SAGA 1.0
Program Manual

compiled and edited by

Eric D. Glendening

Department of Chemistry and Physics
Indiana State University, Terre Haute, Indiana 47809

e-mail: glendening@indstate.edu
phone: (812) 237-2235

Table of Contents

A. Introduction

A.1. What is SAGA?

SAGA is a Fortran program that implements a simulated annealing (SA) method and a genetic algorithm (GA) for the global minimization of Lennard-Jones (LJ) clusters. SAGA provides the following functionality:

- calculation of idealized geometries using a variety of packing motifs
- energy evaluation using Lennard-Jones or extended Lennard-Jones potentials
- local minimization by numerical or analytic gradient-based methods
- global minimization by simulated annealing (SA) or a genetic algorithm (GA)
- evaluation of zero-point energies and vibrational frequencies
- analysis of vertical binding energies

SAGA is available, free of charge, as a Windows binary executable or as Fortran-compilable source code for Unix/Linux platforms.

A.2. Program Distribution, Installation, and Execution

SAGA 1.0 can be obtained at no charge at the website http://carbon.indstate.edu/saga. A binary distribution for Windows and a source-code distribution for Unix/Linux are available.

A.3.1. Windows

1. Download the compressed binary distribution from the SAGA website. The distribution includes four files:

- saga.bat – a batch file to run SAGA calculations.
- saga-1.0.exe – the binary SAGA executable file.
- cygwin1.dll – the cygwin dynamic link library.
- tutorial.dat – a sample SAGA input file (see A.4).

2. Create a SAGA folder, double-click the binary distribution to open it, and copy the four files of the distribution into the SAGA folder.

3. Edit the saga.bat batch file. (Right-click saga.bat, and select "Edit.") Set the path to include SAGA folder (e.g. set path=%path%;c:\SAGA). Exit and save saga.bat.

4. Select Run from the Start Menu. Type "command" in the Run window, and click OK. A DOS command window will open. Change to the SAGA directory (e.g. "cd c:\SAGA").

5. Run the sample SAGA calculation by typing "saga tutorial" at the command prompt. The SAGA calculation will write output to the file tutorial.log. See Section A.5 for discussion of the

tutorial.log output.

A.3.2. Unix/Linux

Download the gzipped/tar distribution file from the SAGA website. The distribution includes the full source code for SAGA 1.0. Installation requires a Fortran compiler and GNU make.

Copy the SAGA tar file (saga-1.0.tar.gz) to an installation directory. Untar the file

```
tar -zxvf saga-1.0.tar.gz
```

This creates the saga directory with source (saga/src) and binary (saga/bin) subdirectories.

Modify the Makefile in saga/src. Set the compiler variable to gfortran, g77, or pgf77 depending on the compiler that is available on your system. The SAGA source code is principally Fortran-77 and is largely portable so other compilers may work as well.

Type “make” in the saga/src directory to build the saga-1.0 executable. The executable will be installed in the saga/bin directory. Edit the saga script in the saga root directory as needed.

Ensure that you are in the saga root directory and execute the tutorial calculation:

```
saga tutorial
```

Output from this calculation will be directed to the file tutorial.log in the same directory. See Section A.4 for an overview of the tutorial output.

A.3. Input and Output

The user controls SAGA execution using a text-based data file, $job.dat, where $job is the filename environment variable set by the user. Output is directed to standard output, and geometry information (in XMol) format is written to the files $job.xyz and $job.vbe.

The SAGA distribution includes the following saga script for running SAGA calculations:

```
#!/bin/tcsh
setenv job $1
if (-e $job.log) rm -i $job.log
if (-e $job.log) exit
./bin/saga-1.0.exe >& $job.log &
```

For example, typing “saga ar23” at the Unix command prompt in the saga directory starts the SAGA calculation, directing the program to read input from ar23.dat and to write output to ar23.log. Cluster geometries would be written to ar23.xyz, and geometries associated with the vertical binding energy analysis, if requested, would be written to ar23.vbe.

The corresponding Windows batch file is:

```
@echo off
REM job name
set job=%1
REM set path to SAGA and cygwin dll:
set path=%path%;c:\saga
REM execute SAGA calculation:
if not exist %job%.dat echo Missing %job%.dat input file
if     exist %job%.dat saga-1.0.exe > %job%.log 2>%1
```

A.4. A Tutorial Example

Consider a sample calculation of the Ar_{23} cluster using Lennard-Jones parameters (σ = 3.465 Å; ε = 80.6 cm^{-1}) from gas viscosity measurements. The input for this calculation is a text file (tutorial.dat):

```
$control  natoms=23  runtyp=anneal  zpe  freq  vbe  $end
$potential  label=Ar  type=lj  param=3.465,80.6 $end
```

The first line of this file is the $control group that specifies details of the desired cluster (23 atoms) and calculation type. The calculation performed includes global energy minimization by simulated annealing (runtyp=anneal) with zero-point energy (zpe) and vibrational frequencies (freq) evaluation and vertical binding energy (vbe) analysis.

The second line is the $potential group that specifies the identity of the atoms in the cluster (label=Ar), the type of potential to be employed (type=lj), and the parameters for that potential (param=3.465,80.6).

