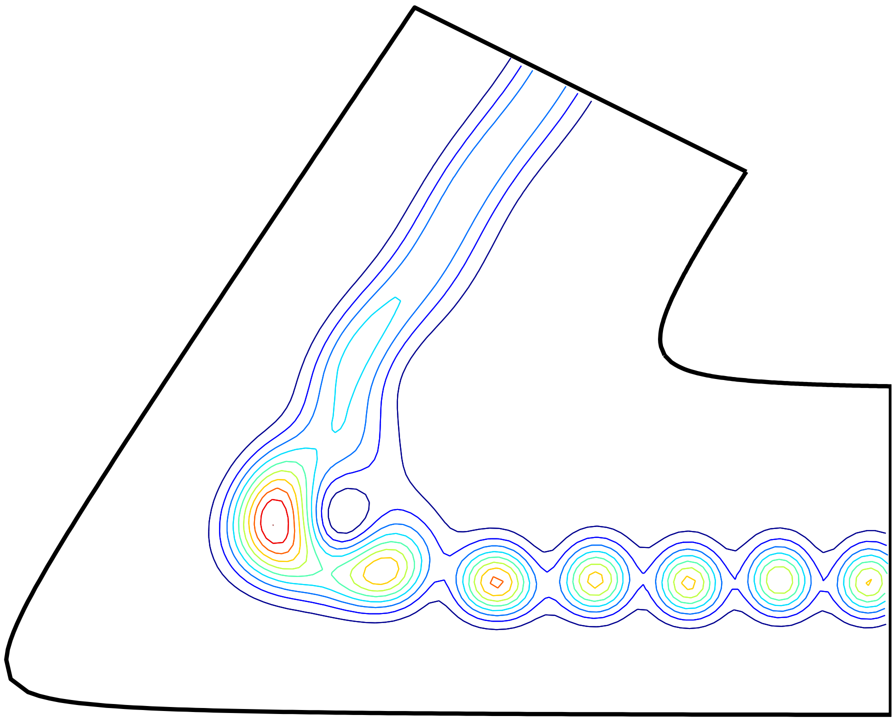
COLSCAT2D

Version 2.0

Reference Manual

Michael Warehime with contributions from Millard H. Alexander & Dvir Kafri



```
mh = 1836.15264;     % mass of hydrogen in au
m = [mh,mh,mh];      % H + H2
E = 0.03;            % scattering energy (hartree)

% generate mesh
[p,t,pf,bnd] = meshgen(m,[6 6],[.25 9 .25 9],[.1 .6],0.2,'vhh2');

% scattering calculation
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,0,1,'vhh2',4);


% plot probability density
hold on;
tricontour(pn,tln,abs(psi{1}(:,1)).^2,10);
plot(bnd(:,1),bnd(:,2),'linewidth',2,'color','k');
axis([1 6 0.4 5]);
axis off
```

Contents

1 Introduction

1.1 What is COLSCAT2D ?

COLSCAT2D or COLlinear atom-diatom SCATtering is a MATLAB script to use the finite element method to solve numerically quantum scattering dynamics for collinear atom-diatom systems. The suite of subroutines provided can be used for any A+BC→AB+C collinear reaction.

1.2 Download and Installation

The source code is available for download from http://www2.chem.umd.edu/groups/alexander/FEM. To ensure that all subroutines are accessible add the **COL**SCAT2D folder to the MATLAB search path. In release 2013 of MATLAB the search path is accessed on the 'HOME' tab, in the 'ENVIRONMENT' field. Be sure to select the 'add with subfolders' option and then browse for the location of your local **COL**SCAT2D directory.

1.3 Related Works

The **COL**SCAT2D reference manual complements the published work of Mick Warehime and Millard Alexander. [**?**, **?**] The descriptions of many of the subroutines in **COL**SCAT2D refer to this publication.

1.4 How to Read this Manual

The manual is broken into six parts:

i. Overview of the steps involved in a reactive scattering calculation (section 3)

ii. Introduction to the design and operation of the software (section 4).

iii. Tutorials on how to perform reactive scattering calculations (section 5).

iv. Description of parallelization procedures (section 6).

v. Brief explanation of the components of the **COL**SCAT2D package (section 7).

vi. Details of the functions/subroutines themselves, including the input and output call lists for each of the major subroutines, and, in many cases, examples of how to use the functions/subroutine. This last section is intended to facilitate the improvement and extension of **COL**SCAT2D .

The tutorials provide input files to calculate reactive scattering dynamics for several exemplary systems. Learning from these tutorials, the user can create inputs for new systems or modify (by isotopic substitution, for example) the prepared inputs.

Much of the first half of this manual shows how to take advantage of the powerful visualization capabilities contained in MATLAB. If you are interested in expanding the scope of **COL**SCAT2D , you should find enough details here to let you extend the existing code. Do not hesitate to contact MW for more information about any of the information provided in this manual.

The reader is assumed to have a working knowledge of MATLAB.

1.5 Authors and Acknowledgements

Mick Warehime developed the **COL**SCAT2D suite and documentation. Dvir Kafri provided much debugging help and insightful discussion. Millard Alexander provided encouragement and editorial assistance. The U. S. National Science Foundation financially underwrote the project.

2 Quickstart Guide

This section is designed for those users who aren't interested in the details of **COL**SCAT2D and want to just start calculating things. First we recommend running the pre-made inputs `hh2.m`, `fh2.m` , and `fhcl.m` to ensure that the **COL**SCAT2D directory has been successfully linked to MATLAB's search path. Once these three inputs have been tested you are ready to create your own input files.

Besides specifying the mass of each atom in atomic units and the desired scattering energies (total system energy in hartrees) there are two important things that need to be defined

1) potential energy function - this routine needs to have the exact input/output format as outlined below in subsection 7.28.

2) triangulation data - the outputs from `meshgen.m`, i.e. `p`, `t`, and `pf` from subsection 7.16. You can use a hand made mesh made up of nodal points `p` and connectivity matrix `t`. `colscat2d.m` Will be expecting `pf` as an input which define the limits of reactant and product boundary, as is shown in Figure 7.16.

With a working potential energy function and properly defined mesh you can easily run the `colsca2dt.m` subroutine to determine the scattering dynamics of your system. Consult the examples below on how to visualize the scattering probabilities (subsection 5.1) or the scattering wave function or probability density, its current, and curl (subsection 5.5).

3 Time-independent scattering

A scattering calculation in a time-independent framework, regardless of the method used, involves a few key steps: (Here we make reference, wherever possible, to our published paper [?])

- **Specification of the system**
  A (collinear) chemical reaction is defined by the masses of the atoms and by the potential energy surface. The masses of the atoms determines the transformation between the bond, Jacobi and mass-weighted Jacobi coordinate systems, as well as the angle of rotation between the Jacobi coordinates appropriate to the reactants and to the products. See Sec. IIA of [?].

  Subsequently, and most importantly, the reaction is controlled by a potential energy surface. Any user must provide a script to determine the PES for the system of interest in the directory `inputs`. Potential energy surfaces for the H+H_2, F+H_2, and F+HCl systems are provided already in the distribution.

  Finally, it is necessary to specify the grid of energies for the subsequent scattering calculations.

- **Specification of the physical domain and mesh**

  In the finite-element method the wave function is expanded in piecewise functions defined on a triangular A typical domain for reactive scattering in two dimensions, in shown in Fig. 1 of [?].

This is defined by the maximum values of the Jacobi A+BC and AB+C separation distances and the values of the potential energy which define the inner and outer boundaries.

With the physical domain fixed, by specifying the approximate size of triangles in the mesh and the connectivity, we create the triangular In practice this is done with a customized version of Persson and Strang's `distmesh`. [?]

- **Specification of the asymptotic vibrational states**

  In the generic A+BC$\rightarrow$AB+C reaction, atom A collides with molecule BC in vibrational level v_a. The outcome can be

  - elastic: A+BC(v)$\rightarrow$ A+BC(v)
  - non-reactive, but inelastic: A+BC(v)$\rightarrow$A+BC(v$'\neq$v)
  - reactive: A+BC(v)$\rightarrow$ AB(v$'$)+C

  The amplitude for each of these processes is obtained directly from the FEM calculation performed in **COL**SCAT2D . The vibrational wave functions must be obtained before carrying out the scattering calculation.

- **Scattering Calculation**

  At this point, we then loop over a vector of collision energies, to obtain the scattering amplitudes for elastic, inelastic, and reactive outcomes. This involves a number of steps:

  a. **Building the FEM matrix**
     On the triangulation mesh we need to determine the matrix representation of $\hat{T}+2\mu(\hat{V}-E)$. This is the **A** matrix defined in Eq. (26) of [?].

  b. **Extending the FEM matrix**
     Then, to accommodate the undefined boundary conditions of a scattering problem, we need to build the rectangular **B**, **F** and **I** matrices which appear in Eq. (30) of [?].

  c. **Building the inhomogenous part**
     Finally, we need to build the right-hand-side vector, **b**, (or triangular right-hand-side matrix) in Eq. (30) of [?].

  d. **Solving the linear equations**
     The wave function and the S-Matrix are solutions to a set of linear equations [Eq. (30) of [?]. To solve these, we make use of MATLAB's powerful backslash (\) superoperator [?].

- **Post-processing**

  After repeating steps (a)–(d) at each desired energy, the script then post-processes the data generated to extract the scattering amplitudes and the wave functions. One can also generate the probability density current field and its curl.

4 How COLSCAT2D Works

In this section we provide a step by step analysis of how `colscat2d.m` works. We show the code in its entirety to facilitate this discussion.

```
colscat.m:  main driver script

 1  % COLlinear atom-diatom SCATtering
 2  function [n,r,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,iState,iSurf,vinput,n)
 3  % add points to account for the polynomial order n
 4  [pn,tn,tln,np] = polymesh(p,t,n);
 5
 6  %% determine the points on the boundary of the mesh
 7  [ba,bc,pn] = boundaryind(pn,pf);
 8
 9  %% paramater structure (reduced masses and transformation parameters)
10  M = mass(m);
11
12  %% potential energy function (in terms of bond coordinates)
13  pb = tobond(pn,m);
14  [v,va,vc,ns] = feval(vinput,pb(:,1),pb(:,2),ba,bc);
15
16  % boundary unscaled jacobi coordinates
17  [Ra,ra,Rc,rc] = jacobi(pn,ba,bc,M);
18
19  % boundary vibrational functions
20  viba = cell(ns,1); vibc = cell(ns,1);
21  for ii=1:ns
22      viba{ii} = vibfem(ra,va(:,ii),M.mubc,n);
23      vibc{ii} = vibfem(rc,vc(:,ii),M.muab,n);
24  end
25
26  %% FEM MATRICES (2nd order 2D)
27  [T,V,O] = fem2d(pn,tn,v,n);
28
29  %% Scattering probabilities as a function of energy
30  le = length(E); U = cell(le,1);
31  parfor jj = 1:le
32      U{jj} = solver2d(E(jj),T,V,O,M,viba,vibc,iState,iSurf,ba,bc,np,ns,Ra,Rc);
33  end
34
35  % post process the parallel data
36  [n,r,psi] = nrsort2d(U,le,np,ns,iState,max(E),viba,vibc);
```

line 2 - User Defined Inputs
There are nine user input parameters; the mass vector `m = [`m_a`, `m_b`, `m_c`]` in atomic units, the vector containing the scattering energies (total energy), `E`, in hartrees, the list of nodal points, `p`, the connectivity matrix, `t`, the points at the corner of each boundary, `pf`, the initial vibrational state of the reactants, `iState`, the initial electronic surface of the system `iSurf` (for single electronic surface calculations this should always be 1), the potential energy subroutine function handle `vinput` and finally the polynomial order of the FEM basis functions `n`. We give three sample input files: `COLSCAT/inputs/hh2/hh2.m`, `COLSCAT/inputs/fh2/fh2.m` and `COLSCAT/inputs/fhcl/fhcl.m`.

The most important inputs are the mesh, defined by the nodal points `p` and the connectivity matrix `t`, and the specification of the potential energy function, which is defined by a reference to a function handle in the string `vinput`. In subsection 7.16 and subsection 7.28 we provide more details on how to prepare these inputs.

line 4 - Polymesh
COLSCAT2D allows you to perform a FEM calculation using polynomial basis functions up to order 5. In order to do this we have to add additional nodes, which are stored in `pn`, the new connectivity matrix `tn`, and a linearized form of this same connectivity matrix `tl` which can be used to plot the wave function.

line 7 - Boundary
Once the additional nodes have been added to the mesh we need to know which nodal points lie on the boundary. This is accomplished via the `boundaryind.m` subroutine. The `boundary` subroutine also 'straightens' the boundaries and accordingly returns the nodal points as an output. This straightening is required because the mesh generation routine of Persson and Strang [**?**] does not fix the boundaries to a single line. This subroutine works by first finding all points near the readtant and product boundaries within some tolerance and then fixes them to the straight line defined between the points in `pf`.

line 10 - Reduced Masses/Transformation Parameters
Given the user defined inputs, **COL**SCAT2D first calculates the relevant reduced mass parameters and the transformation parameters. These transformation parameters include the λ scale factors for the transformation to mass-weighted Jacobi coordinates and the skew angle between the reactant and product coordinate systems. These parameters are described in Sec. 2 of the original paper [**?**] and here in subsection 7.12.

line 13-14 - Potential Energy Surface
Next **COL**SCAT2D invokes the potential energy input function (described in Section subsection 7.28) to determine the PES at the nodal points. Note `vinput` takes values in terms of bond coordinates, i.e. `pb`, rather than mass-scaled Jacobi coordinates, i.e. `p`, because it is typical that the potential energy surface is known in terms of these bond coordinates. This output of the potential energy input function must also provide the potential energy along the reactants boundary, `va`, and the products boundary, `vc`, which are used in the next step. This subroutine also returns the number of electronic surfaces used in the calculation. Again, for now this parameter, `ns` is always 1.

lines 18-25 - Boundary Vibrational Structure
The `jacobi.m` subroutine returns the unscaled diatomic Jacobi coordinates for the reactant arrangement (the BC diatomic separation coordinate, `ra`, as well for the product arrangement (the AB diatomic separation coordinate, `rc`. These unscaled Jacobi coordinates are used to evaluate the boundary vibrational wave functions, χ_γ, in Eq. (12) of the original paper. [**?**]

Next these vibrational wave functions on both the reactants, χ_a, and products, χ_a, boundaries are calculated. To determine the boundary vibrational wave functions for the reactant diatomic, for example, one passes the reactant unscaled Jacobi diatomic separation coordinate, `ra`, the potential energy surface evaluated at the reactant nodes, `va`, and the reduced mass of the reactant diatom, `M.mubc`, to the `vibfem.m` subroutine. This routine uses a 1-dimensional, PN (N-th order polynomial hat functions) finite element code to calculate one bound-state wave function for each nodal point on the given boundary, the energy associated with each of these bound state wave functions.

The `vibfem.m` subroutine uses the bound state wave functions as well as the overlap matrix from the 1D PN FEM calculation to determine the boundary integrals, i.e. the integral appearing in Eq. (44) of the original paper. [?] Furthermore, the `for` loop found is part of **COL**SCAT2D's ability to perform calculations on multiple, coupled electronic surfaces. This will be discussed in a future update.

line 28 - FE Matrices
With values of the potential energy surface at the nodal points, **COL**SCAT2D determines the finite-element matrices, **V**, **T** and **O** [Eqs. (15-17) in the original paper [?]]. The values of the integrals themselves are linear combinations of the values of the mass-scaled Jacobi coordinates of each nodal point **T** and **O**, and in the case of the potential matrix **V**, linear combinations of the PES at the nodal points.

The are five scripts `PNintegrals2d.m`, where N = 1:5, (**??**) show how the coefficients of these linear combinations are determined, (integrals of the PN basis functions in standard triangle coordinates), once and hard-coded for every scattering calculation. This hard-coding allows the FEM matrices to be built using MATLAB's `sparse.m` subroutine, i.e. a vectorized sparse matrix construction operation.

lines 31-34 - Solver Subroutine
The next step is to call the main solver routine. Directly outside of the call to the `solver2d.m` subroutine is a `for` loop over all values of the total energy. This is an input vector, in the form $\mathtt{E} = \mathtt{E}_1 : \Delta\mathtt{E} : \mathtt{E}_2$. Scattering calculations are done at all energies from E_1 to E_2 in steps of ΔE. If the MATLAB Parallel Computing Toolbox (PCT) is available, replace `for` by `parfor`. This replacement instructs MATLAB to parallelize this loop over energy transparently, requiring no input from the user. In section 6 we provide more information about the implementation of parallelization.

Again to facilitate discussion of this subroutine we include the MATLAB script `solver2d.m` in its entirety.

`solver2d.m` extends the FEM problem and solves the scattering problem

```
function U = solver2d(E,T,V,O,M,viba,vibc,iState,iSurf,ba,bc,np,ns)

% account for matlabs nonzero indexing v=[0,1,2,3] --> v=[1,2,3,4]
iState = iState+1;

% define the A matrix (Eq (32))
A = T+2*M.mu*(V-E*O);

% get the extension submatrices
[B ,F ,I] = extendmat2d(viba,vibc,ba,bc,M,E,np,ns);

% the r.h.s. for all initial states
b = extendrhs2d(viba,ba,bc,iState,iSurf,M,E,np,ns);

% extend the FEM system
Q = [A, -B; I, -F];

% solve linear system (Eq (30))
U = Q\b;
```

line 4
The first step in the `solver2d.m` subroutine is add 1 to the `iState` variable. This is because MATLAB's indexing starts from 1, and quantum numbers, such as the vibrational quantum index, v, start at 0.

line 7
Next the unextended FEM matrix, `A`, is constructed. This makes up the left hand side of Schrödinger's equation. *It should be noted that 2D bound state problems can be solved by determining the eigenvalues of the* **A** *matrix, however we only consider reactive potential surfaces in this work.*

line 10
The next step is to extend the FEM matrix problem, i.e. implement Eq. 30 in the referenced paper [?]. To do this the vibrational structures, initial state and initial electronic surface vectors as well some descriptive parameters are passed to the `extendmat2d.m` subroutine.

