

# reskit: a toolkit to determine the poles of an S-matrix

Journal Item

# `reskit`: a toolkit to determine the poles of an S-matrix

Peter S. Bingham[a], Jimena D. Gorfinkiel[a,*]

[a]*School of Physical Sciences, The Open University, MK67AA Walton Hall, Milton Keynes, United Kingdom*

**Abstract**

We present the python package `reskit`, developed to calculate the coefficients of an analytical continuation of the S-matrix of a physical system by means of Padé approximants and, from this fit, determine the complex poles of this S-matrix.

The current implementation of the program is restricted to elastic scattering, i.e. cases in which all channels have the same energy. It has been tested using as input *ab initio* scattering data for electron-molecule collisions obtained for energies along the real axis. The identification and characterisation of the resonances present in the systems using reskit is described and discussed.

*Keywords:* Padé approximants; S-matrix; resonances;

**PROGRAM SUMMARY**

*Program Title:* `reskit`

*Licensing provisions(please choose one):* MIT

*Corresponding author.
*E-mail address:* J.Gorfinkiel@open.ac.uk

*Programming language:* Python 2.7

*Nature of problem(approx. 50-250 words):* the S-matrix of a system is usually calculated as a function of real energy (for example by varying the kinetic energy of a projectile in a scattering process). Locating its complex poles requires some type of extrapolation on the Riemann sheets relating the total energy, in the complex plane, to the momentum. The program determines the complex poles of the fitted polynomial S-matrix that identify its resonant, bound and virtual states.

*Solution method(approx. 50-250 words):* the program is based on the method developed by Rakityansky and collaborators [1] in which a Padé approximant is used to fit provided S-matrix data, determined as a function of a real energy. Expressing the S-matrix in the Jost form, the generally complex poles of the S-matrix can be found by determining the stability of the roots of the determinant of the Jost matrix as S-matrix data for an increasing number of energies are used for the fit.

*Additional comments including Restrictions and Unusual features (approx. 50-250 words)*: the current implementation requires that all channels/states have the same energy and that no long-range Coulomb interaction potential is present. The first of these restrictions results in a simplified S-matrix that is defined solely as a function of the momentum in the complex plane.

[1] P. Ogunbade, S. Rakityansky, S-matrix parametrization as a way of locating quantum resonances and bound states: multichannel case, in: Proceedings of the 2nd South Africa - JINR SYMPOSIUM, Models and Methods in Few- and Many-Body Systems, JINR, 2010, 52–61.

## 1. Introduction

Resonances are an important phenomenon in atomic and molecular physics: for example, their presence results in the enhancement of processes like collision-induced electronic excitation. They affect both photodetachment and photoionization and are crucial in cold collisions. In addition, some physical processes are initiated by the formation of a resonance, an example of which is dissociative electron attachment (DEA). In DEA, an electron temporarily attaches itself to a molecule forming a resonance and, as a result, one or several bonds in the molecule are broken [1]. Therefore, the identification and characterisation of resonances is an important task necessary for the understanding and modelling of many atomic and molecular phenomena.

Several techniques and programs have been developed to identify and characterise resonances using, as input, scattering data (i.e. K-matrices, S-matrices and derived quantities). Some [2, 3] are based on fitting the eigenphase sum or phase shift to a Breit-Wigner profile [4, 5]. However, this approach fails under certain circumstances; for example, when two or more resonances overlap, when a resonance is very close to a channel threshold or when the resonance is very wide. In the latter case the problem is that it is very difficult to separate the resonant and the non-resonant (background) contribution to the eigenphase sum. There have been attempts to address the shortcomings of using the Breit-Wigner formula through the development of more elaborate approximations [6], although the authors are not aware of any publicly available computational implementations of the improved approaches. An alternative technique for identifying and characterising resonances is based on the analysis of the time-delay [7]. Resonances

can be located by analysing the eigenvalues of the time-delay matrix. This technique is particularly effective for overlapping resonances. Computational implementations of the technique are available [8, 9]. We note that a number of approaches, not based on scattering methods, have been developed for the identification of resonances. These are based on conventional bound state methodologies, stabilization techniques, complex absorbing potentials, etc.. One of them is based on the analytical continuation of the electron affinities of the target molecules in the presence of a perturbation potential [10, 11]. In this method, known as the regularized method of analytic continuation, conventional quantum chemistry calculations are employed to determine the real energies of the system in the presence of a perturbation. The results are then analytically continued into the complex energy plane.

Rakinyansky *et al.* [12] have proposed an approach that takes advantage of the fact that resonances are associated with poles of the S-matrix [4, 5] and that the S-matrix in Jost form can be expressed as the product of a matrix and the inverse of a related matrix. Analytical continuation is achieved by expressing this Jost form using Padé approximants. By fitting to S-matrix data for sufficient discrete values along the real energy axis the coefficients of the Padé expansions can be found. The roots of the polynomial expansion of the determinant of the matrix that is inverted will correspond to the poles of the rational S-matrix. Since bound and virtual states are also associated to these poles (that are generally located on different sheets of the Riemann surface), the method actually enables the identification of all these types of states of a system. In addition, the rational S-matrices can be used to determine quantities like the cross section, time-delay, etc.

In this paper, we present a python toolkit, `reskit`, based on the approach of Rakinyansky *et al.*. The toolkit provides the routines necessary to perform the S-matrix fit and the pole determination for cases in which all the channels involved in the process have the same energy, i.e. for an elastic process. Rakitiansky and collaborators proposed approaches to both elastic [13] and inelastic [12] cases. In our software, we have implemented the more general inelastic case although, at the moment, the implementation only works for elastic cases for which the complexities of dealing with Riemann sheets are avoided. An implementation for the inelastic cases is expected to follow.

## 2. Method

We begin our discussion of the theory presented in [12] by considering an $N$ channel scattering system involving both elastic and inelastic channels. For the purpose of discussion and because the tests presented and analysed in Section 5 correspond to electron-molecule scattering we will talk in terms of an electron as the incident particle. The technique (and the derived software) can be applied to any non-relativistic scattering system, provided that sufficient S-matrix data has been obtained.

A channel refers to a discrete state of the scattering system at asymptote and, in electron-molecule scattering, it is normally defined by the angular momentum of the scattering electron and the internal state of the target molecule. A channel, labelled $n$, is considered open if it is energetically accessible at asymptote, i.e. if the total energy of the system, $E$, is greater than or equal to its energy $E_n^{th}$.

We note that it is customary in electron scattering studies to take the

lowest (ground) internal state of the target to correspond to the zero of energy. In this case $E_n^{th}$=0 for all channels associated to this internal state and the $E_n^{th}$ for all other channels become threshold energies.

*2.1. Parametrised S and Jost matrices*

The multi-channel S-matrix can be written in terms of matrices of the *out* and *in* Jost matrices as the product:

$$\mathbf{S}(E) = \mathbf{F}^{out}(E)[\mathbf{F}^{in}(E)]^{-1} \tag{1}$$

For an $N$-channel scattering problem the Jost matrices can be expressed in the following form:

$$F_{mn}^{out/in}(E) = \frac{1}{2}\left[A_{mn}(E) \pm iB_{mn}(E)\right] \tag{2}$$

where $m$ and $n$ are the channel indices over the range $1, 2, ..., N$. The quantities $A_{mn}$ and $B_{mn}$ can be regarded as the coefficients when the wavefunction at asymptote is expanded in terms of the Riccati-Bessel and Riccati-Neumann functions.

