Monte Carlo program: runmc

Main program in called *runmc*.
The main fortran-code file which corresponds to this program is *mc_main.f90*

The program uses the input files which describe the system to produce the trajectories of the particles.

How to compile

In command line run

```
  make runmc
```

How to use

Command line arguments:

```
      runmc  parameters.prm system.composition  input.moltab [ extra_parameters ]
```

parameters.prm - file, which include main parameters of the simulation
system.composition - file, which describes the numbers of molecules of each kind in the system
input.moltab - initial configuration of the system
extra_parameters - coma separated list of pair parameter1=value1,parameter2=value2

Input files:

parameters.prm

Each line of this file is:
- empty
- comment (starting with #)
- pair parameter=value

The values can be either numbers or strings (without brackets, spaces). Each parameter has its own type.
The value-pair can then be followed by the comment

Example parameters file:

```
sr = 4      # rmax = sr / alpha
            # err_r =  Q (sr/ alpha L^3)^(1/2) exp(-sr^2) / sr^2
            #  typical value sr = 3..4
sk = 4      # kmax = s(L*alpha) / pi
           # err_k = Q (sk/2 alpha L^3)^(1/2) exp(-sk^2) / sk^2
           # typical value sk = 3..4
alphaL = 0    # alpha * L  , by default (given alphaL=0) :=> alphaL= 2*sr
           # rmax =  sr / alpha < L/2 ==>  alpha*L > 2*sr
           # alpha*L = alpha [L^-1] = 2*sr corresponds to the cutoff L/2 (default, if alphaL=0)

temp =298.15                   # temperature, K
external_permutivity = 50       # epsilon_ext

BoxLength = 0                 # if box_length = 0 --> to be recalculated from density

density = 30                   # particles/nm^3
dr_a =  0.4                  # max. dispacement in Angstroems
d_angle_degree = 40            # max rotation  in degree
xlambda_c =0.5                # coupling constant for torque ( note: normally xlambda_f = xlambda_c AND xla
mbda_f = 0.5 )
xlambda_f = 0.5               # coupling constant for  force

# simulation

pressure = 100000  #  in Pascals, i.e. J/m^3, 1Bar = 100000 Pa, 1.atm = 101325 Pa
pressure_step_multiplier = 5  # make NPT step at average each nmol * pressure_step_multiplier steps
                          # set 0 for NVT simulation
max_volume_scaling = 1.05   #  vnew = vold * max_volume_scaling ** lambda, -1 < lambda < 1
max_cycle = 10000         # number of cycles to do. each cycle includes nmol moves

n_store_traj_interval = 100   # save trajectory frames each nmol * n_store_traj_interval steps
                         # ( each n_store_traj_interval_cycles)
n_store_energy_interval = 100
   # interval to

# format of input moltab file

input_nbytes_xyz = 2           # number of bytes per coordinate sample
input_nbytes_ang = 2           # number of bytes per angular sample

# format of the output moltab file

output_nbytes_xyz = 2           # number of bytes per coordinate sample
output_nbytes_ang = 2           # number of bytes per angular sample

# output files

traj_file = traj.moltraj  # trajectory file
energy_file = energy.dat  # file with energies
boxlength_file = boxlength.dat    # file with lengths of the boxes ( stored at each boxlength change)
frames_file = frames.dat    # data about the boxlength (each n_store_traj_interval) + acceptance rates
```

```
freq_file = frequences.dat             # frequences file, contains the information about the probability to peack each molecule for movement or exchange
```

system.composition

This file contains the infomation about the numbers of molecules of each kind in the system.
The format is:
The first line - one number : total number of different kinds of molecules in the system
following lines : molecule_file n_molecules

Example:

```
3
SPCE.mol  100
Na_Dang.mol 5
Cl_Dang.mol 5
```

This file describes the system of 100 SPCE water molecules and 5 NaCl ion pairs

molecule file (.mol)

Contains the infomation about the molecule structure and force field

One line of file describes one atom.
The format is:
atom_name x y z sigma epsilon charge mass hard_core_diameter

Coordinates, sigma and hard_core_diameter are in angstroems, espilon in kcal/mol, mass is only used to determine the center of mass of the molecule (it should not be necessarily the "thue" atomic mass)

Example (file SPCE.mol):

```
O    0.000000000000000   0.000000000000000   0.0000000000  3.1655  0.1554  -0.84760  16.0  2.0
H    1.000000000000000   0.000000000000000   0.0000000000  0.000  0.00   0.42380   0.0  0
H   -0.325568154457156   0.945518575599317   0.0000000000  0.000  0.00   0.42380   0.0  0
```

input.moltab

The file contains the information about the coordinates of centers of masses of the molecules and their orientations.

The file is binary. It consists of the 6 number records, which correspond to the coordinates and Euler angles:
x y z theta phi psi
each value is represented in fixed-point format, by 2 byte unsigned integer number in little-endian encoding.

For the coordinates:
 the value of 0 corresponds to the relative coordinate -BoxLength/2
 the maximum value (65535) corresponds to +BoxLength/2

For the angles:
 0 corresponds to 0
 the maximum value (65535) corresponds to 2pi

Frequences file

The file is optional, given by the parameter freq_file, (often called frequences.dat)

Determines how often will certiain molecules be moved/exchanged

The format:
First (non-comment) line - number of intervals
Next line: from-to move_rate xchange_rate
Example:

```
3
1-100    10  0
101-105 100  100
106-110 100  100
```

This means that the molecules with the numbers 1-100 will be picked up for moving with the relative probabi lity 10, and for exchange - never (0)

For the molecules from intervals 101-105 and 106-110 the relative probabilities are 100 and 100.

The total probability to pick the given molecule is calculated as the relative probability for the molecule divi ded by the total relative probability of all molecules. The probabilities for each action (move or exchange) ar e calculated separately.

For example, in our case: we have 100 water molecule with move probability 10, and 10 ions with move pro bability 100. This gives 100*10 + 10 * 100 = 2000 (the total relative probability)

So, the probability to move the given water molecule is 10/2000 = 1/200

the probability to move some water molecule is 100/200 = 1/2

The probability to move the given ion is 100/2000 = 1/20

the probability to move some ion is 10/20 = 1/2

For the exchange we have: total relative probability: 0*100 + 100*10 = 1000

the probability to pick the water for exchange is zero.

the probability to pick the given ion for exchange is 100/1000 = 1/10

Output files:

trajectory

The name of the file is given in parameters.prm in traj_file.

Ususally the name is traj.moltraj

The format binary, sequential frames.

Each frame contains the coordinates and orientations of the molecules. The format is the same as in input.mol tab

frames

The file contains the information about the frames.

It is a text file.

Each line

framecount   BoxLength   n_mv_steps n_mv_accepted   n_vol_steps n_vol_accepted n_xchg_steps n_xchg _accepted   current_time

where
frame_count - the current number of frame
BoxLength - size of the box (in Agnstroems)
n_mv_steps - number of tries to move
n_mv_steps - number of moves accepted
n_vol_steps - number of tries to change the volume
n_vol_accepted - number of accepted changes of the volume
n_xchg_steps - number fo tries to exchange the molecules
n_xchg_accepted   - number of accepted exchanges
current_time   - time in seconds since 1 Jan 1970 (unix format time)

energy   (energy_file in parameters)

The file contains the information about the energy components.

It is a text file.

Each line
framecount total_energy total_electrostatics total_lj real_electrostatics real_lj6 real_lj12 kspace_electrostatics kspace_lj

boxlength (boxlength_file in parameters)

Format: *framecount boxlength*

boxlength in angstroems

the length is stored each time when it has been changed (not each 100 or 1000 frames)

List of programs

clearmc.f90 : program clearmc

Deletes all the files listed in parameters.prm namely: energy_file,boxlength_file,frames_file,traj_file
Is used before re-start the simulation.
Usage:
clearmc parameters.prm

Arguments:
parameters.prm - the file with the parameters of the simulation

forces2text.f90 : program forces2text

Convert the binary frame with forces trajectory to the text format
Usage:
forces2text forces.ftab > forces.txt'

Arguments:
forces.ftab - one frame of force trajectory (NAtom * 12 bytes, Natom*3 floats) '
forces.txt - file which contains the text representation of forces components (3 columns)

genMoleculeTable.f90 : program genMoleculeTable

generate the positions of the atoms for the given composition
Usage:
genMoleculeTable system.composition output.moltab [nbytes_xyz nbytes_ang [random_seed]]

Arguments:

output.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi)
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)
random_seed - - optional argument. The seed for the random number generator

mc_accum_luc.f90 : program mc_accum_luc

calculate the projections

mc_accum_luc: do statistical processing of the simulation data'
Usage:
mc_accum_luc parameters.prm system.composition output.proj first_frame[-last_frame] [step_size] [maxmn]

system.composition - - number and types of molecules in the system
output.proj - - projections file

NOTE: quantities like pressure, compressibility etc are calculated incorrectly
NOTE: density is not updates and thus is calculated incorrectly (always 0.0332891 particles/nm^3).
to get a correct assymptote, the projections should be re-normalized to the real inverse density mean(V^-1) which can be get from frames.dat file

mc_calc_forces.f90 : program mc_calc_forces

Calculate the forces for the given trajectory and store them to file
mc_calc_forces: calculate forces for the given simulation trajectorory
Usage:
mc_calc_forces parameters.prm system.composition forces.ftraj [extra_parameters]

Arguments:
system.composition - - number and types of molecules in the system'
extra_parameters - - coma separated prm1=val1,prm2=val2,... NO SPACES ALLOWED! '
forces.ftraj - - forces trajectory file. In float (24+8) format. See FloatingPoint.f90 for details

mccont.f90 : program mccont

Continue the MC simulation
Usage:
mccont parameters.prm system.composition [extra_parameters]

Arguments:
system.composition - - number and types of molecules in the system
extra_parameters - - coma separated prm1=val1,prm2=val2,... NO SPACES ALLOWED!
first frame is the last frame from the traj_file (given in parameters)'
first boxlength is taken from the last frame in frames_file

mc_main.f90 : program runmc_main

Main Program: runmc
The program uses the input files which describe the system to produce the trajectories of the particles.

Usage:
runmc parameters.prm system.composition input.moltab [extra_parameters]

Arguments:
parameters.prm - - file, which include main parameters of the simulation
system.composition - - file, which describes the numbers of molecules of each kind in the system
input.moltab - - initial configuration of the system
extra_parameters - - coma separated list of pair parameter1=value1,parameter2=value2

mc_make_round_holes.f90 : program mc_make_round_holes

create spherical holes in the given box with molecules
Usage:
mc_make_round_holes system.composition input.moltab BoxLength holes.txt output.moltab [nbytes_xyz nbytes_ang]

Arguments:
input.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi)
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3
holes.txt - - first line - number of holes, next lines: x y z R
output.moltab - - binary file where the molecules which intersect with holes
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)

mc_mean_force.f90 : program mc_mean_force

calculate the mean forces for specific molecules
mc_mean_force: calculate the mean force projection between the molecules'

Usage:
mc_mean_force parameters.prm system.composition forces.ftraj rangeA rangeB maxR dr output.dat

Arguments:
system.composition - - number and types of molecules in the system'
extra_parameters - - coma separated prm1=val1,prm2=val2,... NO SPACES ALLOWED! '
forces.ftraj - - forces trajectory file. In float (24+8) format. See FloatingPoint.f90 for details'
rangeA,rangeB - - ranges for the 1st and 2nd molecules'
in format num1[-num2]'
the MeanForce will be calculated for all pairs A-B where A is in range A, B is in rangeB'
maxR, - dr - samples for MeanForce will be [0:dr:maxR]'
output.dat - - four columns: r sum(f12) N(r) sum(f12)/N(r) where N(r) is number of AB pairs at distance r'

mcrdf.f90 : program mcrdf

calculate the Radial distribution functions between the atoms
Usage:
mcrdf parameters.prm composition output_prefix [dr Rmax [mol_labels [nskip[-maxfram]]]] '

-
parameters.prm - - parameters of the simulation. in format prm = val at each line '
they should have AT LEAST such fields: '
frames_file = ...'
traj_file = ... '
output_nbytes_xyz = ... '
output_nbytes_arg = ... '
composition - - number and types of molecules in the system'
traj - - trajectory binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
frames.dat - - information about the boxlength at each frame in traj '
output_prefix - - prefix for output files'
output_files are: output_prefixN1_N2.dat, where N1_N2 are numbers or labels of species'
ouput files are text three-coulomn files. '
The columns are : r count(r) g(r) cnt2(r) g2(r) cnt3(r) g3(r) cnt4(r)'
where:'
count(r) = number of particles found in [r;r+dr] '
g(r) = count(r) / N_total * Vmax / dV '
(N_total - total number of distances counted)'
'
cnt2(r) = sum_frame sum_ij 1/r_ij^2 where r_ij in [r;r+dr] '
g2(r) = 1/(4pi rho^2 dr N_frames) 1/<V> cnt2(r) '
'
cnt3(r) = sum_frame 1/V_frame sum_ij 1/r_ij '
g3(r) = 1/(4pi rho^2 dr N_framess) cnt3(r)'
NOTE !!! ONLY cnt4 (8th column) corresponds to the usual way of collecting the data --> only 8th column should be used for any comparisons '
'
dr,Rmax - - bin size and size for Rdf in angstroems. Default dr=0.1 Rmax=12. '
Note - : Nomrally Rmax should be less than min(BoxLength)/2 '
mol_labels - - optional coma separated labels used to produce the output files. If no labels given, numbers are used'
nskip - - number of frames to skip before the start of counting g(r) '
maxfram - - the number of frame to stop the accumulation

mc_rmsd.f90 : program mc_rmsd

calculate the displacement from the original position for a given set of molecules
Usage:
mc_rmsd system.composition traj.moltraj frames.dat interval > output.dat'

Arguments:
interval - - in format num1-num2, where num1 and num2 are the numbers of the first and the last molecules for rmsd '
output.dat - - text file with columns (one per molecule), displacement from the initial position'

moltab2xyz.f90 : program moltab2xyz

convert the binary file with molecular positions to the xyz format
Usage:
moltab2xyz system.composition input.moltab output.xyz [BoxLength [nbytes_xyz nbytes_ang]]'