The `extendmat2d.m` subroutine is one of the most dense and opaque subroutines provided in the **COL**SCAT2D package. In summary this subroutine calculates the outgoing integrated boundary conditions on the reactant and product boundaries, (`Ba` and `Bc`), and standard outgoing boundary conditions on the reactant product boundaries, (`Fa` and `Fc`). Next `extendmat2d.m` uses the boundary indices, `ba` and `bc`, and the built-in `sparse.m` routine to build the **B**, **I** and **F** matrices.

line 13
Once the FEM matrix extensions have been calculated, `solver2d.m` calculates the r.h.s. of Eq. (30), using `extendrhs2d.m`. This subroutine calculates the incoming integrated and standard boundary conditions, `bi`, and `fi`, and builds the sparse components the r.h.s. of the extended linear system referred to above and stores this in the `b` matrix in Eq. 30.

line 16
The `solver2d.m` routine then builds the extended FEM matrix, `Q`, and

line 19
finally `solver2d.m` uses MATLAB's '\' superoperator [?] to solve the linear system `QU` = `b`. It is called a superoperator because it checks the size (square vs rectangular) and type (sparse vs full) of the matrix that it tries to invert, in this case `Q`, and chooses the appropriate algorithm. MATLAB's built-in matrix division is very impressive and furthermore incredibly simple to implement. The curious user may try `full(Q)\full(b)` instead of the call on line 25 of `solver2d.m` to see just how fast and accurate the sparse matrix division algorithm hidden in the '\' operator really is.

It should be noted that the information at the i^{th} scattering energy, namely the decomposition of the 'Q' matrix, could in theory be passed to the solver to precondition the solution at the $i+1^{th}$ energy. The speeds we observe are so fast that this has not been necessary so far, however, it may be a potential source of optimization in the future.

line 37 - Parse the Parallel Data
The outputs of MATLAB's `parfor` loops must be stored in cell type variables. In this case the `parfor` loop is over energy values, and accordingly, the data is indexed by energy. Ultimately, `nrsort2d.m` changes the data storage from indexed by energy to indexed by electronic surface and initial conditions which greatly simplifies plotting the data.

5 COLSCAT Examples

5.1 Tutorial A: H+H_2 Scattering Probabilities

This example treats collinear H+H_2 scattering dynamics on a single potential energy surface. The following input file generates both non-reactive and reactive probabilities as a function of total energy. The potential energy surface provided here for this system is that of Mielke *et al.* [?], which is calculated by the script `vhh2.m`. As part of the input file, we pass the reference name '`vhh2`' to the desired potential energy surface. More details on constructing the script for the potential energy surface are given in subsection 7.28.

this section constitutes the first bullet item in Sec. 3

```
% symmetric scattering example H+H2
mh = 1836.15264;        % mass of hydrogen in au
m  = [mh,mh,mh];        % [H,H,H]
E = 0.01:0.0001:0.06;   % energy range
iState = 0;             % initial vibrational state of reactants

%% calulate mesh for HH2 problem
[p,t,pf] = meshgen(m,[6 6],[.25 9 .25 9],[.1 .6],0.3,'vhh2');

%%  main scattering calculation using Polynomial order 4 basis functions, P4
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,iState,1,'vhh2',4);

% collision energy in eV
ecol = (E-viba{1}.e(1))*27.211;

figure(1)
plot(ecol,N{1})
legend('0-0','0-1','0-2')
title('Non-Reactive Probabilities for H + H_2(v=0)','fontsize',24)
set(gca,'fontsize',20)
axis([ecol([1 end]) 0 1])
set(gca,'xtick',0:.25:1.5)
xlabel('Collision Energy, eV')

figure(2)
plot(ecol,R{1})
legend('0-0','0-1','0-2')
title('Reactive Probabilities for H + H_2(v=0)','fontsize',24)
set(gca,'fontsize',20)
axis([ecol([1 end]) 0 1])
set(gca,'xtick',0:.25:1.5)
xlabel('Collision Energy, eV')
```

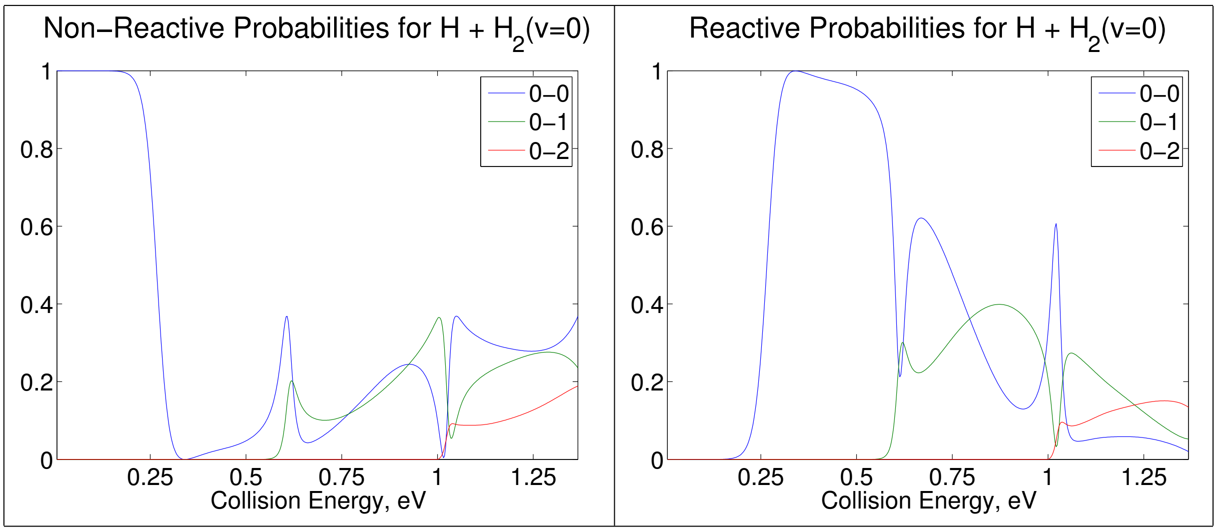


Figure 1: (Left) Plotted output from the script subsection 5.3, combining MATLAB figures (1) and (2). The probability to not react $H+H_2(v=0) \rightarrow H+H_2(v=0,1,2)$ as a function of collision energy. (right) The probability to react as a function of collision energy for $H+H_2(v=0) \rightarrow H_2(v=0,1,2) + H$.

5.2 Tutorial B: Vibrationally Excited States

The input variable 'iState' allows the user to change the vibrational state of the reactant diatomic species. In the previous calculation the input parameter 'iState=0' was used to investigate the dynamics of the reaction $H+H_2(v=0)$. By simply changing this value to 'iState=1' study reaction of the vibrationally excited state $H+H_2(v=1)$. The results of this calculation are shown below.

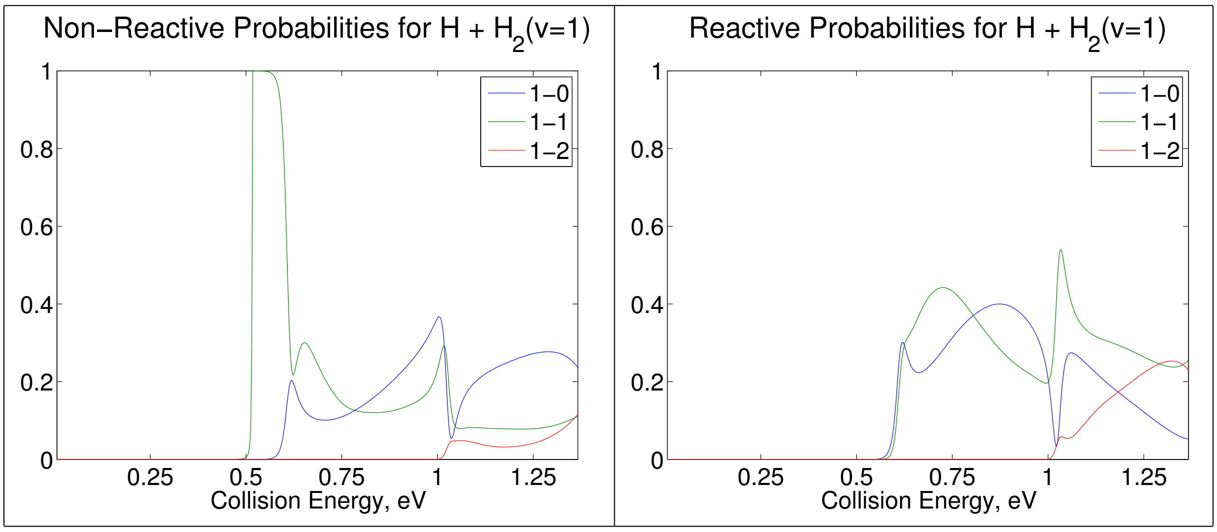


Figure 2: Vibrationally Excited Reactants. (left) Inelastic scattering probabilities, i.e. non-reactive probabilities, for the reaction of $H+H_2(v=1) \rightarrow H+H_2(v=0,1,2)$ as a function of collision energy. (right) Reactive scattering probabilities for this same vibrationally excited system.

5.3 Tutorial C: Multiple Initial Conditions

Not only is it easy to experiment with vibrationally excited reactants, **COL**SCAT2D makes it very easy to calculate the entire S-matrix in a single calculation. By changing the parameter `'iState'` from a scalar to a vector, you can sample multiple initial conditions in a single calculation. This is particularly convenient for two reasons: a) it speeds up the total calculation time because the matrix decomposition, which is the slowest part of the code, is only done once for multiple r.h.s.'s and b) the data for multiple initial conditions is stored and handled efficiently for post processing. Comparing the input prepared below with that above in subsection 5.1 reveals how easy it is to calculate the scattering dynamics of multiple initial conditions simultaneously.

```
% symmetric scattering example H+H2
mh = 1836.15264;       % mass of hydrogen in au
m  = [mh,mh,mh];       % [H,H,H]
E = 0.01:0.0001:0.06;  % energy range
iState = 1:2;            % initial vibrational state of reactants
iSurf = [1 1];            % initial electronic surfaces (for 1 state both entries must be 1)
%% calulate mesh for HH2 problem
[p,t,pf] = meshgen(m,[6 6],[.25 9 .25 9],[.1 .6],0.3,'vhh2');

%%  main scattering calculation using Polynomial order 4 basis functions, P4
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,iState,iSurf,'vhh2',4);

% collision energy in eV
ecol = (E-viba{1}.e(1))*27.211;

figure(1)
plot(ecol,R{1})
legend('1-0','1-1','1-2')
title('Reactive Probabilities for H + H_2(v=1)','fontsize',24)
set(gca,'fontsize',20)
axis([ecol([1 end]) 0 1])
set(gca,'xtick',0:.25:1.5)
xlabel('Collision Energy, eV')

figure(2)
plot(ecol,R{2})
legend('2-0','2-1','2-2')
title('Reactive Probabilities for H + H_2(v=2)','fontsize',24)
set(gca,'fontsize',20)
axis([ecol([1 end]) 0 1])
set(gca,'xtick',0:.25:1.5)
xlabel('Collision Energy, eV')
```

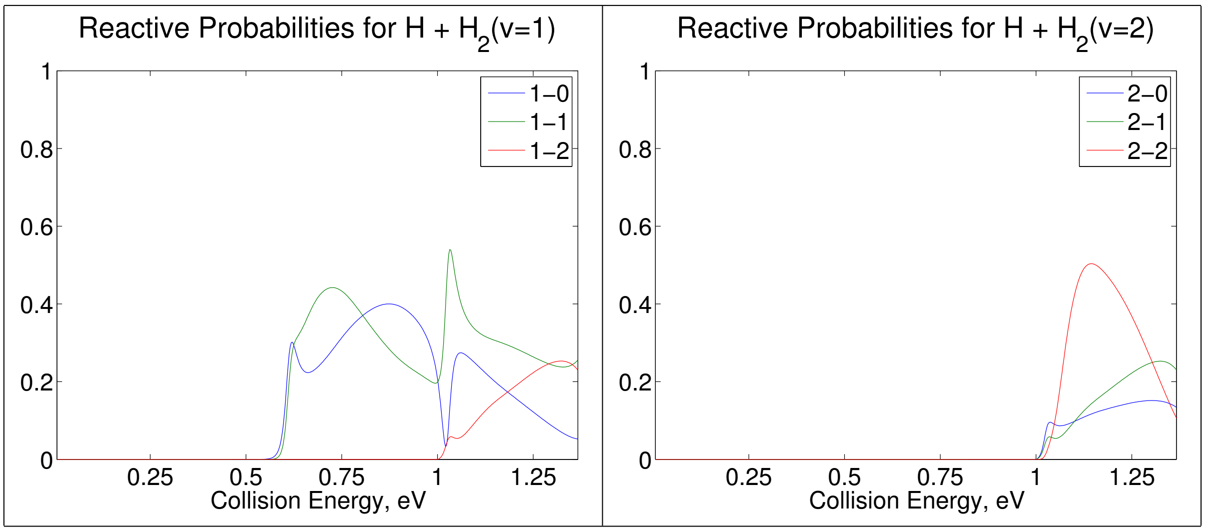


Figure 3: (left) Reactive scattering probabilities, i.e. non-reactive probabilities, for the reaction of $H+H_2(v=1) \rightarrow H+H_2(v=0,1,2)$ as a function of collision energy. (right) Reactive scattering probabilities for the $H+H_2(v=2) \rightarrow H+H_2(v=0,1,2)$ system as a function of collision energy.

5.4 Tutorial D: H+DH – Isotopic Effects

COLSCAT2D makes it incredibly easy to analyze isotopic effects for collinear atom-diatom reactive scattering dynamics. The following input code calculates the scattering dynamics for H+DH using almost exactly the same input file as the $H+H_2$ system. The only difference between the H+DH input and the $H+H_2$ input file are the factor of two in the definition of the mass vector `m`. Because the potential energy does not depend on the masses of the involved nuclei, only their charge, we can use the same potential energy routine as we did for the $H+H_2$ calculation.

```
% Scattering Dynamics of H + DH
mh = 1836.15264;     % mass of hydrogen in au
m  = [mh,2*mh,mh];     % [H,D,H]
E = 0.01:0.0001:0.06; % energy range


%% calulate mesh for HDH problem
[p,t,pf] = meshgen(m,[6 6],[.25 9 .25 9],[.1 .6],0.3,'vhh2');

%%  main scattering calculation
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,0,1,'vhh2',4);

% collision energy in eV
ecol = (E-viba{1}.e(1))*27.211;

% plot reflection coefficients for first three vibrational states
figure(1)
plot(ecol,N{1})
legend('0-0','0-1','0-2')
title('Non-reactive Probabilities for H + DH(v=0)','fontsize',24)
set(gca,'fontsize',20)
axis([ecol([1 end]) 0 1])
```

```
set(gca,'xtick',0:.25:1.5)
xlabel('Collision Energy, eV')

% plot reflection coefficients for first three vibrational states
figure(2)
plot(ecol,R{1})
legend('0-0','0-1','0-2')
title('Reactive Probabilities for H + DH(v=0)','fontsize',24)
set(gca,'fontsize',20)
axis([ecol([1 end]) 0 1])
set(gca,'xtick',0:.25:1.5)
xlabel('Collision Energy, eV')
```

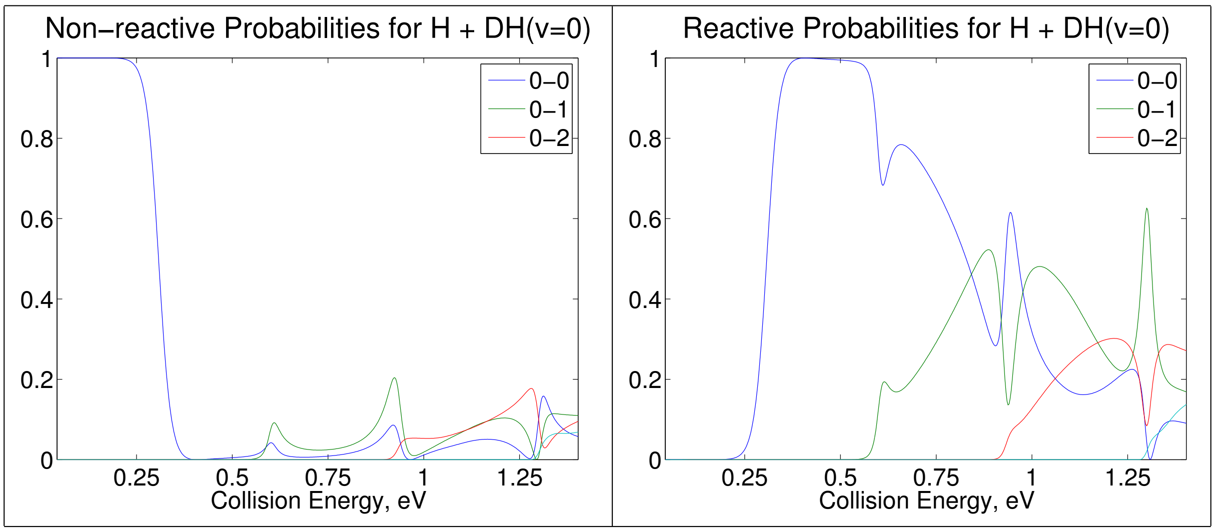


Figure 4: (left) Non-reactive (inelastic) H+DH(v=0) → H+DH(v=0,1,2) probabilities as a function of collision energy. (right) Reactive H+DH(v=0) → HD(v=0,1,2)+H probabilities as a function of collision energy.

