DL-FIND

A Geometry Optimiser for Atomistic Simulation Codes

Project Plan and User Manual

Revision
Johannes Kästner, Thomas Keal, Joanne Carr
STFC Daresbury Laboratory,
December 4, 2021

Note: the interface to the calling program is documented in section 3.1 on page 5.

1 General aim and target

DL-FIND is a modern and flexible structure optimiser to be included in electronic structure codes. It provides a stable way to find reaction energy differences as well as activation barriers. This can be performed starting from input structures in the region of the reactant and the product. The whole process should require as little action from the user as possible. The code should parallelise at least the energy and gradient evaluations and possibly expensive calculations within the optimiser. All units write restart information at regular intervals to enable a restart of the optimiser.

DL-FIND incorporates an MPI split-communicator taskfarming parallelisation framework. This is employed in the parallel optimisers (genetic algorithm and stochastic search), finite-difference Hessian evaluation, the nudged elastic band method and instanton calculations.

DL-FIND should be cited as: Johannes Kästner, Joanne M. Carr, Thomas W. Keal, Walter Thiel, Adrian Wander, and Paul Sherwood, *J. Phys. Chem. A*, 2009, 113 (43), 11856-11865.

A light-weighted description of some capabilities of DL-FIND can be found in the 2007 issue of Frontiers:

`http://www.cse.scitech.ac.uk/about_us/Frontiers2007/Kaestner - Finding Minima - CSE Frontiers 2007.pdf`

Excited state and parallel optimisation methods are covered in the 2009 Frontiers article:

`http://www.cse.scitech.ac.uk/about_us/Frontiers2009/Keal - Geometry Optimisation - Frontiers2009.pdf`

1.1 Functionalities

Emphasised: remains to be implemented

1.1.1 Coordinate systems:

- Cartesians (including frozen atoms and components)
- Mass-weighted cartesians (including frozen atoms and components)
- Internals (including all constraints):
  - DLC (delocalised internal coordinates)
  - DLC-TC (total connection)
  - HDLC
  - HDLC-TC
- *Fractional coordinates and unit cell optimisation?*
- *Parallel optimisation in internal coordinates?*

1.1.2 Combinations of coordinates (images):

- NEB (nudged elastic band)
- *(Growing) String method*
- Dimer method [1–3]
- *Replica path method following [4]*

All of the combinations should work with all versions of coordinate systems.

1.1.3 Optimisers:

- steepest descent
- conjugate gradient
- Newton-Raphson/quasi-Newton
- Damped dynamics
- L-BFGS
- P-RFO Hessian update mechanisms: Powell [5] and Bofill [6]. Hessian either by input or by finite-difference. In the latter case either in cartesians (then the update also in cartesians, and one can output frequencies), or in internals.

  A criterion can be specified with absolute eigenvalues of the hessian below that criterion are frozen (considered to be "soft"). At maximum 6 eigenmodes are frozen.
- *Reaction path following / IRC ?*

1.1.4 Microiterative optimisation for QM/MM

- Microiterative minimisation with L-BFGS for both regions
- Microiterative TS search with P-RFO inner region/L-BFGS environment
- Microiterative TS search with dimer inner region/L-BFGS environment
- Microiterative NEB with L-BFGS minimised environment

Microiterative optimisation methods improve the efficiency of QM/MM optimisation by separating the active atoms into an inner region (which should contain the QM region) and an outer environment. After each step taken in the inner region, the environment is relaxed completely. The idea is to minimise expensive inner region (macroiterative) evaluations at the cost of increasing the number of environment (microiterative) cycles.

Microiterative methods only save overall calculation time if QM region calculations are not performed during the microiterations. In the case of electrostatic embedding QM/MM calculations with ChemShell, the electrostatic influence of the QM region is approximated by fitting point charges at the QM atom sites to an electrostatic potential generated by the QM code.

In all cases the outer environment region is relaxed using L-BFGS. For the transition state and reaction path methods, this is equivalent to specifying spectator degrees of freedom (setting weights to zero) in a standard optimisation. This is useful for eliminating complications that can be caused by irrelevant degrees of freedom. For P-RFO this also means that the Hessian is only calculated over the inner degrees of freedom, which can dramatically reduce the cost of the Hessian calculation.

1.1.5 Line search algorithms:

- Simple scaling of the proposed step (covering the maximum step length)
- Trust radius based on energy decrease
- Trust radius based on the projection of the gradient on the step
- *Trust radius based on the overlap of the lowest eigenmode (for P-RFO)*
- *full line search ? which algorithm?*

1.1.6 Conical intersection optimisations:

- Penalty function
- Gradient projection
- Lagrange-Newton

1.1.7 Population-based search:

- Random (stochastic) search [7, 8]
- Genetic algorithm [9–11]

1.1.8 A global task manager:

Should define which methods are used depending on the input. In case a method fails, this should be recognised and another method should be tried.

