PyDiatomic Documentation

Release 0.1

PyDiatomic team

November 30, 2016

Contents

Start by having a look at the README.

Contents:

CHAPTER 1

PyDiatomic README

1.1 Introduction

PyDiatomic solves the time-independent coupled-channel Schroedinger equation using the Johnson renormalized Numerov method [1]. This is very compact and stable algorithm.

The code is directed to the computation of photodissociation cross sections for diatomic molecules. The coupling of electronic states results in transition profile broadening, line-shape asymmetry, and intensity sharing. A coupled-channel calculation is the only correct method compute the photodissociation cross-section.

1.2 Installation

PyDiatomic requires Python 3.5 (*), numpy and scipy. If you don't already have Python, we recommend an "all in one" Python package such as the Anaconda Python Distribution, which is available for free.

Download the latest version from github

```
git clone https://github.com/stggh/PyDiatomic.git
```

cd to the PyDiatomic directory, and use

```
python3 setup.py install --user
```

Or, if you wish to edit the PyAbel source code without re-installing each time

```
python3 setup.py develop --user
```

(*) due to the use of infix matrix multiplication `@`. To run with python < 3.5, replace `A @ B` with `np.dot(A, B)` in `cse.py` and `expectation.py`.

1.3 Example of use

PyDiatomic has a wrapper classes `cse.Cse()` and `cse.Xs()`

`cse.Cse()` set ups the CSE problem (interaction matrix of potential energy curves, and couplings) and solves the coupled channel Schroedinger equation for an initial guess energy.

Input parameters may be specified in the class instance, or they will be requested if required.

```
import cse
X = cse.Cse()    # class instance
# CSE: reduced mass a.u. [O2=7.99745751]:    # requested parameters
# CSE: potential energy curves [X3S-1.dat]:
X.solve(800)    # solves TISE for energy ~ 800 cm-1
# attributes
# X.Bv                  X.mu                  X.set_mu
# X.R                   X.node_count          X.solve
# X.VT                  X.pecfs               X.vib
# X.cm                  X.rot                 X.wavefunction
# X.energy              X.rotational_constant
# X.limits              X.set_coupling
X.cm
# 787.3978354211097
X.vib
# 0
```

`cse.Xs()` evaluates two couple channel problems, for an intitial and final set of coupled channels, to calculate the photodissociation cross section.

```
import numpy as np
import cse
Y = cse.Xs()
# CSE: reduced mass a.u. [O2=7.99745751]:
# CSE: potential energy curves [X3S-1.dat]:
# CSE: potential energy curves [X3S-1.dat]: B3S-1.dat, E3S-1.dat
# CSE: coupling B3S-1.dat <-> E3S-1.dat cm-1 [0]? 4000
# CSE: dipolemoment filename or value B3S-1.dat <- X3S-1.dat : 1
# CSE: dipolemoment filename or value E3S-1.dat <- X3S-1.dat : 0
Y.calculate_xs(transition_energy=np.arange(110, 174, 0.1), eni=800)
# attributes
# Y.calculate_xs  Y.gs            Y.set_param     Y.xs
# Y.dipolemoment  Y.nopen         Y.us            Y.wavenumber
# and those associated with the initial and final states
#
# Y.gs.Bv                 Y.gs.mu                   Y.gs.set_mu
# Y.gs.R                  Y.gs.node_count           Y.gs.solve
# Y.gs.VT                 Y.gs.pecfs                Y.gs.vib
# Y.gs.cm                 Y.gs.rot                  Y.gs.wavefunction
# Y.gs.energy             Y.gs.rotational_constant
# Y.gs.limits             Y.gs.set_coupling
#
# Y.us.R                  Y.us.node_count           Y.us.set_coupling
# Y.us.VT                 Y.us.pecfs                Y.us.set_mu
# Y.us.limits             Y.us.rot                  Y.us.solve
# Y.us.mu                 Y.us.rotational_constant
```

A simple $^3\Sigma_u^- \leftrightarrow {}^3\Sigma_u^-$ Rydberg-valence coupling in O_2

```
import numpy as np
import cse
import matplotlib.pyplot as plt

Z = cse.Xs('O2', VTi=['X3S-1.dat'], VTf=['B3S-1.dat', 'E3S-1.dat'],
           coupf=[4000], dipolemoment=[1, 0],
           transition_energy=np.arange(110, 174, 0.1), eni=800)

plt.plot(Z.wavenumber, Z.xs*1.0e16)
plt.xlabel("Wavenumber (cm$^{-1}$)")
```

```
plt.ylabel("Cross section ($10^{-16}$ cm$^{2}$)")
plt.axis(ymin=-0.2)
plt.title("O$_{2}$ $^{3}\Sigma_{u}^{-}$ Rydberg-valence interaction")
plt.savefig("RVxs.png", dpi=75)
plt.show()
```