5.5 Tutorial E: F+H_2 – Probability Density, Density Current and its Curl

Another convenient aspect of the MATLAB implementation of **COL**SCAT2D is the ease with which one can visualize the quantum nature of the scattering. As in the previous two examples we will use a single state scattering calculation, F+H_2, to illustrate the various built-in visualization capabilities. Shown below, is the input file for the F+H_2 reaction. Here, we have translated the Muckerman V PES [?] to a fast and simple MATLAB script `fh2_muck`. The following script produces figures of the reactive probabilities as a function of collision energy, the wave function for a given collision energy and the probability density current field and the curl of this vector field.

```
clear all; close all; clc;
% reactive scattering of F+H2
mh = 1836.15264;          % mass of hydrogen in au
m  = mh*[19/1.008 1 1];    % [F,H,H]
E = 0.0099:0.0001:0.0282; % energy range

% calculate mesh with approximately
[p,t,pf] = meshgen(m,[8.5 8],[.8 9 .3 9],[.12 .4],0.2,'vfh2_MV');

% determine scattering probabilities and wave functions
tic
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,0,1,'vfh2_MV',3);
toc

% collision energy in eV
ecol = (E-viba{1}.e(1))*27.211;

ll = 73; % pick a scattering energy to analyze in figures 2-4

figure(1);
hold all;
plot(ecol,[R{1}(:,2)*100 R{1}(:,3:4)]);
plot(ecol([1 1]),[0 1],'k','linewidth',2);
title('Reactive Probabilities for F + H_2(v=0)','fontsize',24);
set(gca,'fontsize',20);
xlabel('Collision Energy, eV');
legend('0-1(X100)','0-2','0-3','location','north');
axis([ecol([1 end]) 0 1]);

figure(2)
tricontour(pn,tln,abs(psi{1}(:,ll)).^2,25);
text(7,4.75,'FH+H','fontsize',18);
text(8,0.4,'F+H_2','fontsize',18);
title('Probability Density E_{col} = 0.197 eV','fontsize',24);
set(gca,'fontsize',20);
axis off;


figure(3);
hold on
axis([3 9 0.5 5])
% classically forbidden region contour
tricontour(pn,tln,v,[E(ll) E(ll)]);
colormap([1,0,0]);
% current density
[Jx,Jy] = flux(tn,pn,psi{1}(:,ll),m);
% plot the prob current only at the original mesh (too many pts otherwise)
```

```
x = p(:,1);y = p(:,2);
quiver(x(t(:)),y(t(:)),Jx(t(:)),Jy(t(:)));
title('Probability Density Current, E_{col} = 0.197 eV','fontsize',24);
text(7,4.75,'FH+H','fontsize',18);
text(8,0.4,'F+H_2','fontsize',18);
set(gca,'fontsize',20);
axis off;

figure(4);
curlZ = tricurl(tn,pn,psi{1}(:,11),m);
tricontour(pn,tln,curlZ,15);
text(7,4.75,'FH+H','fontsize',18);
text(8,0.4,'F+H_2','fontsize',18);
title('Curl Density Current E_{col} = 0.197 eV','fontsize',24);
axis off;
```

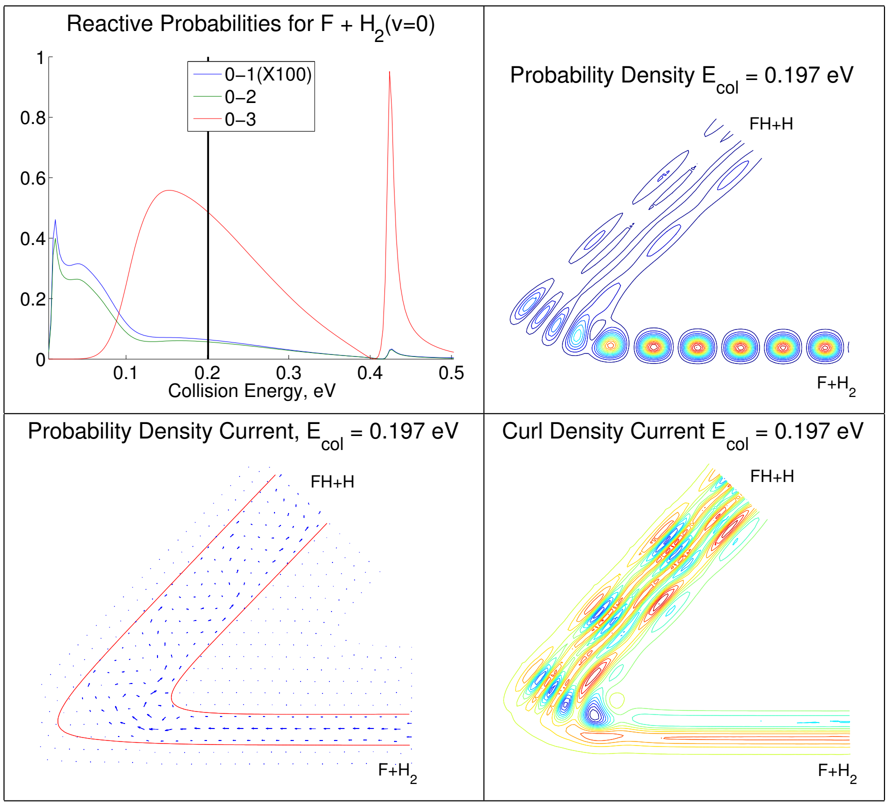


Figure 5: (top left) F+H_2(v=0) $\rightarrow$ H+FH(v=1,2,3) reaction probabilities as a function of collision energy. The 0$\rightarrow$1 probability is multiplied by a factor of 100 for clarity. The vertical black line marks a collision energy of 0.197 eV, to which the three remaining panels refer. (top right) The probability density, $|\Psi|^2$, for the reaction of H_2(v=0); (bottom left) the current density density field, the red lines delimit the classically forbidden region; (bottom right) the vector curl of the current density density field.

5.6 Tutorial F: F+HCl – Two State scattering

In this tutorial we outline how to use **COL**SCAT2D for coupled potential calculations. To start we analyze the potential energy surface subroutine for the two state F+HCl reactive scattering calculation.

```
function [vnode,va,vc,ns] = vfhcl_f12(x,y,ba,bc)

% get the sigma and pi potentials from interpolated ab initio data
% the spin-orbit data is calculated from a fit to ab initio data
[vsig,vpi,vso] = vfhcl_fast(x,y);

% OMEGA BASIS
D = [2/3,  sqrt(2)/3,  sqrt(2)/3, 1/3;    % vsig
    1/3, -sqrt(2)/3, -sqrt(2)/3, 2/3;     % vpi
    -1,        0,          0,     2];     % vso

vnode = [vsig vpi vso]*D;

ns = 2;
if nargout==4
    va = vnode(ba,[1 4]);
    vc = vnode(bc,[1 4]);
end
```

The first step of this routine is to interpolate the ab initio data of the Σ, Π and spin orbit potential surfaces. The `vfhcl_fast.m` subroutine also performs an asymptotic dampening of these potentials under the hood in order to set $V_\Sigma = V_\Pi$ asymptotically and to set the the minimum of the potential to HCl(r_e) at $R_a = 24$.

Once the potential in the Λ basis is loaded it is transformed to the j_a basis everywhere in space. The `[vsig vpi vso]` matrix has dimension `np`×`np`, where `np` is the number of points in the triangulation (output from polymesh subsection 7.20). Multiplying this by the `D` matrix, which has dimension 3×4, gives the vnode matrix of dimension `nt`×4. At each of the nodal points the potential is represented by a 2D matrix and it is more convenient to store the points as follows

$$\texttt{vnode}(i,:) = [v_{11}(i), \quad v_{12}(i), \quad v_{21}(i), \quad v_{22}(i)]$$

where

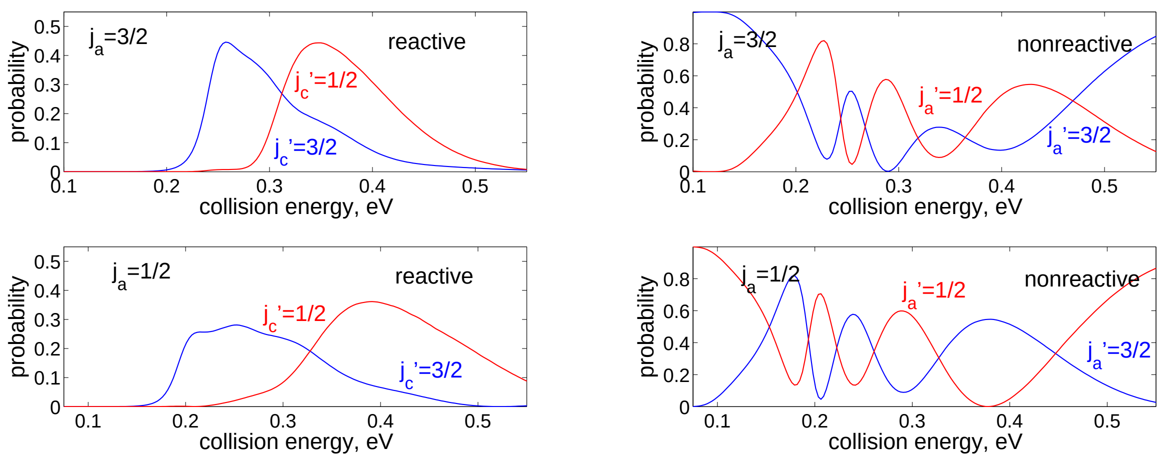
$$V^j(R_a(i), r_a(i)) = \left[\begin{array}{cc} v_{11}(i) & v_{12}(i) \\ v_{21}(i) & v_{22}(i) \end{array}\right]$$

Finally the `ns` variable tells **COL**SCAT2D that two states will be used in this calculation and the potential energy surfaces v_{11} and v_{22} are given along the reactant and product boundaries. These asymptotic potentials are used to calculated the vibrational states on the boundaries. *note: the potential is set to zero within* `vfhcl_fast` *and is therefore not zeroed again in this subroutine.*
The following snippet of code shows how to use this two-state potential surface to perform the reactive scattering calculations.

```matlab
clear all, close all, clc
% reactive scattering of F+HCl on klos PES
global m

me = 1822.88862599;              % electron mass
m = [18.998, 1.008, 35.453]*me;  % mass of [F,H,Cl] in amu
E = 0.0063:.0001:.06;            % energy range hartree

% generate mesh

 [p,t,pf,bnd] = meshgen(m,[24 24],[.6 7 1.2 7],[.1 1],0.15,'vfhcl_f12');

%% Call main Szcattering Routine
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,[0 0],[1 2],'vfhcl_f12',5);

%% initial state ja=3/2
% collision energy
econv = 27.211;
ecol32 = (E-viba{1}.e(1))*econv;
% reactive probabilities
figure(1)
hold on
plot(ecol32,R{1,1}(:,4),'color','b',ecol32,R{2,1}(:,4),'color','r','linewidth',1)

set(gca,'fontsize',18,'ytick',0:.1:.5,'xtick',0:.1:.55)
axis([.1 .55 0 .55]);
box on
text(.125,.45,'j_a=3/2','fontsize',21)
text(.415,.45,'reactive','fontsize',21)
text(.305,.07,'j_c''=3/2','fontsize',21,'color','b')
text(.325,.3,'j_c''=1/2','fontsize',21,'color','r')
xlabel('collision energy, eV','fontsize',21)
ylabel('probability','fontsize',21)
set(gcf,'position',[0 500 560 210])

% nonreactive probabilities
figure(2)
hold on
plot(ecol32,N{1,1}(:,1),'color','b','linewidth',ecol32,N{2,1}(:,1),'color','r','linewidth',1);

set(gca,'fontsize',18,'ytick',0:.2:.8,'xtick',0:.1:.55)
axis([.1 0.55 0 1]);
box on
text(.415,.8,'nonreactive','fontsize',21)
text(.445,.2,'j_a''=3/2','fontsize',21,'color','b')
text(.32,.45,'j_a''=1/2','fontsize',21,'color','r')
text(.125,.8,'j_a=3/2','fontsize',21)
xlabel('collision energy, eV','fontsize',21)
ylabel('probability','fontsize',21)
set(gcf,'position',[600 500 560 210])

%% initial state ja = 1/2
ecol12 = (E-viba{2}.e(1))*econv;

% reactive probabilities
figure(3)
hold on
plot(ecol12,R{1,2}(:,4),'color','b',ecol12,R{2,2}(:,4),'color','r','linewidth',1)
```

```
set(gca,'fontsize',18,'ytick',0:.1:.5,'xtick',0:.1:.55)
axis([0.075 .55 0 .55]);
text(.125,.45,'j_a=1/2','fontsize',21)
text(.415,.45,'reactive','fontsize',21)
text(.42,.11,'j_c''=3/2','fontsize',21,'color','b')
text(.28,.305,'j_c''=1/2','fontsize',21,'color','r')
xlabel('collision energy, eV','fontsize',21)
ylabel('probability','fontsize',21)
set(gcf,'position',[0 150 560 210])

% nonreactive probabilities
figure(4)
hold on
plot(ecol12,N{1,2}(:,1),'color','b',ecol12,N{2,2}(:,1),'color','r','linewidth',1);

set(gca,'fontsize',18,'ytick',0:.2:.8,'xtick',0:.1:.55)
axis([0.075 0.55 0 1]);
text(.125,.8,'j_a=1/2','fontsize',21)
text(.415,.8,'nonreactive','fontsize',21)
text(.48,.325,'j_a''=3/2','fontsize',21,'color','b')
text(.29,.7,'j_a''=1/2','fontsize',21,'color','r')
xlabel('collision energy, eV','fontsize',21)
ylabel('probability','fontsize',21)
set(gcf,'position',[600 150 560 210])
```

This script simultaneous evaluates the reactive scattering of F($j_a = 3/2$) + HCl($v_a = 0$) and F($j_a = 1/2$) + HCl($v_a = 0$). The following figures are produced when this script runs. These four figures illustrates how **COL**SCAT2D stores the results of multistate scattering, namely `R{i,j}`(:,k) is the reactive scattering probability from the initial state corresponding to the A(j_a = `iSurf`(i))+ BC(v$_a$ = `iState(i)`) $\rightarrow$ AB(v$_c{}'= k$) + C($j_c{}' = j$).



5.7 Tutorial G: F+H_2 – Two State scattering with Mixed Boundary Conditions

The following script outlines a two-state calculation where the initial conditions are given in a different basis than the actual calculation.

```
clear all, close all, clc
% reactive scattering of F+H2
global m
mh = 1836.15264;          % mass of hydrogen in au
m  = mh*[19/1.008 1 1];   % [F,H,H]
E = 0.0117;                % energy range

% transformation of reactant Da and product Dc channels
Da =[sqrt(2)*1i 1; -1i sqrt(2)]/sqrt(3);
Dc = eye(2);

% calculate mesh
[p,t,pf,bnd] = meshgen(m,[10.5 10],[1 9 .5 9],[.15 .265],0.15,'vfh2_multi_nan');

% determine scattering probabilities and wave functions
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d_clebsch(m,E,p,t,pf,[ 0 0 ],[1 2],'vfh2_multi',4,Da,Dc);

%% calcualte the flux on the adiabatic surfaces
% mesh grid of interpolated range of interest
box = [3.2 10 .5 1.9];
h = 0.05;
[x,y] = meshgrid(box(1):h:box(2),box(3):h:box(4));

% interpolate wave function on both adiabats to mesh grid
Fpsi1 = scatteredInterpolant(pn(:,1),pn(:,2),psi{1,1});
Fpsi2 = scatteredInterpolant(pn(:,1),pn(:,2),psi{2,1});
psi1 = reshape(Fpsi1(x(:),y(:)),size(x));
psi2 = reshape(Fpsi2(x(:),y(:)),size(y));

% calculate adiabatic potentials at bond coordinates
pb = tobond([x(:) y(:)],m);
[vsig,vpi,vso] = vfh2_sigpi(pb(:,1),pb(:,2));
[a1,a2,theta] = adiabats(vsig,vpi,vso);

a1 = reshape(a1,size(x));
theta = reshape(theta,size(x));
vsig = reshape(vsig,size(x));
vpi = reshape(vpi,size(x));
vso = reshape(vso,size(x));
vdif = vsig-vpi;

[Jx1,Jy1,Jx2,Jy2] = meshflux(psi1,psi2,theta,m,h,h);
[Jxx,~] = gradient(Jx1,h,h);
[~,Jyy] = gradient(Jy1,h,h);
div = Jxx+Jyy;

figure(1)
hold on
contour(x,y,div,[-6:.3:-.3 .3:.3:6])
contour(x,y,a1,[E E],'r')
contour(x,y,vdif,[0 0],'k--')
contour(x,y,abs(vdif)-3*vso,[0 0],'k')
axis([3. 10 .7 1.7])
```

```
axis off
text(6,1.6,'\nabla x J^{a1}','fontsize',21)
set(gcf,'position',[150 150 375 280])
```

Evaluating this script yields the following figure, which shows the divergence of the current density on the lower adiabatic surface.

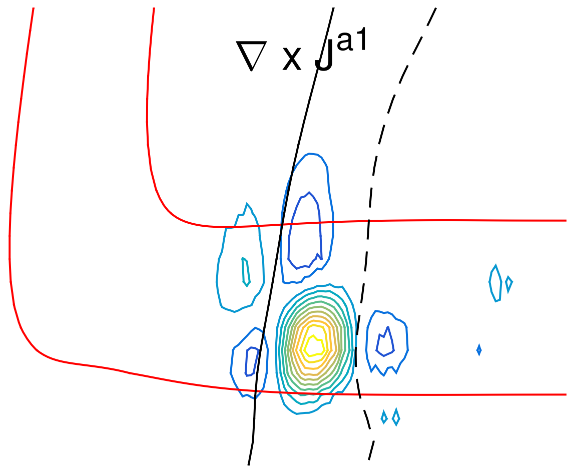


Figure 6: The divergence of the current density on the lower adiabatic surface for the reaction F(j_a = 3/2) + H_2(v_a=0). The solid red line shows the classically forbidden region of space. The dashed black line shows the seam where $V_\Sigma = V_\Pi$. The solid black line shows the seam where $V_{diff} = 3V_{SO}$.

This script can easily be modified to determine the divergence of the flux from the upper adiabatic surface by using the following two lines

```
% interpolate wave function on both adiabats to mesh grid
Fpsi1 = scatteredInterpolant(pn(:,1),pn(:,2),psi{1,2});
Fpsi2 = scatteredInterpolant(pn(:,1),pn(:,2),psi{2,2});
```

From the continuity theorem, however, one notes that the divergence of the current density on the lower adiabatic surface is simply the opposite of the divergence of the current density on the upper adiabatic surface.

