Computer Physics Communications 215 (2017) 137–148

Contents lists available at ScienceDirect

Computer Physics Communications

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

TIMEDELn: A programme for the detection and parametrization of overlapping resonances using the time-delay method[☆]

Duncan A. Little[a], Jonathan Tennyson[a,*], Martin Plummer[b,*], Clifford J. Noble[b,1], Andrew G. Sunderland[b]

[a] Department of Physics & Astronomy, University College London, Gower Street, London WC1E 6BT, United Kingdom
[b] Scientific Computing Department, STFC Daresbury Laboratory, Sci-tech Daresbury, Cheshire WA4 4AD, United Kingdom

ARTICLE INFO

Article history:
Received 2 July 2016
Received in revised form 19 November 2016
Accepted 6 January 2017
Available online 16 January 2017

Keywords:
(Overlapping) resonances
Scattering
Time-delay
K-matrix
Branching ratio
Eigenphase
Multi-level parallelism

ABSTRACT

TIMEDELn implements the time-delay method of determining resonance parameters from the characteristic Lorentzian form displayed by the largest eigenvalues of the time-delay matrix. TIMEDELn constructs the time-delay matrix from input K-matrices and analyses its eigenvalues. This new version implements multi-resonance fitting and may be run serially or as a high performance parallel code with three levels of parallelism. TIMEDELn takes K-matrices from a scattering calculation, either read from a file or calculated on a dynamically adjusted grid, and calculates the time-delay matrix. This is then diagonalized, with the largest eigenvalue representing the longest time-delay experienced by the scattering particle. A resonance shows up as a characteristic Lorentzian form in the time-delay: the programme searches the time-delay eigenvalues for maxima and traces resonances when they pass through different eigenvalues, separating overlapping resonances. It also performs the fitting of the calculated data to the Lorentzian form and outputs resonance positions and widths. Any remaining overlapping resonances can be fitted jointly. The branching ratios of decay into the open channels can also be found. The programme may be run serially or in parallel with three levels of parallelism. The parallel code modules are abstracted from the main physics code and can be used independently.

New version programme summary
Programme Title: TIMEDELn
Programme Files doi: http://dx.doi.org/10.17632/wmv4f42xnz.1
Licencing provisions: MIT
Programming language: FORTRAN
Journal reference of previous version: Computer Phys. Comms., **114**, 236–242 (1998).
Does the new version supersede the previous version?: Yes
Nature of problem: TIMEDELn detects and parametrizes resonances, including overlapping resonances when provided with the K-matrix of the scattering problem.
Solution method: Resonances are identified by peaks in the largest few eigenvalues of the time-delay matrix.
Reasons for the new version: TIMEDELn includes a new procedure for fitting multiple overlapping resonances. It has also been parallelized to allow studies of complex systems (atoms and molecules) and generation of bulk data.
Summary of revisions: TIMEDELn analyses the largest eigenvalues of the time-delay matrix and identifies those with resonance features which are then separated and fitted [6]. It has been modularized with calls to external libraries and user supplied routines abstracted for ease of modification. It has been parallelized, with a choice of a specific module allowing multi-level parallel structures or serial execution if preferred. It can run bulk simulations of 'similar but different' calculations (for example, varying fixed-nuclear geometries).

☆ This paper and its associated computer programme are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding authors.
E-mail addresses: duncan.little.11@ucl.ac.uk (D.A. Little), j.tennyson@ucl.ac.uk (J. Tennyson), martin.plummer@stfc.ac.uk (M. Plummer), andrew.sunderland@stfc.ac.uk (A.G. Sunderland).
1 Deceased.

http://dx.doi.org/10.1016/j.cpc.2017.01.005

Restrictions: When 'target' energies are calculated or supplied, the energy of the incident particle (electron) is currently defined with respect to the lowest supplied target energy (the ground state), although an expert user or developer would be able to modify this.
Unusual features: TIMEDELn can be run from a user-supplied file for K-matrices or can be implemented to generate these as required.
External routines/libraries: Lapack [1], Minpack [2], options for alternatives (e.g. NAG [3]), option for MPI [4]
Additional comments: TIMEDELn has been implemented as part of the UKRMol suite of codes [7].

[1] E. Anderson et al., LAPACK Users' Guide, third edition, (Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999) http://www.netlib.org/lapack/
[2] LMDIF1 and dependencies, MINPACK Fortran numerical library (University of Chicago, Argonne National Laboratory, USA, 1999), http://www.netlib.org/minpack/
[3] NAG Fortran Library Mark 25 (Numerical Algorithms Group, Oxford, UK, 2015), http://www.nag.co.uk/numeric/fl/FLdescription.asp/
[4] The Message Passing Interface, standards for MPI are available from the MPI Forum, http://www.mpi-forum.org/
[5] D.T. Stibbe and J. Tennyson, *Computer Phys. Comms.*, **114**, 236–242 (1998).
[6] D.A. Little and J. Tennyson, *J. Phys. B: At. Mol. Opt. Phys.*, **47**, 105204 (2014).
[7] J.M. Carr, P.G. Galiatsatos, J.D. Gorfinkiel, A.G. Harvey, M.A. Lysaght, D. Madden, Z. Masin, M. Plummer, J. Tennyson, H.N. Varambhia, *Eur. Phys. J. D*, **66**, 58 (2012)

1. Introduction

Resonances are formed when colliding particles become temporarily trapped in long-lived states in the continuum. Resonances are important in almost all low to intermediate energy collision processes and dominant in some of them. This makes resonance detection and characterization an important part of many theoretical procedures used for studying collision physics.

A number of methods are available for characterizing resonances; specifically for electron scattering [1]. These include methods based on the fitting of eigenphase sums [2–4] plus others which exploit the properties of the S-matrix [5] and of the R-matrix formalism [6]. In this work we focus on the use of time-delays to characterize resonances. The 'time-delay method' was found to give the most reliable and complete determination of the resonance parameters in a recent comprehensive comparison of resonance detection methods for the electron –N_2^+ collision problem [7].

The time-delay method was originally proposed by Smith [8] as an alternative method of characterizing resonances to the *de facto* fitting of the eigenphase sum. In classical terms the time-delay can be thought of as the difference in time a colliding particle experiences with or without an interaction with the target. The time-delay matrix is formed at a scattering energy E from the scattering matrix, **S**, and the time operator, $-i\hbar\frac{d}{dE}$:

$$\mathbf{Q}(E) = -i\hbar\mathbf{S}^*\frac{d\mathbf{S}}{dE}, \tag{1}$$

and has dimensions $n_o \times n_o$ where n_o is the number of open channels. By diagonalizing the time-delay matrix eigenvalues, q_i, and eigenvectors, $|\psi_{q_i}\rangle$ can be found, that is,

$$\mathbf{Q}|\psi_{q_i}\rangle = q_i|\psi_{q_i}\rangle. \tag{2}$$

The largest eigenvalue of **Q**, q_1, represents the longest time-delay of the incident electron [8]. The trace of **Q**, the sum of the eigenvalues, may be represented as a sum of Lorentzian functions as it varies with energy. These Lorentzian functions describing the resonances combined with a non-resonant background (see, for example, [9,10]). If the resonances do not overlap, these Lorentzians show up clearly as peaks in the largest eigenvalue q_1 which can be fitted for position and width. Additionally, the branching ratios β_i, the probability of decay into a different asymptotic channel, can be found using the squared moduli of the components of the normalized eigenvector associated with q_1, that is, if $|\psi_{q_1}\rangle$ has components $\alpha_{j,1}, j = 1, \ldots, n_o$, then

$$\beta_i = \left|\alpha_{i,1}\right|^2. \tag{3}$$

Recently, there have been a number of other studies on use of the time-delay method for characterizing resonances [9–12]. The method has been successfully applied to a number of systems with many overlapping resonances [7,10,13–17].