1.1.9 Parallelisation:

The parallel optimisers (stochastic search and genetic algorithm), finite-difference Hessian evaluation and the nudged elastic band method can be run in taskfarming mode using MPI (message passing interface), where each taskfarm (workgroup) calculates the energy and gradient for a non-overlapping subset of the total required gradients. The load-balancing is static, and the number of workgroups must be a factor of the total number of processors for a given job. If the number of workgroups requested is less than the total number of processors, then the single-point energy and gradient calculations for each individual in a workgroup can also be parallelised. This is handled by the program that provides the energy and gradient routines. Thus, two-level parallelisation is possible (as implemented, for example, in the task-farming version of ChemShell).

Wrappers for the required MPI routines are located in the file dlf_mpi.f90, and the corresponding "dummy" subroutines are in dlf_serial.f90. Which of the corresponding .o object files is linked should depend on the build option chosen. For example, the default build of the standalone DL-FIND with its test driver program gives a serial executable. A `make parallel` command will produce the parallel-enabled standalone executable, Pfind.x. See makefile and Makefile.standalone for details.

Notes:

- If DL-FIND is in charge of disentangling the standard output from all the processors, then the variable `keep_alloutput` in `dlf_global_module.f90` is used. Rather than being an input option, it is hardwired in the code (so that it can be known before any output occurs). For example, `keep_alloutput` is used in `main.f90` (when a standalone program is made for testing) in the interface subroutine `dlf_output` as follows: if true, then for processor n ($n/=0$) in the global communicator (`MPI_COMM_WORLD`), the file output.proc$<n>$ is opened on unit `stdout`; if false, then for processor n ($n/=0$), /dev/null is "opened" on unit `stdout`. Output from the rank-zero processor in either case goes to standard out or a named file on unit `stdout`.

  If such I/O issues are dealt with by the main, calling program, then simply use the subroutine `dlf_output` to change the DL-FIND defaults (`stdout` and `stderr` stored in `dlf_global_module.f90`) for the required unit numbers, if necessary.

- Integer variables `mpi_rk` and `mpi_ik` are declared in `dlf_mpi_module` and set in subroutine `dlf_mpi_initialize` to the values of the MPI datatypes `MPI_Double_precision`

and `MPI_Integer`, respectively. This matches the use of real(rk) in DL-FIND's declarations, where rk = kind(1.d0). However, problems will arise if a compiler flag is used to change the nature of double precision numbers, as the MPI libraries will (probably) have been complied with a different specification for double precision. Could use `MPI_Sizeof` and `MPI_Type_match_size` instead if this turns out to be a common problem.

- Random numbers are an integral part of the parallel optimisers. Therefore, the seeding of the random-number generator is dealt with in dlf_parallel_opt.f90. Current policy is that only the rank-zero processor generates random numbers for the parallel optimisers. Only the routines that manipulate the population need random numbers, so currently only the rank-zero processor does any work in such routines.

1.1.10 Restart mechanism:

The optimiser should be fully restartable. Status: everything is restartable, except for the instanton part. There, most information during the runs is stored as well, but one can not directly start from the global restart files and continue exactly where the last run has ended (a practical solution, but not consistent with the rest of DL-FIND)

1.1.11 Compilation:

The portland compiler version 7.1 and 7.1-1 (and 7.1-2) do not compile the code properly. They do not save the contents of the hdlc derived type. This is a compiler bug. The portland compilers 6.1 and 7.0-4 do compile it (as well as v9 and later).

2 Quality Assurance Plan

An automated testing system (Buildbot) is used to test DL-FIND as part of the ChemShell and GAMESS-UK distributions. The ChemShell build of DL-FIND is tested with PGI Fortran, Intel Fortran, g95 and GNU Fortran.

3 Software Design Plan

Language: Fortran 95 + TR 15581 (technical report: meaning, allocatable arrays in derived types are used). Pointers are only to be used where strictly necessary.

3.1 Interface to the Calling Program / API

DL-FIND is designed as a library to be linked to quantum chemical or MD codes. However, for testing purposes, a driver module (`main.f90`) providing some analytic energy functions is used. DL-FIND is included in GAMESS-UK and ChemShell.

The routine to be called by the main program is `dl_find(nvarin,nvarin2,nspec,master)`, in the file `dl_find.f90`. It only returns after the complete geometry optimisation.

The main program has to provide the following routines, which are called from `dl_find.f90` and `dlf_formstep.f90`:

- `dlf_get_params(...)` Provide input parameters. The argument list is expected to change as DL-FIND is developed. New arguments should only be appended to existing ones which makes it possible to keep interfaces up to date. The arguments are documented in `dlf_global_module.f90`.
- `dlf_get_gradient(nvar,coords,energy,gradient,iimage,status)` Calculate the energy and gradient at the position coords.
- `dlf_get_hessian(nvar,coords,hessian,status)` Calculate the Hessian at the position coords.
- `dlf_get_multistate_gradients(nvar,coords, energy, gradient, coupling, needcoupling, iimage, status)` Calculate gradients of a pair of electronic states and optionally the interstate coupling between them. Required for conical intersection search.
- `dlf_put_coords(nvar,switch,energy,coords,iam)` Feeds a geometry back to the calling program. If switch is 1, coords contains the actual geometry. If switch is 2, coords contains the transition mode. The presence of iam allows the behaviour on the rank zero processor to be coded differently: for example, turning off the writing of coordinates to a file from all but the rank-zero processor.
- `dlf_error()` Error termination. Return to the calling program. `dlf_error` should not return.
- `dlf_update()` Allows the calling code to update any neighbour list, i.e. allows for discontinuities in the potential energy surface. This routine is called after a reset of the optimisation algorithm.