The SAGA calculation is executed by typing the command "saga tutorial" at the Unix or DOS command prompt, assuming that SAGA is installed as described Section A.3. The calculation writes, to the same directory of the tutorial.dat file, three output files, tutorial.log, tutorial.xyz, and tutorial.vbe. We consider each of these files in turn.

A.5.1. Output File (tutorial.log)

SAGA echoes program options selected in the input file. All options are reported, including default values for options that were not specified by the user. Descriptions of the six input groups ($control, $potential, $opt, $anneal, $genetic, and $xyz) are given in Section B.

```
-------------------------------------------------
  SAGA 1.0 - Simulated Annealing/Genetic Algorithm
-------------------------------------------------

$control input
   natoms = 23            nshell = .false.
   runtyp = anneal        method = bfgs          geometry = xyz
   vbe = 0.01 cm-1        zpe = .true.           freq = 2
   seed = -1325555771

$potential input
   label = Ar             type = lj              mass = 39.948 amu
   param = 3.465,80.6

$opt default parameters
   ethr = 0.0001 cm-1     sthr = 0.0001 A        gthr = 0.003 cm-1/A
   maxit = 200
```

```
$anneal default parameters
   macro = 1000           micro = 100           ethr = 0.0001 cm-1
   tinit = auto           tscale = 0.75 cm-1

No $xyz input; generating initial icosahedral geometry
```

Because no cluster geometry in specified in the input file, the program automatically generates an idealized icosahedral geometry.

SAGA proceeds to optimize the cluster geometry, seeking the global energy minimum using a simulated annealing algorithm (see Section A.5).

```
Geometry optimization by simulated annealing:

 macro  micro         total
  iter   iter   acc    iter       Ecur          Emax          Emin        temp
                                  (cm-1)        (cm-1)        (cm-1)      (cm-1)
-------------------------------------------------------------------------------
    0                          -7117.5087
    1   2300   2025    2300    -7081.5394   -6790.2095   -7483.2645  304.808
    2   2300   1957    4600    -7150.7784   -6722.6442   -7483.2645  228.606
    3   2300   1782    6900    -7175.2796   -6790.1911   -7483.2645  171.455
    4   2300   1677    9200    -7350.5924   -6832.2637   -7483.2645  128.591
    5   2300   1456   11500    -7182.2946   -6857.9036   -7483.2645   96.443
    6   2300   1267   13800    -7223.9116   -6913.3953   -7483.2645   72.332
    7   2300   1081   16100    -7361.0186   -7011.7007   -7483.2645   54.249
    8   2300    759   18400    -7302.0546   -7086.6524   -7483.2645   40.687
    9   2300    347   20700    -7483.2645   -7171.6738   -7483.2645   30.515
   10   2300    215   23000    -7483.2645   -7234.5063   -7483.2645   22.886
   11   2300    121   25300    -7483.2645   -7302.0546   -7483.2645   17.165
   12   2300     99   27600    -7483.2645   -7483.2645   -7483.2645   12.874
-------------------------------------------------------------------------------
  converged

Energy = -7483.264 cm-1
```

The initial icosahedral geometry has an energy of -7117.5.8 cm^{-1} (relative to separated atoms).

The annealing algorithm reports the status of the search at the conclusion of each macro-iteration. For example, the first macro-iteration performed 2300 (100×natoms) local geometry optimizations (micro-iterations) of the cluster with an annealing temperature of 304.8 cm^{-1}. Of the 2300 configurations considered, 2025 were accepted by the Metropolis algorithm. The least stable and most stable of these geometries had respective energies of -6790.2 cm^{-1} and -7483.3 cm^{-1}.

Macro-iterations continue, with annealing temperature scaled by 0.75 after each iteration, until the energies of the least stable and most stable geometries differ by less than less than 0.0001 cm^{-}

[1] (the default convergence threshold, ethr of the $anneal group).

The annealing algorithm performs a total of 27,600 local optimizations before identifying a geometry of energy -7483.3 cm^{-1} as the global minimum.

[Note that the algorithm had already identified at least one geometry of this energy during the first macro-iteration (and in each macro-iteration thereafter). The user can reduce the calculation time by decreasing the number of micro-iterations (see $anneal in Section B.5), which is probably reasonable for a small cluster like Ar_{23}. However, decreasing the number of micro-iterations is problematic as cluster size increases. The number of local minima on the potential energy surface increases exponentially with cluster size, and it becomes increasingly challenging to ensure that the algorithm adequately searches the surface for the global minimum.]