Resonance detection and fitting becomes particularly complicated in the case of multiple, overlapping resonances. For example, in the case of electron collisions with charged targets, such as electron – N_2^+ [7], there are many resonances associated with each Rydberg series converging on each of the excited states of the ion; these resonances overlap considerably with each other and valence states embedded in the continuum. Resonances in an eigenphase sum have a Breit–Wigner form [2], which has the appearance of an arctan function. Separating the behaviour of resonances of this form becomes increasingly difficult as their width and energy separation decreases. The Lorentzian form of resonances in a time-delay means that isolating a single resonance out of many overlapping resonances is greatly simplified. Multiple eigenvalues of the time-delay matrix can be used so that resonances can be tracked as they move from being the longest time-delay (the largest eigenvalue) to being the second longest (the second largest eigenvalue) and so forth. Indeed, this is the principle on which the newly developed fitting method presented here is based. The alternative procedure of directly fitting a sum of Lorentzian functions plus a background term to the trace of **Q**, has been successfully used by Aiba et al. [10] for continuum resonances of He and Ps^-. Additionally, the fitting of the time-delay removes the majority of the background which can be significant with many channels, a characteristic of electron–molecule collisions.

Stibbe and Tennyson developed a time-delay procedure which they implemented with the UK molecular R-matrix codes [18]. This procedure, which was designed to detect and fit single resonances, was made available as an independent code TIMEDEL [19]. Reference [19] also summarizes the general process of calculating the time-delay matrix on an adaptive energy grid. In this work we present an updated version of this code. The new code, TIMEDELn, allows for the detection, separation and fitting of overlapping resonances by considering the largest few eigenvalues of the time-delay matrix. One drawback of the time-delay method compared to eigenphase fitting with a code such as RESON [3] is that fitting large numbers of resonances is comparatively expensive computationally; RESON simply diagonalizes the K-matrix whereas TIMEDEL has to construct both the S-matrix and its derivative, and then perform a diagonalization. Furthermore, experience shows [7] that in complicated cases TIMEDEL successfully fits many more resonances than RESON. For these reasons we have also explored methods of implementing a parallelized version of TIMEDELn. The results of these studies are also presented here.

2. Implementation

As before [19], TIMEDELn has two distinct sections; the first computes the time-delay matrix, **Q**, over a given energy range and requires the input of K-matrices, the second fits the computed time-delay matrix for position and width and finds branching ratios if appropriate. The K-matrices required by TIMEDELn as its basic input can simply be provided for fitting resonances, or TIMEDELn can be linked directly into the K-matrix generating code so that it can specify the energies for which it requires the K-matrices to be computed based on an adaptive energy grid. The code is implemented in this fashion as part of the UKRMol codes [20]. The UKRMol codes are available as freeware from the CCPForge programme repository (http://ccpforge.cse.rl.ac.uk/) and can also be run using the Quantemol-N expert system [21].

In this implementation the code is also designed to avoid calculations in the threshold region, that is, just above the energy of the ground state or above or below the energy of an excited state of the target molecule. The time-delay of an electron with zero kinetic energy is infinite, thus just above a threshold, where the kinetic energy of the electron is close to zero, significant numerical problems are encountered. To avoid this region the time-delay calculation is started at some energy above a threshold (default 0.05 eV). We note that this can cause problems with Feshbach resonances which lie very close to their parent target state [22]. Furthermore, for charged systems, when approaching a threshold from below, the width of the Rydberg states converging on the threshold above becomes increasingly narrow. As the time-delay matrix is calculated using a numerical derivative with respect to the energy there is a point at which the width of the resonance is smaller than the energy gap with which the numerical derivative is calculated. Therefore a maximum principal quantum number of the Rydberg series is set (default $n = 10$) at which the calculation stops before it reaches the threshold. The time-delays are calculated within each threshold energy range, analysed for resonances and then any resonances detected are fitted. The calculation is stopped when it reaches the final energy of the overall range to be calculated.

The time-delay matrix **Q** is found using a numerical derivative of the S-matrix, see Eq. (1). The S-matrix is found from the K-matrix by the way of

$$\mathbf{S} = \frac{1 + i\mathbf{K}}{1 - i\mathbf{K}}. \tag{4}$$

K-matrices are required or calculated at energies $E + dE/2$ and $E - dE/2$ where dE is by default at most 10^{-5} 'input energy units' (eV, Ryd, Hartree) as specified by the user, who may also change this limit. Tests show that this value gives smooth time-delays for each threshold energy range. The derivative of the S-matrix with respect to energy is found using the K-matrices and is multiplied by the complex conjugate of the average of the two S-matrices to find **Q**, see Eq. (1). Only the eigenvalues of **Q** are calculated at this point; the eigenvectors are calculated in the fitting process to find branching ratios once the position of a resonance has been determined. An adaptive grid of energies is used across each range. The energy separation of each grid point is proportional to the inverse of the time-delay; the narrowest resonances therefore have the highest density of points and areas where there are no resonances are skipped over. The grid is limited by default to having a minimum spacing of 10^{-15} 'input energy units' (the initial, and maximal, grid spacing has default 0.1 units). Other values may be specified.

Once the eigenvalues have been found over an entire threshold range the module enters the fitting routine. Multiple resonances appear as interspersed Lorentzians in the highest eigenvalue, see Fig. 1. Discontinuities occur when the length of the time-delay of one resonance overtakes that of another; the eigenvalues associated with each resonance switch. Consequently, if only the first eigenvalue is fitted, information is lost when a resonance becomes the second and third eigenvalues. An example of this is given in Fig. 1. The original version of TIMEDEL [19] only fitted the largest eigenvalue, this eigenvalue is plotted against energy in panel (a) of Fig. 1. Only two resonances, one at ∼0.527 eV and another at ∼0.595 eV, are completely apparent. Another resonance at ∼0.52725 eV is partially obscured by the resonance at ∼0.527 eV. If the second and third eigenvalues are included then the structure of the wider resonances obscured by the narrow resonance ∼0.527 eV is elucidated. Although this extra information is often unnecessary as Lorentzians are symmetric functions, it becomes important when the peak of the resonance is in one of the lower eigenvalues. This description may be compared with the discussion by Aiba et al. [10] (see their figures 2–5) on characterizing overlapping resonances.

Indeed, this development reveals resonances that would have been previously left completely unidentified, those which have a similar position and marginally larger width to that of another resonance. Finding these resonances becomes particularly important when working in a lower symmetry group to that of the molecule being studied; resonances that appear in the same (lower) symmetry group but actually have different symmetries may be obscured by one another. The peak of a resonance of this type is within the second eigenvalue and only the tails switch into the first eigenvalue, for example the lowest resonance feature shown in panel (c) of Fig. 1.

A new subroutine, EIGSORT, tracks the resonances as they switch between eigenvalues by finding avoided crossings. Avoided crossings are detected as minima in the difference between two eigenvalues. When a minimum is detected the eigenvalues associated with each resonance are switched. Resonances are detected by searching for maxima, checks are performed to ensure a maximum is the peak of a resonance and not a discontinuity left by the EIGSORT routine or numerical noise. Fitting limits are set by a change in sign of the derivative $\frac{dq}{dE}$ on either side of the resonance peak. The resonances are then fitted with the form:

$$q(E) = \frac{\hbar \Gamma}{(E - E_r)^2 + (\Gamma/2)^2} + bg(E) \tag{5}$$

where $q(E)$ is the fit to the sorted time-delays (i.e. the sorted combination of eigenvalues), E_r is the resonance peak position, Γ is the width and $bg(E)$ is the background (constant over the range of each resonance: note that the output values of $bg(E)$ and the standard deviation of the fit are given in atomic units with $\hbar$ = 1). If the fitted peak of the resonance is outside the fitting limits then it is clear that this is a false detection or a bad fit and the fit is ignored. Panel (c) of Fig. 1 gives an example of fitting the sorted eigenvalues. Although this method is subject to occasional false detections, extensive testing on N_2^+ showed that it was robust and produced excellent fits [7]. Indeed this work demonstrated that for the electron – N_2^+ problem TIMEDELn was capable of successfully characterizing many more resonances than Breit–Wigner fits to the eigenphase sum [3], and that use of the R-matrix specific QB method [6] did not give accurate results.