These may be C routines but have to be Fortran-callable. Examples of interface routines are available in `main.f90`, `dlf.c`, and `dlfind_gamess.m`.

If the taskfarming functionality is required, then the following additions to the main program should also be made:

- call `dlf_mpi_initialize()` to either have DL-FIND set up the global MPI communications, including calling `MPI_Init`, or to get the required parameters for the global communications that have already been set up in the main program. In the former case, this call should be placed before any I/O occurs, as close to the start of the main program as possible, as usual with `MPI_Init`. If the main program sets up the communications, then this call should be made after the rank of each processor and the total number of processors are known (ideally, immediately afterwards). This call is an obligatory addition to the main program.
- call `dlf_mpi_finalize()` to close down MPI before exiting the main program. Not necessary if the main program calls `MPI_Finalize` already; can be added safely after such a call in the main program (but is redundant).

- call `dlf_output(dlf_stdout,dlf_stderr)` passing unit numbers for I/O from the main program to DL-FIND. The call may be omitted if it is not necessary to change the defaults set in `dlf_global_module.f90`, and if the main program deals with the output from different processors.

- provide subroutine `dlf_output(dlf_stdout,dlf_stderr)`, if required, to set the DL-FIND variables stdout and stderr. If necessary, implement the strategy for dealing with standard output from the different processors here.

- provide subroutine `dlf_put_procinfo(dlf_nprocs,dlf_iam,dlf_global_comm)`. Called from `dlf_mpi_initialize` if `MPI_Init` was called there. Passes the total number of processors, the rank of the current process and a variable set to the handle for the global communicator (`MPI_COMM_WORLD`) to the main program.

- provide subroutine `dlf_get_procinfo(dlf_nprocs,dlf_iam,dlf_global_comm)`. Called from `dlf_mpi_initialize` if `MPI_Init` had already been called by the main program. The total number of processors, the rank of the current process and a variable set to the handle for the global communicator (`MPI_COMM_WORLD`) are passed from the main program to DL-FIND.

- provide subroutine `dlf_put_taskfarm(dlf_ntasks,dlf_nprocs_per_task, dlf_iam_in_task,dlf_mytask,dlf_task_comm,dlf_ax_tasks_comm)`. Called from `dlf_make_taskfarm` if DL-FIND sets up the split communicators. Passes to the main program the number of taskfarms (workgroups), the number of processors per farm, the rank of the current process in its farm, the rank of the current process's farm, and variables containing the handles for communicators within each farm and for the rank-*n* processor in each farm. The main program indicates to DL-FIND which of the two should set up the split communicators via an argument to the general interface routine `dlf_get_params`. `tdlf_farm` $= 0$ means the main program does the setup; `tdlf_farm` $\neq 0$ means DL-FIND does.

- provide subroutine `dlf_get_taskfarm(dlf_ntasks,dlf_nprocs_per_task, dlf_iam_in_task,dlf_mytask,dlf_task_comm,dlf_ax_tasks_comm)`. Called from dlf_make_taskfarm if the main program sets up the split communicators. The number of taskfarms (workgroups), the number of processors per farm, the rank of the current process in its farm, the rank of the current process's farm, and variables containing the handles for communicators within each farm and for the rank-*n* processor in each farm, are all passed from the main program to DL-FIND. The main program indicates to DL-FIND which of the two should set up the split communicators via an argument to the general interface routine `dlf_get_params`. `tdlf_farm` $= 0$ means the main program does the setup; `tdlf_farm` $\neq 0$ means DL-FIND does.

- ensure either `MPI_Abort` or `dlf_mpi_abort()` is called from the `dlf_error()` subroutine.

3.2 Program Units and Their Contents:

3.2.1 Modules

The code makes use of four modules that can be used from any subroutine throughout the DL-FIND: `dlf_parameter_module`, which only provides the real kind `rk`. It is located in the file `dlf_stat_module.f90`. `dlf_checkpoint`: variables and subroutines for reading and writing checkpoint files. `dlf_stat`: statistics. May be deleted and replaced in the future. `dlf_global`: contains many global parameters and also arrays (Cartesian coordinates, internal coordinates, the step, ...). All those parameters are part of a variable `glob`, which has a derived type (also defined in the module `dlf_global`). `pi`, `stdout`, and `stderr` are defined there as well.

All other modules must only be used within the file they are defined! If data of these module should be provided to other files (units), get- and set routines must be used!

