Obtaining the Socorro code

1. Follow the links from the home page to download socorro.tgz (a gzipped tar file).

2. On a unix file system, execute "gunzip socorro.tgz" to convert socorro.tgz to socorro.tar and then execute "tar -xvf socorro.tar" to unpack this file.

3. Read /socorro/LICENSE, which explains the conditions for copying and using Socorro (GNU General Public License). If you agree with these conditions, proceed to install the software.

Installation

1. Required Software

Socorro requires both a Fortran 90/95 compiler and a C compiler. Several other software packages are also required:

- FFTW 2.1.5 (http://www.fftw.org). Newer versions won't work. We're in the process of switching the interface to support 3.0 and above. There also support for SGI FFT routines.
- BLAS library
- LAPACK library
- MPI library
- Gnu make or equivalent
- Perl

2. Obtain a make.conf file

You will need to manually copy a make.conf from the makefiles directory to the socorro root directory (here). After that edit it to your hearts content.

Most of the variables are pretty obvious, but two critical variables are not. The

first, MPLIBS (Multi-Processor Libraries), should contain all
the libraries needed for socorro including MPI. The other,
UPLIBS (UniProcessor Libraries), should NOT contain anything
related to MPI.

Once you've made a stab at configuring make.conf
you are ready to continue.

3. Run configure

Currently configure creates the files cpointer_mod.f90 and
ctof_io.h. These file take into account the endian-ness
of your machine and provide the C <-> Fortran interface
passing data. It also creates all the local Makefiles.

To skip making the C <-> F90 interface add the option

```
--no-interface
```

when invoking configure. You will need to manually create
or copy cpointer_mod.f90 and ctof_io.h in the src directory
before running configure.

Configure is run by typing:

```
./configure
```

with the optional option mentioned above.

If this works fine you can skip to step 3.

Cross-compiling or C<->F90 problems

If you are cross-compiling or mci doesn't work you will
need to manually run the "mci" program to generate the
2 files mentioned above. You can do this by first
changing to the tools/config directory and typing

```
cc -o mci make_c_interface.c
```

on the compile machine. Replace "cc" with your C
compiler and any other misc options you need.

Next run the executable "mci" on the destination

machine. The machine you are going to be running socorro on. This will create the 2 files cpointer_mod.f90 and ctof_io.h. These files should then be copied to the socorro source directory "src".

After you've done this re-run configure skipping the interface part:

```
./configure --no-interface
```

This will create the appropriate file links in the different directories.

4. Build Socorro

Just type

```
make
```

and hope for the best. This will build socorro and also the routine taginfo located in tools/taginfo.

Socorro input and output files

Socorro requires a set of input files, with the exact number depending on the type of calculation being performed, and a unix directory system. The input file names and the directories where the input files are located can be customized by editing /socorro/src/path_mod.f90. The only exception is the control file, which must be located in the run directory. The names of the output files can likewise be customized by editing /socorro/src/path_mod.f90, however their locations can only be the run directory. Descriptions of the various input and output files are given below using the default directories and names set in path_mod.f90.

Input files:

argvf: Parameters used to specify and control a run. The parameters are denoted with the tags described further below.

data/crystal: Lattice vectors, atom types, and atom positions. The format is:
<identifier_string>

<lattice constant>
<lattice vector 1>
<lattice vector 2>
<lattice vector 3>
one of ["cartesian", "lattice", or specific variations of these]
<number_of_atoms>
ATOM_TAG <position in cartesian or lattice coordinates>
ATOM_TAG <position in cartesian or lattice coordinates>
.....
(EOF)

data/NCP.ATOM_TAG: Data for the ATOM_TAG norm-conserving pseudopotential.

data/PAW.ATOM_TAG: Data for the ATOM_TAG projector-augmented-wave functions.