Fig. 1.1: Example calculated cross section

See also *examples/example_O2xs.py* and *example_rkr.py*:

1.3.1 Rotation

```
import cse

X = cse.Cse('O2', VT=['X3S-1.dat'])  # include path to potential curve
X.solve(900, rot=0)
X.cm
# 787.3978354211097
X.Bv
# 1.4376793638070153
X.solve(900, 20)
X.cm
# 1390.369249612629
# (1390.369-787.398)/(20*21) = 1.4356
```

1.4 Timing

Each transition energy solution to the coupled-channel Schroedinger equation is a separate calculation. PyDiatomic uses `multiprocessing` to perform these calculations in parallel, resulting in a substantial reduction in execution time on multiprocessor systems. e.g. for `example_O2xs.py`:

machine	GHz	CPU(s)	time (sec)
Xenon E5-2697	2.6	64	6
i7-6700	3.4	8	17
Macbook pro i5	2.4	4	63
raspberry pi 3	1.35	4	127

1.5 Documentation

PyDiatomic documentation is available at readthedocs.

1.6 Historical

PyDiatomic is a Python implementation of the Johnson renormalized Numerov method. It provides a simple introduction to the profound effects of channel-coupling in the calculation of diatomic photodissociation spectra.

More sophisticated C and Fortran implementations have been in use for a number of years, see references below. These were developed by Stephen Gibson (ANU), Brenton Lewis (ANU), and Alan Heays (ANU and Leiden).

1.7 Reference

[1] B.R. Johnson "The renormalized Numerov method applied to calculating the bound states of the coupled-channel Schroedinger equation" J. Chem. Phys. 69, 4678 (1978)

[2] B.R. Lewis, S.T. Gibson, F. T. Hawes, and L. W. Torop "A new model for the Schumann-Runge bands of O2" Phys. Chem. Earth(C) 26 519 (2001)

[3] A. N. Heays "Photoabsorption and photodissociation in molecular nitrogen, PhD Thesis (2011)

1.8 Citation

If you find PyDiatomic useful in your work please consider citing this project.

CHAPTER 2

PyDiatomic package

CHAPTER 3

Examples

Contents:

3.1 Example: example_O2xs.py

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pylab as plt
import time

import cse

evcm = 8065.541   # conversion factor eV -> cm-1

d = 'potentials/'  # directory name

wavelength = np.arange(110, 174.1, 0.1)  # nm

# initialize CSE problem - any missing essential parameters are requested
# mu - reduced mass
# eni - initial state guess energy
# VTI - initial state(s)
# VTf - final coupled states
# coupf - homogeneous coupling
# dipolemoment - transition moments

X = cse.Xs(mu='O2', VTi=[d+'X3S-1.dat'], eni=800,
                    VTf=[d+'B3S-1.dat', d+'3P1.dat', d+'E3S-1.dat',
                         d+'3PR1.dat'],
                    coupf=[40, 4000, 0, 0, 7000, 0],
                    dipolemoment=[1, 0, 0, 0.3])
                    #, transition_energy=wavelength)  # <--- alternative direct call

print("CSE: calculating cross section speeded by Python multiprocessing",
      " Pool.map")
print("     from {:.2f} to {:.2f} in {:.2f} nm steps ... ".
      format(wavelength[0], wavelength[-1], wavelength[1]-wavelength[0]))

t0 = time.time()
X.calculate_xs(transition_energy=wavelength)
print("CSE: ...  in {:.2g} seconds".format(time.time()-t0))
```

```
print('CSE: E(v"={:d}) = {:.2f} cm-1, {:.3g} eV'.format(X.gs.node_count(),
                                                        X.gs.cm, X.gs.energy))

# graphics ---------------------------------------
ax0 = plt.subplot2grid((2, 4), (0, 0), colspan=2, rowspan=2)
ax1 = plt.subplot2grid((2, 4), (0, 2), colspan=2, rowspan=2)

X.wavenumber /= 1.0e4
X.total = np.zeros_like(X.wavenumber)
for j in range(X.nopen):
   X.total[:] += X.xs[:, j]
   if X.us.pecfs[j][-7] == 'S':
       ax0.plot(X.xs[:, j], X.wavenumber, label=X.us.pecfs[j], color='b')
   else:
       ax0.plot(X.xs[:, j], X.wavenumber, label=X.us.pecfs[j], color='r',
                                                                 ls='--')