3.2.2 Main Units:

- Main Unit (`dl_find.f90`)
- Convergence tester
- Coordinate transform
- Optimisation algorithms (`dlf_formstep.f90` and routines called from there)
- Scalestep: line search and trust radius approaches
- Utility units

3.2.3 Other details:

File units that are used longer than for an immediate write (> 1000): 1001 – 1001+nimage,max 1050 for xyz of NEB (`dlf_neb.f90`)

When arrays are allocated, the variables `glob%storage` and `glob%maxstorage` should be adjusted accordingly to enable control over the memory usage.

3.3 Program Documentation

The documentation is done using the robodoc tool. Each (important) subroutine should have an entry marking its connection within the code, the input and output variables of the global module, and its main purpose. An example file is `dlf_dimer.f90`.

Details of the documentation:

Fortran subroutines are to be documented as functions. The header gives the unit and the full subroutine name "NAME" should not be specified "SYNOPSIS" should be the actual code statement beginning the subroutine.

Module header names should be something short and descriptive rather than the actual Fortran name.

4 Code fragments obtained from external sources

Part of `dlf_lbfgs.f90` were obtained from `http://www.ece.northwestern.edu/ñocedal/lbfgs.html`.

The HDLC part has been taken from the HDLC optimiser in ChemShell. These are the files: `dlf_hdlc_constraint.f90` `dlf_hdlc_hdlclib.f90` `dlf_hdlc_interface.f90` `dlf_hdlc_matrixlib.f90` and `dlf_hdlc_primitive.f90`. Walter Thiel agreed to make them publically available (even under GNU license), as long as he gets DL-FIND for the MNDO code:

> [...] I have included the coordinate transformation part of the HDLC optimiser into DL-find. I expect HDLCs to work with the dimer method, and maybe they even work with NEB. I hope there are no objections from you as long as it only goes into ChemShell and gamess. However, I would like to make it available more broadly. It would be interesting to include it into other Daresbury codes (I mainly think of DL_POLY and possibly Crystal – if those people are interested, maybe the solid state people (around Walter Temmerman) are also interested). At a later stage, I think it would also be worth making DL-find with the HDLC coordinate transform available under the GNU license. Alternatively, I would have to make a DL-find version that runs without HDLCs.
>
> What do think about that? I.e. would you give us (DL) permission to distribute part of the HDLCopt code more widely?
>
> Dear Johannes,
>
> all this is fine with me. It is a good idea to make HDLCopt available to a wider audience. In return, I would like to include DL-find in the MNDO program (with no restrictions concerning its distribution). I had planned to add other optimisers to MNDO anyway, and it would obviously be good not to duplicate such work.
>
> Best wishes and Happy New Year,
>
> Walter Thiel

5 Test Plan

The driver program will enable tests of all functionalities with analytic energy function. Should noisy gradients be tested as well, a random part can be added to the gradient. Test cases from analytic 2-dimensional potentials showing minima and transition states (e.g. the Müller–Brown potential [12]) as well as clusters of Lennard–Jones particles will be used. The latter can be extended to systems with very many degrees of freedom.

6 Method documentation

This section documents non-standard implementations in DL-FIND. Published standard methods are not documented here.

6.1 Weights

A list of weights (one for each atom) can be specified in the input array coords2. It will be remapped onto weights of each degree of freedom (internal coordinates) to be optimised.

It can be used to restrict the NEB path to a certain set of atoms, or the direction of the dimer.

6.2 Instanton search and reaction rates

The code for calculating instantons is not documented in the ChemShell documentation for the time being. Thus, some user documentation is summarised here.

6.2.1 Work flow

1. **Location of a minimum and a saddle point on the potential energy surface** ("classical transition state") associated with the minimum (i.e. no barrier between the saddle point and the minimum). Hessians at both of these stationary points have to be calculated (ChemShell: `thermal=true`, DL-FIND: iopt=11). This writes files `qts_reactant.txt` and `qts_hessian_rs.txt` in case of the reactant, and `qts_ts.txt` and `qts_hessian_ts.txt` in case of the transition state. In case of a TS, the crossover temperature T_c is calculated:

$$T_\mathrm{c} = \frac{\hbar\omega_b}{2\pi k_\mathrm{B}} \tag{1}$$

   with ω_b being the absolute value of the imaginary frequency.

2. **Rates without tunnelling:** the files `qts_reactant.txt`, `qts_ts.txt`, and `class.in` have to be provided. The latter is an input file of the following format:
   first line: ignored
   second line: number of zero eigenvalues for the reactant and for the TS (e.g. "`6 6`")
   third line: starting temperature, end temperature, number of temperature steps (e.g. `300. 150. 20`).
   fourth line: "T" if bimolecular rates should be calculated, see below (6.2.2 on page 14)