The program next reports the vibrational frequencies ($3N - 6 = 63$ values with degeneracies given in parentheses), zero-point energy (895.4 cm^{-1}), total energy (-6587.8 cm^{-1}), and vertical binding energy analysis for the cluster:

```
Frequencies (cm-1):  10.04(2), 11.17(2), 11.90, 13.52, 14.11(2), 14.85,
 15.81(2), 16.47, 16.84, 17.97(2), 18.63, 18.69(2), 18.85(2), 19.05(2),
 19.32, 22.07(2), 22.64, 22.78, 23.11(2), 23.37, 26.40(2), 26.67(2), 28.00,
 28.51(2), 28.54, 28.84, 29.61, 29.72(2), 30.10(2), 31.56, 32.94, 33.04(3),
 33.95, 34.51(2), 36.79(2), 50.35, 52.96(2), 55.35(2), 57.85, 64.90(2),
 76.25

     E = -7483.264 cm-1
   ZPE =   895.447 cm-1
 E+ZPE = -6587.818 cm-1

Vertical Binding Energies:

 Monomers   Cnt    N    Rav      VBE
                        (A)     (cm-1)
---------------------------------------
   1-3        3   12   3.736    476.64
   4-5        2    9   3.892    405.16
   6-11       6    8   3.939    351.42
  12-17       6    6   3.947    265.50
  18-23       6    6   3.973    256.92
---------------------------------------
 *Total*                       7483.26

53.41 of 1024 megabytes memory used
Total run time = 322.9 sec
```

The vertical binding energy analysis sorts (and renumbers) the atom monomers according to the extent to which each contributes to the total binding energy, the energy required to fully dissociate the cluster. Vertical binding energy for an atom is defined as one-half of the energy change that results when that atom is removed from the cluster without changing the relative

positions of the remaining atoms. The sum of vertical binding energies over all atoms yields the total binding energy.

Three atoms in Ar_{23} (numbered 1-3) have the strongest binding energies (476.6 cm^{-1} each) that principally results from strong interactions with 12 nearest neighbor atoms. These are the three atoms at the center of the Ar_{23} cluster (see below), and the average distance between these atoms and their nearest neighbors is 3.736 Å. The atoms contributing least to the binding energy are six atoms (18-23) on the surface of the cluster, each having only six nearest neighbor atoms.

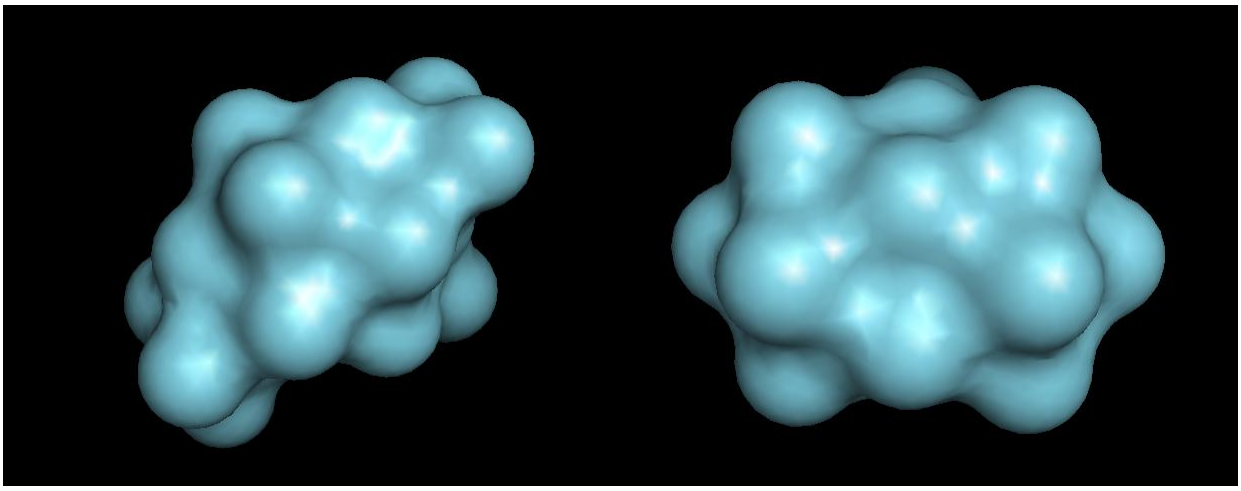
A.5.2. XMol File (tutorial.xyz)

SAGA writes an XMol-formatted text file of cluster geometries considered during geometry optimization. The file consists of a series of concatenated geometries, each expressed in Cartesian coordinates, from the initial geometry to the geometry identified by simulated annealing as the global minimum. The initial (zeroth) geometry is:

```
   23
Geom =     0   Energy =     -7117.50865
Ar      -1.043564   -0.633974    0.771605
Ar       0.488073   -1.563224    4.049204
Ar       2.553409   -1.719658    0.735773
Ar      -2.384783    0.462009   -2.530441
Ar      -3.364728   -3.062098   -0.830749
Ar       1.340239    1.836014    2.178256
Ar       0.237877   -2.553353   -2.185816
Ar      -4.620085    0.392265    0.659413
Ar      -1.730157    3.066157    0.180880
Ar       1.212178    1.183912   -1.470195
Ar      -0.322523   -4.273079    1.309966
Ar      -2.127626    1.334626    3.765406
Ar      -3.298892   -2.411270    3.121210
Ar       1.394190    1.712176    5.908534
Ar       4.124345   -0.230017    3.841653
Ar       4.614885    1.457909    0.202482
Ar       0.508427   -0.021374   -5.015618
Ar      -2.675234   -2.603693   -4.675778
Ar       4.335329    3.575098    3.657027
Ar       3.790273   -1.104701   -2.897985
Ar      -5.864262   -1.002759   -2.841922
Ar      -0.519413    3.595806   -3.736857
Ar       3.352040    2.563230   -4.196048
```

SAGA writes a cluster configuration to the XMol file every micro-iteration that the annealing algorithm identifies a configuration of lower energy (lower than any identified previously). Thus, at the conclusion of the SAGA calculation, the XMol file lists a series of cluster configurations, each of successively lower energy. The last configuration listed is that which likely corresponds to the global minimum.