data/kpoints: (optional) Sampling points in the Brillouin zone. The format is:
<number of sampling points>
one of ["cartesian", "lattice", or specific variations of these]
<point in cartesian or lattice coordinates> <degeneracy>
.....
(EOF)

data/initial_velocity: (optional) Initial velocities for an MD run. The format is:
<velocity components for atom 1>
<velocity components for atom 2>
.....
(EOF)

data/lgroup: (optional) Lattice point-group operations. The format is:
<number of point operators> <number of translations per point operation>
<space>
<3x3 matrix denoting a point-group operator in lattice representation>
.....
(EOF)

data/restartf: (optional) Restart file.

data/stopf: (optional) File containing commands to halt a running program.
<stop> causes a program to halt gracefully
<abort> causes a program to halt as soon as possible

Output files:

errorf_xxx: Error and warning messages for process xxx.

diaryf: Diary of the run.

new_crystal: New crystal file after an update of the atom positions.

new velocity: New atom velocities after an MD update.

new_kpoints: (optional) Brillouin zone sampling points generated using the Monkhorst-Pack scheme.

new_lgroup: (optional) Lattice point-group operations.

new_pgroup: (optional) Point-group operations of the space group.

new sgroup: (optional) Space-group operations.

new_restartf: (optional) Restart file.

md_trajectory: Atom positions during an MD run.

Running test calculations

To run the test in /socorro/testdata/si_ncp:

```
# cd run
# .linktest si_ncp
# ./socorro
```

To run the test in /socorro/testdata/paw_si:

```
# cd run
# ./linktest paw_si
# ./socorro
```

Control tags

tag formats

The lines below explain tags and their associated values which can be used to control a socorro calculation. To

use a tag, simply give it on a separate line in argvf followed by the tag value or values. For example, to specify
a wavefunction cutoff of 20 Ryd add the following line to argvf:

```
wf_cutoff  20.0d0
```

Note that: Character parameters are generally accepted with three levels of capitalization, for example, NONE, None, none.
Logical parameters can be capitalized or not.
Energies should be given in Rydberg units.
Distances should be given in atomic units (Bohr radii).

Tag descriptions are grouped according to code areas, and the format for the descriptions is as follows:

tag: Meaning
type; valid values or range of numerical values; default value
modules where tag is sought; object where value is stored; name of variable holding the value
first option if tag is not found, second option if tag is not found, ...
Note #1
Note #2
.
.
.

Wavefunction related tags:

wf_cutoff: Determines the number of plane waves in the wavefunction expansion.
real number; >= 0.0; no default
multibasis_mod, layout_mod (optional); protobasis_obj; pb%cutoff
run aborts
Expansion includes plane waves for which (G + k)^2 < wf_cutoff, where G is a wave vector and k is a sampling point

nbands: Number of bands (wavefunctions at a Brillouin zone sampling point).
integer; >= total charge/2; no default
multibasis_mod; protobasis_obj; pb%nbands
run aborts

kpoints: Method of obtaining sampling points (k-points) in the Brillouin zone.
character; GMP, GBP, USP; GMP
kpoints_mod; kpoints_obj; kp%o%mode

default is used

For GMP (Generate Monkhorst Pack), a mesh is generated from mpparams (see below). These points are closed using the

lattice group and then reduced using the point group of the space group (augmented with inversion if it is not

already present in the space group).

For GBP (Generate Baldereschi Point), the point at (0.25,0.25,0.25) is used for sampling. This mode is meant to be

used only in molecular dynamics calculations and only with a cubic cell and the C_1 space group.

For USP (Read Special Points), sampling points are read from file /data/kpoints. The format is

```
----------------------------------------
n
rep
k1(1) k1(2) k1(3) d(1)
k2(1) k2(2) k2(3) d(2)
.
.
.
kn(1) kn(2) kn(3) d(n)
----------------------------------------
```

where n is the number of sampling points, rep is the representation (lattice or cartesian), ki(j) is the j'th coordinate of the i'th sampling point, and d(i) is the degeneracy. The points are assumed to consistent with the lattice group and space group.

mpparams: Numbers of Brillouin zone sampling points along the three reciprocal lattice vectors.

3 integers; -inf < i < +inf; no defaults

kpoints_mod; kpoints_obj; kp%o%mpp

run aborts

Basically Monkhorst-Pack parameters with shifts of the mesh accomplished by giving one or more negative parameters. Values along two reciprocal lattice vectors related by a point group operation must be the same.