Arguments:
input.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3 '

moltab_bin2text.f90 : program moltab_bin2text

Convert the binary molecular coordinates to the text form
Usage:
moltab_bin2text system.composition input.moltab BoxLength output.moltext [nbytes_xyz nbytes_ang] '

Arguments:
input.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3 '
output.moltext - - text file where of the same format (x,y,z,theta,phi,psi)
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '

moltab_text2bin.f90 : program moltab_text2bin

Convert the text file x y z theta phi psi to the binary moltab format
Usage:
moltab_text2bin system.composition input.moltext BoxLength output.moltab [nbytes_xyz nbytes_ang] '

Arguments:
input.moltext - - in text format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3 '
output.molbin - - bin file where of the same format'
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '

List of the files

AtomicData.f90 : Module AtomicData .
The module contains the data structures and the functions to manipulate the coordinated of the atoms

BiasedRandom.f90 : Module BiasedRandom .
BiasedRandom - generate the random numbers 1..N , where the probability of i is proportional to prob(i)

clearmc.f90 : program clearmc .
the program is used to clear the files created during the simulation, such as parameter.prm, trajectory and

frames files.

composition.f90 : Module composition .
the module contains the data structure and the functions to deal with the system composition (numbers of the molecules of each type)

constants.f90 : Module Constants .
The module contains the constants used in the program

error.f90 : Module Error .
The module contains the functions and data structures to deal with run-time errors

EwaldSumExternal.f90 : Module EwaldSumExternal .
The module contains the data structures and the functions to deal with the external permutivity component of the ewald sums, i.e. 2 pi / (2 external_permutivity + 1) (mu_x^2 + mu_y^2 + mu_z^2) where mu = SUM_j q_j r_j

EwaldSumKSpace.f90 : Module EwaldSumKSpace .
This Module contains structures and functions which deal with the K-Space component of the Ewald Sum (both LJ and Coulomb)

EwaldSumRealSpace.f90 : Module EwaldSumRealSpace .
Data structures and functions which deal with the real component of the Ewald sum

EwaldSumTails.f90 : Module EwaldSumTails .
This module contains the functions to compute the coordinate-independent parts of the Ewald sum

FloatingPoint.f90 : Module FloatingPoint .
Module to work with the floating point numbers (construct from mantisse and exponent, store to and restore from file)

ForceKSpace.f90 : Module ForceKSpace .
Module to calculate forces originating from KSpace Ewald sum

forces2text.f90 : program forces2text .
Convert the binary frame with forces trajectory to the text format

FourierGrid.f90 : Module FourierGrid .
Grid in fourier space

Functions.f90 : Module Functions .
Functions used in calculation of Ewald sums Written by Luc Belloni in MC for water

genMoleculeTable.f90 : program genMoleculeTable .
generate the positions of the atoms for the given composition

geometry.f90 : Module geometry .
Data structures and functions for geometrical calculations

io.f90 : Module io .
input/output interface

LJTypes.f90 : Module LJTypes .

Lennard Jones parameters for the pairs of atom types

matrix3x3.f90 : Module matrix3x3 .
Operations with matrices 3x3

mc_accum_luc.f90 : program mc_accum_luc .
calculate the projections

MCAccumLuc.f90 : Module MCAccumLuc .
Module which contains the functions necessary for the projection calculations

mc_calc_forces.f90 : program mc_calc_forces .
Calculate the forces for the given trajectory and store them to file

mccont.f90 : program mccont .
Continue the MC simulation

MC.f90 : Module MC .
The Module which defines the functions and data structures needed to perform the monte carlo steps

MCLuc.f90 : Module MCLuc .
Module which includes the procedures written by Luc Belloni for the MC for water. Is used for the test purposes only (to check that my results are the same as Luc's).

mc_main.f90 : program runmc_main .
Main Program: runmc

mc_make_round_holes.f90 : program mc_make_round_holes .
create spherical holes in the given box with molecules

mc_mean_force.f90 : program mc_mean_force .
calculate the mean forces for specific molecules

mcrdf.f90 : program mcrdf .
calculate the Radial distribution functions between the atoms

mc_rmsd.f90 : program mc_rmsd .
calculate the displacement from the original position for a given set of molecules

module_periodic_table.f90 : Module periodic_table .
this module contains the masses for the elements in periodic table actually, it is not used, since the mass now is given in mol file

Molecule.f90 : Module Molecule .
The module contains the datastructure to store the structure of the molecule

MoleculeHandler.f90 : Module MoleculeHandler .
the global storage for the molecule structures. Each molecule has it's identifier in MoleculeHandler

MoleculeTable.f90 : Module MoleculeTable .
Molecule table contains the coordinates and orientations of the molecules and the indeces of molecule type by atom types and first and last atom indeces in each molecule

moltab2xyz.f90 : program moltab2xyz .
convert the binary file with molecular positions to the xyz format

moltab_bin2text.f90 : program moltab_bin2text .
Convert the binary molecular coordinates to the text form

moltab_text2bin.f90 : program moltab_text2bin .
Convert the text file x y z theta phi psi to the binary moltab format

MonteCarloMove.f90 : Module MonteCarloMove .

parameters.f90 : Module Parameters .
parameters of the simulation

random.f90 : Module MRandom .
functions to work with random numbers random number

RealSumLocal.f90 : Module RealSumLocal .

RhoSquared.f90 : Module RhoSquared .
the module contains the coordinate-dependent sums of sin and cos for the Ewald sumation in KSpace

runmc.f90 : Module runmc .
Module which performs the MC cycles. Is called from the mc_main.f90

scale_box.f90 : Module ScaleBox .
change the size of the box (and thus the atom coordinates)

string.f90 : Module String .
Functions to work with the strings

SumSinCosKR.f90 : Module SumSinCosKR .
SUM_i cos(kR_i), SUM_i sin(kR_i). Used in EwaldSums in KSpace

SystemSettings.f90 : Module SystemSettings .
System-dependent constants

List of the data types, functions and subroutines

AtomicData.f90 : Module AtomicData

The module contains the data structures and the functions to manipulate the coordinated of the atoms

Type TAtomicData

Fields:
xx,yy,zz - coordinates of atoms in internal coordinates: BoxLength == 1
sigma,epsilon,charge - LJ parameters of the atoms in internal coordinates: sigma/BoxLength, epsilon/kT, charge/e
hard_core_angstr - in angstroems, because internal units are very inconvenient
atomnames - labels of the atoms
molnum_by_atomnum - the index which can be used to detemine the molecule number of the atom with a given number
BoxLength - in Angstroems
natom - number of atoms
nalloc - allocated size of the arrays
hasAtomnames - indicates that the labels of the atoms was read from the input files (can be used for export into xyz format)

subroutine AtomicData_alloc

Declaration:
subroutine AtomicData_alloc(this,nalloc,BoxLength,allocAtomNames)

Description:
Allocate the arrays in the AtomicData structure

Parameters:
this - the AtomicData structure (contains the arrays to be allocated)
nalloc - size of the arrays
BoxLength - Length of the box (in Angstroems)
allocAtomNames - whether or not the atom label arrays should be allocated

subroutine AtomicData_dealloc

Declaration:
subroutine AtomicData_dealloc(this)

Description:
Deallocate the AtomicData structure

Parameters:
this - AtomicData structure

subroutine AtomicData_save_to_xyz

Declaration:
subroutine AtomicData_save_to_xyz(this,filename)

Description:
writes the AtomicData structure to the file in xyz format

Parameters:
this - AtomicData structure
filename - the name of the file to save the data

BiasedRandom.f90 : Module BiasedRandom

BiasedRandom - generate the random numbers 1..N , where the probability of i is proportional to prob(i)
TBiasedRandom -
Fields:
N - number of intervals
freq_prob - probabilities to move the particle: freq_prob = freq_num / SUM(freq_num)
freq_beg, - freq_end begins and ends of the intervals for each molecule

subroutine BiasedRandom_alloc

Declaration:
subroutine BiasedRandom_alloc(this,N)

Description:
Allocate the BiasedRandom structure

Parameters:
this - BiasedRandom structure
N - number of the elements to be allocated

subroutine BiasedRandom_dealloc

Declaration:
subroutine BiasedRandom_dealloc(this)

Description:
Deallocate the BiasedRandom structure

Parameters:
this - BiasedRandom structure

subroutine BiasedRandom_init

Declaration:
subroutine BiasedRandom_init(this, freq_num)

Description:
initializes the frequency array in the BiasedRandom structure

Parameters:
this - BiasedRandom structure
freq_num - frequences distribution (maybe not normalized to 1)

function BiasedRandom_choose

Declaration:
function BiasedRandom_choose(this)

Description:
chooses the random number using the given frequences defined in BiasedRandom structure

Parameters:
this - BiasedRandom structure
Return value:
random integer number from 1..N with probability proportional to the given frequences array

clearmc.f90 : program clearmc

the program is used to clear the files created during the simulation, such as parameter.prm, trajectory and frames files.
Deletes all the files listed in parameters.prm namely: energy_file,boxlength_file,frames_file,traj_file
Is used before re-start the simulation.
Usage:
clearmc parameters.prm

Arguments:
parameters.prm - the file with the parameters of the simulation

subroutine rm_file

Declaration:
subroutine rm_file(fname)

Description:
remove the file with a given name

Parameters:
fname - name of the file

composition.f90 : Module composition

the module contains the data structure and the functions to deal with the system composition (numbers of the molecules of each type)
TComposition -
Fields:

n_types - number of different types of molecules
mol_types - types (indeces in MoleculeHandler array) of the molecules.
mol_numbers - numbers of the molecules of each type
nalloc - allocated size of the arrays

subroutine Composition_nulify

Declaration:
subroutine Composition_nulify(this)

Description:
set the composition to the "zero" state. Is used before the Composition_read_from_file, where the arrays are allocated automatically.

Parameters:
this - composition structure

subroutine Composition_alloc

Declaration:
subroutine Composition_alloc(this,nalloc)

Description:
allocate the composition structure

Parameters:
this - composition structure
nalloc - size of the arrays to be allocated

subroutine Composition_dealloc

Declaration:
subroutine Composition_dealloc(this)

Description:
deallocate the composition structure

Parameters:
this - composition structure

subroutine Composition_dealloc_molecules

Declaration:
subroutine Composition_dealloc_molecules(this)

Description:
deallocate the molecules which were allocated when read form the file. Can be used before the Composition_dealloc

Parameters:
this - composition structure

subroutine Composition_read_from_file

Declaration:
subroutine Composition_read_from_file(this, fname, dont_load_molecules)

Description:
read the composition from file
The format of the composition file: first line - number of species,
each next line - name of the mol file and the number of molecules.

Parameters:
this - composition structure
fname - name of the composition file
dont_load_molecules - flag which shows that the structures of the molecules should not be read

function Composition_count_molecules

Declaration:
function Composition_count_molecules(this)

Description:
Count the molecules in the composition

Parameters:
this - composition structure
Return value:
number of the molecules in the composition

function Composition_count_atoms

Declaration:
function Composition_count_atoms(this)

Description:
Count the total number of atoms in the composition

Parameters:
this - the composition structure
Return value:

the total number of atoms in all molecules in the composition

constants.f90 : Module Constants

The module contains the constants used in the program
pi - pi
two_pi - 2*pi
four_pi - 4*pi
boltz - Boltzmann constant in Jouls
elec - charge of the electron in Coulombs
epsi0 - dialectrical permutivity of vacuum [in SI]
kcal_mol - kcal/mol in Joul
LN2 - natural logarithm of 2

error.f90 : Module Error

The module contains the functions and data structures to deal with run-time errors
how do errors work:
if there is an error in the program, the function sets error_message to some value, and run throw with some error code

throw checks, if the error is in catch list.
if it is, that means that the programmer thought about the possibility of that error, thus functions just returns
otherwise, the executions stops
error codes:
ERROR_IO - input/output error
ERROR_LIMITS - error with limits (sizes of arrays)
ERROR_PARAMETER - incorrect parameter given
ERROR_INITIALIZATION - the function was called before the initialization
ERROR_WRONG_FUNCTION - the function was called for incorrect data

subroutine error_set_catch

Declaration:
subroutine error_set_catch(err_code)

Description:
the function which is used to set the error catch for the specific error code, which means that this code will not cause the stop of the program

Parameters:
err_code - error code

subroutine error_clear_catch

Declaration:

subroutine error_clear_catch(err_code)

Description:
the function clears the catch for the error, that means that the errors with err_code will cause the stop of the program

Parameters:
err_code - error code

subroutine error_throw

Declaration:
subroutine error_throw(err_code)

Description:
the subroutine is called then some errorneus situation occurs.