A number of vizualization packages (including Molden and PyMol) can process the XMol formatted file to display images of the geometries. PyMol was used to prepare the following images of the initial and final geometries of this tutorial calculation:



The image on the left is that of the icosahedral geometry for Ar_{23}, and that on the right is the D_{3h} symmetric geometry of the global minimum.

SAGA writes a geometry to the XMol file each time the simulated annealing (or genetic) algorithm identifies a geometry of lower energy than any previously considered.

A.5.3. VBE File (tutorial.vbe)

A.5. Simulated Annealing Implementation

Simulated annealing (SA) is a global minimization technique that allows a cluster to sample a wide range of configurations at an adjustable simulation "temperature." As this temperature is decreased, the cluster becomes increasing constrained to explore low energy configurations, and, in the limit of very low temperature, converges on a single low energy configuration. If system cooling is performed at a sufficiently slow rate, the SA algorithm is guaranteed to converge on the configuration of lowest energy, that is the configuration corresponding to the global minimum on the potential energy surface.

The SA algorithm in SAGA is implemented as follows: At any particular time during the minimization, the system has a current reference configuration with energy E_{ref} and a simulation temperature T. The system "moves" to a trial configuration of energy E_{trial}. If the energy of the trial configuration is lower than that of the reference, then the trial replaces the reference and the algorithm proceeds to explore other "moves". If, instead, the energy of the trial configuration is higher than that of the reference, then the algorithm evaluates the Boltzmann probability factor

$$\text{Prob}(E_{trial}) \propto \exp\left[-\frac{(E_{trial}-E_{ref})}{T}\right]$$

for the trial and selects a random number between 0.0 and 1.0. If the random number is smaller than the Boltzmann factor, the trial replaces the reference; otherwise, the trial is discarded.

SA tracks the reference configurations of highest and lowest energies for fixed T. If the energies of these configurations differ by less than threshold (0.001 cm^{-1}, by default), the algorithm assumes that a configuration corresponding to the global minimum has been identified, and the search terminates.

Two important elements of the SA algorithm that influence convergence behavior include (i) the "move" that generates trial configurations and (ii) the temperature scheduling that controls the rate at which the system undergoes annealing.

The "Move": Trial configurations are generated by selecting one atom of the cluster, moving it to a different location on the surface of the cluster, and performing a conjugate-gradient optimization to the nearest local energy minimum. The algorithm used to select atoms counts the number of nearest neighboring centers. Although it is possible for any atom to be selected, it is most likely that one on the cluster surface (having fewer nearest neighboring centers) is chosen for the move. The selected atom is then moved to one of two positions, either to a random position on the cluster surface or onto one of the principal rotation axes (the axis of highest moment of inertia). Moving the atom on the principal axis generally yields a trial configuration that is more nearly spherical than the reference configuration and may, therefore, have lower energy. The algorithm chooses the random position or position on the principal axis with equal probability.

Temperature Scheduling: Annealing is controlled by partitioning the SA minimization into

macro and micro iterations. The simulation temperature is fixed for each macro iteration. The macro iteration consists of a number of micro iterations (equal to 100×natoms, by default). Energy convergence is tested at the completion of a macro iteration. If not converged, the temperature is reduced (scaled by 0.75, by default), and SA proceeds to the next macro iteration.

The initial temperature, if not specified by the user, is determined by performing 10×natoms moves and setting the initial temperature to the difference of the resulting configurations of highest and lowest energies.

It is possible for SA macro iterations to converge to a configuration having an energy higher than that of another configuration sampled during the course of the minimization. SA always tracks the configuration of lowest absolute energy. If SA converges to a configuration of higher energy, SA is restarted, beginning from the configuration of lowest energy. This ensures that SA will only ever report, upon convergence, the lowest energy configuration found.

A.6. Genetic Algorithm Implementation

A genetic algorithm (GA) is a global optimization method that mimics the natural selection processes of evolution. Members of a population mate to form offspring which, according to their fitness, are either discarded or join the population. Mutations occasionally occur, and the resulting mutant offspring can potentially join the population too, whether they are fit or not.

The GA implemented in SAGA is similar to that described by D. M. Deaven, N. Tit, J. R. Morris, and K. M. Ho, *Chem. Phys. Lett.* **256**, 195 (1996). See Section B.6 for a description of GA program parameters. We briefly describe our algorithm here.

SAGA's GA maintains a population consisting of a small number (typically 20) of cluster configurations. The fitness of each member of this population is determined by its energy, with lower energy corresponding to higher fitness. When no diversity exists in the population (that is, all members have identical fitness, or energy), we judge that the search algorithm has likely converged on the globally optimal configuration and terminate the search.