The point at zero for a reciprocal lattice vector is included if the value for that vector is < 0.

Values 0 and -1 yield the same sampling points as the value 1.

For a hexagonal lattice, sampling points fall on the line through the origin and parallel to the c-axis.

save_kpoints; .true., .false.; .false.

kpoints_mod; NA; NA

default is used

The file is saved in the run directory with the name "new_kpoints". The format is as given above with rep =

"lattice".
Saves are performed only when the kpoints_mode is GMP (the default setting).

kt: Determines band occupations.
real number; > 0.0; no default
electrons_mod; electrons_obj; el%o%kt
run aborts
Fermi function is used to determine band occupations with kt defining the energy broadening

wf_init: Determines how the Kohn-Sham functions are initialized
character; > RANDOM, DIAGNOSTIC; RANDOM
multivector_mod; NA; NA
default is used
RANDOM: Fourier coefficients are set to random numbers between -0.5 and +0.5
DIAGNOSTIC: Fourier coefficients are set to Ross Lippert's diagnostic values

Mesh related tags:

den_cutoff: Determines the number of plane waves in the fields (e.g. electron density) expansions.
real number; >= 0.0; no default
layout_mod; layout_obj; lay%o%cutoff
derived from dims, taken as 4*wf_cutoff
Expansion includes plane waves for which G^2 < den_cutoff where G is a wave vector

dims: Numbers of real-space mesh points along the three lattice vectors.
3 integers; > 0; no defaults
layout_mod; layout_obj; lay%o%dims
derived from den_cutoff (derived values are used if they are larger than dims)
Values along two lattice vectors related by a space-group operation must be the same
Values may be increased to accomodate radix set of the FFT routine

Iterative solver related tags:

max_cycles: Maximum number of iterations used to converge the electronic structure.
integer; >= 0; 40
config_mod; config_obj; cfg%o%max_cycles
default value used

cvg_mode: Measure used to determine if the electronic structure is converged.
character; NONE, ENERGY, DENSITY; DENSITY
config_mod; config_obj; cfg%o%cvg_mode
default value used
NONE causes iteration to continue up to max_cycles
ENERGY causes iteration to continue until the total energy is below a tolerance set by cfg%o%energy_tol
DENSITY causes iteration to continue until the electron density residual is below a tolerance set by cfg%o%dens_tol

energy_tol: Tolerance for determining convergence when cfg%o%cvg_mode = ENERGY.
real number; > 0.0; 1.0e-6
config_mod; config_obj; cfg%o%energy_tol
default value used
The electronic structure is converged when the total energy is below cfg%o%energy_tol

dens_tol: Tolerance for determining convergence when cfg%o%cvg_mode = DENSITY.
real number; > 0.0; 1.0e-8
config_mod; config_obj; cfg%o%dens_tol
default value used
The electronic structure is converged when the electron density residual is below cfg%o%dens_tol

Eigensolver related tags:

solver_method: Type of eigensolver to use.
character; CG, GCG, BD; CG
eigensolver_mod; eigensolver_obj; es%o%method
default is used
CG invokes a conjugate gradients solver
GCG invokes a Grassman conjugate gradients solver
BD invokes a block Davidson solver

solver_dir: Number of wavefunction updates per call to the eigensolver.

integer; > 0; 2 (es%o%method = CG), 10 (es%o%method = GCG), 10 (es%o%method = BD)
eigensolver_mod; eigensolver_obj; es%o%max_dir
default is used

solver_tol: Tolerance used to determine convergence of the eigenfunctions.
real number; > 0.0; 1.0e-4 (es%o%method = CG), 1.0e-5 (es%o%method = GCG), 1.0e-4 (es%o%method = BD)
eigensolver_mod; eigensolver_obj; es%o%res_tol
default is used
Tolerance refers to the largest wavefunction residual

remap_type: Type of routine used to remap wavefunctions.
character; MPI, CUSTOM; CUSTOM
multibasis_mod; multibasis_obj; mb%o%remap_type
default is used
MPI invokes the MPI_ALLTOALLV routine
CUSTOM invokes the C remap_2d routine written by Steve Plimpton

