Computer Physics Communications 220 (2017) 319–331

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

ARC: An open-source library for calculating properties of alkali Rydberg atoms☆

N. Šibalić [a,*], J.D. Pritchard [b], C.S. Adams [a], K.J. Weatherill [a]

[a] Joint Quantum Center (JQC) Durham-Newcastle, Department of Physics, Durham University, South Road, Durham, DH1 3LE, United Kingdom
[b] Department of Physics, SUPA, University of Strathclyde, 107 Rottenrow East, Glasgow, G4 0NG, United Kingdom

ARTICLE INFO

Article history:
Received 13 December 2016
Received in revised form 12 June 2017
Accepted 17 June 2017
Available online 5 August 2017

Keywords:
Alkali atom
Matrix elements
Dipole–dipole interactions
Stark shift
Förster resonances

ABSTRACT

We present an object-oriented Python library for the computation of properties of highly-excited Rydberg states of alkali atoms. These include single-body effects such as dipole matrix elements, excited-state lifetimes (radiative and black-body limited) and Stark maps of atoms in external electric fields, as well as two-atom interaction potentials accounting for dipole and quadrupole coupling effects valid at both long and short range for arbitrary placement of the atomic dipoles. The package is cross-referenced to precise measurements of atomic energy levels and features extensive documentation to facilitate rapid upgrade or expansion by users. This library has direct application in the field of quantum information and quantum optics which exploit the strong Rydberg dipolar interactions for two-qubit gates, robust atom-light interfaces and simulating quantum many-body physics, as well as the field of metrology using Rydberg atoms as precise microwave electrometers.

Program summary

Program Title: ARC: Alkali Rydberg Calculator
Program Files doi: http://dx.doi.org/10.17632/hm5n8w628c.1
Licensing provisions: BSD-3-Clause
Programming language: Python 2.7 or 3.5, with C extension
External Routines: NumPy [1], SciPy [1], Matplotlib [2]
Nature of problem: Calculating atomic properties of alkali atoms including lifetimes, energies, Stark shifts and dipole–dipole interaction strengths using matrix elements evaluated from radial wavefunctions.
Solution method: Numerical integration of radial Schrödinger equation to obtain atomic wavefunctions, which are then used to evaluate dipole matrix elements. Properties are calculated using second order perturbation theory or exact diagonalisation of the interaction Hamiltonian, yielding results valid even at large external fields or small interatomic separation.
Restrictions: External electric field fixed to be parallel to quantisation axis.
Supplementary material: Detailed documentation (.html), and Jupyter notebook with examples and benchmarking runs (.html and .ipynb).
[1] T.E. Oliphant, Comput. Sci. Eng. 9, 10 (2007). http://www.scipy.org/.
[2] J.D. Hunter, Comput. Sci. Eng. 9, 90 (2007). http://matplotlib.org/.

1. Introduction

Highly-excited atoms, in so called Rydberg states, provide strong atom–atom interactions, and large optical nonlinearities. They are a flourishing field for quantum information processing [1,2] and quantum optics [3–5] in the few to single excitation regime, as well as many-body physics [6–11], in the many-excitations limit. Their exploration requires detailed knowledge of both the single-atom properties, such as lifetimes, energies and transition dipoles elements, as well as atom pair properties, such as their interactions and strongly perturbed energy levels for atoms at distances in the micrometre range. Rydberg states are also highly sensitive to external DC and AC fields, making them excellent candidates for precision electrometry and imaging in the microwave [12,13] and terahertz [14] range, as well as performing state engineering to tune pair-state interaction potentials [15–17].

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: nikolasibalic@physics.org (N. Šibalić).

http://dx.doi.org/10.1016/j.cpc.2017.06.015

Although many of the relevant calculations share the same primitives, such as numeric integration of atomic wavefunctions based on measured energy levels and model core potentials, these basic efforts have been repeated by many groups independently so far. To date no single common resource has emerged for building complex calculations, or for performing quick numerical estimates. To this end, we have developed Alkali Rydberg Calculator (ARC) [18], a library of routines written in Python, using object-oriented programming to make a modular, reusable and extendable collection of routines and data for performing useful calculations of single-atom and two-atom properties, like level diagrams, interactions and transition strengths for alkali-metal atoms.

The hierarchical nature of the package helps organise possible calculations into abstraction levels, allowing one to pick the information at the relevant level for the calculation at hand. The data for individual atomic species is provided as classes that inherit calculation methods defined as abstract classes, allowing one easily to check and update relevant data, should future measurements improve some of the experimentally estimated values. Detailed documentation is provided for all the ARC's modules [19]. In addition, the code is commented, cross-referenced in-line and uses self-descriptive names. Whenever possible, the class and function names reflect the hierarchical structure of atomic physics knowledge and the natural decompositions of the calculations, not the low-level implementation details. In addition to the documentation, ARC has example snippets provided as an IPython notebook [20], giving an overview of Rydberg physics and how to perform calculations using the package library. This is a good starting point for new users.

To facilitate the initial adoption of the package and to allow quick calculations useful in the research planning stages, we are also providing a web-interface to basic functions of the package [21]. This allows any device with a web browser to access the web-server, that will use the ARC package to perform the calculations and obtain results that are transferred back to the users' web browser. In the process, the web service self-generates and outputs the code, so it can be used as an example-on-demand service, providing a starting point for more complex calculations.

This paper is organised as follows. An overview of the ARC architecture is presented in Section 2, where we introduce the theoretical framework for performing Rydberg state calculations, e.g. calculating wavefunctions and diagonalising interaction Hamiltonians. Here we also provide illustrative examples for building up calculations and visualising results with the provided tools. Initial setup for the ARC library is presented in Section 3, and specific details relating to the implementation are discussed in Section 4. Finally, Section 5 briefly summarises the package and outlines future possible expansions for the library, with a complete list of ARC classes, methods and functions detailed in Appendix A. Detailed documentation of the module is provided in .html format in the Supplemental Material [19] or available from the ARC website [21], along with an IPython notebook [20] that contains numerous additional examples and code snippets that reproduce many of the results from the literature.

2. ARC architecture and modules

2.1. Overview

The structure of the ARC library is shown in Fig. 1. At the lowest level, the `wigner` module implements angular momentum algebra (Wigner 3-j and 6-j coefficients and the WignerD rotation matrix), and `arc_c_extensions` is a Python extension coded in C to provide fast calculation of the radial part of the atomic wavefunctions. On a higher level, `alkali_atom_functions` uses these low-level functions to build general methods for calculating single-atom properties, which are contained within the abstract class `AlkaliAtom` that implements calculation of dipole matrix elements, transition rates, energy levels etc. The module `alkali_atom_data` defines an explicit class for each alkali element (e.g. `Rubidium87()`,`Caesium()`) that encodes all relevant physical parameters and inherits the calculation methods from the parent `AlkaliAtom` class. These atom classes can be passed as arguments to either of the core calculation modules, `calculations_atom_single` that implements interactive energy level diagrams (`LevelPlot`) and calculates Stark maps for atoms in external fields (`StarkMap`), or `calculations_atom_pairstate` for dealing with two-atom effects such as long-range dipole–dipole interactions. This pair-state module provides a sophisticated interface to automatically identify Förster resonances for atoms in weak electric fields (`StarkMapResonances`) and calculate atomic interaction potentials at both long and short range including up to quadrupole–quadrupole couplings (`PairStateInteractions`). In the following we will outline the basic functionality of the ARC module, provide the theoretical framework for the various modules and give examples of relevant calculations implemented in the library.

2.2. The *AlkaliAtom* class

Almost all calculated quantities in the ARC library can be derived from knowledge of the Rydberg state energy levels and matrix elements. The functions to calculate these values along with other primitive single-atom properties are encapsulated within the methods of the abstract class `AlkaliAtom` defined by the module `alkali_atom_functions`. This class is used in `alkali_atom_data` where each alkali-metal atom (`Lithium6`, `Lithium7`, `Sodium`, `Potassium39`, `Potassium40`, `Potassium41`, `Rubidium85`, `Rubidium87` and `Caesium`, as well as `Hydrogen`) inherits calculation methods from this abstract class, and specifies all the necessary numerical values for a given atom. Calculations with the ARC library begin by initiating an atom class associated with the relevant atom, as shown in the example below for caesium.

```
from arc import *        # initialise ARC library
atom = Caesium()         # create atom Object
```

Physical properties can then be determined using the methods in the form `atom.functionName(parameters)`, where a complete list of available methods is listed in Table B.2 and documentation [19]. The sections below outline the key properties.

2.2.1. Rydberg atom energy levels

Energy of the Rydberg state with principal quantum number n and orbital and total angular momentum ℓ and j respectively, are calculated from the Rydberg formula

$$E_{n,\ell,j} = E_{\mathrm{IP}} - \frac{\mathrm{Ry}}{(n-\delta_{n,\ell,j})^2}, \tag{1}$$

where E_{IP} is the ionisation energy threshold (`ionisationEnergy`), Ry is the Rydberg constant corrected for the reduced mass (`scaledRydbergConstant`), $\mathrm{Ry} = m_e/(m+m_e)\mathrm{Ry}_\infty$ and $\delta_{n,\ell,j}$ is the quantum defect (`getQuantumDefect`) given by

$$\delta_{n,\ell,j} = \delta_0 + \frac{\delta_2}{(n-\delta_0)^2} + \frac{\delta_4}{(n-\delta_0)^4} + \cdots, \tag{2}$$

where $\delta_0, \delta_2, \ldots$ are modified Rydberg–Ritz coefficients obtained by fitting precise measurements of the atomic energy levels [22].

Energies E for a given Rydberg state relative to the ionisation limit can be obtained using `getEnergy` (in eV), as well as methods `getTransitionWavelength` and `getTransitionFrequency` to return the wavelength (m) or frequency (Hz) of a transition between two Rydberg states. Energies and transitions are given relative to the centres of gravity of the hyperfine split ground and excited states.

