



Science & Technology
Facilities Council

Lecture 4 – Introduction to ChemShell

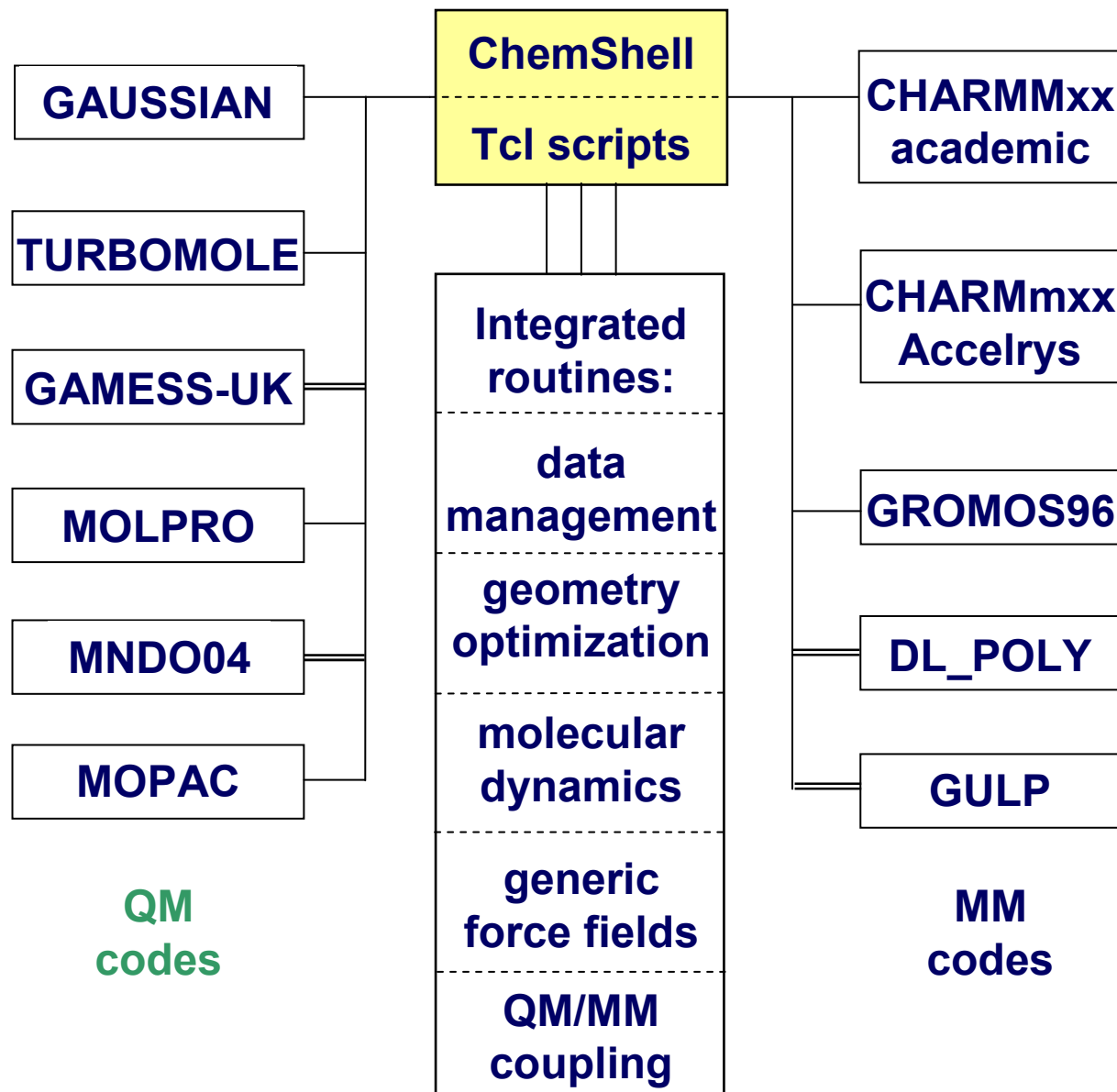
Paul Sherwood
p.sherwood@dl.ac.uk



- Background and Design
- Basic Command Structure
- Simple Tcl and ChemShell scripting
- Objects
 - *Fragment, zmatrix etc*
- Energy and Gradient evaluators
- Passing options to modules and sub-modules
- Loading objects, z-matrices and cartesian coords
- QM Code Interfaces
- MM Modules – DL_POLY, GULP, CHARMM
- Object input, output and caching
- Parallelisation
- Introduction to the practical session



- ChemShell arose because
 - There was a need to couple QM and MM functionality, to extend GAMESS-UK for complex systems
 - Desire for flexible control of complex jobs in a way not possible using traditional input files
- Different approaches to QM/MM
 - QM added as an extension to MM/MD forcefield
 - CHARMM/GAMESS-UK
 - MM environment added to a small-molecule treatment
 - ONIOM (G98, G03)
 - GAMESS-UK/AMBER
 - Gaussian/AMBER (Manchester)
 - Modular scheme with a range of QM and MM methods
 - emphasis on flexibility
 - specialised optimisation algorithms
 - e.g. ChemShell





- ChemShell comprises code in
 - **Tcl**
 - Control scripts and user input files
 - Interfaces to 3rd party executables e.g. Turbomole
 - User-written extensions
 - **C**
 - implementations of new Tcl commands
 - Object management (fragment, zmatrix, matrix, field)
 - Tcl and C APIs
 - I/O
 - **Fortran77 and Fortran90/95**
 - QM and MM codes: GAMESS-UK, GULP, MNDO97, DL_POLY
 - MD and Geometry Optimisation drivers



- The main author of ChemShell is Paul Sherwood.
- The MD and MM modules are based on code taken from the DL_POLY package written by W. Smith.
- The current project combines software development work of three academic groups active in the area