   These files are required to run DL-FIND with `rate=true` (or iopt=13). The output (stdout or the file `arrhenius`) can directly be used in an Arrhenius plot: $1000/T$ in Kelvin, $\log_{10}$ of the classical rates in s^{-1} ($\mathrm{cm}^3\mathrm{s}^{-1}$ in case of bimolecular rates) calculated completely classical, with quantised vibrations (which includes the zero-point vibrational energy) and including tunnelling approximatively via the simplified Wigner correction:

$$\kappa(T) = 1 + \frac{1}{24}(\beta\hbar\omega_b)^2 = 1 + \frac{1}{24}\left(\frac{2\pi T_\mathrm{c}}{T}\right)^2, \quad \kappa(T_\mathrm{c}) = 1 + \frac{(2\pi)^2}{24} \approx 2.645 \tag{2}$$

   For temperatures above the crossover temperature T_c, the full Wigner-corrected rates is also given:

$$\kappa(T) = \frac{\beta\hbar\omega_b/2}{\sin(\beta\hbar\omega_b/2)} \tag{3}$$

   In the last column the exact analytical quantum rates for a symmetric Eckart barrier fitted to the particular system (barrier hight and ω_b) are shown. All degrees of freedom perpendicular to the reaction coordinate are approximated as quantum mechanical harmonic oscillators.

KIEs can be calculated directly by first running DL-FIND with `rate=true` (or iopt=13) on the Hessians for the light isotopologue. The file `arrhenius` of this run can be copied to `rate_H` to the directory where the rate with heavier isotopes is to be calculated. There, the same class.in as in the light case (at least the same temperature parameters) should be used. The rates obtained with the light nucleids is read and the KIE is directly calculated and written to a file called `kie`.

3. **Optimisation of the first instanton starting from the classical TS:** The file `qts_hessian_ts.txt` has to be renamed to `qts_hessian.txt`. All geometrical data are read in from `qts_hessian.txt`. However, `coords` and `coords2` still have to be provided (for historic reasons, number of atoms, ...), but are ignored (as in all instanton optimisations and rate calculations. A finite value of `distort` specifies how far the images will be spread along the unstable mode of the classical TS, see [13]. Newton–Raphson optimisation (`optimiser=NR` / iopt=20) is recommended [13, 14]. A QTS search is chosen by `qts=true` (ChemShell) or icoord=190 (DL-FIND). The NR optimiser is modified to avoid convergence to higher-order saddle points [14]. This avoids the collapse of the instanton path to the classical TS.

   Instanton searches are performed in mass-weighted coordinates with masses consistent with atomic units (electron mass, m_e). That is, the mass of a hydrogen atom (^{1}H) is 1837.15 m_e. This scales all distances up by a factor of 42.695 ($=$ (atomic mass unit$/m_e$)$^{1/2}$) compared to mass-scaled coordinates. Thus, the tolerance criterion (`tolerance`) has to be smaller by the same factor to achieve equivalent convergence. A tolerance of 10^{-7} (input as `1.E-7` in ChemShell) is usually sufficient, a tolerance of 10^{-8} is also often still possible. Since NR converges quadratically, the more stringent tolerance generally does not increase the number of steps dramatically.

   If NR (or P-RFO) is used, the updated Hessian will be used to calculate a preliminary estimate of the rate (if `qts_reactant.txt` is available). In that case, `qts_hessian_upd.txt` will be written, which contains only the updated Hessian. `qts_coords.txt` will in any case be written. It acts as input for subsequent recalculation of the Hessian and a rate calculation.

   Restarting instanton searches: Proper restart information (check files) is not written for the time being. Using NR, a restart is possible, though, by renaming `qts_hessian_intermediate.txt` (which is written after each step) to `qts_hessian.txt` and starting the simulation again. It will start from the Hessian and the geometry after the last full set of energies has been obtained.

4. **Instanton rate calculation:** `qts_coords.txt` from a previous instanton optimisation is read (`coords` and `coords2` are ignored). The temperature is also read from `qts_coords.txt`. The rate calculation is chosen by `qtsrate=true` or iopt=12. Hessians at all images and the rate are calculated as described in [14]. `qts_hessian.txt` is written, which acts as input for subsequent instanton optimisations.

   Restarting of rate calculations is also only possible by using the Hessian information written for each image in `qts_hessian_imageX.txt`. For these files to

be read, set `inithessian=6`.

5. **Next instanton optimisation in sequential cooling:** Starting from `qts_hessian.txt` at a previous (in general higher) temperature, another instanton is calculated. Distort should be zero, all other parameters are the same as in 3. The number of images may be increased. For optimal interpolation, the number of new images P_n should be related to the number of old images P_o by:

$$P_\mathrm{n} = k\,P_\mathrm{o} - k + 1 \tag{4}$$

with $k > 1$ being an integer. This ensures $k-1$ new images between each pair of old images.

6. **Instanton KIEs** can be calculated by starting out from a Hessian (`qts_hessian.txt`) for a different isotopologue and changing the masses in the input. The Hessian will be re-weighted accordingly. The instanton geometry has to be re-optimised. The file `qts_reactant.txt` obtained with changed masses can not be used. Instead, a file `qts_hessian_rs.txt` (which includes the masses, so from a recent version of DL-FIND) can be provided. The Hessian of the reactant obtained from that file will also be re-weighted.