TIMEDELn also calculates a standard deviation for the fit, see Eq. (6) of [19], using a sum of squared differences between Eq. (5) and the actual computed time-delays, over grid points in the energy range $E_r \pm \Gamma$. We note that if the sorting procedure still leaves some resonances overlapping within their fitting limits, which should be studied carefully on the equivalent of Fig. 1, then the fit can revert to a sum over Lorentzians as in Eq. (5) of [19] which approximates the trace of **Q** with the largest eigenvalue, or the programme can be forced to fit resonances individually. We recommend checking both these options. In these cases, visual inspection of the time-delay eigenvalues, output both before and after sorting, is strongly recommended to gain insight, see also Aiba et al. [10]. More accurate K-matrices may be needed to remove inaccuracies, or for genuine resonances, the programme allows data from re-runs over finer localized energy grids to be combined with existing data to aid the correct interpretation of the resonant structure. In general, the output time-delay eigenvalues should ideally always be inspected and/or plotted in case of missed resonances, in particular if the main output from TIMEDELn includes resonance fitting with a high standard deviation, or reports that possible resonances have been rejected from the fitting procedure. There are, however, cases where visually inspecting every resonance is not feasible, for example, when calculating potential energy curves of electronic states embedded in the continuum [7]. In this case plotting the potential energy curves reveals poorly fitted resonances which appear as discontinuities in otherwise smooth curves. Additionally, the magnitude of the resonance width is very sensitive to poor fitting, thus plotting width against internuclear separation may also reveal poorly fitted resonances which were not apparent when plotting the potential energy curve.

When the position of a resonance has been determined the time-delay matrix is recomputed at this energy and the peak branching ratios are computed from Eq. (3). The branching ratio gives the probability of the autoionization of the resonant electron to a partial wave of a specific channel associated with an electronic state of the ion. Partial widths for each partial wave of a channel are calculated by multiplying Γ by the β_i. The partial width for a specific channel can be found by summing over the partial widths of the partial waves, as was done by Little et al. [23]. Having autoionization widths resolved in this way is necessary for a dissociative recombination cross-section calculation which includes core-excited bound states, see Little et al. [23]. The extraction of branching ratios using (3) is accurate when the resonance is isolated and the peak appears in the largest eigenvalue, as shown by Smith [8]. When resonances are strongly overlapping, the relationship between the time-delay eigenvalues and the branching ratios is not given straightforwardly by (3), as investigated by Shimamura [11]. However, we have found empirically that if the separation leads to clearly defined Lorentzian peaks (with the peak maxima originally in the largest eigenvalue) then the formula gives consistent results away from the avoided crossings (see [23], also data from the current indicative benchmarks and test runs are available from the authors). The choice to use the calculated branching ratios should be carefully considered by the user. Branching ratios are also functions of energy and can vary across the resonance, although in the case of [23] the branching ratios were effectively constant across the narrow resonances (supported by data from the indicative benchmarks and more broadly from the pyrimidine test run in Section 4.3). The code provides options to calculate branching ratios across a range of energies within the resonance in the routine BRANCHING_RATIOS.

Extending the fitting method to overlapping resonances in this way is similar in concept to the approach introduced by Shimamura et al. [9]. In their approach resonances are parametrized by a mixing parameter β which describes the level of avoidance between two eigenvalues of **Q**. An expression is then derived to fit these two resonances as they overlap with each other. The method described here differs in that it is purely numerical and an expression was not formally derived to describe overlapping resonances. The fitting routine simply searches for minima in the difference between the eigenvalues and switches when one is found.

3. Programme structure

The parallel structure of TIMEDELn has been carefully written so that the physics modules contain no direct calls to parallel routines and use a minimal number of control parameters: the majority of the parallel calls and structure control is kept within a module serial_parallel which in turn calls routines in a module comm_mpi, the latter containing interfacing routines to direct MPI [24] calls, and to module sgc_mod which contains a scalable global counter. In the supplied code, serial_parallel comes in two versions, the full parallel version and a serial implementation version which contains dummy routines and sets benign values of the public control parameters. Thus the file ptimedel.f90 which contains the actual resonance finding and fitting (PTIMEDEL and subsidiary routines) is identical for serial and parallel runs.

The only other direct communication with the parallel and other control routines is in the 'main' programme. The serial version simply calls PTIMEDEL (and any other sub-programmes the user may also wish to run) whereas the parallel main code first changes to multiple sub-directories for simultaneous runs as required by the user, splits the MPI_COMM_WORLD communicator to match these subdirectory calculations, calls PTIMEDEL and finally calls MPI_Finalize: again, other sub-programme calls may be inserted by the user (expert user-developers may wish to return to the initial directory for other work). This initial parallelization allows controlled task-farming of many 'similar but different' jobs and was originally developed to allow bulk production of resonance data for the N_2 calculations (as in [23,25]) at various fixed internuclear separations in a single job, independently of the hardware and operating system. More generally it can be used for bulk calculations for different molecular geometries, or different irreducible representations, although it is up to the user to make sure the 'similar but different' rule applies, to maintain reasonable load balancing.

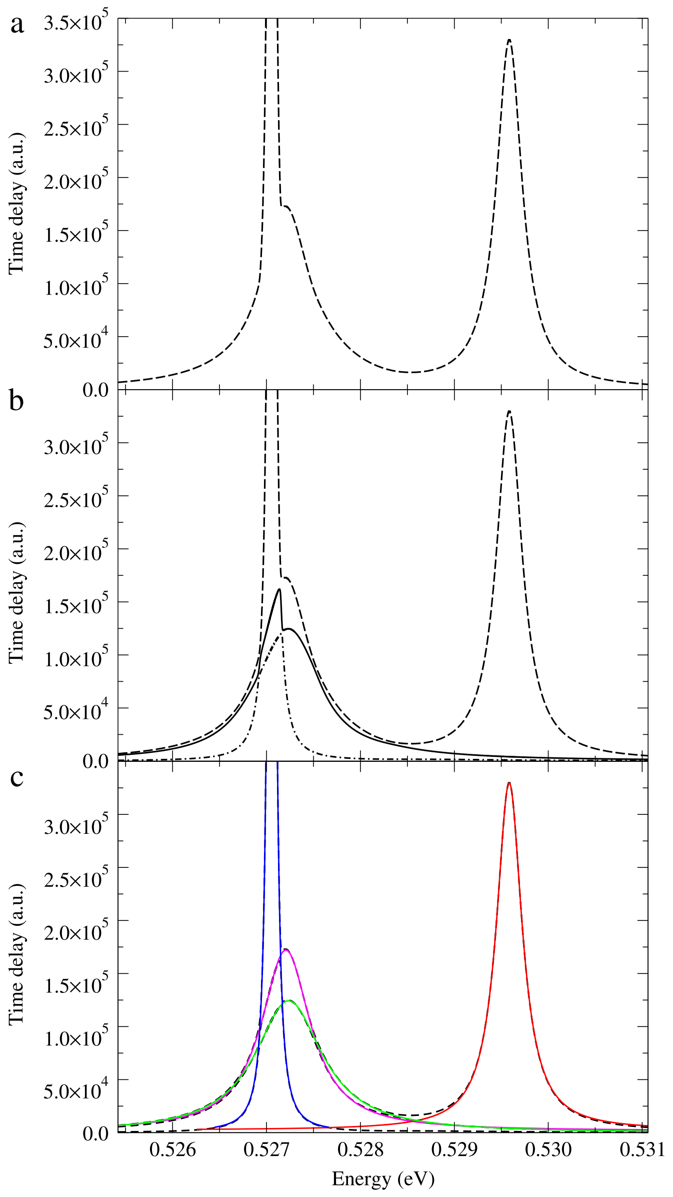

Fig. 1. An example of fitting overlapping resonances using multiple eigenvalues. The original version of TIMEDEL only fitted the longest eigenvalue (dashed line) of the time-delay matrix, this eigenvalue is plotted against energy in panel (a). If the second (solid line) and third (dashed-dot line) longest eigenvalues are included, as shown in panel (b), it becomes clear that a significant amount of information is being ignored if only the longest eigenvalue is fitted. That is, the green and magenta resonances shown in panel (c) would have been badly fitted or missed entirely. This example is taken from a calculation on electron - N_2^+ collisions [7]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In addition, for serial and parallel cases the calls to the user supplied routines for producing K-matrices, and to the mathematical method library routines for linear algebra and minimization are in distinct interfacing modules, respectively timedel_user_k_matrices and timedel_maths_library_calls.

timedel_user_k_matrices contains the storage arrays for the K-matrices and target energies, and interface routines K_MATRIX_SETUP and GET_K_MATRICES, called from PTIMEDEL, which respectively call the user-supplied routines SETUPKMAT (energy-independent preparation) and GETKMAT (the K-matrix at a given energy, called twice within GET_K_MATRICES for the pair of energies). Any modifications of the user supplied routines and calls are thus restricted to this module: the K-matrix call can be parallelized within a supplied MPI sub-communicator (the supplied code includes an example parallel call).

timedel_maths_library_calls contains interfacing routines called from PTIMEDEL and its subsidiary routines which directly call library routines. The module routines supplied contain options for LAPACK [26] (ZHEEV, ZGESV) and MINPACK [27] (LMDIF1) or the NAG library [28] (f02haf, f04adf, e04fyf), the NAG calls are commented in the version supplied. The use of the interfacing module again allows straightforward modifications or replacements by the user.