5.8 Convergence

The numerical accuracy of **COL**SCAT2D can be tested by checking the conditions that all scattering probabilities add up to unity for a given collision energy. This ensures that there is conservation of probability. We can derive this from the following consideration, we expect the flux at the reactant boundary (a channel) to be the opposite of the flux (c channel) at the product boundary, i.e.

$$\mathbf{J}_a = -\mathbf{J}_c$$

where

$$\mathbf{J}_\gamma = \frac{-i\hbar}{2\mu_\gamma}\left\{\Psi_{\Gamma_\gamma}(\hat{n}\cdot\nabla)\Psi^*_{\Gamma_\gamma} - [(\hat{n}\cdot\nabla)\psi_{\Gamma_\gamma}]\psi^*_{\Gamma_\gamma})\right\}$$

where Ψ_{Γ_γ} is the projection of the wave function onto the γ channel boundary, $(\hat{n} \cdot \nabla)$ is the directional derivative normal to the boundary and μ is the reduced mass of the system in that channel, μ_γ from Eq. 11 in [?]. From the boundary condition we know that analytic form of the wave function on each boundary can be written as follows

$$\Psi_{\Gamma_a} = \frac{\exp\left[-ik_0R_a\right]\chi_0(r_a)}{\sqrt{v_a}} + \sum_j S^a_{0j} \frac{\exp\left[ik_jR_a\right]\chi^a_j(r_a)}{\sqrt{v_a}}$$

and

$$\Psi_{\Gamma_c} = \sum_{j'} S^c_{0j'} \frac{\exp\left[ik_{j'}R_c\right]\chi^c_{j'}(r_c)}{\sqrt{v_c}}$$

Using the orthogonality of the χ functions we can use the fact that $\mathbf{J}_a = -\mathbf{J}_c$ to derive

$$1 = \sum_j |S^a_{0j}|^2 + \sum_{j'} |S^c_{0j'}|^2$$

Relating this derivation to the language of **COL**SCAT2D the S-matrix element $|S^a_{0j}|^2$ is `N1(i,j)`, where i is corresponds to the ith value of the vector of total energy, `E(i)`. Likewise the S-matrix element $|S^c_{0j'}|^2$ is `N1(i,j')`. For a given single electronic surface calculation this can be tested in using the following command

```
% test the convergence of the colscat calculation at each energy
 S = sum(N{1},2)+sum(R{1},2)
```

The accuracy of a given calculation in **COL**SCAT2D can usually be improved by adjusting three things: (1) increase the number of mesh points in the grid creation subroutine by decreasing the value of `dq`, note: this is especially relevant at high collision energies, (2) increase the polynomial order of the basis functions used in the FEM calculation until the calculation converges, or (3) try increasing the size of the domain itself *and* the number of mesh points. To do this experiment with increasing the parameters, `'Rmax'` and `'vcont'` in `meshgen.m`.

5.9 Output Structure

With the many options for multiple states and electronic surfaces it is important to comment on how the output variables are organized.

5.9.1 Scattering Amplitudes

The non-reactive scattering amplitudes `N` and the reactive scattering amplitudes `R` are stored in what MATLAB calls a 'cell'. These two cell structures have dimension number of electronic states $\times$ the number of initial conditions. For example, consider a calculation with two initial conditions (Tutorial C in subsection 5.3), `N` and `R` have size 1×2.

Stored inside of each of these cells is a matrix of dimension `le` $\times$ (`nao+nco`), where `le` is the length of the `E` vector, i.e. the total number of input scattering energies, `nao` is the number of 'open' vibrational states in the reactant channel and `nco` is the number of 'open' in the product channel. The following portion of code shows how to calculate, simultaneously, the scattering amplitudes for a desired set of initial and final vibrational states.

```
% symmetric scattering example H+H2
mh = 1836.15264;       % mass of hydrogen in au
m  = [mh,mh,mh];       % [H,H,H]
E = 0.01:0.0001:0.06;  % energy range
iState = 1:2;            % initial vibrational state of reactants

%% calulate mesh for HH2 problem
[p,t,pf] = meshgen(m,[6 6],[.25 9 .25 9],[.1 .6],0.3,'vhh2');

%%  main scattering calculation using Polynomial order 4 basis functions, P4
[N,R,psi,viba,vibc,v,pn,tn,tln] = colscat2d(m,E,p,t,pf,iState,[1 1],'vhh2',4);

% collision energy in eV
ecol = (E-viba{1}.e(1))*27.211;

% the non-reactive probabilities H+H2(v=1) --> H+H2(v=0)
% N{1} refers to the non-reactive probabilities for initial state iState(1)
figure(1)
plot(ecol,N{1}(:,1));
title('H + H_2(v=1)\rightarrow H + H_2(v=0)','fontsize',24);
set(gca,'fontsize',20);
axis([ecol([1 end]) 0 1]);
set(gca,'xtick',0:.25:1.5);
xlabel('Collision Energy, eV');

% the reactive probabilities H+H2(v=2) --> H+H2(v=1)
% R{2} refers to the reactive probabilities for initial state iState(1)
figure(2)
plot(ecol,R{2}(:,2),'g');
title('H + H_2(v=2)\rightarrow H_2(v=1)+H','fontsize',24);
set(gca,'fontsize',20);
axis([ecol([1 end]) 0 1]);
set(gca,'xtick',0:.25:1.5);
xlabel('Collision Energy, eV');
```

5.10 Wave function and nodal points

The wave function output variable, `psi`, also has a cell type with dimension 1 $\times$ number of initial conditions. However, each cell in the `psi` variable contains a matrix that has dimension `np` $\times$ `le` where `np` is the number of nodal points in the mesh and `le` is the length of the `E` vector, i.e. the total number of input scattering energies.

The coordinates of the nodal points in the triangulation are stored in the variable `pn`. The `pn` variable is a matrix with dimension `np`$\times$2 and contains the mass-scaled Jacobi coordinates of the nodal points. Some subroutines ask for the full `pn` variable and other times you may be asked for the coordinates individually. The first coordinate, R_a, can be addressed using the syntax `pn(:,1)` and similarly the second coordinate, r_a, is addressed via the call `pn(:,2)`.

After running the following code produces plots of the probability density for various choices of initial and final states.

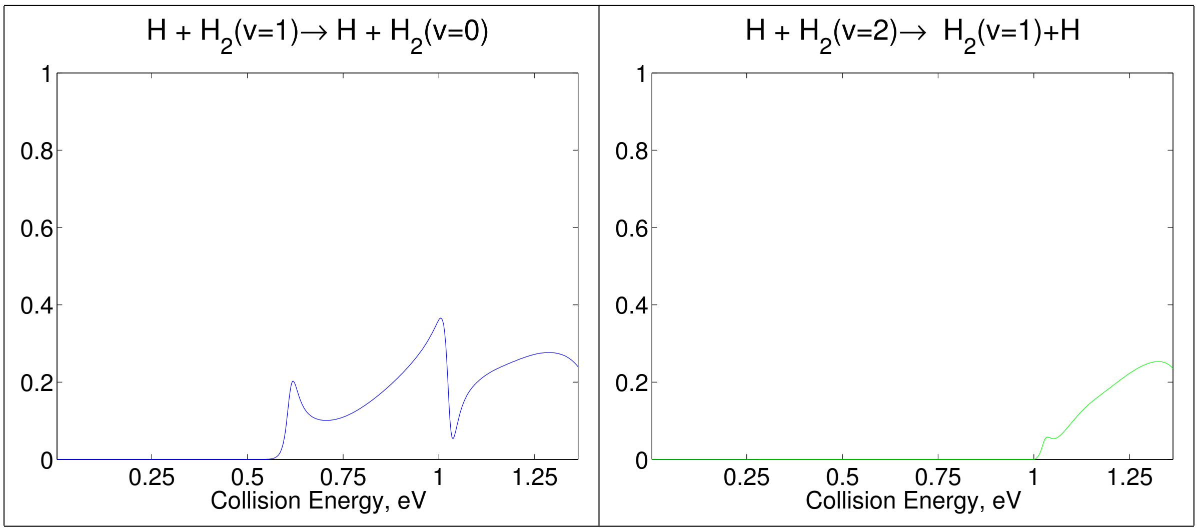


Figure 7: (left) Non-reactive (inelastic) $H+H_2(v=1) \rightarrow H+H_2(v=0)$ probabilities as a function of collision energy. (right) Reactive $H+H_2(v=2) \rightarrow H_2(v=1)+H$ probabilities as a function of collision energy.

```
% the probability density for H+H2(v=1)
% psi{1}(:,100) refers to the wave function with initial state v = iState(1)
% with a total energy = E(400) = 0.0499 hartree = 1.3578 eV
figure(3)
tricontour(pn,tln,abs(psi{1}(:,250)).^2,20)
title('Probability Density E_{tot} = 0.950 eV','fontsize',24);
set(gca,'fontsize',20);
axis off;
axis off;

% the probability density for H+H2(v=2)
% psi{2}(:,213) refers to the wave function with initial state v = iState(2)
% with a total energy = E(400) = 0.0349 hartree = 0.950 eV
figure(4)
tricontour(pn,tln,abs(psi{2}(:,400)).^2,25)
title('Probability Density E_{tot} = 1.358 eV','fontsize',24);
set(gca,'fontsize',20);
axis off;
```

5.11 Two-state Output Structure

The scripts in subsection 5.6 (`N` and `R`) and subsection 5.7 (`psi`) give detailed examples of how to reference the two state outputs generated by **COL**SCAT2D .

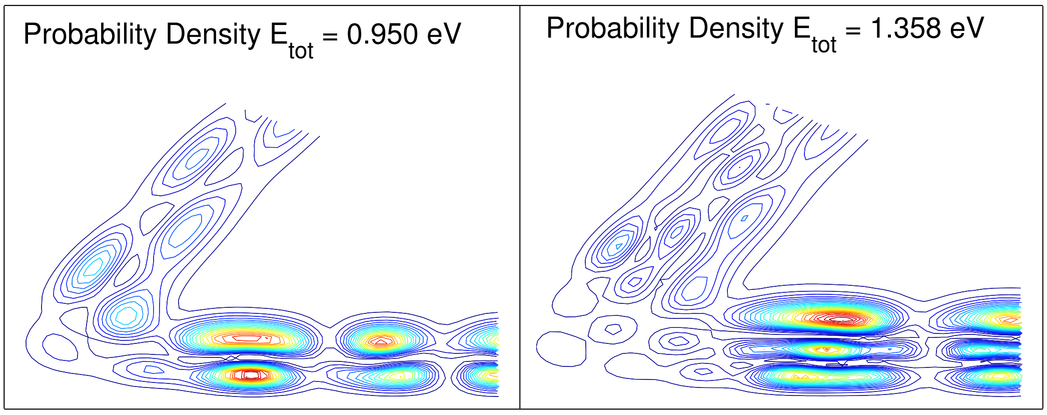


Figure 8: Probability densities for (left panel) H+H_2(v=1) at E_{tot} = 1.3578 eV, and (right) H+H_2(v=2) at E_{tot} = 1.5020 eV. We can see the nodes in the vibrational wave function, one node for v=1 in the left panel, and two nodes for v=2 in the right panel near the reactants.

6 Parallelization and the Parallel Computing Toolbox

With access to MATLAB's parallel computing toolbox (PCT) the user can easily parallelize any **COL**SCAT2D calculation. The PCT distributes the calls to the solver subroutine at each value of the total energy. To invoke parallelized code, you need only replace the '`for`' loop on line 33 with a '`parfor`'. MATLAB transparently oversees the distribution of the calculation.

Serial `for` loop (all versions)

```
%% Scattering probability as a function of energy
np = length(pn); le = length(E); U = cell(le,1);
for jj = 1:le
   U{jj} = solver2d(E(jj),T,V,O,M,viba,vibc,iState,iSurf,ba,bc,np,ns,Ra,Rc);
end
```

Parallel `for` loop (MATLAB2013 and after)

```
%% Scattering probability as a function of energy
np = length(pn); le = length(E); U = cell(le,1);
parfor jj = 1:le
   U{jj} = solver2d(E(jj),T,V,O,M,viba,vibc,iState,iSurf,ba,bc,np,ns,Ra,Rc);
end
```

Parallel `for` loop (before MATLAB2013)
If you have the parallel computing toolbox on a older version of MATLAB you must open the pool of cores before you begin the calculation and close them after the calculation. The pool is opened with the command `matlabpool(cores)` where cores is the number of cores in the pool. Closing the pool is done with `matlabpool close`. This should be done outside the call to **COL**SCAT2D .

7 Function Reference

7.1 adiabats.m

Inputs	Description	Dimension
vsig	ground state in Λ basis	$np \times 1$
vpi	first excited state in Λ basis	$np \times 1$
vso	spin-orbit constant	$np \times 1$
Outputs		
a1	ground state surface in adiabatic basis	$np \times 1$
a2	first excited state surface in adiabatic basis	$np \times 1$
theta	mixing angle that describes the rotation from diabetic to adiabatic bases	$np \times 1$

This subroutine takes the potential energy surfaces in the Λ basis and returns the adiabatic potentials and the mixing angle. This subroutine assumes the 2-state potential energy system of the form

$$V^{\Lambda}(u_1,u_2) = \begin{bmatrix} V_{\Sigma}(u_1,u_2) & -\sqrt{2}B(u_1,u_2) \\ -\sqrt{2}B(u_1,u_2) & V_{\Pi}(u_1,u_2)+B(u_1,u_2) \end{bmatrix}$$

where (u_1,u_2) are the bond coordinates for the given system, $B(u_1,u_2)$ is the spin orbit constant, `vso`. `theta` is the mixing angle that diagonalizes V^{Λ} at each point in space and is defined as follows

$$\tan\theta = \frac{2c}{a-b}$$

where $a = V_{\Sigma}$, $b = V_{\Pi} + B$ and $c = -\sqrt{2}B$. The mixing angle can then be used to determine the current density in the adiabatic bases as in Section 7.15.

7.2 boundaryind.m

This routine takes the mesh as its only input and determines which points lie on the reactant and product boundaries. These indices are ultimately used to construct the sparse matrices that extend the FEM matrices in the subroutine `extendmat2d.m` (subsection 7.5).

Inputs	Description	Dimension
pn	nodal points coordinate matrix (subsection 7.20)	$np \times 2$
pf	fixed points that define reactant and product mesh (subsection 7.16)	4×2
Parameters		
tol	distance tolerance for deciding if a point is on the boundary	
Outputs		
ba	index of nodes along reactant boundary (a-channel) sorted in increasing values of r_a.	$1 \times na$
bc	index of nodes along product boundary (c-channel) sorted in increasing values of r_c.	$1 \times nc$
pn	nodal points coordinate matrix (with shifted boundary points)	$np \times 2$

```
% generate a small mesh using the h+h2 mesh generator
mh = 1836.15264;     % mass of hydrogen in au
m = [mh,mh,mh];      % H + H2

% generate mesh
[p,t,pf,bnd] = meshgen(m,[6 6],[.25 9 .25 9],[.1 .6],0.4,'vhh2');

% generate a quadratic mesh
[pn,tn,tln] = polymesh(p,t,2);

% determine the points along the boundary
[ba,bc,pn] = boundary(pn,pf);

% plot the mesh and highlight the boundary points
hold on;
patch('vertices',pn,'faces',tln,'edgecol','k','facecol',[.8,.9,1]);
plot(pn(ba,1),pn(ba,2),'ko','markersize',8);
plot(pn(bc,1),pn(bc,2),'ro','markersize',8);
axis off
```

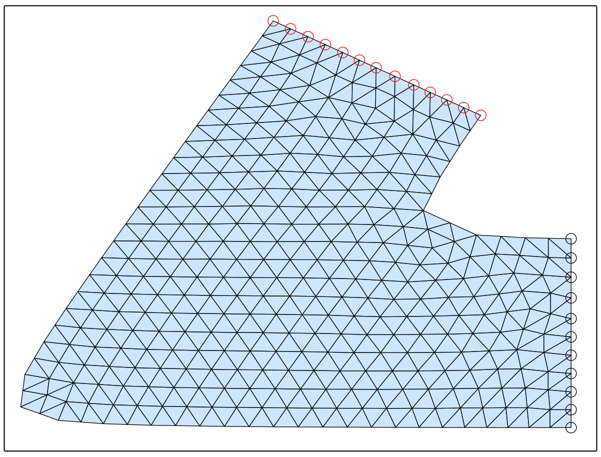


Figure 9: Given the mesh output from application of the meshgen routine, the `boundaryind.m` subroutine identifies which nodal points lie on the reactant boundary (ba indices shown in black) and which points on the product boundary (bc indices shown in red).

7.3 colscat2d.m

See the tutorials in section 5 for how to create inputs for and execute the main **COL**SCAT2D program. The scattering output variables `N`, `R` and `psi` are discussed above in subsection 5.9.

Inputs	Description	Dimension
m	mass vector in atomic units [m$_A$, m$_B$, m$_C$]	1 × 3
E	total energy vector in hartree	1 × le
p	points in the triangulation (subsection 7.16)	varies
t	connectivity matrix of triangulation (subsection 7.16)	varies
pf	fixed points on mesh define boundaries (subsection 7.16)	2 × 4
iState	initial vibrational state of BC	scalar
iSurf	initial electronic surface of A	scalar
vinput	function reference (subsection 7.28)	function handle
n	polynomial order of FEM calculation (1-5)	scalar
Outputs		
N	non-reactive probabilities	cell: nSurf × nStates
R	reactive probabilities	cell: nSurf × nStates
psi	wave function	cell: nSurf × nStates
viba	reactant boudnary vibrational structure (subsection 7.27)	
vibc	product boudnary vibrational structure (subsection 7.27)	
v	potential energy surface evaluated at nodal points	np × 1
pn	nodal points on mesh designed for polynomial order n (subsection 7.16)	2×np
tn	connectivity matrix of pn (subsection 7.16)	varies with n
tln	linearized connectivity matrix of pn (subsection 7.16)	varies with n

7.4 colscat2d_clebsch.m

See the tutorials in section 5 for how to create inputs for and execute the main **COL**SCAT2D program. The scattering output variables `N`, `R` and `psi` are discussed above in subsection 5.9.