We now apply the technique presented in [12] to obtain parametrised forms of (1) and (2). Considering the channel momenta:

$$k_n = \pm\sqrt{E - E_n^{th}} \tag{3}$$

and channel angular momenta $l_n$, Ogunbade and Rakityansky show [12] that when no Coulomb (long range) interaction is present between the particles [14] the coefficients in (2) can be written in the factorised form:

$$A_{mn} = \frac{k_n^{l_n+1}}{k_m^{l_m+1}} \tilde{A}_{mn}, \qquad B_{mn} = k_m^{l_m} k_n^{l_n+1} \tilde{B}_{mn} \tag{4}$$

We expand $\tilde{A}_{mn}$ and $\tilde{B}_{mn}$ as a power series:

$$\tilde{\mathbf{A}}(E) = \sum_{\mu=0}^{M} \boldsymbol{\alpha}^{(\mu)} E^{\mu} \qquad \text{and} \qquad \tilde{\mathbf{B}}(E) = \sum_{\mu=0}^{M} \boldsymbol{\beta}^{(\mu)} E^{\mu} \tag{5}$$

where $\boldsymbol{\alpha}^{(\mu)}$ and $\boldsymbol{\beta}^{(\mu)}$ are matrices of coefficients. Using (4) and (5) and inserting them into (2) will give the form of the Jost matrices in terms of the coefficients $\boldsymbol{\alpha}^{(\mu)}$ and $\boldsymbol{\beta}^{(\mu)}$. These can then be inserted into (1) which will then, after some algebra and enforcement of the zero energy behaviour [4] $S_{mn}(E) \xrightarrow[k_m \to 0]{} \delta_{mn} + O(k_m^q)$, yield the following linear system:

$$\sum_{\mu=1}^{M} \left[ \frac{k_n^{l_n+1}}{k_m^{l_m+1}} \left( S_{mm}(E_i) - 1 \right) \alpha_{mn}^{(\mu)} - i k_m^{l_m} k_n^{l_n+1} \left( S_{mm}(E_i) + 1 \right) \beta_{mn}^{(\mu)} \right.$$

$$\left. + \sum_{\substack{j=1 \\ j \neq m}}^{N} S_{mj}(E_i) \left( \frac{k_n^{l_n+1}}{k_j^{l_j+1}} \alpha_{jn}^{(\mu)} - i k_j^{l_j} k_n^{l_n+1} \beta_{jn}^{(\mu)} \right) E_i^{\mu} \right] = \delta_{mn} - S_{mn}(E_i) \tag{6}$$

where $S_{mj}(E_i)$ are the elements of an S-matrix obtained by some computational method for a discrete set of energies $\{E_i\}$.

$M$ as defined in equation (5) is the maximum power of the two expansions, for which there will be a total of $2(M+1)$ unknown coefficients. However, since the zeroth coefficients are given by the zero energy enforcement, the S-matrix elements must be provided for $N_{pts}$=$2M$ unique energy points in order to solve (6) and calculate the elements of $\boldsymbol{\alpha}^{(\mu)}$ and $\boldsymbol{\beta}^{(\mu)}$. This will allow the elements $F_{mn}^{in/out}(E)$ of $\mathbf{F}^{in/out}(E)$ to be determined and thus the parametrised S and Jost matrices obtained.

The first release of our software considers application to multichannel, elastic systems only. Here all the thresholds can be set to zero and, using equation (4), equation (2) can be expressed purely in terms of a single $k$ as:

$$F_{mn}^{out/in}(k) = \frac{1}{2}\sum_{\mu=0}^{M}\left[\alpha_{mn}^{(\mu)}k^{l_n-l_m+2\mu} \pm i\beta_{mn}^{(\mu)}k^{l_n+l_m+1+2\mu}\right] \tag{7}$$

Since $F_{mn}^{out/in}(k)$ is a polynomial function of $k$, and the coefficients have been obtained solving equation (6), we can utilise well known numerical methods for the solving of polynomial systems [15, 16, 17].

*2.2. Locating the S-matrix poles*

The S-matrix poles are obtained by first finding the roots of:

$$\det\left[\mathbf{F}^{in}(k)\right] = 0 \tag{8}$$

The total number of roots will be dictated by the degree of the polynomial obtained after insertion of (7) into (8) and the subsequent expansion of the determinant. This means it will depend on the system that we choose to solve: the number of channels, their angular momenta and, of course, $M$. Generally not all of these roots correspond to true poles of the S-matrix. However, as pointed out in [12], the roots corresponding to the true poles can be expected to converge onto the 'true' value as $M$ increases while the other roots will move about in a random manner. Therefore, locating the poles involves comparing the roots across a range of increasing values of $M$.

To assist in the understanding of the following, more detailed, description of the routines involved in this task, we introduce the following notation:

- The increasing $M$ sequence is indexed using $m$.

- The set of roots found at a particular $M_m$ is labelled $R_m$.
- Root $i$ within the root set $R_m$ is is denoted $R_m[i]$
- The number of steps across successive $M_m$ for which a comparison test will be applied is denoted by $cf$.
- Real and imaginary components of the root are indicated using $\mathbb{R}$ and $\mathbb{I}$ subscripts respectively.

The procedure will use a comparison test (explained below) to determine, for each root in $R_m$, if any of the roots in $R_{m+1}$ is close enough to it. Those in $R_{m+1}$ that pass the comparison test (and are thus designated close) are flagged as possible poles. If a root in $R_m$ is close to more than one in $R_{m+1}$ then the root in $R_{m+1}$ that is closest to the root in $R_m$ is used. At this point if $cf = 1$ then the test is complete and the flagged roots are designated as candidate poles.

If $cf > 1$ then the flagged roots in $R_{m+1}$ are used to test against all of those in $R_{m+2}$ in a manner analogous to that just described, with the flagged roots updating to those in $R_{m+2}$ that passed the comparison test. This continues until $cf$ steps are reached, at which point the final set of flagged roots in $R_{m+cf}$ are designated as candidate poles. At this point the routine described in Section 2.3 is executed to assign quality indicators to each of the candidate poles. Before discussing this we first describe the comparison test.

The comparison test is applied to all possible pairs of roots for successive steps. The test is applied separately to the real and imaginary components:

- If the component is greater than a zero value $zk$ for both steps, then $cdiff$ is calculated using:

$$cdiff_{\mathbb{R},\mathbb{I}} = \frac{|R_n[i]_{\mathbb{R},\mathbb{I}} - R_{n-1}[j]_{\mathbb{R},\mathbb{I}}|}{\max(|R_n[i]_{\mathbb{R},\mathbb{I}}|, |R_{n-1}[j]_{\mathbb{R},\mathbb{I}}|)} \tag{9}$$

- If the component is smaller than or equal to $zk$ for one of the two steps, then it is set to $zk$ for that step and $cdiff$ is calculated using expression (9).

- If the component is smaller than $zk$ for both steps then it will be set to $zk$ and $cdiff$ will also be set to zero.

If both $cdiff_{\mathbb{R}}$ and $cdiff_{\mathbb{I}}$ are less than a respective comparison threshold $dk$ (note that the same value is used for both real and imaginary components) then the comparison test passes and the root is flagged as a candidate pole.

### *2.3. Assessing the quality of the candidate poles*

The identification of a candidate pole in a single application of a location test will therefore depend on $dk$, $zk$ and $M$. In practice, the location test is applied for a range of $dk$ and $M$ as determined by the input provided by the user; $zk$ is also an input parameter in `reskit`. We note that the number of significant figures in the input data (given by the method and computational approached used to generate it) will have an effect on the identification of poles (see below).

Unfortunately, not all candidate poles are true poles of the system, so a way of quantifying the likelihood that these poles are true poles is required. We do this by defining, and calculating, some 'quality indicators' (QI). This

involves applying the location routine repeatedly across a range of $M$ values for decreasing $dk$. The QI are determined in the following way:

- The routine keeps count of how many times a particular candidate pole is located over the $M$ ranges for all of the $dk$s. This value defines the first QI, $\Sigma M$.
- The second QI is the lowest $dk$ for which a candidate pole is identified and is referred to as $\vee dk$.

These QI should provide the user with a mechanism to ascertain the confidence that a candidate pole is a true pole of the system, particularly when compared to any others also discovered in the same data set. However, care must be exercised when choosing the starting value of $dk$ and the value assigned to $zk$, especially with respect to the precision and size of the input data.

## 3. Using the Software

The software to locate S-matrix poles using the method described in Section 2 is made available either from a number of independent python packages or via an encapsulating python package named `reskit`, all of which are installable via provided distutils setup.py scripts. In addition, a pip requirements.txt is provided with `reskit` containing specific version numbers for the known working `reskit` dependencies. We recommend installing into an isolated python environment using, for example, the virtualenv tool.

As well as pulling together the underlying packages (referred to as the `reskit` Utilities) `reskit` has also been designed to provide archiving of

results, handling of low-level parameters and numeric type abstraction. Since it has a modular design, new techniques can be easily constructed using the Utilities and added as what is referred to as a Tool.

As a way of introducing the package and its use, we present and discuss an example program that uses `reskit` to find S-matrix poles using an *ab initio* data set as input; the program shown is a simplification of the test code included with the `reskit` release. The poles are found using the `MCSMatFit` Tool.

```
import reskit as rk
import channelutil as cu
import ukrmolmatreader as rmol

# Use mpmath types (optional)
rk.use_mpmath_types(dps=100)

# Read in the K-matrix data
kmatdict,_ = rmol.read_Kmats("kmatrix_input_pyrazine.txt")

# Get a calculator with units and channel angular momenta
calc = rk.get_asym_calc(rk.rydbergs, [3,5,5])
# Initialise the data into the required container
dkmat = rk.get_dmat_from_discrete(rk.Kmat, kmatdict, calc,
    "pyrazine")
# Slice the data set (optional)
dkmat2 = dkmat[0:1200]

```

```
18 sfittool = rk.get_tool(rk.mcsmatfit, dkmat2, "results")
19
20 # Perform the calculation of the poles and the quality
      indicators
21 cfins = sfittool.get_elastic_Fins(range(2,32,2))
22 sfittool.find_stable_Smat_poles(cfins)
```

To begin, the required modules are imported: `reskit`, `channelutil` and `ukrmolmatreader`. The *ab initio* K-matrix data is read using the `ukrmolmatreader` (line 9). Further details of this data set, obtained for the pyrazine molecule can be found in Section 5. The `ukrmolmatreader` parser is specific to the particular file structure of our input data but all that is required by `reskit` is a python dictionary of either S-, K- or T-matrices keyed by energy (translation to the required S-matrix will be done internally if required). `reskit` supports both python types using `numpy` operations (default) or the arbitrary precision offered by the `mpmath` python library. Line 6 specifies to use `mpmath` with 100 decimal places.

