Grow: User Guide

Grow: a program suite for growing Potential Energy Surfaces

Version 2.2

This User Guide was written by Deborah Crittenden[‡], with some reproduction of the Grow 1.0 User Guide written by Michael Collins. The Grow 2.2 suite of programs has been developed by Michael Collins, Meredith Jordan, Keiran Thompson, Ryan Bettens, Alexander Duncan and Deborah Crittenden

[‡] e-mail: d.crittenden@chem.usyd.edu.au

Table of Contents

1. What's new in Grow 2.2?

Grow 2.2 is based upon the original Grow 1.0 suite of programs developed by Meredith Jordan, Keiran Thompson, Ryan Bettens, Alexander Duncan and Michael Collins. A number of significant changes have been made to the original code. The major new features include:

- addition of a Quantum Diffusion Monte Carlo (QDMC) module for calculation of the lowest rovibrational eigenstate of a system
- python scripts to control the growing process, which replace the shell scripts in the original version. This makes the Grow package more robust and gives it greater portability.
- the ability to use the quantum chemistry package Aces2 as a source of *ab initio* data, in addition to Gaussian. (Note: there is a known bug in the Aces code that calculates numerical second derivatives from first derivatives)
- the ability to interpolate potential energy surfaces from density functional data, using any exchange or correlation functional implemented in Gaussian03
- rearranged (modular) input files
- a second collection of python scripts to control the growing process, designed to function on machines with queuing systems (NQS, PBS).

These additions and improvements have been implemented by Deborah Crittenden and Keiran Thompson. As Grow 2.2 is significantly different to Grow 1.0, this user manual is written as stand-alone documentation, and some information from the Grow 1.0 manual may be reproduced here.

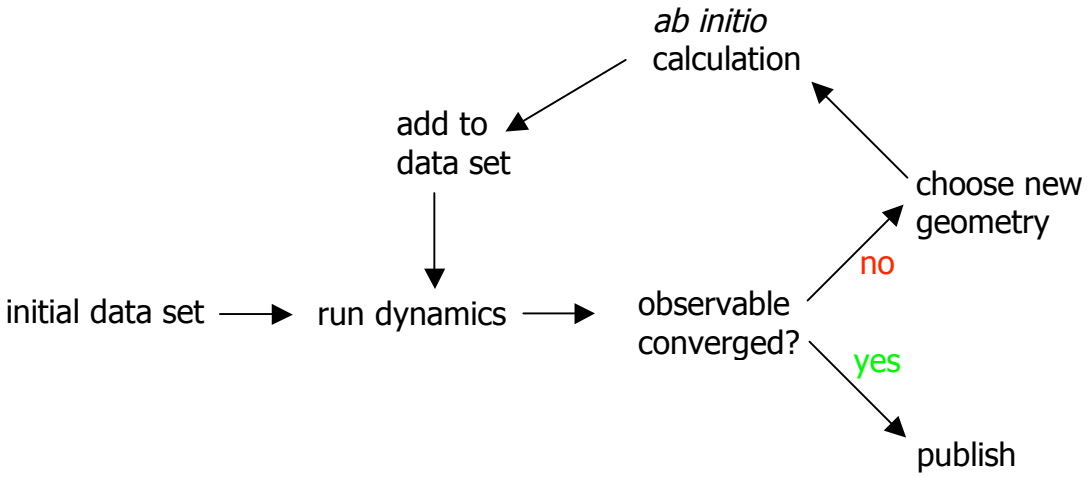
2. Conceptual overview

a. What is the Grow 2.2 package?

The Grow 2.2 package is a collection of scripts and programs that allow the user to construct molecular potential energy surfaces for either unimolecular/bimolecular reactions or bound-state systems. Classical trajectory simulations are used to sample configuration space while constructing reactive surfaces while Quantum Diffusion Monte Carlo (QDMC) simulations are used for bound-state systems. Classical trajectory simulations can be carried out on a reactive surface to calculate reaction cross sections, observable properties of the products (such as angular momentum and vibrational energy distributions), and to explore the reaction mechanism. QDMC simulations can be carried out to calculate the lowest energy rovibrational state of a bound-state system, and to calculate the exact anharmonic zero point energy.

b. How is a potential energy surface constructed?

The potential energy surface is constructed as an interpolation of *ab initio* data. The algorithm requires *ab initio* calculation of the energy, energy gradient and second derivatives at a number of molecular configurations. The number of such calculations required depends on the size of the system, the quantity being calculated and the accuracy required of the potential energy surface. However, it is reasonable to expect that you will need to carry out a minimum of around a thousand *ab initio* calculations. The individual configurations and their associated energy information will be referred to as 'data points'. The collection of data points that define the potential energy surface will be referred to as 'the PES data set'. The Grow package is designed to produce the most accurate possible PES for the least number of *ab initio* calculations by carefully selecting the location of the data points from configurations accessed during a trajectory simulution (for reactive surfaces) or QDMC simulation (for bound-state surfaces), as these represent the chemically relevant regions of configuration space. The algorithm for constructing the PES is illustrated diagrammatically below:



A detailed description of the theory and methodology underlying the Grow package can be found in the references at the end of this chapter. A brief description is given in Appendix 1. A simple (but informative) one-dimensional illustration of the interpolation scheme at work is provided in Appendix 2.

c. References

1. J. Ischtwan and M.A. Collins, *J. Chem. Phys.* **100**, 8080 (1994)

2. M.J.T. Jordan, K.C. Thompson and M.A. Collins, *J. Chem. Phys.* **102**, 5647 (1995)

3. M.J.T. Jordan, K.C. Thompson and M.A. Collins, *J. Chem. Phys.* **103**, 9669 (1995)

4. M.J.T. Jordan, K.C. Thompson and M.A. Collins, *J. Chem. Phys.* **104**, 4600 (1996)

5. K.C. Thompson and M.A. Collins, *J. Chem. Soc, Faraday Trans.* **93**, 871 (1997)

6. K.C. Thompson, M.J.T. Jordan and M.A. Collins, *J. Chem. Phys.* **108**, 8302 (1998)

7. R.P.A. Bettens and M.A. Collins, *J. Chem. Phys.* **111**, 816 (1999)

8. R.P.A. Bettens, *J. Am. Chem. Soc.* **125**, 584 (2003)

9. D.L. Crittenden, K.C. Thompson, M. Chebib and M.J.T. Jordan, *J. Chem. Phys.* **121**, 9844 (2004)

10. D.L. Crittenden and M.J.T. Jordan, *J. Chem. Phys.* **122**, 044102 (2005)

3. *Ab initio* methods available

a. Gaussian (98 and 03)

Analytic frequencies: hf, rhf, rohf, b3lyp, pw91, mp2, rmp2

Analytic gradients: mp4(sdq), ccd, rccd, mp2-force (do not use mp2-force! Provided for debugging purposes only)

Energies only: romp2, rmp4, mp4, ccsd, rccsd, ccsd(t),rccsd(t), mp2-en (do not use mp2-en! Provided for debugging purposes only)

b. Aces2

Analytic frequencies: rhf, uhf, rohf

Analytic gradients: rmp2, ump2, romp2, rmp4, ump4, rqcisd, uqcisd, rqcisd(t), uqcisd(t), rccsd, uccsd, roccsd, rccsd(t), uccsd(t), roccsd(t), eom-ccsd, qccsd

c. Tips and recommendations

For ground state systems, we recommend using at least a density functional method such as B3LYP or PW91, as implemented in the Gaussian03 suite of *ab initio* programs. Of course, it is necessary to benchmark these methods against a higher level of *ab initio* theory for the relative energies of geometries in the chemically relevant regions of configuration space. Additional density functional methods can be easily incorporated by editing the g03_dat.py file:

- add text 'functional_name' to the dictionary of methods and availability of derivatives in the available['analytic frequencies'] list
- add the appropriate Gaussian03 command string to the dictionary of methods and command strings, following the syntax of the currently available density functional methods (B3LYP and PW91). It should only be necessary to make a direct copy and replace the functional name to effect this change. Note: In Gaussian03 it is necessary to specify BOTH the exchange AND correlation functionals. Hence the correct nomenclature for PW91 is PW91PW91.

If a more systematic treatment of electron correlation is required, we recommend using second order Moller-Plesset Perturbation Theory (rmp2), as implemented in Gaussian03. If this treatment of electron correlation is insufficient, we recommend using a coupled-cluster method (either rccsd or rccsd(t)), as implemented in Aces2.

Coupled-cluster methods are recommended over configuration interaction methods as they possess better formal properties (most notably size-consistency) for an approximately equivalent computational cost. You will probably find, however, that it is impossible to construct a converged potential energy surface for any system with more than a couple of atoms within a reasonable time frame using these methods. If you do choose to use an Aces2 coupled-cluster method, we provide an important caveat here: Aces2 has a known bug in the code that calculates the second derivative matrix, which may introduce a small (but unquantifiable) error. Use with caution!

For excited state systems, we recommend extra careful inspection of the level of *ab initio* theory required. If a density functional method is appropriate, then this will be by far the most computationally efficient method. Otherwise, we recommend using a restricted open coupled-cluster method (roccsd or roccsd(t)) or equations-of-motion coupled-cluster method (eom-ccsd) as implemented in Aces2, subject to the above caveat on Aces2 calculations.

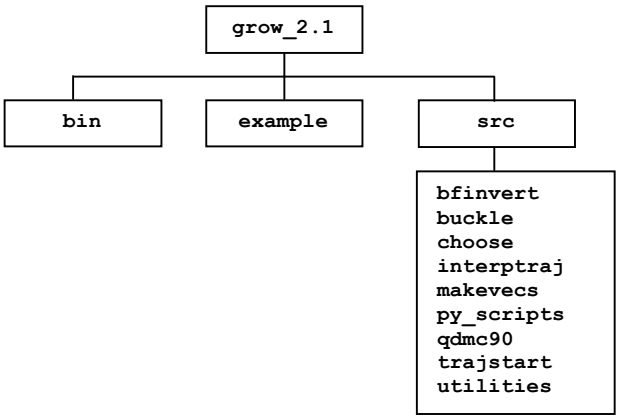
4. Installing Grow 2.2

Grow 2.2 is provided as a tarred archive. To extract the Grow 2.2 package, copy the archive to your home directory and execute the command:

tar xvf grow2.2.tar

(or tar –xvf grow2.2.tar for older versions of tar)

The archive you have extracted should possess the following directory structure:



You will need to ensure that grow2.2/bin/ is in your path. For C-shell users, this means adding a line similar to:

set path = ($path /home/username/grow2.2/bin)

to your .cshrc file.

For bash users, this means adding a line similar to:

PATH = $HOME/grow2.2/bin:$PATH; export PATH

to your .profile file.

In addition, you will need to have a python interpreter installed on your system, and accessible from the command line. C-shell users will need to have a line similar to:

set path = ($path /usr/opt/python2.2/bin/python)

in their .cshrc file, where /usr/opt/ is the directory where the python2.2 package has been installed. Similarly, bash users should have a similar line:

PATH = /usr/opt/python2.2/bin/python:$PATH; export PATH

in their .profile file.