#ax0.plot(X.total, X.wavenumber, ls='-', color='gray', label='total', alpha=0.3)
ax0.legend(loc=0, frameon=False, fontsize=10)
ax0.set_ylabel("wavnumber ($10^4$cm$^{-1}$)")
ax0.set_xlabel("cross section (cm$^{2}$)")
ax0.axis(xmin=1.5e-17, xmax=-0.1e-17, ymin=4, ymax=10)
ax0.set_title("photodissociation cross section", fontsize=12)

for j, pec in enumerate(X.gs.pecfs):
   ax1.plot(X.gs.R, X.gs.VT[j, j]*evcm, color='k', label=pec)

for j, pec in enumerate(X.us.pecfs):
   if X.us.pecfs[j][-7] == 'S':
       ax1.plot(X.us.R, X.us.VT[j, j]*evcm, 'b', label=pec)
   else:
       ax1.plot(X.us.R, X.us.VT[j, j]*evcm, 'r--', label=pec)

ax1.annotate('$X{}^{3}\Sigma_{g}^{-}$', (0.6, 55000), color='k')
ax1.annotate('$B{}^{3}\Sigma_{u}^{-}$', (1.7, 55000), color='b')
ax1.annotate('$E{}^{3}\Sigma_{u}^{-}$', (0.9, 72000), color='b')
ax1.annotate('${}^{3}\Pi$', (1.34, 65000), color='r')


ax1.set_title("diabatic PECs", fontsize=12)
ax1.axis(xmin=0.5, xmax=2, ymin=40000+X.gs.cm, ymax=100000+X.gs.cm)
ax1.set_xlabel("R ($\AA$)")
#ax1.set_ylabel("V (eV)")
ax1.axes.get_yaxis().set_visible(False)

plt.suptitle("example_O2xs.py", fontsize=12)

plt.savefig("data/example_O2xs.png", dpi=100)
plt.show()
```

3.2 Example: example_rkr.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from __future__ import absolute_import
```

```
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

"""
  Rydberg-Klein-Rees evaluation of a potential energy curve from spectroscopic constants

  Stephen.Gibson@anu.edu.au
  2016
"""

import cse
import numpy as np
import scipy.constants as const
from scipy.interpolate import splrep, splev
import matplotlib.pyplot as plt
import sys

fn = input("RKR: Spectroscopic constants filename [data/GB.dat]: ")
fn = 'data/GB.dat' if fn=='' else fn

try:
    vv, Gv, Bv = np.loadtxt(fn, unpack=True)
except FileNotFoundError:
    print("RKR: file '{:s}' not found".format(fn))

# reduced mass - see Huber+Herzberg - default is O2
mu = input("RKR: Molecule reduced mass [7.99745751]: ")
mu = 7.99745751 if mu=='' else float(mu)

# De
De = input("RKR: De [42021.47 cm-1]: ")
De = 42021.47 if De=='' else float(De)

# outer limb extension
limb = input("RKR: Outer-limb LeRoy(L) or Morse(M) [L]: ")
if limb=='': limb='L'

R, PEC, RTP, PTP = cse.rkr(mu, vv, Gv, Bv, De, limb, dv=0.1,
                           Rgrid=np.arange(0.005, 10.004, 0.005))

data = np.column_stack((R.T, PEC.T))
np.savetxt("data/RKR.dat",data)
print("RKR: potential curve written to 'data/RKR.dat'")

plt.plot(R, PEC, RTP[::10], PTP[::10], 'o')
plt.axis(xmin=0.5, xmax=5, ymin=-0.1, ymax=10)
plt.xlabel("R($\AA$)")
plt.ylabel("E(eV)")
plt.savefig("data/example_rkr.png", dpi=100)
plt.show()
```

CHAPTER 4

Johnson Renormalization

[B. R. Johnson JCP **69**, 4678 (1978)]

$$\mathbf{T}_n = -\frac{2(\Delta R)^2\mu}{12\hbar^2}(E\mathbf{I} - \mathbf{V}_n)$$

$$\left.\begin{array}{l}\mathbf{F}_n = \mathbf{W}_n\boldsymbol{\chi}_n = [\mathbf{I} - \mathbf{T}_n]\chi \\ \mathbf{F}_{n+1} - \mathbf{U}_n\mathbf{F}_n + \mathbf{F}_{n-1} = \mathbf{0} \\ \mathbf{R}_n = \mathbf{F}_{n+1}\mathbf{F}_n^{-1}\end{array}\right\} \Rightarrow \begin{array}{l}\mathbf{R}_n = \mathbf{U}_n - \mathbf{R}_{n-1}^{-1} \\ \mathbf{U}_n = 12\mathbf{W}_n^{-1} - 10\mathbf{I}\end{array}$$

$$\mathbf{V} = \begin{pmatrix} V_{00} & V_{01} & V_{02} & \dots & V_{0n} \\ & V_{11} & V_{12} & \dots & V_{1n} \\ & & \ddots & & \vdots \\ & & & \ddots & V_{nn} \end{pmatrix}$$