The angular momenta, spin (for the calculation of the cross section) and energy units are provided by the user and then contained as an `AsymCalc` (asymptotic calculator) object, which is returned from the `get_asym_calc` function (line 12). In our example, it specifies that the unit of energies are Rydberg (Ry) and that there are three channels of angular momenta 3, 5 and 5.

The call to `get_dmat_from_discrete` in line 14 translates the user's scattering data, stored in `kmatdict`, into a `reskit` compatible container type. Here, the first argument specifies the scattering matrix type of the user

data, the scattering matrix itself is supplied as the second argument. The third argument is the `AsymCalc` instance, `calc`, that was returned in line 12. The `"pyrazine"` argument is a string given to describe the data set, which is used to build the path for saving and loading the results.

An accompanying function to `get_dmat_from_discrete`, which is called `get_dmat_from_continuous`, exists for the situation when the scattering data is given by an functional expression.

After creation of the `reskit` compatible container, it is reduced in size in line 16. This reduction is carried out using the python dictionary interface, in this case taking the first 1200 energy points only.

The first argument of the `rk.get_tool` function in line 18 is specifying the required Tool (`MCSMatFit`), the second is the `reskit` container created above, `dkmat2`. The third argument specifies a path into which any results (for example the coefficients calculated by `MCSMatFit`) generated by the Tool can be saved; if it is not supplied then the Tool will only programmatically return the generated results without any writes to disk. More is said on the storage of results at the end of this section.

`rk.get_tool` has an optional fourth parameter, `param_file_path` to allow overriding the default arguments for the low level routines. These include additional parameters for the `numpy` and `mpmath` routines, as well as more advanced `reskit` related parameters. By convention, each Tool provides a default.yaml file containing a set of default arguments. The user can supply their own arguments by passing a path argument for `param_file_path`, specifying the location of their overriding yaml file. Any user provided yaml file should match the structure and naming of the default yaml it is overrid-

ing.

Finally Lines 21 and 22 call the functions that perform the actual calculation. `get_elastic_Fins` accepts a range of $N_{pts}$ and returns a list of $F^{in}$ (see equation 2) for each of the supplied $N_{pts}$ (in this case the list returned from the `range` function inclusively contains $N_{pts}$ from 2 to 30 with a step of 2). It does this by solving equation (6) for the coefficients and then using expansion (7). The $F^{in}$ (contained in the list `cfins`) are then passed to the `find_stable_Smat_poles` function which finds the roots and calculates the QIs as described in Section 2.3.

As mentioned, results from the calculations are written to disk if a results path has been specified in the call to the `get_tool` function. `reskit` provides a 'smart archive', in that results are written to a location depending on the supplied data, the arguments of the calculation and whether `numpy` or `mpmath` types are used. This means that any result written into the archive will have enough accompanying information to allow it to be easily reproduced and to allow correct loading of any intermediate calculation results if available and required. All of the results are written relative to the path supplied to the `get_tool` function. As an example, the path to the `find_stable_Smat_poles` results in our example will look like:

```
results/pyrazine/mpmath_100/(0,1200,None)/mcsmatfit/poles/default/[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30]
```

The names `results` and `pyrazine` are provided by the user as the example above shows. `(0,1200,None)` refers to the reduction of the data set in line 16, `mpmath_100` and `mcsmatfit` to the types and Tool used in the calculation and `default` to the yaml file containing the Tool arguments. The

list of numbers in square brackets are the $N_{pts}$ used in the calculation. Similar paths will exist for the coefficients and roots obtained during the calculation, so that, for example, if the user decides to rerun the calculation including $N_{pts} = 32$ with no other changes, the existing coefficients and roots (i.e. those for values of $N_{pts}$ already used) will be loaded from the archive. The final results will then be stored in an alternative folder at the end of the above path named `[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32]`.

## 4. `reskit` package

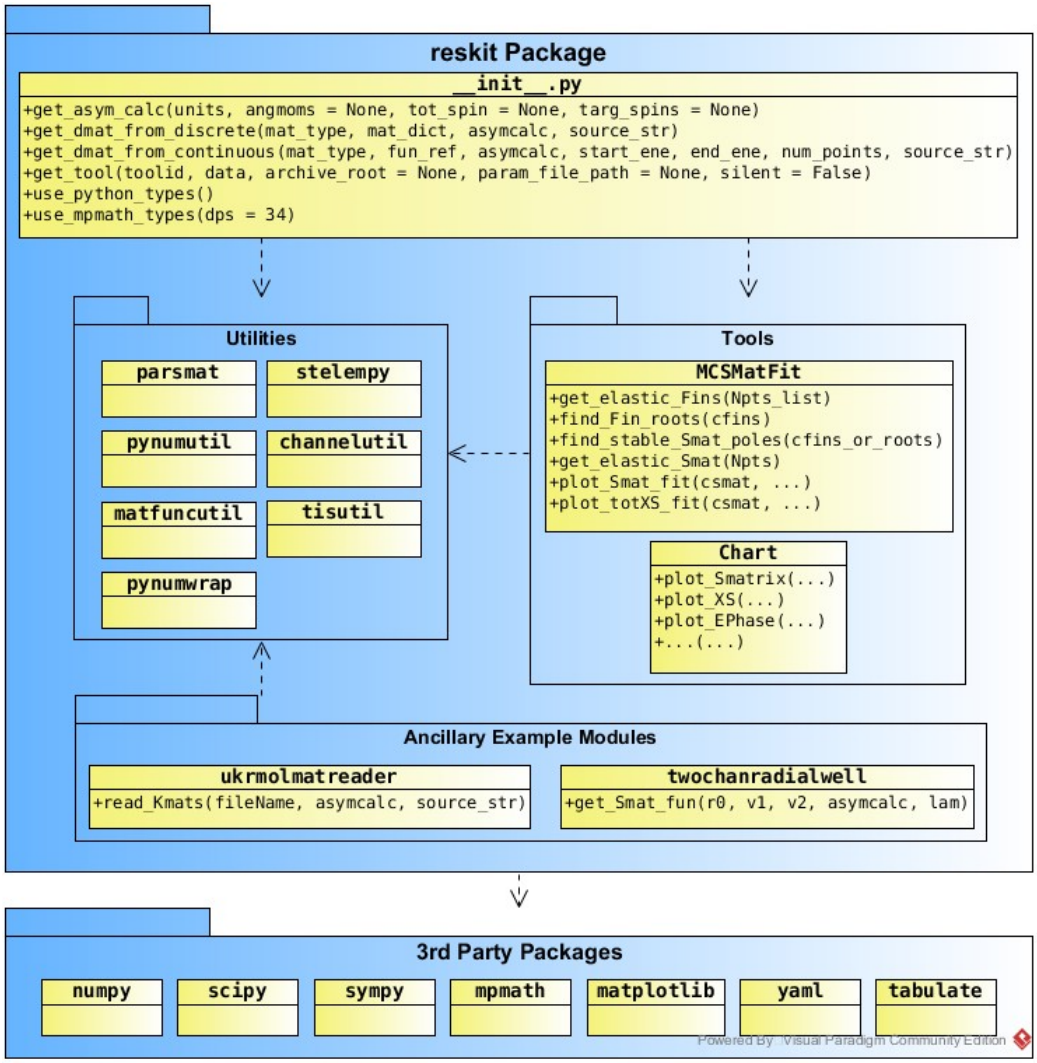


Figure 1: Schematic diagram detailing high level API and public functions for the `reskit` python package. The `reskit __init__.py` file contains the main interface functions. The `Tools` group contains the Tools that are returned from the `get_tool` function in the `__init__.py`. Also shown are the low-level `Utilities`, the two modules provided for test and demonstration purposes and the required third party packages.

Figure 1 shows the architecture of `reskit`. At the lowest level is the Utilities group. This group encompasses a set of python packages which provide numerical operations and other fundamental functionality. The Tools, of which there are currently two, are dependent on the Utilities and are shown along with their public interfaces.