Projector related tags:

projector_type: Type of non-local projectors to use.
character; RECIPROCAL, REAL; RECIPROCAL
hamiltonian_mod; h_common_obj; hc%o%projector_type
default value used
RECIPROCAL invokes reciprocal-spaced projectors
REAL invokes real-space projectors constructed using the scheme proposed by King-Smith et al.

projector_radius_xxx: Radius used to optimize real-space projectors for atom type xxx.
real number; < radius of sphere which will fit inside the supercell; 4.0
ncp_data_mod; ncp_data_obj; pd%o%r_opt
run aborts

optimization_points: Number of points used to optimize the real-space projectors.
integer; > 0; 171
ncp_data_mod; NA; NA
default is used
RESET THIS VALUE ONLY IF YOU UNDERSTAND THE OPTIMIZATION ROUTINE!

Mixer related tags:

mix_field: Field to be mixed.
character; DENSITY, POTENTIAL; DENSITY
fields_mod; fields_obj; f%o%mix_field
default is used

mix_type: Type of mixing scheme to use.
character; NONE, SIMPLE, PULAY, ANDERSON; PULAY
mixer_mod; mixer_obj; mx%o%mix_type
default is used

mix_weight_pf: Global proportion of the new field to mix with the old field.
real number; 0 < mix_weight_pf <= 1; 0.8 (SIMPLE), 0.8 (PULAY), 0.8 (ANDERSON)
mixer_mod; mixer_obj; mx%o%weight_pf
default is used
Used only with mix_type = SIMPLE, PULAY, or ANDERSON.

mix_weight: Determines the distribution of mixing weights for different wave vector coefficients.
character; CONSTANT, KERKER; CONSTANT
mixer_mod; mixer_obj; mx%o%weight_type
default is used
Used only with mix_type = SIMPLE, PULAY, or ANDERSON.
For mix_weight = CONSTANT, weight is mix_weight_pf independent of wave vector.
For mix_weight = KERKER, weight depends on wave vector according to mix_weight_q.

mix_weight_q: Wave-vector magnitude at which mixing weight makes a transition from low to high.
real number; >= 0.0; 0.8 (SIMPLE), 0.8 (PULAY), 0.8 (ANDERSON)
mixer_mod; mixer_obj; mx%o%weight_q
default is used
Used only with mix_weight = KERKER.
For wave vector G, weight is mix_weight_pf/(1.0 + q^2/G^2)

mix_metric: Weighting for the mixing residuals.
character; UNITY, KERKER; UNITY
mixer_mod; mixer_obj; mx%o%metric_type
default is used
Used only with mix_type = PULAY or ANDERSON
For mix_metric = UNITY, weight is 1.0 independent of wave vector.
For mix_weight = KERKER, weight depends on wave vector according to mix_metric_q.

mix_metric_q: Wave-vector magnitude at which metric weight makes a transition from high to low.
real number; >= 0.0; 0.8 (PULAY), 0.8 (ANDERSON)
mixer_mod; mixer_obj; mx%o%weight_q
default is used
Used only with mix_metric = KERKER.
For wave vector G, weight is (1.0 + q^2/G^2)

mix_saves: Maximum number of mixing residuals to save.
integer; 1 <= mix_saves <= 20; 5
mixer_mod; mixer_obj; mx%o%max_saves
default is used
Used only with mix_type = PULAY or ANDERSON

Symmetry related tags:

lattice_symmetry: Determines how the lattice group is obtained.
character; AUTO, USER; AUTO
symmetry_mod; point_group_obj; pg%o%mode (AUTO or USER)
default is used
For AUTO, the lattice group is generated from the lattice.
For USER, the lattice group is read from file lgroup with format given below.
The lattice group is used to close the Monkhorst-Pack k-point mesh.