We note that a module timedel_data_for_minimization contains data arrays and subsidiary subroutines (in a choice of scalar and vectorized form) required for the LMDIF1/e04fyf routines. This is separate from the main ptimedel_mod module to maintain a straightforward compilation structure. There is also a module precisn which sets the required precision of real variables and certain others (details are in the accompanying programme user's manual), a module timedel_data_module which stores the namelist input data and some associated arrays and parameters, and a module read_serial_parallel which reads in the input namelist (one task) and calls routines in serial_parallel to cascade the information through the communicators (or do nothing) as required. A utility routine STDEV, which evaluates the standard deviation of each fit, is supplied in both simple scalar and vector alternative versions, to make use of current architectures which include automatic vectorization.

Finally, at various points in PTIMEDEL, the system routines CPU_CLOCK and SYSTEM_CLOCK are called to get CPU and elapsed timing information respectively. These calls play no part in the calculation. The structure of TIMEDELN is shown in Figs. 2 and 3.

3.1. *Serial ('physics code') structure: ptimedel*

The current implementation of PTIMEDEL was tested and used with the UKRmol code suite, in particular a module k_adapt. This module is essentially a customized stripped down version of UKRMol module rsolve [20] and contains the necessary routines to set up the K-matrix calculation (SETUPKMAT) and calculate the K-matrices at the energies required by the adaptive grid (GETKMAT). This module also reads in the target state energies used to calculate thresholds.[2]

PTIMEDEL starts by reading in the namelist TIME (see description of inputs below). The user is required to input an initial and final energy; all other inputs have default values. The K-matrix calculation is then set up by the subroutine K_MATRIX_SETUP. TIMEDEL then finds threshold energy ranges using the target state energies returned by K_MATRIX_SETUP. The time-delay calculation then begins with the first energy point in the first threshold. Apart from the initial energy, which is specified by the user, the limits of each threshold-to-threshold grid are chosen as described in Section 2, in order that disruptive behaviour is avoided: precise definitions are given in the accompanying user's manual so that the user may control these limits.

The first set of K-matrices is calculated by passing the first two energy values ($E + dE/2$ and $E - dE/2$) to the subroutine GET_K_MATRICES, which calls GETKMAT twice to return two K-matrices. FINDTIMEDEL takes the K-matrices and passes them to the subroutine KTOSMAT which converts them to S-matrices (see Eq. (4)) using the SOLVE_LINEAR_EQUATIONS routine in timedel_maths_library_calls. Using the S-matrices the subroutine TIMED, contained in FINDTIMEDEL, then finds and diagonalizes the time-delay matrix (see Eq. (1)) using the library module routine DIAGONALIZE_DESCENDING_ORDER. The highest five eigenvalues (or n_o if $n_o \leq 5$, where n_o is the number of open channels) are stored (the user may change the default value from 5). The programme then recalculates the grid-spacing and an associated new value of dE based on the size of the longest time-delay (the first eigenvalue) using an inversely proportional relationship and moves on to the next energy point E (see [19]). This method ensures the number of grid points for each resonance is close to a required (input or default) number [19].

Once all the energy grid points have been found for the given threshold energy range, all of the eigenvalues (5 or n_o per point) are passed to the subroutine EIGSORT. EIGSORT tracks resonances as they move through different eigenvalues by switching every time an avoided crossing is detected (see Fig. 1). Once the eigenvalues are sorted they are passed to DISCONRM which removes discontinuities left by the sorting process by replacing them with a linear interpolant, this reduces the number of false resonance detections.

The eigenvalues are now ready to be fitted and are passed to the fitting subroutine FITTING. By default TIMEDEL considers the first three eigenvalues for fitting; testing showed this to be sufficient. The number of eigenvalues can be increased to five or n_o (if $n_o < 5$) but this should not be necessary. FITTING searches the sorted eigenvalues for maxima and performs a number of checks to ensure that the maxima are not numerical noise or artefacts left by EIGSORT or DISCONRM. The positions of the maxima are then passed to FOUNDRES. FOUNDRES finds fitting limits by looking for a change in sign of the gradient on either side of the maxima[3]; FITLORS then fits the resonance with the form given in Eq. (5) using the maths library module routine MINIMIZE_SUM_OF_SQUARES. The branching ratios are then found at the fitted resonance position using GET_K_MATRICES (an appropriate small dE is set for this) and FINDTIMEDEL. The branching ratio calculation is isolated in routine BRANCHING_RATIOS, contained in FOUNDRES. There are options for calculating a range of values across the resonance, and the user may customize this routine, for example, to produce average values if desired, without affecting the rest of the code. Once the FITTING subroutine has completed, PTIMEDEL moves onto the next threshold. When all thresholds have been calculated PTIMEDEL returns to the 'main' programme.

We note that the work performed once a threshold to threshold grid has been defined and time-delays calculated, is performed in a contained routine in PTIMEDEL called SORT_AND_FIT_TIME_DELAYS. This coding allows the same routine to be called if energy values and pairs of K-matrices, and optionally target energies, are read in rather than calculated using the adaptive grid. This option has been retained from the original TIMEDEL code [19] but now the sorting procedure is also applied. The option of reading in K-matrices is purely serial once the split of MPI_COMM_WORLD across 'geometries' has been made.

3.2. *Parallel structure*

The parallel version of TIMEDELN supplied has a hierarchy of three levels of parallelization, with scope for further low-level parallelization. As already noted, an overall wrapper, controlled in the main programme, allows several calculations to be run as a single job. The main programme is compiled from within a script PTIMEDEL_CREATE.SH which takes in three parallel control parameters: the number of geometries, the number of tasks per geometry and the number of tasks per pair of K-matrices. At runtime, the main programme checks the consistency of the number of MPI tasks with the control parameters. The global communicator is then split into NGEOMS sub-communicators for the separate calculations: files cwrapper.f90 and chdir_c.c allow the code to change to an individual directory, created and with input files in place as part of the script, to run the calculation. This parallelism is present for ease of producing bulk data: we remind the user that load-balancing, while carried out within a calculation, is not checked between calculations.

The second layer of parallelization is designed for optimum load-balancing within a calculation. Whereas the serial code scans successively between thresholds in energy order, the parallel code uses the target state thresholds to assign energy ranges to groups

[2] For general users also using the UKRmol suite, k_adapt requires the rsolve namelist RSLVIN to run, which should be placed after the TIMEDELN namelist TIME in the input file 'time_del.inp'. We note that the k_adapt routine SETUPKMAT reads in the target and channel dataset (by default fort.10) and the R-matrix poles, amplitudes and the multipole expansion of asymptotic potentials (by default fort.21). It also sets up the energy-independent parts of the scattering calculations to be performed by GETKMAT.

[3] If the sorted eigenvalues still contain overlapping resonances, as discussed in Section 2 and also in Section 4.3, the code can, if desired, revert to the 'sum of Lorentzians' procedure of [19] using the parameter **nes** (see Section 4.1) to determine the limits.