The `reskit` package file `__init__.py` provides access to the Tools and other general functionality, such as the initialisation of data into the required containers and type configuration. Also shown in figure 1 are two classes used to obtain the data for the examples provided (Ancilliary Example Modules) as well as the external packages required by `reskit`.

The typical user isn't required to have any knowledge of the packages contained within the Utilities group; the more advanced user can refer to the detailed documentation provided within each of the package folders. This documentation provides instructions for using each Utility package either as a standalone entity or for building Tools and/or other Utilities that can interact with or be integrated into `reskit`. The remainder of this section will provide a brief overview of the various Utility packages before presenting a more detailed reference of the `reskit` and Tools interfaces.

### *4.1. `reskit` Utilities*

The Utilities group contains the building blocks of `reskit`. It includes numerical routines that are used internally by the Tools, as well as the containers that are returned to the user from the `reskit` interface (e.g. from the `get_dmat_from_discrete` function called in the example program given in Section 3) and are used throughout the `reskit` software.

The `pynumwrap` package wraps the two different numeric types, standard python and `mpmath`, supported by `reskit` behind a common interface. The user can specify what types they want to work with by calling the `use_python_types()` or `use_mpmath_types(dps)`. The latter of these functions offers arithmetic to a specified number of decimal places (`dps`) and as such allows more accurate (but slower) calculations to be carried out.

The `matfuncutil` package provides containers for the discrete and continuous representations of data that are typically associated with a scattering problem. The discrete containers for matrix, vector and scalar types are named `dMat`, `dVec` and `dSca` respectively and provide functionality for charting and operations such as slicing and selection of the fit data, as well as useful functions such as `eigenvalues()` and `gradient()`. Operating on a container will transform the `dMat` (or `dVec` or `dSca`) into an appropriate container containing the results of the operation.

The analogous `matfuncutil` continuous containers are named `cMat`, `cVec` and `cSca`. The main purpose here is to wrap a provided function reference and to allow easy discretisation to an appropriate discrete container. In addition, the `cMatSympypoly` and `cScaSympypoly` continuous containers are provided as extensions of the `cMat` and `cSca` to further represent symbolic matrices and scalars via the popular `sympy` python package. These symbolic containers provide a simple interface for determinant and root finding.

`tisutil` provides extension of the `matfuncutil` containers to the various scattering representations: `dSmat`, `dKmat`, `dTmat`, `cSmat`, `cKmat` and `cTmat`. These containers are returned from the `get_dmat_from_discrete`

and `get_dmat_from_continuous` functions and passed to the `get_tool` function in the `reskit __init__.py` package file. They are also returned from the functions in the `ukrmolmatreader` and `twochanradialwell` example modules. These containers provide functionality for converting between the various scattering representations, routines for the calculation of cross sections (`dXSmat`) and eigenphase sums (`dEPhaseSca`), as well as the interface for the `matfuncutil` discrete containers. `cMatSympypolyk` is also provided as an extension of the `matfuncutil cMatSympypoly` container, to allow parmetrisation of a symbolic matrix by energy using the `AsymCalc`. The `cFinMatSympypolyk`, returned from `get_elastic_Fin`, is a `cMatSympypolyk`.

The implementation of the technique described in Section 2 for the location of S-matrix poles is mainly contained in the `parsmat` and `stelempy` Utility packages. `parsmat` is responsible for solving Equation (6) for the $\boldsymbol{\alpha}^{(\mu)}$ and $\boldsymbol{\beta}^{(\mu)}$ coefficients and using them to form either a rational S-matrix (Equation (1)) or an $F^{in}$ (Equation (2)), depending on user choice. For the location of the S-matrix poles, a list of $F^{in}$ are first returned as `cFinMatSympypolyk` containers. The roots of the $F^{in}$ are then found using the `cMatSympypolyk.find_roots` function and passed to the `stelempy` package. `stelempy` was designed to find and quantify stable elements over a number of sets and allows the S-matrix poles to be identified using the technique described in Sections 2.2 and 2.3. The comparison test, summarised by Equation (9), is implemented as the `RationalCompare1` class in the `pynumutil` package.

`channelutil` contains the `AsymCalc` class containing the channel infor-

mation, such as threshold energies, angular momenta and spin as well as functions to convert between energy and momentum. The `reskit __init__.py` file provides the helper function `get_asym_calc` to return an `AsymCalc`.

### *4.2. `reskit` Interface*

The interface to `reskit` is through the `__init__.py` package file shown in figure 1. This section provides a summary of the available functions. Some of these are employed when a discrete set of scattering data (generated by other codes) is used and some when a functional expression is provided for these data. The parameters of each of the functions are listed and described.

`get_asym_calc`: Returns an `AsymCalc` for converting from momentum to energy.

- `units` (*int*): Specification of the energy units. Available options are `reskit.rydbergs`, `reskit.hartrees` and `reskit.eVs`.
- `angmoms` (*list of ints, optional*): Specification of the angular momenta in each of the channels. Defaults to zero in all channels.
- `tot_spin` (*float, optional*): Specification of the total spin of the system. Defaults to $\frac{1}{2}$. Only required for calculation of cross sections.
- `targ_spins` (*float or list of floats, optional*): Specification of the spin of the target electronic state associated with each of the channels. Defaults to zero in all channels. Only required for calculation of cross sections.

`get_dmat_from_discrete`: Converts discrete energy dependent scattering data into a reskit compatible container (i.e. `dSmat`, `dTmat` or `dKmat`).

Types must match those specified using the `use_python_types` or `use_mpmath_types` functions.

- `mat_type` (*int*): Specification of the scattering matrix type. Available options are `reskit.Smat`, `reskit.Kmat` and `reskit.Tmat`.
- `mat_dict` (*energy dict of scattering matrices*): Scattering data to be used in the calculation. Can be either floats or `mpmath` types.
- `asymcalc` (*AsymCalc*): As returned from the `get_asym_calc` function.
- `source_str` (*str*): String provided to uniquely identify the scattering data. Will be used in the archiving of results.

`get_dmat_from_continuous`: Discretises continuous energy dependent scattering data into a reskit compatible container (i.e. `dSmat`, `dTmat` or `dKmat`). Types must match those specified using the `use_python_types` or `use_mpmath_types` functions.

- `mat_type` (*int*): As for `get_dmat_from_discrete`.
- `fun_ref` (*function with float parameter*): An energy function describing the elements of the scattering matrix. Can be either python float or `mpmath.mpf` type.
- `asymcalc` (*AsymCalc*): As for `get_dmat_from_discrete`.
- `start_ene` (*float*): Start energy for the discretisation.
- `end_ene` (*float*): End energy for the discretisation.
- `num_points` (*float*): Number of energy points for the discretisation.
- `source_str` (*str*): As for `get_dmat_from_discrete`.

`get_tool`: Initialises and returns a Tool.

- `toolid` (*int*): Specification of the Tool. Available options are `reskit.chart` and `reskit.mcsmatfit`.
- `data` : Tool data. This is the data container to be used by the Tool.
- `archive_root` (*str, optional*): Specification of the location into which reskit will write its results.
- `param_file_path` (*str, optional*): Location of an existing yaml file containing overrides for the more advanced routine parameters.
- `silent` (*bool, optional*): Switch determining whether to suppress output to console.

`use_python_types`: Specifies to use python types.

`use_mpmath_types`: Specifies to use `mpmath` types.

- `dps` (*int*): Specifies the `mpmath` precision.

### *4.3. Tools*

In this section we describe the functionality provided by the Tools. `MCSMatFit` performs the analytic fit and pole identification whereas `Chart` provides functionality for the basic plotting of data.

#### *4.3.1. `MCSMatFit`*

These routines perform the fit and identify the poles as described in Section 2.

`get_elastic_Fins`: Performs $F^{in}$ fits using the specified list of fit points and returns a list of `cFinMatSympypolyk`.

- `Npts_list` (*list of ints*): List of $N_{pts}$ to be used for successive fits.

`find_Fin_roots`: Returns the roots of a list of parameterised $F^{in}$ as a list of python complex or `mpmath.mpc` types. There are additional advanced parameters supplied via the Tool yaml file.

- `cfins` (*list of cFinMatSympypolyk*): List of parameterised $F^{in}$.

`find_stable_Smat_poles`: Finds the S-matrix poles by identifying stable roots. The input can be either a list of roots or the $F^{in}$ themselves. There are additional advanced parameters supplied via the Tool yaml file.

- `cfins_or_roots` (*list of either cFinMatSympypolyk or list of floats*): As returned from either `get_elastic_Fins` or `find_Fin_roots`.

`get_elastic_Smat`: Performs an S-matrix fit using the specified number of fit points and returns a `cSmat`.