symmetry: Determines whether or not space-group symmetry is used.
character; FULL, AUTO, OFF; AUTO
symmetry_mod; space_group_obj; sg%o%mode
default is used
For FULL, the space group is generated from the lattice group and atom positions. A new space group is generated
when the lattice group or atom positions change.
For AUTO, the space group is generated from the lattice group and atom positions. A new space group is generated
when the lattice group changes and when the old space group is not compatible with new atom positions.
For OFF, the trivial space group (C_1) is generated.
The space group (augmented by inversion if it is not already present) is used to reduce the closed Monkhorst-Pack points.

symmetrize_atoms: Symmetrizes the starting atom positions.
character; ON, OFF; OFF
external_mod; NA; NA

default is used

list_lattice_group: Prints the lattice-group operations to diaryf.
logical; .TRUE., .FALSE.; .FALSE.
symmetry_mod; NA; NA
default is used

list_space_group: Prints the space-group operations to diaryf.
logical; .TRUE., .FALSE.; .FALSE.
symmetry_mod; NA; NA
default is used

save_lattice_group; .true., .false.; .false.
symmetry_mod; NA; NA
default is used
The file is saved in the run directory with the name "new_lgroup". The format is as follows:
number_of_point_operators number_of_translations_per_point_operation
<space>
3x3 matrix denoting the first point operator (in lattice representation)
<space>
3x3 matrix denoting the next point operator (in lattice representation)
<space>
.
.

save_space_group; .true., .false.; .false.
symmetry_mod; NA; NA
default is used
The file is saved in the run directory with the name "new_sgroup". The format is as follows:
# of point operators # of translations per point operation
<space>
3x3 matrix denoting the first point operator (in lattice representation)
first translation for the first point operator
second translation for the first point operator
.
.
last translation for the first point operator
<space>
3x3 matrix denoting the next point operator (in lattice representation)
first translation for the next point operator
second translation for the next point operator
.
.
last translation for the next point operator
<space>

Functional related tags:

functional: Type of exchange-correlation functional to use.
character; LDA, PW91, PBE, BLYP, YLDA1, YLDA2; LDA
exc_mod; xc_obj; xc%o%method
default is used
YLDA's are LDA type functionals by Armiento and Mattsson with an alternative separation of exchange and correlation.
YLDA's should give approximately the same results as LDA. Please notify Ann E. Mattsson if not.

correlation: Which LDA correlation to use.
character; PZ, PW, VWN, LYP; depending on functional (see below)
exc_mod; xc_obj; xc%o%ctype
default is used (see below)

functional	default
LDA	PZ (PRB 23, 5048 (1981))
PW91 and PBE	PW (PRB 45, 13244 (1992))
BLYP	LYP (PRB 37, 785 (1988))

VWN (Can. J. Phys. 58, 1200 (1980)) not used as default.
Note that YLDA's don't have separate correlation.

functional_method: Method to calculate the xc potential for non-LDA.
character; WB, TRAD (see below); WB
exc_mod; xc_obj; xc%o%method
default is used
TRAD is not implemented yet.
This tag is ignored if used with LDA type functionals.

derivatives: Method to calculate derivatives of the functional.
character; NUM, ANALYT; depending on functional (see below)
exc_mod; xc_obj; xc%o%dermethod
default is used (see below)
ANALYT is only available for LDA with PZ or PW correlation where it is also default.
NUM is default for other functionals.

Atomic-representation related tags:

atomic_representation: Method for representing atoms.
character; NCP, PAW; NCP
atomic_operators_mod; atomic_operators_obj; ao%type
default is used

mix_atomic_density: Switch to invoke mixing of the PAW atomic density.
logical; .TRUE., .FALSE.; .FALSE.
atomic_operators_paw_mod; atomic_operators_paw_obj; aops%o%mix_adens
default is used

mix_atomic_density_weight: Proportion of the new adens to mix with the old adens.
real number; 0 < mix_atomic_density_weight <= 1; 0.5
atomic_operators_paw_mod; atomic_operators_paw_obj; aops%o%adens_mix_weight
default is used

Post-processing related tags:

forces: Controls whether or not the forces are computed automatically
character; ON, OFF, .TRUE., .FALSE.; OFF
config_mod; NA; NA
default is used
Note: Even when the forces tag = OFF, the forces will be computed when x_forces(cfg)
or diary_forces(cfg) is called.