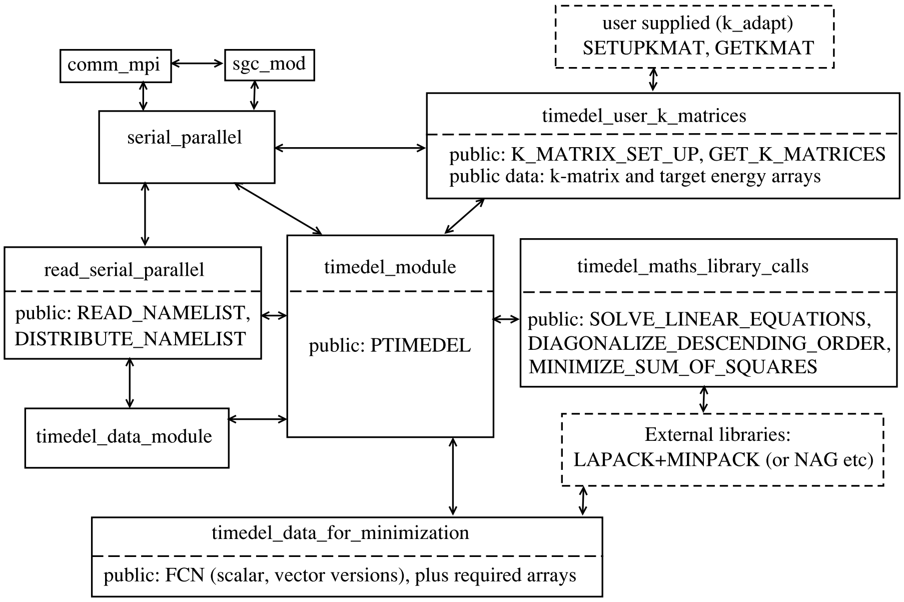
user supplied (k_adapt)
SETUPKMAT, GETKMAT
comm_mpi
sgc_mod
serial_parallel
timedel_user_k_matrices
public: K_MATRIX_SET_UP, GET_K_MATRICES
public data: k-matrix and target energy arrays
read_serial_parallel
public: READ_NAMELIST, DISTRIBUTE_NAMELIST
timedel_module
public: PTIMEDEL
timedel_maths_library_calls
public: SOLVE_LINEAR_EQUATIONS, DIAGONALIZE_DESCENDING_ORDER, MINIMIZE_SUM_OF_SQUARES
timedel_data_module
External libraries:
LAPACK+MINPACK (or NAG etc)
timedel_data_for_minimization
public: FCN (scalar, vector versions), plus required arrays

Fig. 2. The module structure of TIMEDEL. Details of the parallel modules are given in the programme user's manual. Links to minor modules and routines (such as module precisn and module serial_parallel routine STOP_RUN) are omitted for space and clarity. Dashed line boxes represent external routines.

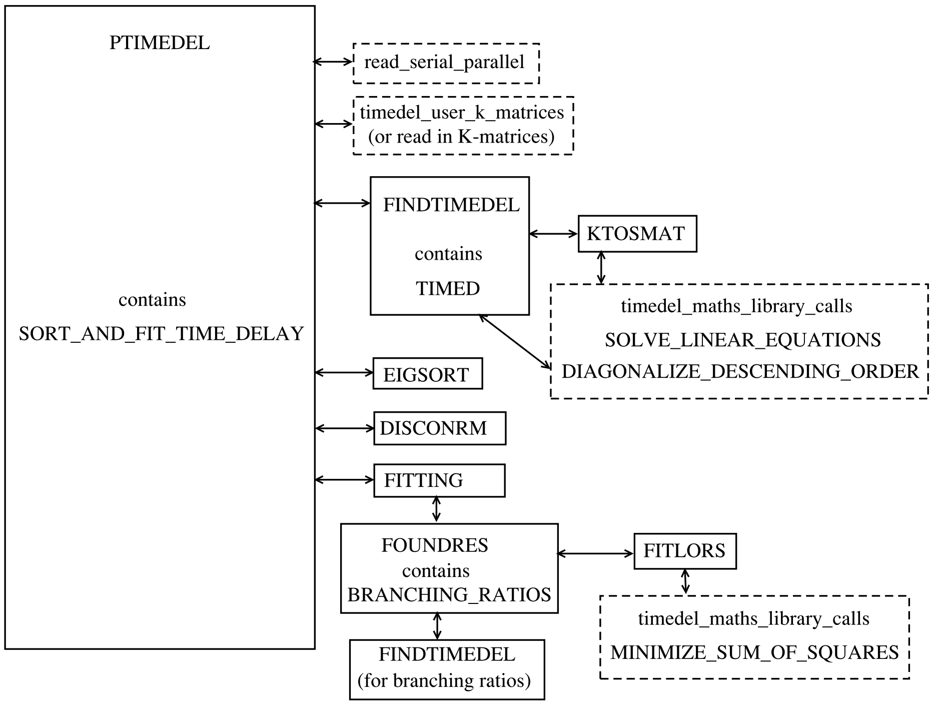

Fig. 3. The 'physics code' call structure for routine PTIMEDEL. Dashed line boxes represent routines outside module timedel_module. The calls to module serial_parallel are omitted for space and clarity, and are described in the text.

of tasks (or single tasks if the third parallelization layer is not used). The resonance finding and fitting within one range (between two thresholds) is independent of that in the other ranges. The output file names given in the next section are chosen so that a set of files is produced for each task group. Once a task group has finished its current energy range, it moves on to the next available range from the Fortran do loop over energy ranges. The code provides two options for this, either a simple round-robin in which each task group leader works through loop iterations in order, or preferably a scalable global counter (file sgc_mod.f90, written in Fortran 2003 and based on an example from the MPICH package examples [29]) in which each task group leader is assigned the next available iteration by inspecting and updating the global counter on an MPI-IO data window. This scalable global counter approach means that short or smooth energy ranges with few adaptive grid points may be passed through rapidly while another task group has been assigned a more intensive range, for example the initial range from low energy to the first threshold, or a range with detailed resonance structure and a dense grid. Thus the full calculation should only take as long as the most time-consuming energy range plus a small amount of machine-dependent parallel overhead. The round-robin approach has minimal overhead but has the disadvantage that two (or more) time-consuming ranges may be assigned to the same task group.

The third layer of parallelization involves splitting the calculation communicator, called GEOM_COMM, into several sub-communicators labelled KMAT_COMM whose group leaders form a communicator SGC_COMM (this splitting procedure also produces other communicators equivalent to SGC_COMM between corresponding 'other ranks' in KMAT_COMM, but these are not used in the current scheme). The SGC_COMM tasks are assigned energy ranges and use the counter, and then broadcast the information to their KMAT_COMM members. The KMAT_COMM communicator may then be used to form the user-supplied K-matrices. The K-matrix generation is assumed to involve the most effort as it effectively involves performing a scattering calculation for each of the pair of K-matrices at that energy point (with the UKRMol R-matrix method used by the authors and collaborators, 'outer region' calculations are required here [20]). In the module timedel_user_k_matrices which contains the call to the user-supplied routine GETKMAT, KMAT_COMM may be used to generate the K-matrix pair in parallel (replacing MPI_COMM_WORLD as the top-level communicator in the user-supplied routines) with a final broadcast of the pair of K-matrices to all members of KMAT_COMM. The code supplied contains a simple example in which the first two tasks in KMAT_COMM perform serial calculations for $E + dE$ and $E - dE$ respectively and broadcast the results within KMAT_COMM, halving the serial run-time for this operation.

The parallelism has been deliberately abstracted away from PTIMEDEL so that firstly, the same PTIMEDEL is used for serial and parallel calculations, and secondly, the parallel structures can be applied more generally and beyond the current resonance fitting application. The module serial_parallel controls the particular communicator splitting and provides the interface to the abstracted parallel routines required for TIMEDELN. The module also handles parallel I/O. The input namelists are read in by the group leader of GEOM_COMM and cascaded to SGC_COMM tasks and then to their KMAT_COMM members. Separate output files are produced from each SGC_COMM task, with a stub plus numerical label naming convention. We intend to publish expanded versions of the module files comm_mpi.f90 and sgc_mod.f90 separately as they are also in use in other applications (see, for example [30]), so a brief description of their properties as required for TIMEDELN is given in the programme user's manual.

We note that the three-way parallel functionality may be extended by additional splitting of the KMAT_COMM communicator by the user, or any of the communicators may be set to have one MPI task for simplified calculations. The user-supplied routine may use, for example, OpenMP [31] threads for each GEOM_COMM task rather than, or in addition to, a multiple-task KMAT_COMM MPI solution. The user may also wish to introduce appropriate parallelism for heterogeneous accelerator architectures within the K-matrix calculations (see, for example, [32]).