Finally, the grow scripts assume that you have the ab initio packages configured such that they may be executed from the command line by the user. Again, this requires the executables to be present in the user's path (as above).

Having successfully extracted the source code and template input files, the next step is to recompile the executables for the platform you are working on. To do this, you will have to go through each the directories in grow2.2/sourcecode (except the py_scripts directory) and edit the FFLAG and F77 variables at the top of the Makefile to conform with the Fortran compiler you are using, then re-make the files. We have provided appropriate Makefiles for compiling these executables on DEC Alpha machines and Linux machines. To remove the executables and object files from the sourcecode subdirectories (especially useful where disk quota limits are an issue), type make clean in each directory after having compiled the executables using make.

5. Setting up the system

a. Deciding on a molecule/reaction and the appropriate dynamics method

The first step is to decide what system it is you want to study and what it is you want to know about the system you are studying. Unfortunately, neither of these steps are automated, as yet. If you are interested in studying a reactive system – either dissociation of a unimolecular system into two fragments or a bimolecular collision reaction or adhesion of a molecule to a surface – then you will need to use the *classical trajectory* code to perform your sampling and calculate your observables. If, on the other hand, you are interested in calculating the ground state nuclear vibrational wavefunction of a bound system – the method is equally applicable to tightly and loosely bound systems – then you will need to use the *quantum diffusion Monte Carlo* code.

b. Deciding on an appropriate level of *ab initio* theory

It is important to note that you need to choose an appropriate level of *ab initio* theory for the problem you are interested in. Clearly, the chemical accuracy of the results produced by Grow will depend on the chemical accuracy of the method you choose to construct the PES. To state the contrapositive; if you choose a chemically inaccurate *ab initio* method to construct the PES, then Grow will construct an accurate approximation to a chemically inaccurate PES, resulting in chemically inaccurate calculated observables. While it would be ideal to be able to use a highly correlated method with a large basis set for every problem of interest, in practice there must be a trade-off between chemical accuracy and computational expense.

Beside the usual consideration of the time required to calculate the energy of your system, it is also necessary to consider the time required to calculate the first and second derivatives of the energy. This is dependent on both the number of calculations required to calculate the derivatives and the time each calculation will take.

Method	Number of calculations
analytic second derivatives	1
analytic gradients	2(3N-6) + 1
energies only	(3N-6)(3N-5) + 1

Overall, it is always fastest to calculate second derivatives analytically, and calculating second derivatives by numerical differencing of first derivatives is always faster than calculating second derivatives by numerical differencing of energies. This generally becomes a **major consideration** in choosing which method to use to construct a PES, given the large number of *ab initio* calculations necessary.

c. Generating geometries for the initial data set

If you are constructing a PES for a reactive system, you will need to perform some preliminary quantum chemistry to evaluate a known reaction path. If you know that there are multiple reaction paths, then it is wise to include as much information about these paths as you can access. It is important to note that you must include the minimum energy geometry for the reactants and sufficient long-range data for the separated products. The minimum number of geometries required is three: one which represents the minimum energy geometry, one which represents the transition state and one which represents the separated products. If the products are neutral, then an interfragment separation of 10-13 Bohr is probably sufficient for negligible remaining interaction. If the fragments possess large dipoles, then 15-20 Bohr may be better and if one species is charged then an interfragment separation of > 20 Bohr will probably be necessary.

If you are constructing a PES for a bound system, you will need to first determine all the low-lying minima and, if possible, transition states to interconversion between those minima. While only one geometry, corresponding to the global minimum, is required to generate the initial data set, convergence of the potential energy surface can be rapidly accelerated by judicious choice of initial geometries.

6. Setting up the input files

a. Overview

Before describing the input files in detail, a summary table of the input files and the dependencies of each program on these files is provided:

File	Programs
IN_ATOMPERMS	trajstart, interptraj, qdmc
IN_DX_SIZE	diffsteps
IN_EMS	trajstart
IN_GROW	startPOT, grow
IN_INTERP	trajstart, interptraj, qdmc
IN_ISEED	trajstart, interptraj, qdmc
IN_QDMC	qdmc
IN_SYSTEM	trajstart, interptraj, qdmc
IN_TRAJ	interptraj

In the following example input files, water will be used as the test system. The example files will be given and discussed in the same order as you will need to set them up. Please note that all lines in the input files **must** be included – the format of these files is assumed by the programs that use them.

b. IN_SYSTEM

```
IN_SYSTEM file for water
enter the number of atoms
3
enter the chemical symbols for the atoms, separated by commas
O,H,H
enter the atomic masses in amu
15.9994 1.00794 1.00794
The number of atoms in fragments a and b respectively are
3,0
The atom identification numbers in fragment a are
1,2,3
The atom identification numbers in fragment b are

enter the bond lengths, in order, which give the cutoff for a
real bond to exist between atoms i and j (units are bohr)
4.0
4.0
4.0
```

Notes:

Line 1: A heading to describe the relevant system

Lines 2 and 3: The number of atoms may not exceed the maximum value specified by the parameter natomm in various fortran programs (generally found in the .inc files). The programs have been set up with natomm = 35. If you wish to study a larger system than this, you will need to edit the source files.

Lines 4 and 5: Enter the element symbols, separated by commas only (no spaces). The order of the atoms here **is important**. All previous *ab initio* jobs run must have the atoms in this order. All subsequent *ab initio* jobs will have the atoms in this order.

Lines 6 and 7: The atomic masses must be entered in the same order as the chemical symbols

Lines 8 – 13: These parameters define the reactant species in the bimolecular collision or unimolecular reaction, and hence are used by programs associated with performing

classical trajectory simulations. For bound-state calculations, these lines must still be present, but they are skipped during the process of reading in parameters.

Lines 14,15 to end: Again, these parameters are only necessary for defining fragments during a reaction. If the distance between two atoms is less than the bond length given here, the atoms are considered bound. Conversely, the atoms are considered unbound when their separation exceeds the value given here. Note: the identity of the bonds are defined by:

1-2				
1-3	2-3			
1-4	2-4	3-4		
1-5	2-5	3-5		
1-N	2-N	3-N		N-N

The list of bond lengths is generated by reading down each column, starting with column 1 and appending subsequent columns.

c. IN_INTERP

```
IN_INTERP file for water
Enter 1 or 2 for one or two part weight function
1
Enter the weight powers q and p
2, 9
Enter the weight cutoffs for the inner and outer neighbour lists
1.d-5   1.d-4
Enter the # of time steps for updates of the outer and inner neighbour lists
10   5
Enter the maximum acceptable energy from POT, and expected minimum energy
-75.5000000       -75.70622799
Enter how many neighbours are used to define the confidence lengths
10
Enter the energy error tolerance which defines the confidence lengths
5.d-4
Number of clusters for neighbourlist reduction
1
```

Line 1: A heading to describe the relevant system

Lines 2 and 3: Generally, the two-part weight function should be used. However, when the number of data points is less than approx. 100 (calculated as the total number of data points, including points generated by nuclear permutation symmetry = $N_{perm} \times N_{data}$), there will be insufficient data to allow this. The one-part weight function should only be used to grow the surface until there are sufficient data points to use the two-part weight function.

Lines 4 and 5: The weight powers (q and p) should be given by q=2, p >> (3N-3)/2, where N is the number of atoms. We recommend p = 9 for a 3 atom system, p = 12 for 4 and 5 atoms systems and p =15 for 6 atoms.

Lines 6 and 7: The weight cutoffs define the minimum contribution each data point must make to determining the overall energy (in the Taylor series summation) for the data point to be included on a list of data points "near" an arbitrary configuration. If the weight cutoff is small, more data points will be included on the neighbourlist, and your simulations will run more slowly. However, the accuracy of the surface will be

higher. Conversely, if the weight cutoff is larger, less data points will be included on the neighbourlist and your simulations will run faster, although the accuracy of the surface will be lower. The weight cutoffs can probably be left at the values shown, 1.d-5 and 1.d-4. This implies an error of around 0.01 mE_h in the PES due to neglect of minor contributions to the Taylor series summation.

Lines 8 and 9: The number of time steps for updates of the inner and outer neighbourlists are predominantly dependent on the time step, for both QDMC and trajectory simulations. We recommend setting these values to 10 and 5. If the neighbourlists are not updated frequently enough, trajectories will fail to conserve energy. The QDMC algorithm is less sensitive to small discontinuities in the PES and thus also less sensitive to the values of these parameters.

Lines 10 and 11: For trajectory simulations, the maximum acceptable energy should be set to zero while the PES is growing (all molecular energies are negative, so all are acceptable). The expected minimum energy should be set equal to the lowest energy data point in the POT file. For QDMC simulations, the aim is to accurately describe the *low energy* regions of the PES. In order to increase computational efficiency, we recommend setting the maximum acceptable energy equal to approximately double the harmonic zero-point energy of the system of interest.

Note: during the early iterations of the growing process, the scheme may choose some new data points at ridiculous geometries with very high energies. When the PES is finished growing, there may be a small increase in interpolation accuracy is such data points are disregarded. Setting the maximum energy below the energies of such data points will cause the interpolation section of the QDMC or trajectory programs to completely disregard such data points.

Lines 12-15: These parameters are used with the 2-part weight function only. The number of neighbours used to define the confidence radius is the variable M in Equation 3.13 of Bettens & Collins, *J. Chem. Phys.* **111**, 816 (1999). The actual value is not critical. For a four-atom system, a value of 24 – 48 is suitable, for 5 or more atoms, a value in the range 50 – 150 may be more appropriate. The energy error tolerance is the parameter Etol in Bettens & Collins, *J. Chem. Phys.* **111**, 816 (1999). A value of 2.d-4 (0.2 mE_h) is reasonable for a four-atom system, while a larger system might be given a slightly larger value. This is approximately how small you might expect the average interpolation error to be.