 - Daresbury Laboratory (UK)
 - the group of Prof. Walter Thiel at the Max Planck Institute für Kohlenforschung, Mülheim (DE)
 - the group of Prof C.R.A. Catlow at the Royal Institution (UK).
- Some significant individual contributions:
 - Alex de Vries (QM/MM models, newopt optimiser)
 - Alex Turner and Salomon Billeter (HDLC optimiser).
 - Stephan Thiel (GROMOS interface)
 - Johannes Kästner (QM/MM-FEP, New optimisation methods)
 - Hans Martin Senn (Nose-Hoover chain thermostat).
 - Alexey Sokol (solid-state embedding methods and associated cluster preparation routines)
- ChemShell forms the basis of the QUASI software developed under an EU project. Contributions of the QUASI project partners and financial support of the CEC is gratefully acknowledged, as is financial support from Shell KSLA (Amsterdam).



- ChemShell control files are Tcl Scripts
 - Usually we use a **.chm** suffix
- ChemShell commands have some additional structure, usually they take the following form

`command arg1=value1 arg2=value2`

Arguments can serve many functions

`mm_defs=dl_poly.ff` Identify a data file to use

`coords=c` Use object `c` as the source of the structure

`use_pairlist=yes` Provide a Boolean flag (yes/no, 1/0, on/off)

`list_option=full` Provide a keyword setting

`theory=gamess` Indicate which compute module to use

Sometimes

`command arg1=value1 arg2=value2 data`



- Variable Assignment (all variables are strings)
`set a 1`
- Variable use
`[set a]`
`$a`
- Command result substitution
`[<Tcl command>]`
- Numerical expressions
 - `set a [expr 2 * $b]`
- Output to stdout
 - `puts stdout "this is an output string"`



- Lists - often passed to ChemShell commands as arguments

```
set a { 1 2 3 }
```

```
set a "1 2 3"
```

```
set a [ list 1 2 3 ]
```