The general case for which pre-calculated pairs of K-matrices are read in has been left as serial code, as the actual resonance finding and fitting is fast using the library routines (which could themselves be linked to OpenMP parallel versions). However, for the test run we have included an option whereby a set of pre-prepared K-matrices are read-in during a parallel run rather than calculated. This parallel test run will only work for a specific resonance calculation as the energy grid spacing needs to match that derived from the adaptive grid, however the user may test various parallelization strategies (replicating data in separate directories for the top level): in each directory the K-matrices are contained in a single file and are selected by the MPI tasks as required. This option may also be used to refine parameters controlling the sorting and fitting procedures for a given grid without having to recalculate K-matrices.

4. The code package

The code package contains the Fortran files, makefiles and a manual on how to use the code, as well as data and output for serial and parallel test runs.

The main TIMEDELN code comes in three Fortran files found in directories libouter and libouter_serial (along with example Makefiles): serial_parallel_p.f90, serial_parallel_s.f90 and ptimedel.f90, plus the two scripts ptimedel_create.sh and stimedel_create.sh in the top-level directory which contain the 'main programme' and linking commands. The file precisn.f90 is needed in libouter for parallel set-up: this simple module is included in serial_parallel_s.f90. The additional files for parallel runs are described in the user's manual. All the files compile with Fortran 95 compliant compilers except the scalable global counter file sgc_mod.f90 which requires Fortran 2003, however we have not had problems compiling this file on the compilers noted in the test run section. If necessary, the file can be omitted, the limited number of calls to module sgc in serial_parallel_p.f90 may be commented and the round-robin technique (**sgc_choice** false, see Section 4.1) used.

The sample Makefiles have various options. Three example Makefiles, (for Intel, Cray and GNU compilers respectively) are included for both libouter and the parallel linking compilation, and assume a consistent compilation throughout with 'standard' definitions of integer type, assumed as supplied to be 4-byte (Fortran integer*4), to be compatible with the MPI routines in comm_mpi.f90 and sgc_mod.f90: the parallel machines the code has been tested on all assume that type 'MPI_Integer' is equivalent to Fortran integer*4. An additional sample Intel Makefile allows the use of 8-byte integers outside MPI calls, requested by the UKRmol community.

We note that adapting the example Cray and GNU Makefiles (for local systems or other compilers) is usually straightforward as the basic flags and links needed are simple to adjust. Intel Makefiles, particularly links to the correct system and mathematical libraries, tend to be much more complicated unless local system engineers have built in automatic links. The illustrations in the user's manual use the example Intel Makefiles to help 'scratch-builds' of the package.

4.1. Input data

The input required for basic use of TIMEDELN is fairly brief and is contained in the namelist TIME, read by the programme from file 'time_del.inp'. This namelist also contains a fairly large number of other parameters which the more experienced user may use to change default settings from run to run. The user's manual provides a detailed description of each of the input parameters which supersedes that given for the original TIMEDEL [19], as well as other coded parameters which can be altered before compilation. In the manual and this article, the namelist and other parameters are written in bold typeface.

4.2. Parallel framework development and indicative benchmarks

The parallelization of TIMEDELN was partly developed, using an early test version of the electron $-N_2^+$ data previously referenced, as part of a PRACE optimization and development project [33] with an aim of producing a code that could also be practically applied to resonance studies of biological molecules such as DNA and RNA bases and introduced the round robin approach. Benchmark results [34] showed that the task farm allowed 1024 e-N_2^+ geometries to run at a time 1.64 relative to 16 geometries (the ideal would be 1, but load-imbalance increases as internuclear separation varies). For a single geometry, 4 tasks using the round robin approach gave a speed-up of 3.3 compared to 1 task [34]. The full abstracted parallel framework was developed subsequent to this proof of concept work incorporating the already available comm_mpi and sgc_mod modules.

A benchmark test for a truncated electron $-N_2^+$ test limited to 4 threshold to threshold groups, treated here as an abstract example, gave the following results using the Intel compiler on the UK system ARCHER [35]. The thresholds of the benchmark N_2^+ 'target' states are at 0.89 eV, 3.6 eV, 5.5 eV with initial and final energies 0.05 eV and 6 eV respectively, thus we have 1 large range, 1 medium range and 2 short ranges. However the actual time taken depends on the number of resonances and the adaptive grid. and in fact the low energy region, with relatively high time-delay values, dominates: with two (or three with two active) tasks in KMAT_COMM the run took 29–30 min. With more than one active task in SGC_COM, the runs took a total of 19–20 min, with the lowest energy range handled by one KMAT_COMM and the other ranges bundled together according to the number of tasks in SGC_COMM, with individual times around $\sim$11 min (two active tasks in SGC_COMM), and $\sim$6 and $\sim$7 min (three active tasks in SGC_COMM). The apparent overhead includes some genuine overhead due to writing to standard output and remaining inefficiency with MPI-IO, but also shows some disparity with some very narrow near-threshold resonances detected in some runs but not in others according to precise values of the grid positions (i.e. for scientific purposes close to threshold a finer study would be needed). We note that the overall timings for runs with a single task in KMAT_COMM took $\sim$56 min and $\sim$36 min respectively (K-matrix energy-independent set-up time was negligible).

Additional benchmarking results for electron collisions with both N_2^+ and pyrimidine may be found in [32].

4.3. Test run

The test run provided, in the form of pre-calculated K-matrices, is of low-energy electron pyrimidine collision (irreducible representation A1) resonances. The original scientific calculation is described in detail in [17]: the authors of this paper provided R-matrix method data allowing us to generate adaptive grid time delays, and the scattering case is now also part of the UKRmol test suite [20]. The data is for a single geometry, however the test case can be set up for multiple identical geometries, thus with artificially good load balancing at this level of parallelism. A full description of the test run is given in the user's manual. Data is supplied for both a serial and a full parallel test with pre-calculated K-matrices provided to reproduce a previous adaptive grid calculation (namelist parameter **adapt** set true), and also for a separate non-adaptive run with pairs of K-matrices read in (**adapt** set false) as in [19]. Here we provide a shorter description of the parallel test: some of the code parameters described in the user manual are mentioned (and assumed to be known) here: for example the hard-coded logical parameter **test_para** in module timedel_user_k_matrices must be set true for the pre-calculated 'adaptive' test runs and set false for new adaptive grid calculations.

The case, with incident electron energy varying from 0.01 to 7 eV, involves 11 threshold to threshold energy grid groups, two of which are large: the first large group is from 0.01 eV to just below the first excited threshold at 4.54 eV, the second large group is from just above 5.63 eV to just below 6.46 eV and contains a double resonance. The other threshold groups have no resonances and can also be quite narrow with target energy levels close together (but not degenerate). Thus if **sgc_0_sleeper**[4] is not required to be true, the SGC_COMM communicator should give good results with 2 task groups (one large grid for each task group, smaller grids shared) and ideally fastest time to solution results with 3 (and above, up to 11 task groups) though the third and additional task groups will spend time idle. In the actual tests, **sgc_0_sleeper** set true was needed, so that the minimum number of task groups needed for parallel work is 3, with fastest time to solution for 4 task groups in SGC_COMM, though we recall that the 'sleeper' task group consists of a single task holding the MPI-IO window. Since the energy grid containing the resonances is the 9th energy grid, the round robin approach will give bad results for even numbers of task groups and good results for odd numbers. All benchmark results for the generation of the data followed this pattern (details are available from the authors, see also [32]). The simple 2-active-tasks parallel generation of K-matrices halved the execution time as hoped for (factor 0.5–0.55 for various communicator groupings on an ARCHER node). A fully parallelized 'k_adapt' module with KMAT_COMM having (for example) 64–128 tasks (or more) would significantly reduce overall execution time from the $\sim$51–55 min (dominated by the 0.01 to 4.54 eV sub-range) taken using the two-way parallelization.

To avoid over-large data-files, the test run data uses a relatively large value of **gridmin**, 0.005. Fig. 4 shows time-delay results over the first energy range in three forms: the actual time-delays, the sorted values after EIGSORT and the further modifications after DISCONRM. This illustrates the changing of eigenvalue labels and further partial linearization, deliberately more crudely than the detailed 'final' results presented in Fig. 1. The overall effect on this energy sub-grid is to stop the programme from mistaking unphysical behaviour for resonances: the artificial distortions that appear after sorting are deliberate (note also that in practice the default **noise_barrier** will stop false resonance fitting). Fig. 5 presents similar results for the sub-grid with the two resonances. Here it may be seen that the EIGSORT routine separates the resonances as expected. We present these results to emphasize that the user should study these outputs carefully in new calculations, in order to interpret what may be complex patterns of resonances.

