$ provides the radiation field throughout the model atmosphere that is necessary to compute the $\Gamma$ operator. Typically the optical depth formulation of (35) is solved as it is more numerically robust (de la Cruz Rodríguez & Piskunov 2013; Janett et al. 2018).

#### 2.4.1. Formal Solver

The formal solver is the technique by which the RTE (34) is solved and the approximate operator $\Psi^*$ is computed. By default we adopt the third-order Bézier spline short-characteristics approach of de la Cruz Rodríguez & Piskunov (2013) and de la Cruz Rodríguez et al. (2019); however, investigation is also under way into the use of pragmatic formal solvers as discussed in Janett et al. (2018) and the BESSER formal solver of Štěpán & Trujillo Bueno (2013).

In the short-characteristics approach the formal solver is provided with the opacity and source function at discretized points throughout the atmosphere, and the behavior of these between the known points is assumed to follow a simple function that can be analytically integrated, in this case a third-order Bézier spline. It is important to choose an interpolating function that varies smoothly and minimizes or, better yet, eliminates under- and overshoots in the interpolant. The third-order Bézier spline has proven to be robust in this setting and has been applied in other modern codes such as STiC (de la Cruz Rodríguez et al. 2019) and SNAPI (Milić & van Noort 2018).

The integration routine proceeds from one end of the atmosphere to the other, accumulating these terms through the analytic short-characteristics integration to obtain the up- or down-going intensity for this ray at each point in the atmosphere.

The approximate $\Psi$ operator $\Psi^*$ is simply the diagonal of the true $\Psi$ operator, a matrix that would map the vector of emissivity to the intensity. $\Psi^*$ is trivially computed during the formal solution from the local contribution terms to the intensity and the local opacity.

The simple linear short-characteristics formal solver is also present and new formal solvers that conform to the interface used in *Lightweaver* can be compiled separately and loaded from a shared code library, allowing *Lightweaver* to serve as a testbed without the need to modify the core package.

### 2.5. PRD

The effects of PRD are important for some NLTE lines, typically strong resonance lines and lower-density regions where radiative effects dominate over collisional effects (Hubený & Mihalas 2014). For a complete treatment of the theory describing PRD lines we direct readers to Hubený & Mihalas (2014) and references therein, but we will provide a basic overview here. The common assumption of CRD in spectral lines is that $\psi = \phi$. The argument is that most lines are formed in regions with sufficient elastic collisions that atoms are well distributed across the sub-states of each energy level. Emission is therefore not correlated with the absorbed photon that excited the atom into this state. When the plasma is less collisional, there is said to be a natural population of a particular level, i.e., a population where the emission frequency is correlated to the absorption frequency. In this case the emission profile $\psi$ differs from the absorption profile $\phi$, and these coherent scattering effects must be considered.

*Lightweaver* currently adopts the iterative PRD approach presented in Uitenbroek (2001), but may also in future implement a direct solution, as it may prove more robust than the iterative approach for some highly dynamic problems, despite the higher computational cost. Currently cross-redistribution is not implemented, but the groundwork is present, and the remaining changes would be a simple extension following Uitenbroek (2001) and the RH code.

In the common case where flows are lower than the thermal Doppler velocity, the integrations needed to solve the PRD equations can be simplified by assuming isotropy of the radiation field. This is known as angle-averaged PRD. In cases with stronger flows we instead employ the hybrid PRD approach of Leenaarts et al. (2012) which consists of computing $\rho$ in the atom's rest frame. This approximation agrees quite well with a full angle-dependent treatment, is simple to implement, and much faster to evaluate than the full angle-dependent case. Due to the additional computation effort involved in PRD calculations, regardless of the method used, lines need to be explicitly labeled as PRD.

The derivation of the PRD equations and their numerical implementation is described in Appendix C.

### 2.6. Self-consistent Electron Density

The MALI technique assumes that the electron density is known a priori, but this is often not the case. Assuming that the electron density can be given by the LTE ionization state of the plasma can yield substantially incorrect results for chromospheric and prominence lines (Heinzel 1995; Paletou 1995; Bjørgen et al. 2019). An additional iteration process is therefore needed to determine the correct electron density within the framework of the NLTE problem.

While not quite as robust as the pure MALI treatment a secondary Newton–Raphson iteration to self-consistently compute electron density was proposed by Heinzel (1995) and Paletou (1995), and forms the basis of the method implemented here. The time-dependent case is based on Kašparová et al. (2003), and the numerical implementation of both of these is described in Appendix D.

### 2.7. Collisional Rates

Based on the RH code, a number of different formulations for collisional rates are available in *Lightweaver*. Currently these include tabulated collision strength ($\Omega$) against temperature for excitation of ions by electrons, tabulated collisional ionization and excitation rates of neutrals by electrons (known in RH as CI and CE), tabulated collisional excitation by

protons, neutral hydrogen, and charge exchange with these species (CP, CH, CH+, and CH0 respectively). Additionally the collisional ionization rates of Arnaud & Rothenflug (1985) and Burgess & Chidichimo (1983) are present. These can be extended further in user code with no modifications to the base library. The choice of pre-implemented collisional rates in the *Lightweaver* "standard library" allows the direct conversion of the majority of model atoms that are distributed with RH to also be distributed with *Lightweaver*. The collisional rates for a level depend only on the local parameters, and have no wavelength dependence, therefore the implementation is much more straightforward and does not require complicated numerical integration.

By default, the collisional rates are re-evaluated at the start of each formal solution, although this can be disabled by the user.

### 2.8. Full Stokes Treatment

We also support Zeeman splitting and polarization effects where the complete set of anomalous Zeeman splitting terms can be computed from the quantum numbers $J$, $L$, and $S$ for the levels considered through the $LS$ coupling formalism, or a classical Zeeman triplet computed from an effective Landé $g$-factor present in the definition of a line. *Lightweaver* does not support full Stokes iteration of the populations, but provides support for both the field-free and polarization-free approaches (Trujillo Bueno & Landi Degl'Innocenti 1996). The final formal solution is then undertaken with the third-order Bézier spline Diagonal Element Lambda Operator method of de la Cruz Rodríguez & Piskunov (2013).

### 2.9. Miscellaneous

Like RH, *Lightweaver* utilizes base SI units throughout, with the singular exception of wavelength being treated in nanometers. The units of a variable are therefore easy to determine, with little consideration of derived units. In the remainder of this section we will discuss other small implementation details of the code.

#### 2.9.1. Collisional–Radiative Switching

The collisional-radiative switching (CRSW) technique of Hummer & Voels (1988) is available in *Lightweaver*. Using MALI, many problems will converge without much issue; however, in the case of strong atmospheric gradients the corrections to the populations in early iterations can be overly large and drive the system into a poorly conditioned state. To avoid this the CRSW technique multiplies the collisional contributions to $\Gamma$ by a significant factor, so as to force the system into LTE. This factor is slowly reduced, allowing a graceful departure from LTE toward NLTE. The exact decay of this parameter can be configured by the user.

#### 2.9.2. Isotopes

Isotopic models are also supported as valid atomic models. By default the abundances for all elements and their isotopic proportions are taken from Asplund et al. (2009); however, these can easily be modified by the user.

#### 2.9.3. Equation of State

*Lightweaver* contains a simple equation of state and background opacity package based on Mihalas (1978), implemented by Wittmann, and ported to Python by J. de la Cruz Rodriguez.[7] This equation of state has also been used in SIR (Ruiz Cobo & del Toro Iniesta 1992) and NICOLE (Socas-Navarro et al. 2015). In *Lightweaver* it is often used to determine the values of unknown parameters in a provided model atmosphere, and determining an LTE hydrostatic stratification if necessary (based on NICOLE). The equation of state also provides an estimate of the reference opacity $\tau_{500}$ at 500 nm for model atmospheres that provide a height- or column-mass-based stratification.

**Table 1**
References for Components Present in Default Background Opacity Package

| Component | References |
|---|---|
| H free–free | Mihalas (1978) |
| $H_2^-$ free–free | Bell (1980) |
| $H_2^+$ free–free | Bates (1952) |
| $H_2$ Rayleigh scattering | Victor & Dalgarno (1969) |
| $H^-$ bound–free | Geltman (1962), Mihalas (1978) |
| $H^-$ free–free | Stilley & Callaway (1970), Mihalas (1978) |
| $H^-$ free–free (>9113 nm) | John (1988) |
| OH bound–free | Kurucz et al. (1987) |
| CH bound–free | Kurucz et al. (1987) |
| General Rayleigh scattering | Mihalas (1978) |

#### 2.9.4. Molecules

While molecular lines are not currently supported by *Lightweaver*, it can compute molecular formation in instantaneous chemical equilibrium, using the same molecular models as RH. These molecules reduce the populations of the atoms bound up in them and some (OH, CH, and $H^-$) contribute to the background opacity. The $H^-$ population is always computed, due to its importance in obtaining correct background opacities.

