pySpawn User Manual

Version 0.0

by Benjamin G. Levine

Michigan State University

2017

Contents

[2 Introduction 3](#_Toc496190930)

[2.1 What Is pySpawn? 3](#_Toc496190931)

[2.2 What Are The Primary Goals of pySpawn? 3](#_Toc496190932)

[2.3 Features 3](#_Toc496190933)

[2.4 Citing pySpawn 3](#_Toc496190934)

[2.5 License 3](#_Toc496190935)

[2.6 Acknowledgements 4](#_Toc496190936)

[3 Obtaining and Installing PySpawn 5](#_Toc496190937)

[3.1 Obtaining PySpawn 5](#_Toc496190938)

[3.2 Dependencies 5](#_Toc496190939)

[3.3 Installing PySpawn 5](#_Toc496190940)

[4 Abbreviations Used in this Manual 6](#_Toc496190941)

[5 Guided Tour of Your First PySpawn Simulation 7](#_Toc496190942)

[5.1 Building Hessian File for Initial Condition Sampling 7](#_Toc496190943)

[5.2 AIMS Simulation Script 7](#_Toc496190944)

# Introduction

## What Is pySpawn?

PySpawn is an implementation of ab initio multiple spawning (AIMS) in python 2.7.

## What Are The Primary Goals of pySpawn?

PySpawn is designed with the following goals in mind:

* The core of pySpawn is designed to be small and thus easy to maintain.
* PySpawn is designed to be extensible. At present, the following functionality can, in principle, be implemented without modification of existing code:
  + Interfaces to additional electronic structure codes/methods
  + Quantum mechanical Hamiltonians (e.g. adiabatic representation vs. diabatics vs. diabatized Gaussians on adiabatic surfaces, interpolated vs. analytical derivative couplings, spin orbit Hamiltonian, explicit light field)
  + Quantum mechanical integrators
  + Classical mechanical integrators
* PySpawn is designed with large, shared computer resources in mind. Specifically, pySpawn is designed to facilitate
  + Subdivision of large jobs into small chunks
  + Restart
  + Parallelization

## Features

* Runs FMS/AIMS jobs in the adiabatic representation
* Derivative couplings computed by norm preserving interpolation (NPI)
* PySpawn interfaces to a development version of TeraChem via a protobuf interface

## Citing pySpawn

If you use pySpawn please cite the following:

“pySpawn is a nonadiabatic molecular dynamics software package written by Benjamin G. Levine”

This citation will eventually be replaced by a true publication.

## License

PySpawn is distributed under the MIT License:

Copyright (c) 2017 Benjamin G. Levine

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## Acknowledgements

* The protobuf interface was developed by Stefan Seritan in the group of Todd J. Martínez at Stanford University.
* Many thanks to B. Scott Fales, Todd J. Martínez, and Stefan Seritan for their contributions and input.

# Obtaining and Installing PySpawn

## Obtaining PySpawn

PySpawn can be obtained from github: <https://github.com/blevine37/pySpawn17>.

## Dependencies

PySpawn requires the following dependencies:

* numpy
* h5py
* tcpb (TeraChem protobuf interface; required to run with TeraChem only)
* google (required by tcpb for TeraChem interface only)
* protobuf (required by tcpb for TeraChem interface only)

These can easily be installed using pip:

pip install numpy h5py google protobuf

## Installing PySpawn

Having obtained pySpawn and installed the necessary dependencies, cd to the top directory and run

python setup.py install

# Abbreviations Used in this Manual

Various abbreviations will be used throughout this manual:

AIMS ab initio multiple spawning

a.u. atomic units

FMS full multiple spawning

NPI norm-preserving interpolation

PES potential energy surface

TBF trajectory basis function

# Guided Tour of Your First PySpawn Simulation

This chapter will trace the example calculation in examples/ethylene\_sacasscf. This job simulates the dynamics of ethene on the SA2-CAS(2/2)/6-31G potential energy surface. A single initial nuclear basis function is initiated, with its initial position and momentum sampled from the ground state vibrational Wigner distribution computed in the harmonic approximation. This section will not detail all the inner workings of the code, but instead will highlight

* the series of steps needed to run a simulation with pySpawn
* the input scripts
* the output structure

A more detailed description of the workflow, input, and analysis module will be presented in subsequent chapters.