[4] **sgc_0_sleeper** is included in the namelist to activate additional coding which overcomes deficiencies in certain implementations of MPI-IO: if activated the single MPI task associated with the scalable global counter window plays no other direct part in the calculation to avoid unwanted blocking of other tasks: details are in the manual.

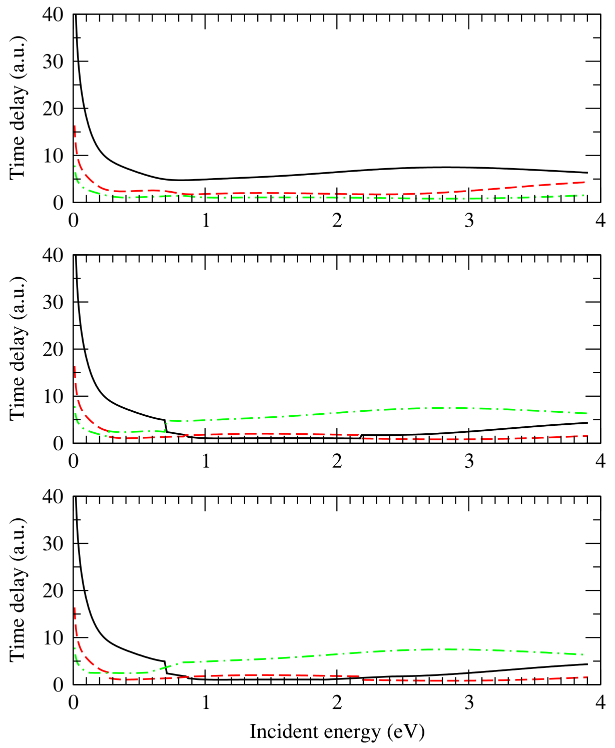

Fig. 4. Time-delay output for the pyrimidine test run for the first energy sub-grid, as described in the text. The top graph shows the first three calculated eigenvalues, the middle graph shows the first three 'mixed' eigenvalues after the routine EIGSORT and the bottom graph shows the slightly modified values following DISCONRM. Note that in EIGSORT the first 5 time-delay eigenvalues are sorted to produce the 'mixed' values.

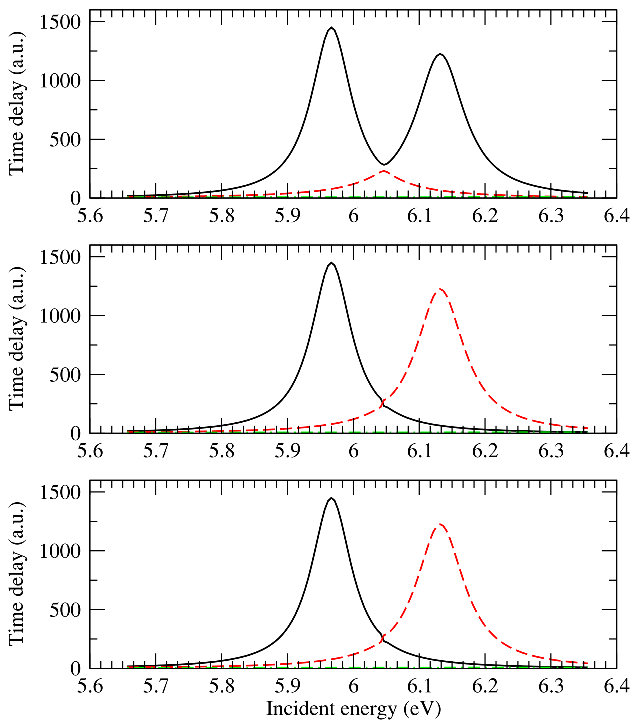

Fig. 5. Time-delay output for the pyrimidine test run for the energy sub-grid containng resonances, as described in the text. The top graph shows the first three calulated eigenvalues, the middle graph shows the first three 'mixed' eigenvalues after the routine EIGSORT, with the resonances separated, and the bottom graph shows the values following DISCONRM. Note that in EIGSORT, the first 5 time-delay eigenvalues are sorted to produce the 'mixed' values.

The actual numbers for fitted resonance position, width and background are given in the output files in the Test_runs directory and are reproduced below. The resonance output appears in the main output files (these are named and generated in PTIMEDEL as time_del.out.{00, 01, . . .}) as:

```
 Considering energies between threshold energies:
   5.63162389115563        6.45526394557738
 Energy range considered=
   5.65662389115563      ->   6.36078170823800
 Fitting resonance set with e-value label:                     1
 In this fitting, call       0  noisy local maxima were ignored
                     1 maxima were found in fitting call

Attempting to fit resonance set  1 as 1 resonance(s)

 Fitting limits:   5.65662389115563      ->   6.35662389115567
                   Position (eV )       Width (eV )
Resonance: 1. 1    0.5966776558743D+01  0.7474072108737D-01
Fitted with background=-.5300D+01 and St. Dev.=0.2057D-03
Effective P.Q.N.= 0.5277517150085D+01
 Peak branching ratios:  4.103020919284603E-005  1.681294746867281E-004
```

with further values for the peak branching ratios (here there are 84 open channels), and

```
 Fitting resonance set with e-value label:                     2
 In this fitting, call                     0  noisy local maxima were ignored
                     1 maxima were found in fitting call

Attempting to fit resonance set  1 as 1 resonance(s)

 Fitting limits:   5.65662389115563      ->   6.35662389115567
                   Position (eV )       Width (eV )
Resonance: 1. 1    0.6131940164156D+01  0.8855301679824D-01
Fitted with background=-.3336D+01 and St. Dev.=0.6364D-03
Effective P.Q.N.= 0.6486904521421D+01
 Peak branching ratios:  1.746261984742436E-003  8.295745778172306E-003
```

The 'effective principal quantum number' value relates the resonance position to the next threshold (threshold energy − (Effective Q.N.)$^{-2}$ in Rydbergs: see the definition of input variable **maxn**).

The labelling shows details of the reordered 'eigenvalue' label, the number of groups of resonances for that value and the number of resonances associated with that group. Ideally there should only be 1 resonance per group, but with closely spaced resonances within each other's width there may be several, in which case the Lorentzian fit is applied to the group unless **force_single** is invoked (fitting limits should be checked here). Also, if the background structure has a broad maximum associated with it, this may be considered as a resonance: the user should examine the calculated time-delays carefully and consider if this is physical. Running identical calculations with or without **force_single** is straightforward with the **test_para** parameter.

The resonance positions and widths are written separately in Hartrees to a file with stub name 'resonances' (see the definition of **writbr**). If **writbr** is set true, then additional files are produced for the peak branching ratios, all partial widths and the first 3 partial widths. The format for these files (which require **adapt** to be true) is set in FOUNDRES as follows:

```
write(iw_res,'(2e22.13)') < position, width> ! in Hartrees
if (writbr) then
write(iw_br,'(30e22.13)') br(1:nopen)
                 ! in Hartrees, adjust format if nopen > 30
write(iw_xbr,'(32e22.13)') < position, width>  &
       & (br(l) * < width > ), l = 1, nopen
                 ! in Hartrees, adjust format if nopen > 30
write(iw_gwid,'(5e22.13)') < position, width>  &
       & (br(l) * < width > ), l = 1, 3      ! in Hartrees
```

In addition, if **writbr** is true and a parameter **num_b_ratios** is set greater than the default value 1, then branching ratios across the resonance are calculated and written to files with stub name 'resonance_range_branching_ratios' (details are in the manual). Note that for this test run **writbr** is set false as supplied. The K-matrix data needed to calculate the peak branching ratios is included in fort.20 (parallel test). To run the test runs, follow the instructions in the programme manual.

For the parallel test, the user should compare timings to see if **sgc_0_sleeper** is required (if it is required, setting it to false will effectively serialize the calculation).

The output files should be compared with the supplied files and reasonable agreement should be obtained. The resonance data, and branching ratio data in the adaptive grid test, are written to the main output files and to separate files. In the parallel case, note that each SGC_COMM task writes a separate file, so that sets of output files are produced which together contain all the data in the supplied output files.