Parameters:
err_code - err_code describes the situation (see error codes above)

EwaldSumExternal.f90 : Module EwaldSumExternal

The module contains the data structures and the functions to deal with the external permutivity component of the ewald sums, i.e.
2 pi / (2 external_permutivity + 1) (mu_x^2 + mu_y^2 + mu_z^2)
where mu = SUM_j q_j r_j
TEwaldSumExternal -
Fields:
mu_x,mu_y,mu_z - mu = SUM_j q_j r_j
xx,yy,zz,charge - pointers to the coordinates and charges arrays (usually stored in AtomicData)
natom - number of atoms

subroutine EwaldSumExternal_init

Declaration:
subroutine EwaldSumExternal_init(this,xx,yy,zz,charge,natom)

Description:
initialize the EwaldSumExternal structure

Parameters:
this - EwaldSumExternal structure
xx,yy,zz,charge - coordinates and charges
natom - number of atoms

subroutine EwaldSumExternal_calc_mu

Declaration:
subroutine EwaldSumExternal_calc_mu(this)

Description:
calculate the moment mu=(mu_x,mu_y,mu_z)
this - EwaldSumExternal structure
pure

function EwaldSumExternal_calc_energy

Declaration:
function EwaldSumExternal_calc_energy(this)

Description:
calculate the energy E = kext * (mu_x^2 + mu_y^2 + mu_z^2)
where kext = 2 pi / (2 external_permutivity + 1)

Parameters:
this - EwaldSumExternal structure
Return value:
external permutivity component of the ewald sum
pure

function EwaldSumExternal_calc_dU

Declaration:
function EwaldSumExternal_calc_dU(this, dmu_x,dmu_y,dmu_z)

Description:
Calculate the energy difference dU = kext*[(mu+dmu)^2 - mu^2]

Parameters:
this - EwaldSumExternal structure
dmu_x, - dmu_y, dmu_z x,y,z components of the dmu vector

subroutine external_sum_calc_forces

Declaration:
subroutine external_sum_calc_forces(mu_x, mu_y, mu_z, charge, natom, fx,fy,fz)

Description:
Calculate the forces F = dU/dr

Parameters:
mu_x,mu_y,mu_z - components of mu vector
charge - charges (array)
natom - number of atoms (length of charge array)
fx,fy,fz - Output arguments: forces (arrays)

EwaldSumKSpace.f90 : Module EwaldSumKSpace

This Module contains structures and functions which deal with the K-Space component of the Ewald Sum (both LJ and Coulomb)
TEwaldSumKSpace -
Fields:
grid - K-Space grid
rho_squared_total - RhoSquared structure which contains the SUM cos(kR), SUM_i sin(k*R_i) for Coulomb and LJ (see Module RhoSquared)
atomic_data - Pointer to the AtomicData structure
lj_types - LJTypes structure which contains the individual and pair LJ parameters for the atoms
beta, - beta6, beta12 beta(k) (for coulomb), beta6(k) and beta12(k) - for LJ Ewald
beta_LJ - beta_LJ^t1t2(k) = 4 epsilon(sigma^12 beta12(k) - sigma6 beta6(k))
energy,energy_coulomb,energy_LJ - total energy and energy components
Temporary arrays for delta_LJ_energy calculations (EwaldSumKSpace_calculate_dU_LJ)
twoC_plus_dC_ppp,twoC_plus_dC_ppm,twoC_plus_dC_pmp,twoC_plus_dC_pmm -
twoS_plus_dS_ppp,twoS_plus_dS_ppm,twoS_plus_dS_pmp,twoS_plus_dS_pmm -

subroutine EwaldSumKSpace_alloc

Declaration:
subroutine EwaldSumKSpace_alloc(this,atomic_data,lj_types)

Description:
Allocate the EwaldSumKSpace structure

Parameters:
this - EwaldSumKSpace structure
atomic_data - AtomicData (coordinates + charges)
lj_types - LJTypes array (sigma,epsilon for each pair)

subroutine EwaldSumKSpace_dealloc

Declaration:
subroutine EwaldSumKSpace_dealloc(this)

Description:
Deallocate the EwaldSumKSpace structure

Parameters:
this - EwaldSumKSpace structure

subroutine EwaldSumKSpace_init

Declaration:
subroutine EwaldSumKSpace_init(this,scale_beta)

Description:
initialize all the arrays needed for the calculation

Parameters:
this - EwaldSumKSpace structure
scale_beta - if is used within the volume change step the beta(k) should be scaled. Then scale_beta is the coefficient. Otherwise, it can be not given (optional) or zero. Then the beta(k) will be re-calculated

subroutine EwaldSumKSpace_calc_total_energy

Declaration:
subroutine EwaldSumKSpace_calc_total_energy(this)

Description:
Calculate the Ewald KSpace component of the energy

Parameters:
this - EwaldSumKSpace structure

subroutine EwaldSumKSpace_initBeta

Declaration:
subroutine EwaldSumKSpace_initBeta(this)

Description:
Calculate the beta function. Is called from EwaldSumKSpace_init

Parameters:
this - EwaldSumKSpace structure

subroutine EwaldSumKSpace_initBetaLJ

Declaration:
subroutine EwaldSumKSpace_initBetaLJ(this)

Description:
Calculate the betaLJ(k)
betaLJ(t1t2) = epsilon(t1,t2)* (sigma(t1,t2)^12* beta12 - sigma(t1,t2)^6 * beta6)

Parameters:

this - EwaldSumKSpace structure

function EwaldSumKSpace_calc_coulomb_energy

Declaration:
function EwaldSumKSpace_calc_coulomb_energy(this)

Description:
calculate the KSpace coulomb energy component
which is SUM beta(k) rho_squared(k)

Parameters:
this - EwaldSumKSpace structure
Return value:
coulomb energy component

function EwaldSumKSpace_calc_LJ_energy

Declaration:
function EwaldSumKSpace_calc_LJ_energy(this)

Description:
calculates LJ energy using the beta_LJ:
note : C^t(sxsysz),S^t(sxsysz) should be pre-calculated (rho_square_total)

Parameters:
this - EwaldSumKSpace strucutre
Return value:
LJ component of the energy

function EwaldSumKSpace_calc_dU_coulomb

Declaration:
function EwaldSumKSpace_calc_dU_coulomb(this, delta_sumsincos_coulomb)

Description:
Calculates the coulomb energy change dU

Parameters:
this - EwaldSumKSpace structure
delta_sumsincos_coulomb - changes of the sums of sins and cos (for the given molecule) (see Module SumSinCosKR)
Return value:
coulomb energy change dU

function EwaldSumKSpace_calc_dU_LJ

Declaration:
function EwaldSumKSpace_calc_dU_LJ(this, delta_sumsincos_LJ,type_is_present)

Description:
calculate the LJ energy change

Parameters:
delta_sumsincos_LJ - changes of sums of sin and cos (for a given molecule).
type_is_present - logical array which indicates which LJ diameters are present in the delta_sumsincos_LJ
Return value:
Change of the LJ energy dU_LJ

EwaldSumRealSpace.f90 : Module EwaldSumRealSpace

Data structures and functions which deal with the real component of the Ewald sum
TEwaldSumRealSpace -
Fields:
atomic_data - AtomicData (coordinates, charges)
lj_types - LJ parametrs for each pair of atom types
first_atom,last_atom - the first and the last atoms (used for the partial sums which include only the atoms of the certain molecule)
natom,nalloc - number of atoms and the size of the allocated arrays
uu_ew, - uu_lj6, uu_lj12 electrostatic, LJ6 and LJ12 components of the energy
uu_ew_intra,uu_lj6_intra,uu_lj12_intra - components of the intra-molecular interaction (not used to my knowledge)
uu - energy per atom. U = sum_i uu(i)
uu_back - for the full sum uu_back => uu. Otherwise - it is the energy change per molecule
Fx,Fy,Fz - 1..Nnew: forces on the atoms of the molecule (at new coordinates)
Fx_back,Fy_back,Fz_back - 1..Ntot: forces of the new coords on the all molecules in the system
for the full sum Fx,y,z_back => Fx,y,z
xmol,ymol,zmol - 1..Nmol: positions of atoms of the molecule (pointer to xx,yy,zz)
for the full sum xmol,ymol,zmol => atomic_data % xx,yy,zz
full_sum - logcal which indicates the full (or molecular) sum

subroutine EwaldSumRealSpace_alloc_full

Declaration:
subroutine EwaldSumRealSpace_alloc_full(this, atomic_data, lj_types)

Description:
allocates the arrays for the full sum

Parameters:
this - EwaldSumRealSpace structure
atomic_data - coordinates and charges
lj_types - LJ parameters for each pair of atom types

subroutine EwaldSumRealSpace_alloc_partial

Declaration:
subroutine EwaldSumRealSpace_alloc_partial(this,atomic_data,lj_types)

Description:
allocate arrays for the partial (molecular) sum
this - EwaldSumRealSpace
atomic_data - AtomicData (coordinates and charges)
lj_types - LJ parameters for each pair of atom types

subroutine EwaldSumRealSpace_set_molecule

Declaration:
subroutine EwaldSumRealSpace_set_molecule(this, xmol, ymol, zmol, first_atom, last_atom)

Description:
set the molecule (to the partial sum only)

Parameters:
this - EwaldSumRealSpace
xmol,ymol,zmol - coordinates of the molecule atoms (either pointer to AtomicData or xnew,ynew,znew arrays)
first_atom, - last_atom first_atom, last_atom are used to know charges and lj parameters

subroutine EwaldSumRealSpace_dealloc_full

Declaration:
subroutine EwaldSumRealSpace_dealloc_full(this)

Description:
Deallocate the full sum

Parameters:
this - EwaldSumRealSpace structure

subroutine EwaldSumRealSpace_dealloc_partial

Declaration:
subroutine EwaldSumRealSpace_dealloc_partial(this)

Description:
Deallocate the partial sum
this - EwaldSumRealSpace structure

subroutine EwaldSumRealSpace_calc

Declaration:
subroutine EwaldSumRealSpace_calc(this,overlap,first,last)

Description:
Calculate the sum. The energy per atom uu and forces arrays will also be initialized
note: function works for both cases : full sum and sum of one molecule
in case of the full sum xnew = xx, ynew = yy, znew = zz
otherwise, other arrays xmol,ymol,zmol should be provided
for the full sum run EwaldSumRealSpace_calc(this,this % atomic_data % xx,this % atomic_data % yy, this % atomic_data % zz)

Parameters:
this - EwaldSimRealSpace
overlap - logical output: indicates if the overlap occured during the sum calculation
first,last - for the calculation of the molecule-molecule interactions, like u12. For the usual (even partial) sum calculations first=1, last=natom (or can be omited and thus set by default).

EwaldSumTails.f90 : Module EwaldSumTails

This module contains the functions to compute the coordinate-independent parts of the Ewald sum
The position-independent "tails" of the ewald sum:

Coulomb Term:
alpha / sqrt(pi) SUM q_i^2

LJ6 Term:
1/6 pi^1.5 alpha^3 / V * SUM_ij A_ij - alpha^6/12 SUM_j A_jj

LJ12 Term:
1/1080 V * pi^1.5 alpha^9 SUM_ij A_ij - alpha^12/1440 SUM_j A_jj

function ewald_sum_coulomb_tail

Declaration:
function ewald_sum_coulomb_tail(charge , natom)

Description:
computes the position-independent coulomb term

Parameters:
charge - charges array
natom - number of atoms
Return value:
alpha / sqrt(pi) SUM q_i^2

subroutine ewald_sum_lj_tails

Declaration:
subroutine ewald_sum_lj_tails(comp, lj_types, lj6_tail, lj12_tail)

Description:
Compute the position-independet terms of the Ewald LJ sums

Parameters:
comp - composition of the system
lj_types - LJ parameters for each pair of atom types
lj6_tail - ouput: LJ6 term
lj12_tail - output: LJ12 term

FloatingPoint.f90 : Module FloatingPoint

Module to work with the floating point numbers (construct from mantisse and exponent, store to and restore from file)
Why in general to use this module, not the standard one functions?
Because, the standard functions can be system- or hardware- dependent.
This module is completely system independent
subroutines to extract mantisse and exponent
in representation a = sign*m*2^n, where 1 <= m < 2

Format to save:
mm = (|m|-1)*(2^(mw-1)-1) + (m<0)*2^(mw-1)
nn = |n| + (n<0)*2^(nw-1)

where mw stands for "mantisse width", number of bits for mantisse (normally 24)
nw - "exponent width", number of bits for exponent (normally 8)

The values mm,nn are stored in the file in the little-endian format.
I did not optimize for non-whole byte widths, so if e.g. mv=18,nw=6 it will use 4 bytes anyway

function myint

Declaration:
function myint(x)

Description:
myint is the maximal integer number smaller than x

Parameters:
x - real number x
Return value:
maximal integer smaller than x

subroutine mantisse_exponent_real

Declaration:
subroutine mantisse_exponent_real(a,m,n)

Description:
Get mantisse and exponent for the real number

Parameters:
a - real number
m - output: mantisse(real)
n - output: exponent(integer)

subroutine mantisse_exponent_bits

Declaration:
subroutine mantisse_exponent_bits(m,n,mantisse_width,exp_width,mm,nn)

Description:
convert mantisse and exponent to the integer numbers
a = sign * m * 2^n
m,n taken from mantisse_exponent_real

Parameters:
m - mantisse (real)
n - exponent (integer)
mantisse_width,exp_width - number of bits in mantisse and exponent

function construct_float

Declaration:
function construct_float(mm,nn,mantisse_width,exp_width)

Description:
construct the real number from integer values representing mantisse and exponent

Parameters:
mm,nn - integer values representing mantisse and exponent
mantisse_width,exp_width - number of bits in mantisse and exponent

subroutine write_float

Declaration:
subroutine write_float(hfile,x,mantisse_width_in,exp_width_in)

Description:
write real number to file

Parameters:
hfile - file handler
x - real number
mantisse_width_in,exp_width_in - number of bits in mantisse and exponent

function read_float

Declaration:
function read_float(hfile,mantisse_width_in,exp_width_in)

Description:
read real number from file

Parameters:
hfile - file handler
mantisse_width_in,exp_width_in - number of bits in mantisse and exponent
Return value:
real number read from file

ForceKSpace.f90 : Module ForceKSpace

Module to calculate forces originating from KSpace Ewald sum

subroutine ForceKSpace_coulomb

Declaration:
subroutine ForceKSpace_coulomb(beta,sumsincos_coulomb,sincos_kr_coulomb, fx_tot, fy_tot, fz_tot)

Description:
Coulomb component of forces
beta - in reality it is already (1+sgn(kx)) beta(k_m)
sumsincos_coulomb - SumSinCosKR structures for coulomb part: SUM_i sin(kR_i), SUM_i cos(kR_i)
sincos_kr_coulomb - in this case it is q_p sin,cos(sx*x*kx + sy*y*ky +sz*zk*z)
to be calculated with SinCosKR_fill...
fx_tot,fy_tot,fz_tot - output: components of the force

subroutine ForceKSpace_LJ

Declaration:
subroutine ForceKSpace_LJ(beta_LJ,sumsincos_LJ,ntype,ityp,sincos_kr_ityp, fx_tot, fy_tot, fz_tot)

Description:

Calculate the LJ component of forces
beta_LJ - 2D array (t1,t2) <--> beta_LJ((t1-1)*Ntype + t2)
sumsincos_LJ - sumsincos for all types (see SumSinCosKR)
ntype - number of types
ityp - type of the atom for which we are calculationg the forces
sincos_kr_ityp - sin,cos(sx*x*kx + sy*y*ky + sz*z*kz) for that atom
fx_tot,fy_tot,fz_tot - output: the force components

forces2text.f90 : program forces2text

Convert the binary frame with forces trajectory to the text format
Usage:
forces2text forces.ftab > forces.txt'