GA begins by generating a diverse population of candidate configurations. One of these candidates is the configuration supplied by the user via the $xyz group, if available. Other candidates are taken from a selection of idealized icosahedral, cubic closed packed, hexagonal closed packed, and body centered cubic configurations. Conjugate-gradient optimization to the nearest local minimum is performed on each candidate configuration.

A mating procedure is then used to generate a child configuration that inherits traits of two parent configurations. The procedure randomly selects two candidates of the population to serve as the parents. The first parent undergoes a random rigid rotation about its center of mass, which is coincident with the origin of the coordinate system. All atoms of the rotated configuration that lie below the *xy* plane are discarded. The second parent is also randomly rotated, and the atoms in it that lie above the *xy* plane are discarded. A child configuration is subsequently assembled by combining the "upper half" of the first parent with the "lower half" of the second. Care is taken to ensure that the total number of atoms is conserved during this procedure. Furthermore, atoms at the interface of the two halves may be moved to the surface of the cluster if particularly close contacts are revealed. The resulting child configuration is finally optimized to the nearest local minimum using a conjugate-gradient method.

A child configuration has the potential to enter the population by replacing one of the candidates. The algorithm targets the candidate of highest energy. If the child has lower energy than this candidate, the candidate is eliminated and the child assumes its position in the population. A child will also replace the candidate if the child is a mutant. A child configuration of energy E_{child} is designated a mutant if a random real number selected in the range 0.0 to 1.0 is less than the mutation rate (4%, by default). Mutations maintain diversity in the population, allowing the GA to sample a wide range of configurations and facilitating the search for a global solution (minimum). GA would have a tendency to converge on local solutions if mutants were consistently discarded.

GA iteratively applies the mating procedure, using parents to generate new children that may

enter the population, until all members of the population have identical low energy.

The GA implemented in SAGA is principally controlled by two keywords, "pop" and "mutate" of the $genetic group. Increasing either from its default value will force GA to explore more fully the configuration space for a global minimum. Increasing these values will also increase, potentially dramatically, the time required to complete the calculation. Decreasing either value will, of course, have the opposite effect. The default values were selected because they are perceived to offer a reasonable balance of configuration exploration and run time.

GA results are sensitive to the random number generator seed of the $control group. Random number tests are used to select parent configurations, to rotate the parents in the mating procedure, and to identify mutants. Thus, different seeds (which can result from starting SAGA calculations at different times; see "seed" in Section B.2) have the potential to yield differing globally optimized cluster configurations. The default values for the "pop" and "mutate" parameters were selected, in part, because that the final GA-optimized cluster is usually independent of the seed value.

B. User's Guide

B.1. Input Philosophy

The user controls SAGA via a text-based input file that consists of one more "groups," each group providing information to direct a portion of the calculation. A group has the following form:

$*group keyword1 keyword2* . . . $end

A group begins with the $*group* identifier ($control, $potential, $opt, $anneal, $genetic, or $xyz) and ends with a $end delimiter.

The group generally includes one or more keywords, as described in Sections B.2-B.7. Each keyword controls some aspect of the SAGA calculation. Numerical values associated with these keywords should immediately follow the keyword, usually separated from the keyword by an optional "=" character. For example,

$opt ethr=0.002 $end

sets the energy convergence threshold of the conjugate-gradient optimizer to 0.002 cm^{-1}. Units, and "%" signs (for percentage values), are understood and should not be explicitly expressed with the keyword.

Only the $control group is required; all other groups are optional.

The user should adhere to the following guidelines for groups when preparing input files.

- Input is case-insensitive and free format.
- Groups may be specified in any order within the input file.
- The group identifier (such as $control) must be the first word on the line of the input file on which it appears.
- Groups may extend across multiple lines of the input file.
- Groups cannot overlap or be nested. For example, the group $potential … $end cannot appear within the group $control … $end.
- If multiple groups have the same identifier (such as a pair of $control groups), only the first group encountered, reading from the beginning of the file, is parsed; any other group is ignored.
- Any information in the input file that appears outside a group is ignored. The "!" character can be used within a group to denote comments. Any text following a "!" character is ignored.

Sample input files are given in Section B.8.

B.2. $control Group Input

The $control group determines the principal tasks to be performed during the SAGA calculation. The user must specify either the number of atoms (natoms) or the number of shells (nshell) in this group. All other keywords are optional. Default values given in brackets.

keyword	*description*
natoms	Number of atoms in cluster; must be specified if nshell is not. A range of atoms (e.g. natoms=2-13) will direct SAGA to perform a series of calculations, from natoms=2 to natoms=13 in single atom increments.
nshell	Number of shells of atoms in cluster; must be specified if natoms is not. natoms is automatically set to the number of atoms in the Mackay icosahedral cluster consisting of nshell shells. The number of atoms, *n*, in the cluster is given by $n = \frac{1}{3}(10N^3 + 15N^2 + 11N + 3)$ where *N* is the number of shells.
runtyp	Type of calculation to perform. Allowed types include: energy – energy evaluation only force – evaluate energy and forces only [optimize] – local conjugate-gradient energy minimization anneal – global minimization by simulated annealing genetic – global minimization by genetic algorithm
method	Conjugate-gradient optimizer. [bfgs] – analytic gradient-based method (generally faster than Powell) powell – numerical gradient-based method
geometry	Origin of initial cluster configuration. [xyz] – read $xyz group, if available; otherwise icos icos – icosahedral configuration

ccp – cubic close packed configuration
fcc – face centered cubic configuration (=ccp)
hcp – hexagonal close packed configuration
bcc – body centered cubic configuration

vbe — Perform vertical binding energy analysis (see Section B.9).