V_{ii} diabatic potential energy curves, $V_{ij\neq i}$

off-diagonal coupling terms [H. Lefebvre Brion and R. W. Field table 2.2 page 39.]

$\Delta\Omega = 0$ homogeneous
$\Delta\Omega = \pm 1$ heterogenous - J dependent.

4.1 Outward Solution

$$\begin{array}{ll}\mathbf{R}_n = \mathbf{U}_n - \mathbf{R}_{n-1}^{-1} & n = 1 \rightarrow m \text{ with } \mathbf{R}_0^{-1} = 0 \\ \mathbf{F}_n = \mathbf{R}_n^{-1}\mathbf{F}_{n+1} & n = m \rightarrow 0 \text{ with } \mathbf{F}_\infty = \mathbf{W}_\infty\boldsymbol{\chi}_\infty\end{array}$$

Except when $|\mathbf{R}_n| \sim 0$ then

$\mathbf{R}_n^{-1}$ is not well defined.

Use $\mathbf{F}_n = \mathbf{U}_{n+1}\mathbf{F}_{n+1} - \mathbf{F}_{n+2}$

4.2 Inward Solution (ˆ - matrices)

$$\begin{array}{ll}\hat{\mathbf{R}}_n = \mathbf{U}_n - \hat{\mathbf{R}}_{n+1}^{-1} & n = \infty \to m \text{ with } \hat{\mathbf{R}}_\infty^{-1} = 0 \\ \mathbf{F}_n = \hat{\mathbf{R}}_n^{-1}\mathbf{F}_{n-1} & n = m \to \infty \text{ with } \mathbf{F}_0 = \mathbf{W}_0\boldsymbol{\chi}_0\end{array}$$

Except when $|\mathbf{R}_n| \sim 0$ **then** $\mathbf{R}_n^{-1}$ is not well defined.

Use $\mathbf{F}_n = \mathbf{U}_{n-1}\mathbf{F}_{n-1} - \mathbf{F}_{n-2}$

4.3 Multiple Open Channels

n_{open} linearly independent solutions:

$$\mathbf{R}(R=\infty) = \begin{pmatrix} 1 & 0 & \dots & 0 & \\ 0 & 1 & \dots & 0 & \\ \vdots & \vdots & \ddots & \dots & \vdots \\ 0 & 0 & \dots & 1 & \end{pmatrix} \to \text{CSE} \to \boldsymbol{\chi}(R) = \begin{pmatrix} \chi_{00} & \chi_{01} & \chi_{02} & \cdots & \chi_{0N_{\text{open}}} \\ \chi_{10} & \chi_{11} & \chi_{12} & \cdots & \\ \vdots & \vdots & \vdots & & \vdots \\ \chi_{N_{\text{total}}0} & & & \cdots & \chi_{N_{\text{total}}N_{\text{open}}} \end{pmatrix}$$

4.3.1 Normalization

[Mies - Molecular Physics **14**, 953 (1980).]

$\boldsymbol{\chi} = \mathbf{JA} + \mathbf{NB}$

$\mathbf{F}^K = \boldsymbol{\chi}\mathbf{A}^{-1} = \mathbf{J} + \mathbf{NK}$

where

$\mathbf{K} = \mathbf{BA}^{-1} = \mathbf{U}\tan\boldsymbol{\xi}\hat{\mathbf{U}}.$

Physical solution

$\mathbf{F}^S = \mathbf{F}^k\mathbf{U}\cos\boldsymbol{\xi}e^{\mathrm{i}\boldsymbol{\xi}}\hat{\mathbf{U}} = \mathrm{i}e^{-\mathrm{i}\mathbf{k}R} - \mathrm{i}e^{\mathrm{i}\mathbf{k}R}\mathbf{S}$

Determine matrices , by energy normalization of each wavefunction.

$\chi_{ij} = \left(\frac{2\mu}{\hbar^2\pi}\right)^{\frac{1}{2}} \frac{1}{\sqrt{k}} \left[J_i a_{ij} + N_i b_{ij}\right]$ for potential i.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search