   In an approximation (FPA) one can keep the instanton geometry fixed and just change the masses [15]. This is done by calculating an instanton rate with `inithessian=5` (read the Hessian from file rather than recalculating it) and changing the masses.

For lower temperature (compared to T_c) the number of images necessary can be kept at bay by adapting the integration grid (dtau) to the potential energy along the instanton path [14]. This only makes sense if the instanton path has reached the reactant minimum. It can be achieved by setting `nebk=1` (this is not interpreted as the NEB force constant, but as a parameter which can vary from 0 to 1. One corresponds to a fully adaptive grid).

If KIEs should be calculated with `rate=true` (or iopt=13), it is not necessary to recalculate the Hessian for the heavier isotopologue. The Hessians for the reactant and product are read in. If the masses provided via the calling code (ChemShell or main.f90) are different from the ones in the Hessian files, the Hessians with the new masses will be calculated.

In older versions of DL-FIND the masses were not written into the file `qts_hessian.txt`. The mass is needed there for calculating KIEs (unless one wants to recalculate the whole Hessian), however. A workaround is: delete everything below the second line in `qts_reactant.txt`. Rename the Hessian file for which masses are needed to `qts_hessian_rs.txt`, and run DL-FIND with `rate=true` (or iopt=13). It is important that the masses provided to DL-FIND by the calling code are the same as the ones used to calculate the Hessian. Then, DL-FIND will write a file `qts_hessian_rs_mass.txt` with the masses in, which can be used as `qts_hessian.txt` or `qts_hessian_rs.txt` in subsequent calculations.

An instanton rate can be calculated from an existing Hessian (i.e. from the file `qts_hessian.txt`) by setting `inithessian=5`.

Hessians of the individual images can be read in (all or just a part) from files `qts_hessian_imageX.txt`) by setting `inithessian=6`.

6.2.2 Bimolecular rates

Bimolecular rates are at the moment implemented in two ways. The simple one is for one atom reacting with a molecule. `qts_reactant.txt` refers to the reactant molecule. I.e. it has 3 degrees of freedom less than the classical TS. `qts_reactant.txt` has to be adapted manually: the energy of the incoming atom has to be added to the third line (which contains the energy of the reactant molecule). Additionally, the mass on the incoming atom (in atomic mass units) should be appended at the third line (thus, two real values in the third line).

The relative translational partition function of the incoming atom will be calculated and replaces the vibrational partition function for three degrees of freedom. The rate is internally calculated in atomic units (as is the case for uni-molecular reactions), but will be converted to molecules cm^3 s^{-1} upon output.

For two molecules reacting with each other with more than one atom in each, a second value has to be added to the third line of `qts_reactant.txt` as well. However, any negative value will do. It is only used as a label. The Hessians of the reactants are read from the files `qts_hessian_rs.txt` and `qts_hessian_rs2.txt` which are obtained from previous DL-FIND runs.

6.2.3 Tunnelling splittings

Tunnelling splittings of the vibrational ground state level following [16] can be calculated by setting `tsplit=true` in ChemShell or qtsflag=1 in DL-FIND. Every time a rate is calculated, the tunnelling splitting is calculated as well. Tunnelling splittings only make sense for symmetric molecules and barriers.

7 Documentation of the Input Options – User Documentation

System documentation is available in the source code after the subroutine headers. The input options will be explained here (User Documentation).

Print level (Variable `printl`):

0 no printout

2 print something

4 be verbose

6 debug

Type of coordinate system (Variable `icoord`):

"unit place" means `icoord` modulo 10

0–9 The whole system is to be treated as one image

Unit place 0 Cartesians

Unit place 1 HDLC - internals [17]

Unit place 2 HDLC - TC [17]

Unit place 3 DLC - internals [17]

Unit place 4 DLC - TC [17]

Unit place 5 Mass-weighted Cartesians ($\sqrt{m}x$) – will be deleted: use glob%massweight for that now!

1X Lagrange-Newton conical intersection search, with two extra coordinates corresponding to the gradient difference vector and interstate coupling gradient constraints.

10X NEB with endpoints free

11X NEB with endpoints moving only perpendicular to their tangent direction

12X NEB with frozen endpoints.

13X NEB with endpoints free. Only initialisation in coordiates X, optimisation in cartesians.

14X NEB with endpoints moving only perpendicular to their tangent direction. Only initialisation in coordiates X, optimisation in cartesians.

15X NEB with frozen endpoints. Only initialisation in coordiates X, optimisation in cartesians.

The NEB version implemented is the "improved-tangent" NEB [18] (also called "upwind scheme") with a climbing image.

190 Quantum transiton state search

20X Dimer method [1, 2]. Translation and rotation of the dimer are covered by the optimiser specified trough `iopt`. Requires two energy evaluation per iteration.

21X Dimer method. Rotation of the dimer is done by a line search within the dimer module, two energy calculations are used per rotation. Requires at least two energy evaluation per iteration.

22X Dimer method. Rotation of the dimer is done by a line search within the dimer module, one energy calculation is done per iteration, the other one is interpolated. Requires at least two energy evaluation per iteration.