#### 2.9.5. Background Treatment

The default implementation of background emissivities, opacities, and scattering terms currently follows that of RH, but a more general interface that is trivially overrideable in user code without modifying the framework is also present. The components present in the default background opacity package are listed in Table 1. The OH and CH opacities are not present unless these molecules are explicitly loaded and instantaneous chemical equilibrium is computed as discussed in Section 2.9.4.

#### 2.9.6. Interpolation

For interpolation duties, other than those in the formal solver and calculation of the PRD terms, we adopt the rapid, but robust fourth-order weighted essentially non-oscillatory approach presented in Janett et al. (2019). While this technique does not guarantee monotonicity around discontinuities, the over- and under-shoots remain very small, with no ringing artifacts, and we feel that the high quality of the solution in smooth regions makes it worthwhile. We have provided a performant implementation of this technique as a separate Python package that is available through `pip` as `weno4`, on GitHub[8] and archived on Zenodo (Osborne 2021c).

[7] https://github.com/jaimedelacruz/witt/
[8] https://github.com/Goobley/Weno4Interpolation

## 3. Description of Major Code Components

In this section we provide a brief overview of the components of *Lightweaver* a user will typically interact with. The frontend is entirely constructed in Python with a binding layer written in Cython[9] (Behnel et al. 2011) to allow it communicate with the C++ backend.

### 3.1. Atomic Models

The information stored in model atoms used by contemporary codes is more than simple atomic data, and in essence these codes are defining their own ad hoc scripting languages to support reading the various terms encoded in these files. As we have access to a high-level dynamic language in the form of Python, it is reasonable to model these as object hierarchies where we can take advantage of the common Python convention that `obj == eval(repr(obj))` i.e., evaluating the textual representation of the object generates an equivalent object.

The code in Figure 1 shows the complete source necessary for a three-level + continuum hydrogen atom. It is constructed from nested Python classes, and through inheritance user code that implements the same interfaces will be able to extend these further.

For example, taking the `quadrature` component of a spectral line, we see here that the `LinearCoreExpWings` class is used. This is a derived class of `LineQuadrature` and any derived instance of this class can be used here. The requirements are that it provide at least functions `Doppler_units`, `wavelength`, and `__repr__`, that return the quadrature in Doppler units and wavelength respectively, and specify how to print the object so it can be re-evaluated. The last of these is trivial and there are plenty of examples throughout the *Lightweaver* codebase. Line-broadening terms are implemented similarly. One strength of the model atoms being implemented in terms of objects is the ease by which they can be manipulated with a simple script before being used or saved in text form (or an optimized Python object storage format such as `pickle`[10]).

### 3.2. Radiative Set

During the configuration of a simulation, all atomic models, whether "active" (full NLTE), "detailed static" (transitions computed in detail, but populations fixed), or "passive" (background contributions only) are stored in a `RadiativeSet` object. This is responsible for producing the common wavelength grid from all transitions taken into account, and the final grid for each transition while taking the other transitions into account as discussed in Section 2.3.1. These data are returned in a `SpectrumConfiguration` object, which can create another instance of itself, covering a restricted range of wavelengths, that is often used for computing a final formal solution over a line in detail, after the NLTE iteration is complete.

The `RadiativeSet` is also responsible for determining the LTE populations of these species from their models and the atmospheric data provided. During this process the electron density can be assumed fixed, as provided in the atmospheric data, or can be iterated to be self-consistent with the LTE populations. In the future this object will also be responsible for optionally applying a variant of the second-order escape probability method of Hummer & Rybicki (1982), applied to MALI by Judge (2017), which currently resides in the C++ backend. These LTE and initial condition atomic population data are returned in a `SpeciesStateTable`.

### 3.3. Species State Table

The `SpeciesStateTable` is responsible for holding both the LTE and NLTE populations of the species (and molecules) present in the simulation, as well as the radiative rates for species treated in detail. This object can also update the LTE and $H^-$ populations given an updated set of atmospheric data, thus facilitating time-dependent simulations. The arrays in this object are updated automatically by the C++ backend, as they are in fact shared *by reference*, and the backend is operating directly on the same memory, with no duplication necessary. This is achieved with a lightweight C++ library allowing multidimensional views onto a non-owned segment of data. These arrays provide a subset of NumPy functionality, and are limited to handling contiguous memory for performance. Thanks to the use of C++ templates for various data types, these have been verified to compile to assembly equivalent to access into a flat array, with no performance loss, but substantially greater memory safety than raw pointers, and the option to enable bounds-checking during debugging (by adjusting compilation flags).

### 3.4. Context

The code objects discussed so far primarily describe the configuration of the simulation which is then controlled by the `Context` object. The `Context` takes these data, in addition to several other configuration options, such as the whether to use hybrid PRD, charge conservation, CRSW, Ng acceleration (Ng 1974), multithreading options, and initial solution to use (which, as discussed in Section 3.2, will be moved to the frontend in future). The effects of most of these options can also be achieved by calling some extra methods, but are simplified when used as arguments to the `Context` initializer. During initialization the `Context` computes background opacity, emissivity and scattering, and line profiles, and maps the data into a form which can be used by the backend.

After this initial setup the `Context` can be used to interact with the backend by calling various methods. These include the following.

1. `formal_sol_gamma_matrices` which evaluates the collisional rates and formal solution for all wavelengths, and constructs the $\Gamma$ operator.
2. `single_stokes_fs` which computes the polarized line profiles (if not already present), and computes a full Stokes formal solution.
3. `prd_redistribute` which performs a number of PRD sub-iterations, until either the maximum number of sub-iterations is performed, or the update size falls under a configurable tolerance.
4. `stat_equil` which computes the solution of the statistical equilibrium equations given the previously computed $\Gamma$ operator.
5. `time_dep_update` which computes the solution of the kinetic equilibrium equations for one step of a provided duration.

[9] https://cython.org/
[10] https://docs.python.org/3/library/pickle.html

```
AtomicModel(element=PeriodicTable['H'],
        levels=[
                AtomicLevel(E=     0.000, g=2, label="H I 1S 2SE", stage=0, J=Fraction(1, 2), L=0,
                    S=Fraction(1, 2)),
                AtomicLevel(E= 82258.211, g=8, label="H I 2P 2PO", stage=0, J=Fraction(7, 2), L=1,
                    S=Fraction(1, 2)),
                AtomicLevel(E= 97491.219, g=18, label="H I 3D 2DE", stage=0, J=Fraction(17, 2), L=2,
                    S=Fraction(1, 2)),
                AtomicLevel(E=109677.617, g=1, label="H II", stage=1, J=None, L=None, S=None),
        ],
        lines=[
                VoigtLine(j=1, i=0, f=4.162e-01, type=LineType.CRD,
                    quadrature=LinearCoreExpWings(qCore=15, qWing=600, Nlambda=100),
                    broadening=LineBroadening(natural=[RadiativeBroadening(gamma=4.7e+08)],
                    elastic=[VdwUnsold(vals=[1.0, 1.0]), QuadraticStarkBroadening(coeff=1),
                    HydrogenLinearStarkBroadening()])),
                VoigtLine(j=2, i=0, f=7.910e-02, type=LineType.CRD,
                    quadrature=LinearCoreExpWings(qCore=5, qWing=250, Nlambda=40),
                    broadening=LineBroadening(natural=[RadiativeBroadening(gamma=9.98e+07)],
                    elastic=[VdwUnsold(vals=[1.0, 1.0]), QuadraticStarkBroadening(coeff=1),
                    HydrogenLinearStarkBroadening()])),
                VoigtLine(j=2, i=1, f=6.407e-01, type=LineType.CRD,
                    quadrature=LinearCoreExpWings(qCore=3, qWing=200, Nlambda=40),
                    broadening=LineBroadening(natural=[RadiativeBroadening(gamma=9.98e+07)],
                    elastic=[VdwUnsold(vals=[1.0, 1.0]), QuadraticStarkBroadening(coeff=1),
                    HydrogenLinearStarkBroadening()])),
        ],
        continua=[
                HydrogenicContinuum(j=3, i=0, NlambdaGen=20, alpha0=6.152e-22, minWavelength=50),
                HydrogenicContinuum(j=3, i=1, NlambdaGen=20, alpha0=1.379e-21,
                    minWavelength=91.176),
                HydrogenicContinuum(j=3, i=2, NlambdaGen=20, alpha0=2.149e-21,
                    minWavelength=205.147),
        ],
        collisions=[
                CE(j=1, i=0, temperature=[3000.0, 5000.0, 7000.0, 10000.0, 20000.0, 30000.0],
                    rates=[9.75e-16, 6.098e-16, 4.535e-16, 3.365e-16, 2.008e-16, 1.56e-16]),
                CE(j=2, i=0, temperature=[3000.0, 5000.0, 7000.0, 10000.0, 20000.0, 30000.0],
                    rates=[1.437e-16, 9.069e-17, 6.798e-17, 5.097e-17, 3.118e-17, 2.461e-17]),
                CE(j=2, i=1, temperature=[3000.0, 5000.0, 7000.0, 10000.0, 20000.0, 30000.0],
                    rates=[1.127e-14, 8.077e-15, 6.716e-15, 5.691e-15, 4.419e-15, 3.89e-15]),
                CI(j=3, i=0, temperature=[3000.0, 5000.0, 7000.0, 10000.0, 20000.0, 30000.0],
                    rates=[2.635e-17, 2.864e-17, 3.076e-17, 3.365e-17, 4.138e-17, 4.703e-17]),
                CI(j=3, i=1, temperature=[3000.0, 5000.0, 7000.0, 10000.0, 20000.0, 30000.0],
                    rates=[5.34e-16, 6.596e-16, 7.546e-16, 8.583e-16, 1.025e-15, 1.069e-15]),
                CI(j=3, i=2, temperature=[3000.0, 5000.0, 7000.0, 10000.0, 20000.0, 30000.0],
                    rates=[2.215e-15, 2.792e-15, 3.169e-15, 3.518e-15, 3.884e-15, 3.828e-15]),
])
```