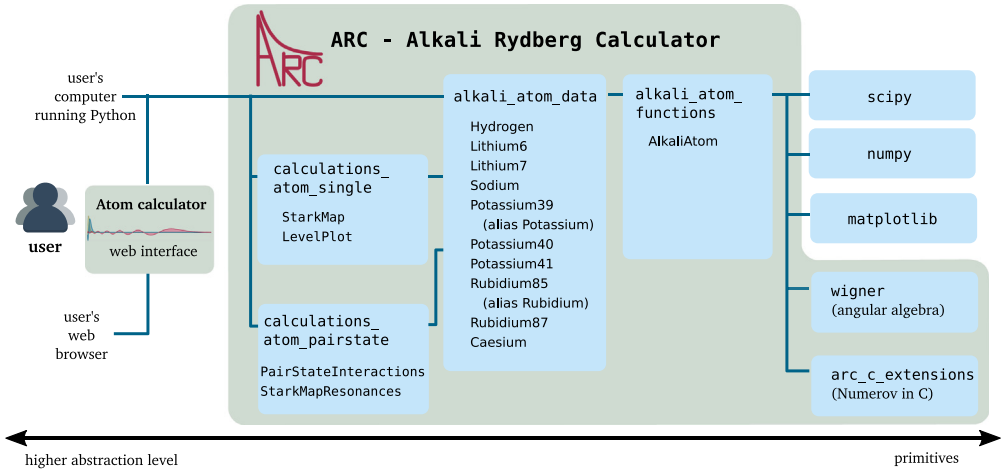

Fig. 1. Overview of the Alkali Rydberg Calculator (ARC) package for Python. Object-oriented structure with hierarchy reflecting the structure of atomic physics calculations is used. This allows the user to choose the abstraction level at which one wants to work, from low-level implementations of basic functions of `alkali_atom_functions.AlkaliAtom` for finding dipole matrix elements and lifetimes, to high-level functions that allow automatic Förster resonance finding and exploration of complex energy-level diagrams of atomic pair-states at small inter-atomic separations in `calculations_atom_pairstate`.

2.2.2. Rydberg atom wavefunctions

In order to calculate matrix elements for electric dipole and quadrupole couplings of the Rydberg states, it is necessary to calculate the radial wavefunctions $R(r)$ by numerically integrating the Radial Schrödinger equation for the valence electron. Using the substitution $\rho(r) = rR(r)$, we can remove the first order differential to obtain an equation in the form:

$$-\frac{1}{2\mu}\frac{\mathrm{d}^2\rho}{\mathrm{d}r^2} + \left[\frac{\ell(\ell+1)}{2\,\mu\,r^2} + V(r)\right]\rho(r) = E\,\rho(r), \tag{3}$$

where $V(r)$ is the spherically symmetric central potential (`potential`). For hydrogen, and high orbital angular momentum states $\ell > 3$, this is simply the Coulomb potential of $V(r) = -1/r + V_{so}(r)$, with added (relativistic) spin–orbit interaction $V_{so}(r) = \alpha\,\mathbf{L}\cdot\mathbf{S}/(2r^3)$, where α is the fine structure constant. However for alkali atoms with closed shells it is necessary to introduce a model potential that gives a Coulomb potential at long-range and at short range accounts for effects of the core penetration of the valence electron. We adopt the core potential (`corePotential`) of Marinescu et al. [23] given by the form

$$V(r) = -\frac{Z_\ell(r)}{r} - \frac{\alpha_c}{2r^4}(1 - e^{-(r/r_c)^6}) + V_{so}(r), \tag{4}$$

where α_c is the core polarisability, r_c is a cutoff radius introduced to avoid divergence of the polarisation potential at short range and the radial charge (`effectiveCharge`) is given by

$$Z_\ell(r) = 1 + (z-1)\,e^{-a_1 r} - r\,(a_3 + a_4 r)\,e^{-a_2 r}, \tag{5}$$

with ℓ-dependent parameters $a_{1..4}$ and r_c taken from Table 1 of Ref. [23] which were obtained by fitting the model potential to measured energy levels for each element.

Using this model potential and the known Rydberg energy from above, the radial wavefunctions can be calculated by numerically integrating Eq. (3), as shown on Fig. 2. This is achieved by performing a transformation to integrate the function $X(r) = R(r)r^{3/4}$ in terms of the scaled co-ordinate $x = \sqrt{r}$ [24] that gives an approximately constant number of points across each period of oscillation in the wavefunction. The result is a second-order differential equation of the form

$$\frac{\mathrm{d}^2X}{\mathrm{d}x^2} = g(x)\,X, \tag{6}$$

which is efficiently solved using the Numerov method [25,26]

$$\begin{aligned}&[1 - T(x+h)]\;X(x+h) + [1 - T(x-h)]\;X(x-h)\\ &\quad = [2 + 10T(x)]\;X(x) + O(h^6),\end{aligned} \tag{7}$$

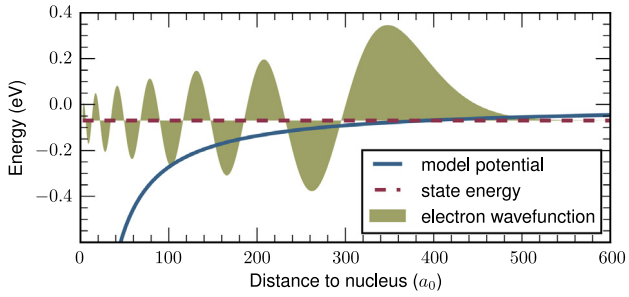

Fig. 2. Electron wavefunctions are calculated by Numerov integration, here shown for caesium 18 $S_{1/2}m_j = 1/2$ state. Solving the radial part of the Schrödinger equation in the model potential from [23], using state energy from quantum defects or NIST ASD database [28]. These calculations are the starting step in calculating all atom-light couplings (dipolar and higher order).

where h is the step size, $T(x) = h^2 g(x)/12$ and

$$g(x) = 8\,\mu\,x^2(V(r) - E) + \frac{\left(2\ell + \frac{1}{2}\right)\left(2\ell + \frac{3}{2}\right)}{x^2}. \tag{8}$$

It is necessary to truncate the range of integration as at short range the model becomes unphysical and diverges, whilst at long-range the wavefunction decays to zero. Following Ref. [27], the limits of integration are set to use an inner radius of $r_i = \sqrt[3]{\alpha_c}$, and an outer radius of $r_o = 2n(n+15)$ which is much larger than the classical turning point of the wavefunction. To minimise errors introduced by the approximate model potential at short range, the integration is performed inwards, starting at r_o. For high orbital momentum states ($\ell > 3$) divergence can occur even before r_i, which is automatically detected and integration is stopped at the closest wavefunction node before divergence occurred. The wavefunctions are then normalised, and returned by `radialWavefunction`.

2.2.3. Matrix elements

Relevant properties for alkali atoms such as lifetimes (Section 2.2.5), Stark shifts (Section 2.3.2) and atomic interaction potentials (Section 2.4.1) require evaluation of the electric dipole and electric quadrupole matrix elements from state $|n, \ell, m_\ell\rangle$ to $|n', \ell', m'_\ell\rangle$. For electric dipole transitions, the interaction is dependent upon matrix elements of the form $\mathcal{H} = -e\boldsymbol{r}\cdot\hat{\boldsymbol{\epsilon}}$ where $\hat{\boldsymbol{\epsilon}}$ is the electric field polarisation vector. Expanding the operator in

the spherical basis and using the fact that the calculation can be separated into radial overlap, and angular overlap of the electron wavefunction with operator $\mathcal{H}$, the resulting matrix element can be evaluated in terms of angular momentum terms and radial matrix elements using the Wigner–Ekart theorem [29]

$$\langle n,\ell,m_\ell|r_q|n',\ell',m'_\ell\rangle = (-1)^{\ell-m_\ell}\begin{pmatrix}\ell & 1 & \ell'\\ -m_\ell & -q & m'_\ell\end{pmatrix} \times \langle\ell||r||\ell'\rangle, \tag{9}$$

where q represents the electric field polarisation ($q = \pm1, 0$ driving $\sigma^{\pm}, \pi$ transitions respectively), and the braces denote a Wigner-3j symbol (`Wigner3j`). We use Condon–Shortley phase convention [30] for spherical harmonics. The reduced matrix element $\langle\ell||r||\ell'\rangle$ (`getReducedMatrixElementL`) is given by

$$\langle\ell||r||\ell'\rangle = (-1)^{\ell}\sqrt{(2\ell+1)(2\ell'+1)}\begin{pmatrix}\ell & 1 & \ell\\ 0 & 0 & 0\end{pmatrix} \times \mathcal{R}_{n\ell\to n'\ell'}, \tag{10}$$

where the radial matrix element (`getRadialMatrixElement`) is evaluated from

$$\mathcal{R}_{n\ell\to n'\ell'} = \int_{r_i}^{r_o} R_{n,\ell}(r)\, r\, R_{n',\ell'}(r)\, r^2\, \mathrm{d}r, \tag{11}$$

using numerical integration of the calculated wavefunctions.

Transforming these to the fine-structure basis Eq. (9) can be expressed in terms of states j, m_j (`getDipoleMatrixElement`) as

$$\begin{aligned}&\langle n,\ell,j,m_j|r_q|n',\ell',j',m'_j\rangle\\ &= (-1)^{j-m_j+\ell+s+j'+1}\sqrt{(2j+1)(2j'+1)}\\ &\times\begin{pmatrix}j & 1 & j'\\ -m_j & -q & m'_j\end{pmatrix}\begin{Bmatrix}j & 1 & j'\\ \ell' & s & \ell\end{Bmatrix}\langle\ell||r||\ell'\rangle,\end{aligned} \tag{12}$$

where curly braces denote a Wigner-6j symbol (`Wigner6j`).

To account for higher order multipole moments in the interaction between atom pairs (see Section 2.4.1), it is also necessary to calculate quadrupole matrix elements (`getQuadrupoleMatrixElement`) of the form

$$\mathcal{R}^{Q}_{n\ell\to n'\ell'} = \int_{r_i}^{r_o} R_{n,\ell}(r)\, r^2\, R_{n',\ell'}(r)\, r^2\mathrm{d}r. \tag{13}$$

As with the quantum defect model for the energies, these numerical approaches provide accurate estimates of the dipole and quadrupole terms for highly-excited states but have a large error for low-lying states where the electron density is weighted close to the atomic core where integration is most sensitive to the divergence of the model potential. To overcome this limitation, values for dipole matrix elements available in the literature either from direct measurement or more complex coupled-cluster calculations [31,32] are tabulated (accessible through the function `getLiteratureDME`). Before calculating a matrix element, the ARC library first checks if a literature value exists and if so utilises the tabulated value with the smallest estimated error. Otherwise a numerical integration is performed using the method outlined above. For each element, these tabulated values are contained within an easily readable `.csv` file (file name specified in `literatureDMEfilename`) that lists the value along with relevant bibliographical reference information, making it easy for users to add new values at a later date.