30X "Chain" search (similar to NEB, but path is expanded in arbitrary basis functions). Better name required.

In all dimer versions: If a second set of coordinates is provided, it determines the dimer direction, if not, the dimer direction is randomised.

Multistate calculations (Variable `imultistate`):

0 Single state calculation (default)

1 Conical intersection optimisation (penalty function algorithm).

2 Conical intersection optimisation (gradient projection algorithm).

3 Conical intersection optimisation (Lagrange-Newton algorithm).

Type of optimisation algorithm (Variable `iopt`):

0 Steepest descent

1 Conjugate gradient following Polak–Ribière [19] (with automatic restarts based on the criterion by Powell and Beale) that is not coded properly ...

2 Conjugate gradient following Polak–Ribière [19] with CG restart every 10 steps (hardcoded at the moment)

3 L-BFGS [20, 21]

9 Test delta for finite-difference in gradients (19 energy and gradient evaluations)

10 P-RFO [22–25] A switching mechanism for the mode to be followed is included, but does not seem to help in any of the cases I tried sofar.

11 Just calculate the Hessian and do a thermal analysis (harmonic approximation for entropy, ...)

12 Calculate the Hessians of all images and the qTS rate (if inithessian=5: just the rate, read the Hessians)

13 Rate without tunneling (only with Wigner correction)

20 Newton–Raphson/quasi-Newton

30 Damped dynamics using the variables timestep, fric0, fricfac, and fricp. The frictions are defined that 0 corresponds to free (undamped) dynamics, and 1 corresponds to steepest descent.

51 Random (stochastic) search [7,8], using the variables po_pop_size, po_radius, po_contraction, po_tolerance_r, po_tolerance_g, po_distribution, po_maxcycle, po_scalefac.

52 Genetic algorithm [9–11], using the variables po_pop_size, po_radius, po_tolerance_g, po_maxcycle, po_init_pop_size, po_reset, po_mutation_rate, po_death_rate, po_nsave

Type of line search or trust radius (Variable `iline`):

0 simple scaling of the proposed step, taking maxstep into account

1 Trust radius based on energy as acceptance criterion (recommended for L-BFGS optimisation)

2 Trust radius based on gradient as acceptance criterion (recommended for CG optimisation)

3 Hard-core line search. Does not work at the moment...

Type of initial Hessian (Variable `inithessian`):

0 Calculate externally using `dlf_get_hessian`. Defaults to two point finite difference if an external Hessian is unavailable.

1 Build by one point finite difference of the gradient

2 Build by two point finite difference of the gradient

3 Build a diagonal Hessian with a single one point finite difference

4 Set the Hessian to be an identity matrix

5 Only for instanton calculations: read the Hessian from `qts_hessian.txt`

6 Only for instanton calculations: read the Hessian from files for each image`qts_hessian_imageX.txt`

Hessian update mechanism (Variable `update`):

0 No update. Always recalculate the Hessian

1 Powell update [5]

2 Bofill update [6]

3 BFGS update

Fragment and frozen atom information (Variable `spec`): The array `spec` has an entry for each atom. Meaning:

$>$**0** Fragment (residue) number the atom belongs to. Active in the optimisation.

0 Active, and treated in Cartesian coordinates, even if other atoms in the system are covered by (H)DLCs.

-1 Frozen

-2 x-component frozen (Cartesians only)

-3 y-component frozen (Cartesians only)

-4 z-component frozen (Cartesians only)

-23 x and y-components frozen (Cartesians only)

-24 x and z-components frozen (Cartesians only)

-34 y and z-components frozen (Cartesians only)

Atoms with `spec` < -1 will be completely frozen if used in HDLCs. One may use Cartesian constraints there to specify frozen components. Atoms with `spec` > 0 will be free in Cartesian coordinats.

After this array, `spec` also contains the following information:

nz entries of nuclear charges (same order as coords)

5*ncons entries of constraints (typ, atom1,atom2, atom3, atom4)

2*nconn entries of connections (atom1 atom2)

nat entries of microiterative region specification

i.e. `nspec= nat + nz + 5*ncons + 2*nconn + nat`

Microiterative optimisation (Variable `imicroiter`):

0 Standard (non-microiterative) optimisation

1 Microiterative optimisation

Note imicroiter is also used internally to keep track of whether the optimisation is in a macroiterative (imicroiter=1) or microiterative (imcroiter=2) loop.

The inner region specification for microiterative optimisation is part of the `spec` array with a section of length `nat`. A 0 entry signifies a standard optimisation or outer region as appropriate, while 1 signifies an inner region atom.

Other parameters There are numerous other paramers that can be set via the routine `dlf_get_params`. These are at the moment explained in the global module, `dlf_global_module.f90`.

7.0.1 Restarting

`dump` after how many energy and gradient evaluations is a restart file (checkpoint file) to be written? Default: 0 (never).

`restart` 0: new run (default), 1: read all checkpoint files and start from those.