- `Npts` (*int*): Number of points to use in the fit. Must be an even number.

`plot_Smat_fit`: Plots the specified matrix element(s) of the original and rational S-matrices and the fit points used. There are additional advanced parameters supplied via the Tool yaml file.

- `csmat` (*cSmat*): Rational S-matrix returned from `get_elastic_Smat`.
- `num_plot_points, units, i, j, logx, logy, imag` : Refer to the `chart` Tool for description.

`plot_XS_fit`: Plots the cross sections obtained from the original and rational S-matrices along with the fit points used. There are additional advanced parameters supplied via the Tool yaml file.

- `csmat` (*cSmat*): Rational S-matrix returned from `get_elastic_Smat`.

- `num_plot_points, units, logx, logy` : Refer to the `chart` Tool for description.

*4.3.2. Chart*

`plot_raw, plot_Smatrix, plot_Kmatrix, plot_Tmatrix, plot_UniOpSMat, plot_EphaseSum, plot_XS`: Plots various scattering related quantities. A png image of the plot will be automatically saved into the archive.

- `start / end` (*int or float, optional*): Indicates the start/end index (if int) or the nearest start/end energy (if float).
- `num_plot_points` (*int, optional*): The number of points to plot, evenly distributed between start and end.
- `units` (*int, optional*): If specified, then will convert to these units prior to plotting. Available options are `reskit.rydbergs`, `reskit.hartrees` and `reskit.eVs`.
- `logx` (*bool, optional*): Switch to turn on x-axis log plotting.
- `logy` (*bool, optional*): Switch to turn on y-axis log plotting.
- `imag` (*bool, optional*): For complex quantities, switch to plot the imaginary component. By default just plots the real component.
- `i` (*int, optional*): Zero-based row index to plot. Default is to plot all rows.
- `j` (*int, optional*): Zero-based column index to plot. Default is to plot all columns.

The `i` and `j` parameters are only available when the quantity to plot is a matrix (i.e. not for `plot_Ephase` and `plot_XS`).

## 5. Example runs

In this section, we present some results obtained with the `reskit` package and discuss how to interpret them, in particular what the values of the QI say about the likelihood that candidate poles identified in the calculations are true poles of the system. The calculations correspond to the test runs that are provided in the code release and the figures presented have been created using the charting functionality provided by `reskit`. Atomic units are used throughout.

### *5.1. Square well*

A simple test system is the elastic $s$-wave scattering two-channel radial square well for which the interaction potential between channels $n$ and $m$ can be written as:

$$V_{mn}(r) = \begin{cases} 0 & r > a \\ v_{mn} & r < a \end{cases} \tag{10}$$

where $v_{mn}$ is a 2×2 matrix containing the channel potentials, $v_{11}$ and $v_{22}$, and a coupling factor $\lambda$:

$$v_{mn} = -\begin{pmatrix} v_{11} & 0.5\lambda \\ 0.5\lambda & v_{22} \end{pmatrix} \tag{11}$$

Here $v_{11}$, $v_{22}$ and $\lambda$ are constants. The exact solution for the S-matrix for this system is known [4].

Table 1 compares the values of the S-matrix poles obtained using `reskit` with those obtained numerically[1] from the exact solutions given in [4] for a

[1]The exact solutions given in [4] describe scattering from a general two channel radial

well with $a$=1.0, $v_{11}$=$v_{22}$=2.0 and $\lambda$=1.0. For the `reskit` calculation, even $N_{pts}$ were used by fitting $N_{pts}$=$2M$ points in the interval 1.0-8.0 Ha, with a starting $dk$ of $10^{-4}$ and testing across 3 successive $M$ (i.e. $cf$=2). The $N_{pts}$ points were selected from a set of 1000 evenly spaced energies points for which the S-matrix was calculated using the functional form given in [4]; 100 decimal places were used throughout these calculations.

The first two poles in the table correspond to bound states while all the others correspond to resonances: both the resonance pole and its corresponding antipole [5] are tabulated. Three things are immediately clear from the table: (i) the method identifies poles well outside the range of the real energies for which S-matrix data is provided; (ii) the QI get worse (that is, $\vee dk$ increases and $\Sigma M$ decreases), as the method 'extrapolates' further away from the data provided; (iii) use of a larger value for $M$ (in the same energy range) enables the identification of poles further from the energy range of the input data set as well as improving the QI for the poles already identified.

Table 1 shows clearly that the QI gets worse as poles move further away from both the real axis and the (real) energies for which the S-matrix data is provided. To further illustrate the effect of 'extrapolating' away from the real axis, figure 2 shows how one of the QI, $\Sigma M$, changes as the imaginary part of a pole (i.e. the width of the resonance it corresponds to) increases. The data corresponds to a specific resonance as the depth of the wells is varied. In order to eliminate the effect of extrapolating the real component of the

---

well. For the bound states the poles were located graphically from a transcendental equation derived from these solutions by the authors. For the resonances, they were obtained using the secant method to locate the zeros of the denominator of the S-matrix

energy, the calculation for each depth was done by shifting the energy range for which the S-matrices are provided, so that the real part of the pole energy is located approximately in the centre of the data set. Unsurprisingly, $\Sigma M$ decreases as the width of the resonance increases, even though all the poles identified by `reskit` are true poles of the system. The example demonstrates that smaller QI ($\Sigma M$ but also $\vee dk$) are to be expected for wider resonances: this should not be necessarily taken to indicate the pole identified is not a true pole of the system.

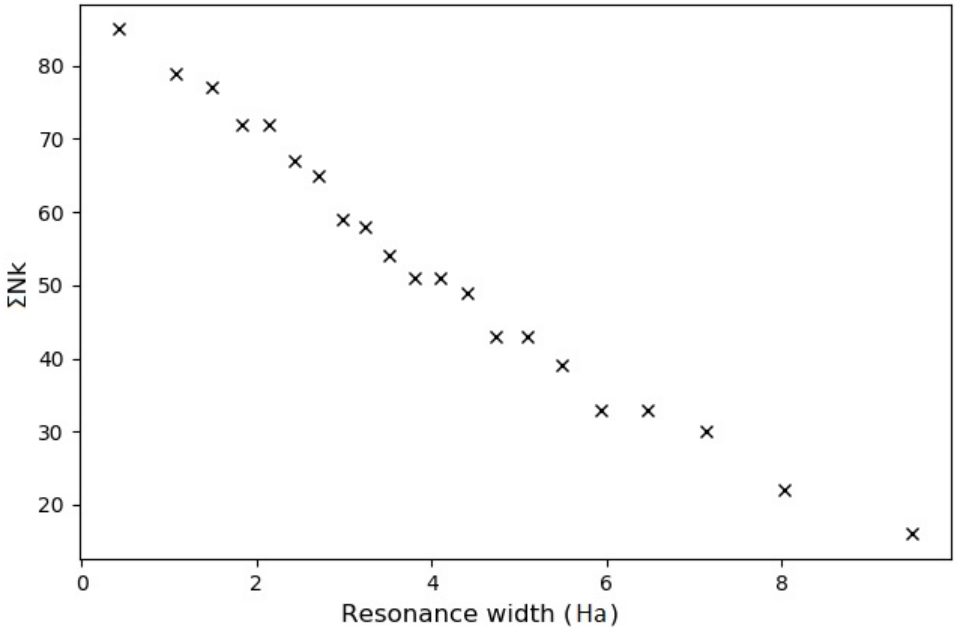


Figure 2: Example pole for the square analytical well: $\Sigma M$ for resonances of different widths (obtained by varying the depth of the wells). Fit data was chosen over a range of 7.0 Ha centred around the real part of the resonance energy. A starting $dk$ of $10^{-4}$ was used, with $cf$=2.

Finally, as an illustration of the convergence of a root, table 2 shows

the value of the root that converges to the third pole listed in table 1 for increasing values of $M$. It can be seen that, in this calculation, both real and imaginary components of the pole converge to more than 20 decimal places. As we will see, this is not the case when *ab initio* S-matrix data is used.

### *5.2. Pyrazine*

This is an example of using `reskit` to locate poles corresponding to resonances from *ab initio* K-matrix data. This data corresponds to 3-channel elastic electron scattering from pyrazine, $C_4N_2H_4$. The *ab initio* R-matrix data has been obtained using the UKRmol suite [18] and the R-matrix method [19] at the static-exchange plus polarization level (for more details of the calculation, see [20]). This molecule belongs to the $D_{2h}$ point group and this example involves channels of $A_u$ symmetry. It is well known [20] that pyrazine possesses a shape resonance (that is, described by elastic scattering) of $A_u$ symmetry with energy around 1 eV and width $\approx 10^{-2}$ eV.