**Figure 1.** Complete code for a three level + continuum hydrogen atomic model.

6. `nr_post_update` which computes the self-consistent electron density following Section 2.6. In the case of statistical equilibrium this is called automatically, if the `Context` was initialized in charge conservation mode.
7. `update_deps` which updates various quantities such as the background opacity and line profiles to handle modifications to the model atmosphere (e.g., computing a finite-difference response function or changing time-steps in the case of a time-dependent simulation).
8. `compute_rays` which can compute a formal solution for one or multiple different viewing angles, optionally with full Stokes RT.

The `Context` and all other associated Cython components have been designed to support the Python `pickle` serialization and deserialization standard. Therefore it is possible to save an entire context to disk, and reload it and continue processing with only a few lines of code. This will be discussed further in Section 3.5.

### 3.5. Parallelization

Two forms of parallelization are supported by *Lightweaver*, one explicitly, and the other implicitly. The `Context` object explicitly supports parallelization of the formal solver over multiple threads of a single process by splitting the wavelengths over threads. This approach also applies to PRD lines, for which the scattering integral can be computed in parallel.

When the aim is to process multiple atmospheres, for example in the case of a finite-difference response function or a 1.5D atmosphere simulation, it is more efficient to use *Lightweaver* in a multiprocess mode. This can be done simply using, e.g., `ProcessPoolExecutor`[11] from the Python standard library for single computing nodes, or a Python MPI implementation for a multinode cluster. This method of computing is supported by the ability to `pickle` the `Context`, allowing the entire simulation state to be shipped

[11] https://docs.python.org/3/library/concurrent.futures.html

```python
from lightweaver.fal import Falc82
from lightweaver.rh_atoms import H_6_atom, C_atom, O_atom, Si_atom, Al_atom, CaII_atom, Fe_atom,
    He_atom, MgII_atom, N_atom, Na_atom, S_atom
import lightweaver as lw
import matplotlib.pyplot as plt
import numpy as np

def iterate_ctx(ctx, Nscatter=3, NmaxIter=500):
    for i in range(NmaxIter):
        dJ = ctx.formal_sol_gamma_matrices()
        # NOTE(cmo): Do some initial iterations without touching the
        # populations to lambda iterate the background scattering terms
        if i < Nscatter:
            continue
        delta = ctx.stat_equil()

        # NOTE(cmo): Check convergence
        if dJ < 3e-3 and delta < 1e-3:
            print('Iterations taken: %d' % (i+1))
            print('-'*60)
            return

wave = np.linspace(853.9444, 854.9444, 1001)
def synth_8542(atmos, conserve, useNe):
    # NOTE(cmo): Configure the Gauss-Legendre angular quadrature for 5 rays
    atmos.quadrature(5)

    # NOTE(cmo): Construct the RadiativeSet with the following atomic models
    aSet = lw.RadiativeSet([H_6_atom(), C_atom(), O_atom(), Si_atom(), Al_atom(), CaII_atom(),
                            Fe_atom(), He_atom(), MgII_atom(), N_atom(), Na_atom(), S_atom()])
    # NOTE(cmo): Set Hydrogen and Calcium to active
    aSet.set_active('H', 'Ca')
    # NOTE(cmo): Compute the SpectrumConfiguration for this RadiativeSet
    spect = aSet.compute_wavelength_grid()

    # NOTE(cmo): If we're using the electron density provided with FAL C, then
    # compute the associated LTE populations, otherwise find a solution for
    # self consistent LTE populations and electron density.
    if useNe:
        eqPops = aSet.compute_eq_pops(atmos)
    else:
        eqPops = aSet.iterate_lte_ne_eq_pops(atmos)

    # NOTE(cmo): Construct the Context, optionally setting chargeConservation and the number of
        threads to use.
    ctx = lw.Context(atmos, spect, eqPops, conserveCharge=conserve, Nthreads=8)

    # NOTE(cmo): Iterate the NLTE problem to convergence
    iterate_ctx(ctx)
    # NOTE(cmo): Compute a detailed solution to Ca II 8542 on the 1 nm wavelength grid above
    Iwave = ctx.compute_rays(wave, [1.0], stokes=False)
    return Iwave

# NOTE(cmo): Load an atmosphere. In this case we include a copy of FAL C, but
# Lightweaver also supports loading atmospheres in the MULTI format, and it is
# also simple to do so from the raw data components
atmosRef = Falc82()
# NOTE(cmo): Ca II 8542 with the reference electron density in the FAL C atmosphere
IwaveRef = synth_8542(atmosRef, conserve=False, useNe=True)

atmosCons = Falc82()
# NOTE(cmo): Ca II 8542 with the electron density obtained from charge conservation
IwaveCons = synth_8542(atmosCons, conserve=True, useNe=False)

atmosLte = Falc82()
# NOTE(cmo): Ca II 8542 with LTE electron density
IwaveLte = synth_8542(atmosLte, conserve=False, useNe=False)
```

**Figure 2.** Simple program comparing the results for Ca II 8542 Å with different electron densities.

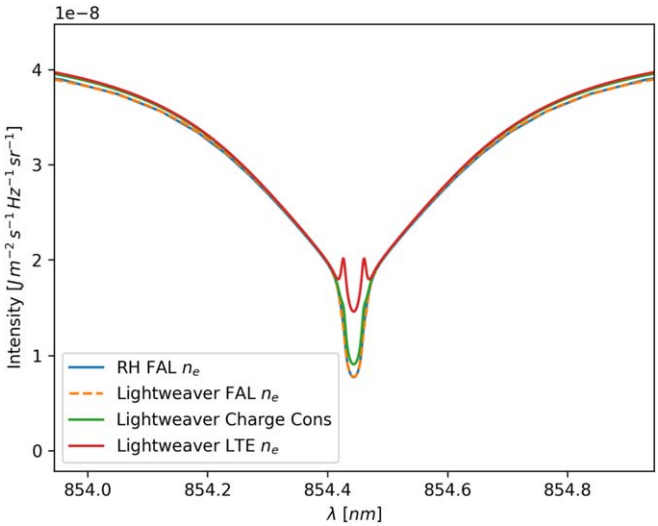


**Figure 3.** Comparison between RH and *Lightweaver* for the Ca II 8542 Å line with different electron density solutions.

between nodes in a single block if desired (although messages this large are taxing on process interconnects, and in many cases it is simpler to simply ship the data necessary to reconstruct the `Context` on a different node).

### 3.6. Example

The code in Figure 2 presents a simple script for running the comparison between the line profiles obtained for Ca II 8542 Å with different electron densities in the FAL C atmosphere (Fontenla et al. 1993).

These are plotted against the RH solution for the electron density given in FAL C in Figure 3.