Lines 16-17: The number of clusters for neighbourlist reduction parameter is only used by the QDMC code. You should aim to have between 20-50 data points per cluster by considering the total number of data points, including points generated by nuclear permutation symmetry = $N_{perm} \times N_{data}$.

d. IN_ATOMPERMS

```
IN_ATOMPERMS file for water
 enter the order of the group
        2
 enter each atomic perm old..new, separated by a comment line
        1        1
        2        2
        3        3
 *******
        1        1
        2        3
        3        2
 *******
```

This file defines the symmetry of the molecule in terms of the allowed permutations of the indistinguishable atoms in the molecule. You can generate the list of all possible permutations by executing the cnpiperms utility from the working directory, provided you have already set up the IN_SYSTEM file. If this results in a group which is unmanageably large, and you wish to regard some permutations as infeasible, a smaller permutation symmetry group can be constructed by either:

- labelling the atoms in the IN_SYSTEM file e.g. O,H1,H2 will result in distinguishable hydrogen atoms
- constructing the IN_ATOMPERMS file by hand. If you wish to construct this file by hand, the contents of the file should adhere to the following syntax:

Line 1: A heading to describe the relevant system

Lines 2 and 3: A heading – "enter the order of the group" followed by the number of permutations you wish to allow, entered on a separate line

Line 4: A heading – "enter each atomic permutation, old… new, separated by a comment line"

Line 5 to end: the permutations, in the format shown above, with a blank line between each set. Note: the first set of permutations **must** correspond to the identity permutation.

e. IN_DX_SIZE

---

step size for numerical derivatives by finite difference

0.001

Line 1: title line

Line 2: the step size (in Bohr) for displacements in internal coordinates

f. IN_ISEED

2109689680

Line 1: blank line

Line 2: initial seed for random number generator

g. IN_GROW

```
################################################
## Grow2.2 input file to control python scripts - simple      ##
################################################
molecule name                          water
maximum number of iterations           4
convergence check every                4
sampling method is                     qdmc
convergence method is                  qdmc
rms points to add per iteration        1
hwt points to add per iteration        9
ab initio package                      g98
charge                                 0
spin                                   1
scratch directory                      /scratch/
read-write file                        default
memory limit                           100mb
number of processors                   default
job type (simple or scaled)            simple
simple method                          mp2
simple basis set
O H 0
3-21G
****

+++
universal basis set                    3-21G
```

```
################################################
## Grow2.2 input file to control python scripts - scaled      ##
################################################
molecule name                          water
maximum number of iterations           4
convergence check every                4
sampling method is                     qdmc
convergence method is                  qdmc
rms points to add per iteration        1
hwt points to add per iteration        9
ab initio package                      g98
charge                                 0
spin                                   1
scratch directory                      /scratch/
read-write file                        default
memory limit                           100mb
number of processors                   default
job type (simple or scaled)            scaled
high correlation method                mp4(sdq)
low correlation method                 mp2
small basis set
O H 0
3-21G
****

+++
large basis set
O H 0
6-31G
****

+++
universal basis - small                3-21G
universal basis - large                6-31G
```

Line 1: The name of the system is used as the suffix for a large number of temporary files created by the script. This name is used for reference purposes by the scripts, and must be referred to when restarting a startPOT or grow run.

Line 2: The maximum number of iterations is the number of cycles of sampling, choosing and *ab initio* calculations that will be carried out by the script. The total number of ab initio calculations that will be carried out is equal to the number of iterations times the total number of points (h-weight and rms) chosen per iteration. The QDMC sampling calculations use the input file IN_QDMC.small and the trajectory sampling calculations use the input file IN_TRAJ.small.

Line 3: Convergence check calculations use the input files IN_QDMC.large and IN_TRAJ.large for QDMC and trajectory simulations, respectively.

Lines 4 and 5: The keywords for sampling and convergence check simulations are qdmc (for quantum diffusion Monte Carlo) and classical (for classical trajectories).

Lines 6 and 7: The number of points to add per iteration using one of two criterion for choosing 'optimal' data points – either by root-mean-squared deviation from the existing data (rms) or high probability of being in a chemically relevant region of configuration space (hwt).

Line 8: The *ab initio* packages available are g98, g03 and aces2.

Lines 9 – 14: Are machine-specific and system-specific variables used to set up the *ab initio* calculations. Charge and spin have their usual meanings from *ab initio* theory. The scratch directory **must** be the full address (no use of wildcards such as ~ to specify home directory) and **must** end in /. The read-write file string must be in the format required by the *ab initio* program chosen. We recommend leaving this set as 'default', which will use the default read-write files set up by the *ab initio* program. The memory limit card specifies the amount of virtual memory requested in the *ab initio* input file. The number of processors line is only valid for Gaussian03. For all other *ab initio* programs, and for single-processor jobs in Gaussian03, we recommend that you leave this setting as 'default'.

Lines 15 – end: Determine the level of *ab initio* theory used to construct the PES. Jobs may be either 'simple' – calculated at a single level of theory, or 'scaled' – the PES is calculated at a relatively low level of theory then corrected for basis set and electron correlation deficiencies in a G2-type fashion, according to the formula:

$$\Delta E_{high,large} = \Delta E_{high,small} + \Delta E_{low,large} - 2\Delta E_{low,small}$$

The simple and scaled calculations require different input, as illustrated in the examples above.

The *ab initio* method for the simple calculations is specified on the 'simple method' line. The available methods and keywords are given in Chapter 3. The basis sets must be specified twice – once using the term 'simple basis set' and once using the term 'universal basis set'. The 'simple basis set' input is used in constructing Gaussian input files, and the basis set must be specified according to the Gaussian Gen basis specifications. This basis set input is read from the line following 'simple basis set' to the line containing the delimiter '+++'. The basis set read from the end of the line containing 'universal basis set' is used in constructing Aces2 input files.

The *ab initio* methods for the scaled calculations are read from the 'high correlation method' and 'low correlation method' lines. Again, the basis sets must be specified twice – once in Gaussian input format and once in Aces2 input format. The small basis set Gaussian input is read from the line following 'small basis set' to the delimiter '+++'. Similarly, the large basis set is read from the line following 'large basis set' to the delimiter '+++'. The Aces2 basis sets are read from the ends of the 'universal basis – small' and 'universal basis – large' lines.

Note: the basis set input in **both** Gaussian and Aces2 format **must** be present.

h. IN_EMS

```
The Markov walk control parameters
Enter the number of Markov steps, and the stepsize for fragment a
1000,0.02
Enter the number of Markov steps, and the stepsize for fragment b
1000,0.02
bonds to constrain (bond number, length in bohr)
0  4.0
0  4.0
0  4.0
Cartesian geometry for separated initial fragments (Bohr)
 0.0000000000000000E+00   0.0000000000000000E+00   0.0000000000000000E+00
 0.0000000000000000E+00   0.2098017302776257E-15   0.1823368639565938E+01
 0.0000000000000000E+00   0.1784174494261178E+01  -0.3760246930460953E+00
```

This input file contains parameters to control the Monte Carlo (Markov walk) generation of initial atomic positions and velocities for the classical trajectories, according to the "efficient microcanonical sampling" scheme of Schranz et al. *J. Chem. Phys.* **94**, 1487 (1991). The program, trajstart, produces a microcanonical vibrational energy distribution for each reactant fragment, with zero rotational angular momentum.

Line 1: title line

Lines 2 – 5: The number of Markov steps and stepsize for each fragment can probably be left at the values shown. For further details on the appropriate values for these parameters, consult the reference given below.

Line 6 and the next $^{N}C_2$ lines (where N is the number of atoms): A constraint flag of 0 means that the bond is unconstrained during the Markov walk whereas a constraint flag of 1 means that the bond is constrained to the value given (useful for constraining a molecule to remain in the vicinity of a saddle point). Although the actual bondlength for unconstrained bonds is irrelevant, a value must be present.

Final N+1 lines: A title line followed by the Cartesian geometry of the separated initial fragments (in Bohr). Note: although the atom labels must **not** be included here, the atom order for these geometries must be the same as specified in the IN_SYSTEM file.

i. IN_TRAJ

```
System is water
Enter the time step, # of steps, # of steps between prints, # of trajectories
0.001d0,   20000,    200,   100
Enter the energy of frag a, energy of frag b, relative trans energy (in au)
0.010,0.0,0.0
Enter the allowed relative error in the energy conservation of traj.
4.d-7
Enter the maximum impact parameter, initial fragment separation (in au)
0.00,15.0
Enter the fraction of trajectory points output
1.d0
Do we stop trajectories with low energy? 0=no, 1=yes
1
```

Line 1: A heading to describe the relevant system

Lines 2 and 3: A time step for the velocity Verlet trajectory integrator (in atomic units: 1 a.u. = fs). The value chosen for this parameter is motivated by the trade-off between accuracy (as measured by conservation of energy during the trajectory simulation) and computational expense (the computer time required to simulate a given length of real time). If the time step is too large, the trajectories will not conserve energy with the tolerance given on line 7. It may require some experimentation to determine the optimal value of this parameter. Note: energy conservation is also related to frequency of neighbourlist updates, as specified in the file IN_INTERP.

For reactive systems, the number of allowed time steps can be as large as you like, since the trajectories are automatically stopped when the product fragments are separated by the original reactant separation given on line 9. For classical simulations of bound-state systems (note: this is possible, but not recommended), you should consider the time scale of the event you are interested in observing, and set the number of time steps accordingly.

The number of time steps between prints parameter determines the frequency at which output information is written to the files OUT_INTERP and TOUT (The TOUT file contains the set of configurations from which the new data points are

chosen by the program choose). Warning: if this parameter is excessively small, then the number of configurations written to TOUT may exceed the maximum specified in the source file choose.inc in the choose/ directory. Further, the OUT_INTERP file may become unmanageably large.

The number of trajectories to be performed should be around 10 in the IN_TRAJ.small file and around 1000 in the IN_TRAJ.large file. The maximum possible number of trajectories is defined in the traj.inc file in both the trajstart/ and interptraj/ directories.

Lines 4 and 5: The internal energies of the fragments are total energies (kinetic + potential), given relative to the equilibrium energy of the fragment. Note: this energies must be specified in atomic units (E_h).

Lines 6 and 7: The absolute relative error in energy conservation which is allowed during a trajectory before the trajectory is terminated and counted as "bad". Typically, this parameter should be on the order of 1.d-7. This would, for example, correspond to an error of 1.d-5 E_h in a system with a total energy of 100 E_h.

Lines 8 and 9: The maximum impact parameter can be set to zero in IN_TRAJ.small. In IN_TRAJ.large, a reasonable value would be the sum of the approximate van der Waals radii of the two reactant fragments. If you wish to calculate an accurate reaction cross section, the maximum impact parameter should be set to the minimum impact parameter at which no reaction is observed, to ensure that all possible reactive trajectories have been sampled. How to determine this value for the maximum impact parameter is presented in Appendix 3.

The initial fragment separation determines the "extent" of the PES in the reactant and product "valleys". All trajectories will start with the two fragments separated by this distance (unless constrained by bond length in IN_EMS), and any trajectory is terminated if the fragments are moving apart and separated by more than this distance. As the TOUT file cannot contain geometries with fragments separated by more than this distance, the POT file will also not contain data points for such geometries. The value of the initial fragment separation has to be chosen on physical grounds. We suggest that 10-15 Bohr is sensible for neutral systems with no strongly dipolar fragments, 15-20 Bohr if dipoles are large and 20-25 Bohr for charged systems.

Lines 10 and 11: This parameter should be set to 1.0 in IN_TRAJ.small and 0.0 in IN_TRAJ.large. Using these parameters, the molecular configuration is printed to the TOUT file with the frequency determined by the 'number of steps between prints' parameter on line 3 for the sampling runs. No molecular configurations are printed to file during the convergence runs.