pressure: Controls whether or not the pressure is computed automatically
character; ON, OFF, .TRUE., .FALSE.; OFF
config_mod; NA; NA
default is used
Note: Even when the pressure tag = OFF, the pressure will be computed when x_pressure(cfg)
or diary_pressure(cfg) is called.

stress_tensor: Controls whether or not the stress tensor is computed automatically
character; ON, OFF, .TRUE., .FALSE.; ON

config_mod; NA; NA
default is used
Note: Even when the stress_tensor tag = OFF, the stress_tensor will be computed when x_stress_tensor(cfg)
or diary_stress_tensor(cfg) is called.

Restart related tags:

restart: Determines whether or not to restart and how much information to read
character; OFF, F, FE; OFF
config_mod; NA; NA
default is used
OFF: restart information will not be read
F: fields information (atomic and grid densities) will be read
FE: fields and electrons (Kohn-Sham functions) information will be read
EFE: fields, electrons, and external (crystal) information will be read (not currently implemented)

write_restart: Determines whether or not to write a restart file and how much information to write
character; OFF, F, FE, EFE; OFF
config_mod; NA; NA
default is used
OFF: restart information will not be written
F: fields information (atomic and grid densities) will be written
FE: fields and electrons (Kohn-Sham functions) information will be written
EFE: fields, electrons, and external (crystal) information will be written (not currently implemented)
Note: The restart file is written at the end of the cfg constructor. To write a restart file after a
cfg update (from the socorro level), call the public write_restart(cfg,tag) routine with tag set
to F, FE, or EFE.

Structural Optimization related tags:

relax_method
character; NONE, STEEPEST_DESCENTS, SD, CONJUGATE_GRADIENT, CG, QUENCHED_MINIMIZATION, QM; NONE

relax_force_tol
real number; ; 1.d-3

relax_max_steps
integer; 0 < relax_max_steps ; 100

relax_prefactor
real number; 0 < relax_prefactor; 1.0

relax_time_step
real number; 0 < relax_time_step; 1.0

ENERGY MINIMIZATION:

Fixed volume optimization of the atomic positions. All atoms are moved - there is currently no method for constraining specific atoms. This is called by the statement "if (optimize_lattice(cfg)) call diary(cfg)" in socorro.f90.
optimize_lattice returns a logical that indicates whether any modifications to cfg have occurred. The cfg that is returned corresponds to the optimized positions.

The relevant options in argvf are as follows

relax_method: Specify the relaxation method to be used.

NONE default Do not perform a structural optimization

STEEPEST_DESCENTS, SD 'steepest descents' (I have seen different definitions of 'steepest descents'.) What is implemented here is the simple approach of changing the coordinates by F*relax_prefactor where F is the current force and relax_prefactor is a parameter that you can set (see below). This is usually NOT the best way to get to a minimum. It was included because it was the obvious first thing to code and is a standard 'brute force' approach to optimization.

CONJUGATE_GRADIENT, CG 'conjugate gradients'
Implements a conjugate gradient search for the minimum energy structure. (See Press, et.al, 'Numerical Recipes' for a description of this algorithm.) This is typically the best way to go when the initial positions are close to the minimum.

QUENCHED_MINIMIZATION, QM 'quenched MD'
This implements an algorithm described in Della Valle and Andersen, J. Chem. Phys. 97, 2682 (1992). It performs a molecular dynamics simulation using the 'velocity Verlet' algorithm with the following modifications. At each time step and for each particle, the velocity

is reset as follows. If the projection of the force along the velocity is positive, the velocity is replaced by the projection of the velocity along the direction of the force. If the projection is negative, the velocity is set to zero. This will quench the dynamics to the minimum. This approach seems to be best suited for getting close to the minimum when the initial guess may be poor.

relax_force_tol: This is the stopping criteria for all methods. The code stops when the root mean square value of the force components is less than this value. default: 1.d-3.