The agreement between RH and *Lightweaver* is extremely good when solving the same problem, as shown by the blue and dashed orange curves. The red curve, using LTE electron density, shows the importance of the correct electron density for Ca II 8542 Å. The green curve, which is the solution computed with charge conservation from an LTE starting solution, approaches the reference electron density solution, and represents a self-consistent solution for electron density taking into account H and Ca in NLTE; however, the final differences between the charge conserved solution and the reference solution are probably due to other elements, such as Fe, being treated in LTE.

### 3.7. Advanced Example

In this section we present a more advanced example, first demonstrating the implementation of a different line profile (in this case Doppler) in a Ca II model atom, and then using this modified model in a program that reprocesses output from the RADYN (Carlsson & Stein 1992; Allred et al. 2015) radiation–hydrodynamic code in a time-dependent fashion.

Figure 4 demonstrates the modification of a five-level + continuum Ca II atom to use Doppler line profiles. The `DopplerLine` class is first defined, with a new implementation of the `compute_phi` method expected on an instance of `AtomicLine`. It is then necessary to define the `NoOpBroadener` for the `LineBroadening` object provided to these lines; this class does nothing but provide a comparison against itself and a `__repr__` method, allowing the model atom to be constructed from `repr` as discussed in Section 3.1.

Finally, the model atom is constructed as before, but using the newly defined `DopplerLine` class. In this way model atoms can contain features such as different line profiles and collision rate parameterizations that are not known to the core *Lightweaver* package but remain compartmentalized in user code.

Figure 5 shows the small amount of code needed to construct a specialized program for synthesizing radiation from a preprocessed RADYN simulation (where the thermodynamic parameters of the atmosphere have been interpolated onto a fixed spatial grid) in a time-dependent fashion. The simple method presented here ignores the advection of the populations by the plasma flows, but updates the calcium populations in a time-dependent fashion. The hydrogen populations are loaded from the RADYN output and are used directly in the "detailed static" mode of operation. Combining the Ca II atom with Doppler line profiles from Figures 4 and 5 it is easy to perform this same synthesis twice, once with the traditional Voigt profiles and once with the Doppler profiles. The code present in these listings with the data file in the associated repository (Osborne 2021b) are all that is needed to run this simulation.

First, the preprocessed data are loaded and an atmosphere object `atmos` is constructed from the initial timestep of the data. Several functions are then defined.

1. `construct_context_for` constructs a `Context` for an atmosphere and collection of model atoms, similarly to Figure 2.
2. `initial_stat_eq` computes the statistical equilibrium solution using this context similarly to `iterate_ctx` in Figure 2.
3. `load_step` loads the thermodynamic atmospheric properties, and hydrogen populations from the chosen timestep into the `Context`, before recomputing the line profiles and background opacities via `ctx.update_deps`.
4. `compute_time_dependent_profiles` then uses the `load_step` to load each timestep of the data present, solve the RT problem, and advance the calcium populations in time. It returns a `list` of the outgoing radiation from each timestep in the data.

```python
import numpy as np
import pickle
from dataclasses import dataclass
import matplotlib.pyplot as plt
import lightweaver as lw
from copy import deepcopy
from lightweaver.rh_atoms import H_6_atom, C_atom, O_atom, Si_atom, Al_atom, CaII_atom, Fe_atom,
    He_atom, Mg_atom, N_atom, Na_atom, S_atom
from lightweaver.atomic_model import AtomicLine, LineProfileState, LineProfileResult, AtomicModel,
    LineType, LinearCoreExpWings
from lightweaver.broadening import LineBroadener, LineBroadening

@dataclass(eq=False, repr=False)
class DopplerLine(AtomicLine):
    def compute_phi(self, state: LineProfileState) -> LineProfileResult:
        vBroad = self.atom.vBroad(state.atmos) if state.vBroad is None else state.vBroad
        xBase = (state.wavelength - self.lambda0) * lw.CLight / self.lambda0
        phi = np.zeros((state.wavelength.shape[0], state.atmos.Nrays, 2, state.atmos.Nspace))

        for mu in range(state.vlosMu.shape[0]):
            for toObs, sign in enumerate([-1.0, 1.0]):
                for k in range(state.atmos.Nspace):
                    xk = (xBase + sign * state.vlosMu[mu, k]) / vBroad[k]
                    phi[:, mu, toObs, k] = np.exp(-xk**2) / (np.sqrt(np.pi) * vBroad[k])

        return LineProfileResult(phi=phi, aDamp=np.zeros_like(vBroad),
                                 Qelast=np.zeros_like(vBroad))

@dataclass
class NoOpBroadener(LineBroadener):
    def __eq__(self, other):
        if type(self) is type(other):
            return True
        return False

    def __repr__(self):
        return 'NoOpBroadener()'

PureDopplerBroadening = lambda: LineBroadening(natural=[NoOpBroadener()], elastic=[NoOpBroadener()])
CaIIDoppler = lambda: AtomicModel(element=lw.PeriodicTable['Ca'],
                                  levels=[level for level in CaII_atom().levels],
                                  lines=[
                                     DopplerLine(j=3, i=0, f=3.412e-01, type=LineType.CRD,
                                         quadrature=LinearCoreExpWings(qCore=30, qWing=1500,
                                         Nlambda=50), broadening=PureDopplerBroadening()),
                                     DopplerLine(j=4, i=0, f=6.807e-01, type=LineType.CRD,
                                         quadrature=LinearCoreExpWings(qCore=30, qWing=1500,
                                         Nlambda=50), broadening=PureDopplerBroadening()),
                                     DopplerLine(j=3, i=1, f=5.956e-02, type=LineType.CRD,
                                         quadrature=LinearCoreExpWings(qCore=4, qWing=200, Nlambda=40),
                                         broadening=PureDopplerBroadening()),
                                     DopplerLine(j=4, i=1, f=1.219e-02, type=LineType.CRD,
                                         quadrature=LinearCoreExpWings(qCore=4, qWing=150, Nlambda=40),
                                         broadening=PureDopplerBroadening()),
                                     DopplerLine(j=4, i=2, f=7.242e-02, type=LineType.CRD,
                                         quadrature=LinearCoreExpWings(qCore=4, qWing=200, Nlambda=80),
                                         broadening=PureDopplerBroadening())
                                  ],
                                  continua=[cont for cont in CaII_atom().continua],
                                  collisions=[col for col in CaII_atom().collisions])
```

**Figure 4.** Configuring a Ca II model atom with Doppler profiles.

Finally these functions are applied twice to construct two different simulations, one for each of the different calcium model atoms used.

A complete reprocessing of the RADYN simulation, considering the effects of advection of the atomic populations is substantially more complex, and outside the scope of the core *Lightweaver* framework. However, after implementing a suitable advection scheme the code presented here could easily be adapted.

### 3.8. Performance Comparison

Taking the FAL C example (with given electron density) presented in Section 3.6, for a five-level + continuum model

```
with open('RadynF9NoIncRad1sData.pickle', 'rb') as pkl:
    radynData = pickle.load(pkl)

atmos = lw.Atmosphere.make_1d(scale=lw.ScaleType.Geometric, depthScale=radynData['z'],
                              temperature=radynData['temperature'][0], vlos=radynData['vel'][0],
                              vturb=np.ones_like(radynData['temperature'][0]) * 2e3,
                              ne=radynData['ne'][0], nHTot=radynData['nh'][0].sum(axis=0))
atmos.quadrature(5)

def construct_context_for(atmos, *atoms):
    aSet = lw.RadiativeSet(atoms)
    aSet.set_active('Ca')
    aSet.set_detailed_static('H')
    spect = aSet.compute_wavelength_grid()
    eqPops = aSet.compute_eq_pops(atmos)
    eqPops.atomicPops['H'].n[...] = radynData['nh'][0]
    ctx = lw.Context(atmos, spect, eqPops, Nthreads=16)
    return ctx

def initial_stat_eq(ctx, popsTol=1e-3):
    for i in range(500):
        dJ = ctx.formal_sol_gamma_matrices()
        if i < 5:
            continue
        dPops = ctx.stat_equil()
        if dPops < popsTol:
            print('Initial Statistical Equilibrium after %d iterations' % i)
            break

def load_step(ctx, stepIdx):
    atmos = ctx.kwargs['atmos']
    eqPops = ctx.eqPops
    atmos.temperature[:] = radynData['temperature'][stepIdx]
    atmos.ne[:] = radynData['ne'][stepIdx]
    atmos.vlos[:] = radynData['vel'][stepIdx]
    eqPops.atomicPops['H'].n[...] = radynData['nh'][stepIdx]
    ctx.update_deps()

def compute_time_dependent_profiles(ctx):
    I = []
    for t in range(radynData['time'].shape[0] - 1):
        dt = radynData['dt'][t+1]
        prevTimePops = None
        for it in range(200):
            ctx.formal_sol_gamma_matrices()
            dPops, prevTimePops = ctx.time_dep_update(dt, prevTimePops)
            if dPops < 1e-3 and it > 2:
                print('--- Step %d done ---' % t)
                break
        I.append(np.copy(ctx.spect.I))
        load_step(ctx, t+1)
    return I

ctx = construct_context_for(atmos, H_6_atom(), CaII_atom(), C_atom(), O_atom(),
                                   Si_atom(), Al_atom(), Fe_atom(), He_atom(),
                                   Mg_atom(), N_atom(), Na_atom(), S_atom())
atmosDoppler = deepcopy(atmos)
ctxDoppler = construct_context_for(atmosDoppler, H_6_atom(), CaIIDoppler(), C_atom(), O_atom(),
                                                 Si_atom(), Al_atom(), Fe_atom(), He_atom(),
                                                 Mg_atom(), N_atom(), Na_atom(), S_atom())



initial_stat_eq(ctx)
I = compute_time_dependent_profiles(ctx)

initial_stat_eq(ctxDoppler)
IDoppler = compute_time_dependent_profiles(ctxDoppler)
```