2.2.4. Rabi frequency

An important parameter in experiments with Rydberg atoms is the Rabi frequency $\Omega = \boldsymbol{d}\cdot\boldsymbol{\mathcal{E}}/\hbar$ where $\boldsymbol{d}$ is the dipole matrix element for the transition and $\boldsymbol{\mathcal{E}}$ is the electric field of the laser driving the transition. For a transition from state $|n_1\ell_1j_1m_{j_1}\rangle \to |n_2\ell_2j_2m_{j_2}\rangle$ using a laser with power P and $1/e^2$ beam radius w, the Rabi frequency (in rad s^{-1}) can be obtained as

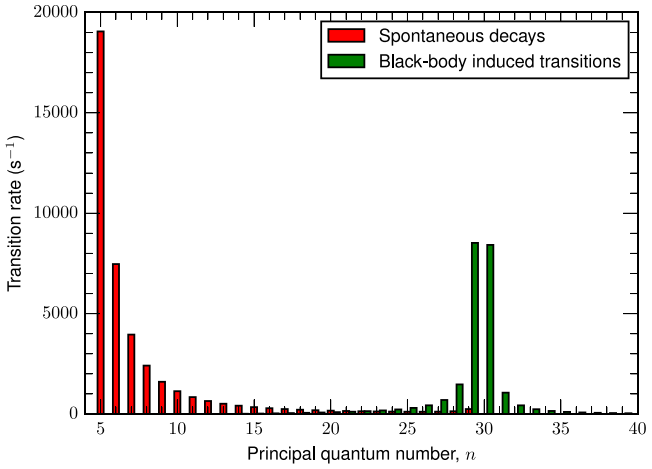

Fig. 3. Spontaneous decays and black-body induced transitions from Rb 30 $S_{1/2}$ to n $P_{1/2,3/2}$ for environment temperature of 300 K. These transitions have to be included in calculation of excited-state lifetime for Rb 30 $S_{1/2}$ state.

```
atom.getRabiFrequency(n1,l1,j1,mj1,n2,l2,j2,mj2,P,w)
```

The related function `atom.getRabiFrequency2` returns the Rabi frequency calculated from the electric field amplitude.

2.2.5. Excited state lifetimes

Radiative lifetimes of alkali atoms can be calculated using dipole matrix elements to determine the Einstein A-coefficient [33] for transitions (see `getTransitionRate`),

$$A_{n\ell\to n'\ell'} = \frac{4\omega^3_{nn'}}{3c^3}\frac{\ell_{\max}}{2\ell+1}\mathcal{R}^2_{n\ell\to n'\ell'}, \tag{14}$$

where $\omega_{nn'}$ is the frequency of the transition from $|n\ell\rangle$ to $|n'\ell\rangle$. The total radiative decay Γ_0 is then obtained by summing over all dipole-coupled states $|n'\ell'\rangle$ of lower energy,

$$\Gamma_0 = \sum_{n\ell>n'\ell'} A_{n\ell\to n'\ell'}. \tag{15}$$

For an environment at finite temperature, it is necessary to account for the interaction of the atom with black-body radiation (BBR) [34]. This causes stimulated emission and absorption which depend on the effective number of BBR photons per mode, $\bar{n}_\omega$, at temperature T given by the Planck distribution

$$\bar{n}_\omega = \frac{1}{e^{\hbar\omega_{nn'}/k_BT}-1}, \tag{16}$$

multiplied by the A-coefficient resulting in a BBR transition rate

$$\Gamma_{\mathrm{BBR}} = \sum_{n'\ell'} A_{n\ell\to n'\ell'}\,\bar{n}_{\omega_{nn'}}, \tag{17}$$

where the summation n', ℓ' includes also states higher in energy, since BBR can drive these transitions. Finally, the effective lifetime τ (see `getStateLifetime`) is calculated from $1/\tau = \Gamma_0 + \Gamma_{\mathrm{BBR}}$.

Fig. 3 shows the relative contribution to the excited-state lifetime of Rb 30 $S_{1/2}$ due to radiative decay (dominated by low-lying states due to the ω^3 scaling in Eq. (14)) and black-body decay at 300 K calculated using ARC (for full code see Appendix A). Comparison of our calculated lifetimes [20] with previous work on radiative [33] and BBR induced lifetimes for alkali-metals [34] yields excellent agreement.

2.2.6. Atomic vapour properties

For experiments using thermal vapours of alkali atoms, a number of useful functions are provided for returning atomic vapour pressure (getPressure) or number density (getNumberDensity), as well as the average interatomic distance (getAverageInteratomicSpacing) and atomic speed (getAverageSpeed) at a given temperature. For full listing of functions see Table B.2 and full documentation in Supplemental [19].

2.3. Single-atom calculations

The module calculations_atom_single provides calculations of single-atom properties. It supports plotting of atom energy levels with interactive finding of associated transition wavelengths and frequencies (Section 2.3.1), i.e. Grotrian diagrams, and atom-energy-level shifts in electric fields (Section 2.3.2), i.e. Stark maps.

2.3.1. Level plots

The LevelPlot class facilitates simple plotting of atomic energy levels. It provides an interactive plot for exploring transition wavelengths and frequencies. For example, generation and plotting of the caesium energy level diagram, including ℓ states from S to D for principal quantum numbers from $n = 6$ to $n = 60$, can be realised with:

```
atom = Caesium()
levels = LevelPlot(atom)
# parameters: nmin, nmax, lmin, lmax
levels.makeLevels(6,60,0,3)
levels.drawLevels()
levels.showPlot()
```

This simple example also demonstrates that the more complicated calculations are implemented as a compact classes, whose initialisation and methods closely follow naming and stages one would perform in a manual calculation. Yet, working on high abstraction level, one can obtain information directly relevant for research in just a couple of high-level commands [20].

2.3.2. Stark shifts

Calculation of atomic Stark shifts in external static electric fields provides a tool for both precision electrometry and a mechanism for tuning interatomic interactions to a Förster resonance to exploit strong resonant dipole–dipole interactions (Section 2.4.4). To find the energy of the atom in an electric field $\mathcal{E}$ applied along the z-axis it is necessary to find the eigenvalues of the system described by the Stark Hamiltonian

$$\mathcal{H} = \mathcal{H}_0 + \mathcal{E}\hat{z}, \tag{18}$$

where $\mathcal{H}_0$ is the Hamiltonian for the unperturbed atomic energy levels and $\mathcal{E}$ is the applied electric field. The electric field term causes a mixing of the bare atomic energy levels due to coupling by the Stark interaction matrix elements $\langle n, \ell, j, m_j|\mathcal{E}\hat{z}|n', \ell', j', m_j'\rangle$ which can be evaluated from Eq. (12) with $q = 0$. The selection rules of the Stark Hamiltonian give $\Delta m_j = 0$, $\Delta\ell = \pm 1$ such that only states with projection m_j are coupled together, so each Stark map can be constructed by taking basis states with the same m_j value. Following the method of Zimmerman et al. [27], Stark shifts are calculated by exact diagonalisation of the Hamiltonian.

Stark shift calculations are handled using the StarkMap class, which is initialised by passing the appropriate atom class. For a target state $|n, \ell, j, m_j\rangle$, a range of n values from $n_{\min}$ to $n_{\max}$ with values of ℓ up to $\ell_{\max}$ is required to define the basis states for the Stark Hamiltonian. Convergence is typically achieved using $\ell_{\max}$ of 20 and $n_{\max} - n_{\min} \sim 10$, however for large applied fields or higher values of n it will be necessary to increase the basis size to account for the strong mixing of the energy levels. Finally, the Hamiltonian is diagonalised for each value of the electric field (defined in V/cm). As an example, to calculate the Stark map shown in Fig. 4 for the 28 $S_{1/2}$ $m_j = 1/2$ state in Cs we include states $n_{\min} = 23$ to $n_{\max} = 32$ with $l_{\max} = 20$ for 600 equidistant electric field values in the range from 0 to 600 V/cm. The corresponding program call is:

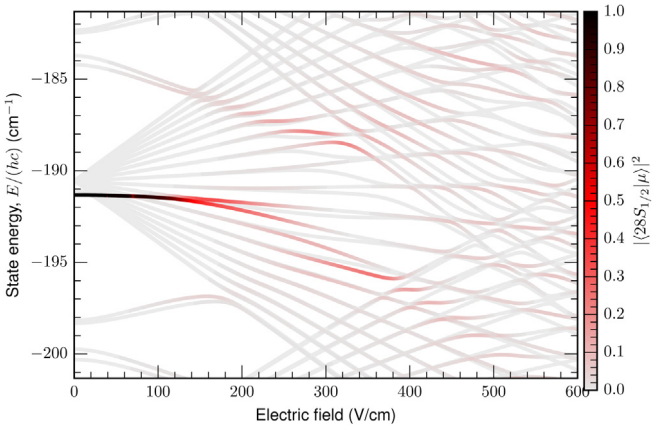

Fig. 4. Example of Stark map calculation, showing caesium 28 $S_{1/2}$ $m_j = 1/2$ state perturbation by DC electric field. Colour highlights contribution of the $|28\,S_{1/2}\ m_j = 1/2\rangle$ state in the atom eigenstates $|\mu\rangle$. (color online)

```
calc = StarkMap(Caesium())
# parameters: n, l, j, mj, nmin, nmax, lmax
calc.defineBasis(28, 0, 0.5, 0.5, 23, 32, 20)
calc.diagonalise(np.linspace(0.,600*1e2,600))
calc.plotLevelDiagram()
calc.showPlot()
```

In interactive mode, the plot can be clicked to obtain the dominant contribution of the basis states within each eigenstate and by default the figure is highlighted in proportion to the fractional contribution of the target state (in this case 28 $S_{1/2}$ $m_j = 1/2$). An alternative option is to highlight the plot proportional to the transition probability for laser excitation from state $|n', \ell', j', m_{j'}\rangle$ with a laser polarisation driving q polarised transition (where $q = \pm 1, 0$ corresponds to $\sigma^{\pm}$, π). This is enabled using the optional parameter drivingFromState in the call to the diagonalise method of StarkMap. For example, the corresponding code for driving σ^+ transition from 7 $S_{1/2}$ $m_j = -1/2$ would be

```
calc.diagonalise(np.linspace(00.,6000,600),\
    drivingFromState=[7,0,0.5,-0.5,+1])
```

Finally, to evaluate the static polarisability α_0 of the target state in units of MHz/(V/cm)^2, the getPolarizability method is called on the StarkMap object, i.e.

```
print("%.5f MHz cm^2 / V^2 " % calc.getPolarizability())
```

2.4. Pair-state calculations

The module calculations_atom_pairstate contains classes and methods for the calculation and visualisation of long-range and short range interactions (PairStateInteractions), as well as an automated tool for identifying Förster resonances (StarkMapResonances) for electric field tuning of the interaction potential.