\$ is evaluated within [list ..] and “ “ but not { }

- [list ...] construct is best for building nested lists using variables

- Arrays - not used in ChemShell arguments, useful for user scripts

- Associative - can be indexed using any string

```
set a(1) 1
```

```
set a(fred) x
```

```
parray a
```



- Continuation lines:
 - can escape the newline

```
tclsh % set a "this \  
  is \  
a single variable"  
this is a single variable  
tclsh %
```

- { } will incorporate newlines into the list

```
tclsh % set a {this  
  is  
a single variable}  
this  
  is  
a single variable  
tclsh %
```



■ Procedures

- Sometimes needed to pass to ChemShell commands to provide an action

```
proc my_procedure { my_arg1 my_arg2 args } {  
  puts stdout "my_procedure"  
  return "the result"  
}
```

■ Files

```
set fp [ open my.dat w ]  
puts $fp "set x $x"  
close $fp  
.....  
source my.dat
```



```
c_create coords=c {
C 1.0 0.0 0.5
.....
}

dynamics dyn1 coords=c theory=mndo : { hamiltonian=pm3 } \
    temperature=300 timestep=0.005
dyn1 initvel
set nstep 0
while {$nstep < 10000 } {
    dyn1 force
    dyn1 step
    .....
    # additional Tcl commands here
    incr nstep
}
dyn1 configure temperature=300
# etc
dyn1 destroy
```



- ChemShell object types
 - **fragment** - molecular structure
 - creation: `c_create`, `load_pdb` Universal!!
 - **zmatrix**
 - `z_create`, `newopt`, `z_surface`
 - **matrix**
 - creation: `create_matrix`, energy and gradient evaluators, dynamics
 - **field**
 - creation: `cluster_potential` etc, graphical display, charge fits



- Between calculations, and sometimes between commands in a script, objects are stored as files. Usually there is no suffix, objects are distinguished internally by the block structure.

```
% cat c
block = fragment records = 0
block = title records = 1
phosphine
block = coordinates records = 34
p  4.451659000000000e+00  0.000000000000000e+00  -8.17756491786826e-16
c  6.185735500000000e+00  -2.30082107458395e+00  1.93061811508830e+00
c  8.21288557680633e+00  -3.57856465464377e+00  9.30188790875432e-01
c  9.49481331797844e+00  -5.31433733714784e+00  2.37468849115612e+00
c  8.74959098234423e+00  -5.77236643959209e+00  4.81961751564967e+00
.....
```

- Multi block objects are initiated by an empty block (e.g. fragment)
- Unrecognised blocks are silently ignored