**Figure 5.** Performing a basic time-dependent synthesis from RADYN simulation with and without Doppler line profiles in the Ca II model atom.

**Table 2**
Single-threaded Comparison between RH and *Lightweaver* for a FAL C Atmosphere with Both H and Ca Active

| Configuration | Time (s) |
|---|---|
| RH wall time | 8.81 |
| RH setup and iteration only | 8.79 |
| *Lightweaver* wall time | 11.05 |
| *Lightweaver* setup and iteration only | 8.47 |

**Table 3**
Multithreaded Comparison between RH and *Lightweaver* for a FAL C Atmosphere with Both H and Ca Active

| Configuration | Time (s) |
|---|---|
| RH wall time | 7.37 |
| RH setup and iteration only | 7.35 |
| *Lightweaver* wall time | 5.32 |
| *Lightweaver* setup and iteration only | 2.49 |

hydrogen atom and a five-level + continuum model calcium atom we will compare the performance of *Lightweaver* with RH. These comparison tests were run on an Intel Xeon E3-1270 v3 (four cores/eight logical threads, Haswell micro-architecture) with 1600 MHz DDR3 memory inside the Windows Subsystem for Linux environment in Microsoft Windows 10.0.18363.1016. The compiled components of both codes were compiled with the GNU compiler collection 7.5.0, and *Lightweaver* was run using Python 3.8.2.

In both codes the atomic populations were initialized to LTE and Ng acceleration was disabled, to allow direct comparison of the iteration speed. All tests were run five times, and the final result is the mean of these. The variability from run-to-run is extremely low, so is not shown here.

The single-threaded results are shown in Table 2. The difference the total wall time (real-world elapsed time) and the "setup and iteration only" time is the time taken to load and correctly configure the model atmosphere. The FAL C model used is defined on a column mass stratification. Both RH and *Lightweaver* work in geometric height, so this stratification must first be converted. This is a simple procedure with the total hydrogen density specified in the model atmosphere file. To make the model atmosphere easy to manipulate on its own in *Lightweaver*, the continuum optical depth $\tau_{500}$ is also evaluated at the same time. In RH this step takes place after the background opacities are computed, and these can be used directly (hence the very low cost of this step). To improve flexibility in *Lightweaver*, this term uses background opacities obtained from the equation of state package discussed in Section 2.9.3. Many improvements could be made to the speed of this package by reimplementing its most numerically costly functions in a more performant language (or perhaps binding the pre-existing FORTRAN version to Python). In practice, this one-off cost is rarely an issue as many models are now specified in terms of height, and *Lightweaver* does not require the calculation of column mass and continuum optical depth when a height stratified atmospheric definition is provided.

The codes were also compared when running on multiple threads. In this case eight threads were used in both codes, as this provided the fastest execution on this system. These results are shown in Table 3. The constant cost of the $\tau_{500}$ conversion in *Lightweaver* can again be seen in these results. *Lightweaver*'s threading model, which uses a thread-pool and lockless accumulation of the Gamma operator, provides significantly faster results at the cost of slightly higher memory consumption (one copy of the $\Gamma$ matrix per atom per thread, and one copy of the accumulation terms (see Appendix B) per atom per thread). Ignoring the aforementioned expensive one-off cost of computing $\tau_{500}$ for the model atmosphere via the equation of state, RH achieves a 1.2× speedup from utilizing multiple threads, whereas *Lightweaver* achieves a 3.4× speedup. Accounting for the high cost of RH's threading model on Windows (where thread creation is very costly), this test was also run on a computer running CentOS 7, where a maximum speedup of 1.5× was recorded.

## 4. Conclusions

We have presented a brief overview of NLTE RT, and the methods used to solve associated problems employed by the *Lightweaver* framework. We have also discussed the design of the framework, and hope that it will allow simpler experimentation with RT methods due to its "factoring out" of common operations into composable building blocks, and providing a single-language approach to running and analyzing simulations thanks to the extensive pre-existing set of scientific tools available in Python. The nature of the framework allows programs for specialized tasks to be written far more easily than is possible in the traditional "configuration file"-based monolithic code. We are currently working on multidimensional extensions to the framework, to allow the synthesis of radiation from 2D and simple 3D atmospheres, but do not anticipate applying the advanced domain decomposition techniques of, e.g., Multi3d (Leenaarts & Carlsson 2009) or PORTA (Štěpán & Trujillo Bueno 2013); however, such an extension would be relatively simple thanks to the modularity of the codebase and simple serialization of `Context` state.

C.M.J.O. acknowledges support from the UK's Science and Technology Facilities Council (STFC) doctoral training grant ST/R504750/1 and is grateful for the financial aid of the STFC, CU Boulder, and the National Solar Observatory (NSO) for allowing the research trip to NSO where *Lightweaver* was designed. The authors are also grateful to the reviewer for helpful comments and suggestions to improve the manuscript.

*Lightweaver* builds on the huge efforts of the scientific Python community and we acknowledge the NumPy (Harris et al. 2020), SciPy (Virtanen et al. 2020), Matplotlib (Hunter 2007), Cython (Behnel et al. 2011), and Astropy (Robitaille et al. 2013; Price-Whelan et al. 2018) packages.

## Appendix A
## Line Broadening

In the following we describe how the different broadening terms arise and are implemented in *Lightweaver* as the basis for the standard Voigt line profile. Natural broadening arises due to the finite lifetime of atomic states (described by $A_{ji}$) and the consequent variation in transitional energy due to the Heisenberg uncertainty principle. The energy of an atomic transition is no longer perfectly defined, and instead takes the form of a Lorentzian distribution with damping coefficient $\Gamma_{\rm rad} = \sum_{i<j} A_{ji}$. This description only accounts for broadening due to spontaneous emission, and a strong radiation field can modify this. It may sometimes be useful to tune this parameter

to account for observations, and lines that are not present in a simplified model atom; $\Gamma_{\rm rad}$ is therefore a free parameter for each line in *Lightweaver*.

Doppler broadening is due to the random thermal motions of particles within the plasma. In *Lightweaver*, we take the broadening velocity to be $v_{\rm broad} = \sqrt{2k_{\rm B}T/W + v_{\rm turb}^2}$, where $W$ is the particle's mass, and $v_{\rm turb}$ is the microturbulent velocity specified for the atmosphere in question. As Doppler broadening produces a Gaussian line profile, and radiative broadening produces a Lorentzian, the line profile due to both of these effects is the convolution of these, known as a Voigt profile. Now, normalizing with respect to the Doppler width

$$\Delta\nu_D = \frac{v_{\rm broad}\nu_{ij}}{c}, \tag{A1}$$

we have the absorption profile

$$\varphi_{ij}(x) = \frac{H(a_{\rm damp}, x)}{\sqrt{\pi}}, \tag{A2}$$

where

$$x = \frac{\nu - \nu_{ij}}{\Delta\nu_D}, \tag{A3}$$

and

$$a_{\rm damp} = \frac{\Gamma}{4\pi\Delta\nu_D}. \tag{A4}$$

$\Gamma$ is the sum of all "typical" broadenening terms (radiative, Stark, and van der Waals), and $H(a, x)$ is the Voigt function ($\int_{-\infty}^{\infty} H(a, x)dx = \sqrt{\pi}$). The normalization is such that

$$\frac{1}{4\pi}\oint\int \varphi_{ij}(x, \boldsymbol{d})dx\, d\Omega = 1 \tag{A5}$$

when accounting for a directionally varying line profile. The line profile $\varphi$ describes the photon absorption probability at a certain position in Doppler units. To be dimensionally consistent with an integration over frequency we define the frequency dependent line profile $\phi_{ij}$ such that

$$\int_0^{\infty} \phi_{ij}\, d\nu = 1 \tag{A6}$$

i.e.,

$$\phi_{ij} = \frac{\varphi_{ij}}{\Delta\nu_D}. \tag{A7}$$

Similarly to the line absorption profile, the properties of a continuum are controlled by the atomic cross-section at a given frequency. The cross-sections can be described in two different ways in *Lightweaver*: either as a *hydrogenic* continuum, in which case the cross-section falls off as $1/\nu^3$ with increasing frequency from the continuum edge, or as a tabulated cross-section, whereby the cross-section is provided at different wavelengths, and then interpolated between.