The default energy resolution is 0.01 cm^{-1}. A alternate resolution (e.g. 0.05 cm^{-1}) can be specified by vbe=0.05.

VBE analysis can alternatively be enabled or disabled by vbe=.true. (with default resolution) or vbe=.false., respectively.

zpe — Calculate zero-point energy, for optimized geometries only.

ZPE analysis can alternatively be enabled or disabled by zpe=.true. or zpe=.false., respectively.

freq — Print vibrational frequencies, for optimized geometries only.

By default, frequencies are printed to two digits after the decimal point [freq=2]. Use freq=*n* to specify the desired precision (to *n* digits).

seed — Set integer seed for random number generator.

If not specified, the seed is determined by a call to the system time function time().

B.3. $potential Group Input

The $potential group specifies details of the Lennard-Jones (LJ) or extended Lennard-Jones (ELJ) interaction potential used in the SAGA calculation. The $potential group is optional. Default values are given in brackets.

keyword	*description*
label	Character label for atom monomers [Ar]. Any label, up to 16 characters, may be used, but an atomic symbol is required if zero-point energies are calculated.
mass	Atomic mass, in amu. If label is an atomic symbol, mass is set by default to that of the most abundant isotope. SAGA only uses the atomic mass when zero-point energies are calculated.
type	Type of interaction potential. Allowed types include: [lj] – Lennard-Jones potential elj – Extended Lennard-Jones potential
param	Ordered LJ or ELJ potential parameters, with energies expressed in cm^{-1} and distances in Å. For type=lj, the σ and ε parameters are listed. For example, a Lennard-Jones potential for Ar (σ=3.405 Å and ε=83.26 cm^{-1}) is specified by param=3.405,83.26. For type=elj, the ordered parameters c_6, c_8, c_{10}, c_{12}, etc. are listed. The number of terms in the ELJ potential is determined by the number of parameters listed with this option. The ELJ potential with 4-term parameter list param=-5.1904e5,0,0,8.0891e8 is identical to the LJ potential specified above, where $c_6=-4\varepsilon\sigma^6$, $c_8=c_{10}=0$, and $c_{12}=4\varepsilon\sigma^{12}$. Unless otherwise specified, SAGA uses LJ potential parameters from Hirschfelder, Curtiss, and Bird (from second virial coefficient determinations) for the rare gas atoms He, Ne, Ar, Kr, and Xe. For ELJ, SAGA uses the 6-term parameters from Schwerdtfeger, et al., *Phys. Ref. B,*

73, 064112 (2006) for He, Ne, Ar, and Kr.

B.4. $opt Group Input

The $opt group allows the user to change the default convergence thresholds for the conjugate-gradient (BFGS/Powell) optimizers. All keywords in this group are optional. Default values are given in brackets.

The $opt group controls local optimization of cluster configurations. Global optimization is controlled by the $anneal or $genetic groups.

keyword	*description*
maxit	Maximum number of iterations [200].
ethr	Energy convergence threshold [0.0001 cm^{-1}].
	The energy is converged when two consecutive energy evaluations differ by less than *ethr*.
sthr	Step convergence threshold [0.0001 Å].
	The geometry is converged when the root-mean-square displacement (step) from one geometry to the next is less than *sthr*.
gthr	Gradient convergence threshold [0.003 cm^{-1}/Å].
	The gradients are converged when the root-mean-square gradient is less than *gthr*.

B.5. $anneal Group Input

The $anneal group controls SAGA's simulated annealing algorithm. All keywords in the $anneal group are optional. Default values are given in brackets.

See Section A.6 for a brief description of the algorithm implemented in SAGA.

keyword	*description*
macro	Maximum number of macro iterations [1000].
micro	Scale factor for micro iterations [100]. The total number of micro iterations per macro iteration is *micro*×*natoms*.
ethr	Energy convergence threshold [0.001 cm^{-1}]. Simulated annealing terminates when the configurations of highest and lowest energy for a macro iteration have energies differing by less than *ethr*.
tinit	Initial annealing temperature. By default, the annealing algorithm examines 10×*natoms* randomly generated configurations, setting the initial temperature equal to the largest energy difference. If specified, give the annealing temperature in cm^{-1}.
tscale	Temperature scaling factor [0.75]. The annealing temperature is scaled by this factor after each macro iteration.

B.6. $genetic Group Input

The $genetic group controls SAGA's genetic algorithm. All keywords in the $genetic group are optional. Default values are given in brackets.

See Section A.7 for a brief description of the algorithm implemented in SAGA.

keyword	*description*
pop	Number of configurations in population [20].
mutate	Mutation frequency [4%]. Mutants (high energy child configurations) are accepted into the population at roughly this frequency.
maxit	Maximum number of iterations [1000000].
print	Print frequency [1000]. Intermediate results are reported every *print* iterations.
ethr	Energy convergence threshold [0.001 cm^{-1}]. The genetic algorithm terminates when all configurations of the population have energies differing by less than *ethr*.