Lines 12 and 13: Trajectories with low energy should be stopped during the sampling phase: this parameter should be set to 1 in IN_TRAJ.small. This enables the Grow algorithm to add data where the existing data predicts "unphysical" holes in the PES. Trajectories with low energy should be allowed to continue during convergence runs: this parameter should be set to 0 in IN_TRAJ.large.

j. IN_QDMC

```
The IN_QDMC file for water
number of seed geometries
1
numbers of walkers: min, max, walkers per geom
500,1500,1000
blocks: total number, number until equilibrium, steps per block
10,5,1000
descendant weighting: # of generations, delay between generations, number of steps
5, 250, 1000
time step
1.0
feedback parameter
1.0
maximum step size for initial displacements from equilibrium
0.5
probability of writing a walker to TOUT
1.0
histograms: upper and lower bounds, bin size
15.0, 0.0, 0.05
restart (y for yes, n for no)
n
symmetrize bond histograms? (y for yes, n for no)
n
geometry in bohr (additional geoms must be separated by a blank line)
 0.0000000000000000E+00   0.0000000000000000E+00   0.0000000000000000E+00
 0.0000000000000000E+00   0.3454907916233456E-16   1.8226681435876140E+00
 0.0000000000000000E+00   1.7834890555985659E+00  -0.3758732226199827E+00
```

Line 1: title line

Lines 2 and 3: The number of geometries at which to base populations of walkers/replicas. For molecules with a single, well-defined minimum energy conformation (like water and methane), this parameter should be one. For molecules with multiple approximately isoenergetic local minima (generally, but not necessarily, accessible through torsional motion), it is more sensible to 'seed' the simulation with equal populations of walkers created in each minimum.

Lines 4 and 5: The minimum number of walkers, an upper bound on the total number of walkers, and the number of walkers to create near each minimum energy geometry. Note: the total number of walkers (number of seed geometries × number of walkers per seed geometry) should fall approximately in the middle of the minimum and maximum numbers of walkers.

Lines 6 and 7: Collectively, these parameters determine the length of the QDMC run. The total number of steps per simulation is the product of the 'number of blocks' with the 'number of diffusion steps per block'. This is divided into an equilibration stage and a production stage. The simulation is divided into 'blocks' for two reasons:

1. to facilitate calculation of error bars, as discussed in Appendix 4
2. to moderate the frequency with which sampled configurations are written to the TOUT file. Following equilibration, the entire ensemble of replicas is printed to the TOUT file at the end of each block. Therefore, the number of geometries can be calculated as ['total number of blocks' – 'number to equilibrium'] × total number of walkers.

The total number of steps should vary according to system size and run type (sampling or convergence). We recommend 10,000 time steps for sampling runs and at least 50,000 time steps per convergence run. Typically, 10 or 20 separate convergence runs are required to obtain production quality results.

Lines 8 and 9: The descendant weighting algorithm (as discussed in Appendix 4) is used to calculate vibrationally-averaged internal coordinates. The results obtained should be independent of the parameters chosen here, provided that the generations are initiated at a long enough time interval and that the descendant weighting runs are long enough. The error bars on the observables calculated by descendant weighting can be reduced by increasing the number of generations. Note: the total number of descendant weighting steps plus time step offset for initiation may not exceed the total number of steps you have specified for the production stage of the QDMC run.

Lines 10 and 11: The time step (in atomic units of imaginary time!). The time step can probably be left at the value given, 1.0 a.u.

Lines 12 and 13: The feedback parameter controls the sensitivity of relationship between the number of replicas created in each branching step and the trial energy. Again, this parameter can probably be left at the value given, 1.0.

Lines 14 and 15: The initial geometries of the walkers are generated by displacement along the Cartesian axes from the equilibrium geometry. This parameter controls the magnitude of this displacement.

Lines 16 and 17: This parameter should be set to 1.0 in IN_QDMC.small and 0.0 in IN_QDMC.large.

Lines 18 and 19: The wavefunction histograms are constructed by binning the replicas into internal coordinates at every step of the QDMC simulation. The upper bound should be set to approximately double the longest bond length in the equilibrium geometry. The lower bound should be left at 0.0. The bin size determines the resolution of the histograms and the choice of bin size is motivated by two factors: the number of data available for binning and the required resolution. A higher resolution (smaller bin size) requires more data i.e. a longer simulation run. For most purposes, the bin size can probably be left at 0.05 Bohr.

Lines 20 and 21: The QDMC simulation can be restarted from the population of a previous simulation, using the IN_RESTART file. If the flag in IN_QDMC is set to 'y', then the initial population of walkers will be taken from IN_RESTART. If the flag is 'n', then the initial distribution is generated by random displacements of the atoms from equilibrium. The functionality is particularly useful for performing multiple convergence runs. This flag must be set to 'n' for in both IN_QDMC.small and IN_QDMC.large, for the first few iterations of the Grow script (until the at least one sampling → convergence cycle has taken place). We recommend only setting this flag to 'y' for production runs on the converged surface.

Lines 22 and 23: The bond histograms can be automatically symmetrised to incorporate nuclear permutation symmetry by setting this flag to 'y'. This symmetrisation can also be carried out by post-processing of the histogram data.

Lines 24 and 25: The Cartesian coordinates of the equilibrium geometry/geometries (in Bohr). Note: the first geometry follows the title line directly, but additional geometries **must** be separated by a blank line. Note: although the atom labels must **not** be included here, the atom order for these geometries must be the same as specified in the IN_SYSTEM file.

7. Assembling a crude potential energy surface

Once you have generated the geometries corresponding to all known reaction paths for a reactive surface or all known local minima for a bound surface (described in section 5c), and set up the IN_SYSTEM file, you will need to run the 'startPOT' script to generate the *ab initio* data points. The 'startPOT' script works by searching for the 'standard orientation' keyword to extract geometries contained in a Gaussian log file (g03file.log). You should edit your Gaussian log file so that it contains only the geometry/geometries that you wish to add as data points (especially important if you have generated the geometries using a geometry optimisation or relaxed scan algorithm – these output files will contain many similar geometries that do not all need to be included in the data set).

The syntax for invoking the 'startPOT' script is:

```
python2.2 ../sourcecode/py_scripts/startPOT.py IN_GROW_molecule g03file.log
```

If, for some reason, one of your *ab initio* jobs dies in the process of assembling your crude PES, 'startPOT' can be restarted using the command:

```
python2.2 ../sourcecode/py_scripts/startPOT.py restart molecule
```

Warning: 'startPOT' assumes that you want to create a PES data file from scratch, and will overwrite any data previously generated and saved as the 'POT' file. If you want to run 'startPOT' twice (using two different Gaussian log files, for example) you will need to either run the jobs in separate directories or save the first POT file generated under a different name.

8. Growing an accurate potential energy surface

The required input files for growing a potential energy surface using either trajectory or QDMC sampling are listed in the table below:

trajectory	QDMC
IN_SYSTEM	IN_SYSTEM
IN_ATOMPERMS	IN_ATOMPERMS
IN_GROW	IN_GROW
IN_INTERP	IN_INTERP
IN_ISEED	IN_ISEED
IN_DX_SIZE	IN_DX_SIZE
IN_EMS	IN_QDMC.small
IN_TRAJ.small	IN_QDMC.large
IN_TRAJ.large	

Once the requisite files have been set up and you have generated your crude PES (as described in section 7 above), you are ready to grow! The syntax for executing the 'grow' script is:

```
python2.2 ../sourcecode/py_scripts/grow.py IN_GROW_molecule
```

If, for some reason, one of your sampling or *ab initio* jobs dies in the process of growing your surface, the script can be restarted using the command:

```
python2.2 ../sourcecode/py_scripts/startPOT.py restart molecule
```

9. Running trajectory and quantum diffusion Monte Carlo simulations

The trajectory and QDMC codes can both be executed independently of the grow scripts. The syntax for executing the trajectory code is:

trajstart > inic

interptraj < inic

The syntax for executing the QDMC code is:

qdmc90 N > OUT_QDMC.N

where N indexes the simulation number, to prevent overwriting output files.

10. Understanding the output – trajectories

The trajectory output data is written to the files:

File name	Produced by	Contains
OUT_INITIAL	trajstart	echo of input parameters in the IN_* files
OUT_IMPACTPAR	trajstart	the impact parameter, energy and bondlengths at the initial configuration of each trajectory
inic	trajstart	initial conditions for the trajectories
OUT_SUMMARY	interptraj	summary of the output of a trajectory simulation, in terms of fragment products and number of 'low energy', 'bad' and unfinished trajectories
OUT_ANG	interptraj	the instantaneous rotational angular momentum of the fragments at the completion of each trajectory
OUT_BTS	interptraj	the outcome of each trajectory in terms of the product composition
OUT_FINALBONDS	interptraj	the value of the bondlengths (Bohr) when the trajectory is terminated
OUT_FINALCV	interptraj	the Cartesian coordinates and velocities of the atoms when the trajectory is terminated
OUT_INTERP	interptraj	echo of input parameters in IN_INTERP. Also contains periodic single-line progress reports of the trajectories, using the abbreviations defined below
OUT_VEL	interptraj	the kinetic energy of the relative motion of the two product fragments at the completion of the trajectory
TOUT	interptraj	collection of configurations accessed during the trajectory simulations

Abbreviation	Definition
timps	time in picoseconds
nforc	number of data points used in the interpolation of the potential energy at the current molecular configuration
nfin	the status of the trajectory at termination: 0 = unfinished 1 = completed satisfactorily 2 = total energy not conserved - “bad” 3 = low energy trajectory (passed below expected minimum value)
errck	the relative error in the total energy
ek	current total kinetic energy
en	current potential energy

11. Understanding the output – QDMC

The QDMC data is output into 5 files: OUT_QDMC, OUT_BOND_HIST, OUT_DIHED_HIST, TOUT and IN_RESTART.

OUT_QDMC contains an echo of the input parameters, followed by a header, then periodic progress reports on the ensemble energy and number of replicas. At the end of the simulation, the vibrationally-averaged zero-point energy (in kJ/mol and Eh), vibrationally averaged bond lengths and an estimate of the error in the zero-point energy (calculated using the 'blocking algorithm' – see Appendix 4) are printed.

The OUT_BOND_HIST and OUT_DIHED_HIST files contain the wavefunction histograms for the bond lengths and dihedral angles, respectively. These files may require some post-processing preceding visualization and some fortran utility files have been provided for this purpose (see section 13).

The TOUT file contains the collection of replicas accessed during the QDMC simulation, as printed out at the end of each block.

The IN_RESTART file is a print out of the Cartesian coordinates of the replicas at the last step in the QDMC simulation. The title line contains information about the number of replicas and the ensemble energy at the final step of the simulation.