2.4.1. Interatomic interactions

Pair-wise interactions between two atoms with internuclear separation R, and electron coordinates $\mathbf{r}_1$ and $\mathbf{r}_2$ relative to the respective nuclei, can be expanded in multipole form as [35]

$$V(R) = \sum_{L_1,L_2=1}^{\infty} \frac{V_{L_1,L_2}(\mathbf{r}_1,\mathbf{r}_2)}{R^{L_1+L_2+1}}, \tag{19}$$

where $L_1+L_2=2$, $L_1+L_2=3$ and $L_1+L_2=4$ correspond respectively to dipole–dipole, dipole–quadrupole and quadrupole–quadrupole interactions, and

$$\begin{aligned} V_{L_1,L_2}(\mathbf{r}_1,\mathbf{r}_2) = & \frac{(-1)^{L_2}4\pi}{\sqrt{(2L_1+1)(2L_2+1)}} \\ & \times \sum_m \sqrt{\binom{L_1+L_2}{L_1+m}\binom{L_1+L_2}{L_2+m}} r_1^{L_1} r_2^{L_2} \\ & \times Y_{L_1,m}(\hat{r}_1) Y_{L_2,-m}(\hat{r}_2), \end{aligned} \tag{20}$$

where $\binom{n}{k}$ is the binomial coefficient and $Y_{L,m}(\hat{r})$ spherical harmonics. In Eq. (20) the quantisation axis is oriented along internuclear axis **R**. On the other hand, the atomic states' quantisation axis is typically defined with respect to the driving laser, being directed along the laser propagation direction for circularly polarised laser beams, or in plane of the electric field vector, perpendicular to the laser propagation direction, for linearly polarised driving, or in the direction of applied static electric and magnetic fields. When the atomic quantisation axis is along the internuclear axis, matrix elements $\langle j_1 m_{j1}|V_{L_1,L_2}|j_2, m_{j2}\rangle$ are easily evaluated. When the quantisation axis is not oriented along **R**, their relative orientation can be described with polar angle θ and azimuthal angle ϕ that the inter-atomic axis makes with the quantisation axis [Fig. 5(a)]. Atom states $|j, m_j\rangle$ are then rotated with WignerD matrices $w_D(\theta,\phi)$ (`wignerDmatrix`), so that inter-atomic axis defines the quantisation direction [29]. The coupling can then be easily calculated between the rotated states $|s\rangle = w_D(\theta,\phi)|j, m_j\rangle$. Note that $|j, m_j\rangle$ in the rotated basis $|j', m_j'\rangle$ is now, in general, a superposition of m_j' states.

This multipole expansion is valid as long as the wavefunctions of the atoms do not overlap, which is the case for interatomic separations R larger than the Le Roy radius [36]

$$R_{LR} = 2\left(\langle r_1^2\rangle^{1/2} + \langle r_2^2\rangle^{1/2}\right). \tag{21}$$

Evaluation of the Le Roy radius can be achieved using the function `getLeRoyRadius`. For example the Le Roy radius for two Caesium atoms in the n S state is 0.1 μm for $n \sim 20$ and reaches 1 μm for $n \sim 60$, marking the interatomic distance below which the results of the pair-state diagonalisation become invalid.

To understand the effect of interactions, consider a pair of atoms in Rydberg pair-state $|r, r\rangle$ coupled via $V(R)$ to Rydberg pair-state $|r', r''\rangle$ which has an energy defect $\Delta_{r',r''} = 2E_r - E_{r'} - E_{r''}$. In the pair-state basis $\{|rr\rangle, |r'r''\rangle\}$ their interaction is described with the Hamiltonian

$$\mathcal{H}_{\text{int}} = \begin{pmatrix} 0 & V(R) \\ V(R) & \Delta_{r'r''} \end{pmatrix}. \tag{22}$$

We see that at short range where $V(R) \gg \Delta_{r',r''}$ the splitting of the energy eigen-states $\pm V(R)$ is dominated by the pair-state interaction energy. Assuming that V(R) has non-zero dipole–dipole term of the multipole expansion [$L_1+L_2=2$ in Eq. (20)], this corresponds to the resonant dipole–dipole regime, giving eigenstates' energy distance dependence $\propto C_3/R^3$. At large R where $\Delta_{r',r''} \gg V(R)$, the interaction is second order leading to an energy shift of $-V(R)^2/\Delta_{r',r''} = -C_6/R^6$, known as the van der Waals regime. The cross-over between these regimes occurs at the van der Waals radius R_{vdw} where $V(R_{vdw}) = \Delta_{r',r''}$.

For a real system, whilst the pair-state with the smallest $\Delta_{r',r''}$ dominates the resulting interaction shift, it is necessary to consider the effects of all near-resonant pair-states that are coupled by the $V(R)$ interaction term above. Calculations of these pair-wise interaction potentials are handled by the `PairStateInteractions` class that is initialised by specifying the element name and target pair-state $n_1, l_1, j_1, n_2, l_2, j_2, m_{j_1}, m_{j_2}$ whose behaviour we want to explore.

```
calc=PairStateInteractions(atom(),n1,l1,j1,n2,l2,j2,mj1,mj2)
```

2.4.2. Long-range limit

At large separation, for off-resonantly ($\Delta_{r',r''} \neq 0$) coupled states, the dipole–dipole interactions dominate to give an interaction of the form $-C_6/R^6$ van der Waals potential where the sign depends on the energy defect of the closest dipole-coupled pair-states $|r'r''\rangle$, leading to attractive or repulsive interactions accordingly. In the long-range limit $V(R) \ll \Delta$ (where Δ is the energy defect of the closest dipole-coupled pair-state), the C_6 coefficient for pair-state $|rr\rangle$ can be evaluated using second-order perturbation theory as

$$C_6 = \sum_{r',r''} \frac{|\langle r'r''|V(R)|rr\rangle|^2}{\Delta_{r',r''}}, \tag{23}$$

where the sum runs over all pair-states $|r'r''\rangle$ whose energy differs from the pair-states $|rr\rangle$ energy for $\Delta_{r',r''} < \Delta_E$, where Δ_E is some maximal energy defect that provides a truncation of the basis states. This calculation is performed using the method `getC6perturbatively`, returning C_6 in units of GHz μm^6. Users must specify the relative orientation of the atoms, the range of principal quantum numbers for the states used in calculation, and the maximal energy defect. For example for the interaction of two rubidium atoms in 60 $S_{1/2}$, $m_{j_1} = 1/2$, $m_{j_2} = -1/2$ states, whose inter-atomic axis is set at an angle of $\theta = \pi/6$ with respect to the quantisation axis [Fig. 5(a)], the C_6 interaction term can be calculated from states with principal quantum number differing maximally $\delta n = 5$ from the $n = 60$, and maximal energy difference in pair-state energies of $h \times 25$ GHz as

```
# parameters: atom, n1, l1, j1, n2, l2, j2, mj1, mj2
calc = PairStateInteractions(Rubidium(), 60, 0, 0.5, 60, 0, 0.5,
    0.5, -0.5)\
# parameters: theta, phi, deltan, deltaE
c6 = calc.getC6perturbatively(pi/6, 0, 5, 25.e9);
print("C_6 = %.0f GHz (mu m)^6" % c6)
```

Using this function, the anisotropy of the $V(R)$ interaction can be easily identified as shown in Fig. 5(b) which plots the magnitude of C_6 for a pair of atoms in the 60 $D_{5/2}$ state of Rubidium for $\theta = 0 \ldots 2\pi$.

2.4.3. Exact interaction potential

To evaluate the interaction potential for arbitrary separation [valid down to the Le Roy radius, Eq. (21)], it is necessary to diagonalise the matrix $V(R)$ containing all the interatomic couplings for each separation R to obtain exact values of the eigenvalues and eigenstates describing the interaction between atomic pair-states. Due to the strong admixing of states, this process requires careful choice of atomic basis states, including higher order orbital angular momentum states up to ℓ_{max}. For each distance R, the matrix is diagonalised using an efficient ARPACK package provided through Numpy, and the n_{eig} eigenstates whose eigenvalues are closest to the target pair-state are returned.

Fig. 6 provides an example of the interaction potential for a pair of atoms initially in the $|60\ S_{1/2}\ m_j = 1/2, 60\ S_{1/2}\ m_j = -1/2\rangle$

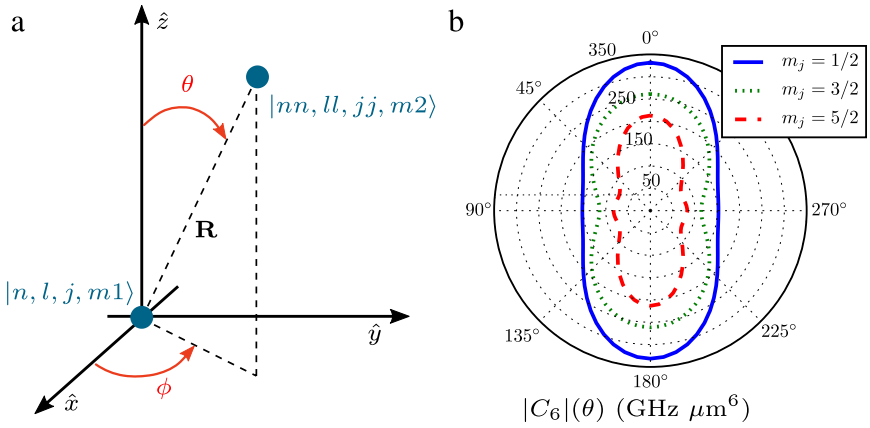

Fig. 5. Orientation of the two interacting atoms and their anisotropic interactions. (a) Orientation of the two atoms is defined as a polar θ and axial ψ angle that the inter-atomic axis makes with respect to the quantisation axis $\hat{z}$. (b) Anisotropy of long-range interactions is illustrated in the case of two rubidium atoms, both in the 60 $D_{5/2}$ m_j state, through the dependence of the C_6 interaction coefficient on angle θ between the quantisation axis and the inter-molecular axis.

state of Rubidium for 200 interatomic spacings in the range of 0.5–10 μm, finding $n_{\text{eig}} = 150$ closest eigenstates, accounting for states with orbital angular momentum up to $l_{\text{max}} = 4$ created using the code

```
calc = PairStateInteractions(Rubidium(), 60, 0, 0.5, 60, 0, 0.5,
    0.5, -0.5)
# parameters: theta, phi, nMax, lMax, maxEnergy
calc.defineBasis (0, 0, 5, 4, 25.e9)
calc.diagonalise (np.linspace (0.5,10.0,200) ,150)
calc.plotLevelDiagram()
calc.showPlot()
```

As with the StarkMap method, the figure is interactive allowing users to determine the dominant composition (expressed in the pair-state basis) of a given eigenstate. The default behaviour is to highlight the eigenvalues proportional to the fractional contribution of the original target pair-state. Alternatively, as shown on Fig. 6], the optional parameter drivingFromState is used in the call to diagnolise to give highlighting in proportion to the relative laser coupling strength from a given state, assuming that one atom is already in one of the two states that make the initially specified pair-state.