It is only possible to restart a job with most parameters equal to those in the checkpoint file. Exeptions (input parameters that may be different from the ones in the previous run, i.e. that are not written to the checkpoint file): `maxcycle`. If different parameters are to be used, the latest geometry should be used and the optimiser should be started from scratch (`restart=0`).

8 Collection ...

Force weighted intenal coordinates: have a look at http://www.molpro.net/molpro-user/archive/all/msg00071.html

At the moment, I am experiencing problems with NEB in internal coordinates. While the structure may change continously, the internals may not. Consider H_2CO separation: it has two dihedrals. In the bound configuration, both are 180°. In the dissociated configuration, one is 0 and one is 180, as H_2 moves to one side of CO. How to deal with that?

One possible solution: Impropers are automatically put onto atoms with nearly planar configuration (what if more than 3 connections?). If an improper is placed on an atom, this one should not be in the middle of a dihedral that include these atoms. This, however, had be removed again as it leads to an underdetermined systems when not all atoms next to the planar atom are monovalent.

svn properties:

```
svn propset svn:keywords "URL Author Date Rev Id" dlf_util.f90
```

References

[1] G. Henkelman and H. Jónsson: A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives. *J. Chem. Phys.* **111**, 7010 (1999).

[2] A. Heyden, A. T. Bell, and F. J. Keil: Efficient methods for finding transition states in chemical reactions: Comparison of improved dimer method and partitioned rational function optimization method. *J. Chem. Phys.* **123**, 224101 (2005).

[3] J. Kästner and P. Sherwood: Superlinearly converging dimer method for transition state search. *J. Chem. Phys.* **128**, 014106 (2008).

[4] H. L. Woodcock, M. Hodošček, P. Sherwood, Y. S. Lee, H. F. Schaefer III, and B. R. Brooks: Exploring the quantum mechanical/molecular mechanical replica path method: a pathway optimization of the chorismate to prephenate Claisen rearrangement catalyzed by chorismate mutase. *Theor. Chem. Acc.* **109**, 140 (2003).

[5] M. J. D. Powell: . *Math. Prog.* **26**, 1 (1971).

[6] J. M. Bofill: Updated Hessian matrix and the restricted step method for locating transition structures. *J. Comput. Chem.* **15**, 1 (1994).

[7] S. H. Brooks: A discussion of random methods for seeking maxima. *Operations Research* **6**, 244 (1957).

[8] R. Luus, and T. H. I. Jaakola: Optimization by Direct Search and Systematic Reduction of the Size of Search Region. *AIChE Journal* **19**, (1973).

[9] J. H. Holland: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, (1975).

[10] D. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, New York, (1989).

[11] R. L. Haupt, and S. E. Haupt: Practical genetic algorithms. Wiley, New York, (1998).

[12] K. Müller and L. D. Brown: Location of saddle points and minimum energy paths by a constrained simplex optimization procedure. *Theor. Chim. Acta* **53**, 75 (1979).

[13] J. B. Rommel, T. P. M. Goumans, and J. Kästner: Locating instantons in many degrees of freedom. *J. Chem. Theory Comput.* **7**, 690 (2011).

[14] J. B. Rommel, J. Kästner: Adaptive Integration Grids in Instanton Theory Improve the Numerical Accuracy at Low Temperature. *J. Chem. Phys.* **134**, 184107 (2011).

[15] J. Meisner, J. B. Rommel, J. Kästner: Kinetic Isotope Effects Calculated with the Instanton Method *J. Comput. Chem.* **32**, 3456 (2011).

[16] J. O. Richardson and S. C. Althorpe: Ring-polymer instanton method for calculating tunneling splittings. *J. Chem. Phys.* **134**, 054109 (2011).

[17] S. R. Billeter, A. J. Turner, and W. Thiel: Linear scaling geometry optimization and transition state search in hybrid delocalised internal coordinates. *Phys. Chem. Chem. Phys.* **2**, 2177 (2000).

[18] G. Henkelman and H. Jónsson: Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.* **113**, 9978 (2000).

[19] E. Polak and G. Ribière: Note sur la convergence de directions conjugées. *Ref. Fra. Inf. Rech. Op.* **3**, 35 (1969).

[20] D. C. Liu and J. Nocedal: On the limited memory BFGS method for large scale optimization. *Math. Program.* **45**, 503 (1989).

[21] J. Nocedal: Updating quasi-Newton matrices with limited storage. *Math. Comp.* **35**, 773 (1980).

[22] C. J. Cerjan and W. H. Miller: On finding transition states. *J. Chem. Phys.* **75**, 2800 (1981).

[23] J. Simons, P. Jørgensen, H. Taylor, and J. Ozment: Walking on potential energy surfaces. *J. Phys. Chem.* **87**, 2745 (1983).

[24] A. Banerjee, N. Adams, J. Simons, and R. Shepard: Search for stationary points on surfaces. *J. Phys. Chem.* **89**, 52 (1985).

[25] J. Baker: An algorithm for the location of transition states. *J. Comput. Chem.* **7**, 385 (1986).