12. Understanding the output – grow scripts and other programs

Collectively, the grow scripts and fortran programs produce the PES data set, which is stored in the file named 'POT'. This is **the most important** file produced (and also the most computationally expensive to produce), so be very careful not to delete or overwrite it. The POT file contains a title line followed by the processed *ab initio* data for each data point. The processed *ab initio* data is structured in the following way:

- Heading – "--- data point # XX'
- The N(N-1)/2 bondlengths, in the standard order
- The N(N-1)/2 coefficients, U_{1i}, of the reciprocal bondlengths in the first local internal coordinate, followed by the N(N-1)/2 coefficients, U_{2i}, down to $U_{(3N-6)i}$.
- The energy of the data point
- The (3N-6) first derivatives with respect to the local internal coordinates
- The (3N-6) diagonal second derivatives with respect to the local internal coordinates.

There are a number of other files produced by the fortran programs. These are generally uninteresting, except for debugging, but are listed here for completeness.

COUT – produced by the program 'choose'. It contains the geometries which have been selected from the TOUT file to become the next data point/s.

OUT_SV1 and OUT_SV2 – produced by the program 'buckle'. Contain the singular values for the intrafragment (SV1) and interfragment (SV2) B matrix (see reference 6 in section 2)

OUT_MOL – produced by the program 'buckle'. Contains the original geometry from COUT and the distorted geometry produced by 'buckle'.

13. The programs

Program	Function
choose	Reads in the file 'TOUT', containing configurations accessed during the trajectory or QDMC simulation, and chooses a specified number for inclusion as data points in the 'POT' file.
buckle	Distorts the geometry selected by 'choose' so that the molecule is not planar or linear, to avoid singularities in the transformation to local internal coordinates.
bfinvert	Constructs the local internal coordinates and transforms the derivative data generated by the *ab initio* calculation.
trajstart	Generates initial conditions for the trajectories to be propagated using 'interptraj'.
interptraj	Propagates the trajectories. Creates various OUT_* files that contain information about the calculated observables.
py_scripts	Control the grow process. The main files of importance to the user are grow.py, startPOT.py and the data files (e.g. g98_dat.py). The data files can be edited to enable additional ab initio methods (see section 3c).
qdmc90	Calculates the exact ground-state nuclear vibrational wavefunction, vibrationally-averaged zero-point energy, vibrationally-averaged bond lengths and wavefunction histograms.
makevecs	Makes the displacement vectors needed for the efficient calculation of numerical derivatives.
utilities	An assortment of useful little programs, whose functions are described below. All the utility programs take data from standard input and write to standard output e.g. ptoe < POT > energies.out. These programs also read the IN_SYSTEM file, so should only be used in the directory in which you are growing the PES.

Program	Function	Syntax
Ang2Bohr	Converts Cartesian geometries from Angstrom to Bohr	Ang2Bohr < cartfile > outfile
Bohr2Ang	Converts Cartesian geometries from Bohr to Angstrom	Bohr2Ang < cartfile > outfile
basiscorr	Combines the results of high and low level calculations for 'scaled' potential energy surfaces	Used exclusively by the scripts
cart2dist	Takes Cartesian coordinates and outputs the pair-wise distances	cart2dist < cartfile > bondfile
cnpiperms	Generates the IN_ATOMPERMS file by constructing the complete nuclear permutation symmetry (CNP) group	cnpiperms
diffsteps	Uses the CARTVECS file to generate the grad.geoms file which contains the geometries required for calculating numerical derivatives	Used exclusively by the scripts
dist2cart	Takes the pair-wise distances and generates the Cartesian coordinates	dist2cart < bondfile > cartfile
en2freq	Calculates the numerical second derivatives from the *ab initio* energies of the geometries in the grad.geoms file	Used exclusively by the scripts
grad2freq	Calculates the numerical second derivatives from the *ab initio* gradients of the geometries in the grad.geoms file	Used exclusively by the scripts
ptob	Extracts the bond lengths of the data points in the POT file	ptob < POT > outfile
ptoe	Extracts the energies of the data points in the POT file	ptoe < POT > outfile
ptof	Extracts the forces of the data points in the POT file	ptof < POT> outfile
rotranz	Rotates a geometry (in Cartesian	Used exclusively by the

	coordinates) into “Z-matrix orientation”	scripts
xmolmovie	Reads in the TOUT file and creates an output file that can be read by the ‘xmol’ program to create a movie of a trajectory	xmolmovie > outfile

14. Using Grow on a supercomputer

A separate installation of Grow is provided for use on machines with automated queuing systems (and wall time limits). Note: the current 'pbs' implementation of Grow has limited functionality, compared to the standard version. Notably, this implementation can only construct 'simple' potential energy surfaces (not scaled) and only for *ab initio* methods with analytic second derivatives. Further, the current implementation is only set up to use quantum diffusion Monte Carlo sampling (although the modifications required for trajectory sampling should be trivial). The major difference between 'pbs Grow' and 'standard Grow' is that the standard Grow script runs continuously in the background while the pbs Grow script is suspended while the sampling, choosing and *ab initio* calculations are carried out by separate scripts submitted to the pbs queuing system. In total, four such scripts are required:

- choosehwt.scr
- chooserms.scr
- qdmc.scr
- molecule.scr

Examples of these script files (again, using water as a test system) will be given on the following pages.

a. choosehwt.scr

```
#!/bin/bash
#PBS -P g23
#PBS -l walltime=01:00:00
#PBS -l vmem=1024MB
#PBS -l ncpus=1
#PBS -wd

choose < chooseHWt.inp

python2.2 ../sourcecode/py_scripts/grow.py restart water
```

Lines 1 – 6: machine and system-dependent information required to run the pbs job

Line 7: blank line (included for clarity, can be omitted)

Line 8: 'choose' program is called. Note: this syntax assumes that 'choose' can be executed from the command line.

Line 9: blank line (included for clarity, can be omitted)

Line 10: the grow script **must** be restarted after choose has completed. This call assumes that you are working in a grow2.2 subdirectory e.g. grow2.2/water/

b. chooserms.scr

```
#!/bin/bash
#PBS -P g23
#PBS -l walltime=01:00:00
#PBS -l vmem=1024MB
#PBS -l ncpus=1
#PBS -wd

choose < chooseRMS.inp

python2.2 ../sourcecode/py_scripts/grow.py restart water
```

This file is identical to the choosehwt.scr file, except that the file chooseRMS.inp is used as input to the choose program rather than the file chooseHWt.inp.

c. qdmc.scr

```
#!/bin/bash
#PBS -P g23
#PBS -l walltime=20:00:00
#PBS -l vmem=1024MB
#PBS -l ncpus=1
#PBS -wd
#PBS -l jobfs=1000MB

qdmc90 1 > OUT_QDMC.1
cp TOUT.1 TOUT

python2.2 ../sourcecode/py_scripts/grow.py restart water
```

Lines 1 – 7: machine and system-dependent information required to run the pbs job

Line 9: blank line (included for clarity, can be omitted)

Lines 10 and 11: 'qdmc90' program is called. Note: this syntax assumes that 'qdmc90' can be executed from the command line. The number following the qdmc90 call is not optional – it is required as input to the qdmc90 program, and is used to index all the output files (including TOUT). As the grow scripts assume TOUT is not indexed, it is necessary to copy TOUT.1 back to TOUT.

Line 12: blank line (included for clarity, can be omitted)

Line 13: the grow script **must** be restarted after qdmc90 has completed. This call assumes that you are working in a grow2.2 subdirectory e.g. grow2.2/water/

d. molecule.scr (In this case, water.scr – must have same stem as molecule name specified in IN_GROW_molecule)

```
#!/bin/bash
#PBS -P g23
#PBS -l walltime=1:00:00
#PBS -l jobfs=5GB
#PBS -l vmem=700MB
#PBS -l software=g03
#PBS -l ncpus=1
#PBS -wd
#PBS -q express

USE_DOT=1; export USE_DOT
USE_G03=1; export USE_G03
. /opt/etc/system_profile
GAUSS_SCRDIR=$PBS_JOBFS; export GAUSS_SCRDIR

g03 < water.com > water.log

python2.2 ../sourcecode/py_scripts/grow.py restart water
```

Lines 1 – 14: machine and system-dependent information required to run the pbs job
Line 15: blank line (included for clarity, can be omitted)
Line 16: The ab initio package is called. In this case, a Gaussian03 job is run. Note: the com file stem be the same as the molecule name entered in the IN_GROW_molecule file. Likewise, the log file stem must be the same as the molecule name entered in the IN_GROW_molecule file. This syntax assumes that the *ab initio* package can be executed from the command line.
Line 12: blank line (included for clarity, can be omitted)
Line 13: the grow script **must** be restarted after the *ab initio* job has completed. This call assumes that you are working in a grow2.2 subdirectory e.g. grow2.2/water/

15. Trouble-shooting

Grow2.2 is still very much a work-in-progress, and is by no means unbreakable. A number of possible complications may arise, and some of these (the ones we know about or can anticipate) are addressed below:

1. One (or all) of the fortran programs crash with a "segmentation fault" or other memory allocation error. The most likely cause of such behaviour is that the array dimension limits are inappropriate (too small). You will need to edit the *.inc files in each of the sourcecode/ subdirectories, and re-make the executables. Important parameter statement variables that you may need to change are:
   - natomm = the maximum number of atoms allowed in the system
   - ndatam = the maximum number of data points allowed in the POT file
   - ngroupm = the maximum allowed number of elements in the symmetry group
   - maxt = the maximum allowed number of configurations in the TOUT file. If this number is exceeded, you should decrease the frequency of print outs by increasing the number of time steps between prints for trajectories or decreasing the number of blocks run for qdmc while increasing the block size to keep the total number of steps constant.
   - nmax = the maximum allowed number of trajectories that can be specified in IN_TRAJ. You can make this as big as your RAM allows.
   - maxfo = the maximum allowed number of data points in the outer neighbour list
   - maxf = the maximum number of allowed data points in the inner neighbour list
2. For large systems with a large number of data points in the PES data set, the qdmc90 code may require more memory than you have available to your machine. The solution is to use the 'intermediate memory' version of the code we have supplied in the directory sourcecode/qdmc90_intmem. Note: this version will only work with large data sets.
3. Trajstart crashes, complaining about the presence of identical data points. Remove the identical data points and change the number in the IN_ISEED file then continue.