By default, PairStateInteractions includes only dipole coupling between the states in calculating level diagrams. Interactions up to quadrupole–quadrupole term [including all terms $L_1 + L_2 \leq 4$ in Eq. (20)] can be included by setting optional parameter interactionsUpTo = 2 during the PairStateInteractions initialisation, which can be important for the short-distance structure of level diagram [37]. For evaluation of long-range potential curves this additional term makes only small perturbations to the asymptotic C_6 behaviour. However, for accurate determination of the molecular levels of the short-range potential wells it is important to include the higher order multipole terms. Convergence should also be checked by increasing the basis size and re-evaluating the relevant parameters to ensure all the relevant states are included in the calculation basis.

Following diagonalisation of the pair-state interaction matrix, the long-range (C_6) and short range (C_3) dispersion coefficients can be evaluated using the methods getC3fromLevelDiagram and getC6fromLevelDiagram respectively, which perform a fit to the eigen-energies associated with the state containing the largest admixture of the target state. A method for finding the cross-over distance between van der Waals and resonant dipole–dipole interactions, i.e. van der Waals radius R_{vdw}, is also provided (getVdwFromLevelDiagram).

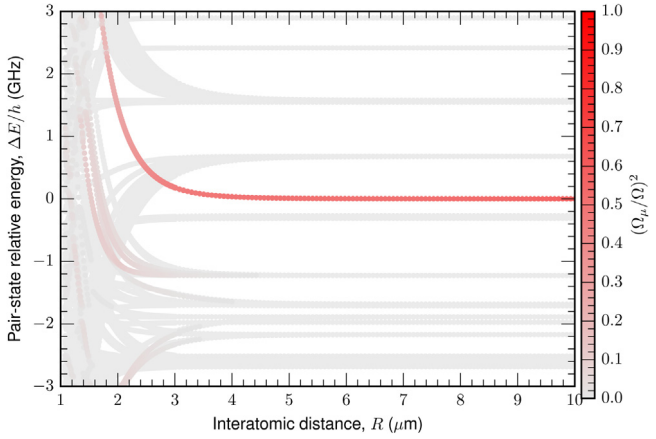

Fig. 6. Example of interactions between the atoms, causing level shifts in the atomic pair-states. Here shown in the vicinity of Rubidium $|60\, S_{1/2}\, m_j = 1/2, 60\, S_{1/2}\, m_j = -1/2\rangle$ state. Highlighting is proportional to the driving strength from the $5\, P_{3/2}\, m_j = 3/2$ state with coupling laser driving σ^- transitions. Atom interatomic direction is oriented along the quantisation axis ($\theta, \phi = 0$). Using the provided methods we can find van der Waals radius $R_{\text{vdW}} = 2.4$ μ m, and short-range $C_3 = 16.8$ GHz μ m^3 and long range $C_6 = 135$ GHz μ m^6 dispersion coefficients. (color online)

2.4.4. Stark tuned Förster resonances

As outlined in Section 2.4.1, the finite pair-state energy defects Δ associated with the closest dipole-coupled pair-states lead to a transition from first order resonant dipole–dipole interactions ($\propto R^{-3}$) to second order van der Waals ($\propto R^{-6}$) at the van der Waals radius. Using external electric fields however, it is possible to Stark shift the pair-states into resonance to obtain long-range $\propto R^{-3}$ interactions for all values of R, known as a Förster resonance [38–41].

To identify suitable Förster resonances the StarkMapResonances class is used, taking in a pair of target pair-states and performing diagonalisation of the Stark Hamiltonian of Eq. (18) in the pair-state basis. Due to the angular momentum selection rules of the dipole operator $V(R)$, the target pair-state can be dipole-coupled to pair-states with $\Delta m_j = \pm 1, 0$, it is necessary to calculate Stark maps for a range of m_j manifolds. Following diagonalisation, only pair-states with energy close to the target state are considered, with any states not dipole-coupled to the target pair-state being discarded. As the electric field coupling leads to significant mixing of the zero-field pair-states, the algorithm identifies the basis state containing the largest target state fraction at a given electric field to test for the closest dipole-coupled pair-states. Finally, an interactive plot routine enables users to identify states that have a Förster

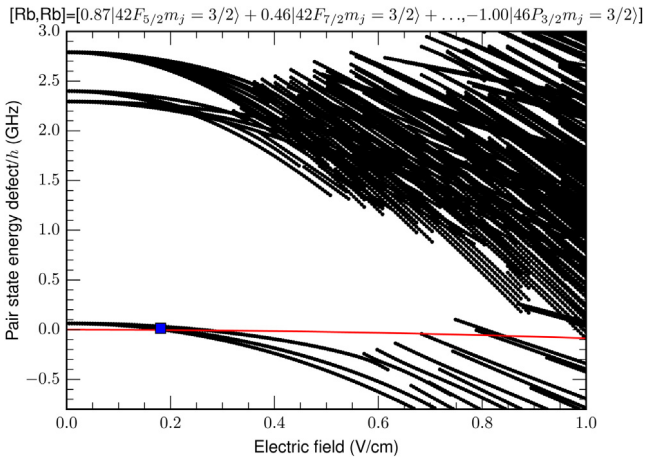

Fig. 7. Example plot produced by `StarkMapResonances` for finding Förster resonances for Rubidium pair-state 44 $D_{5/2}$ $m_j = 5/2$ shown as grey (red) line. Pair-states whose main admixed state is dipole-coupled to the original pair-state are shown as black lines. Clicking on one of these states, marked with square (blue) on the plot, reveals its composition in the plot title. In this case, it is a mixture of 42 $F_{5/2}$ and 42 $F_{7/2}$ states in one of the atoms, and essentially unperturbed 46 $P_{3/2}$ state that is almost resonant with original pair-state at electric field of $E \approx 0.18$ V/cm. (color online)

resonance. Note, unlike the previous class, `StarkMapResonances` accepts two atom classes at initialisation making it possible to determine inter-species resonances.

Fig. 7 shows an example plot to identify Förster resonances for a pair of atoms in the 44 $D_{5/2} m_j = 5/2$ state of Rubidium that is generated using the following code

```
state = [44,2,2.5,2.5]
calculation = StarkMapResonances(Rubidium(), state, Rubidium(),
    state)
# nMin, nMax, lMax, rangeEfield, [energyRange]
calculation.findResonances(39, 49, 20, np.linspace(0,100,200),
    energyRange=[-0.8e9,3.e9])
calculation.showPlot()
```

In the above example, similarly as when finding a Stark map (Section 2.3.2), we have to specify a range of acceptable principal quantum numbers $[n_{\min}, n_{\max}]$ and maximal orbital angular momentum $l_{\max}$ for the basis states for Stark map calculations, as well as the electric field range (in this case 0–1 V/cm at 200 equidistant points). An additional argument `energyRange` defines the energy window within which we will keep the resonant pair-states. In the above given example, this is in range $h \times [-0.8, 3.0]$ GHz. From the interactive plot, selection of the pair-state eigenvalues shows a Förster resonance occurring at 0.18 V/cm with the 42 F, 46 $P_{3/2}$ pair-state where, due to the electric coupling, the F state is an admixture of the 42 $F_{5/2,7/2}$ $m_j = 3/2$ states.

3. Installation and usage

It is assumed the pre-requisites (Numpy, SciPy and Matplotlib) are installed and can be located by the Python interpreter (see e.g. [42]). Both Python 2.7 and 3.5 are supported. To achieve good performance, it is recommended to use Numpy packages that connect to optimised backends, like ATLAS [43]. Prepackaged Python distributions, like Anaconda [42], provide this out-of-the box. The ARC library can be downloaded online [18] as a .zip file release. Installation is performed by extracting the downloaded .zip archive and copying the arc subfolder into the root of your project directory. It is important that Python has write access to the folder where the package is located, so that database files (stored in `arc/data/`) can be updated and used.

3.1. Optimised Numerov integrator

Integration of the atomic wavefunctions to obtain dipole matrix elements is numerically intensive and by default ARC uses an optimised Numerov integration routine implemented with C extensions for Python (`arc_c_extensions`). In the unlikely case the precompiled executable `arc_c_extensions.so` is incompatible with the installed system, please install a C compiler and compile `arc_c_extensions.c` located in the ARC root folder using by calling from command line

```
python setupc.py build_ext --inplace
```

Note that path to arc directory should not contain spaces in order for `setupc.py` script to work. Alternatively, the native Python solver can be used by setting the optional argument `cpp_numerov=False` when initialising the atom class, however this is not recommended for intensive calculations as it results in substantially slower execution.

3.2. Getting started

To initialise the library, use the following code at the start of a Python script or interactive IPython notebook with

```
# locate ARC Directory
import sys, os
rootDir = '/path/to/root/directory/for/arc'
sys.path.append(rootDir)
os.chdir(rootDir)
# import ARC library
from arc import *
```

This firsts sets a path to the directory containing ARC package directory on your computer.[1] This is the recommended way of using the package in a research environment, since users can easily access, check and change the underlying code and constants according to their needs. ARC is now ready for use, and can be tested using the example code above. Numerous additional examples are provided in IPython notebook "An Introduction to Rydberg atoms with ARC" [20].

4. Implementation

4.1. Physical constants

As mentioned above, the atomic properties are encapsulated in classes which contain the relevant atomic properties determined from the literature along with model potential coefficients taken from Marinescu et al. [23] which have been optimised against measured energy levels. For each atom, asymptotic expansions of the quantum defects are used to determine the energy levels of states with high principal quantum number to high accuracy, using measured values of the ionisation energy, Rydberg constant and quantum defects taken from Li [44,22], Na [22], K [22], Rb [45–48], and Cs [49–51]. For the low-lying states, the quantum defects do not accurately reproduce the measured energies and instead energy levels are determined from data in the NIST ASD database [28]. By default, the cut-off between tabulated and calculated energies is determined by the point at which the error in the calculated energy exceeds 0.02%.

[1] Note that in the directory path, a backslash (\) should be used on Windows machines, instead of forward slash used on UNIX based machines (Linux, MacOS).

4.2. Tracking calculation progress

For tracking progress of calculations, most functions (see documentation [19] for details) provide optional `bool` arguments, that turn on printing of additional information about calculations. For example, `progressOutput` prints basic status information, and is recommended for everyday use. If `debugOutput` is set to `True`, more verbose information about basis states and couplings will be printed in the standard output.

4.3. Memoization

In order to achieve good performance, memoization is used throughout. That is, when functions receive request for calculation, memorised previous results are checked, and the value is retrieved from memory if it already exists. Both an in-application SQL database (SQLite) and standard arrays are used for this. Memoization is used for dipole and quadrupole matrix element calculations and angular coupling factors, that are independent of principal quantum number of the considered state, including Wigner-*n* J coefficients and WignerD matrices. Single-atom and pair-state calculations will automatically update databases if new values are encountered, speeding up the future calculations. Updating dipole and quadrupole matrix element database can be done manually too, by calling `updateDipoleMatrixElementsFile`.