Inputs	Description	Dimension
m	mass vector in atomic units [m_A, m_B, m_C]	1×3
E	total energy vector in hartree	$1 \times$ le
p	points in the triangulation (subsection 7.16)	varies
t	connectivity matrix of triangulation (subsection 7.16)	varies
pf	fixed points on mesh define boundaries (subsection 7.16)	2×4
iState	initial vibrational state of BC	scalar
iSurf	initial electronic surface of A	scalar
vinput	function reference (subsection 7.28)	function handle
n	polynomial order of FEM calculation (1-5)	scalar
Da	clebsch gordon matrix if reactant channel is different basis than potential basis	2×2
Dc	clebsch gordon matrix if product channel is different basis than potential basis	2×2
Outputs		
N	non-reactive probabilities	cell: nSurf $\times$ nStates
R	reactive probabilities	cell: nSurf $\times$ nStates
psi	wave function	cell: nSurf $\times$ nStates
viba	reactant boudnary vibrational structure (subsection 7.27)	
vibc	product boudnary vibrational structure (subsection 7.27)	
v	potential energy surface evaluated at nodal points	np $\times$ 1
pn	nodal points on mesh designed for polynomial order n (subsection 7.16)	2$\times$np
tn	connectivity matrix of pn (subsection 7.16)	varies with n
tln	linearized connectivity matrix of pn (subsection 7.16)	varies with n

The `colscat2d_clebsch.m` is very similar to the original `colscat2d.m` subroutine but allows for either the reactant or product channel to be in a different basis than the solution basis. This assumes that the transformed reactant/product basis can be described as a linear combination of the potential states (such as the Clebsch-Gordon transformation from coupled to uncoupled angular momentum states). The *Da* and *Dc* matrices give the transformation in the reactant and product channels, respectively. If both of these matrices are set to unity than the original **COL**SCAT2D functionality is obtained. The solution to the coupled matrix problem with natural (untransformed boundary conditions) can be written as

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & -\mathbf{B}_a^1 & -\mathbf{B}_c^1 & 0 & 0 \\ \mathbf{A}_{12} & \mathbf{A}_{22} & 0 & 0 & -\mathbf{B}_a^2 & -\mathbf{B}_c^2 \\ \mathbf{I}_a^1 & 0 & -\mathbf{F}_a^1 & 0 & 0 & 0 \\ \mathbf{I}_c^1 & 0 & 0 & -\mathbf{F}_c^1 & 0 & 0 \\ 0 & \mathbf{I}_a^2 & 0 & 0 & -\mathbf{F}_a^2 & 0 \\ 0 & \mathbf{I}_c^2 & 0 & 0 & 0 & -\mathbf{F}_c^2 \end{bmatrix} \begin{bmatrix} \mathbf{C}^1 \\ \mathbf{C}^2 \\ \mathbf{S}_a^1 \\ \mathbf{S}_c^1 \\ \mathbf{S}_a^2 \\ \mathbf{S}_c^2 \end{bmatrix} = \begin{bmatrix} \mathbf{b}^1 \\ 0 \\ \mathbf{f}^1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The above relationship is ideal for describing systems with multiple states where the potential in both the reactant and product arrangements is diagonal. The potential must be asymptotically diagonal because the non-zero offdiagonal terms will create coupling far away from the rearrangement region. Ultimately this creates sensitivity in the results to the location of the boundaries. Small changes in the asymptotic values of R_a or R_c which is not physically reasonable.

In cases where the coupled potential surface is not diagonal asymptotically we can use the Da or Dc matrix to get the desired form of the potential. In the case of F+H$_2$ the reactant arrangement is diagonal in the j_a basis but the product arrangement is diagonal in the Λ basis. The following matrix problem shows how `colscat2d_clebsch.m` implements this change of basis for the F+H$_2$ system. (note the rearrangement of the scattering probabilities in the vector of unknowns)

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{11} & -D_{11}\mathbf{B}_a^{3/2} & -D_{21}\mathbf{B}_a^{1/2} & -\mathbf{B}_c^{\Sigma} & 0 \\ \mathbf{A}_{12} & \mathbf{A}_{12} & -D_{12}\mathbf{B}_a^{3/2} & -D_{22}\mathbf{B}_a^{1/2} & 0 & -\mathbf{B}_c^{\Pi} \\ \mathbf{I}_a^{\Sigma} & 0 & -D_{11}\mathbf{F}_a^{3/2} & -D_{21}\mathbf{F}_a^{1/2} & 0 & 0 \\ 0 & \mathbf{I}_a^{\Pi} & -D_{12}\mathbf{F}_a^{3/2} & -D_{22}\mathbf{F}_a^{1/2} & 0 & 0 \\ \mathbf{I}_c^{\Sigma} & 0 & 0 & 0 & -\mathbf{F}_c^{\Sigma} & 0 \\ 0 & \mathbf{I}_c^{\Pi} & 0 & 0 & 0 & -\mathbf{F}_c^{\Pi} \end{bmatrix} \begin{bmatrix} \mathbf{C}^{\Sigma} \\ \mathbf{C}^{\Pi} \\ \mathbf{S}_a^{3/2} \\ \mathbf{S}_a^{1/2} \\ \mathbf{S}_c^{\Sigma} \\ \mathbf{S}_c^{\Pi} \end{bmatrix} = \begin{bmatrix} D_{11}\mathbf{b}^{3/2} \\ D_{12}\mathbf{b}^{3/2} \\ D_{11}\mathbf{f}^{3/2} \\ D_{12}\mathbf{f}^{3/2} \\ 0 \\ 0 \end{bmatrix}$$

The transformation from the Λ basis to the j_a basis for the F+H$_2$ can be written as follows

$$|j\rangle = \mathbf{D}\,|\Lambda\rangle$$

where

$$\mathbf{D} = \frac{1}{\sqrt{3}} \begin{bmatrix} i\sqrt{2} & 1 \\ -i & \sqrt{2} \end{bmatrix}$$

We would like to write the boundary condition on the reactant side as an incoming part on the $j_a = 3/2$ state and outgoing part on both the $j_a = 3/2$ and $j_a = 1/2$ states. In the j_a basis this looks like

$$\Psi_{\Gamma_a} = \mathit{incoming}\,|3/2\rangle + \mathit{outgoing}\,|3/2\rangle + \mathit{outgoing}\,|1/2\rangle$$

We can then use the above transformation to write this in terms of the electronic states actually used in the calculation namely the Λ states.

$$\Psi_{\Gamma_a} = \mathit{incoming}(D_{11}\,|\Sigma\rangle + D_{12}\,|\Pi\rangle) + \mathit{outgoing}(D_{11}\,|\Sigma\rangle + D_{12}\,|\Pi\rangle) + \mathit{outgoing}(D_{21}\,|\Sigma\rangle + D_{22}\,|\Pi\rangle)$$

The coloring of the transformed and untransformed matrix equations shows how this change of basis in the boundary conditions is implemented. When `colscat_clebsch.m` builds the extended matrices it simply scales $\mathbf{B}$, $\mathbf{F}$ and $\mathbf{b}$ using the matrix elements of $\mathbf{D}$ and makes a slightly less sparse matrix problem. It is clear that if $\mathbf{D}$ is unity the original implementation is recovered.

IMPORTANT - Notice that the incoming matrices $\mathbf{B}_a^j$ are in a different basis than the outgoing ones $\mathbf{B}_c^{\Lambda}$. This requires that the supplied boundary potential (subsection 7.28) gives the potential in the correct basis on each boundary. (In the case of F+H$_2$ the `va` output of `vinput` should be in the j_a basis to build the colored $\mathbf{B}_a$, $\mathbf{F}_a$ and $\mathbf{b}_a$ matrices and the output `vc` should be in the Λ basis to build the $\mathbf{B}_c$, $\mathbf{F}_c$ and $\mathbf{b}_c$ matrices. See (subsection 5.7) for an example of how to implement these mixed boundary conditions.

7.5 extendmat2d.m

The `extendmat2d.m` is used to extend the system from a simple bound state calculation to include the scattering boundary conditions.

Inputs	Description	Dimension
viba	reactant boudnary vibrational structure (subsection 7.27)	
vibc	product boudnary vibrational structure (subsection 7.27)	
ba	indices on a-channel boundary (reactants)	1×na
bc	indices on c-channel boundary (products)	1×nc
M	mass structure (subsection 7.12)	
E	total energy of scattering calculation	scalar
np	number of nodal points in mesh	scalar
ns	number of electronic surfaces in calculation	scalar
Ra	Asymptotic limit of mass-scaled jacobi vector R in reactant channel	scalar
Rc	Asymptotic limit of mass-scaled jacobi vector R in product channel	scalar
Outputs		
B	extended boundary integral FEM matrix	np*ns×(na+nc)*ns
F	extended boundary condition FEM matrix	(na+nc)*ns×(na+nc)*ns
I	extended unity-like FEM matrix	(na+nc)*ns×np*ns

where `na` is the number of nodal points on the reactant boundary, `nc` is the number of points on the product boundary and `ns` is the number of electronic surfaces involved in the calculation, respectively.

7.6 extendmat2d_clebsch.m

Builds the extended right hand side of the linear system of equations using the input transformation matrices `Da` and `Dc`. The transformed matrix equation is shown in subsection 7.4.

7.7 extendrhs2d.m

The `extendrhs2d.m` subroutine is used to build the right hand side of the extended FEM problem. One column of the right hand side is built for each initial condition. The initial conditions are identified by the initial vibrational state (`iState`) and the initial electronic surface (`iSurf`). `nstate` is the number of initial states, i.e. the length of the `iState` vector.

Inputs	Description	Dimension
viba	reactant boudnary vibrational structure (subsection 7.27)	
ba	indices on a-channel boundary (reactants)	1× na
bc	indices on c-channel boundary (products)	1× nc
iState	initial vibrational state of BC	1 × nstate
iSurf	initial atomic electronic state of A	1 × nstate
M	mass structure (subsection 7.12)	
E	total energy of scattering calculation	scalar
np	number of nodal points in mesh	scalar
ns	number of electronic surfaces in calculation	scalar
Outputs		
b	boundary integral and boundary condition of initial state	(np+na+nc)*ns × nstate

7.8 extendrhs2d_clebsch.m

Builds the right hand side of the linear system of equations using the input transformation matrix `Da`. The transformed matrix equation is shown in subsection 7.4.

7.9 fem2d.m

This script builds the FEM matrices using only the nodal coordinates and the value of the potential energy surface evaluated at each node. The simplicity of the code is somewhat obscured by the vectorized construction of the matrices. This script will build the FEM matrices for any polynomial order PN. Note: increased polynomial order has improved accuracy but decreased speed because these FEM matrices are less sparse due to increased size of the connectivity matrices (subsection 7.20). Below the basics of `fem2d.m` for a P2 are provided but these results are easily generalized to any polynomial order.

Inputs	Description	Dimension
pn	nodal points in quadratic mesh	np × 2
tn	quadratic mesh	nt × 6
v	potential energy surface at each node	np × 1
n	polynomial order of the FEM basis functions	scalar
Outputs		
T	kinetic energy PN FEM matrix	(np*ns) × (np*ns)
V	potential energy PN FEM matrix	(np*ns) × (np*ns)
O	kinetic energy PN FEM matrix	(np*ns) × (np*ns)

In Figure 7.9 we provide a sample triangulation with 6 triangles and 19 nodes; 7 vertex nodes shown in black and 12 midpoint nodes shown in red. The following table uses the notation of the published paper [?] to describe the FEM basis functions for the first seven nodes in this quadratic triangulation.

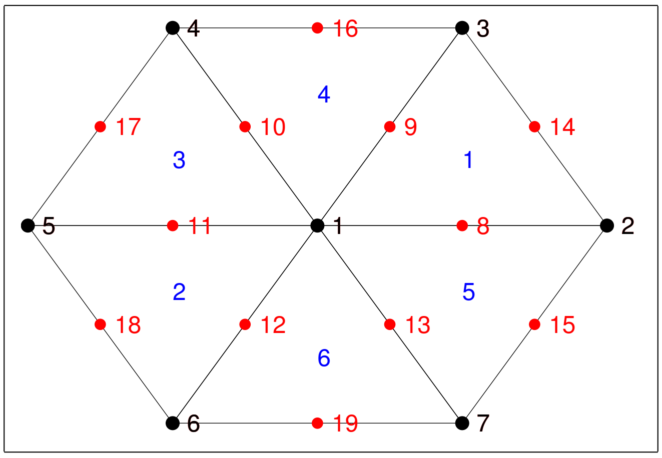


Figure 10: A sample P2 mesh and numbering scheme. The nodes of the linear mesh are labeled 1–7 in black. In the quadratic mesh additional nodes are added at the midpoints, labeled 8–19 in red. Finally the triangles are numbered 1–6 in blue.

node	basis functions
1	$\Phi_1 = \phi_{11} + \phi_{12} + \phi_{13} + \phi_{14} + \phi_{15} + \phi_{16}$
2	$\Phi_2 = \phi_{21} + \phi_{25}$
3	$\Phi_3 = \phi_{31} + \phi_{34}$
4	$\Phi_4 = \phi_{43} + \phi_{44}$
5	$\Phi_5 = \phi_{52} + \phi_{53}$
6	$\Phi_6 = \phi_{62} + \phi_{66}$
7	$\Phi_7 = \phi_{75} + \phi_{76}$
8	$\Phi_8 = \phi_{81} + \phi_{85}$
9	$\Phi_9 = \phi_{91} + \phi_{94}$
10	$\Phi_{10} = \phi_{10,3} + \phi_{10,4}$
11	$\Phi_{11} = \phi_{11,2} + \phi_{11,3}$
12	$\Phi_{12} = \phi_{12,2} + \phi_{12,6}$
13	$\Phi_{13} = \phi_{13,5} + \phi_{13,6}$
14	$\Phi_{14} = \phi_{14,1}$
15	$\Phi_{15} = \phi_{15,5}$
16	$\Phi_{16} = \phi_{16,4}$
17	$\Phi_{17} = \phi_{17,3}$
18	$\Phi_{18} = \phi_{18,2}$
19	$\Phi_{19} = \phi_{19,6}$

Table 1: Basis functions of a sample quadratic triangulation.

where ϕ_{ij} is only defined over triangle j, is unity at node i and is zero at every other node associated with triangle j. To compute the integrals in Eq. (31) of the published paper [**?**] one could, in principle, loop over every node and compute the nonzero integrals.

$$O_{ij} = \int_\Omega \Phi_i \Phi_j \; dRdr \tag{1}$$

For example, one could directly substitute the basis functions from Table 7.9 into the first term in Eq.

(31) of the published paper [?]. Using the expansion in terms of the bivariate ϕ functions, we have

$$O_{ij} = \sum_{k,l} \int_\Omega \delta_{kl} \phi_{ik} \phi_{jl} \; dRdr \tag{2}$$

where we sum (k) over all triangles associated with node i and (l) over all triangles associated with node j. The δ_{kl} term reminds us that these basis functions are locally defined, and hence these integrals are non-zero unless k and l refer to the same triangle.

We make this notion more concrete by picking two nodes, say node 1 and node 7, from Figure 7.9. The matrix element for the **O** matrix between these two nodes would then be

$$\begin{aligned} O_{17} &= \int_\Omega \Phi_1 \Phi_7 \; dRdr \\ &= \int_\Omega (\phi_{11} + \phi_{12} + \phi_{13} + \phi_{14} + \phi_{15} + \phi_{16})(\phi_{75} + \phi_{76}) \; dRdr \\ &= \int_\Omega \phi_{15}\phi_{75} \; dRdr + \int_\Omega \phi_{16}\phi_{76} \; dRdr \end{aligned} \tag{3}$$

Calculating the integrals in this fashion would require an explicit loop over all nodal points, and an inner loop over all possible overlapping nodes. To avoid looping over nodal indices and make use of MATLAB's vectorization capabilities. We map each triangle to the standard triangle, which is defined by the points (0,0), (0,0.5), (0,1), (0.5,0.5), (1,0), and (0.5,0). Six basis functions are associated with the standard triangle. Each basis function is unity at a given node and zero at the other five nodes. These are describe in detail in Section III D of the associated publication [?].

We want to evaluate the integrals shown in Eqs. (35–37) in the referenced paper. The **T** and **O** matrix elements require the evaluation of integrals containing 2 basis functions, hence there are 36 possible combinations for a given triangle with 6 basis functions. Similarly, the **V** matrix elements involve 3 basis functions and hence have 216 possible combinations for a triangle with 6 associated basis functions, note: 15 of these values are redundant by symmetry. These possible combinations include symmetric values, i.e. $\int_\Omega \phi_{ik}\phi_{jk} = \int_\Omega \phi_{jk}\phi_{ik}$.

All 36 elements of the **T** and **O** matrices, and all 216 elements of the **V** matrix are determined in two steps. First the so-called 'constructor' matrices are formed by calculating the coefficients of the basis functions for the standard triangle and then using an analytic expression to evaluate the O_{ij}, T_{ij} and V_{ijk} matrix elements for the standard triangle basis functions. Along with the nodal coordinates, the potential energy surface at the nodal points and these 'constructor' matrices the full FEM matrices can be built using a single matrix multiplication.

The script `stdint2d.m` determines the values of the 'constructor' matrices as follows. Any one of the three desired matrix elements can be written in general as follows

$$X_{ij} = \sum_{a,b}^{a+b \le P} c_{ab} \int_0^1 \int_0^{1-y} x^a y^b \; dxdy$$

The double integral has the following analytic solution

$$\int_0^1 \int_0^{1-y} x^a y^b \, dxdy = \frac{a!b!}{(a+b+2)!} = \tilde{X}_{ab}$$

which can easily be vectorized in MATLAB. Finally we can write the coefficients, the $c'_{ab}s$ in matrix form and get a matrix expression for this integral, namely,

$$X = C^T \tilde{X} C$$

Different choices of C and $\tilde{X}$ yield different constructor matrices. These constructor matrices are created anew for each calculation. They could be calculated a single time and then stored and loaded from memory, however, given they depend only analytic expressions for a given value of polynomial order `N`, they can be vectorized and hence their determination is on the orders of hundredths of second. These constructor matrices are then passed to the `fem2d.m` script to build the full FEM matrices. Section 7.23 contains more detail on how these integrals are computed.

With the 'constructor' matrices in hand, i.e. `t1`, `v1` and `o1`, the variables `Xscalar` and `Yscalar` are used to build the final FEM matrices. These two variables identify all 36 combinations of nodal points (in the case of P2) for every single triangle. These two variables are the core of the vectorized construction of the FEM matrices.