For the `reskit` calculation, even $N_{pts}$ were used in the interval $7.35\times10^{-4}$ to $\sim$0.441368 Ry, with a starting $dk$ of $10^{-4}$, $cf = 3$ and $M_{max} = 20$. The energy interval corresponds to the range where we are interested in locating resonances. For the radial square well, resonances with a real component well outside the range are identified with high QI; however, this is not the case when (less precise) *ab initio* data is used, so the energy range should be chosen more carefully.

Table 3 summarizes the candidate poles. It is immediately clear that the QI in this calculation are, in general, poorer than for the case of the square well, even though the real component of all the candidates poles is inside the energy range of the initial S-matrix data. The candidate pole with

the best QI values (the smallest $\vee dk$ and largest $\Sigma M$) corresponds to the known resonance of the system. This resonance is visible in the elastic cross section plotted in figure 3. This figure shows that the S-matrix obtained using Padé approximants (with $M = 10$) describes the scattering process very well within the region of the data points provided: the agreement between the cross sections obtained from both S-matrices is excellent.

The other poles/antipole pairs in table 3 have significantly poorer QI. Two of them (with $\vee dk = 10^{-5}$) have relatively big imaginary components. The last three have an extremely small $\Sigma M$ and a negative real component: the pair with a non-zero imaginary component is clearly non-physical and we shall not discuss it further.

The issue to address is whether these candidate poles are true poles of the system or not. We make a distinction here: it is possible for poles of the S-matrix of a system not to manifest themselves in physical observables. Therefore it is possible that, even if no resonance (or virtual state) has been identified in a certain energy region by scattering calculations or experiments, a true pole of the S-matrix is present in that region.

The last candidate pole has zero imaginary component and would correspond to a bound/virtual state of the anion $C_4N_2H_4^-$: pyrazine is a system that does not support bound anionic states and the cross section does not show the behaviour expected in the presence of a virtual state. Two explanations are therefore possible: (i) this is a true pole of the rational S-matrix, that therefore does not describe the system well in the energy region around 0 eV but this does not seem likely from figure 3; (ii) this is not a true pole of the rational S-matrix: the poor QI indicate poor convergence of a root for

only a few values of $M$. It is always possible that, by chance, two random roots from successive $M$ have energies less than $dk$ apart and are therefore identified as the same (stable) root. This will be obviously more likely for larger values of $dk$ (but less so for bigger $cf$). Also, the higher the number of channels, the more roots of the S-matrix (for a specific number of fit points) so the likelihood of random roots being identified as candidate poles increases.

| Functional (Ha) | | Parametrised (Ha) | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | $M_{max} = 15$ | | $M_{max} = 20$ | |
| Real | Imag | Real | Imag | $\vee dk$ | $\Sigma M$ | $\vee dk$ | $\Sigma M$ |
| -0.4657 | 0 | -0.4657 | 0 | 1e-27 | 102 | 1e-49 | 289 |
| -0.0307 | 0 | -0.0306 | 0 | 1e-27 | 99 | 1e-49 | 286 |
| 6.131 | 5.94 | 6.131 | 5.940 | 1e-24 | 92 | 1e-45 | 261 |
| 6.129 | -5.94 | 6.131 | -5.940 | 1e-24 | 92 | 1e-45 | 261 |
| 6.482 | 7.13 | 6.482 | 7.129 | 1e-23 | 84 | 1e-42 | 242 |
| 6.48 | -7.13 | 6.482 | -7.129 | 1e-23 | 84 | 1e-42 | 242 |
| 24.481 | 14.65 | 24.481 | 14.650 | 1e-11 | 24 | 1e-25 | 106 |
| 24.482 | -14.65 | 24.481 | -14.650 | 1e-11 | 24 | 1e-25 | 106 |
| 24.696 | 16.623 | 24.696 | 16.623 | 1e-10 | 23 | 1e-24 | 103 |
| 24.697 | -16.623 | 24.696 | -16.623 | 1e-10 | 23 | 1e-24 | 103 |
| 53.041 | 24.572 | 53.040 | 24.571 | 1e-4 | 2 | 1e-14 | 37 |
| 53.04 | -24.572 | 53.040 | -24.571 | 1e-4 | 2 | 1e-14 | 37 |
| 53.168 | 27.347 | 53.168 | 27.347 | 1e-4 | 2 | 1e-14 | 36 |
| 53.169 | -27.347 | 53.168 | -27.347 | 1e-4 | 2 | 1e-14 | 36 |
| 91.663 | 35.372 | 91.662 | 35.371 | - | - | 1e-7 | 8 |
| 91.663 | -35.372 | 91.662 | -35.371 | - | - | 1e-7 | 8 |
| 91.726 | 38.952 | 91.726 | 38.952 | - | - | 1e-7 | 8 |
| 91.726 | -38.952 | 91.726 | -38.952 | - | - | 1e-7 | 8 |

Table 1: S-matrix poles for a two channel, elastic radial square well of width $a$=1.0, depth $v_{11}$=$v_{22}$=2.0 and coupling factor $\lambda$=1.0, as described in [4]. The left-hand column lists the real and imaginary parts, in Hartree (Ha), of the poles obtained numerically from the exact solutions given in [4] while the second column shows the components of the poles located using `reskit`. Results have been truncated to the approximate precision limit of the numerical method used for the left-hand column. The right-hand columns show the QI, $\vee dk$ and $\Sigma M$, for two values of $M_{max}$; if no value is provided for a pole that indicates the pole has not been identified. The input data used in the calculations was specified to 100 decimal places.

| $M$ | Real | Imag |
|---|---|---|
| 2 | 0.947548796707301856015280400671 | 0.265748262516494625022746083132 |
| 3 | 0.137528668651726452078587289159 | 2.333754499518979210913580125889 |
| 4 | 6.105231600017854127389513402724 | 5.940297892975684034502238988090 |
| 5 | 6.114611628886600628783996046546 | 5.941232293795682020948957147385 |
| 6 | 6.131507564765944392832600313036 | 5.940191201112208768288428174064 |
| 7 | 6.131497980760253892033660974623 | 5.940223739578295869731962652009 |
| 8 | 6.131497977767842537429753059739 | 5.940223834729429137996505041263 |
| 9 | 6.131497977999105847638466698984 | 5.940223835529722433870519755797 |
| 10 | 6.131497977998788582791881404577 | 5.940223835529424360668159225567 |
| 11 | 6.131497977998787434690863840223 | 5.940223835529423904158295614678 |
| 12 | 6.131497977998787433834742378356 | 5.940223835529423904169672091908 |
| 13 | 6.131497977998787433834779192569 | 5.940223835529423904169648952163 |
| 14 | 6.131497977998787433834779503703 | 5.940223835529423904169648154793 |
| 15 | 6.131497977998787433834779503717 | 5.940223835529423904169648154758 |

Table 2: Convergence of the pole near 6.13 Ha in table 1 for increasing $M$. These values were extracted from the pole and root tables contained in the author_results folder supplied with `reskit`; specifically the dk1e-04.dat files for $M > 5$ and from the individual root files for the smaller $M$.

| **Real energy** | **Imag energy** | $\vee dk$ | $\Sigma M$ |
|---|---|---|---|
| 0.076641273137 | 6.5298521000e-4 | 1e-11 | 52 |
| 0.076641273137 | -6.5298521000e-4 | 1e-11 | 52 |
| 0.068281 | 0.17860 | 1e-5 | 7 |
| 0.068281 | -0.17860 | 1e-5 | 7 |
| 0.30872 | 0.24350 | 1e-5 | 6 |
| 0.30872 | -0.24350 | 1e-5 | 6 |
| -0.05360 | 0.09512 | 1e-4 | 3 |
| -0.05360 | -0.09512 | 1e-4 | 3 |
| -0.09335 | 0 | 1e-4 | 3 |

Table 3: Real and imaginary components of the poles and QI for 3 channel elastic $A_u$ symmetry scattering from pyrazine obtained with `reskit` using $M_{max} = 20$ in the energy interval $7.35\times10^{-4}$ to 0.441368 Ry, a starting $dk$ of $10^{-4}$ and $cf$=3. Any quantity whose absolute value is less than $10^{-20}$ is regarded, and tabulated, as zero. Note that the poles are listed in order of deteriorating QI, not in energy order.