4.4. Saving and retrieving calculations

Large self-contained calculations, such as Stark maps or pair-state interaction potentials, are wrapped as classes to enable easy access to the calculations parameters, with associated visualisation and exploration methods to make the results easy to disseminate. An added benefit is that calculations can be saved for future use, as illustrated in the example below:

```
calc = PairStateInteractions(Rubidium(),
                             60,0,0.5,60,0,0.5,   0.5,0.5)
# do something with initialised calculation
calc.defineBasis(0,0,  5,5,  25.e9)
calc.diagonalise(np.linspace(0.5,10.0,200),150)
saveCalculation(calc, "myCalculation.pkl")
```

This calculation can then be retrieved at a later time, or even on another machine, and continued, or further explored. For example:

```
calc = loadSavedCalculation("myCalculation.pkl")
# continue calculation
calc.plotLevelDiagram()
calc.showPlot()
```

4.5. Exporting data to file

If one wants to use the obtained results in another program, for further analysis and calculation, both `PairStateInteractions` and `StarkMap` provide `exportData` method that allows saving data in .csv format, that is human-readable and easy to import in other software tools. As a commented header of the exported files, ARC will record details of the calculation parameters, in human-readable form. Provided that the initial calculation is variable `calc`, its data is exported by calling

```
calc.exportData("rootFileName")
```

The program will output list of files (typically three files) storing the calculation data, whose names will be starting with "`rootFileName`".

4.6. Advanced interfacing of ARC with other projects

In addition to graphical exploration, and exporting relevant data as .csv files, advanced users incorporating ARC in their own projects might want to access directly Stark or pair-state interaction matrices, and corresponding basis states that are generated by `defineBasis` methods of the corresponding calculation classes. Basis (pair-) states are accessible for both methods in `basisStates` array. Stark matrix can be assembled as the sum of the diagonal matrix `mat1` recording state energies, and the off-diagonal matrix that depends on applied field and can be obtained as product of electric field and `mat2`. For pair-state interactions, the interaction matrix can be obtained as a sum of interatomic-distance independent matrix `matDiagonal`, recording energy defects of pair-states, and distance dependent matrices stored in `matR`, where `matR[0]`, `matR[1]` and `matR[2]` store respectively dipole–dipole, dipole–quadrupole and quadrupole–quadrupole interaction coefficients for interaction terms [Eq. (20)] scaling respectively with distance R as $\propto R^{-3}$, $\propto R^{-4}$ and $\propto R^{-5}$. For more details about this, examples of use, and other accessible calculated information, we refer the reader to the detailed documentation [19].

5. Outlook

This paper describes version 1.2 of ARC that can be downloaded from [21]. Supplementary information provides full documentation [19] for functions listed in Appendix B, as well as an IPython notebook [20] that contains more elaborate examples, code snippets and benchmarks against published results. This notebook provides an introduction to most of the capabilities of the ARC, as well as a good starting point for users, providing useful code snippets. Numerous examples also provide quantitative interactive introduction for anyone starting in the field of Rydberg atomic physics. In addition, for quick estimates, when one is in the lab, conference or other meeting, we provide a web-interface to the package [21]. For the latest versions of the package and documentation, please download material from the ARC GitHub page [18].

In the future, the package can be extended to include calculations of dressing potentials [52], magic wavelengths [53], atom-wall interactions [54,55], photoionisation, collisional cross-sections [56], tensor polarisability, molecular bound states [57], effects of magnetic fields, microwave tuning of interactions [17] and other atomic properties. These can be added as calculation tools, in separate classes, built on top of the existing library. Alkaline earth elements can be included too. While these were beyond the scope of the current project, we hope that this project can provide an initial seed for a much bigger community project. The code is hosted on GitHub [18], allowing easy community involvement and improvements. A proposal of some basic philosophy for the development is provided in the documentation [19].

We hope the library will increase accessibility to the existing knowledge. Up to now a lot of relevant information, although in principle derivable from existing literature, required quite lengthy and error-prone calculations. The developed hierarchical object-oriented structure allows one to retrieve relevant information at the appropriate level of abstraction, without the need to deal with lower-level details. In addition, we hope to establish growing code base for common calculations, in the spirit of other open-source community projects like, Numpy and SciPy [58], adding to the growing stack of atomic physics tools like ElecSus [59], The Software Atom [60] and QuTIP [61]. We especially highlight recent related development of C++ program Pairinteraction for pair-state calculations [62]. We hope that these efforts will allow more research groups to explore the rich physics achievable when one uses all the available transitions in the atoms. In addition to exploring Rydberg physics, library can be useful for practical

Table B.1
Class and function listing of `alkali_atom_functions` module.

Name (parameters)	Short description
`AlkaliAtom`([preferQuantumDefects, cpp_numerov])	Implements general calculations for alkali atoms (see Table B.2)
`NumerovBack`(innerLimit, outerLimit, kfun, ...)	Full Python implementation of Numerov integration
`saveCalculation`(calculation, fileName)	Saves calculation for future use
`loadSavedCalculation`(fileName)	Loads previously saved calculation
`printStateString`(n, l, j)	Returns state spectroscopic label for $\|n, l, j\rangle$

Table B.2
Methods and function listing of `alkali_atom_functions.AlkaliAtom` class. Typical relative uncertanties are obtained from comparison to measured values.

Name (parameters)	Short description (units)	Typical rel. accuracy
`getDipoleMatrixElement`(n1, l1, ...)	Reduced dipole matrix element ($a_0 e$)	$\sim 10^{-2}$
`getTransitionWavelength`(n1, l1, ...)	Calculated transition wavelength in vacuum (m)	$\sim 10^{-6}$
`getTransitionFrequency`(n1, l1, ...)	Calculated transition frequency (Hz)	$\sim 10^{-6}$
`getRabiFrequency`(n1, l1, j1, mj1, ...)	Returns a Rabi frequency (angular, i.e. $\Omega = 2\pi \times \nu$) for resonant excitation with a specified laser beam in the centre of TEM00 mode (rad s^{-1})	$\sim 10^{-2}$
`getRabiFrequency2`(n1, l1, j1, mj1, ...)	Returns a Rabi frequency (angular, i.e. $\Omega = 2\pi \times \nu$) for resonant excitation with a specified electric field driving amplitude (rad s^{-1})	$\sim 10^{-2}$
`getStateLifetime`(n, l, j[, ...])	Returns the lifetime of the state (s)	$\sim 10^{-2}$
`getTransitionRate`(n1, l1, j1, n2, ...)	Transition rate due to coupling to vacuum modes (black body included) (s^{-1})	$\sim 10^{-2}$
`getReducedMatrixElementJ_asymmetric`(n1, ...)	Reduced matrix element in J basis, defined in asymmetric notation ($a_0 e$)	$\sim 10^{-2}$
`getReducedMatrixElementJ`(n1, l1, ...)	Reduced matrix element in J basis, symmetric notation ($a_0 e$)	$\sim 10^{-2}$
`getReducedMatrixElementL`(n1, l1, ...)	Reduced matrix element in L basis, symmetric notation ($a_0 e$)	$\sim 10^{-2}$
`getRadialMatrixElement`(n1, l1, ...)	Radial part of the dipole matrix element ($a_0 e$)	$\sim 10^{-2}$
`getQuadrupoleMatrixElement`(n1, ...)	Radial part of the quadrupole matrix element ($a_0^2 e$)	$\sim 10^{-2}$
`getPressure`(temperature)	Vapour pressure at given temperature (Pa)	$\sim (1-5)\cdot 10^{-2}$
`getNumberDensity`(temperature)	Atom number density at given temperature (m^{-3})	$\sim (1-5)\cdot 10^{-2}$
`getAverageInteratomicSpacing`(...)	Returns average interatomic spacing in atomic vapour (m)	$\sim (1-5)\cdot 10^{-2}$
`corePotential`(l, r)	core potential felt by valence electron (a.u)	
`effectiveCharge`(l, r)	effective charge of the core felt by valence electron (a.u)	
`potential`(l, s, j, r)	core potential, including spin–orbit interaction (a.u)	
`radialWavefunction`(l, s, j, ...)	Radial part of electron wavefunction	
`getEnergy`(n, l, j)	Energy of the level relative to the ionisation level (eV)	$\sim 10^{-6}$
`getQuantumDefect`(n, l, j)	Quantum defect of the level.	
`getC6term`(n, l, j, n1, l1, j1, ...)	C6 interaction term for the given two pair-states ($h \times$ Hz m^6)	
`getC3term`(n, l, j, n1, l1, j1, ...)	C3 interaction term for the given two pair-states ($h \times$ Hz m^3)	
`getEnergyDefect`(n, l, j, n1, l1, ...)	Energy defect for the given two pair-states, $E(\|rr\rangle) - E(\|r'r''\rangle)$ (eV)	
`getEnergyDefect2`(n, l, j, nn, ll, ...)	Energy defect for the given two pair-states, $E(\|r_1 r_2\rangle) - E(\|r'r''\rangle)$ (eV)	
`updateDipoleMatrixElementsFile`()	Updates the file with pre-calculated dipole matrix elements.	
`getRadialCoupling`(n, l, j, n1, l1, j1)	Returns radial part of the coupling between two states (dipole, quadrupole) ($a_0 e$ or $a_0^2 e$)	$\sim 10^{-2}$
`getAverageSpeed`(temperature)	Average (mean) speed at a given temperature (m/s)	
`getLiteratureDME`(n1, l1, j1, n2, ...)	Returns literature information on requested transition	

exploration of multi-photon schemes for alkali-atoms excitation [63–65], ionisation [66,67] and upconversion [68–70], as well as state control, e.g. through dressing of the intermediate levels [71] and adiabatic transfers [72].

Acknowledgements

We thank D. Whiting, J. Keaveney, H. Busche, P. Huillery, T. Billam, C. Nicholas, I. Hughes, R. Potvliege and M. Jones for stimulating discussions. This work is supported by Durham University, EPSRC grant EP/N003527/1, FET-PROACT project "RySQ" (H2020-FETPROACT-2014-640378-RYSQ), EPSRC grant "Rydberg soft matter" (EP/M014398/1), DSTL and EPSRC grant EP/M013103/1. Data for the presented examples can be generated by the IPython notebook available in Supplementary material [20]. In the final stages of this project we have been informed by Sebastian Weber about a related efforts on C++ program for pair-state calculations [62].

Appendix A. Example program

Code below generates Fig. 3. For many more examples see Supplemental material [20].

```
from arc import *

# ==== GENERATE DATA (ARC) ====

atom = Rubidium()

pqn = []  # principal quantum number
y = []       # rate at T =0 K
ybb = []  # additional black-body induced transitions

# calculating decay from from 30 S_{1/2}
# to n P_{1/2} and n P_{3/2}
# where n is in range form 5 to 40

for n in xrange(5,40):
    pqn.append(n)

    # transition rate at T = 0K
    noBBR = atom.getTransitionRate(30, 0, 0.5,\
                                   n, 1, 0.5, temperature = 0)\
            + atom.getTransitionRate(30, 0, 0.5,\
                                   n, 1, 1.5, temperature=0 )
```

Table B.3
Class listing of alkali_atom_data module. All these classes inherit properties of alkali_atom_functions.AlkaliAtom from Table B.2.