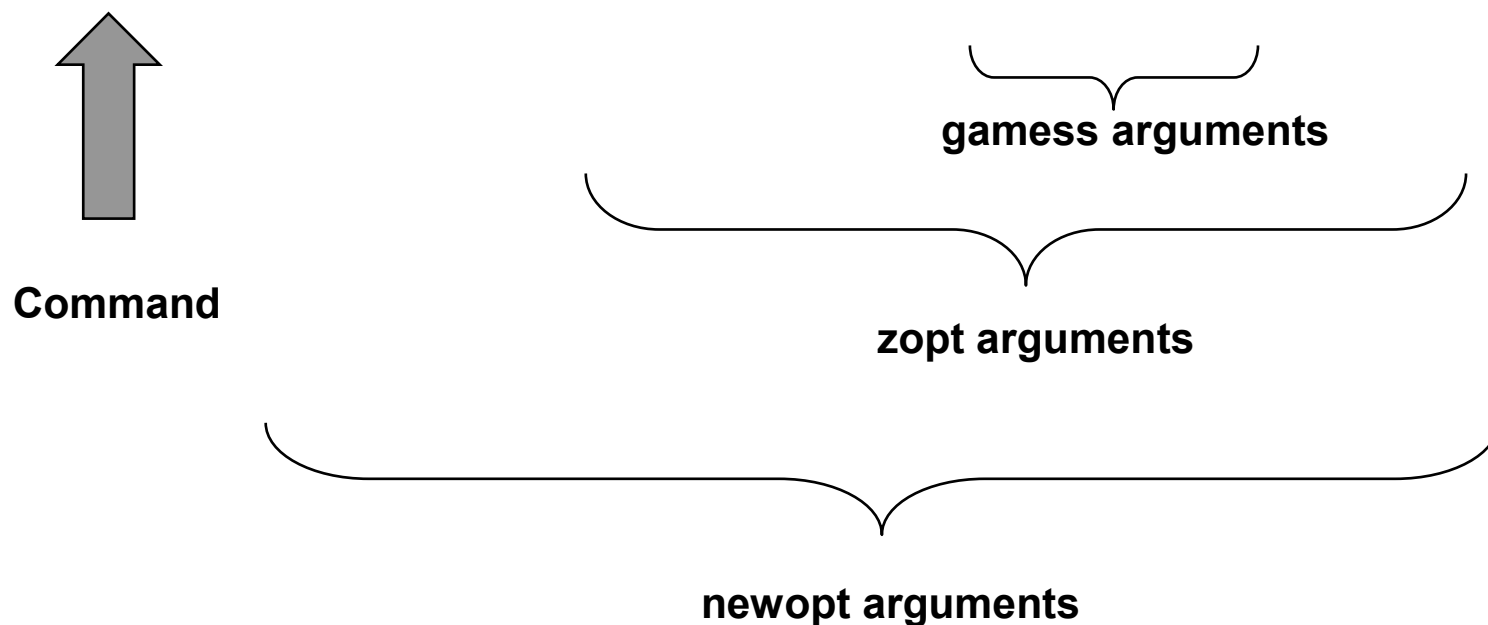


- Many modules are designed to work with a variety of methods to compute the energy and gradient. The procedure relies on
 - the interfaces to the codes being consistent, each comprises a set of callable functions e.g.
 - initialisation, energy, gradient, kill, update
 - the particular set of functions being requested by a command option, usually theory=
- Example evaluators (depends on locally available codes)
 - gamess, turbomole
 - dl_poly, charmm
 - mopac, mndo
 - hybrid
 - You can write your own in Tcl



- The : syntax is used to pass control options to sub-module, e.g. when running the optimiser, to set the options for the module computing the energy and gradient. {} can be used if there is more than one argument to pass on. Nested structures are possible using Tcl lists.

```
newopt function=zopt : {theory=gamess : {basis=sto3g} zmatrix=z }
```





```
z_create zmatrix=z {  
zmatrix angstrom  
c  
x 1 1.0  
n 1 cn 2 ang  
f 1 cf 2 ang 3 phi  
variables  
cn      1.135319  
cf      1.287016  
phi     180.  
constants  
ang     90.  
end  
}
```

```
z_list zmatrix=z
```

```
set p [ z_prepare_input  
zmatrix=z ]  
puts stdout $p
```

- z_create provides input processor for the z-matrix object
- z_list can be used to display the object in a readable form
- z_prepare_input provides the reverse transformation if you need something to edit
- z_to_c provides the cartesian representation



- Can include some atoms specified using cartesian coordinates
- Can use symbols for atom-path values (i1,i2,i3)
- Can append % to symbols to make them unique (e.g. o%1)
- Can create/destroy and set variables using Tcl commands

```
z_create zmatrix=z1 {  
  zmatrix  
  o%1  
  o%2 o%1 3.  
  h%1 o%1 1.8   o%2 121.0  
  h%2 o%2 1.85 o%1 122.0 h%1 29.0  
end  
}  
z_var zmatrix=z1 result=z2 control= "release all"  
z_substitute zmatrix=z2 values= {r2=3.0 r3=2.0}  
z_list zmatrix=z2
```