Arguments:
forces.ftab - one frame of force trajectory (NAtom * 12 bytes, Natom*3 floats) '
forces.txt - file which contains the text representation of forces components (3 columns)

FourierGrid.f90 : Module FourierGrid

Grid in fourier space
includes all k such that |kk|<kmax
TFourierGrid -
Fields:
kmax - limit for |k| < kmax
nk - number of grid points
nalloc - size of allocated arrays
ip - next raw indicator
ip(kk) = 0 -- no charnges
ip(kk) = 1 -- next ky
ip(kk) = 2 -- next kz
ip(kk) = 3 -- end of array
iq - zero indicator (if kx,ky,or kz == 0)
iq(kk) = 1 --> ky=0, kz=0
iq(kk) = 2 --> ky>0, kz=0
iq(kk) = 3 --> ky=0, kz>0
iq(kk) = 4 --> ky>0, kz>0
kx,ky,kz - indeed these are mx,my,mz in my notation, where k = 2pi m/L
xk,xk2 - |k|, k^2

subroutine FourierGrid_calcNalloc

Declaration:
subroutine FourierGrid_calcNalloc(kmax,nalloc)

Description:
calculate number of grid points for a given kmax

Parameters:

subroutine FourierGrid_alloc

Declaration:
subroutine FourierGrid_alloc(this,nalloc)

Description:
allocate the KSpace grid

Parameters:
this - FourierGrid structure
nalloc - number of k values

subroutine FourierGrid_dealloc

Declaration:
subroutine FourierGrid_dealloc(this)

Description:
deallocate the KSpace grid

Parameters:
this - FourierGrid structure

subroutine FourierGrid_init

Declaration:
subroutine FourierGrid_init(this,kmax)

Description:
initialize the KSpace grid (fill the kx,ky,kz,k^2,ip,iq arrays)

Parameters:
this - FourierGrid structure
kmax - maximum value for |k|

Functions.f90 : Module Functions

Functions used in calculation of Ewald sums
Written by Luc Belloni in MC for water

subroutine erfc_Luc_bet6_bet12

Declaration:
subroutine erfc_Luc_bet6_bet12(x,x2,e2,erfk,bet6,bet12)

Description:
calculate the beta6 and beta12 values

Parameters:
x, x2 --> b, b^2
e2 --> exp(-b^2)
erfk --> erfc(b)
b^2 = pi^2 h_m^2 / alpha --> h_m = sqrt(alpha) b/pi
avec erfc_Luc a x<0.5, DL a x>8, integration numerique de x a 8 entre les 2
luc84p150

function erfc_Luc

Declaration:
function erfc_Luc(X,x2,ee2)

Description:
calcualte erfc(x)

Parameters:
x --> b
x2 --> b^2
ee2 --> exp(-b^2)
Return value:
erfc(b)
calcule erfc_Luc(x)=2/racine(pi) integrale de x à infini de exp(-t**2)dt
x2=x**2 et ee2=exp(-x**2)
luc85p108

function erf_Luc

Declaration:
function erf_Luc(X,x2,ee2)

Description:
Calculate the erf(X)

Parameters:
X --> b
x2 --> b2
ee2 --> exp(-b^2)
Return value:

erf(b)
calcule erf_Luc(x)=2/racine(pi) integrale de 0 à x de exp(-t**2)dt
x2=x**2 et ee2=exp(-x**2)
luc85p108

function shi_sans_exp_f

Declaration:
function shi_sans_exp_f(x)

Description:
calcul auto de Shi(x)
luc84p163
en fait, on ne veut pas de facteur exp(x) qui peut occasionner un overflow
donc donner plutot Shi(x)/exp(x)

genMoleculeTable.f90 : program genMoleculeTable

generate the positions of the atoms for the given composition
Usage:
genMoleculeTable system.composition output.moltab [nbytes_xyz nbytes_ang [random_seed]]

Arguments:
output.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi)
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)
random_seed - - optional argument. The seed for the random number generator

geometry.f90 : Module geometry

Data structures and functions for geometrical calculations
TRotation -
Rotation matrix in Luc's convention (i don't know how it works)
used in choosing the rotation of the molecule
Fields:
sin_phi,cos_phi - sin(phi),cos(phi)
sin_theta,cos_theta - sin(theta), cos(theta)
sin_angle,cos_angle - sin(angle), cos(angle)
TRotMatrix -
Rotational matix 3x3
Fields:
xx,xy,xz -
yx,yy,yz -
zx,zy,zz -

subroutine rot_vect

Declaration:
subroutine rot_vect(x,y,z,x1,y1,z1,cc,ss,i)

Description:
rotate the coordinates (x,y,z) by angle alpha along one of 3 axes: x,y,z
fait la rotation de x,y,z en x1,y1,z1 d'angle cc=cos,ss=sin autour d'un (i) des 3 axes principaux
attention: x1,y1,z1 peut etre a la meme place memoire que x,y,z
luc80p176
Taken from the MC code for H2O by Luc Belloni, no changes

Parameters:
x,y,z - - coordinates of the first vector
x1,y1,z1 - - output coordinates
cc,ss - - cos(alpha),sin(alpha)
i - - number of the axis: 1 is x, 2 is y, 3 is z

subroutine prod_vect

Declaration:
subroutine prod_vect(x1,y1,z1,x2,y2,z2,xprod,yprod,zprod)

Description:
vector product

Parameters:
x1,y1,z1 - first vector
x2,y2,z2 - second vector
xprod,yprod,zprod - output: result

subroutine fill_rot_matrix

Declaration:
subroutine fill_rot_matrix(rot,rot_matrix)

Description:
convert rotation in Luc's notation to the rotational matrix

Parameters:
rot - rotational structure in Luc's notation
rot_matrix - 3x3 rotational matrix

subroutine rotate_vect

Declaration:
subroutine rotate_vect(R,xr,yr,zr, xrnew, yrnew, zrnew)

Description:
Rotate vector using the rotational matirx R

Parameters:
R - rotational matrix
xr,yr,zr - input coordinates
xrnew, - yrnew, zrnew coordinates after rotation

subroutine center_of_mass

Declaration:
subroutine center_of_mass(xx,yy,zz,mass,natom, x_center, y_center, z_center)

Description:
compute center of mass of the molecule

Parameters:
xx,yy,zz,mass - coordinates and masses of atoms
natom - number of atoms
x_center,y_center,z_center - output: coordinates of the center of mass

function atan2_two_pi

Declaration:
function atan2_two_pi(y , x)

Description:
compute arctangent from y and x and convert it to the angle from 0 to 2pi

subroutine ZYZRotation_matrix_to_angles

Declaration:
subroutine ZYZRotation_matrix_to_angles(R, theta, phi, psi)

Description:
Convert the rotational matrix to angles, theta phi,psi

Parameters:
R - Rotation matrix
theta,phi,psi - output: angles
Conventional rotation in my case (not Luc's) is:
1) rotate over Oz by 0<psi<2pi
2) rotate over Oy by 0<theta<pi
3) rotate over Oz by 0<phi<2pi

This rotation is used in MoleculeTable for example
I refer it as "ZYZ rotation"
do not mix this with the "Luc's rotation" which is different
ZYZRotation can be defined either by angles (theta,phi,psi) or by the rotation matrix
(note, that direction (clockwise or counter-clockwise is also important)

subroutine rotation_matrix

Declaration:
subroutine rotation_matrix(X_before, X_after, R)

Description:
Calculate the rotation matrix which converts X_before to X_after
i.e. X_after = R*X_before, R = X_after*X_before^-1

Parameters:
X_before,X_after - coordinates of 3 points before and after rotation
R - output: rotation_matrix

subroutine xyz_to_angles

Declaration:
subroutine xyz_to_angles(xx,yy,zz, xx_new,yy_new,zz_new,natom, theta, phi, psi)

Description:
compute the rotation matrix for the given coordinates of atoms

Parameters:
xx,yy,zz - inital coordinates
xx_new, - yy_new, zz_new new coordinates of atoms
should be centered by center of mass (both: before & after)
natom - number of atoms
theta,phi,psi - output: angles which describe the rotation

subroutine xyz_to_angles_two_atoms

Declaration:
subroutine xyz_to_angles_two_atoms(x,y,z,x_new,y_new,z_new, theta, phi, psi)

Description:
special case of the previous subroutine for the case of 2atom molecule
one atom is expected to be in the origin
only coordinates of non_zero atom
x,y,z,x_new,y_new,z_new - coordinates of atoms
should be centered by center of mass
theta,phi,psi - output angles

io.f90 : Module io

input/output interface

subroutine io_init

Declaration:
subroutine io_init

Description:
initialize the io module

subroutine close_all

Declaration:
subroutine close_all

Description:
close all the opened files
integer

function io_open

Declaration:
function io_open(filename,mode)

Description:
opens the file and return file handler

Parameters:
filename - name of the fule
mode - mode: r - read, w - write

subroutine io_close

Declaration:
subroutine io_close(hfile)

Description:
close the file opened with io_open

Parameters:
hfile - file handler

function io_count_file_lines

Declaration:
function io_count_file_lines(filename)

Description:
Count number of lines in the file

Parameters:
Return value:
number of lines in the file

subroutine write_little_endian

Declaration:
subroutine write_little_endian(fid, val, n)

Description:
Write the integer value to the file
Parameter:
fid - file handler
val - the integer value
n - numer of bytes to be written

subroutine read_little_endian

Declaration:
subroutine read_little_endian(fid,n,val)

Description:
read integer value form file

Parameters:
fid - file descriptor
n - number of bytes to read
val - output: value

subroutine write_real_array

Declaration:
subroutine write_real_array(hFile,arr,n)

Description:
write the array to file or on the screen

Parameters:
hFile - file handler. 0 means screen
arr - array
n - number of elements in the array

subroutine write_xyz_array

Declaration:
subroutine write_xyz_array(hFile,arr_x,arr_y,arr_z,n)

Description:
write 3 arrays representing x,y,z coordinates

Parameters:
hFile - file handler (0 means screen)
arr_x,arr_y,arr_z - arrays
n - number of elements

subroutine write_real_matrix

Declaration:
subroutine write_real_matrix(hfile, matrix, m, n)

Description:
write the matrix m*n to the file

Parameters:
matrix - the matrix: array of arrays
m,n - size of the matrix

subroutine write_integer_array

Declaration:
subroutine write_integer_array(hFile,arr,n)

Description:
write real array

Parameters:
hFile - file handler
arr - array
n - number of elements

LJTypes.f90 : Module LJTypes

Lennard Jones parameters for the pairs of atom types

Type LJIndex

the structure to keep the indeces of the atoms which have specific atom type
Fileds:
sigma,epsilon - sigma and epsilon parameters
idx - array of indeces
n,nalloc - number of indeces and allocated size of array

Type TLJTypes

Fields:
index_by_type - get LJIndex for specific atom type
type_by_index - get type by atom index
NType - number of atom types
LJ6Tab - table of LJ coefficients near 1/r^6, e.g. 4epsilon_ij sigma_ij^6
LJ12Tab - table of LJ coefficient near 1/r^12
four_epsilon_tab - table for 4*epsilon_ij
sigma6_tab - table for sigma_ij^6
sigma2_tab - table for sigma_ij^2. Used for checking the overlap

subroutine LJIndex_alloc

Declaration:
subroutine LJIndex_alloc(this,nalloc,sigma,epsilon)

Description:
allocate the LJIndex

Parameters:
this - LJIndex
nalloc - size to be allocated
sigma,epsilon - values of sigma and epsilon

subroutine LJIndex_dealloc

Declaration:
subroutine LJIndex_dealloc(this)

Description:
Dealloate the LJIndex
this - LJIndex

subroutine LJIndex_addIndex

Declaration:
subroutine LJIndex_addIndex(this,i)

Description:
add index to the LJIndex array

Parameters:
this - LJIndex
i - new index

subroutine LJTypes_dealloc

Declaration:
subroutine LJTypes_dealloc(this)

Description:
deallocate the LJTypes array

Parameters:
this - LJTypes structure

subroutine LJTypes_fill

Declaration:
subroutine LJTypes_fill(this,sigma_array,epsilon_array,nsigma)

Description:
Allocate and fill the LJType arrays

Parameters:
this - LJTypes structure
sigma_array, - epsilon_array arrays for simga and epsilon for each atom
nsigma - size of the input arrays

subroutine LJTypes_fill_LJ_tab_index

Declaration:
subroutine LJTypes_fill_LJ_tab_index(this,i,j,sigma12,epsilon12)

Description:
auxilarly subroutine to fill LJ6 LJ12, 4epsilon and other arrays

Parameters:
this - LJTypes
i,j - pair of indeces
sigma12,epsilon12 - sigma and epsilon for this pair
scales all sigma and epsilon. Can be called to re-calculate the LJTypes without re-allocation (for example when the BoxLength changes)

Parameters:
this - LJTypes structure
scale_coeff - scale coefficient

subroutine LJTypes_allocate_LJ_tab

Declaration:
subroutine LJTypes_allocate_LJ_tab(this)

Description:
allocate the LJ Tabs

Parameters:
this - LJTypes

subroutine LJTypes_fill_LJ_tab

Declaration:
subroutine LJTypes_fill_LJ_tab(this)

Description:
use Lorentz-Berthelot rules to fill LJ6 and LJ12 tables

Parameters:
this - LJTypes structure

matrix3x3.f90 : Module matrix3x3

Operations with matrices 3x3
(they are particullary intersting, because can be used for the coordinate-transformations)
Also, for these matrices one knows the explicit relations for determinant and inverse matrices

subroutine matrix3x3_mul

Declaration:
subroutine matrix3x3_mul(A,B,C)

Description:
matrix multiplication C=A*B

Parameters:
A,B - multiplicands
C - result
pure

function matrix3x3_det

Declaration:
function matrix3x3_det(A)