Name (parameters)	Short description
Hydrogen([preferQuantumDefects, cpp_numerov])	Properties of hydrogen atoms
Lithium6([preferQuantumDefects, cpp_numerov])	Properties of lithium 6 atoms
Lithium7([preferQuantumDefects, cpp_numerov])	Properties of lithium 7 atoms
Sodium([preferQuantumDefects, cpp_numerov])	Properties of sodium 23 atoms
Potassium39([preferQuantumDefects, cpp_numerov])	Properties of potassium 39 atoms; alias Potassium(...)
Potassium40([preferQuantumDefects, cpp_numerov])	Properties of potassium 40 atoms
Potassium41([preferQuantumDefects, cpp_numerov])	Properties of potassium 41 atoms
Rubidium85([preferQuantumDefects, cpp_numerov])	Properties of rubidium 85 atoms; alias Rubidium(...)
Rubidium87([preferQuantumDefects, cpp_numerov])	Properties of rubidium 87 atoms
Caesium([preferQuantumDefects, cpp_numerov])	Properties of caesium 133 atoms

Table B.4
Method listing of calculations_atom_single.LevelPlot(atomType) class.

Name (parameters)	Shortdescription
makeLevels(nFrom, nTo, lFrom, lTo)	Constructs energy level diagram in a given range
drawLevels()	Draws a level diagram plot
showPlot()	Shows a level diagram plot

Table B.5
Method listing of calculations_atom_single.StarkMap(atom) class.

Name (parameters)	Short description
defineBasis(n, l, j, mj, nMin, ...)	Initialises basis of states around state of interest
diagonalise(eFieldList[, ...])	Finds atom eigenstates in a given electric field
plotLevelDiagram([units, ...])	Makes a plot of a stark map of energy levels
showPlot([interactive])	Shows plot made by plotLevelDiagram
savePlot([filename])	Saves plot made by plotLevelDiagram
exportData(fileBase[, exportFormat])	Exports StarkMap calculation data
getPolarizability([maxField, ...])	Returns the polarisability of the state (MHz cm^2/V^2)

Table B.6
Method listing of calculations_atom_pairstate.PairStateInteractions(atom, n, l, j, nn, ll, jj, m1, m2, interactionsUpTo=1) class that calculates Rydberg level diagram (spaghetti) for the given pair-state.

Name (parameters)	Short description
defineBasis(theta, ...)	Finds relevant states in the vicinity of the given pair-state
getC6perturbatively(...)	Calculates C6 from second order perturbation theory (GHz μm^6)
getLeRoyRadius()	Returns Le Roy radius for initial pair-state (μm)
diagonalise(rangeR, ...)	Finds eigenstates in atom pair basis
plotLevelDiagram([...])	Plots pair-state level diagram
showPlot([interactive])	Shows level diagram printed by plotLevelDiagram
exportData(fileBase[, ...])	Exports PairStateInteractions calculation data
getC6fromLevelDiagram(...)	Finds C_6 coefficient for original pair-state (GHz μm^6).
getC3fromLevelDiagram(...)	Finds C_3 coefficient for original pair-state (GHz μm^3).
getVdwFromLevelDiagram(...)	Finds r_{vdW} coefficient for original pair-state (μm).

Table B.7
Method listing of calculations_atom_pairstate.StarkMapResonances(atom1, state1, atom2, state2) class that calculates pair-state Stark maps for finding resonances.

Name (parameters)	Short description
findResonances(nMin, ...)	Finds near-resonant dipole-coupled pair-states
showPlot([interactive])	Plots initialstate Stark map and its dipole-coupled resonances

```
    # same, now at T= 300 K
    withBBR = atom.getTransitionRate(30, 0, 0.5,\
                        n, 1, 0.5, temperature=300.0)\
                + atom.getTransitionRate(30, 0, 0.5,\
                        n, 1, 1.5, temperature=300.0)
    y.append(noBBR)
    ybb.append(withBBR-noBBR)

# ==== PLOT(matplotlib) ====

pqn=np.array(pqn)
y = np.array(y)
ybb = np.array(ybb)

width = 0.4
f, ax = plt.subplots()
ax.bar(pqn-width/2.,y,width=width,color="r")
ax.bar(pqn+width/2.,ybb,width=width,color="g")
ax.set_xlabel("Principal quantum number, n")
ax.set_ylabel(r"Transition rate (s^-1)")
plt.legend(("Spontaneous decays","Black-body induced transitions")
      ,fontsize=10)
plt.xlim(4,40)

f.set_size_inches(5.50,4)
# save figure in decays.pdf
plt.savefig("decays.pdf", bbox_inches='tight')
```

Table B.8
Function and class listing of `wigner` module providing support for angular element calculations.

Name (parameters)	Short description
`Wigner3j`(j1,j2,...)	returns Winger 3j-coefficient
`Wigner6j`(j1,j2,...)	returns Wigner 6j-coefficient
`wignerDmatrix`(theta,phi)	Class for obtaining Wigner *D*-matrix

Appendix B. ARC function list

This appendix provides listings of functions, classes and methods contained within the ARC module. For detailed documentation, and more elaborate examples we refer the reader to the documentation in supplemental [19], also accessible via ARC website [21]. (See Tables B.1–B.8.)

Appendix C. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cpc.2017.06.015.