Note that the quantity stored in the variable `phi` in the code is in fact $\varphi/v_{\rm broad}$, as this simplifies construction of some of the expressions. We denote this term $\phi_{\rm num}$, which is used in Appendix B when describing the calculation of emissivity and opacity.

## Appendix B
## MALI Numerical Implementation

In the following the integration and accumulation terms used in the implementation of the MALI method (Section 2.3) are presented.

As discussed in Section 2.3.1 a common wavelength grid is first computed from which the final individual wavelength grid for each transition is extracted. For each transition with its final individual discrete wavelength grid (denoted $\lambda$) we define the integration weights

$$w_{\lambda_a} = \begin{cases} 0.5(\lambda_{a+1} - \lambda_a)D_{ij}, & a = a_{\rm min} \\ 0.5(\lambda_a - \lambda_{a+1})D_{ij}, & a = a_{\rm max} \\ 0.5(\lambda_{a+1} - \lambda_{a-1})D_{ij}, & \text{otherwise}, \end{cases} \tag{B1}$$

with

$$D_{ij} = \begin{cases} c/\lambda_{ij} & \text{bound} - \text{bound} \\ 1 & \text{bound} - \text{free}. \end{cases} \tag{B2}$$

For lines, this can be used to compute the line-profile normalization factor

$$w_{\phi,k} = \left(\sum_{i,\mu} \phi_{{\rm num},\lambda_i,\boldsymbol{d}} w_{\lambda_i} w_\mu\right)^{-1} \tag{B3}$$

which ensures that (A5) holds for each discretized point in the atmosphere $k$. $\phi_{\rm num}$ is defined in Appendix A. This line-profile normalization factor is essential in ensuring that $\Gamma$ is correctly scaled, especially for more sparsely sampled lines. The integration weights for the terms contributing to $\Gamma^R$ for each transition $ij$ at each depth are then

$$w_{\Gamma,ij,\lambda_a} = \begin{cases} w_{\lambda_a} w_\phi 4\pi/(hc), & \text{bound} - \text{bound} \\ w_{\lambda_a} 4\pi/(h\lambda_a), & \text{bound} - \text{free} \end{cases}. \tag{B4}$$

These terms may not be immediately evident upon comparison with (27); the differences arise from ensuring that all terms are integrated in frequency, despite the discretization being performed in wavelength.

From the previous discussion of line profiles in Appendix A, we have

$$\frac{h\nu}{4\pi}\phi_{ij} = \frac{hc}{4\pi}\phi_{{\rm num},ij} \tag{B5}$$

under the reasonable approximation that $\nu = \nu_{ij}$ over the integration range for a spectral line. This latter formulation is used when evaluating $U$ and $V$ in *Lightweaver*. Additionally, we follow Uitenbroek (2001) and define

$$g_{ij} = \begin{cases} g_i/g_j\rho_{ij} = B_{ji}/B_{ij}\rho_{ij} & \text{bound} - \text{bound} \\ n_i^*/n_j^* \exp\left(-\dfrac{h\nu}{k_{\rm B}T}\right) & \text{bound} - \text{free}, \end{cases} \tag{B6}$$

such that

$$V_{ji} = g_{ij}V_{ij}, \tag{B7}$$

and

$$U_{ji} = \frac{2h\nu^3}{c^2}V_{ji}, \tag{B8}$$

which can be expressed as $U_{ji} = A_{ji}/B_{ji}V_{ji}$ for bound–bound transitions. With these expressions the $U$ and $V$ terms are very efficient to evaluate in a vectorized manner, and we therefore choose to not cache them. If one were writing an entirely CRD code, these terms could be computed once per transition and stored, which could be a valuable optimization in some cases, at the expense of computer memory.

During the accumulation of emissivities and opacities at each wavelength and direction for the formal solver we also accumulate the emissivity per atom

$$\mathcal{H} = \sum_{i,j} \eta^{\dagger}_{ij}, \tag{B9}$$

total $U_{ji}$ per level $j$

$$\mathcal{U}_j = \sum_i U^{\dagger}_{ji}, \tag{B10}$$

and effective self-opacity in for each level $l$

$$\mathcal{X}_l = \sum_{j>l} \chi^{\dagger}_{lj} - \sum_{i<l} \chi^{\dagger}_{il}. \tag{B11}$$

If no lines are associated with a wavelength point, then all the sources of emissivity and opacity are direction independent, and these accumulations can only be performed once for all of the directional formal solutions. As the accumulation of terms into $\Gamma^R$ can be done after the formal solution for each direction is performed, the storage of these terms does not increase with the number of directions. If one adopts the "same-transition" preconditioning approach of Rybicki & Hummer (1992) then none of these accumulation arrays are needed. This could be advantageous in the implementation of higher-dimensional schemes, as in most cases there appears to be no difference in convergence speed between the two methods.

The formal solver then provides the values of $I^{\dagger}(\nu, \boldsymbol{d})$ and $\Psi^*_{\nu,\boldsymbol{d}}$, for all depths, one direction at a time. The per-atom $I^{\rm eff}$ term of (28) for this direction can then simply be computed as

$$I^{\rm eff}_{\nu,\boldsymbol{d}} = I^{\dagger}(\nu, \boldsymbol{d}) - \Psi^*_{\nu,\boldsymbol{d}}\mathcal{H}, \tag{B12}$$

where the final term is a simple scalar multiplication, under the assumption that $\Psi^*$ is simply the diagonal of the true $\Psi$ operator (which is currently the case in *Lightweaver*).

With these definitions the integration of the off-diagonal entries of $\Gamma^R$ at each spatial point, for each active atom, is performed by looping over each contributing transition $ij$ such that

$$\Gamma^R_{ij} = \sum_{a,\mu} [w_\mu w_{\Gamma,ij,\lambda_a}(U_{ji} + V_{ji}I^{\rm eff} - \mathcal{X}_i\Psi^*\mathcal{U}_j)], \tag{B13}$$

$$\Gamma^R_{ji} = \sum_{a,\mu} [w_\mu w_{\Gamma,ij,\lambda_a}(V_{ij}I^{\rm eff} - \mathcal{X}_j\Psi^*\mathcal{U}_i)]. \tag{B14}$$

The radiative rates are computed similarly

$$R_{ij} = \sum_{a,\mu} [w_\mu w_{\Gamma,ij,\lambda_a} I^{\dagger} V_{ij}], \tag{B15}$$

$$R_{ji} = \sum_{a,\mu} [w_\mu w_{\Gamma,ij,\lambda_a}(U_{ji} + I^{\dagger} V_{ij})]. \tag{B16}$$

## Appendix C
## PRD Implementation

In the following we describe the terms needed to apply PRD to a spectral line and their implementation in *Lightweaver*. For the previous definition of $\rho_{ij}$ from (18), under the assumptions of a line with an infinitely sharp lower level and broadened upper level, and the validity of PRD being in the atomic frame being approximated by PRD in the observer's frame (Uitenbroek 2001), following Hubený & Mihalas (2014) we then have

$$\rho_{ij}(\nu, \boldsymbol{d}) = 1 + \gamma \frac{\sum_{l<j} n_j B_{lj}}{n_j P_j} \oint \frac{1}{4\pi} \int I(\nu', \boldsymbol{d}') \cdot \left[ \frac{R^{II}_{lji}(\nu', \boldsymbol{d}'; \nu, \boldsymbol{d})}{\phi_{ij}(\nu, \boldsymbol{d})} - \phi_{lj}(\nu', \boldsymbol{d}) \right] d\nu'\, d\Omega', \tag{C1}$$

where $R^{II}$ is the generalized redistribution function for transitions of this kind (Hubený 1982), and $\gamma$ is the branching ratio, or coherency fraction. The summation over $l$ and $lji$ subscript on $R^{II}$ describe the scattering process. When ignoring cross-redistribution (Raman scattering), we have $l = i$, and the summation is replaced by a single term, as we are only considering resonance scattering within the line $i \to j$.