For example, the 5th row of the `tn` variable for the triangulation shown in Fig. 7.9 is

```
tn(5,;) = [2 1 7 8 13 15].
```

The `Xscalar` and `Yscalar` variables identify all 36 pairs of these 6 indices, that is

```
Xscalar(5,:)=[2 1 7 8 13 15 2 1 7 8 13 15 2 1 7 8 13 15... 2 1 7 8 13 15]
```

and

```
Yscalar(5,:)=[2 2 2 2 2 2 1 1 1 1 1 1 7 7 7 7 7 7... 15 15 15 15 15 15].
```

The **O** matrix for example, is then constructed using the following call

```
% overlap matrix: int \phi_i \phi_j
omat=sparse(Xscalar,Yscalar,bsxfun(@times,o1,2*areas));
```

where the `o1` variable has dimension 1×36 and contains the standard triangle overlap integrals and the variable `areas` is dimension 6×1 containing the areas of each triangle in the triangulation. The `bsxfun` creates a matrix with dimension 6×36 where each the ith row is `o1*(2*areas(i))`. Multiplying by two times the area of the ith triangle is identical to the prefactor of the determinant det[**M**] in Eq. 35 [**?**].

From the fifth triangle, there would be a single contribution to the matrix element `omat(1,7)` which would correspond to $\int_\Omega \nabla\phi_{15} \cdot \nabla\phi_{75} \, dRdr$. Similarly, from the sixth row of `Xscalar`, `Yscalar` a similar element would be added to `omat(1,7)` corresponding to $\int_\Omega \nabla\phi_{16} \cdot \nabla\phi_{76} \, dRdr$, which gives us O_{17} in Eqn. (3). The evaluation of the **T** and **V** matrices are slightly more complicated by the presence of the ∇ operators in the case of the kinetic energy matrix elements and the presence of a third basis function in the potential energy matrix integrals. These are covered in more detail in the description of `stdint2d.m` in Section 7.23.

Ultimately, because all of the matrix elements needed for these FE matrices can be calculated in terms of the standard triangle basis functions and are known ahead of time and because we can refer to all matrix elements simultaneously (termed vectorization in MATLAB jargon) we can build the entire **O**, **T** and **V** matrices in a single line for each matrix. Furthermore, because this is similar to looping over the triangles there is no fear of double counting integrals.

7.10 flux.m

One of the outputs of the main **COL**SCAT2D routine is the scattering wave function, Ψ. The probability density current field (the probability flux vector field), **J**, allows us to visualize the quantum hydrodynamic behavior of the chemical reaction. This probability density current field is,

$$\mathbf{J}(q) = \frac{-i\hbar}{2\mu_a}\left\{\Psi(q)\nabla\Psi^*(q) - [\nabla\psi(q)]\psi^*(q)\right\}$$

where μ_a is the reduced mass in the reactant arrangement (subsection 7.12). An example of visualizing the probability density current field of the scattering wave function is shown in subsection 5.5.

Inputs	Description	Dimension
p	matrix of mass-scaled Jacobi coordinates	np $\times$ 2
t	triangulation (connectivity matrix) of p	nt $\times$ 3
psi	scattering wave function	np $\times$ le
m	mass vector $[\mathrm{M_a}, \mathrm{M_b}, \mathrm{M_c}]$	1 $\times$ 3
Outputs		
$\mathrm{J_x}$	flux in the x-direction	np $\times$ 1
$\mathrm{J_y}$	flux in the y-direction	np $\times$ 1

7.11 jacobi.m

Mass-scaled Jacobi coordinates are used for most of the calculations in the **COL**SCAT2D package. However, the boundary functions and their integrals are more naturally written in terms of the unscaled Jacobi coordinates. `jacobi.m` is used to convert from the mass scaled to unscaled Jacobi coordinates along the product and reactant boundaries.

Inputs	Description	Dimension
pn	matrix of mass-scaled Jacobi coordinates	np $\times$ 2
ba	indices on a-channel boundary (reactants)	1$\times$ na
bc	indices on c-channel boundary (products)	1$\times$ nc
M	mass structure (subsection 7.12)	
Outputs		
ra	unscaled Jacobi reactant coordinate (diatomic separation)	na $\times$ 1
rc	unscaled Jacobi product coordinate (diatomic separation)	nc $\times$ 1

7.12 mass.m

The `mass.m` script calculates many scaling parameters used frequently in the scattering calculation. These are stored conveniently in the output structure.

Inputs	Description	Dimension
m	mass vector [M_a, M_b, M_c]	1×3
Outputs		
mass	mass parameter structure	
mass.Ma	mass of atom A in atomic units	scalar
mass.Mb	mass of atom B in atomic units	scalar
mass.Mc	mass of atom C in atomic units	scalar
mass.M	sum of atomic masses	scalar
mass.alpha	skew angle from bond coordinates to mass-scaled Jacobi coordinates	scalar
mass.mu	reduced mass of the system	scalar
mass.muab	A-B diatomic reduced mass	scalar
mass.mubc	B-C diatomic reduced mass	scalar
mass.mua	reduced mass of reactant arrangement (A+BC)	scalar
mass.muc	reduced mass of product arrangement (AB+C)	scalar
mass.lama	reactant conversion factor (bond coordinates to reactant mass-scaled Jacobi coordinates)	scalar
mass.lamc	product conversion factor (bond coordinates to product mass-scaled Jacobi coordinates)	scalar

7.13 mepath.m

We include in **COL**SCAT2D our implementation of the bead on a string method of Weinan *et al.* [?] to determine the minimum energy path. Below is an example of how to call the `mepath.m` subroutine.

Inputs	Description	Dimension
x	internuclear separation coordinate (must be a meshgrid)	nrow × ncol
y	internuclear separation coordinate (must be a meshgrid)	nrow × ncol
v	potential energy surface at each point in the x,y meshgrid	nrow × ncol
Parameters		
dt	time step of steepest descent algorithm	scalar
tol	error tolerance (measured as the change in the numerical integral of vbar)	scalar
maxit	maximum number of iterations in main loop	scalar
n	number of beads on the string	scalar
freq	how many loop steps in between plotting events	scalar
plot opt	1 to visualize beads or 0 for no plotting	boolean
vcont	initial potential contour for the beads	scalar
exitmsg	1 to display how the subroutine exited, 0 to suppress any message on exiting	boolean
Outputs		
phi	coordinates of reaction path	n × 2
vphi	potential energy along reaction path	n × 1
rbar	approximate location of barrier	1 × 2
vbar	approximate value of potential energy at barrier	scalar

```
% generate meshgrid potential
r = 0.25:0.05:8.9;
[x,y]=meshgrid(r,r);
v=vhh2(x(:),y(:)); %here we are calling the H+H2 PES. You can change this as you wish
v = reshape(v,size(x));

% calculate the minimum energy path
[phi,vphi,rbar0,vbar0] = mepath(x,y,v);

% superimpose the minimum energy reaction path on a contour plot of the PES
figure(1);
hold on;
contour(x,y,v,0.01:.05:0.5);
plot(phi(:,1),phi(:,2),'ko');
set(gca,'xtick',0:2:8);
set(gca,'ytick',0:2:8);
set(gca,'fontsize',20);
xlabel('r_{H1H2}','fontsize',24);
ylabel('r_{H2H3}','fontsize',24);

% plot the potential energy along the minimum energy reaction path
figure(2);
plot(vphi*1000);
set(gca,'fontsize',20);
xlabel('reaction coordinate','fontsize',24);
ylabel('potential energy, mHartree','fontsize',24);
```

```
set(gca,'xtick',[]);
axis([0 200 -5 20]);
```

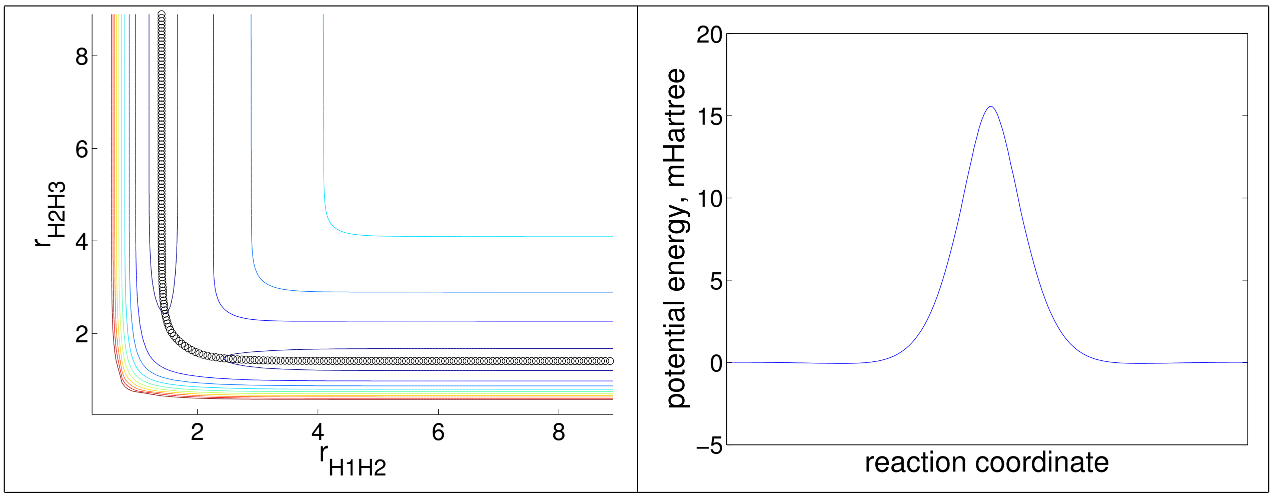


Figure 11: Output of the `mepath.m` script (subsection 7.13). (Left) The minimum energy path superimposed on a contour plot of the H+H_2 PES of Mieklke and co-workers. [?] (Right) The potential along the minimum energy reaction path.

7.14 mepbarrier.m

The `mepbarrier.m` script is almost indentical to `mepath.m` but modified to search only around the barrier location. The sample code below illustrates how to use `mepbarrier.m` to improve the automated estimation of the barrier location for the H+H_2 system.

Inputs	Description	Dimension
rbar0	estimate of barrier location (bond coordinates)	2 × 1
theta	tangent direction of reaction path at barrier (estimate in degrees)	scalar
dr	box size around transition state estimate	scalar
vinput	potential energy surface function handle (subsection 7.28)	
Parameters		
nsearch	number of beads along string in each search direction	
dt	time step of steepest descent algorithm	scalar
tol	error tolerance (measured as the change in the numerical integral of vbar)	scalar
maxit	maximum number of iterations in main loop	scalar
n	number of beads on the string	scalar
freq	how many loop steps in between plotting events	scalar
plot opt	1 to visualize beads or 0 for no plotting	boolean
exitmsg	1 to display how the subroutine exited, 0 to suppress any message on exiting	boolean
Outputs		
rbar	approximate location of barrier	1 × 2
vbar	approximate value of potential energy at barrier	scalar

```
% generate meshgrid potential
r = 0.25:0.025:8.9;
[x,y]=meshgrid(r,r);
v=reshape(vhh2(x(:),y(:)),size(x));

% mass for h+h2
mh = 1836.15264;     % mass of hydrogen in au
m  = [mh,mh,mh];

% rbar0 is the initial guess (output from mep.m)
rbar0 = [1.7596, 1.7582];

% calculate a better approximation to the barrier
[rbar,vbar] = mepbarrier(rbar0,45,.05,'vhh2');

% calculate the TS frequency, w, and the TS direction, theta
[w,theta] = normalmodes(m,rbar,.01,'vhh2');

% plot correction to the barrier location w TS direction
figure(1);
hold on;
axis([1.69 1.81 1.69 1.81]);
contour(x,y,v,.0155:.00001:.016);
plot(rbar0(1),rbar0(2),'bo');
plot(rbar(1),rbar(2),'ro');
arrow(rbar',rbar'+theta(1:2,2)/8,10);
arrow(rbar',rbar'+theta(1:2,4)/8,10);
set(gca,'xtick',1.7:.025:1.81);
set(gca,'ytick',1.7:.025:1.81);
set(gca,'fontsize',20);
```

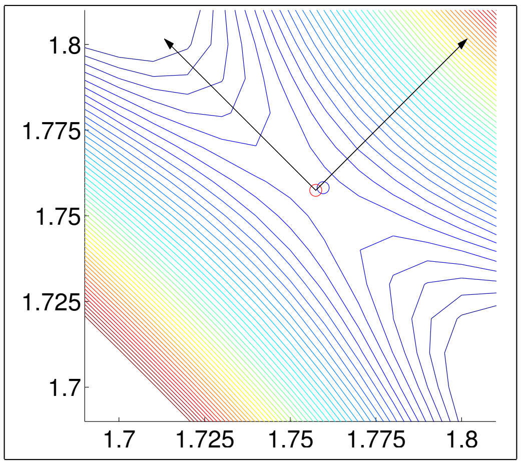


Figure 12: The blue dot marks the initial estimation of the transition state for Mielke's $H+H_2$ PES from the output of `mepath.m`. This initial estimate of the transition state is then recursively improved using another implementation of the minimum energy path algorithm. The second estimation of the transition state is shown with the red dot. This improved barrier location is then used as the input for `normalmodes.m` to determine the direction and vibrational frequency of the transition state. The direction of the two normal mode vibrations at the barrier are shown with the two arrows.

7.15 meshflux.m

Inputs	Description	Dimension
psi1	wave function on lower diabatic surface (assumes meshgrid)	$np \times 1$
psi2	wave function on uppter diabatic surface (assumes meshgrid)	$np \times 1$
theta	mixing angle that describes the rotation from diabetic to adiabatic bases (assumes meshgrid)	$np \times 1$
m	mass vector [M_a, M_b, M_c]	1×3
hx	grid spacing in x-direction	scalar
hy	grid spacing in y-direction	scalar
Outputs		
Jx1	current density on lower adiabatic surface in x direction (R_a direction)	$np \times 1$
Jy1	current density on lower adiabatic surface in y direction (r_a direction)	$np \times 1$
Jx2	current density on first excited adiabatic surface in x direction (R_a direction)	$np \times 1$
Jy2	current density on first excited adiabatic surface in y direction (r_a direction)	$np \times 1$

This subroutine assumes you have interpolated the wave function in a diabatic basis for a two-state system to a mesh grid with uniform spacing hx (hy) in the x-direction (y-direction). Using gradient and the expression for the adiabatic current density from the second paper in this series[?] this subroutine returns the current density in the two state system on the adiabatic surfaces. See Section 5.6 for an example on calling this subroutine using the outputs of **COL**SCAT2D .

7.16 meshgen.m

The `meshgen.m` script is one of the more complicated function in the **COL**SCAT2D suite. This script prepares input for use in the `distmesh2d.m` script written by Persson and Strang. [?] The mesh generation in **COL**SCAT2D is defined by the potential energy surface used for a given system.

To use Persson and Strang's `distmesh` we must define a signed distance function, which given all the points in the mesh returns the signed distance to the boundary of the mesh. While there are many possible ways to define the boundary for the collinear problem, in this mesh generation routine, we have chased to fit the inner and outer potential contours to hyperbolas (a bizarrely accurate fit for a wide range of potential functions). Once these hyperbolas have been defined we make use of `distmesh`'s implicit distance function capability and the values defined in `Rmax` to calculate the distance between every nodal point and the nearest point on the complicated boundary. This is done in the function `dmesh.m`, however this subroutine is not discussed here and the authors recommend Persson and Strang's documentation for how to write signed distance functions for novel boundary definitions. [?]

Inputs	Description	Dimension
m	mass vector [M_a, M_b, M_c]	1×3
Rmax	[Ra_max Rc_max] cut off values in the reactant and product channels in mass-weighted jacobi coordinates	1×2
ubox	bounding box for the potential (in bond coordinates) [u1min u1max u2min u2max]	1×4
vcont	[vin vout] value of PES along inner and outer boundary (hartrees)	1×2
dq	approximate size of triangle in mass-scaled jacobi coordinates	scalar
vinput	potential energy surface function handle	string
Outputs		
p	matrix of all points in mesh	varies $\times$ 2
t	connectivity matrix of points in mesh	varies $\times$ 3
pf	fixed points of mesh (distmesh2d parameter)	4×2
bnd	points approximating the boundary to the mesh	varies $\times$ 2

To generate a new mesh the user must specify the following parameters, `m`, `Rmax`, `ubox`, `vcont`, `dq` and `vinput`. The mass vector, `m`, is simply the mass vector in atomic units. The `Rmax` variable sets the cut-off distance in the reactant and product channels, this value must be provided in terms of the mass-scaled jacobi coordinates, R_a and R_c, for the reactant and product cut-offs, respectively. The `ubox` parameter defines the box, in bond coordinates, used to generate the potential. When converted to mass-scaled jacobi coordinates, ubox must be large enough to contain the values of R_a and R_c. The parameter `vcont` is used to specify the other two cut-off values distances for the mesh, namely `meshgen,m` uses contour lines of the potential to define the inner and outer boundaries. The `dq` parameter is used by `distmesh2d.m` as the approximate size of triangles in the mesh. Lastly, the user must specify the potential energy surface function handle, `vinput`.

Here follows an outline of how to use the `meshgen.m` code to generate meshes suitable for **COL**SCAT2D . We will use the H+H_2 system as an example.