References

[1] D. Paredes-Barato, C.S. Adams, Phys. Rev. Lett. 112 (2014) 040501. http://dx.doi.org/10.1103/PhysRevLett.112.040501.
[2] M. Saffman, J. Phys. B At. Mol. Opt. Phys. 49 (20) (2016) 202001. http://dx.doi.org/10.1088/0953-4075/49/20/202001.
[3] J.D. Pritchard, K.J. Weatherill, C.S. Adams, Annu. Rev. Cold Atoms Mol., World Scientific, 2013, pp. 301–350. http://dx.doi.org/10.1142/9789814440400_0008.
[4] O. Firstenberg, C.S. Adams, S. Hofferberth, J Phys. B At. Mol. Opt. Phys. 49 (2016) 152003. http://dx.doi.org/10.1088/0953-4075/49/15/152003. arXiv:1602.06117.
[5] C. Murray, T. Pohl, in: E. Arimondo, C.C. Lin, S.F. Yelin (Eds.), Quantum and Nonlinear Optics in Strongly Interacting Atomic Ensembles, advances i ed., in: Adv. At. Mol. Opt. Phys., Academic Press, 2016, pp. 321–372 (Chapter 7). http://dx.doi.org/10.1016/bs.aamop.2016.04.005.
[6] C. Ates, T. Pohl, T. Pattard, J.M. Rost, Phys. Rev. A 76 (2007) 013413 http://dx.doi.org/10.1103/PhysRevA.76.013413. URL http://link.aps.org/doi/10.1103/PhysRevA.76.013413.
[7] M. Hoening, W. Abdussalam, M. Fleischhauer, T. Pohl, Phys. Rev. A 90 (2014) 021603(R).
[8] I. Lesanovsky, J.P. Garrahan, Phys. Rev. A 90 (2014) 011603. http://dx.doi.org/10.1103/PhysRevA.90.011603(R).
[9] A. Urvoy, F. Ripka, I. Lesanovsky, D. Booth, J.P. Shaffer, T. Pfau, R. Löw, Phys. Rev. Lett. 114 (2015) 203002. http://dx.doi.org/10.1103/PhysRevLett.114.203002. URL http://arxiv.org/abs/1408.0039.
[10] P. Schauß, J. Zeiher, T. Fukuhara, S. Hild, M. Cheneau, T. Macrì, T. Pohl, I. Bloch, C. Gross, Science 347 (2015) 1455. http://dx.doi.org/10.1126/science.1258351.
[11] N. Šibalić, C.G. Wade, C.S. Adams, K.J. Weatherill, T. Pohl, Phys. Rev. A 94 (2016) 011401(R). http://dx.doi.org/10.1103/PhysRevA.94.011401.
[12] J.A. Sedlacek, A. Schwettmann, H. Kübler, R. Löw, T. Pfau, J.P. Shaffer, Nat. Phys. 8 (2012) 819. http://dx.doi.org/10.1038/nphys2423. URL http://www.nature.com/doifinder/10.1038/nphys2423.
[13] M.T. Simons, J.A. Gordon, C.L. Holloway, Proc. SPIE 9747, Terahertz, RF, Millimeter, Submillimeter-Wave Technol. Appl. IX 97471F, 2016, pp. 1–7. http://dx.doi.org/10.1117/12.2213415. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.2213415.
[14] C.G. Wade, N. Šibalić, N.R. de Melo, J.M. Kondo, C.S. Adams, K.J. Weatherill, Nat. Photonics 11 (2017) 40. http://dx.doi.org/10.1038/nphoton.2016.214. URL http://www.nature.com/doifinder/10.1038/nphoton.2016.214.
[15] I.I. Ryabtsev, D.B. Tretyakov, I.I. Beterov, V.M. Entin, Phys. Rev. Lett. 104 (7) (2010) 073003. http://dx.doi.org/10.1103/PhysRevLett.104.073003. arXiv:0909.3239.
[16] M. Tanasittikosol, J.D. Pritchard, D. Maxwell, A. Gauguet, K.J. Weatherill, R.M. Potvliege, C.S. Adams, J. Phys. B At. Mol. Phys. 44 (2011) 184020. arXiv:1100226v3. URL http://iopscience.iop.org/0953-4075/44/18/184020.
[17] S. Sevinçli, T. Pohl, New J. Phys. 16 (2014) 123036. http://dx.doi.org/10.1088/1367-2630/16/12/123036.
[18] N. Šibalić, J.D. Pritchard, C.S. Adams, K.J. Weatherill, ARC GitHub page. URL https://github.com/nikolasibalic/ARC-Alkali-Rydberg-Calculator.
[19] N. Šibalić, J.D. Pritchard, C.S. Adams, K.J. Weatherill, ARC documentation - supplemental.zip > documentation > index.html, 2016. URL http://arc-alkali-rydberg-calculator.readthedocs.io.
[20] N. Šibalić, J.D. Pritchard, C.S. Adams, K.J. Weatherill, An introduction to Rydberg atoms with ARC - supplemental.zip > documentation > Rydberg_atoms_a_primer.html, 2016.
[21] N. Šibalić, J.D. Pritchard, C.S. Adams, K.J. Weatherill, Atom calculator web page. URL https://atomcalc.jqc.org.uk.
[22] C.-J. Lorenzen, K. Niemax, Phys. Scr. 27 (1983) 300.
[23] M. Marinescu, H.R. Sadeghpour, A. Dalgarno, Phys. Rev. A 49 (1994) 982. URL http://journals.aps.org/pra/abstract/10.1103/PhysRevA.49.982.
[24] S.A. Bhatti, C.L. Cromer, W.E. Cooke, Phys. Rev. A 24 (1) (1981) 161–165. http://dx.doi.org/10.1103/PhysRevA.24.161.
[25] B.V. Numerov, Mon. Not. R. Astron. Soc. 84 (1924) 592.
[26] B.V. Numerov, Astron. Nachr. 230 (1927) 359. http://dx.doi.org/10.1093/mnras/84.8.602.
[27] M.L. Zimmerman, M.G. Littman, M.M. Kash, D. Kleppner, Phys. Rev. A 20 (6) (1979) 2251. http://dx.doi.org/10.1103/PhysRevA.20.2251.
[28] A. Kramida, Y. Ralchenko, J. Reader, N.A.T., NIST Atomic Spectra Database (version 5.4), 2016. URL http://physics.nist.gov/asd.
[29] I.I. Sobelman, Atomic Spectra and Radiative Transitions, Springer-Verlag, Berlin, 1979.
[30] E.U. Condon, G.H. Shortley, The Theory of Atomic Spectra, Cambridge University Press, Cambridge, 1970.
[31] M.S. Safronova, W.R. Johnson, A. Derevianko, Phys. Rev. A 60 (1999) 4476. http://dx.doi.org/10.1103/PhysRevA.60.4476.
[32] M.S. Safronova, C.J. Williams, C.W. Clark, Phys. Rev. A 69 (2004) 022509. http://dx.doi.org/10.1103/PhysRevA.69.022509.
[33] C. Theodosiou, Phys. Rev. A 30 (1984) 2881. URL http://journals.aps.org/pra/abstract/10.1103/PhysRevA.30.2881.
[34] I.I. Beterov, I.I. Ryabtsev, D.B. Tretyakov, V.M. Entin, Phys. Rev. A 79 (2009) 052504. http://dx.doi.org/10.1103/PhysRevA.79.052504. URL http://link.aps.org/doi/10.1103/PhysRevA.79.052504.
[35] K. Singer, J. Stanojevic, M. Weidemüller, R. Coté, J Phys. B 38 (2) (2005) S295. http://dx.doi.org/10.1088/0953-4075/38/2/021.
[36] R.J.L. Roy, Can. J. Phys. 52 (1974) 246. http://dx.doi.org/10.1139/p74-035.
[37] J. Deiglmayr, H. Saßmannshausen, P. Pillet, F. Merkt, Phys. Rev. Lett. 113 (2014) 193001. http://dx.doi.org/10.1103/PhysRevLett.113.193001. arXiv:1410.0579.
[38] T.F. Gallagher, K.A. Safinya, F. Gounand, J.F. Delpech, W. Sandner, R. Kachru, Phys. Rev. A 25 (4) (1982) 1905. http://dx.doi.org/10.1103/PhysRevA.25.1905.
[39] T. Vogt, M. Viteau, J. Zhao, A. Chotia, D. Comparat, P. Pillet, Phys. Rev. Lett. 97 (8) (2006) 083003. http://dx.doi.org/10.1103/PhysRevLett.97.083003.
[40] S. Ravets, H. Labuhn, D. Barredo, T. Lahaye, A. Browaeys, Phys. Rev. A 92 (2015) 020701. http://dx.doi.org/10.1103/PhysRevA.92.020701.
[41] I.I. Beterov, M. Saffman, Phys. Rev. A 92 (2015) 042710. http://dx.doi.org/10.1103/PhysRevA.92.042710. arXiv:1508.07111.
[42] Anaconda Python distribution install. URL https://docs.continuum.io/anaconda/install..
[43] R.C. Whaley, J.J. Dongarra, Proc. 1998 ACM/IEEE Conf. Supercomput, IEEE Computer Society, 1998, pp. 1–27.
[44] P. Goy, J. Liang, M. Gross, S. Haroche, Phys. Rev. A 34 (1986) 2889. http://dx.doi.org/10.1103/PhysRevA.34.2889.
[45] W. Li, I. Mourachko, M.W. Noel, T.F. Gallagher, Phys. Rev. A 67 (2003) 052502. http://dx.doi.org/10.1103/PhysRevA.67.052502.
[46] J. Han, Y. Jamil, D.V.L. Norum, P.J. Tanner, T.F. Gallagher, Phys. Rev. A 74 (5) (2006) 054502. http://dx.doi.org/10.1103/PhysRevA.74.054502.
[47] M. Mack, F. Karlewski, H. Hattermann, S. Höckh, F. Jessen, D. Cano, J. Fortágh, Phys. Rev. A 83 (2011) 052515. http://dx.doi.org/10.1103/PhysRevA.83.052515.
[48] K. Afrousheh, P. Bohlouli-Zanjani, J.A. Petrus, J.D.D. Martin, Phys. Rev. A 74 (2006) 062712. http://dx.doi.org/10.1103/PhysRevA.74.062712.
[49] P. Goy, J.M. Raimond, G. Vitrant, S. Haroche, Phys. Rev. A 26 (5) (1982) 2733. http://dx.doi.org/10.1103/PhysRevA.26.2733.
[50] K.-H. Weber, C.J. Sansonetti, Phys. Rev. A 35 (1987) 4650. URL http://journals.aps.org/pra/abstract/10.1103/PhysRevA.35.4650.
[51] J. Deiglmayr, H. Herburger, H. Saßmannshausen, P. Jansen, H. Schmutz, F. Merkt, Phys. Rev. A 93 (2016) 013424. http://dx.doi.org/10.1103/PhysRevA.93.013424.
[52] R.M.W. van Bijnen, T. Pohl, Phys. Rev. Lett. 114 (2015) 243002. http://dx.doi.org/10.1103/PhysRevLett.114.243002. URL http://link.aps.org/doi/10.1103/PhysRevLett.114.243002.
[53] E.A. Goldschmidt, D.G. Norris, S.B. Koller, R. Wyllie, R.C. Brown, J.V. Porto, U.I. Safronova, M.S. Safronova, Phys. Rev. A 91 (2015) 032518. http://dx.doi.org/10.1103/PhysRevA.91.032518. arXiv:1503.02881v1.
[54] A. Derevianko, W.R. Johnson, M.S. Safronova, J.F. Babb, Phys. Rev. Lett. 82 (1999) 3589. http://dx.doi.org/10.1103/PhysRevLett.82.3589.
[55] D. Bloch, M. Ducloy, in: B. Bederson, H. Walther (Eds.), Atom Wall Interaction, in: Adv. At. Mol. Opt. Phys., vol. 50, Academic Press, 2005, p. 91. URL http://www.sciencedirect.com/science/article/pii/S1049250X05800084.
[56] M. Kiffner, D. Ceresoli, W. Li, D. Jaksch, J. Phys. B At. Mol. Opt. Phys. 49 (2015) 204004. http://dx.doi.org/10.1088/0953-4075/49/20/204004. arXiv:1507.03357.
[57] A. Gaj, A.T. Krupp, J.B. Balewski, R. Löw, S. Hofferberth, T. Pfau, Nature Commun. 5 (2014) 4546. http://dx.doi.org/10.1038/ncomms5546. URL http://www.nature.com/ncomms/2014/140801/ncomms5546/full/ncomms5546.html.

[58] T.E. Oliphant, Comput. Sci. Eng. 9 (2007) 10. http://dx.doi.org/10.1109/MCSE.2007.58.
[59] M.A. Zentile, J. Keaveney, L. Weller, D.J. Whiting, C.S. Adams, I.G. Hughes, Comput. Phys. Comm. 189 (2015) 162. http://dx.doi.org/10.1016/j.cpc.2014.11.023.
[60] J. Javanainen, Comput. Phys. Comm., http://dx.doi.org/10.1016/j.cpc.2016.09.017, arXiv:1610.00791.
[61] J.R. Johansson, P.D. Nation, F. Nori, Comput. Phys. Comm. 184 (4) (2013) 1234–1240. http://dx.doi.org/10.1016/j.cpc.2012.11.019. arXiv:1211.6518.
[62] S. Weber, C. Tresp, H. Menke, A. Urvoy, O. Firstenberg, H.P.. Büchler, S. Hofferberth, J. Phys. B 50 (2017) 133001. http://dx.doi.org/10.1088/1361-6455/aa743a. URL https://pairinteraction.github.io/.
[63] H.S. Moon, L. Lee, J.B. Kim, J. Opt. Soc. Amer. B 24 (2007) 2157. http://dx.doi.org/10.1364/JOSAB.24.002157. URL http://www.opticsinfobase.org/abstract.cfm?URI=josab-24-9-2157.
[64] H.G. Lee, H. Kim, J. Lim, J. Ahn, Phys. Rev. A - At. Mol. Opt. Phys. 88 (2013) 053427. http://dx.doi.org/10.1103/PhysRevA.88.053427.
[65] J.M. Kondo, N. Šibalić, A. Guttridge, C.G. Wade, N.R. De Melo, C.S. Adams, K.J. Weatherill, Opt. Lett. 40 (2015) 5570. http://dx.doi.org/10.1364/OL.40.005570.
[66] E. Courtade, M. Anderlini, D. Ciampini, J.H. Muller, O. Morsch, E. Arimondo, J. Opt. Soc. Amer. B 37 (2004) 967. http://dx.doi.org/10.1364/JOSAB.21.000480.
[67] C. Ates, I. Lesanovsky, C.S. Adams, K.J. Weatherill, Phys. Rev. Lett. 110 (2013) 213003. http://dx.doi.org/10.1103/PhysRevLett.110.213003.
[68] A.M. Akulshin, R.J. McLean, A.I. Sidorov, P. Hannaford, Opt. Express 17 (2009) 22861. http://dx.doi.org/10.1364/OE.17.022861. arXiv:0910.2292.
[69] C.V. Sulham, G.A. Pitz, G.P. Perram, Appl. Phys. B Lasers Opt. 101 (2010) 57. http://dx.doi.org/10.1007/s00340-010-4015-9.
[70] A. Akulshin, D. Budker, R. McLean, Opt. Lett. 39 (2014) 845. URL http://www.ncbi.nlm.nih.gov/pubmed/24562222.
[71] N. Šibalić, J.M. Kondo, C.S. Adams, K.J. Weatherill, Phys. Rev. A 94 (2016) 033840. http://dx.doi.org/10.1103/PhysRevA.94.033840.
[72] D. Maas, C. Rella, P. Antoine, E. Toma, L. Noordam, Phys. Rev. A 59 (1999) 1374. http://dx.doi.org/10.1103/PhysRevA.59.1374. URL http://link.aps.org/doi/10.1103/PhysRevA.59.1374.