The coherency fraction $\gamma$ describes the normalized probability of a photons being re-emitted from the same sublevel of energy level $j$ before an elastic collision that will redistribute it across sublevels, provided that it is re-emitted at all. This is then given by

$$\gamma = \frac{P_j}{P_j + Q_j}, \tag{C2}$$

where $P_j$ is the total rate of transitions out of level $j$ (depopulation rate), and $Q_j$ is the total rate of elastic collisions affecting this level.

Defining $g_{II}(\nu, \nu') = R^{II}(\nu, \nu')/\phi_{ij}(\nu')$, which is normalized such that

$$\frac{1}{4\pi} \oint \int g_{II}(\nu, \nu')\, d\nu'\, d\Omega = 1, \tag{C3}$$

as per Gouttebroze (1986) and Uitenbroek (1989) wherein fast approximations to this function are derived, and ignoring cross-redistribution, we then have

$$\rho_{ij}(\nu, \boldsymbol{d}) = 1 + \gamma \frac{n_i B_{ij}}{n_j P_j} \oint \frac{1}{4\pi} \int I(\nu', \boldsymbol{d}') \cdot [g_{II}(\nu, \nu') - \phi_{ij}(\nu', \boldsymbol{d})]\, d\Omega'\, d\nu'. \tag{C4}$$

Ignoring bulk plasma flows, the integrals over angle and frequency can then be split, providing an angle-averaged form of $\rho$ that is much easier to compute, given by

$$\rho_{ij}(\nu) = 1 + \gamma \frac{n_i B_{ij}}{n_j P_j} \left( \int g_{II}(\nu, \nu') J(\nu') d\nu' - \bar{J}_{ij} \right), \tag{C5}$$

where

$$\bar{J}_{ij} = \frac{1}{4\pi} \oint \int I(\nu, \boldsymbol{d}) \phi(\nu, \boldsymbol{d}) d\nu\, d\Omega = \frac{R_{ij}}{B_{ij}} \tag{C6}$$

is the frequency-integrated mean intensity across the transition.

The numerical implementation of angle-averaged PRD simply follows the method of Uitenbroek (2001), which is briefly summarized below. The redistribution function is often very sharply peaked; an accurate evaluation of the scattering integral therefore requires much finer sampling of the wavelength grid than that which is required to evaluate the terms in $\Gamma^R$. However for each wavelength in a line's grid there is only a small surrounding region for which $g_{II}$ is non-zero. $J(\nu)$ is then interpolated onto a fine grid over this region, over which (C4) is trivially implemented by an application of Simpson's rule. It is essential that the scattering integral component of (C4) be normalized as per (C3) to avoid the addition or destruction of photons in the transition. The term

$$\int g_{II}(\nu, \nu')J(\nu')d\nu' \tag{C7}$$

is then implemented as

$$\frac{\sum_i g_{II}(\nu, \nu'_i)J(\nu'_i)\delta\nu'_i}{\sum_i g_{II}(\nu, \nu'_i)\delta\nu'_i}, \tag{C8}$$

where $\delta\nu'_i$ are the integration weights over this fine grid.

The iterative method currently employed consists of performing a formal solution over the wavelengths where PRD lines are active, and updating $\rho$ using this method *while maintaining the populations fixed*. When solving a PRD problem, a number of these sub-iterations to update $\rho$ (commonly three) are interleaved between every complete formal solution and population update.

For the hybrid PRD case of Leenaarts et al. (2012), used when plasma flows exceed the thermal Doppler velocity, $J(\nu)$ in (C5) is then replaced by $J_{\text{rest}}(\nu)$, the mean intensity in the atom's rest frame. This approximation is much faster to evaluate than the full angle-dependent case, as the accumulation of $J_{\text{rest}}(\nu)$ can be done during the formal solution, using a linear interpolation off the Doppler-shifted frequency grid. $\rho$ can be linearly interpolated from the atomic rest frame during the calculation of the $U$ and $V$ terms, or into a directionally dependent array at the end of each PRD sub-iteration. Currently *Lightweaver* does the former of these.

To ensure that $J_{\text{rest}}$ is accumulated correctly during the PRD sub-iterations, we no longer simply perform formal solutions over wavelengths where PRD lines are present, but also over wavelengths that when shifted back to the rest frame contribute to $J_{\text{rest}}$ in these regions. We have found that this modification, which is not present in RH1.5D (Pereira & Uitenbroek 2015), can dramatically aid convergence in atmospheres with high velocity shifts.

## Appendix D
## Self-consistent Newton–Raphson Electron Density Iteration

From Section 2.3, the equations of statistical equilibrium are given by (12). Let us write this system for a level $i$ of a species $s$ as

$$F_{s,i}(\boldsymbol{n}_s, n_e) = \sum_{j\neq i} n_j P_{ji}(\boldsymbol{n}_s, n_e) - n_i \sum_{j\neq i} P_{ij}(\boldsymbol{n}_s, n_e) = 0. \tag{D1}$$

With a fixed electron density the preconditioned linear formulation of these equations for a species $s$ can be written

$$\Gamma_s \boldsymbol{n}_s = \boldsymbol{0}. \tag{D2}$$

We start by obtaining the solution to this linear system that, in the following, will be denoted $\widetilde{\boldsymbol{n}}_s$. The previous values of these populations, i.e., the ones at which $\Gamma_s$ was evaluated, will once again be denoted with †.

The principle of Newton–Raphson iteration is to compute $F(x_0 + \delta x) = 0$, which can be written as $F(x_0) + J\delta x = 0$, with $J$ the Jacobian of $F$ evaluated at $x_0$, and $\delta x$ some small correction to $x_0$. This can be rearranged to $-J\delta x = F(x_0)$. Applying this to technique (D1) and explanding to first order we have

$$\begin{aligned} &-\sum_j \left( \frac{\partial F_{s,i}(\boldsymbol{n}_s, n_e)}{\partial n_j}\bigg|_{(\widetilde{\boldsymbol{n}}_s, n_e^\dagger)} \delta n_j \right) \\ &- \frac{\partial F_{s,i}(\boldsymbol{n}_s, n_e)}{\partial n_e}\bigg|_{(\widetilde{\boldsymbol{n}}_s, n_e^\dagger)} \delta n_e = F_{s,i}(\widetilde{\boldsymbol{n}}_s, n_e^\dagger), \end{aligned} \tag{D3}$$

where $\delta n_j$ and $\delta n_e$ indicate the corrections to these populations necessary to render them self-consistent. This expression can be written for each level of each active species. Looking more closely at each term we have

$$\frac{\partial F_{s,i}(\boldsymbol{n}_s, n_e)}{\partial n_j}\bigg|_{(\widetilde{\boldsymbol{n}}_s, n_e^\dagger)} = \Gamma_{s,ij}, \tag{D4}$$

and

$$\begin{aligned} \frac{\partial F_{s,i}(\boldsymbol{n}_s, n_e)}{\partial n_e}\bigg|_{(\widetilde{\boldsymbol{n}}_s, n_e^\dagger)} &= \sum_j \left( \frac{\partial \Gamma^C_{s,ij}}{\partial n_e}\bigg|_{(\widetilde{\boldsymbol{n}}_s, n_e^\dagger)} \widetilde{n}_j \right) \\ &+ \sum_j \begin{cases} \Gamma^R_{s,ij}\widetilde{n}_j/n_e^\dagger, & i \to j \quad \text{bound} - \text{free} \\ 0, & \text{otherwise}, \end{cases} \end{aligned} \tag{D5}$$

where the term involving $\Gamma^C$ depends on the exact form of the collisional rates. Due to the number of collisional rate options available in *Lightweaver* this is evaluated through finite differences, which remains relatively efficient due to the local nature of this term.

As can be seen from (D4) and (D5), all terms are linear in $\delta n$ and $\delta n_e$; however, additional constraints are needed to close this system: a constraint on the total population of each species $s$, and a constraint on charge neutrality. In total, this forms a system of $\sum_s N_{\text{level},s} + 1$ equations, where $N_{\text{level},s}$ is the number of levels treated in detail for species $s$, and the summation is performed over all active species. This system can therefore be written at each point in the atmosphere as a block diagonal matrix, with each block being $N_{\text{level},s} \times N_{\text{level},s}$, with a final row and column due to the electron density terms and charge conservation equation that couple all blocks. The block terms are simply $-\Gamma_s$ for each species, and the final column for level $j$ of species $s$ is the additive inverse of the right-hand side of (D5). In our implementation the population conservation equation is

$$\sum_j \delta n_j = n_{\text{total}} - \sum_j \widetilde{n}_j, \tag{D6}$$

where $n_{\text{total}}$ is the total population of the species (derived from abundance and hydrogen density, or mass density). The left-hand side amounts to placing a block row of ones in the Jacobian for all levels in the species. In our case we replace the last equation for each species with this population conservation equation to avoid degeneracies.