4. Even after addition of 100 (or more) data points, the configuration space is not being explored adequately during the trajectory simulation, due to either a) no reaction or b) trajectories failing to conserve energy:
    a. The most likely cause of the trajectories failing to react is that the reaction probability is low at the energy at which you are growing the PES. This is unsatisfactory, since Grow will be unable to add data in the product valley. Possible solutions include:
        i. Increase the number of trajectories specified in IN_TRAJ to more than 10 (say, around 100)
        ii. Terminate Grow. Edit the IN_TRAJ file to vary the energy of the fragments and/or the relative translational energy. Run small sets of trajectories to see if the reaction probability can be increased in this way. If so (say at least 1 in 10 trajectories react), then change IN_TRAJ.small accordingly and restart Grow.
        iii. Terminate Grow. Change IN_EMS, IN_TRAJ and IN_SYSTEM to correspond to the reverse reaction. Run small sets of trajectories to see if the reaction probability for the reverse reaction is at least 10%. If so, change IN_TRAJ.small accordingly and restart Grow to construct the PES from the reverse reaction, at least for around 100 data points.
        iv. Terminate Grow. Change IN_EMS, IN_TRAJ and IN_SYSTEM to correspond to starting trajectories at the saddle point (remember to constrain the appropriate bond in IN_EMS). Run small sets of trajectories to see if both possible asymptotes are reached by the trajectories. If so, change IN_TRAJ.small accordingly and restart Grow to construct the PES from the saddle point, at least for around 100 data points.
    b. After the addition of 100 or more data points, the trajectories may fail to conserve energy, as indicated by a number of ‘bad’ trajectories reported in OUT_SUMMARY. If this number is decreasing with the addition of further data points, you probably don’t need to take any action. However, if this number is remaining constant or increasing, you may need to:

i. Terminate Grow. Change the weight function in IN_INTERP from 1 part to 2 part.
ii. Change the neighbourlist parameters in IN_INTERP. Try decreasing the number of steps between updates of the inner and outer neighbourlists and/or decreasing the cutoff for inclusion of points on the neighbourlists (from 1.d-5 to 1.d-6 for example – an order of magnitude decrease).
iii. Look at the value of the allowed relative error in the energy conservation which you specified in IN_TRAJ.small (and IN_TRAJ.large). This error tolerance may be too low. You can probably accept energy variations of up to 1.d-4 or 1.d-3 while you are growing the PES.

Appendix 1. Interpolated PES

The PES is given by an interpolation of Taylor expansions centered at data points scattered throughtout the configuration space of the system. The surface can be constructed using all the interatomic distances, $\mathbf{R}=\{R_1, R_2,.., R_{N(N-1)/2}\}$ as the basis for the internal coordinates (N = the number of atoms). In practice, we use inverse distances, **Z**, rather than bond lengths, where $Z_k = 1/R_k$. A set of 3N-6 internal coordinates are constructed as linear combinations of the **Z**, with a new set of coordinates defined locally for each data point in the set. That is, the local internal coordinates are:

$$\zeta^{(\mathbf{X}_0)}(\mathbf{X}) \quad = \quad \tilde{\mathbf{U}}^{\mathrm{T}}\mathbf{Z} \quad . \qquad \text{(A.1)}$$

The $\zeta^{(\mathbf{X}_0)}(\mathbf{X})$ is a set of 3*N*-6 linearly independent internal coordinates, for **Z** sufficiently close to some data point $\mathbf{Z}_0 = \mathbf{Z}(\mathbf{X}_0)$.

Since the 3*N*-6 internal coordinates ζ are well defined and linearly independent at any nonplanar, non collinear geometry **X(i)**, the Cartesian derivatives and second derivatives of the PES at **X(i)** can be transformed into $\frac{\partial V}{\partial \zeta_i}$ and $\frac{\partial^2 V}{\partial \zeta_i \partial \zeta_j}$:

$$\left.\frac{\partial \mathrm{V}}{\partial \mathrm{X_n}}\right|_{\mathbf{X(i)}} = \sum_{\mathrm{k=1}}^{\mathrm{3N-6}} \frac{\partial \zeta_\mathrm{k}}{\partial \mathrm{X_n}} \left.\frac{\partial \mathrm{V}}{\partial \zeta_\mathrm{k}}\right|_{\zeta\mathrm{(i)}} \quad ; \qquad \text{(A.2a)}$$

$$\left.\frac{\partial^2 \mathrm{V}}{\partial \mathrm{X_n}\partial \mathrm{X_m}}\right|_{\mathbf{X(i)}} = \sum_{\mathrm{k=1}}^{\mathrm{3N-6}} \sum_{\mathrm{j=1}}^{\mathrm{3N-6}} \frac{\partial \zeta_\mathrm{k}}{\partial \mathrm{X_n}} \frac{\partial \zeta_\mathrm{j}}{\partial \mathrm{X_m}} \left.\frac{\partial^2 \mathrm{V}}{\partial \zeta_\mathrm{k}\partial \zeta_\mathrm{j}}\right|_{\zeta\mathrm{(i)}} + \sum_{\mathrm{k=1}}^{\mathrm{3N-6}} \frac{\partial^2 \zeta_\mathrm{k}}{\partial \mathrm{X_n}\partial \mathrm{X_m}} \left.\frac{\partial \mathrm{V}}{\partial \zeta_\mathrm{k}}\right|_{\zeta\mathrm{(i)}} . \qquad \text{(A.2b)}$$

The calculations related to Eqs (A.1) and (A.2) are performed by the executable bfinvert.

If the data point configuration is (for example) near planar, some of the 3*N*-6 local internal coordinates, ζ, are nearly linearly dependent and inverting Eq. (A.2) to obtain the internal coordinates derivatives of the PES may be numerically unstable. To

avoid this problem, the molecular geometry is distorted (in a minimal sense) by the executable buckle to ensure that Eq. (A.2) is well behaved.

If the required energy and derivatives have been evaluated at each of N_d molecular configurations, a modified Shepard interpolation gives the potential energy at any configuration **Z** as a weighted average of the Taylor series about all N_d data points and their symmetry equivalents:

$$V(\mathbf{Z}) = \sum_{g\in G}\sum_{i=1}^{N_d} w_{g\circ i}(\mathbf{Z}) T_{g\circ i}(\boldsymbol{\zeta}) \quad , \qquad (A.3)$$

with $\boldsymbol{\zeta}$ given by Eq.(A.1). The Taylor expansion, T_i, about the data point $\boldsymbol{\zeta}(i) = \tilde{\mathbf{U}}^T \mathbf{Z}[\mathbf{X}(i)]$, is given by

$$T_i(\boldsymbol{\zeta}) = V[\boldsymbol{\zeta}(i)] + \sum_{k=1}^{3N-6}[\zeta_k - \zeta_k(i)]\left.\frac{\partial V}{\partial \zeta_k}\right|_{\boldsymbol{\zeta}=\boldsymbol{\zeta}(i)} + \frac{1}{2!}\sum_{k=1}^{3N-6}\sum_{j=1}^{3N-6}[\zeta_k - \zeta_k(i)][\zeta_j - \zeta_j(i)]\left.\frac{\partial^2 V}{\partial \zeta_k \partial \zeta_j}\right|_{\boldsymbol{\zeta}=\boldsymbol{\zeta}(i)} . \qquad (A.4)$$

This second order expansion is unchanged by an orthogonal transformation of the independent variables. Hence, if **L** is the matrix which diagonalises the matrix of second derivatives:

$$\mathbf{L}^T \frac{\partial^2 V}{\partial \boldsymbol{\zeta}^2}\mathbf{L} \quad = \quad \mathbf{K} \quad = \quad \mathrm{diag}(k_1, ..., k_{3N-6}) \quad , \qquad (A.5)$$

and we define $\delta\boldsymbol{\eta}(i)$:

$$\delta\boldsymbol{\eta}(i) = \boldsymbol{\eta} - \boldsymbol{\eta}(i) = \mathbf{L}^T[\boldsymbol{\zeta} - \boldsymbol{\zeta}(i)] \qquad (A.6a)$$

$$= \mathbf{L}^T \tilde{\mathbf{U}}^T [\mathbf{Z} - \mathbf{Z}(i)] \qquad (A.6b)$$

$$= \mathbf{M}[\mathbf{Z} - \mathbf{Z}(i)] \quad , \qquad (A.6c)$$

then we can write the Taylor expansion as a sum over fewer terms:

$$T_i(\xi) = V[\eta(i)] + \sum_{k=1}^{3N-6} [\eta_k - \eta_k(i)] \left.\frac{\partial V}{\partial \eta_k}\right|_{\eta=\eta(i)} + [\eta_k - \eta_k(i)]^2 \left.\frac{\partial^2 V}{\partial \eta_k^2}\right|_{\eta=\eta(i)} \quad . \qquad (A.7)$$

This orthogonal transformation and associated, simpler, Taylor expansion is also worked out by bfinvert.

The weight function, w_i, which gives the contribution of the ith Taylor expansion to the potential energy at the configuration **Z**, is discussed below. In Eq.(A.3), G denotes the symmetry group of the molecule; typically the Complete Nuclear Permutation Inversion (CNPI) group, or some subgroup of feasible permutations. Here, $g \circ i$ denotes that the ith data point, **Z**(i), is transformed by the group element g. The sum over g ∈ G means that all permutationally equivalent data points are included in the data set; the energy derivatives at permuted data points being simple permutations of the original derivatives. The data set is "symmetrised", so that the PES of Eq.(A.3) exhibits the full molecular symmetry.

The form of the weight function has been discussed in detail elsewhere. In qualitative terms, we would like the Taylor series, T_i, which is most accurate at **Z** to have the highest weight in Eq.(A.3). However, since we do not know *a priori* which Taylor series is most accurate, a procedure for determining the relative weight for each data point must be established. Firstly, the weights are normalised (sum to unity) by setting

$$w_i(\mathbf{Z}) = \frac{\nu_i(\mathbf{Z})}{\sum_{g \in G} \sum_{k=1}^{N_d} \nu_{g \circ i}(\mathbf{Z})} \quad . \qquad (A.8)$$

One can show that Eq.(A.3) is an interpolation of the energy and of the first and second derivatives at the data points, and that Eq.(A.3) becomes exact in the limit of infinite data density for appropriate choices of the weight functions.