Description:
calculate matrix determinant

Parameters:
A - the matrix
Return value:
determinant det(A)

subroutine matrix3x3_adj

Declaration:
subroutine matrix3x3_adj(A,B)

Description:
auxilarly function to calculate the inverse matrix
adj(A) = det(A) * A^-1

Parameters:
A - input matrix
B - output matrix

subroutine matrix3x3_inv

Declaration:
subroutine matrix3x3_inv(A,B)

Description:
calculate matrix inverse

Parameters:
A - input matrix

B - B=A^-1

mc_accum_luc.f90 : program mc_accum_luc

calculate the projections

mc_accum_luc: do statistical processing of the simulation data'
Usage:
mc_accum_luc parameters.prm system.composition output.proj first_frame[-last_frame] [step_size] [maxmn]

system.composition - - number and types of molecules in the system
output.proj - - projections file

NOTE: quantities like pressure, compressibility etc are calculated incorrectly
NOTE: density is not updates and thus is calculated incorrectly (always 0.0332891 particles/nm^3).
to get a correct assymptote, the projections should be re-normalized to the real inverse density mean(V^-1) which can be get from frames.dat file

subroutine parse_command_line

Declaration:
subroutine parse_command_line

Description:
read the command line arguments

subroutine read_input_files

Declaration:
subroutine read_input_files

Description:
read input files

subroutine allocate_ewald

Declaration:
subroutine allocate_ewald

Description:
allocate the ewald sums

subroutine deallocate_ewald

Declaration:
subroutine deallocate_ewald

Description:
deallocate the ewald sums

subroutine calc_sums

Declaration:
subroutine calc_sums

Description:
calculate the ewald sums

subroutine load_frame

Declaration:
subroutine load_frame

Description:
load next trajectory frame

subroutine deallocate_all

Declaration:
subroutine deallocate_all

Description:
deallocate everything

MCAccumLuc.f90 : Module MCAccumLuc

Module which contains the functions necessary for the projection calculations
Is the interface to the functions written by Luc Belloni for the projection calculation

NOTE: quantities like pressure, compressibility etc are calculated incorrectly
NOTE: density is not updates and thus is calculated incorrectly (always 0.0332891 particles/nm^3).
to get a correct assymptote, the projections should be re-normalized to the real inverse density mean(V^-1)
which can be get from frames.dat file

subroutine accumu_init

Declaration:
subroutine accumu_init(comp,box_length,MAX_MN)

Description:
initialize the accumulation

Parameters:
MAX_MN - maximum value for m,n
comp - system composition
box_length - box length

subroutine accumu_set_frame

Declaration:
subroutine
accumu_set_frame(comp,atomic_data,box_length,kspace_sum,rspace_sum,ext_sum,inp_nittot,inp_naccep)

Description:
initializes the arrays used in projection calculation with the data read from file

Parameters:
comp - system composition
atomic_data - coordinates of atoms
box_length - box length
kspace_sum - EwaldSumKSpace structure
rspace_sum - EwaldSumRealSpace structure
ext_sum - EwaldSumExternal
inp_nittot, - inp_naccep total number of iterations and accepted number of iterations
subroutine accumu
do the accumulation
subroutine accumu_mnmunukhi
do the accumulation for projections

function vector

Declaration:
function vector(r1,r2)

Description:
fait le produit vectoriel r = r1 * r2
luc85 page 23

function harm_sph

Declaration:
function harm_sph(m,mu,mup,beta)

Description:
pour harmonique spherique Rm,mu,mup(Omega=omega,beta,phi)
=exp(-i*omega)*r(m)mu,mup*exp(-i*phi)
calcul l'element mu,mup de la matrice r(m) en fonction de l'angle beta
formule de Wigner dans Messiah eq.72 p922 betement
luc72p143
si mu ou mup nul, formule de recurrence stable
subroutine resul
get the results of accumulation (compute the projections)

function volume_sph_cub

Declaration:
function volume_sph_cub(r)

Description:
calcule le volume egal a l'intersection d'une sphere de rayon r et d'un cube de cote L
tout en unite L
formule de Caillol
ou ma formule, luc82p197
subroutine factoriel

function itriangle

Declaration:
function itriangle(m,n,l)

Description:
nul sauf si |m-n|<l<m+n
rq: ne depend pas de l'ordre des 3 entiers

function delta

Declaration:
function delta(m,n,l)

Description:

function symbol3j

Declaration:
function symbol3j(m,n,l,mu,nu,lu)

Description:
symbole 3j
Messiah page 910 eq.21

subroutine write_projections

Declaration:
subroutine write_projections(hfile)

Description:
write projections to the file

Parameters:
hfile - file handler

mc_calc_forces.f90 : program mc_calc_forces

Calculate the forces for the given trajectory and store them to file
mc_calc_forces: calculate forces for the given simulation trajectorory
Usage:
mc_calc_forces parameters.prm system.composition forces.ftraj [extra_parameters]

Arguments:
system.composition - - number and types of molecules in the system'
extra_parameters - - coma separated prm1=val1,prm2=val2,... NO SPACES ALLOWED! '
forces.ftraj - - forces trajectory file. In float (24+8) format. See FloatingPoint.f90 for details

subroutine parse_command_line

Declaration:
subroutine parse_command_line

Description:
read the command line arguments

subroutine read_input_files

Declaration:
subroutine read_input_files

Description:
read the input files

subroutine allocate_ewald

Declaration:
subroutine allocate_ewald

Description:
allocate the ewald sum structures

subroutine deallocate_ewald

Declaration:
subroutine deallocate_ewald

Description:
deallocate the ewald sum structures

subroutine calc_forces

Declaration:
subroutine calc_forces

Description:
calculate the forces

subroutine load_frame

Declaration:
subroutine load_frame

Description:
load the next trajectory frame

subroutine store_forces

Declaration:
subroutine store_forces

Description:

save the forces to the file

subroutine deallocate_all

Declaration:
subroutine deallocate_all

Description:
deallocate everything

mccont.f90 : program mccont

Continue the MC simulation
Usage:
mccont parameters.prm system.composition [extra_parameters]

Arguments:
system.composition - - number and types of molecules in the system
extra_parameters - - coma separated prm1=val1,prm2=val2,... NO SPACES ALLOWED!
first frame is the last frame from the traj_file (given in parameters)'
first boxlength is taken from the last frame in frames_file

subroutine write_energy_ME

Declaration:
subroutine write_energy_ME

Description:
write the energy components

subroutine parse_command_line

Declaration:
subroutine parse_command_line

Description:
read the command line arguments

subroutine read_input_files

Declaration:
subroutine read_input_files

Description:
read the input files

subroutine deallocate_all

Declaration:
subroutine deallocate_all

Description:
deallocate everything

MC.f90 : Module MC

The Module which defines the functions and data structures needed to perform the monte carlo steps
TMCState -
The structure is used to save the current MC state before the change of the volume
Fields:
kspace_energy, - kspace_energy_coulomb, kspace_energy_lj KSpace energy components
rspace_uu_ew, - rspace_uu_lj6, rspace_uu_lj12 Real space energy components
rspace_uu_ew_intra, - rspace_uu_lj6_intra, rspace_uu_lj12_intra Real space intramolecular components
U_tail, - U_tail_coulomb,U_tail_lj6,U_tail_lj12 total energies
U_ext - --
U_total,U_total_ew, - U_total_lj --
xx,yy,zz - coordinates
sigma - sigmas
fx,fy,fz,uu - real-forces and energy per atom
rho_squared_total - SUM of sin(kr), cos(kr)
mu_x,mu_y,mu_z - mu=SUM q_i*r_i

subroutine MCState_alloc

Declaration:
subroutine MCState_alloc(this, natom, grid, lj_types)

Description:
Allocate the storage for MCState structure
Parameters
this - MCState
natom - number of atoms
grid - KSpace grid
lj_types - LJTypes

subroutine MCState_dealloc

Declaration:

subroutine MCState_dealloc(this)

Description:
Deallocate the MCState
this - MCState

subroutine mc_save_state

Declaration:
subroutine mc_save_state(state)

Description:
Save the current state. Used in volume change

Parameters:
state - MCState structure

subroutine mc_restore_state

Declaration:
subroutine mc_restore_state(state)

Description:
restore the state from the saved ones

Parameters:
state - MCState structure

subroutine mc_alloc

Declaration:
subroutine mc_alloc(comp,mol_table,atomic_data,lj_types)

Description:
allocate the arrays needed for the MC step

Parameters:
comp - composition
mol_table - MoleculeTable
atomic_data - AtomicData
lj_types - LJTypes

subroutine mc_calc_total_sums_dbg

Declaration:
subroutine mc_calc_total_sums_dbg(mol_table,atomic_data,lj_types,scale_beta,overlap)

Description:
DEBUG ONLY

subroutine mc_calc_total_sums

Declaration:
subroutine mc_calc_total_sums(mol_table,atomic_data,lj_types,scale_beta,overlap)

Description:
calculate the KSpace and RealSpace sums

Parameters:
mol_table - Molecule Table (used to know how the atoms are grouped to the molecules)
atomic_data - AtomicData : atomic coordinates
lj_types - LJTypes for each pair of atom types
scale_beta - if the volume is changed - we need to scale the beta function. Otherwise, it can be re-calculated

subroutine initmc

Declaration:
subroutine initmc(comp,mol_table,atomic_data,lj_types)

Description:
initialize the data

Parameters:
comp - composition
mol_table - molecule table (how the atoms are grouped to the molecules)
atomic_data - coordinates of atoms
lj_types - LJ parameters for each pair of atom types

function mc_total_energy

Declaration:
function mc_total_energy()

Description:
calculate the total energy (using the already calculated ewald sums).
Return value:
total energy of the system

subroutine mcvol

Declaration:
subroutine mcvol(mol_table, accepted , force_new_box_length, force_accepted)

Description:
Volume step: try to change the volume of the box

Parameters:
mol_table - MoleculeTable
atomic_data - coordinates of atoms
lj_types - LJ parameters
force_new_box_length - force setting the new box length. For Debug Only! (usually omitted)
force_accepted - set accepted to force_accepted debug only! (usually omitted)

subroutine movemc

Declaration:
subroutine movemc(imol, mol_table, accepted)

Description:
Monte Carlo move step

Parameters:
imol - number of molecule to be tried to move (should be chosen randomly)
mol_table - Molecule Table
accepted - output: whether or not the move was accepted

subroutine mc_xchange

Declaration:
subroutine mc_xchange(imol1, imol2, mol_table, accepted)

Description:
Monte Carlo exchange step

Parameters:
imol1,imol2 - numers of the molecules to exchange
mol_table - MoleculeTable
accepted - output: whether or not the exchange was accepted

subroutine check_distances

Declaration:
subroutine check_distances(xx,yy,zz,xxn,yyn,zzn,natom)

Description:
DEBUG ONLY: checks that the distances between the atoms in two arrays are the same

Parameters:
xx,yy,zz,xxn,yyn,zzn - coordinates of the atoms
natom - number of atoms

subroutine check_consistency

Declaration:
subroutine check_consistency(mol_tab)

Description:
DEBUG ONLY: cheks whether the distances between the atoms in the molecules stored in mol_tab % atomic_data are the same as the distances in the input files

Parameters:
mol_tab - MoleculeTable

MCLuc.f90 : Module MCLuc

Module which includes the procedures written by Luc Belloni for the MC for water. Is used for the test purposes only (to check that my results are the same as Luc's).

Type MCLucInput

The input parameters for the Luc procedures
Fields:
BoxLength - Length of the box
sigma_a - sigma in angtroems
xlj - 4epsilon (in kT)
charg_h - charge of the hydrogen
roh_a,theta_d - distance OH and the angle HOH in water
kmax - maximal value for |k|
rmax2 - maximal value for r^2
alpha - alpha*L
ssr,ssk - sr, sk
temp - temperature
cdiel_ext - dielectrical permutivity
dbjr_a - Bjerum length in Angstroems
dr_a - maximal shift in angstroems
d_angle - maximal rotation in rad
xlambda_f,xlambda_c - constants for force and torq bias

subroutine MCLuc_init_input

Declaration:
subroutine MCLuc_init_input(input)

Description:
initialize the MCLucInput structure with the standard values

Parameters:
input - output: MCLucInput structure

subroutine initmc_Luc

Declaration:
subroutine initmc_Luc(input,xx,yy,zz,n_h2o,systematic_correction)

Description:
run Luc's initmc function

Parameters:
input - MCLucInput structure
xx,yy,zz - atom coordinates
n_h2o - number of h2o molecules
systematic_correction - use or not the Luc's systematic correction (I don't have it so for the comparison it should be .FALSE.)

subroutine rot_vect

Declaration:
subroutine rot_vect(x,y,z,x1,y1,z1,cc,ss,i)

Description:
fait la rotation de x,y,z en x1,y1,z1 d'angle cc=cos,ss=sin autour d'un (i) des 3 axes principaux
attention: x1,y1,z1 peut etre a la meme place memoire que x,y,z
luc80p176

subroutine erfc_Luc_bet6_bet12

Declaration:
subroutine erfc_Luc_bet6_bet12(x,x2,e2,erfk,bet6,bet12)

Description:
x, x2 --> b, b^2
e2 --> exp(-b^2)
erfk --> erfc(b)

b^2 = pi^2 h_m^2 / alpha --> h_m = sqrt(alpha) b/pi
avec erfc_Luc a x<0.5, DL a x>8, integration numerique de x a 8 entre les 2
luc84p150

function erfc_Luc

Declaration:
function erfc_Luc(X,x2,ee2)

Description:
calcule erfc_Luc(x)=2/racine(pi) integrale de x à infini de exp(-t**2)dt
x2=x**2 et ee2=exp(-x**2)
luc85p108

function erf_Luc

Declaration:
function erf_Luc(X,x2,ee2)

Description:
calcule erf_Luc(x)=2/racine(pi) integrale de 0 à x de exp(-t**2)dt
x2=x**2 et ee2=exp(-x**2)
luc85p108

subroutine movemc_Luc

Declaration:
subroutine movemc_Luc(xx,yy,zz,n_h2o, i_h2o,iaccep)