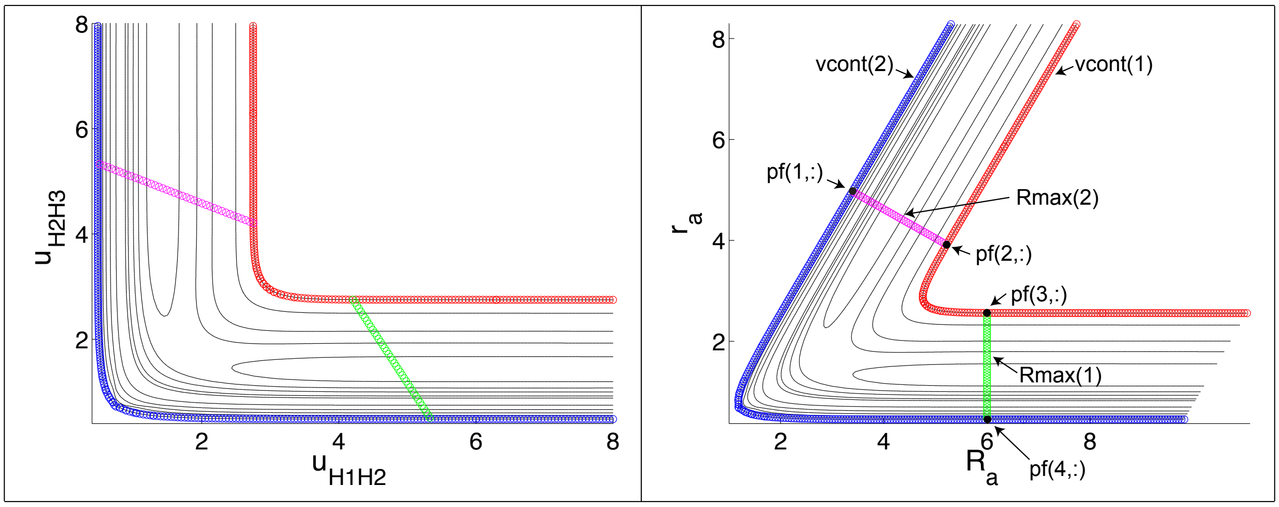


Figure 13: Mesh boundary for H+H_2 using the potential energy surface of Mielke *et al.* [?]. The red and blue curves correspond, respectively, to the outer and inner boundaries. These correspond to equipotential contours of 0.1 and 0.6 hartrees, respectively, hence `vcont = [0.1 0.6]`. The green and magenta lines correspond to `Rmax(1) = 6` and `Rmax(2) = 6`, respectively. The two panels are plots in internuclear separation (bond) coordinates (left) and mass-scaled Jacobi coordinates (right). The input parameters `vcont` and `Rmax` are shown with the output parameter `pf` from the `meshgen.m` subroutine.

Once good choices for `Rmax` and `vcont` have been made we need to choose values for `ubox` and `dq`. The values used for `ubox` are chosen based on the fact that the `vhh2.m` potential energy function used here is an interpolative routine and the values of `ubox` are chose to reflect the domain and range of the available *ab initio* points. We can start with a relatively large value of `dq = 0.3`. Below we show the MATLAB command to generate a mesh with this set of parameters and how to visualize the mesh using MATLAB's `trimesh.m` routine.

```
  %% calulate mesh for HH2 problem
[p,t,pf,bnd] = meshgen(m,[6 6],[0.25 9 0.25 9],[0.1 0.6],0.3,'vhh2');

% plot the triangulation
hold on
trimesh(t,p(:,1),p(:,2),'color','k')
plot(bnd(:,1),bnd(:,2),'r')
axis off
```

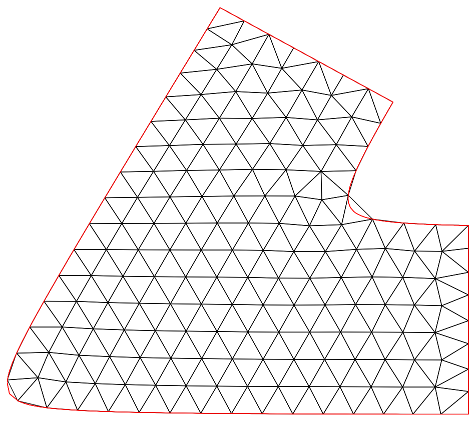


Figure 14: A sample mesh for the H+H_2 system. The red line shows the boundary used by `distmesh` to generate the mesh via the signed distance function `mesh.m`.

The outputs of `meshgen.m` are used directly as inputs for `colscat2d.m`. The user specifies the polynomial order of the basis functions used in the FEM calculation and `colscat2d.m` adds more points to the triangulation of `p` and `t` and described in subsection 7.20, (3 additional points for P2, 7 additional points for P3, 12 additional points for P4 and 18 additional points for P5).

7.17 normalmodes.m

Inputs	Description	Dimension
m	mass vector [M_a, M_b, M_c]	1 × 3
rbar	barrier location (bond coordinates)	2 × 1
h	step size for numerical derivative	scalar
vinput	potential energy surface function handle (subsection 7.28)	
Outputs		
w	normal mode frequency	scalar
theta	direction of transition state vibrations	2 × 4

Given an approximate location of the barrier this script calculates the vibrational frequency and direction of vibration of the normal mode vibrations of the transition state for a given system. This script uses the potential energy input function handle to determine the potential energy surface on a small grid in the neighborhood of the barrier. This grid is then used to determine the hessian of the potential energy surface at the barrier. The hessian is then transformed to the normal coordinates and becomes a diagonal matrix with two spring force constants. The positive valued spring force constant can be used to determine the vibrational frequency of the normal mode vibration at the transition state. This script also returns the directions of the normal mode vibration. The example in subsection 7.14 illustrates how to call `normalmodes.m` and how to plot the transition state direction.

7.18 nrsort2d.m

The output of `solver2d.m` is a cell structure that is indexed by the energy vector. `nrsort2d.m` reorganizes this cell structure to a new cell structure that is indexed by the initial condition. This greatly simplifies the plotting of the wave function and scattering probabilities.

Inputs	Description	Dimension
U	scattering dynamics solution at every energy, (cell)	1 × length(E)
le	number of scattering energies	scalar
np	number of nodal points in mesh	scalar
ns	number of electronic states	scalar
iState	vector containing initial states	vector
maxE	maximum value of the energy vector	scalar
viba	reactant boudnary vibrational structure (subsection 7.27)	
vibc	product boudnary vibrational structure (subsection 7.27)	
Outputs		
n	non-reactive probabilities (cell)	1 × length(iState)
r	reactive probabilities (cell)	1 × length(iState)
psi	scattering wave function (cell)	1 × length(iState)

7.19 nrsort2d_clebsch.m

This performs the same function as `nrsort2d.m` but assumes the reordering of the S-matrix elements as given in subsection 7.4.

7.20 polymesh.m

This subroutine takes any triangulation defined by nodal coordinates, `p` and a connectivity matrix, `t`, and a given polynomial order, `n`, as inputs and determines the new nodal locations, `pn`, for use in an order `n` FEM calculation. Furthermore, this subroutine generates the order `n` connectivity matrix, `tn`, which identifies all nodes that have been added within a given triangle (used by `fem2d.m`). `polymesh.m` also returns the 'linearized' connectivity matrix, `ton`, with allows the user to very easily visualize the results of **COL**SCAT2D . This subroutine is highly optimized and entirely general for any value of n.

Inputs	Description	Dimension
p	node coordinates	varies× 2
t	connectivity matrix	nt × 3
n	polynomial order of FEM basis functions	scalar (1-5)
Outputs		
pn	quadratic node coordinates	npn × 2
tn	quadratic connectivity matrix	nt × varies
tln	linearized quadratic mesh	4nt × 3
np	number of points in the polynomial mesh (`size(pn,1)`)	scalar

7.21 solver2d.m

The `solver2d.m` script solves the linear matrix system $\mathbf{Qu} = \mathbf{b}$, and returns the approximation to the wave function at the nodal points as well as the state-to-state scattering amplitudes.

Inputs	Description	Dimension
E	total energy in hartree	scalar
T	kinetic energy PN FEM matrix (subsection 7.9)	(np*ns) × (np*ns)
V	potential energy PN FEM matrix (subsection 7.9)	(np*ns) × (np*ns)
O	kinetic energy PN FEM matrix (subsection 7.9)	(np*ns) × (np*ns)
ba	indices on a-channel boundary (reactants)	1× na
bc	indices on c-channel boundary (products)	1× nc
M	mass structure (subsection 7.12)	
viba	reactant boudnary vibrational structure (subsection 7.27)	
vibc	product boudnary vibrational structure (subsection 7.27)	
iState	initial vibrational state of BC	scalar
iSurf	initial electronic surface of A	scalar
ns	number of electronic surfaces in calculation (one-state or two-state)	scalar
np	number of nodal points in the mesh	scalar
Outputs		
U	vector of unknowns (psi + S-matrix elements)	(np+na+nc)×le

7.22 solver2d_clebsch.m

The `solver2d.m` script solves the linear matrix system $\mathbf{Qu} = \mathbf{b}$, and returns the approximation to the wave function at the nodal points as well as the state-to-state scattering amplitudes. This version of the subroutine takes two additional parameters to build the extended system using a transformed reactant or product basis. (subsection 7.4)

7.23 stdint2d.m

This subroutine is the heart and soul of the fast, vectorized construction of the FEM matrices used in **COL**SCAT2D. The `stdint2d.m` subroutine takes the polynomial order, `n`, as its only input, and automatically generates the points of the standard triangle required to numerically determine the coefficients of the basis functions of polynomial order `n` for the standard triangle. With these basis functions in hand `stdint2d.m` uses an analytic expression for the integrals of a 2D polynomial over the standard triangle.

Inputs	Description	Dimension
n	polynomial order of basis functions	scalar
Outputs		
t	KE constructor matrix	$\text{nbf}^2 \times 3$
v	potential energy constructor matrix	$\text{nbf} \times \text{nbf}^2$
o	overlap constructor matrix	$1 \times \text{nbf}^2$

nbf = number of basis functions:: `nbf = sum(1:(n+1))`

We copy the entirety of the `stdint2d.m` code here to facilitate the discussion and analysis of this subroutine.

```matlab
function [T,V,O] = stdint2d(n)
% number of basis functions
nbf = (n+1)*(n+2)/2;

% additional points in mesh
[xm,ym] = meshgrid(1:(n-2),(n-2):-1:1);
xm = xm(tril(ones(n-2))>0)'/n;
ym =  ym(tril(ones(n-2))>0)'/n;

% concatenate all points
x = [0; 1; 0; (1:(n-1))'/n; zeros(n-1,1); ((n-1):-1:1)'/n; xm'];
y = [0; 0; 1; zeros(n-1,1); (1:(n-1))'/n; (1:(n-1))'/n; ym'];

% power of each nbf basis functions x^xpow*y^ypow
xpowmat = triu(kron(n:-1:0,ones(n+1,1)))';
ypowmat = bsxfun(@minus,(n:-1:0),xpowmat);
xpow = xpowmat(tril(ones(n+1))>0)';
ypow = ypowmat(tril(ones(n+1))>0)';

% if n = 3
% XY(i,:) [xi^3 xi^2yi xiyi^2 yi^3.... xi yi 1]
XY = bsxfun(@power,x,xpow).*bsxfun(@power,y,ypow);

% matrix of basis function coefficients
C = XY\eye(nbf);

% 2d polynomial integral rule: int(int(x^a*y^b,x,0,1-y),y,0,1) = a!b!/(a+b+2)!

k = bsxfun(@plus,xpow,xpow');
l = bsxfun(@plus,ypow,ypow');
X = factorial(k).*factorial(l)./factorial(k+l+2);

% overlap constructor matrix int phi_i phi_j
O = reshape((C'*X*C)',1,nbf^2);

% potential energy constructor matrix int phi_i phi_j phi_k
K = repmat(k,1,1,nbf)+reshape(kron(xpow,ones(nbf)),nbf,nbf,nbf);
L = repmat(l,1,1,nbf)+reshape(kron(ypow,ones(nbf)),nbf,nbf,nbf);
vij = factorial(K).*factorial(L)./factorial(K+L+2);
vv = zeros(nbf,nbf,nbf);
for m=1:nbf
    vv(:,:,m) = C'*cat(1,vij(:,:,m))*C;
end
V = reshape(reshape(vv,nbf^2,nbf)*C,nbf,nbf^2);

% KE contructor matrix T = int (M\del phi_i)(M del phi_j) Minv = [a b; c d] (see stiffnessn.m)
% dx/dy - partial derivative with respect to x/y matrix operators
dx = sparse((n+1)+(1:(n*(n+1)/2)),find(xpow>0),xpow(xpow>0),nbf,nbf);
dy = sparse((n+1)+(1:(n*(n+1)/2)),find(ypow>0),ypow(ypow>0),nbf,nbf);

Txx = (C'*dx')*X*(dx*C);
Txy = (C'*dx')*X*(dy*C);
Tyx = (C'*dy')*X*(dx*C);
Tyy = (C'*dy')*X*(dy*C);

T = [Txx(:) Txy(:)+Tyx(:) Tyy(:)];
```

Table 2: The generic polynomial basis functions and the vectors of the powers of x and y for each term in these expansions. The analytic expression of the integrals of these basis functions depend only on the powers of x and y.

N	generic basis function	xpow	ypow
1	$c_{10}x + c_{01}y + c_{00}$	[1 0 0]	[0 1 0]
2	$c_{20}x^2 + c_{11}xy + c_{02}y^2 + \ldots$	[2 1 0 1 0 0]	[0 1 2 0 1 0]
	$c_{10}x + c_{01}y + c_{00}$		
3	$c_{30}x^3 + c_{21}x^2y + c_{12}xy^2 + c_{03}y^3 + \ldots$	[3 2 1 0 2 1 0 1 0 0]	[0 1 2 3 0 1 2 0 1 0]
	$c_{20}x^2 + c_{11}xy + c_{02}y^2 + \ldots$		
	$c_{10}x + c_{01}y + c_{00}$		
4	$c_{40}x^4 + c_{31}x^3y + c_{22}x^2y^2 + c_{13}xy^3 + c_{04}y^4 \ldots$	[4 3 2 1 0 3 2 1 0 2 1 0 1 0 0]	[0 1 2 3 4 0 1 2 3 0 1 2 0 1 0]
	$c_{30}x^3 + c_{21}x^2y + c_{12}xy^2 + c_{03}y^3 + \ldots$		
	$c_{20}x^2 + c_{11}xy + c_{02}y^2 + \ldots$		
	$c_{10}x + c_{01}y + c_{00}$		

To start, this subroutine determines the points that must be added to the vertices (0,0), (1,0) and (0,1) to define the polynomial basis functions of the desired order. We choose to add points on a square mesh for convenience. The variables `xm` and `ym` store, respectively, the x and y locations of the interior points that must be added to the standard vertex points. Note: for polynomials orders 1 and 2 there are no points inside the standard triangle. All the points are then concatenated in the following way, x/y = [standard vertex points, edge points, mid points]. The following figure shows how these points are ordered for P4.

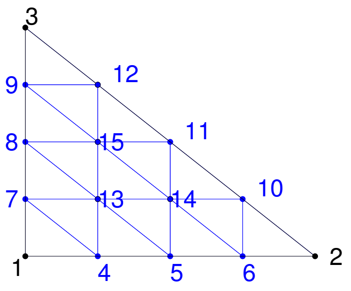


Figure 15: The points of the standard triangle (0,0), (1,0), and (0,1) are show in black. The points that must be added for polynomial order P4 are shown in blue. Also shown is the counting system used in **COL**SCAT2D which gives a consistent way to refer to the basis functions.

Once the additional points are known we then determine the basis functions themselves. To start, in lines 17-26 we define `xpow` and `ypow`, which describe the polynomials themselves. The table (2) should help elucidate their definition.

Given the points that will be used as the nodal points, i.e. `x` and `y`, and the powers of the basis function polynomials, `xpow` and `ypow`, the coefficients of the basis functions are determined by a simple matrix problem $\mathbf{XC} = \mathbf{1}$. For the P1 basis this has the following form

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In the case of P1 `x = [0 1 0]` and `y = [0 0 1]` which gives the solution

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Given the powers x and y of each term in the polynomial expansions shown in Table (2) and their coefficients we can easily determine the requisite integrals. Recall the analytic expression for the integral of $x^a y^b$ over the standard triangle

$$\int_0^1 \int_0^{1-y} x^a y^b \; dxdy = \frac{a!b!}{(a+b+2)!}$$

We can easily integrate any 2D polynomial over the standard triangle using this expression. As an example let us consider the overlap integrals for a second order polynomial basis. Let us define the 'constructor' overlap matrix as $\tilde{\mathbf{O}}$,

$$\tilde{O}_{ij} = \int_\Omega \tilde{\phi}_i \tilde{\phi}_j \; dxdy \tag{4}$$

where $\tilde{\phi}$ signifies a standard triangle basis function. We can expand these basis functions as follows

$$\tilde{\phi}_i(x,y) = \sum_{k,l}^{k+l \le P} c_{i,kl} \, x^k \, y^l \tag{5}$$

Substituting this expression into Eqn. (4) we have

$$\tilde{O}_{ij} = \int_\Omega \left(\sum_{k,l}^{k+l \le P} c_{i,kl} \, x^k \, y^l \right) \left(\sum_{k',l'}^{k'+l' \le P} c_{j,k'l'} \, x^{k'} \, y^{l'} \right) \; dxdy \tag{6}$$

Using the above analytic expression for the integral of a 2D polynomial over the standard triangle we can easily evaluate this expression. We can change this double indexed sum to a sum over a single index, m. For example consider the P3 basis functions and the indices shown in Table (3).

Table 3: An illustration of how to change from a double index k, l to a single index m for the standard triangle basis function polynomial terms.

	x^3	x^2y	xy^2	y^3	x^2	xy	y^2	x	y	1
k	3	2	1	0	2	1	0	1	0	0
l	0	1	2	3	0	1	2	0	1	0
m	1	2	3	4	5	6	7	8	9	10

Using this conversion to a single index we can write Eqn. (6) as follows

$$\tilde{O}_{ij} = \int_\Omega \left(\sum_{m}^{n_{bf}} c_{i,m} \, x^{k(m)} \, y^{l(m)} \right) \left(\sum_{m'}^{n_{bf}} c_{j,m'} \, x^{k(m')} \, y^{l(m')} \right) \; dxdy \tag{7}$$

Pulling the double summation outside of the

$$\tilde{O}_{ij} = c_{i,m} c_{j,m'} \sum_{m,m'} \int_\Omega x^{(k(m)+k(m'))} \, y^{(l(m)+l(m')} \; dxdy \tag{8}$$

and finally we can write Eqn. (8) as a matrix equation,

$$\tilde{O}_{ij} = \mathbf{C}^T \mathbf{X} \mathbf{C} \tag{9}$$

where the matrix elements of $\mathbf{X}$ are defined by the analytic expression for 2D polynomial integrals over the standard triangle given above, i.e.

$$X_{m,m'} = \frac{(k(m)+k(m'))!(l(m)+l(m'))!}{(k(m)+k(m')+l(m)+l(m')+2)!}$$

In the code this matrix is simplified using the `xpow` and `ypow` variables, defined on lines 39-40,

$$k_{ij} = xpow(i) + xpow(j)$$
$$l_{ij} = ypow(i) + ypow(j)$$

These matrices contain all possible powers of x and y for the product of two basis functions. Because the integrals only depend on the power of the polynomial terms we can simply call the `factorial.m` subroutine which acts element-wise. The last step is to multiply by the coefficient matrices on either side, done on line 41. Finally we reshape this matrix from dimension of $nbf \times nbf$ to $1 \times nbf^2$ for use in `fem2d.m`.