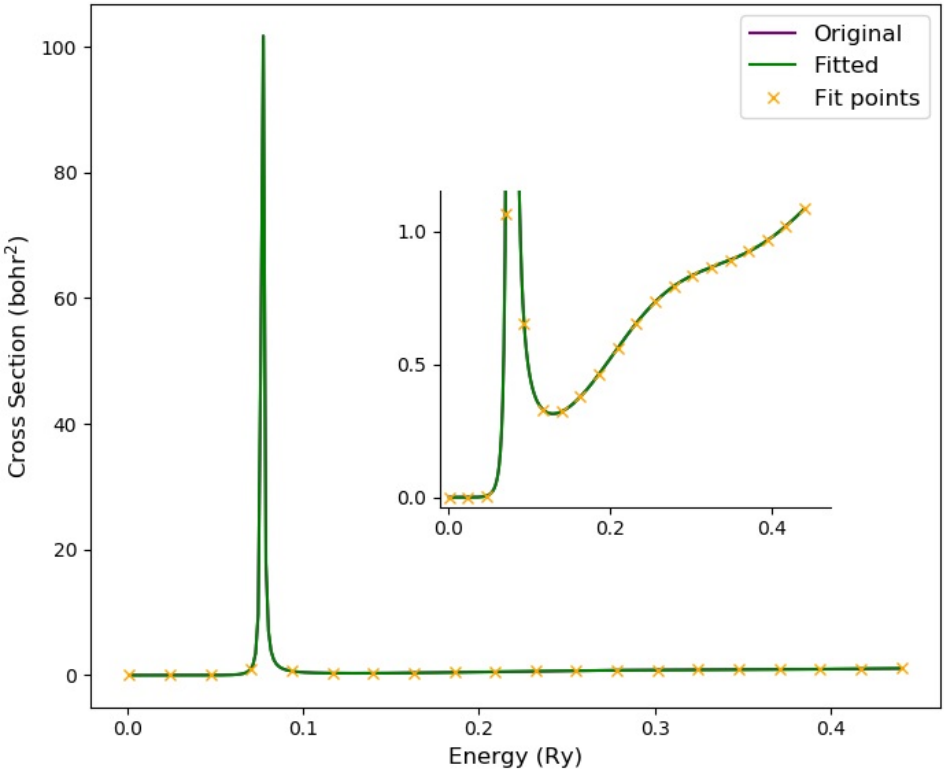


Figure 3: $A_u$ contribution to the elastic cross section for the 3-channel scattering from pyrazine.'Original' indicates the cross section obtained from the *ab initio* S-matrix, 'Fitted' corresponds to the cross section obtained from the rational S-matrix and the 'Fit points' ($N_{pts}$=20) are those used in the fit. The 'Original' line is completely hidden under the 'Fitted' line.

The other candidate poles, that would correspond to resonances, have not been identified in the literature nor do they seem to have an effect in the elastic cross section. Their poor QI could be due, as was shown in the case of the radial square well, to the fact that these poles lie further away from the real axis. However, their QI do not improve significantly when $M_{max}$ is increased to 48, whereas $\Sigma M$ increases significantly (to 80) for the first pole/antipole pair.

One way of establishing whether candidate poles correspond to random roots is to perform the same calculation changing the range of energies for which the *ab initio* data is provided [13]. This will entail either changing the starting energy, the energy range or both. Tests have shown that the energy range must be neither too broad not too narrow as the QI will be poorer; the ranges used in these examples have been found optimal after a number of tests. It is, however, difficult to define a set of energy ranges and starting points for these kinds of tests. In the case of scattering from pyrazine, both changes to the starting point and the range were investigated: in some cases, all candidate poles, except for the first pair at the top of table 3 disappeared. For others, the poles with the large imaginary part remained, but always with very low QI. We therefore take a sceptical approach and deem it unlikely that the candidate poles with poor QI ($\vee dk \geq 10^{-5}$) are true poles of the S-matrix.

The poorer QI for this example compared to those for the square well can probably be attributed to the precision of the input data provided. Whereas in the case of our square well calculations the S-matrix data was determined to 100 decimal places, the *ab initio* R-matrix calculations have significantly

lower precision.

*5.3. Uracil*

In this example, data from a calculation of 6-channel elastic electron scattering from uracil, $C_4N_2O_2H_4$ is used. The *ab initio* data has again been obtained using the UKRmol suite and the R-matrix method. The calculations were performed at the static-exchange level and, in order to reduce the number of channels, only partial waves up to $l$=3 were included (for more details of the calculation on which this one was based see [21]). Unlike pyrazine, uracil is a polar molecule (its dipole moment is around 4.4 D); this means that a long range interaction affects the scattering process.

| **Real energy** | **Imag energy** | $\vee dk$ | $\Sigma M$ |
|---|---|---|---|
| 0.340841151 | 9.93430232e-3 | 1e-9 | 32 |
| 0.340841151 | -9.93430232e-3 | 1e-9 | 32 |
| 0.6090283 | 0.07591980 | 1e-7 | 22 |
| 0.6090283 | -0.07591980 | 1e-7 | 22 |
| 0.1727679 | 8.997753e-3 | 1e-7 | 11 |
| 0.1727679 | -8.997753e-3 | 1e-7 | 11 |
| 0.4118 | 0.2389 | 1e-4 | 4 |
| 0.4118 | -0.2389 | 1e-4 | 4 |

Table 4: Real and imaginary components of the poles and QI for 6 channel elastic scattering from uracil in the A″ symmetry obtained with `reskit` using with $M_{max} = 20$ in the energy interval $7.35\times10^{-4}$ to 0.882000 Ry and $cf = 3$. Again, the poles are listed in order of worsening QI, not in energy order.

This molecule belongs to the $C_s$ point group and the example involves

channels of A″ symmetry. It is well known [20] that this system possesses three shape resonances of this symmetry that will appear approximately at the following energies for a static-exchange calculation: 2.27, 4.05 and 8.0 eV, i.e. 0.17, 0.30 and 0.59 Ry [22].

For the `reskit` calculation, even $N_{pts}$ were used in the interval $7.35\times10^{-4}$ to ∼0.8820 Ry, with a starting $dk$ of $10^{-4}$, $cf = 3$ and $M_{max} = 20$. Table 4 summarises the candidate poles found. Again, we see poorer QI than for the radial square well but also pyrazine, the latter most likely due to the increase in the number of channels. The first three candidate pole/antipole pairs correspond to previously identified resonances. The last one would describe a rather wide resonance: as discussed before, given the size of the imaginary component, we would expect poorer QI. However, we think it more likely that this is not a true pole of the S-matrix, given, in particular the large value of $\vee dk$.

Figure 4 shows the contribution to the cross section of the A″ symmetry obtained from the original *ab initio* S-matrix and the rational one. Once again the agreement between the two is excellent, but only down to around 0.07 Ry. Below that energy, the cross section obtained from the rational S-matrix shows unphysical peaks. We believe this behaviour is related to the enforcement of the condition that the S-matrix becomes the identity matrix when $k \to 0$ to obtain the system of equations 6 is not appropriate for polar molecules in the fixed-nuclei approximation. Clearly, this problem does not prevent the identification of poles at higher energies, but we would expect that it will prevent the identification of poles in the low energy range.

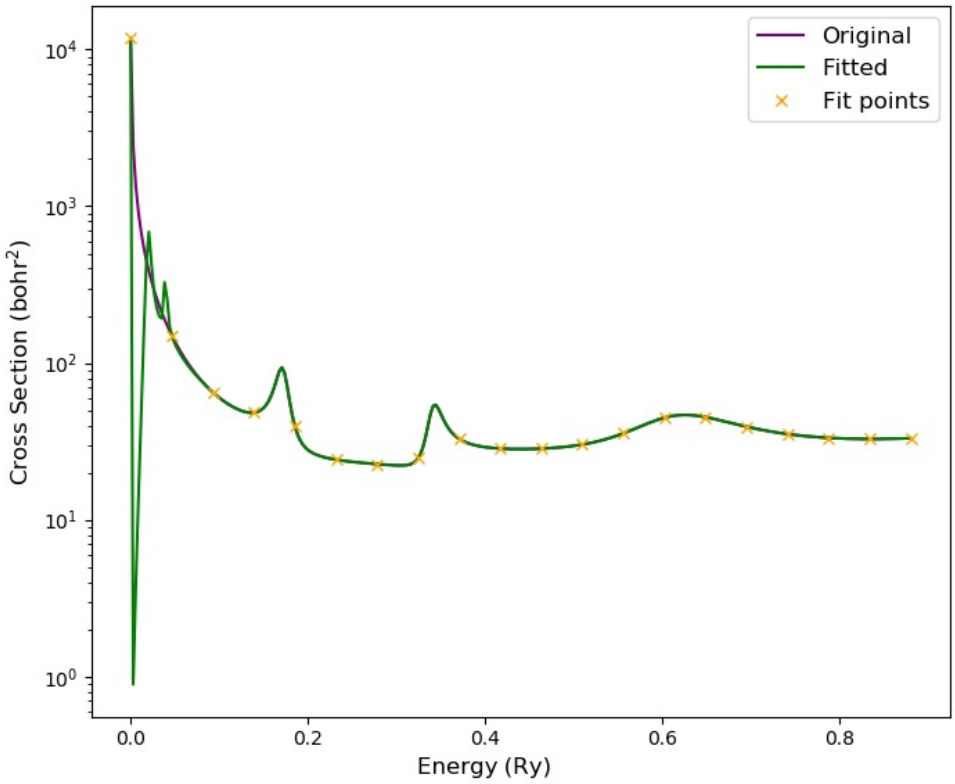


Figure 4: A″ contribution to the elastic cross section for 6-channel scattering from uracil. 'Original' indicates the cross section obtained from the *ab initio* S-matrix, 'Fitted' corresponds to the cross section obtained from the rational S-matrix and the 'Fit points' ($N_{pts}$=20) are those used in the fit.