- Combine cartesian and internal definitions

```
z_create zmatrix=z1 {  
  coordinates  
  ...  
  zmatrix  
  ....  
end  
}
```

- `c_to_z` will create a fully cartesian z-matrix



```
c_create coords=h2o.c {  
  title  
  water dimer  
  coordinates au  
  o  0.0000000000    0.0000000000    0.0000000000  
  h  0.0000000000   -1.4207748912    1.0737442022  
  h  0.0000000000    1.4207748912    1.0737442022  
  o -4.7459987607    0.0000000000   -2.7401036621  
  h -3.1217528345    0.0000000000   -2.0097934033  
  h -4.4867611522    0.0000000000   -4.5020127872  
}
```

- No symbols allowed
- Can also use read_xyz, read_pdb, read_xtl



- Crystallographic cell constants can be provided, along with fractional coordinates

```
#  
c_create coords=mgo.c {  
space_group  
1  
cell_constants angstrom  
4.211200 4.211200 4.211200 90.00 90.00 90.00  
coordinates  
Mg      0.10000000    0.00000000    0.00000000  
O       0.50000000    0.50000000   -0.50000000  
Mg      0.00000000    0.50000000   -0.50000000  
O       0.50000000    1.00000000   -1.00000000  
Mg      0.50000000    0.00000000   -0.50000000  
O       1.00000000    0.50000000   -1.00000000  
Mg      0.50000000    0.50000000    0.00000000  
O       1.00000000    1.00000000   -0.50000000  
}  
list_molecule coords=c  
set p [ c_prepare_input coords=c ]
```



- Alternatively, input the cell explicitly, in `c_create` or attach to the structure later

```
#
c_create coords=d {
title
primitive unit cell of diamond
coordinates au
c      0.8425347285      0.8425347285      0.8425347285
c      -0.8425347285     -0.8425347285     -0.8425347285
cell au
      0.00000      3.37014      3.37014
      3.37014      0.00000      3.37014
      3.37014      3.37014      0.00000
}
extend_fragment coords=d cell_indices= { -2 2 -2 2 -2 2 } result=d2
set_cell coords=d cell= {
      0.00000      3.37014      3.37014
      3.37014      0.00000      3.37014
      3.37014      3.37014      0.00000 }
```



- Provides access to third party codes
 - ❑ GAMESS-UK
 - ❑ MNDO
 - ❑ TURBOMOLE
 - ❑ Molpro
 - ❑ Gaussian98,03
- Interfaces
 - ❑ Standardised argument structure
 - hamiltonian (includes functional)
 - charge, mult, scftype
 - basis (internal library or keywords)
 - accuracy
 - direct
 - symmetry
 - maxcyc...



- Can be built in two ways

- Interface calls GAMESS-UK and the job is executed using `rungames`
 - parallel execution can be requested even if ChemShell is running serially
- GAMESS-UK is built as part of ChemShell
 - mainly intended for parallel machines

```
energy coords=c \  
  theory=gamess : { basis=dzp hamiltonian=b3lyp } \  
energy=e
```

- Notes

- The jobname is `gamess1` unless specified
- Some code-specific options
 - `dumpfile=` specify dumpfile routing
 - `getq =` load vectors from foreign dumpfile



- basisspec has the structure
 { { basis1 atoms-spec1} {basis2 atomspec2} }
- Assignment proceeds left to right using pattern matching for atom labels
- * is a wild card
- This example gives sto-3g for all atoms except o
- Library can be extended in the Tcl script (see <examples/gamess/explicitbas.chm>)
- ECPs are used where appropriate for the basis

```
energy coords=c \  
  theory=gamess : { basisspec = { { sto-3g *} {dzp o} } } \  
energy=e
```