The potential energy matrix elements are computed in a similar fashion, however, we must evaluate

$$\tilde{V}_{ijk} = \int_{\Omega} \tilde{\phi}_i \tilde{\phi}_j \tilde{\phi}_k \; dxdy \tag{10}$$

We must simply create the 3D matrices for the powers of x and y, i.e.

$$K_{ijm} = xpow(i) + xpow(j) + xpow(m)$$
$$L_{ijm} = ypow(i) + ypow(j) + ypow(m)$$

These matrices are simple to evaluate using the element-wise `factorial.m` operation. A little more care is given in lines 49-51 to multiply by the coefficient matrices by concatenating along the 3rd index.

The kinetic energy matrix elements are slightly more complicated to evaluate. The reason for this is the presence of the ∇ operator. When we change coordinates from the mass-scaled Jacobi, (R, r), to the standard triangle coordinates, (x, y), the gradient operator transforms as

$$\boldsymbol{\nabla}_{Rr} = (\mathbf{M}^{-1})^T \boldsymbol{\nabla}_{xy}$$

where

$$\mathbf{M} = \begin{bmatrix} R_2 - R_1 & R_3 - R_1 \\ r_2 - r_1 & r_3 - r_1 \end{bmatrix}$$

We are therefore interested in the following integral over the standard triangle,

$$\tilde{T}_{ij} = \int_{\Omega} [(\mathbf{M}^{-1})^T \boldsymbol{\nabla} \tilde{\phi}_i]^T \cdot [(\mathbf{M}^{-1})^T \boldsymbol{\nabla} \tilde{\phi}_j] \; dxdy \tag{11}$$

$$\tag{12}$$

For now let us simply this expression and let $M^{-1} = [a\ b; c\ d]$. We will correct for this simplification in `stiffness.m`. We also have $\nabla \tilde{\phi}_i = [\partial_x \phi_i,\ \partial_y \phi_i]$. Substituting these two values we have

$$\tilde{T}_{ij} = \int_\Omega (a^2+c^2)\partial_x\phi_i\partial_x\phi_j + (ab+cd)(\partial_x\phi_i\partial_y\phi_j + \partial_y\phi_i\partial_x\phi_j) + (b^2+d^2)\partial_y\phi_i\partial_y\phi_j \, dxdy \quad (13)$$

$$(14)$$

There are four distinct integrals that must be considered. Taking the gradient of the basis functions defined over the standard triangle is quite easy because we have already calculated all the integrals we need to in the **X** matrix. We can very easily construct a `dx` and `dy` matrix that operate on the **C** matrix to yield the appropriate derivative terms. For example, consider the `dx` matrix for P2

$$[x^2, xy, y^2, x, y, 1]\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} = [2x, y, 0, 1, 0, 0]$$

Once the matrix form of the derivative operators are made in lines 50-51, the kinetic energy matrix elements are constructed on lines 53-56. Finally we store T in a slightly different format and the `stiffness.m` subroutine is used to add in the a, b, c and d term. Because these coefficients only depend on the nodal coordinates and their inclusion is easily vectorized.

7.24 tobond.m

The mesh for the finite-element is determined in terms of the mass-scaled Jacobi coordinates, while the potential energy surface input functions, (subsection 7.28), expect bond coordinates. To determine the value of the potential energy surfaces at the node points we first use `tobond.m` to convert the nodal coordinates from mass-scaled Jacobi to bond coordinates.

Inputs	Description	Dimension
R	mass-scaled Jacobi coordinates matrix	n × 2
m	mass vector [M_a, M_b, M_c]	1 × 3
Outputs		
U	bond coordinates matrix	n × 2

7.25 tojac.m

The `tojac.m` script converts bond coordinates to mass-scaled Jacobi coordinates. This is mainly used during the mesh generations routines.

Inputs	Description	Dimension
x	bond coordinate	m × 1
y	bond coordinate	m × 1
m	mass vector [M_a, M_b, M_c]	1 × 3
Outputs		
p	Jacobi coordinates	m × 2

7.26 trigradient.m

This subroutine is no longer supported as it is unstable for small mesh sizes, especially near the boundaries. Instead we include a interpolating data to a square mesh and using the built in gradient function.

7.27 vibfem.m

Once the mesh has been generated and the potential energy surface has been evaluated at each of the nodal points, we need to determine the vibrational energies and wave functions for the BC and AB molecules at the boundaries in the, respectively, reactant and product arrangements. This is done by the script `vibfem.m`.

The `vibfem.m` script is based on a 'PN' FEM solution to Schrödinger's equation in one-dimension. The 1d matrix elements used in this calculation are generated by `stdint1d.m` in a very similar fashion to those generated for the 2D problem in subsection 7.23. Eq. (12) in the referenced paper[?] can be converted to a FE system of matrix equations of the exact same form as Eq. (32). The only difference is that the basis functions and mesh used are 1-dimensional in the case of solving Eq. (12).

Inputs		
r	internuclear separation values (in bohr)	n × 1
v	the potential energy as a function of r (in hartree)	n × 1
mu	diatomic reduced mass	scalar
n	polynomial order of 1D basis functions used in FEM calculation	scalar
Outputs		
vib	vibrational wavefunctions and energies	
vib.wf	Each column is the normalized bound state wave function.	length(r) × length(r)
vib.e	The energy of each bound state wave function.	nStates × 1
vib.iwf	Intergrated product of wave functions times basis functions.	n × nStates

Here we show how to use `vibfem.m` to calculate the bound state wave functions for a harmonic approximation to the H_2 potential.

```
% assume a harmonic oscillator potential for H_2
mu = 1837/2; % reduced mass of H_2
omega = 0.020; % in hartrees
r = -2:0.01:2;
v = 0.5*mu*omega^2*r.^2;

% determine vibrational structure
vib = vibfem(r,v,mu,2);

% plot potential
figure(1)
plot(r,v,'r','linewidth',2);
axis([-2 2 -.01 .8])
ylabel('V_{harm}(r) ','fontsize',24)
xlabel('r_{HH} ','fontsize',24)
set(gca,'ytick',0:0.2:0.8)
set(gca,'fontsize',20)
```

```
% plot vibrational wave functions
figure(2)
plot(r,vib.wf(:,1:3),'linewidth',2)
ylabel('\chi(r_{HH})','fontsize',24)
xlabel('r_{HH}','fontsize',24)
set(gca,'ytick',-1.5:0.5:1.5)
set(gca,'fontsize',20)
legend('v=0','v=1','v=2')
```

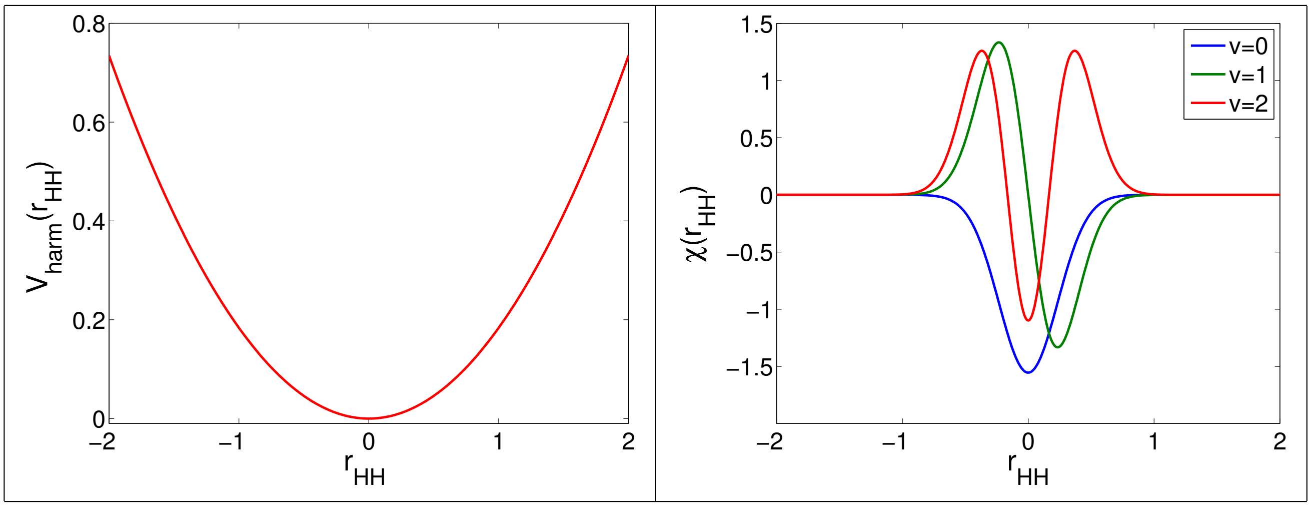


Figure 16: Plots of (left) a harmonic approximation to the potential curve of the H_2 molecule as a function of internuclear separation in bohr and (right) the wavefunctions for the first three states of the harmonic potential.

7.28 vinput

Inputs		
x	B-C separation distance (bond coordinate)	m × n
y	A-B separation distance (bond coordinate)	m × n
ba	indices on reactant boundary (subsection 7.2)	1 × na
bc	indices on product boundary (subsection 7.2)	1 × nc
Outputs		
v	potential energy evaluated at nodal points	m × n
va	potential energy along reactant boundary nodal points	na × ns
vc	potential energy along product boundary nodal points	nc × ns

To run **COL**SCAT2D the user must supply the name of a function to evaluate the potential energy surface over the desired range of internuclear separation coordinates (bond coordinates). The `vinput` function handle must be a string and the referred function must have four input values: the `'x'` and `'y'` coordinates which must be the bond coordinates U_{ab} and U_{bc}, respectively, and the boundary indices `ba` and `bc`. The `ba` and `bc` i indices identify which points in the mesh lie on the reactant and product boundaries, respectively. The potential input function should be single valued; when **COL**SCAT2D calls the `vinput.m` routine it will pass the x and y coordinates as column vectors and expect `vinput.m` to return a single column vector for v.

The potential input file should also be able to handle input matrices, i.e. if x and y are produced from a mesh grid of bond coordinates. This functionality is essential for the `meshgen.m` subroutine to function correctly. It is also useful to allow the potential input functionality to be dependent on the user. To analyze the potential outside of the **COL**SCAT2D main routine it is convenient to get the potential energy surface without the boundary indices. MATLAB makes it fairly east to achieve this functionality using the number of output arguments variable, `'nargout'`. The examples provided `vhh2.m`, `vfh2.m`, and `vfhcl_3body.m` all make use of `nargout` in the following way. If the user is only interested in the potential energy for extraneous analysis, e.g. for use in `mepath.m` or `mepbarrier.m`, then only a single output is requested from the `vinput` function. For example consider the following call for the H+H_2 potential in the vicinity of the barrier

```
% create a mesh grid to analyze the hh2 barrier
[x,y] = meshgrid(1.5:0.01:2,1.5:0.01:2);

% return only the potential energy surface
v = vhh2(x,y);

% plot the result
contour(x,y,v,20)
```

Only a single output variable, `v`, is requested from `vhh2.m`. The use of `nargout` within vhh2 allows the function to check how many outputs the user requested. It only uses the boundary indices, `ba` and `bc`, if the call to the subroutine expects three output functions. Compare the above call to the call that **COL**SCAT2D makes on `lines 19-21`, i.e.

```
%% spline potential at nodes and the center of each triangle
pb = tobond(p,m);
[v,va,vc] = feval(vinput,pb(:,1),pb(:,2),ba,bc);
```

Firstly, because `vinput` is a function handle we must use `feval` to evaluate the function. Secondly, because three outputs are expected, `v`, `va`, and `vc`, the function call must provide the boundary indices.

7.28.1 Two-state Potential Inputs

The difference between single-state and two-state potential inputs is that for a coupled, two-state calculation, the potential energy function in the hamiltonian is actually a matrix, not a single-valued function. For a given point in bond coordinate space, (x,y), **COL**SCAT2D assumes the potential energy has the following, symmetric form,

$$\mathrm{V}(\mathrm{x},\mathrm{y}) = \begin{bmatrix} \mathrm{V}_{11}(\mathrm{x},\mathrm{y}) & \mathrm{V}_{12}(\mathrm{x},\mathrm{y}) \\ \mathrm{V}_{21}(\mathrm{x},\mathrm{y}) & \mathrm{V}_{22}(\mathrm{x},\mathrm{y}) \end{bmatrix}$$

where V_{11}(x,y) is the electronic ground state and V_{22} is the first excited state. Given the input vectors, $\mathrm{x} = [\mathrm{x}_1;\ \mathrm{x}_2;\ \ldots;\ \mathrm{x}_\mathrm{m}]$ and $\mathrm{y} = [\mathrm{y}_1;\ \mathrm{y}_2;\ \ldots;\ \mathrm{y}_\mathrm{m}]$, **COL**SCAT2D will expect the v output to have the following form

$$v = \begin{bmatrix} V_{11}(x_1,y_1) & V_{12}(x_1,y_1) & V_{21}(x_1,y_1) & V_{22}(x_1,y_1) \\ V_{11}(x_2,y_2) & V_{12}(x_2,y_2) & V_{21}(x_2,y_2) & V_{22}(x_2,y_2) \\ \vdots & \vdots & \vdots & \vdots \\ V_{11}(x_m,y_m) & V_{12}(x_m,y_m) & V_{21}(x_m,y_m) & V_{22}(x_m,y_m) \end{bmatrix}$$

The potential coupling matrix, Av, an additional and optional parameter, can be used to convert from one potential basis to another, depending on the desired calculation.

For example, the F+HCl potential routine is based on ab initio data. The ab initio data was calculated for the electronic ground state, V_Σ and the first excited state, V_Π, as well as the spin-orbit constant which couples these two electronic states. These two electronic states are not experimentally measurable and instead we would like to change to the two state, definite J basis, which correspond to the J=1/2 and J=3/2 asymptotic states of the halogen.
To see how to use the Av parameter, let us consider the vfhcl_3body.m script.

```
% sigma and pi potentials using 3body fit
[mhf,mhcl,D] = importpotential('fhcl_parameters_3body_new.txt');

% v3 = [V_sig V_pi]
v3 = v3body(x,y,mhf,mhcl,D);

% spin orbit constant
vso = fhcl_so_approx(x,y,100);

% [vsig vpi vso] * Av ===> Av selects the basis
v = [v3 vso]*Av;
```

The first two commands in this script load parameters that were used in a 3-body expansion fit to ab initio data calculated by Jacek Klos. These parameters are then passed to v3body.m and the output variable, v3, has the following form v3 = $[V_\Sigma \; V_\Pi]$.

Next an approximation to the spin orbit constant is calculated. This approximation simply flips from the spin orbit constant of F to the spin orbit constant of Cl near the barrier. The last argument, '100', is an input parameter that dictates the sharpness of the flipping, larger values indicate a sharper transition.

The ab initio fits and the spin orbit constant are ultimately combined into one matrix

$$[\mathrm{v3\ vso}] = \begin{bmatrix} V_\Sigma(x_1,y_1) & V_\Pi(x_1,y_1) & V_{SO}(x_1,y_1) \\ V_\Sigma(x_2,y_2) & V_\Pi(x_2,y_2) & V_{SO}(x_2,y_2) \\ \vdots & \vdots & \vdots \\ V_\Sigma(x_m,y_m) & V_\Pi(x_m,y_m) & V_{SO}(x_m,y_m) \end{bmatrix}$$

We can use the Av parameter to couple these potential surfaces. The coupling between the Σ and Π states arises from the the spin-orbit interaction. The coupling between these two states can be written as

$$V_{\mathrm{eff}}^{\Sigma/\Pi}(x,y) = \begin{bmatrix} V_\Sigma(x,y) & 0 \\ 0 & V_\Pi(x,y) \end{bmatrix} + \begin{bmatrix} 0 & -\sqrt{2}V_{SO}(x,y) \\ -\sqrt{2}V_{SO}(x,y) & V_{SO}(x,y) \end{bmatrix}$$

The definite-J basis is the basis whose potential energy matrix is asymptotically diagonal. In terms of, $\mathrm{V_\Sigma}$, $\mathrm{V_\Pi}$, and $\mathrm{V_{SO}}$, we can write the definite-J basis as follows

$$\mathrm{V^J_{eff}(x,y)} = \begin{bmatrix} \mathrm{V_{sum}(x,y) - V_{SO}(x,y)} & \mathrm{V_{dif}(x,y)} \\ \mathrm{V_{dif}(x,y)} & \mathrm{V_{sum}(x,y) + 2V_{SO}(x,y)} \end{bmatrix}$$

where $\mathrm{V_{sum}(x,y) = [2V_\Sigma(x,y) + V_\Pi(x,y)]/3}$ and $\mathrm{V_{dif}(x,y)} = \sqrt{2}\mathrm{[V_\Sigma(x,y) - V_\Pi(x,y)]/3}$. With the following choice for the potential coupling matrix, Av, it can be seen that the desired format for v is achieved by the matrix multiplication shown in the last line of the snippet from vfhcl_3body.mw

$$\mathrm{Av} = \begin{bmatrix} 2/3 & \sqrt{2}/3 & \sqrt{2}/3 & 2/3 \\ 1/3 & -\sqrt{2}/3 & -\sqrt{2}/3 & 1/3 \\ -1 & 0 & 0 & 2 \end{bmatrix}$$

Lastly, the expected outputs of the reactant and product potentials, va and vc, are one column for every state, i.e. the diagonal elements of the potential that lie on the boundary.

8 Future improvements

List possible sources of improvements for the most adventurous users:

- hp - adaptivity: an automatic routine to refine the mesh size and/or polynomial order of the basis functions of a given triangle. this allows for ultra fast convergence
- for higher order polynomial basis functions add a constraint such that the gradient is smooth across triangle edges. this may be easily(?) added as an extra set of linear equations to the QU=b equation in solver.m
- use the solution at one energy to precondition the solution at the next energy (not feasible if parfor loop is used)
- add 3-d/rotational dynamics