To bias the weight function towards the Taylor expansions most likely to be accurate, the unnormalised weight function, v_i, is given by

$$\nu_i(\mathbf{Z}) = \left\{ \left[\sum_{k=1}^{N(N-1)/2} \left(\frac{Z_k - Z_k(i)}{d_k(i)} \right)^2 \right]^q + \left[\sum_{k=1}^{N(N-1)/2} \left(\frac{Z_k - Z_k(i)}{d_k(i)} \right)^2 \right]^p \right\}^{-1} . \qquad (A.9)$$

where q = 2 and p = 12 (are the usual values read in from IN_INTERP) The quantities $\{d_k(i),\ k=1,..,N(N-1)/2\}$ define a confidence volume about the ith data point. If $\sum_{k=1}^{N(N-1)/2} \left(\frac{Z_k - Z_k(i)}{d_k(i)} \right)^2 << 1$, then the weight of the ith data point at $\mathbf{Z}$ varies only with the low power q, while if $\sum_{k=1}^{N(N-1)/2} \left(\frac{Z_k - Z_k(i)}{d_k(i)} \right)^2 >> 1$, the weight of the ith data point is rapidly damped by the high power, p. The confidence lengths, $\{d_k(i)\}$, are determined by a Bayesian analysis of the inaccuracy of the ith Taylor expansion at M configurations close to $\mathbf{Z}(i)$:

$$d_n(i)^{-6} = \frac{1}{M} \sum_{k=1}^{M} \frac{\left\{ \left[\frac{\partial V[\mathbf{Z}(k)]}{\partial Z_n} - \frac{\partial T_i[\mathbf{Z}(k)]}{\partial Z_n} \right] [Z_n(k) - Z_n(i)] \right\}^2}{E_{tol}^2 \|\mathbf{Z}(k) - \mathbf{Z}(i)\|^6} . \qquad (A.10)$$

The $\{\mathbf{Z}(k)\}$ are taken to be the nearest M (read from IN_INTERP) neighbouring data points of the remaining $|G| \infty (N_d-1)$ independent points in the data set. The error tolerance (read from IN_INTERP), E_{tol}, defines the accuracy required for $\mathbf{Z}$ to lie within the confidence volume of $\mathbf{Z}(i)$. Since the set of $|G| \infty (N_d -1)$ independent points is totally symmetric with respect to the symmetry group, the "confidence length" d_n(i) has the same value as that associated with the bond $g \circ n$ at the data point $g \circ \mathbf{Z}(i)$. Hence, the confidence lengths need only be evaluated at one version of

each data point. As discussed elsewhere, the accuracy of Eq.(A.3) is relatively insensitive to the values of M and E_{tol}:

The location of the data points in Eq.(A.3) has been determined using the iterative methods developed previously. In summary, an initial set of data points is chosen to lie on or near the relevant reaction paths. The potential of Eq.(A.3) is then well defined in the vicinity of the reaction paths. Classical trajectories are evaluated, with initial conditions appropriate to the reaction(s) of interest, to explore the relevant region of configuration space. Molecular configurations encountered during these trajectories are recorded. One or more of these configurations is then chosen to be a new data point; the *ab initio* energy, gradient and second derivatives are evaluated at that point which is then added to the data set, generating a new version of the PES. This process of simulating the reaction(s), choosing a configuration, performing the *ab intio* calculations and adding a new data point to the set is repeated again and again until the PES is "converged". Convergence is established by performing large scale classical simulations of the reaction(s) of interest periodically during the "growth" of the data set. When the observable properties of interest, eg a reaction probability, do not change with increasing data set size, the PES is taken to be converged.

The methods for choosing a new data point at each iteration have been discussed in detail elsewhere. The "variance sampling" method places data points at configurations where the uncertainly in Eq.(A.3) is highest. The "h weight" method attempts to place data in regions where the trajectories often visit, but where few data points are already present. In the current Grow script we have used both "variance sampling" and "h weight" methods to choose data points following each sampling cycle.

Appendix 2. Simple one-dimensional interpolation illustration

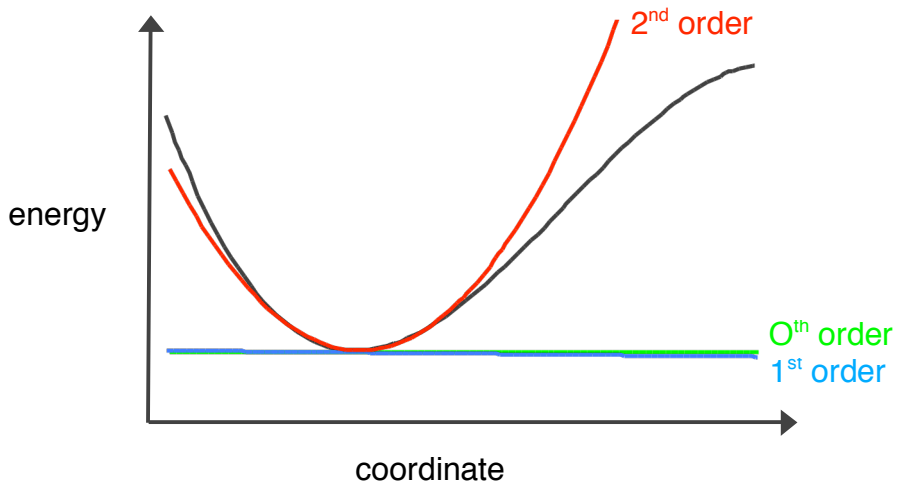


Figure 1. The zeroth, first and second order Taylor expansions about the global minimum

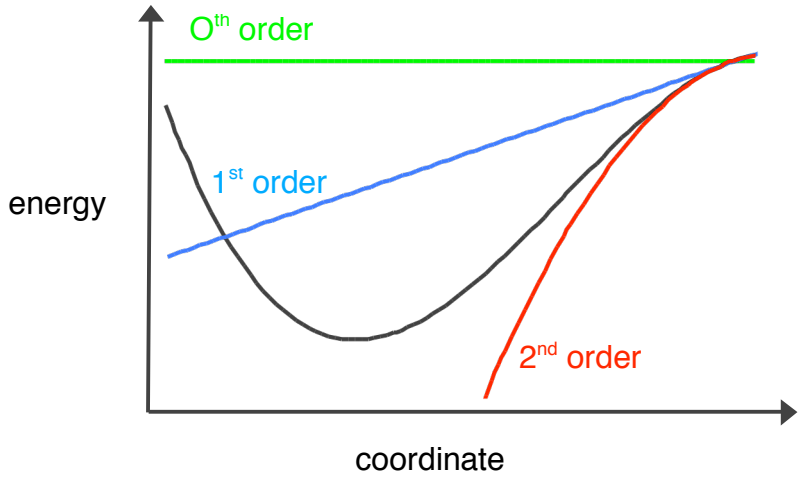


Figure 2. The zeroth, first and second order Taylor expansions about a point far from equilibrium

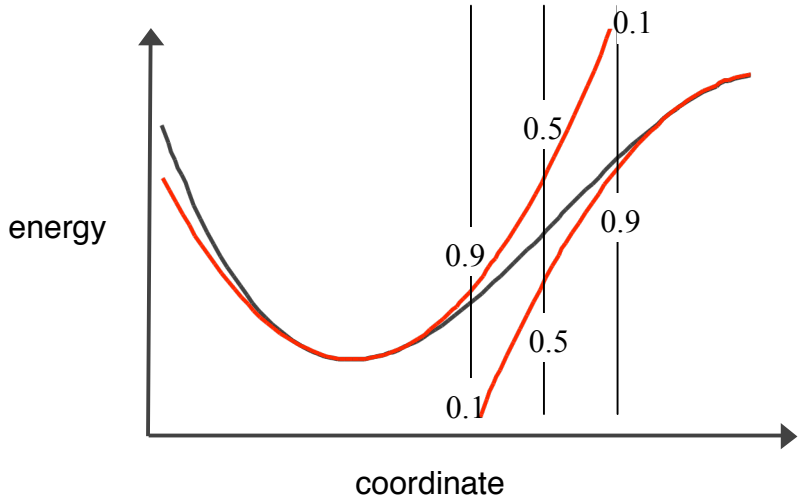


Figure 3. The relative weights of each Taylor expansion at points one quarter, one half and three quarters distance between the points at which the Taylor expansions are based.

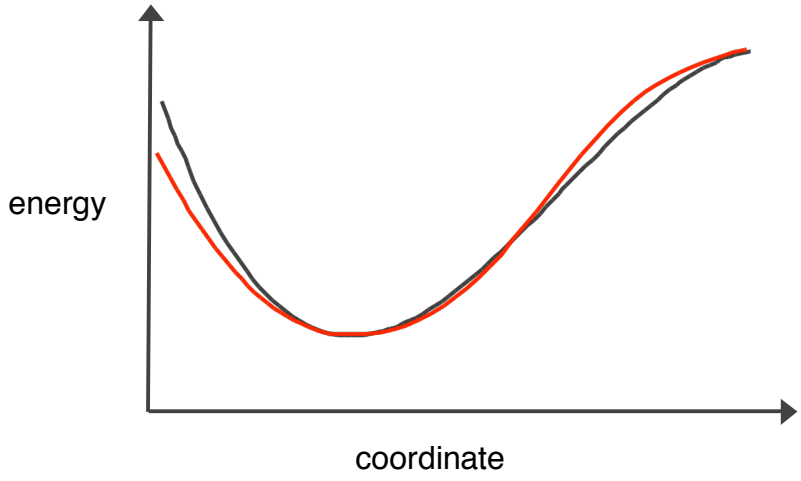
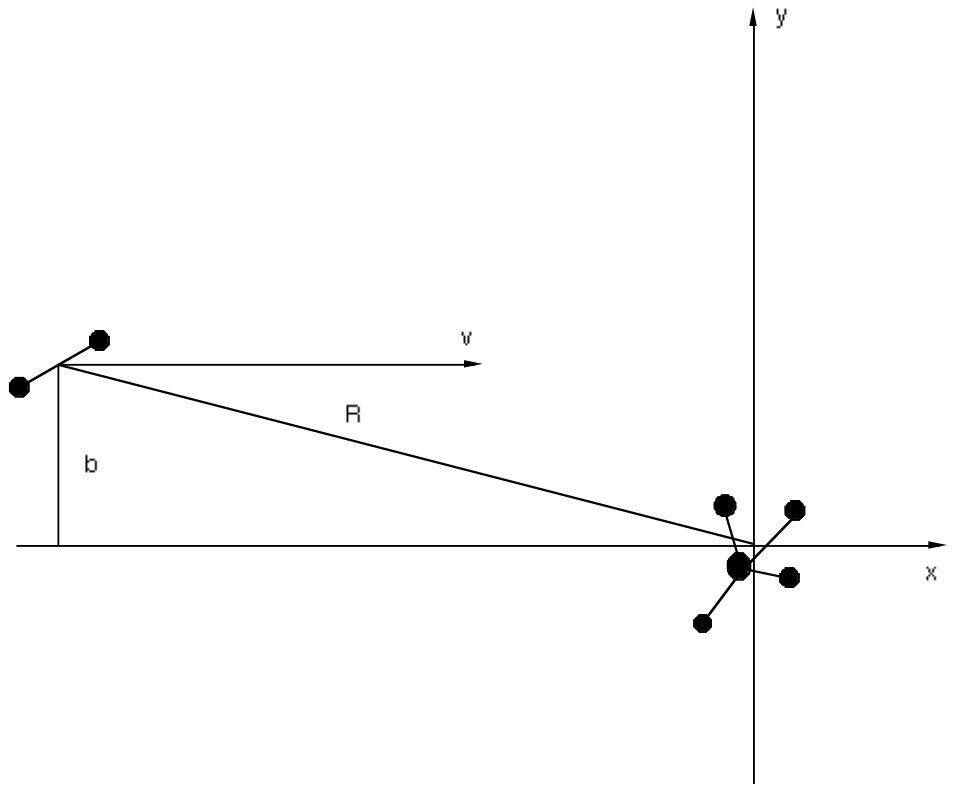


Figure 4. Comparison of the two-point interpolated PES with the actual PES.

Appendix 3. The treatment of classical bimolecular collisions

For simplicity, we can consider one of the reactant molecular fragments as being at rest, with its centre of mass at the origin of a Cartesian axis system. The other fragment is a long way away from the origin, with its centre of mass in the xy plane. Typically, we could put this second centre of mass at some large negative value of x as shown in the figure.