## Building Hessian File for Initial Condition Sampling

The first step in any AIMS simulation is choosing the initial conditions. One popular choice, used here, is to sample the average position and momentum of the initial TBF from the vibrational Wigner distribution, computed in the harmonic approximation. This requires three pieces of information:

* the ground state minimum structure
* the Hessian (2nd derivative) matrix of PES at the ground state minimum
* the masses of the atoms

Before beginning, one must compute the minimum and Hessian and store them in pySpawn’s preferred format: hessian.hdf5. We will assume that the reader can optimize the ground state minimum energy geometry of their chosen molecule. The script build\_hessian.py takes this structure, drives TeraChem to compute the Hessian matrix, and then stores both the geometry and Hessian in hessian.hdf5.

The script is self-explanatory. There are a few important points to keep in mind:

* **Most importantly, if you wish to recompute the Hessian you must remove any existing hessian.hdf5 files!**
* It is your responsibility to make sure that the structure that is inputted is the desired ground state minimum structure.
* The final hessian.hdf5 file contains both the geometry and Hessian matrix. When sampling the initial conditions, both the geometry and Hessian will be drawn from this file.
* If the job should die before the hessian is completed, it can be restarted trivially by running this same script. The restarted job will read the existing hessian.hdf5 file and pick up where it left off.

## AIMS Simulation Script

With hessian.hdf5 in hand, we are ready to run an AIMS simulation of ethene. The python script that does this is start\_c2h4.py. This first section of this script defines several important parameters of the simulation:

* seed is the random number seed (for Wigner initial conditions).
* clas\_prop defines the classical propagator. vv opts for velocity Verlet integration.
* qm\_prop defines the propagator of the quantum amplitudes (the expansion coefficients in the AIMS wave functions). rk2 opts for adaptive second-order Runge-Kutta.
* qm\_ham defines the Hamiltonian used to propagate these amplitudes. adiabatic opts for AIMS in the adiabatic basis with NPI time-derivative couplings.
* potential defines the means by which the electronic structure and PES will be computed. terachem\_cas opts for on-the-fly calculation at a CAS level of theory using TeraChem. This option will work for SA-CASSCF, FOMO-CASCI, or CISNO-CASCI.
* t0 is the initial time of the simulation (in a.u.).
* ts is the time step of the simulation (in a.u.).
* tfinal is the final time of the simulation (in a.u.).
* numdims is dimensionality of the system (3 \* number of atoms).
* numstates is the number of electronic states in the calculation.

The definition of these scalar and string variables is followed by the creation of three dictionary objects:

* tc\_options contains options to be passed to TeraChem. Most of these options are documented in the TeraChem documentation, and thus are not discussed here. Exceptions:
  + atoms is a list of the abbreviations for the atoms in the molecule, in order.
  + cas\_energy\_labels is a list of tuples defining the states whose energies will be passed from TeraChem to pySpawn. The first element of each tuple is the state index (ground state is 0, first excited state is 1…) while the second element is the spin multiplicity (singlet is 1…)
* traj\_params contains parameters of the individual TBFs:
  + time is the initial time of the simulation.
  + timestep defines the time step for classical propagation.
  + maxtime defines the maximum time to which each TBF will be propagated.
  + spawnthresh defines the coupling above which spawning will be triggered.
  + istate is the index of the electronic state to which this TBF belongs (0 is the ground states…)
  + widths is a numpy array containing the widths, α, associated with each degree of freedom (in units of bohr-2).
  + atoms contains the list of atom labels (and should be identical to atoms in tc\_options in all normal cases).
  + masses is a numpy array containing the masses associated with each degree of freedom (in a.u., i.e. electron masses).
  + tc\_options is the dictionary structure defined above which contains options to be passed to TeraChem.
* sim\_params contains parameters of the entire simulation, including the quantum propagation:
  + quantum\_time is the initial time of the simulation (and should be identical to time in the traj\_params object in all normal cases).
  + timestep is the time step for quantum propagation (and should be identical to timestep in the traj\_params object in all normal cases).
  + max\_quantum\_time is the simulation time at which quantum propagation should stop (and should be identical to maxtime in the traj\_params object in all normal cases).
  + qm\_amplitudes is the initial quantum amplitudes.
  + qm\_energy\_shift is an energy shift applied to the diagonal of the Hamiltonian. For numerical convenience, these diagonals should be roughly zero.

Other options for traj\_params and sim\_params are documented in a subsequent chapter.