## 6. Conclusions

We have developed a python toolkit that searches for the complex poles of an S-matrix. The toolkit provides routines to: (i) perform an analytic continuation of the S-matrix from the real axis into the complex plane by constructing the Padé approximant; (ii) locate the roots of the resulting polynomials and (iii) identify those roots that are stable and therefore correspond to true poles of the system by determining two quantities that act as quality indicators. The scattering data can be provided, for real energy values, as S-, T- or K-matrices. The package also includes routines for a number of auxiliary tasks, e.g., determining cross sections, plotting results, etc.

The toolkit was applied to three example cases involving 2 (square well), 3 (electron scattering from pyrazine) and 6 (electron scattering from uracil) channels. For the square well, two bound states and a number of resonance states were identified with high accuracy: the value of QI for most of the poles identified is high. When *ab initio* data for electron scattering from pyrazine and uracil is used, the QI are much lower, probably due to the fewer number of significant figures of the input data. All resonances previously identified for the two molecules are found. Additionally poles corresponding to very wide resonances are identified but deemed likely not to be true poles of the systems.

The QI provide a way of ascertaining the likelihood that an identified pole will be a true pole of the system. However, the further away from the real axis that a pole is located, the poorer the QI indicators will be. As a result, application of the routine to very wide resonances will yield 'low confidence' results, making it hard to decide if the pole is a true pole of the system or

not. In the uracil example presented in this paper, the potential resonance would have had a width of more than 6 eV; resonances of similar widths are understood to play a role in low energy DEA of molecules like formic acid, glycine and uracil [23] (in the latter case, the resonance in question is of A′ symmetry). We have been unable to test the package for the case of a known (i.e. identified by other means) very wide physical resonance.

The present implementation, although based on a general methodology, currently only works for systems in which all the channels have the same energy (i.e. for elastic scattering). For inelastic scattering, locating the roots of the S-matrix is much harder, since the $F^{in}$ is no longer a simple polynomial and techniques such as that proposed by Delves [24] must be employed. Initial tests for the square well and some *ab initio* data using the Delves technique showed some promise but further refinement is required to the routines when applied to the *ab initio* data, especially for the case when resonances are close to threshold/s. Further releases of the code are also expected to incorporate the correct behaviour of the scattering data for polar molecules at very low energies.

## 7. Acknowledgements

We are grateful to Dr. Zdeňek Mašín for bringing this approach to our attention and to Thomas Hird for coding an initial version during a summer undergradaute placement at The Open University.

## 8. References


[1] I. I. Fabrikant, S. Eden, N. J. Mason, J. Fedor, Chapter nine - recent progress in dissociative electron attachment: From diatomics to biomolecules, in: C. C. L. Ennio Arimondo, S. F. Yelin (Eds.), Advances In Atomic, Molecular, and Optical Physics, Vol. 66 of Advances In Atomic, Molecular, and Optical Physics, Academic Press, 2017, pp. 545–657. doi:10.1016/bs.aamop.2017.02.002.

[2] J. Tennyson, C. J. Noble, RESON - A program for the detection and fitting of Breit-Wigner resonances, Computer Physics Communications 33 (4) (1984) 421 – 424. doi:https://doi.org/10.1016/0010-4655(84)90147-4.

[3] K. Bartschat, P. Burke, Resfit - A multichannel resonance fitting program, Computer Physics Communications 41 (1) (1986) 75 – 84. doi:http://dx.doi.org/10.1016/0010-4655(86)90022-6.

[4] R. G. Newton, Scattering theory of waves and particles, Dover Publications Inc., Mineola, NY, 2002. doi:10.1007/978-3-642-88128-2.

[5] J. R. Taylor, Scattering Theory: The quantum Theory on Nonrelativistic Collisions, Wiley, New York, 1972.

[6] T. Belozerova, V. Henner, Overlapping resonances in multichannel reactions, Physics of Particles and Nuclei 29 (1) (1998) 63–87. doi:10.1134/1.953060.

[7] F. T. Smith, Lifetime matrix in collision theory, Physical Review 118 (1) (1960) 349–356. doi:10.1103/PhysRev.118.349.

[8] D. T. Stibbe, J. Tennyson, Timedel: A program for the detection and parameterization of resonances using the time-delay matrix, Computer Physics Communications 114 (1-3) (1998) 236–242. doi:10.1016/s0010-4655(98)00070-8.

[9] D. A. Little, J. Tennyson, M. Plummer, C. J. Noble, A. G. Sunderland, Timedeln: A programme for the detection and parametrization of overlapping resonances using the time-delay method, Computer Physics Communications 215 (2017) 137 – 148. doi:https://doi.org/10.1016/j.cpc.2017.01.005.

[10] J. Horáček, L. Pichl, Calculation of resonance s-matrix poles by means of analytic continuation in the coupling constant, Communications in Computational Physics 21 (4) (2017) 1154–1172. doi:10.4208/cicp.OA-2016-0068.

[11] J. Horáček, I. Paidarová, R. Čurík, On a simple way to calculate electronic resonances for polyatomic molecules, The Journal of Chemical Physics 143 (18) (2015) 184102. doi:10.1063/1.4935052.

[12] P. Ogunbade, S. Rakityansky, S-matrix parametrization as a way of locating quantum resonances and bound states: multichannel case, in: Proceedings of the 2nd South Africa - JINR SYMPOSIUM, Models and Methods in Few- and Many-Body Systems, JINR, 2010, pp. 52–61.

[13] S. A. Rakityansky, S. A. Sofianos, N. Elander, Padé approximation of the S-matrix as a way of locating quantum resonances and bound states,

Journal of Physics A: Mathematical and Theoretical 40 (49) (2007) 14857. doi:10.1088/1751-8113/40/49/017.

[14] S. A. Rakityansky, N. Elander, Analytic structure of the multichannel jost matrix for potentials with coulombic tails, Journal of Mathematical Physics 54 (12) (2013) 122112. doi:10.1063/1.4853855.

[15] R. A. Horn, C. R. Johnson, Matrix Analysis, Cambridge University Press, 1985.

[16] E. Durand, Solutions numeriques des equations algebriques. 1, Equations du type F(x)=0: racines d'un polynome., Paris: Masson, 1960.

[17] I. O. Kerner, Ein gesamtschrittverfahren zur berechnung der nullstellen von polynomen, Numerische Mathematik 8 (3) (1966) 290–&. doi:10.1007/BF02162564.

[18] J. M. Carr, P. G. Galiatsatos, J. D. Gorfinkiel, A. G. Harvey, M. A. Lysaght, D. Madden, Z. Mašín, M. Plummer, J. Tennyson, H. N. Varambhia, UKRmol: a low-energy electron- and positron-molecule scattering suite, Eur. Phys. J. D 66 (3). doi:10.1140/epjd/e2011-20653-6.

[19] P. G. Burke, R-Matrix Theory of Atomic Collisions: Application to Atomic, Molecular and Optical Processes, Springer, 2011.

[20] Z. Mašín, J. D. Gorfinkiel, Elastic and inelastic low-energy electron collisions with pyrazine, The Journal of Chemical Physics 135 (14) (2011) 144308. doi:10.1063/1.3650236.

[21] Z. Mašín, J. D. Gorfinkiel, Resonance formation in low energy electron scattering from uracil, The European Physical Journal D 68 (5) (2014) 112. doi:10.1140/epjd/e2014-40797-y.

[22] F. Kossoski, M. H. F. Bettega, M. T. d. N. Varella, Shape resonance spectra of uracil, 5-fluorouracil, and 5-chlorouracil, J. Chem. Phys. 140 (2) (2014) 024317. doi:http://dx.doi.org/10.1063/1.4861589.

[23] Fabrikant, Ilya I., Theory of dissociative electron attachment: Biomolecules and clusters, EPJ Web of Conferences 84 (2015) 07001. doi:10.1051/epjconf/20158407001.

[24] L. M. Delves, J. N. Lyness, A numerical method for locating the zeros of an analytic function, Math. Comp. 21 (1967) 543. doi:10.1090/S0025-5718-1967-0228165-4.