The second fragment, initially at some large negative x position, is given a y coordinate value of b, called the impact parameter, and a velocity, v, in the +x direction. The impact parameter gets its name from the idea that if there was no interaction between the fragments, the second fragment would pass by the first fragment a distance b away. There is nothing artificial about this arrangement of the fragments; we can always arrange our axis system to make any collision look like this initially. When R (see Figure) is large, there is no interfragment force or potential.

If the fragments have masses m_a and m_b, then the reduced mass for the relative motion is m:

$$m = (m_a m_b)/(m_a + m_b).$$

The relative velocity of the pair is v, and the relative kinetic energy is E_{rel}:

$$E_{rel} = 1/2 \text{ m } v^2.$$

To simulate the range of possible gas phase collisions of these two molecules at a given E_{rel}, and given internal energies of the two fragments, we must evaluate the dynamics of a large number of such collisions which differ in their initial conditions as follows. The orientation of the two fragments must be chosen from a selection of random orientations. The impact parameter, b, must be chosen from a random distribution of values, for which the probability of selecting some b between b and b + db is proportional to $2\pi b\, db$ (if you rotate the figure about the x axis, the collision is the same, and the second fragment has swept out an anulus of area $2\pi b\, db$ in the yz plane of possible approaching second fragments). In practice, we only allow b to vary up to some maximum allowed impact parameter, as for larger values of b, the interfragment forces are never large, i.e. the fragments miss each other, and the collision is trivial. The internal structure of the two fragments, and the velocities of the fragment's atoms with respect to its own centre of mass, must be randomly selected according to some distribution. Here we take a microcanonical distribution for each fragment. The executable `trajstart` does all of the above.

According to classical mechanics, each atom in the system moves subject to Newton's second law:

$$m_i \frac{d^2 \mathbf{r}_i}{dt^2} = -\frac{\partial V}{\partial \mathbf{r}_i},$$

where $i = 1,...,$ natom, and $\mathbf{r}_i = (x_i, y_i, z_i)$ is the cartesian coordinate vector of atom i.

Starting from a configuration like that in the figure, the classical equations of motion for the atoms are solved using a standard "velocity Verlet" algorithm. Time is broken down into finite steps to solve the equations of motion for the collision of these two molecular fragments. As time is incremented in steps, the forces on the atoms are evaluated from the partial derivatives of the interpolated PES with respect to the

atomic cartesian coordinates, and the atomic velocities and positions are adjusted according to the equations of motion. This calculation of the derivatives of the PES (and the PES value) at each time step is the cpu time consuming part of the classical simulation. Some parameters are set in IN_INTERP to make this task as fast as possible. Firstly, the PES formula in Appendix 1 gives the PES as a sum over weighted Taylor expansions about **all** the data points. However, at any particular time step, most of the data points have negligible weights and we do not evaluate their Taylor expansions if their weight falls below some tolerance, wtol. The data points with large enough weights form the "neighbour list". Since, the weights only change slowly with time, the neighbour list changes slowly and is only re-evaluated every so many time steps. Actually, you have to enter, in IN_INTERP two "numbers of time steps" that govern how frequently an "inner" and "outer" neighbour list are re-evaluated. You will find that you also have to enter values for some parameters which determine how the "weights" for each data point are evaluated. The precise meaning of these parameters is discussed in Ref 7 of the Introduction.

When the fragments "hit" each other, they may exchange some atoms (i.e. react), before they fly apart (since they start apart with some relative velocity, they always have enough energy to fly apart, although this may take a long time to occur). When they are far enough apart (say they have reached a distance R again), the interfragment forces are negligible. It is then possible to measure quantities like the rotational angular momentum of the fragments, their relative velocity, etc. The trajectory program interptraj prints out a file, OUT_FINALCV, which contains the cartesian positions and velocities of all the atoms when the fragments have again reached their initial separation.

Classical simulations with this code

Of course, once you have constructed a PES (POT file) you can run classical trajectory simulations independently of the Grow script.

You should:

(i) copy IN_TRAJ.large to IN_TRAJ , i.e. execute

```
cp IN_TRAJ.large IN_TRAJ
```

(ii) Edit IN_TRAJ (see section 6i) to specify the number of trajectories, the energies of the fragments, etc.

(iii) Create the initial conditions for the trajectories by executing

```
trajstart > inic
```

where inic can be any convenient file name.

(iv) Evaluate the classical trajectories by executing

```
interptraj < inic > somefilename
```

Generally, there is no useful output in somefilename. See Chap. 5 for a description of the output from interptraj.

Notes

Some notes on how to evaluate classical cross sections are given below. Sometimes, you may want to repeat one or more trajectories from a larger number that you have evaluated. For example, you may discover, from peering at the OUT files that one trajectory does something odd/interesting, and you would like to look at this trajectory in more detail. You can retrieve the initial conditions for this trajectory from the inic file (just edit the file). Then you might want to change the frequency with which information is printed during a trajectory (number of steps between prints in section 6i). Do not change the timestep for the trajectory, or you will change the trajectory, since the trajectory integration is only approximate. The TOUT file can be used to produce suitable input for a movie of the trajectory with the utility program xmolmovie.

Classical cross sections

It is a simple matter to calculate the classical cross section for a particular reaction from a set of trajectories evaluated in interptraj. The probability or relative frequency of some reaction in the trajectories can be found from OUT_SUMMARY. For example, a 1000 trajectories for $BH^+ + D_2$ might give an OUT_SUMMARY file like this:

number of trajectories with low energy = 0

number of unfinished trajectories = 0
number of bad trajectories = 0

Counts	Fragments
489	12 34
89	134 2
192	123 4
170	124 3
29	14 23
31	13 24

Here 1 = B, 2 = H, 3 = D and 4 = D.
Clearly 489 trajectories did not give reaction; 192 + 170 = 362 gave $BHD^+ + D$; 89 gave $BD_2^+ + H$; 29 + 31 = 60 gave $BD^+ + HD$. The maximum impact parameter for these 1000 trajectories was set in IN_CNT (in this example, it was 9 Bohr = 4.763 Å). Denoting this maximum impact parameter as b_{max}, and the relative frequency of the reaction of interest as f (a fraction between 0 and 1), the reaction cross section, σ, is given by

$$\sigma = f \pi b_{max}^2.$$

This is f times the total area (cross section) of the circle of radius b_{max}, which is the "target area" of the incoming fragment in the figure.

So, in this example, the cross section for $BHD^+ + D$ would be

$$\sigma(BHD^+ + D) = (362/1000)\,\pi\,(4.763)^2 \ \text{Å}^2.$$

$$= 25.8 \ \text{Å}^2$$

Note that this result is subject to a statistical error due to the fact that only a finite sample of trajectories was evaluated. A crude estimate of the standard deviation in the relative frequency, f, that could be expected from repeated samples of 1000 trajectories is d:

$$d = [f\,(1 - f)/1000]^{1/2}$$

The standard deviation for σ is $d\pi\ b_{max}^2$.

Thermal rate coefficients

Version 1.0 of Grow does not provide direct calculation of thermal rate coefficients. k(T) can be evaluated by taking a thermal average of the reaction cross section (approximately, as the cross sections evaluated above apply strictly only to molecules with initial zero rotational angular momentum). To find out how to evaluate this thermal average, see any standard text, e.g. J. I. Steinfeld, J. S. Francisco, and W. L. Hase, *Chemical Kinetics and Dynamics* (Prentice-Hall, Englewood Cliffs, NJ, 1989) Chap. 8, and R. N. Porter and L. M. Raff, in *Dynamics of Molecular Collisions*, edited by W. H. Miller (Plenum, New York, 1976) p. 1.

Appendix 4. The quantum diffusion Monte Carlo algorithm

(1) Establish a population of P_{norm} walkers by random displacement of each atom from its equilibrium position by up to 0.5 Bohr along the Cartesian axes.

(2) Move the atoms, with the atomic displacements sampled from a Gaussian distribution with variance, $v = 2\tau/mass$, where τ is the time step and $mass$ is the mass of the atom being moved (in atomic units).

(3) Calculate the potential energies of the displaced walkers

(4) Evaluate the ensemble energy, E_{ens}, as the average of the walker energies. If n (number of steps) > n_{equil} (number of steps to equilibrium), store the ensemble energy and bin the bondlengths into histograms.

(5) Calculate the trial energy, E_{trial}, according to the equation:

$$E_{trial} = E_{ens} - \left(\frac{P_{curr}}{P_{norm}} - 1.0 \right) \times fbp$$

where E_{ens} is the ensemble energy as defined above, P_{curr} is the current number of walkers, P_{norm} is the initial number of walkers created and fbp is the feedback parameter.

(6) Calculate the branching weight of each walker, according to the equation:

$w_i = e^{(E_{trial} - E_i)\tau} + rand()$

where E_i is the potential energy of the walker and $rand()$ is a random number between 0 and 1.

(7) Perform the branching according to the values of the branching weights:

- $w_i < 1$ → annihilate walker
- $1 < w_i < 2$ → leave walker
- $2 < w_i < 3$ → create 1 extra copy of the walker
- $3 < w_i$ → create 2 extra copies of the walker

Repeat steps 2 to 7 n_{equil} times until both the ensemble energy and trial energy remain relatively constant. Repeat steps 2 to 7 n_{prod} times until the error bars on your energy and nuclear vibrational wavefunction, as calculated using the blocking algorithm described by H. Flyvbjerg and H.G. Petersen (*J. Chem. Phys.* **91**, 461 (1989)) are sufficiently small.

Basically, this algorithm involves calculating the standard deviation in the data set following successive "blocking" transformations, where the blocking transformation is defined by averaging entries 1 and 2, 3 and 4, 5 and 6 etc. There exists a plateau in the value of the standard deviation following a sufficient number of "blocking transformations", given that the data set is sufficiently large. The value of the standard deviation at this plateau is taken as an upper bound for the standard deviation of the data set.

Vibrationally-averaged bondlengths are calculated using the descendant weighting algorithm, as described by M.A. Suhm and R.O. Watts (*Phys. Rep.* **204**, 293 (1991)). Basically, this algorithm involves the following steps:

(1) At regular intervals (say, once every n_{offset} time steps, a total of N_{gen} times) following equilibration, index the walkers and record their configurations.

(2) Propagate the walkers for n_{dw} time steps, keeping track of which walker is descended from which parent

(3) At the end of each generation, count the number of walkers (n) descended from each parent. Evaluate the expectation value of the vibrationally averaged bondlength according to the formula:

$$\langle r \rangle = \frac{\sum_{i=1}^{N_{walker}} r_i n_i}{\sum_{i=1}^{N_{walker}} n_i}$$

(4) Calculate the error in the vibrationally-averaged bondlengths as the standard devation in the mean, averaging over all the N_{gen} generations.