■ Features

- ❑ Energy and gradient routines from DL_POLY (Bill Smith UK, CCP5)
- ❑ General purpose MM energy expression, including approximations to CFF91 (e.g. zeolites), CHARMM, AMBER, MM2
- ❑ Topology generator
 - automatic atom typing
 - parameter assignment based on connectivity
 - topology from CHARMM PSF input
- ❑ FIELD, CONFIG, CONTROL are generated automatically
- ❑ FIELD is built up using terms defined in the file specified by mm_defs= argument
- ❑ Periodic boundary conditions are limited to parallelepiped shaped cells
- ❑ Can have multiple topologies active at one time



- Terms are input using atom symbols (or * wild card)
- Individual keyword terms:
 - bond mm2bond quarbond angle
mm2angle quarangle ptor mm2tor htor
cftor aa-couple aat-couple vdw
powers m_n_vdw 6_vdw mm2_vdw
- Input units are kcal/mol, angstrom etc in line with most forcefield publications
- For full description see the manual



- Forcefield definition can incorporate connectivity-based atom type definitions which will be used to assign types
- Atom types are hierarchical, most specific applicable type will be used (algorithm is iterative)
- e.g. to use different parameters for ipso-C of PPh_3 define a new type by a connection to phosphorous

```
query ci "ipso c"  
supergroup c  
target c  
atom p  
connect 1 2  
endquery
```

```
charge c -0.15  
charge h 0.15
```



```
# dummy forcefield
read_input dl_poly.ff {
bond c c 100 1.5
bond c h 100 1.0
angle c c c 100 120
angle c c h 100 120
vdw h h 2500 1000000
vdw c c 2500 1000000
vdw h c 2500 1000000
htor c c c c 100 0.0 i-j-k-l
charge c -0.15
charge h 0.15
}

energy theory=dl_poly : mm_defs = dl_poly.ff coords=c
energy=e
```



- Forcefield parameters are computed on-the-fly depending on atomic data
- specify forcefield=uff
- Limitations
 - As yet, not a completely validated implementation (not recommended for published work)
 - Rather limited ability to assign atom types and bond orders (so not so universal...)
 - Not a very accurate forcefield anyway
- Used in the CCP1 GUI for MM cleaning of structures



- Replicates CHARMM energy expression (without UREY)
- Uses standard CHARMM datafiles
- Requires CHARMM program + script to run as far as energy evaluation for initial setup
- Atom charges and types are obtained by communication with a running CHARMM process (usually only run once)

```
# run charmm using script provided
charmm.preinit charmm_script=all.charmm coords=charmm.c
# Store type names from the topology file
load_charmm_types2 top_all122_prot.inp charmm_types
# These requires CTCL (i.e. charmm running)
set types [ get_charmm_types ]
set charges [ get_charmm_charges ]
set groups [ get_charmm_groups ]
charmm.shutdown
```



- Then provide dl_poly interface with
 - ❑ .psf (topology) charmm_psf =
 - ❑ .rtf (for atom types) charmm_mass_file=
 - ❑ .inp parameter files charmm_parameter_file=

```
theory=dl_poly : [ list \  
    list_option=full \  
    cutoff = [ expr 15 / 0.529177 ] \  
    scale14 = { 1.0 1.0 } \  
    atom_types= $types \  
    atom_charges= $charges \  
    use_charmm_psf=yes \  
    charmm_psf_file=4tapap_wat961.psf \  
    charmm_parameter_file=par_all22_prot_mod.inp \  
    charmm_mass_file= $top ]
```



- Simple interface to GULP energy and forces
- GULP licensing from Julian Gale
- GULP must be compiled in
 - (this has been included in the workshop versions)
- ChemShell fragment object supports shells
 - Shells are relaxed by GULP with cores fixed, ChemShell typically controls the core positions
- Provide forcefield in standard GULP format