B.7. $xyz Group Input

The user can supply a cluster configuration to SAGA using the optional $xyz group. This group lists the Cartesian coordinates of the atomic centers of the configuration.

```
$xyz
label1   x1   y1   z1
label2   x2   y2   z2
.
.
$end
```

"label" is usually the atomic symbol for the center; it is read but discarded. "x1, y1, z1" are the Cartesian coordinates (in Angstroms) of the first center. The $xyz group is free format.

If the user has not specified a configuration (using the geometry keyword of the $control group), SAGA searches the input file for the $xyz group. If $xyz is present, SAGA parses the group. If not present, SAGA uses a default icosahedral configuration.

The coordinates for any number of centers can be specified in the $xyz group. If the number of centers equals natoms of the $control group, the configuration is read and used as is.

If too many centers are specified in $xyz (>natoms), the "natoms" centers nearest the center-of-mass are retained; all other centers are discarded from the configuration.

If too few centers are specified in $xyz (<natoms), the configuration is augmented with centers positioned at random locations on the surface of the cluster. The algorithm employed in this case uses conjugate-gradient optimizations as centers are successively added to the cluster. Thus, the resulting cluster configuration, while having the requested number of centers, is unlikely to have any centers positioned at precisely the Cartesian coordinates given by the user in the $xyz group.

B.8. Sample Input Files

B.10. Benchmark Calculations

We report benchmark results for SA and GA optimizations for clusters having two to 105 centers. These calculations used the idealized Lennard-Jones potential

$$V = 4\sum_{i<j}\left(r_{ij}^{-12} - r_{ij}^{-6}\right) \qquad (\varepsilon = \sigma = 1.0)$$

with centers of unit mass. The input has the following form

```
$control  natoms=2-55  runtyp=anneal  zpe  seed=1234567  $end
$potential  type=lj  param=1.0,1.0  mass=1  $end
```

where runtyp is either anneal or genetic. The random number seed is fixed for all calculations to ensure reproducible results. All other program options use their default values. (The calculations reported here used a SAGA executable compiled by Portland Group's pgf77 compiler.)

The following table lists the number of conjugate-gradient optimizations (*N*) required to converge the SA and GA searches and the energy and zero-point energy (both in cm^{-1}) of the converged cluster configurations. Reference values are from D. J. Wales and J. P. K. Doye, *J. Phys. Chem. A,* **101**, 5111 (1997) and R. H. Leary and J. P. K. Doye, *Phys. Rev. E* **60**, R6320 (1999).

	simulated annealing			genetic algorithm			
n	N	E	ZPE	N	E	ZPE	ref
2	20	-1.0000	31.038	18	-1.0000	31.038	-1.0000
3	113	-3.0000	91.773	3	-3.0000	91.773	-3.0000
4	410	-6.0000	180.902	6	-6.0000	180.902	-6.0000
5	500	-9.1039	270.067	5	-9.1039	270.067	-9.1039
6	2,684	-12.7121	373.798	10	-12.7121	373.800	-12.7121
7	7,806	-16.5054	470.627	12	-16.5054	470.634	-16.5054
8	15,445	-19.8215	559.551	13	-19.8215	559.540	-19.8215
9	11,700	-24.1134	667.965	22	-24.1134	667.953	-24.1134
10	11,000	-28.4225	772.394	71	-28.4225	772.406	-28.4225
11	14,300	-32.7660	875.474	43	-32.7660	875.478	-32.7660
12	10,800	-37.9676	987.564	15	-37.9676	987.564	-37.9676
13	10,400	-44.3268	1125.336	63	-44.3268	1125.336	-44.3268
14	11,200	-47.8452	1215.214	24	-47.8452	1215.225	-47.8452
15	18,000	-52.3226	1324.447	67	-52.3226	1324.443	-52.3226
16	20,800	-56.8157	1430.674	59	-56.8157	1430.670	-56.8157
17	47,600	-61.3180	1537.495	256	-61.3180	1537.493	-61.3180