5. Conclusion

We have implemented the fitting of additional eigenvalues in the resonance fitting process of the original TIMEDEL programme [19] as well as parallelizing the code. The three-level MPI framework includes a scalable global counter to load-balance adaptive grid calculations and a simple communicator presented to the user for the user-supplied K-matrix calculations. The result is a method of detecting and fitting resonances which can deal with complex overlapping resonances, such as those prevalent in ionic targets. A disadvantage of the time-delay method when compared to fitting the eigenphase sum is inability to scan an energy range to locate the position of resonances. Enough K-matrices need to be calculated to locate the resonances by direct examination of the Q matrices. The addition of parallelization alleviates this issue by significantly speeding up the calculation so that an initial scan need not be necessary; the time-delay can be speedily calculated at a large number of energy points and the resonances then located and fitted. The top-level parallelism allows simultaneous bulk generation of data for 'similar but different' calculations such as varying molecular geometries. The parallel framework has been abstracted from the physics code with simple interfacing routines and may be used for other purposes: the same physics code is used for both parallel and serial compilations. Calls to external libraries have also been abstracted to separate modules so that these may be edited by the user without disturbing the resonance-finding code.

Acknowledgements

We thank Themisys Limited for supporting a studentship for DAL and CCPQ (UK EPSRC Collaborative Computational Project Q) for supporting the collaboration between UCL and Daresbury via the UK EPSRC Service Level Agreement for CCP core support. Parts of the computational work were funded by PRACE as described in the text, we also acknowledge some time on the UK STFC Hartree Centre iDataplex machine (the STFC Hartree Centre is a research collaboratory in association with IBM providing High Performance Computing platforms funded by the UK's investment in e-Infrastructure. The Centre aims to develop and demonstrate next generation software, optimized to take advantage of the move towards exa-scale computing). We acknowledge time on ARCHER (the ARCHER UK National Supercomputing Service) for later and current development. We thank Z Masin and JD Gorfinkiel for providing us with data to generate the pyrimidine test case, and the referees for useful comments.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cpc.2017.01.005.

References

[1] T. Sochi, P.J. Storey, Euro. Phys. J. Plus 128 (2013) 82. http://dx.doi.org/10.1140/epjp/i2013-13082-4.
[2] A.U. Hazi, Phys. Rev. A 19 (1979) 920–922.
[3] J. Tennyson, C.J. Noble, Comput. Phys. Comm. 33 (1984) 421–424.
[4] K. Bartschat, P.G. Burke, Comput. Phys. Comm. 41 (1986) 75–84.
[5] C.J. Noble, M. Dorr, P. Burke, J. Phys. B: At. Mol. Opt. Phys. 26 (1993) 2983–3000.
[6] L. Quigley, K. Berrington, J. Pelan, Comput. Phys. Comm. 114 (1998) 225–235.
[7] D.A. Little, J. Tennyson, J. Phys. B: At. Mol. Opt. Phys. 47 (2014) 105204.
[8] F.T. Smith, Phys. Rev. 118 (1960) 349–356. http://dx.doi.org/10.1103/PhysRev.118.349.
[9] I. Shimamura, J.F. McCann, A. Igarashi, J. Phys. B: At. Mol. Opt. Phys. 39 (2006) 1847–1854. http://dx.doi.org/10.1088/0953-4075/39/8/003.
[10] K. Aiba, A. Igarashi, I. Shimamura, J. Phys. B: At. Mol. Opt. Phys. 40 (2007) F9–F17. http://dx.doi.org/10.1088/0953-4075/40/2/F01.
[11] I. Shimamura, J. Phys. B: At. Mol. Opt. Phys. 44 (2011) 201002. http://dx.doi.org/10.1088/0953-4075/44/20/201002.
[12] I. Shimamura, in: C.A. Nicolaides, E. Brändas, J.R. Sabin (Eds.), Advances in Quantum Chemistry, in: Advances in Quantum Chemistry, Vol. 63, Academic Press, 2012, pp. 165–245. http://dx.doi.org/10.1016/B978-0-12-397009-1.00004-7.
[13] D.T. Stibbe, J. Tennyson, J. Phys. B: At. Mol. Opt. Phys. 29 (1996) 4267–4283.
[14] D.T. Stibbe, J. Tennyson, J. Phys. B: At. Mol. Opt. Phys. 31 (1998) 815–844.
[15] I. Rabadán, J. Tennyson, L.A. Morgan, Chem. Phys. Lett. 285 (1998) 105–113.
[16] I. Baccarelli, F. Sebastianelli, F.A. Gianturco, N. Sanna, Eur. Phys. J. D 51 (2009) 131–136. http://dx.doi.org/10.1140/epjd/e2008-00104-5.
[17] Z. Masin, J.D. Gorfinkiel, J. Chem. Phys. 137 (2012) 204312. http://dx.doi.org/10.1063/1.4767345.
[18] L.A. Morgan, J. Tennyson, C.J. Gillan, Comput. Phys. Comm. 114 (1998) 120–128.
[19] D.T. Stibbe, J. Tennyson, Comput. Phys. Comm. 114 (1998) 236–242.
[20] J.M. Carr, P.G. Galiatsatos, J.D. Gorfinkiel, A.G. Harvey, M.A. Lysaght, D. Madden, Z. Masin, M. Plummer, J. Tennyson, H.N. Varambhia, Eur. Phys. J. D 66 (2012) 58.
[21] J. Tennyson, D.B. Brown, J.J. Munro, I. Rozum, H.N. Varambhia, N. Vinci, J. Phys. Conf. Ser. 86 (2007) 012001.
[22] A. Dora, J. Tennyson, K. Chakrabarti, Euro. J. Phys. D. 70 (2016) 197.
[23] D.A. Little, K. Chakrabarti, J.Z. Mezei, I.F. Schneider, J. Tennyson, Phys. Rev. A 90 (5) (2014) 052705. http://dx.doi.org/10.1103/PhysRevA.90.052705.
[24] The Message Passing Interface: standards for MPI are available from the MPI Forum, 2016. URL: http://www.mpi-forum.org.
[25] J. Tennyson, D.A. Little, in: I.F.S.O. Dulieu, J.R. (Eds.), Dissociative Recombination: Theory, Experiments and Applications, in: EPJ Web of Conferences, Vol. 84, 2015, p. 03002.
[26] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK Users' Guide, third ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999.
[27] LMDIF1 and dependencies: MINPACK Fortran numerical library, University of Chicago, Argonne National Laboratory, USA, 1999. URL http://www.netlib.org/minpack/.
[28] Nag Fortran Library Mark 25, Numerical Algorithms Group, Oxford, UK, 2015. URL http://www.nag.co.uk/numeric/fl/FLdescription.asp.
[29] fetch_and_add_tree.c, from the MPICH trac site, Argonne National Laboratory, 2001. URL http://trac.mpich.org/projects/mpich/browser/test/mpi/rma/fetchandadd_tree.c.
[30] C.J. Noble, A.G. Sunderland, M. Plummer, NAG HECToR dCSE report, Combined-Multicore Parallelism for the UK electron-atom scattering Inner Region R-matrix codes on HECToR, 2012. URL http://www.hector.ac.uk/cse/distributedcse/reports/prmat2/.
[31] The OpenMP specification for parallel programming: standards for OpenMP are available from OpenMP.org, 2016. URL http://www.openmp.org/wp/.
[32] M. Plummer, A.G. Sunderland, D.A. Little, J. Tennyson, C.J. Noble, Developing MPI structures for a specialized physics problem which have further general use: how to balance flexibility, efficiency and clarity, to be published, preprint available, 2016.
[33] PRACE application scalability programme (life science applications), Partnership for Advanced Computing in Europe, 2014. URL http://www.prace-ri.eu/application-scalability.
[34] A.G. Sunderland, M. Plummer, PRACE White Paper, HPC programming to generate electron-molecule resonance data for DNA radiation damage studies, Partnership for Advanced Computing in Europe, 2014. URL http://www.prace-ri.eu/IMG/pdf/wp174.pdf.
[35] ARCHER: the UK National Supercomputing Service, 2013–2018. URL http://www.archer.ac.uk.