- Practical note:
 - Adding the argument `use_second_derivatives=no` may be required if memory available is limited (otherwise GULP attempts to allocate and initialise some large arrays).



```
read_input gulp.ff {  
# from T.S.Bush, J.D.Gale, C.R.A.Catlow and P.D. Battle  
# J. Mater Chem., 4, 831-837 (1994)  
species  
Li core 1.000  
Na core 1.000  
...  
buckingham  
Li core 0 shel 426.480 0.3000 0.00 0.0 10.0  
Na core 0 shel 1271.504 0.3000 0.00 0.0 10.0  
...  
spring  
Mg 349.95  
Ca 34.05  
...  
}  
add_shells coords=mgo.c symbols= {O Mg}  
newopt function=copt : [ list coords=mgo.c theory=gulp : [ list  
mm_defs=gulp.ff ] ]
```



- Full functionality from standard academic CHARMM
- Dual process model
 - CHARMM runs a separate process
 - CHARMM/Tcl interface (CTCL, Alex Turner) uses named pipe to issue CHARMM commands and return results.
- commands to export data for DL_POLY and hybrid modules
 - atomic types and charges
 - neutral groups
 - topologies and parameters
- Access to QM/MM coupling models internal to CHARMM
 - GAMESS(US), MOPAC, GAMESS-UK



- In the tutorials you will encounter two main tools for geometry optimisation

newopt

- A general purpose optimiser (offers BFGS, Conmin etc)
 - Needs choice of a target functions, specified by “function = “
 - copt : cartesian
 - zopt: z-matrix (now also handles cartesians)
 - new functions can be written in Tcl (see example rosenbrock)

hdlcopt

- Hybrid Delocalised Internal Coordinate scheme (Alex Turner, Walter Thiel, Salomon Billeter)
 - Developed within QUASI project to give $O(N)$ overall scaling per step
- Residue specification, often taken from a pdb file (pdb_to_res) allows separate delocalised coordinates to be generated for each residue
- Can perform P-RFO TS search in the first residue with relaxation of the others
 - increased stability for TS searching and much smaller Hessians
- Further information on algorithm
 - S.R. Billeter, A.J. Turner and W. Thiel, PCCP 2000, 2, p 2177

- More methodological information tomorrow.



```
#  
# function zopt allows the newopt optimiser to work with  
# the energy as a function of the internal coordinates of  
# the molecule  
#  
  
newopt function=zopt : { theory=gamess : { basis=dzp } } \  
zmatrix=z
```



```
# functions zopt.* allow the newopt optimiser to work with
# the energy as a function of the internal coordinates of
# the molecule

set args "{theory=gamess : { basis=3-21g } zmatrix=z}"

hessian function=zopt : [ list $args ] \
    hessian=h_fcn_ts method=analytic

newopt function=zopt : [ list $args ] \
    method=baker \
    input_hessian=h_fcn_ts \
    follow_mode=1
```



■ Design Features

- ❑ Generic - can integrate QM, MM, QM/MM trajectories
- ❑ Based on DL_POLY routines
 - Integration by Verlet leap-frog
 - SHAKE constraints
 - Quaternion rigid body motion
 - NVT, NPT, NVE integration
- ❑ Script-based control of primitive steps
 - Simulation Protocols
 - equilibration
 - simulated annealing
 - Tcl access to ChemShell matrix and coordinate objects
 - e.g. force modification for harmonic restraint
 - Data output
 - trajectory output, restart files



- Object oriented syntax follows Tk etc
 - `dynamics dyn1 coords=c ... etc`
- Arguments
 - `theory=` module used to compute energy and forces
 - `coords=` initial configuration of the system
 - `timestep=` integration timestep (ps) [0.0005]
 - `temperature=` simulation temperature (K) [293]
 - `mcstep=` Max step displacement (a.u.) for Monte Carlo [0.2]
 - `taut=` Tau(t) for Berendsen Thermostat (ps) [0.5]
 - `taup=` Tau(p) for Berendsen Barostat (ps) [5.0]
 - `compute_pressure=` Whether to compute pressure and virial (for NVT simulation)
 - `verbose=` Provide additional output
 - `energy_unit=` Unit for output



■ Arguments (cont.)