Description:
Luc's movemc function

Parameters:
xx,yy,zz - coordinates of atoms
n_h2o - number of h2o molecules
i_h2o - number of molecule to move
iaccep - whether the movement is accepted or not

subroutine prod_vect

Declaration:
subroutine prod_vect(x1,y1,z1,x2,y2,z2,x,y,z)

Description:

fait le produit vectoriel r = r1 * r2
luc84 page 160
le resultat peut-etre mis a la place du 1er vecteur!

function shi_sans_exp_f

Declaration:
function shi_sans_exp_f(x)

Description:
calcul auto de Shi(x)
luc84p163
en fait, on ne veut pas de facteur exp(x) qui peut occasionner un overflow
donc donner plutot Shi(x)/exp(x)

function alea

Declaration:
function alea(n)

Description:
random number
Parmaters:
n - code. 0=initialize, 1-give the next random

subroutine piston

Declaration:
subroutine piston(iaccep,press_a3,dlnvmax,xx,yy,zz,n_h2o, systematic_correction)

Description:
Luc's volume step

Parameters:
iaccep - whether accepted or not
press_a3 - pressure in kT/A^3
dlnvmax - maximal value for ln V
xx,yy,zz - coordinates. re-scale if needed
n_h2o - number of h2o molecules
systematic_correction - do the systematic correction or not

subroutine write_energy_LUC

Declaration:

subroutine write_energy_LUC(n_h2o)

Description:
Write the energy calculated by Luc

Parameters:
n_h2o - - number of h2o molecules

subroutine rot_vect_OLD

Declaration:
subroutine rot_vect_OLD(x,y,z,x1,y1,z1,cc,ss,i)

Description:
fait la rotation de x,y,z en x1,y1,z1 d'angle cc=cos,ss=sin autour d'un (i) des 3 axes principaux
attention: x1,y1,z1 peut etre a la meme place memoire que x,y,z
luc80p176

subroutine erfc_Luc_bet6_bet12_OLD

Declaration:
subroutine erfc_Luc_bet6_bet12_OLD(x,x2,e2,erfk,bet6,bet12)

Description:
x, x2 --> b, b^2
e2 --> exp(-b^2)
erfk --> erfc(b)
b^2 = pi^2 h_m^2 / alpha --> h_m = sqrt(alpha) b/pi

mc_main.f90 : program runmc_main

Main Program: runmc
The program uses the input files which describe the system to produce the trajectories of the particles.

Usage:
runmc parameters.prm system.composition input.moltab [extra_parameters]

Arguments:
parameters.prm - - file, which include main parameters of the simulation
system.composition - - file, which describes the numbers of molecules of each kind in the system
input.moltab - - initial configuration of the system
extra_parameters - - coma separated list of pair parameter1=value1,parameter2=value2

subroutine write_energy_ME

Declaration:
subroutine write_energy_ME

Description:
write the energy components (DEBUG ONLY)

subroutine parse_command_line

Declaration:
subroutine parse_command_line

Description:
read command line arguments

subroutine read_input_files

Declaration:
subroutine read_input_files

Description:
read input files

subroutine deallocate_all

Declaration:
subroutine deallocate_all

Description:
deallocate everything

mc_make_round_holes.f90 : program mc_make_round_holes

create spherical holes in the given box with molecules
Usage:
mc_make_round_holes system.composition input.moltab BoxLength holes.txt output.moltab [nbytes_xyz nbytes_ang]

Arguments:
input.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi)
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3
holes.txt - - first line - number of holes, next lines: x y z R
output.moltab - - binary file where the molecules which intersect with holes
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3)

subroutine read_holes_file

Declaration:
subroutine read_holes_file

Description:
read the file with holes
first line - number of holes
next lines - x y z R (in angstroems)

subroutine fill_remove_list

Declaration:
subroutine fill_remove_list

Description:
find indeces of molecules to be removed

mc_mean_force.f90 : program mc_mean_force

calculate the mean forces for specific molecules
mc_mean_force: calculate the mean force projection between the molecules'

Usage:
mc_mean_force parameters.prm system.composition forces.ftraj rangeA rangeB maxR dr output.dat

Arguments:
system.composition - - number and types of molecules in the system'
extra_parameters - - coma separated prm1=val1,prm2=val2,... NO SPACES ALLOWED! '
forces.ftraj - - forces trajectory file. In float (24+8) format. See FloatingPoint.f90 for details'
rangeA,rangeB - - ranges for the 1st and 2nd molecules'
in format num1[-num2]'
the MeanForce will be calculated for all pairs A-B where A is in range A, B is in rangeB'
maxR, - dr - samples for MeanForce will be [0:dr:maxR]'
output.dat - - four columns: r sum(f12) N(r) sum(f12)/N(r) where N(r) is number of AB pairs at distance r'

subroutine calc_mean_force

Declaration:
subroutine calc_mean_force

Description:
calculate the mean force at the given frame

subroutine save_results

Declaration:
subroutine save_results

Description:
write the results to file

subroutine parse_command_line

Declaration:
subroutine parse_command_line

Description:
read the command line arguments

subroutine read_range

Declaration:
subroutine read_range(str,first,last)

Description:
extract the first and last ranges from the string of format first-last

subroutine read_input_files

Declaration:
subroutine read_input_files

Description:
read input files

subroutine load_frame

Declaration:
subroutine load_frame

Description:
load next frame from file

subroutine read_forces

Declaration:
subroutine read_forces(hforce,fx,fy,fz,MaxAtom)

Description:
read the forces from the forces trajectory file

Parameters:
hforce - file handler
fx,fy,fz - output: forces for each atom
MaxAtom - number of atoms

subroutine fill_atom_indeces

Declaration:
subroutine fill_atom_indeces(comp,molnum_by_atomnum,first_atom,last_atom,atom_mass)

Description:
fill molnum_by_atomnum, first_atom, last_atom, atom_mass arrays, using the composition

Parameters:
comp - composition (input)
molnum_by_atomnum - molecule index by atom index (output)
first_atom,last_atom - first and last atom indeces for each molecule (output)
atom_mass - mass of each atom (output)

subroutine calc_mol_forces

Declaration:
subroutine calc_mol_forces(comp,fx,fy,fz,fxmol,fymol,fzmol)

Description:
calculate the forces acting on each molecule

Parameters:
comp - composition
fx,fy,fz - atomic forces
fxmol,fymol,fzmol - molecular forces (output)

subroutine deallocate_all

Declaration:
subroutine deallocate_all

Description:
deallocate everything

mcrdf.f90 : program mcrdf

calculate the Radial distribution functions between the atoms
Usage:
mcrdf parameters.prm composition output_prefix [dr Rmax [mol_labels [nskip[-maxfram]]]] '

-
parameters.prm - - parameters of the simulation. in format prm = val at each line '
they should have AT LEAST such fields: '
frames_file = ...'
traj_file = ... '
output_nbytes_xyz = ... '
output_nbytes_arg = ... '
composition - - number and types of molecules in the system'
traj - - trajectory binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
frames.dat - - information about the boxlength at each frame in traj '
output_prefix - - prefix for output files'
output_files are: output_prefixN1_N2.dat, where N1_N2 are numbers or labels of species'
ouput files are text three-coulomn files. '
The columns are : r count(r) g(r) cnt2(r) g2(r) cnt3(r) g3(r) cnt4(r)'
where:'
count(r) = number of particles found in [r;r+dr] '
g(r) = count(r) / N_total * Vmax / dV '
(N_total - total number of distances counted)'
'
cnt2(r) = sum_frame sum_ij 1/r_ij^2 where r_ij in [r;r+dr] '
g2(r) = 1/(4pi rho^2 dr N_frames) 1/<V> cnt2(r) '
'
cnt3(r) = sum_frame 1/V_frame sum_ij 1/r_ij '
g3(r) = 1/(4pi rho^2 dr N_framess) cnt3(r)'
NOTE !!! ONLY cnt4 (8th column) corresponds to the usual way of collecting the data --> only 8th column should be used for any comparisons '
'
dr,Rmax - - bin size and size for Rdf in angstroems. Default dr=0.1 Rmax=12. '
Note - : Nomrally Rmax should be less than min(BoxLength)/2 '
mol_labels - - optional coma separated labels used to produce the output files. If no labels given, numbers are used'
nskip - - number of frames to skip before the start of counting g(r) '
maxfram - - the number of frame to stop the accumulation

subroutine count_distances

Declaration:
subroutine count_distances

Description:

accumulate the total counts at each distance

subroutine count_volumes

Declaration:
subroutine count_volumes

Description:
calculate the mean volume and mean inverse volume

subroutine write_averages

Declaration:
subroutine write_averages

Description:
write average values

subroutine save_rdfs

Declaration:
subroutine save_rdfs

Description:
save rdfs to file

subroutine parse_command_line

Declaration:
subroutine parse_command_line

Description:
read command line arguments

subroutine read_input_files

Declaration:
subroutine read_input_files

Description:
read the input files

mc_rmsd.f90 : program mc_rmsd

calculate the displacement from the original position for a given set of molecules
Usage:
mc_rmsd system.composition traj.moltraj frames.dat interval > output.dat'

Arguments:
interval - - in format num1-num2, where num1 and num2 are the numbers of the first and the last molecules for rmsd '
output.dat - - text file with columns (one per molecule), displacement from the initial position'

module_periodic_table.f90 : Module periodic_table

this module contains the masses for the elements in periodic table
actually, it is not used, since the mass now is given in mol file

Type element

Fields:
symbol -
name -
number -
amass - average mass in formula units
mass - mass of most abundant isotope in formula units
covalent_radius - in Angstroms
vdw_radius - in Angstroms
e_conv - (0:3) ?
heat_of_formation - in kcal/mol
eht_param - (0:3) in eV
gyrom_ratio - in Mhz/Tesla
gyrom_ratio_isotope - isotope number corresponding with gyrom_ratio
ptable - (0:nelem) periodic table array

function element_by_name

Declaration:
function element_by_name(el)

Description:
return the number of element by its name

Parameters:
el - name of the element
SUBROUTINE init_periodic_table()
initialize the table

Molecule.f90 : Module Molecule

The module contains the datastructure to store the structure of the molecule

Type TMolecule

Fields:
Natoms - number of atoms
x,y,z - coordinates of atoms in angstroems
sigma - sigmas in ansgtroems
epsilon - epsilon kcal/mol
charge - charges of atoms
mass - mass of atoms
hard_core - hard_core diameter of atoms in angstroems
atomnames - names of the atoms
filename - name of the file
hasNames - whether or not the atomnames were initialized

subroutine Molecule_init

Declaration:
subroutine Molecule_init(this)

Description:
set the molecule to "zero" state

Parameters:
this - Molecule

subroutine Molecule_allocate

Declaration:
subroutine Molecule_allocate(this, Natoms)

Description:
allocate the arrays in Molecule structure

Parameters:
this - Molecule
Natoms - number of atoms

subroutine Molecule_deallocate

Declaration:
subroutine Molecule_deallocate(this)

Description:
Deallocate the arrays in Molecule structure

Parameters:
this - Molecule

subroutine Molecule_copy

Declaration:
subroutine Molecule_copy(dest,src)

Description:
Make a copy dest = src. Deallocates and allocates the arrays, if necessary

Parameters:
dest,src - destination and source molecules

subroutine Molecule_read_from_file

Declaration:
subroutine Molecule_read_from_file(this, filename)

Description:
read the molecule from mol file

Parameters:
this - Molecule
filename - name of the mol file

subroutine Molecule_centrate

Declaration:
subroutine Molecule_centrate(this)

Description:
centrate the molecule to the center of mass (i.e. make the coordinates of the center of mass (0,0,0)

Parameters:
this - Molecule

subroutine Molecule_write

Declaration:
subroutine Molecule_write(this,hfile)

Description:
write the molecule to the mol file

MoleculeHandler.f90 : Module MoleculeHandler

the global storage for the molecule structures. Each molecule has it's identifier in MoleculeHandler
each molecule can be accessed using its internal "name" (handler)
these names are available for all parts of the program

subroutine MoleculeHandler_init

Declaration:
subroutine MoleculeHandler_init

Description:
Initialize the handler array

subroutine MoleculeHandler_getMolecule

Declaration:
subroutine MoleculeHandler_getMolecule(h,mol_ptr)

Description:
get the pointer to the molecule by its handler

Parameters:
h - molecule handler
mol_ptr - output: pointer to the molecule

subroutine MoleculeHandler_occupy

Declaration:
subroutine MoleculeHandler_occupy(h,mol_ptr)

Description:
try to occupy the handler, associates mol_ptr with the handler h

Parameters:
h - molecule handler

mol_ptr - output: pointer to the handler h (you should allocate it after occupation)

subroutine MoleculeHandler_release

Declaration:
subroutine MoleculeHandler_release(h)

Description:
release the handler h

Parameters:
h - molecule handler

subroutine MoleculeHandler_clear_all

Declaration:
subroutine MoleculeHandler_clear_all

Description:
release and deallocates all molecules
integer

function MoleculeHandler_getFreeHandler

Declaration:
function MoleculeHandler_getFreeHandler()

Description:
get the free handler

MoleculeTable.f90 : Module MoleculeTable

Molecule table contains the coordinates and orientations of the molecules and the indeces of molecule type by atom types and first and last atom indeces in each molecule

Type TMoleculeTable

Fields
mol_type - molecule types
first_atom,last_atom - first atom indeces of the molecules
x,y,z,theta,phi,psi - theta phi psi: 3 rotatiions, psi over Oz, theta over Ox, phi over Oz
nalloc - allocated size

nmol - number of molecules
MaxMolAtom - number of atoms in the largest molecule

Type TMolTypeSpool

limits - array of n_type elements
limits for the mol_types
at given position, the molecule type is taken randomly
but the probability to take each molecule should be proportional to
the number of molecule of that type left in the spool
so, we take an interval of length n_left, and divide it to the sub-intervals
which are equal to the numbers of molecules left
then we take a random number between 1 and n_left, and check, in which interval it is

limits is an array of n_types numbers, containing the upper boundaries of the intervals
n_left - number of molecules left in the spool
n_types - number of molecule types

subroutine MolTypeSpool_alloc

Declaration:
subroutine MolTypeSpool_alloc(this, mol_numbers , n_types)