The charge conservation equation is given by

$$\delta n_e - \sum_s \sum_j \mathrm{ion}_s(j)\delta n_{s,j} = n_{e,\mathrm{bg}} + \sum_s \sum_j \mathrm{ion}_s(j) - n_e^\dagger, \tag{D7}$$

where $\mathrm{ion}_s(j)$ is the ionization level of the $j$th level of species $s$, and $n_{e,\mathrm{bg}}$ is the electron density due to background species whose populations are not otherwise taken into account during this iteration. The left-hand side is inserted into the final row of the Jacobian.

The right-hand side vector for the Newton–Raphson procedure, where not specified by the constraint equations, is given by $\Gamma_s \widetilde{\boldsymbol{n}_s}$. This system can be solved as a typical matrix-vector system to obtain the corrections. Finally, the populations are corrected as $n = \tilde{n} + \delta n$ and $n_e = n_e^\dagger + \delta n_e$, and all LTE populations must be updated for consistency with the new electron density. As hydrogen is by far the dominant contributor of electrons, we optionally allow the above to only operate on hydrogen, and count all other species as background for the purpose of updating $n_e$, and in some cases this may be more stable.

The time-dependent case then follows a similar derivation to the statistical equilibrium case. Here we start from the $\theta$-method of (33) and similarly to (D1) define

$$G_{s,i}(\boldsymbol{n}_s^{t+1}, n_e) = n_{s,i}^{t+1} - \theta\Delta t F_{s,i}(\boldsymbol{n}_s^{t+1}, n_e) - (1-\theta)\Delta t \Gamma_s^t \boldsymbol{n}_s^t - n_{s,i}^t = 0. \tag{D8}$$

Similarly to (D4) we then have

$$\left.\frac{\partial G_{s,i}(\boldsymbol{n}_s^{t+1}, n_e)}{\partial n_j}\right|_{(\widetilde{\boldsymbol{n}_s^{t+1}}, n_e^\dagger)} = \delta_{ij} - \theta\Delta t \Gamma_{s,ij}, \tag{D9}$$

where $\delta_{ij}$ is the Kronecker delta, and then similarly to (D5)

$$\left.\frac{\partial G_{s,i}(\boldsymbol{n}_s^{t+1}, n_e)}{\partial n_e}\right|_{(\widetilde{\boldsymbol{n}_s^{t+1}}, n_e^\dagger)} = -\theta\Delta t \left.\frac{\partial F_{s,i}(\boldsymbol{n}_s^{t+1}, n_e)}{\partial n_e}\right|_{(\widetilde{\boldsymbol{n}_s^{t+1}}, n_e^\dagger)}. \tag{D10}$$

The Jacobian matrix is constructed in the same way as the statistical equilibrium case, but using the derivatives of $G$. The right-hand-side vector of the Newton–Raphson iteration procedure is then given by (D8), where the superscript $t$ terms are known from the start of the timestep and need to be stored for use in this procedure. The constraint equations remain the same as in the time-independent case. As in the case of time-dependent population updates, *Lightweaver* currently only supports the $\theta = 1$ case, but the groundwork is present for supporting other $\theta$.

## ORCID iDs

Christopher M. J. Osborne https://orcid.org/0000-0002-2299-2800

## References


Allred, J. C., Kowalski, A. F., & Carlsson, M. 2015, ApJ, 809, 104
Arnaud, M., & Rothenflug, R. 1985, A&AS, 60, 425
Asplund, M., Grevesse, N., Sauval, A. J., & Scott, P. 2009, ARA&A, 47, 481
Bates, D. R. 1952, MNRAS, 112, 40
Behnel, S., Bradshaw, R., Citro, C., et al. 2011, CSE, 13, 31
Bell, K. L. 1980, JPhB, 13, 1859
Bjørgen, J. P., Leenaarts, J., Rempel, M., et al. 2019, A&A, 631, A33
Burgess, A., & Chidichimo, M. C. 1983, MNRAS, 1269
Carlsson, M., & Stein, R. F. 1992, ApJL, 397, L59
de la Cruz Rodríguez, J., Leenaarts, J., Danilovic, S., & Uitenbroek, H. 2019, A&A, 623, A74
de la Cruz Rodríguez, J., & Piskunov, N. 2013, ApJ, 764, 33
Fontenla, J., Avrett, E., & Loeser, R. 1993, ApJ, 406, 319
Geltman, S. 1962, ApJ, 136, 935
Gouttebroze, P. 1986, A&A, 195
Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Natur, 585, 357
Heinzel, P. 1995, A&A, 299, 563
Hubený, I. 1982, JQSRT, 27, 593
Hubený, I., & Mihalas, D. 2014, Theory of Stellar Atmospheres (Princeton, NJ: Princeton Univ. Press)
Hummer, D. G., & Rybicki, G. B. 1982, ApJ, 263, 925
Hummer, D. G., & Voels, S. A. 1988, A&A, 279
Hunter, J. D. 2007, CSE, 9, 90
Janett, G., Steiner, O., Alsina Ballester, E., Belluzzi, L., & Mishra, S. 2019, A&A, 624, A104
Janett, G., Steiner, O., & Belluzzi, L. 2018, ApJ, 865, 16
John, T. L. 1988, A&A, 193, 189
Judge, P. G. 2017, ApJ, 851, 5
Kašparová, J., Heinzel, P., Varady, M., & Karlický, M. 2003, in ASP Conf. Proc. 288, Stellar Atmosphere Modeling, ed. I. Hubeny, D. Mihalas, & K. Werner (San Francisco, CA: ASP), 544
Kowalski, A. F., Allred, J. C., Uitenbroek, H., et al. 2017, ApJ, 837, 125
Kurucz, R. L., van Dishoeck, E. F., & Tarafdar, S. P. 1987, ApJ, 322, 992
Leenaarts, J., & Carlsson, M. 2009, in ASP Conf. Ser. 415, The Second Hinode Science Meeting: Beyond Discovery-Toward Understanding, ed. B. Lites et al. (San Francisco, CA: ASP), 87
Leenaarts, J., Pereira, T., & Uitenbroek, H. 2012, A&A, 543, A109
Mihalas, D. 1978, Stellar Atmospheres (San Francisco, CA: Freeman)
Milić, I., & van Noort, M. 2018, A&A, 617, A24
Ng, K. C. 1974, JChPh, 61, 2680
Osborne, C. M. J. 2021a, Lightweaver v0.6.0, Zenodo, doi:10.5281/zenodo.4066860
Osborne, C. M. J. 2021b, Lightweaver Examples v1.0.0, Zenodo, doi:10.5281/zenodo.4762121
Osborne, C. M. J. 2021c, Weno4Interpolation v1.1.1, Zenodo, doi:10.5281/zenodo.4497941
Paletou, F. 1995, A&A, 302, 587
Pereira, T. M. D., & Uitenbroek, H. 2015, A&A, 574, A3
Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., et al. 2018, AJ, 156, 123
Robitaille, T. P., Tollerud, E. J., Greenfield, P., et al. 2013, A&A, 558, A33
Ruiz Cobo, B., & del Toro Iniesta, J. 1992, ApJ, 398, 375
Rybicki, G. B., & Hummer, D. G. 1992, A&A, 262, 209
Socas-Navarro, H., De La Cruz Rodríguez, J., Asensio Ramos, A., Trujillo Bueno, J., & Ruiz Cobo, B. 2015, A&A, 577, A7
Štěpán, J., & Trujillo Bueno, J. 2013, A&A, 557, A143
Stilley, J. L., & Callaway, J. 1970, ApJ, 160, 245
Trujillo Bueno, J., & Landi Degl'Innocenti, E. 1996, SoPh, 164, 135
Uitenbroek, H. 1989, A&A, 216, 310
Uitenbroek, H. 2001, ApJ, 557, 389
Viallet, M., Baraffe, I., & Walder, R. 2011, A&A, 531, A86
Victor, G. A., & Dalgarno, A. 1969, JChPh, 50, 2535
Virtanen, P., Gommers, R., Oliphant, T. E., et al. 2020, Nature Methods, 17, 261