relax_steps: This sets a maximum number of force calls that can be made to attempt to find the minimum. Note that in some cases, the actual number of calls may exceed this slightly since it will always try to finish the line minimizations in the conjugate gradient approach. default: 100

relax_prefactor: This is the constant used in relax_method=SD (see above) Note that the efficiency and stability of this method depends on this choice. If the value is too large, the optimization will become unstable. If it is too small, a large number of force calls is required to get to the minimum. default: 1.0

relax_time_step: This is used in relax_method=QM. It sets the fixed time step for the MD simulation. default: 1.0 (probably too small for most cases.)

atom_mass_xxx This is the mass of the atom in amu used for quench_minimization. Here xxx is replaced by the tag used in the crystal file. There should be one line like this for each type. The default is to assume a mass of 1 amu.

Molecular Dynamics related tags:

md_method
character; NONE, NVE, NVT_RESCALE, NVT_ANDERSON, NVT_HOOVER; NONE

md_time_step
real number; 0 < md_time_step; 100.

md_steps
integer; 0 <= md_steps; 0

md_skip_steps
integer; 0 <= md_skip_steps <= md_steps; 0

md_init_temp
real number; 0 <= md_init_temp; 0.

md_desired_temp
real number; 0 <= md_desired_temp; md_init_temp

md_temp_freq
integer; 0 < md_temp_freq; 1

md_hoover_mass
real number; 0 < md_hoover_mass; 1000.

md_gen_velocities
character; YES, NO; YES

The code can currently perform either a NVE (constant number, volume and energy) micro-canonical simulation or a NVT (constant number, volume and temperature) simulation. For the later case, a variety of standard thermostat methods are implemented. The initial velocities can be either read from the file "data/initial_velocity" or are set randomly based on an input temperature. The integration is performed via the 'velocity Verlet' algorithm (see any book on MD simulations) with a fixed time step. Intermediate atomic positions are output to the file 'md_output' at every time step. The masses are set with the arg parameter atom_mass_xxx (see below).

This is called by the statement "if (run_moldyn(cfg)) call diary(cfg)" in socorro.f90. run_moldyn returns a logical that indicates whether any modifications to cfg have occurred. If an MD simulation is performed, the cfg returned is that for the final time step.

The relevant parameters are

md_method: Specify the molecular dynamics method to be used.

NONE (default) - do no MD.

NVE perform a NVE simulation using a fixed time-step velocity Verlet algorithm.

NVT_RESCALE perform a NVT (isochoric,isothermal) simulation using a

fixed time-step. The temperature control is through periodic rescaling of the velocities to achieve the desired temperature.

NVT_ANDERSON perform a NVT (isochoric, isothermal) simulation using a fixed time-step. The temperature is controlled via a stochastic method due to Anderson (see H C Anderson, J. Chem. Phys. 72, 2384 (1980)). At each time step and for each atom, a random velocity from a Maxwell-Boltzman distribution replaces the velocity with a probability give by 1/temp_freq.

NVT_HOOVER perform a NVT (isochoric, isothermal) simulation using a fixed time-step. The temperature is controlled using the Hoover implementation of the NosÈ thermostat. (See, for example, "Understanding Molecular Simulations" by Frenkel and Smit). The rate of energy flow between the ions and the heat bath is controlled by md_hoover_mass.

md_time_step Time step used for the MD. See note below regarding units. default: 100.

md_steps Number of time steps to integrate the equation of motion. default: 0

md_skip_steps Number of time steps to ignore before starting to compute averages - in other words (skip_steps)*(time_step) is an equilibration time. default: 0

md_init_temp Temperature used to define the initial velocity distributions. The velocities are selected from a Maxwell-Boltzman distribution. Then they are adjusted to give zero total momentum and rescaled to give the exact temperature requested. default: 0

md_desired_temp Target temperature for the isothermal simulation methods. default: md_init_temp

md_temp_freq Parameter that determines the frequency of velocity modifications for the isothermal simulation methods. For NVT_RESCALE, the velocity is rescaled every md_temp_freq time steps. For NVT_ANDERSON, it give the inverse of the probability that an atom will get a random velocity in a given time step.