Description:
allocate the MolTypeSpool

Parameters:
n_types - number of types

subroutine MolTypeSpool_dealloc

Declaration:
subroutine MolTypeSpool_dealloc(this)

Description:
deallocate the MolTypeSpool
integer

function MolTypeSpool_peekMolecule

Declaration:
function MolTypeSpool_peekMolecule(this)

Description:
Peak the molecule (by chance)

Parameters:
this - MolTypeSpool

subroutine MoleculeTable_nulify

Declaration:
subroutine MoleculeTable_nulify(this)

Description:
Set molecule table to the zero state

Parameters:
this - MoleculeTable

subroutine MoleculeTable_alloc

Declaration:
subroutine MoleculeTable_alloc(this, nalloc)

Description:
Allocate the molecule table arrays

Parameters:
this - MoleculeTable
nalloc - size of arrays

subroutine MoleculeTable_copy

Declaration:
subroutine MoleculeTable_copy(dst ,src)

Description:
copy the molecule table
Prameters:
dst,src - destination and source tables

subroutine MoleculeTable_save_binary

Declaration:
subroutine MoleculeTable_save_binary(this,fid,nbytes_xyz,nbytes_ang)

Description:
save the MoleculeTable in binary format (see trajectory file)

Parameters:
this - MoleculeTable
fid - file handler
nbytes_xyz - number of bytes per coordinate sampe (fixed point). allowed values: 1,2,3
nbytes_ang - number of bytes per angular sample. allowed values: 1,2,3

subroutine MoleculeTable_save_text

Declaration:
subroutine MoleculeTable_save_text(this,fid,BoxLength)

Description:
Save the molecule table in the text format
Parameters
this - MoleculeTable
fid - file handler
BoxLength - BoxLength

subroutine MoleculeTable_load_binary

Declaration:
subroutine MoleculeTable_load_binary(this,fid,comp, nbytes_xyz,nbytes_ang)

Description:
load molecule table from the binary format

Parameters:
this - MoleculeTable
fid - file handler
comp - composition
nbytes_xyz - number of bytes per coordinate sampe (fixed point). allowed values: 1,2,3
nbytes_ang - number of bytes per angular sample. allowed values: 1,2,3

subroutine MoleculeTable_load_text

Declaration:
subroutine MoleculeTable_load_text(this,fid,comp,BoxLength)

Description:
Load the MoleculeTable from the text file

Parameters:
this - MoleculeTable
fid - file handler

comp - composition
BoxLength - BoxLength

subroutine MoleculeTable_dealloc

Declaration:
subroutine MoleculeTable_dealloc(this)

Description:
deallocate the molecule table
Parmeters:
this - MoleculeTable

subroutine MoleculeTable_placeCube

Declaration:
subroutine MoleculeTable_placeCube(this,x_shft,y_shft,z_shft, i, m)

Description:
places m particles in a 1x1x1 cube
Parameters
this - MoleculeTable
x_shft,y_shft,z_shft - dispacement of the cube
i - Total particle counter. Updated in the subroutine .
m - m^3 particles have to be placed

subroutine MoleculeTable_placeMoleculesToGrid

Declaration:
subroutine MoleculeTable_placeMoleculesToGrid(this, mol_types, mol_numbers, n_types)

Description:
place molecules into the MoleculeTable

Parameters:
this - MoleculeTable
mol_types - types of the molecules. Array of n_types elements
(actually, these should be their handlers, loaded to memory
this function does not use this, but to convrt them to atom coors it is necessary)
mol_numbers - array of n_types elements. How many molecules of each type will be placed into the box
n_types - number of mol_types and mol_numbers

subroutine MoleculeTable_fillAtomicData

Declaration:
subroutine MoleculeTable_fillAtomicData(this, atomic_data, BoxLength, kT_kcal_mol)

Description:
convert the molecule coordinates in molecule table to the atom coordinates in the AtomicData
atomicData should be pre-allocated
atomnames are optional and filled only if fillAtomNames=.TRUE. AND corresponding molecules have atomnames
this - MoleculeTable
atomic_data - AtomicData (output)
BoxLength - can be taken from parameters. But if not - it is parameters independent
kT_kcal_mol - kt in kcal/mol. sometimes can be set to 1, if for example, only coordinates matter and epsilon is irrelevant

subroutine MoleculeTable_calcPositionsOrientations

Declaration:
subroutine MoleculeTable_calcPositionsOrientations(this, xx, yy, zz, BoxLength)

Description:
Extract the positions and the orientations of the molecules from the coordinates of the atoms

Parameters:
this - MoleculeTable
xx,yy,zz - coordinates of the atoms
BoxLength - box length

moltab2xyz.f90 : program moltab2xyz

convert the binary file with molecular positions to the xyz format
Usage:
moltab2xyz system.composition input.moltab output.xyz [BoxLength [nbytes_xyz nbytes_ang]]'

Arguments:
input.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3 '

moltab_bin2text.f90 : program moltab_bin2text

Convert the binary molecular coordinates to the text form
Usage:
moltab_bin2text system.composition input.moltab BoxLength output.moltext [nbytes_xyz nbytes_ang] '

Arguments:
input.moltab - - in binary format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3 '

output.moltext - - text file where of the same format (x,y,z,theta,phi,psi)
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '

moltab_text2bin.f90 : program moltab_text2bin

Convert the text file x y z theta phi psi to the binary moltab format
Usage:
moltab_text2bin system.composition input.moltext BoxLength output.moltab [nbytes_xyz nbytes_ang] '

Arguments:
input.moltext - - in text format, (fixed point): Nmolecule records: (x,y,z,theta,phi,psi) '
BoxLength - - in Angstroems. If not given - is recalculated to 33.3 particles/nm^3 '
output.molbin - - bin file where of the same format'
nbytes_xyz - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '
nbytes_ang - - bytes per fixed point for coordinats (default: 2, allowed: 1,2,3) '

MonteCarloMove.f90 : Module MonteCarloMove

functions for choosing of new position and calculation of the
Molecule-independent functions!
With minimal changes taken from MC code for H2O by Prof. Luc Belloni
TMonteCarloMove -
Fields:
fxtotold,fytotold,fztotold,ftotold - old forces
couplxold,couplyold,couplzold,couplold - old torques
umaxold - lambda_f * F_tot * dr
vmaxold - lambda_c * tau_tot * d_angle
cold,qold -
vx,vy,vz -
deltx,delty,deltz - displacement
angle -
fxtotnew,fytotnew,fztotnew,ftotnew - new forces
couplxnew,couplynew,couplznew,couplnew - new torq
umaxnew,vmaxnew -
cnew,qnew -

subroutine MonteCarloMove_init_old

Declaration:
subroutine MonteCarloMove_init_old(this, fx_old,fy_old,fz_old,torq_x_old,torq_y_old,torq_z_old)

Description:
initialize the move structure with old forces and torques

Parameters:
this - MonteCarloMove structure
fx_old,fy_old,fz_old - old force (of the molecue)

torq_x_old,torq_y_old,torq_z_old - old torq (molecular)

subroutine MonteCarloMove_init_new

Declaration:
subroutine MonteCarloMove_init_new(this, fx_new,fy_new,fz_new,torq_x_new,torq_y_new,torq_z_new)

Description:
Set new forces and torques

Parameters:
this - MonteCarloMove
fx_new, - fy_new, fz_new new force
torq_x_new, - torq_y_new, torq_z_new new torq
determine the displacement
Taken from MC code for H2O by Luc Belloni, with minimal changes
is universal for any molecules
deplacement dans une sphere de rayon dr avec proba exp(lamda.F.deltar)/Cold
luc70p194 et luc84p160

Parameters:
this - MonteCarloMove
deltx,delty,deltz - result

subroutine MonteCarloMove_chooseRotation

Declaration:
subroutine MonteCarloMove_chooseRotation(this,rot)

Description:
determine the rotation
Taken from MC code for H2O by Luc Belloni, with minimal changes
is universal for any molecules

Parameters:
this - MonteCarloMove
rot - rotation (in Luc's format, see geometry.f90)

subroutine MonteCarloMove_accept_or_decline

Declaration:
subroutine MonteCarloMove_accept_or_decline(this, dutot, accepted)

Description:
accept or decline the move

Parameters:

this - MonteCarloMove
dutot - total change of the energy
accepted - output: accepted or declined

subroutine MonteCarloMove_accept_or_decline_simple

Declaration:
subroutine MonteCarloMove_accept_or_decline_simple(dutot, accepted)

Description:
without the force bias (used in exchange move)

Parameters:
dutot - total change of energy
accepted - output: accepted or declined

parameters.f90 : Module Parameters

parameters of the simulation
everything which is independent of anything is in constants
everything else is here (e.g. which depends on kT, box Length etc.)
Parameters_initialized - indicates that the parameters was initialized. Consider also Parameters_areConsistent(AtomicData)
Parameters_areRead - indicates that the parameters are read from the parameters file
sr - rmax = sr / alpha
err_r = Q (sr/ alpha L^3)^(1/2) exp(-sr^2) / sr^2
typical value sr = 3..4
sk - kmax = s(L*alpha) / pi
err_k = Q (sk/2 alpha L^3)^(1/2) exp(-sk^2) / sk^2
typical value sk = 3..4
alpha - alpha * L , by default 2*sr
rmax = sr / alpha < L/2 ==> alpha*L > 2*sr
alpha*L = alpha [L^-1] = 2*sr corresponds to the cutoff L/2 (default, if alpha=0)
temp - temperature, K
external_permutivity - epsilon_ext
BoxLength - if box_length = 0 --> to be recalculated from density
density - /particles/nm^3
dr_a - max. dispacement in Angstroems
d_angle_degree - max rotation
xlambda_c - coupling constant for torque (note: normally xlambda_f = xlambda_c AND xlambda_f = 0.5)
xlambda_f - coupling constant for force
rnd_seed - random seed
pressure - in Pascals, i.e. J/m^3
pressure_step_multiplier - make NPT step at average each nmol * pressure_step_multiplier steps
set 0 for NVT simulation
max_volume_scaling - vnew = vold * max_volume_scaling ** lambda, -1 < lambda < 1
max_cycle - number of cycles to do
n_store_traj_interval - save trajectory frames each nmol * n_store_traj_interval steps
(each n_store_traj_interval_cycles)

n_store_energy_interval - save energy interval
input_nbytes_xyz - , input_nbytes_ang = 2 number of bytes to store values in input moltab file
output_nbytes_xyz - , output_nbytes_ang = 2 number of bytes to store values in the trajectory files
traj_file - 'traj.moltab' traj output file
energy_file - 'energy.dat' energy output file
boxlength_file - 'boxlength.dat' boxlength output file
frames_file - 'frames.dat' frames output file
param_out_file - 'parameters.out' parameters output file
freq_file - 'none' frequences input file
rmax - rmax = sr / alpha
kmax - kmax = sk*(alpha L) / pi
kT_kcal_mol - boltz * temp
rmax2 - rmax^2
dbjr_a - bjerum length in Angtroem: elec**2/(4.d0*pi*epsi0*boltz*temp)/1.d-10
dbjr - dbjr/BoxLength
xclb - coulomb potential prefactor. Just more clear name.
alpha_over_sqrt_pi - alpha/sqrt(pi), use in EwaldSumTails
two_alpha_over_sqrt_pi - 2alpha/sqrt(pi), use in real coulomb forces
kext - for external sum: 2 pi / (2 external_permutivity + 1)
minus_two_kext - - 2 * kext = -4pi/(2 external_permutivity + 1), used in external sum forces
dr - maximal displacement = dr_a / BoxLength
d_angle - maximal rotation in radians
log_max_volume_scaling - log(max_volume_scaling), used in mcvol
pressure_angstr_kT - pressure in kT/A^3

subroutine read_parameter

Declaration:
subroutine read_parameter(nam,val)

Description:
initialize the parameter using the text pair nam and val

Parameters:
nam,val - nam and val strings read from the input file as nam=val

subroutine parameters_write

Declaration:
subroutine parameters_write(h)

Description:
write the parameters in the text format to the file

Parameters:
h - file handler

subroutine Parameters_read_string

Declaration:
subroutine Parameters_read_string(str)

Description:
read the parameter nam=val pair from the string

Parameters:
str - input string

subroutine Parameters_read

Declaration:
subroutine Parameters_read(fname)

Description:
read parameters fule

Parameters:
fname - file name
pure

function BoxLength_from_density

Declaration:
function BoxLength_from_density(density, nmol)

Description:
calculate the box length from the density

Parameters:
density - in particles/nm^3
nmol - number of molecules
Return value:
BoxLength

subroutine Parameters_init

Declaration:
subroutine Parameters_init(filename,nmol,extra_parameters_string)

Description:
initialize the parameters of simulation

Parameters:
filename - name of the parameters file
nmol - number of molecules
extra_parameters_string - extra parameters in format nam1=val1,nam2=val2,...

subroutine Parameters_recalc

Declaration:
subroutine Parameters_recalc(box_length)

Description:
recalculate the parameters for the new boxlength,temp etc

Parameters:
box_length - if BoxLength = 0 --> recalculate from density and Nmol

random.f90 : Module MRandom

functions to work with random numbers random number

subroutine randomize

Declaration:
subroutine randomize(seed)

Description:
initialize the random number generator with seed

subroutine set_rand_epsilon

Declaration:
subroutine set_rand_epsilon(eps)

Description:
set the minimal random value. The randoms will be generated in [eps;1-eps] (to avoid zero results which sometimes cause errors)

Parameters:
eps - epsilon

function rand

Declaration:

function rand()

Description:
get the random value from (0;1)
integer

function random

Declaration:
function random(N)

Description:
get the integer random value from 0 to N-1

RealSumLocal.f90 : Module RealSumLocal

auxilarly functions for particle-particle interactions
to be used inside the Real Space Ewald Sums