- ❑ rigid_groups= rigid group (quaternion definitions)
- ❑ constraints= interatomic distances for SHAKE
- ❑ ensemble= Choice of ensemble [NVE]
- ❑ frozen= List of frozen atoms
- ❑ trajectory_type= Additional fields for trajectory (> 0 for velocity, > 1 for forces)
- ❑ trajectory_file= dynamics.trj file for trajectory output

■ Methods:

- ❑ Dyn1 configure temperature=300
 - configure - modify simulation parameters
 - initvel - initialise random velocities
 - forces - evaluate molecular forces
 - step - Take MD step



- update
- mctest
- output
- printe
- get
- trajectory file
- destroy
- fcap
- load
- dump
- dumpdlp
- Request MM or QM/MM pairlist update
- Test step (Monte Carlo only)
- print data (debugging use only)
- print step number, kinetic, potential, total energy, temperature, pressure volume and virial.
- Return a variable from the dynamics, temperature, input_temperature, pressure, input_pressure, total_energy, kinetic_energy, potential_energy time
- Output the current configuration to the trajectory
- free memory and destroy object
- force cap
- recover positions/velocities
- save positions/velocities
- write REVCON



```
dynamics dyn1 coords=c theory=mndo \  
    temperature=300 timestep=0.005  
dyn1 initvel  
set nstep 0  
while {$nstep < 10000 } {  
    dyn1 force  
    dyn1 step  
    .....  
    # additional Tcl commands here  
    incr nstep  
}  
dyn1 configure temperature=300  
# etc  
dyn1 destroy
```



- If you access an object from a disk, ChemShell will always update the disk copy when it has finished (there is no easy way of telling if a command or procedure has changed it).
- Usually this is harmless (e.g. output formats are precise enough), but unrecognised data in the input will not be present in the output, take a copy if you need to keep it.
- e.g. if a GAMESS-UK punchfile contains a fragment object and a single data field (e.g. the potential) you can use it as both a fragment object and a field object

```
% rungamess test  
% cp test.pun my_structure  
% cp test.pun my_field  
% chemsh  
.....
```



- During a run objects can be cached in memory, the command to request this is the name of the object (similar to a declaration in a compiled language)

```
#  
fragment c  
c_create coords=c {  
h 0 0 0  
h 0 0 1  
}  
list_molecule coords=c  
delete_object c  
# No file is created here
```

- **Confusion of objects with files can lead to unexpected behaviour!!**



- ChemShell can be compiled with message passing libraries
 - MPI
 - TCGMSG/Global Array

- Interpreter runs only on node 0
 - All data objects are read/written from node 0

- Compiled-in energy/gradient evaluations are in parallel
 - DL_POLY
 - GULP
 - GAMESS-UK (if compiled in)

- It is also possible to couple serial ChemShell to parallel QM codes (e.g. Turbomole)



- On HPCx there are two pre-compiled parallel binaries
 - MPI
 - Global Array

`/usr/local/packages/chemsh/ChemShell-3.0_{ga,mpi}`

- The differences relate to the different parallel implementations of GAMESS-UK which are built into them
 - Global version Array has a wider range of parallel functionality
 - MPI has a more scalable SCF driver (newscf) which is better for large problems on large node counts.



- Running on HPCx
 - ❑ Parallel batch system running LoadLeveler Queueing system
 - ❑ Course accounts course01 to course15 are available
 - ❑ We have been assigned 1 16 processor node.
 - ❑ Sample job scripts for use with the course accounts are provided for GAMESS-UK and ChemShell
 - ❑ Run with lsubmit
 - % lsubmit test.script
 - ❑ Check queues with llq, Cancel jobs with llcancel
 - ❑ Interactive runs will execute on the head node (small jobs only please!).

- For more information see the HPCx user guide at <http://www.hpcx.ac.uk/support/introduction/>