atom_mass_xxx This is the mass of the atom in amu. Here xxx is replaced by the tag used in the crystal file. There should be one line like this for each type. The default is to assume a mass of 1 amu.

md_hoover_mass Used by NVT_HOOVER. It is the effective mass associated with the additional coordinate added to control the temperature. It may need to be adjusted by trial and error. default: 1000.

md_gen_velocities Specify the method for initializing velocities

YES Determine the initial velocities based on md_init_temp. (default)

NO Read the initial velocities from the file 'initial_velocity' in the run directory. This file contains a line with the x, y, and z velocity for each atom on a separate line. The order of the atoms is assumed to be the same as in the crystal file.

Transition State Finding related tags:

ts_method
character; 0, 1; 0

dimer_separation
real number; 0 < dimer_separation; 0.1

dimer_force_tol
real number; 0 < dimer_force_tol; 0.001

dimer_max_steps
integer; 0 < dimer_max_steps; 100

Currently, there is only one transition state finding method implemented, the 'dimer method'. This is described in detail in Henkelman and JÛnsson, J. Chem. Phys. 111, 7010 (1999). This method attempts to find saddle points with one unstable mode (ie a single negative value of the Hessian matrix.) The trick is to avoid computing the second derivatives of the energy. To do this, two configurations that differ by a fixed distance are considered. (This is the 'dimer'.) The dimer is alternately rotated such that the separation is along the direction of minimum curvature and then translated in the direction of the saddle point. The current implementation does not incorporate all of the sophisticated algorithms for optimization discussed in the paper. Also, there is currently no way to initialize the first orientation of the dimer other than choosing a direction at random. (Waiting on the new I/O routines before implementing this.) This takes a large number of forces calls, especially to get the

first estimate of the dimer orientation.

This is called by the statement
"if (transition_state(cfg)) call diary(cfg)" in socorro.f90.
transition_state returns a logical that indicates whether any
modifications to cfg have occurred. On return after a transition state
search, cfg corresponds to the transition state.

The relevant parameters are the following

ts_method Specify the transition state method

0 (default) do nothing

1 implement the dimer method

dimer_separation The magnitude of the real space separation between the
two configurations of the dimer. The algorithms are based on the
assumption that this is small. Of course, if it is too small,
numerical errors can become unacceptable because many of the
calculations are based on differences between the calculations for
each of the two configurations. default: 0.10

dimer_force_tol Stopping criteria which is based on the rms forces
on the 'dimer'. Ideally, this will correspond to the net forces
at the saddle point. default: 1.d-3

dimer_max_steps Maximum number of calculations of the dimer properties
before the algorithm stops. Note that each calculation of a dimer
property requires 2 electronic structure calculations. In some cases,
it may make somewhat more calculations in order to stop at a sensible
place. default: 100

NOTE ON UNITS:

The convention in the code is that energies are in Rydbergs and that
distances are in Bohrs. I have made the choice to have the code work
with the nuclear masses in units of the electron mass - keep in the spirit
of atomic units. For convenience, when masses are entered, they are assumed
to be in amu (atomic mass units) and are converted in the code
to electron masses. Having made this choice for the mass unit, the choice
of the time unit is now fixed. The unit of time is 3.421E-17 sec. Since
a typical MD time step is on the order of a few femtoseconds (fs),
the typical time steps will be on the order of 100 in the units used here.

There is a subtlety in the determination of the temperature. In classical thermodynamics (MD is classical in the treatment of the ionic motion), each degree of freedom has a kinetic energy of kT/2. The issue is related to the number of degrees of freedom. If the MD method used conserves the total momentum, then the number of degrees of freedom is 3(N-1). If the total momentum is not conserved, the number of degrees of freedom is 3N. The NVE and NVT_RESCALE methods conserve the total momentum. (Actually, for NVT_RESCALE the total momentum remains zero if it starts out zero. The initial velocities are generated in the code to have zero total momentum.) For these methods, the code uses 3(N-1) degrees of freedom to determine the temperature. For NVT_ANDERSON, the total momentum is not conserved, so the code uses 3N degrees of freedom to compute the temperature.