Type TRealSumLocal

auxilarly variables
r,r2 - distance
a,a2,a6 - a = alpha * r, a2 = a^2, a6= a^6
exp_minus_a2 - exp(-a^2)
erfc_a - erfc(a) = erfc(alpha r)
C6,C12 - C_w = sum_{l=0}^{w/2-1} l^{2l} / l! where w=6,12
used in LJ energy and forces
four_epsilon - 4eps
sigma_over_r_6 - (sigma/r)^6
potew - electrostatic interaction potential q_iq_j * erfc(alpha r) / r
potlj12,potlj6 - Lennard Jones potential compontnts: U_w = A_ij exp(-alpha r)/r^w C_w, w=6,12
pot - total potential potew + potlj12 - potlj6
FR - Force radial component
F_p = SUM_ij F_R(i,j) (r_i - r_j)
where FR = FR_C + FR_LJ (definitions see above, explanations in ewald.pdf)

subroutine RealSumLocal_init_electrostatics

Declaration:
subroutine RealSumLocal_init_electrostatics(this, r2, sameMolecule)

Description:
init variables used for electrostatic energy and forces calculation

Parameters:
this - RelSumLocal
r2 - r^2
sameMolecule - indicates that the particles are in the same molecule
for compatibility with Luc Belloni code

subroutine RealSumLocal_calc_electrostatics

Declaration:
subroutine RealSumLocal_calc_electrostatics(this, qi, qj)

Description:
calculate the electrostatic interactions
run init_electrostatics first

Parameters:
this - RealSumLocal
qi, - qj charges

subroutine RealSumLocal_init_LJ

Declaration:
subroutine RealSumLocal_init_LJ(this,sigma6)

Description:
note: LJ calculations should always be performed after the coulomb
at least: you should run RealSumLocal_init_coulomb before

Parameters:
this - RealSumLocal
sigma6 - sigma^6

subroutine RealSumLocal_calc_LJ

Declaration:
subroutine RealSumLocal_calc_LJ(this,four_epsilon)

Description:
Calculate LJ potentials and update FR (add LJ force radial component)

Parameters:
this - RealSumLocal
four_epsilon - 4eps

RhoSquared.f90 : Module RhoSquared

the module contains the coordinate-dependent sums of sin and cos for the Ewald sumation in KSpace
TRhoSquared -
Fields:
lj_types - LJTypes
grid - KSpace grid
sumsincos_coulomb - SUM_i sin(kR_i), SUM_i cos(kR_i)
rho_squared - rho^2 = rho(+++)^2 + sgn(ky)rho(+-+)^2 + sgn(kz)rho(++-)^2 + sgn(kykz)rho(+--)^2
sumsincos_LJ - SUM_i sin(kR_i) for each type of atoms
type_is_present - to accelerate calc F2: if not type_is_present --> no summation needed
sincos_xyz - temporary array containing the sin(kR) cos(kR)

subroutine RhoSquared_alloc

Declaration:
subroutine RhoSquared_alloc(this,grid,lj_types)

Description:
Allocate the RhoSquare arrays

Parameters:
this - RhoSquare structure
grid - KSpace grid
lj_types - LJTypes

subroutine RhoSquared_dealloc

Declaration:
subroutine RhoSquared_dealloc(this)

Description:
deallocate the RhoSquared structure

Parameters:
this - RhoSquared structure

subroutine RhoSquared_nulify

Declaration:
subroutine RhoSquared_nulify(this)

Description:
Set RhoSquared to the "zero" state

Parameters:

this - RhoSquared

subroutine RhoSquared_addAtom

Declaration:
subroutine RhoSquared_addAtom(this,xx,yy,zz,charge,lj_type,sincos_kr_coulomb,sincos_kr_LJ)

Description:
Add new atom to the sums

Parameters:
this - RhoSquared structure
xx,yy,zz,charge - coordinates and charge
lj_type - type of atom
sincos_kr_coulomb, - sincos_kr_LJ output: sin(kRi),cos(kri) for coulomb and LJ case

subroutine RhoSquared_addAtoms

Declaration:
subroutine RhoSquared_addAtoms(this,xx,yy,zz,charge,type_by_index,natom)

Description:
Add many atoms

Parameters:
this - RhoSquared
xx,yy,zz,charge - coordinates and charges
type_by_index - type by atom number
natom - number of atoms

subroutine RhoSquared_calc_rho_squared

Declaration:
subroutine RhoSquared_calc_rho_squared(this)

Description:
calculate rho^2
note: this % sumsincos_coulomb should be calculated before (see calc_sumsincos_coulomb)
not: this % rho_squared should be allocated before
rho^2 = rho(+++)^2 + sgn(ky) rho(+-+)^2 + sgn(kz) rho(++-)^2 + sgn(kykz) rho(+--)^2

Parameters:
this - RhoSquared structure

subroutine RhoSquared_copy

Declaration:
subroutine RhoSquared_copy(dst, src)

Description:
Copy data in the RhoSquared structure

Parameters:
dst,src - destinatioion, source

subroutine RhoSquared_add

Declaration:
subroutine RhoSquared_add(this, rho_squared)

Description:
Add two RhoSquared structures, i.e. this=this+rho_squared

Parameters:
this, - rho_squared summands

subroutine RhoSquared_sub

Declaration:
subroutine RhoSquared_sub(this, rho_squared)

Description:
substract: this=this-rho_squared \

Parameters:
this, - rho_squared operands

subroutine RhoSquared_save_to_file

Declaration:
subroutine RhoSquared_save_to_file(this,filename)

Description:
Save to file
For debug only
format columns:
coulomb cos_ppp,cos_pmp,... sin_ppp,sin_pmp,...
LJ1 cos_ppp,cos_pmp,... sin_ppp,sin_pmp,...
LJ2 com_ppp,cos_pmp,...

...

Parameters:
this - RhoSquared structure
filename - name of the fle

runmc.f90 : Module runmc

Module which performs the MC cycles. Is called from the mc_main.f90

subroutine init_freq

Declaration:
subroutine init_freq(nmol)

Description:
initialize the frequences (read them from file or just make flat if no file given)

Parameters:
nmol - number of molecules

subroutine read_freq_file

Declaration:
subroutine read_freq_file(fname, mv_num, xchange_num, nmol)

Description:
read frequences from file

Frequences file format:

first non-comment line - number of intervals

next lines:

column 1: interval in format first-last

column 2: relative probability to choose the molecule in this interval

column 3: realtive probability to xchange the molecules from this interval

Parameters:
fname - name of the file
mv_num, - xchange_num output: relative probabilities of movement and exchange
nmol - number of molecules

subroutine run_mc

Declaration:
subroutine run_mc(mol_table, ncycle)

Description:
run the MC cycles

Parameters:
mol_table - MoleculeTable
ncycle - max number of cycles

subroutine store_energy

Declaration:
subroutine store_energy(iframe)

Description:
save the energies to the energy file

Parameters:
iframe - current frame

subroutine store_boxlength

Declaration:
subroutine store_boxlength(iframe)

Description:
save current boxlength to the box length file

Parameters:
iframe - current frame number

subroutine store_traj_frame

Declaration:
subroutine store_traj_frame(mol_tab,framecount)

Description:
save current molecule positions to the trajectory file

Parameters:
mol_tab - MoleculeTable
framecount - current frame

subroutine read_last_frame

Declaration:
subroutine read_last_frame(frames_file,nframes)

Description:
read the last frame from the frames.dat file
PArameters:
frames_file - frames file
nframes - output: number of frames

scale_box.f90 : Module ScaleBox

change the size of the box (and thus the atom coordinates)

subroutine scale_box

Declaration:
subroutine scale_box(this, mol_tab, NewBoxLength)

Description:
change the size of the box. The centers of mass remain the same in the relative coordinates, but the bond length change.

Parameters:
this - AtomicData (coordinates)
mol_tab - MoleculeTable
NewBoxLength - new length of the box

string.f90 : Module String

Functions to work with the strings
pure

function str_get_next_pos

Declaration:
function str_get_next_pos(str,offset,ch)

Description:
get the position of the first occurance of the symbol ch in the string str starting from the position offset
if not found then -1 is returned

Parameters:
str - string to search

offset - start position
ch - character
pure

function str_isempty

Declaration:
function str_isempty(str)

Description:
check if the string is empty (contains only spaces)

subroutine str_subs

Declaration:
subroutine str_subs(str, ch_old, ch_new)

Description:
substitute in the string str the symbol ch_old with the symbol ch_new

SumSinCosKR.f90 : Module SumSinCosKR

SUM_i cos(kR_i), SUM_i sin(kR_i). Used in EwaldSums in KSpace
TSinCosXYZ -
temporary arrays for calculation
the reason to create them: they can be once allocated and then re-used
Fields:
nk - number of elements allocated
cos_xkx,cos_yky,cos_zkz - cos
sin_xkx,sin_yky,sin_zkz - sin
TSinCosKR -
sin,cos(sx*x*kx + sy*y*ky + sz*z*kz)
Fields:
grid - KSpace grid
cos_ppp - cos (xkx + yky + zkz) (+++)
sin_ppp - sin (xkx + yky + zkz) (+++)
cos_ppm - cos (xkx + yky - zkz) (++-)
sin_ppm - sin (xkx + yky - zkz) (++-)
cos_pmp - cos (xkx - yky + zkz) (+-+)
sin_pmp - sin (xkx - yky + zkz) (+-+)
cos_pmm - cos (xkx - yky - zkz) (+--)
sin_pmm - sin (xkx - yky - zkz) (+--)
TSumSinCosKR -
SUM sin(kR), SUM cos(kR)
Fields:
grid - KSpace grid
sumcos_ppp - SUM_s cos (xkx + yky + zkz) (+++)

sumsin_ppp - SUM_s sin (xkx + yky + zkz) (+++)
sumcos_ppm - SUM_s cos (xkx + yky - zkz) (++-)
sumsin_ppm - SUM_s sin (xkx + yky - zkz) (++-)
sumcos_pmp - SUM_s cos (xkx - yky + zkz) (+-+)
sumsin_pmp - SUM_s sin (xkx - yky + zkz) (+-+)
sumcos_pmm - SUM_s cos (xkx - yky - zkz) (+--)
sumsin_pmm - SUM_s sin (xkx - yky - zkz) (+--)

subroutine SinCosXYZ_alloc

Declaration:
subroutine SinCosXYZ_alloc(this,kmax)

Description:
allocate SinCosXYZ structure

Parameters:
this - SinCosXYZ
kmax - max |k|

subroutine SinCosXYZ_dealloc

Declaration:
subroutine SinCosXYZ_dealloc(this)

Description:
Deallocate SinCosXYZ

Parameters:
this - SinCosXYZ

subroutine SinCosXYZ_fill

Declaration:
subroutine SinCosXYZ_fill(this,x,y,z,kmax)

Description:
initialize SinCosXYZ with x,y,z and given max |k|

Parameters:
this - SinCosXYZ
x,y,z - x,y,z
kmax - max |k|

subroutine SinCosKR_alloc

Declaration:
subroutine SinCosKR_alloc(this,grid)

Description:
Allocate SinCosKR structure

Parameters:
this - SinCosKR
grid - KSpace grid

subroutine SinCosKR_dealloc

Declaration:
subroutine SinCosKR_dealloc(this)

Description:
Deallocate SinCosKR structure

Parameters:
this - SinCosKR

subroutine SinCosKR_fill

Declaration:
subroutine SinCosKR_fill(this,sincos_xyz,charge)

Description:
fill the sumcos for a given atom i.e. xkx+yky+zkz, xkx +yky - zkz etc...

Parameters:
this - SinCosKR
sincos_xyz - sin,cos(xkx,yky,zkz)
to be filled before with
SinCosXYZ_fill(sincos_xyz,x,y,z, this % grid % kmax)
charge - only for Coulomb sum. Use charge=1 for LJ
set all arrays to zero

Parameters:
this - SinCosKR

subroutine SinCosKR_copy

Declaration:

subroutine SinCosKR_copy(dst, src)

Description:
Make a copy dst = src

Parameters:
dst - destination
src - source

subroutine SinCosKR_mulScalar

Declaration:
subroutine SinCosKR_mulScalar(this, multiplier)

Description:
multiply by scalar: this = this * multiplier

Parameters:
this - SinCosKR
multiplier - scalar multiplier

subroutine SumSinCosKR_alloc

Declaration:
subroutine SumSinCosKR_alloc(this,grid)

Description:
allocate SumSinCosKR structure

Parameters:
this - SumSinCosKR

subroutine SumSinCosKR_dealloc

Declaration:
subroutine SumSinCosKR_dealloc(this)

Description:
Deallocate SumSinCosKR

Parameters:
this - SumSinCosKR

subroutine SumSinCosKR_nulify

Declaration:
subroutine SumSinCosKR_nulify(this)

Description:
Set SumSinCosKR to zero

Parameters:
this - SumSinCosKR

subroutine SumSinCosKR_addAtom

Declaration:
subroutine SumSinCosKR_addAtom(this,sincos_kr)

Description:
add atom

Parameters:
this - SumSinCosKR
sincos_kr - sincoskr for the new atom

subroutine SumSinCosKR_addSum

Declaration:
subroutine SumSinCosKR_addSum(this,sumsincos)

Description:
add sum (many atoms) this = this + sumsincos
Parameyers:
this - SumSinCosKR
sumsincos - another SumSinCosKR for the new atoms

subroutine SumSinCosKR_subSum

Declaration:
subroutine SumSinCosKR_subSum(this,sumsincos)

Description:
substract atoms: this = this - sumsincos

Parameters:
this - this SumSinCosKR
sumsincos - atoms to substract

subroutine SumSinCosKR_copy

Declaration:
subroutine SumSinCosKR_copy(dst, src)

Description:
Copy: dst = src

Parameters:
dst - destination
src - source

subroutine SumSinCosKR_mulScalar

Declaration:
subroutine SumSinCosKR_mulScalar(this, multiplier)

Description:

Parameters:
this - SumSinCosKR
multiplier - scalar multiplier

SystemSettings.f90 : Module SystemSettings

System-dependent constants
SYSTEM_STRING_LENGTH - Standart string length
STDERR - standard error output file
STDIN - standard input file
STDOUT - standard output file
SEEK_SET - relative to the beginging of the file
SEEK_CURR - relative to the current position
SEEK_END - relative to the end of the file

Type TRealArrayPointer

the structure is used to have pointers to the array pointers
ptr - pointer