18	28,800	-66.5309	1639.037	806	-66.5309	1639.059	-66.5309
19	20,900	-72.6598	1764.022	914	-72.6598	1764.018	-72.6598
20	24,000	-77.1770	1867.799	207	-77.1770	1867.818	-77.1770
21	52,500	-81.6846	1966.994	368	-81.6846	1967.029	-81.6846
22	39,600	-86.8098	2070.673	609	-86.8098	2070.684	-86.8098
23	27,600	-92.8445	2184.346	1,549	-92.8445	2184.349	-92.8445
24	55,200	-97.3488	2285.448	355	-97.3488	2285.476	-97.3488
25	40,000	-102.3727	2382.419	390	-102.3727	2382.408	-102.3727
26	31,200	-108.3156	2486.206	1,604	-108.3156	2486.251	-108.3156
27	91,800	-112.8736	2601.497	463	-112.8255	2590.893	-112.8736
28	67,200	-117.8224	2696.257	2,237	-117.8224	2696.297	-117.8224
29	31,900	-123.5874	2786.588	2,690	-123.5874	2786.641	-123.5874
30	42,000	-128.2866	2905.035	13,035	-128.2866	2905.028	-128.2866
31	195,300	-133.5864	3183.123	12,701	-133.5864	3183.118	-133.5864
32	35,200	-139.6355	3313.923	8,562	-139.6355	3313.914	-139.6355
33	122,100	-144.8427	3432.647	6,107	-144.8427	3432.644	-144.8427
34	78,200	-150.0445	3551.231	17,059	-150.0445	3551.245	-150.0445
35	42,000	-155.7566	3674.148	22,833	-155.7566	3674.161	-155.7566
36	57,600	-161.8254	3804.606	20,850	-161.8254	3804.570	-161.8254
37	92,500	-167.0337	3923.156	7,822	-167.0337	3923.078	-167.0337
38	38,000	-173.9284	4122.638	18,706	-173.9284	4122.697	-173.9284
39	42,900	-180.0332	4194.595	12,026	-180.0332	4194.700	-180.0332
40	64,000	-185.2498	4313.406	14,895	-185.2498	4313.389	-185.2498
41	102,500	-190.5363	4432.495	26,084	-190.5363	4432.373	-190.5363
42	63,000	-196.2775	4554.548	82,679	-196.2775	4554.590	-196.2775
43	107,500	-202.3647	4684.114	35,733	-202.3647	4684.063	-202.3647
44	92,400	-207.6887	4803.263	64,701	-207.6887	4803.065	-207.6887
45	81,000	-213.7849	4930.366	47,496	-213.7849	4930.416	-213.7849
46	64,400	-220.6803	5071.920	54,564	-220.6803	5071.909	-220.6803
47	75,200	-226.0123	5191.074	17,943	-226.0123	5191.073	-226.0123
48	96,000	-232.1995	5317.091	29,752	-232.1995	5317.218	-232.1995
49	63,700	-239.0919	5457.723	24,797	-239.0919	5457.775	-239.0919
50	80,000	-244.5499	5577.785	54,687	-244.5499	5577.775	-244.5499
51	122,400	-251.2540	5711.317	12,148	-251.2179	5711.453	-251.2540
52	135,200	-258.2300	5851.700	10,046	-258.2300	5851.608	-258.2300
53	132,500	-265.2030	5991.759	6,229	-265.2030	5991.708	-265.2030
54	59,400	-272.2086	6131.446	6,221	-272.2086	6131.361	-272.2086

55	71,500	-279.2485	6270.606	10,470	-279.2485	6270.606	-279.2485
56	173,600	-283.6431	6367.806	6,522	-283.6431	6367.759	-283.6431
57	96,900	-288.3426	6478.024	9,178	-288.1401	6465.544	-288.3426
58	191,400	-294.3781	6612.767	53,596	-293.5232	6588.821	-294.3781
59	118,000	-299.7381	6724.895	71,384	-299.6161	6722.723	-299.7381
60	84,000	-305.8755	6858.700	107,497	-305.8755	6858.646	-305.8755
61	79,300	-312.0089	6991.722	159,104	-311.0455	6966.305	-312.0089
62	198,400	-317.3539	7099.495	240,183	-317.3539	7099.552	-317.3539
63	296,100	-323.4897	7232.289	250,970	-323.4897	7232.344	-323.4897
64	153,600	-326.9407	7370.541	478,736	-328.6562	7339.869	-329.6201
65	663,000	-334.9715	7473.232	418,793	-334.9147	7470.979	-334.9715
66	105,600	-341.1106	7602.680	459,352	-341.1106	7602.757	-341.1106
67	120,600	-347.2520	7735.177	625,403	-347.2520	7735.072	-347.2520
68	149,600	-353.3945	7828.871		no conv		-353.3945
69	124,200	-359.8826	7981.459				-359.8826
70	98,000	-366.8922	8119.835				-366.8923
71	92,300	-373.3497	8232.831	410,367	-372.0996	8227.458	-373.3497
72	165,600	-378.6373	8352.727	675,590	-378.6373	8352.797	-378.6373
73	109,500	-384.7894	8470.055	620,331	-384.7894	8469.996	-384.7894
74	125,800	-390.9085	8602.374	420,957	-390.1573	8591.446	-390.9085
75	330,000	-396.2822	8723.741		no conv		-397.4923
76	539,600	-402.3846	8840.879		no conv		-402.8949
77	146,300	-408.5183	8973.360	581,581	-406.2577	9144.669	-409.0835
78	171,600	-414.7944	9113.937	62,745	-412.7818	9279.530	-414.7944
79	126,400	-421.8109	9252.316	97,615	-420.2081	9434.627	-421.8109
80	128,000	-428.0836	9355.751	15,028	-424.6806	9529.573	-428.0836
81	121,500	-434.3436	9459.383		no conv		-434.3436
82	131,200	-440.0414	9585.849		no conv		-440.5504
83	157,700	-445.8006	9695.817				-446.9241
84	655,200	-452.0248	9869.847				-452.6572
85	161,500	-459.0558	10008.208				-459.0558
86	137,600	-465.2379	10109.999		no conv		-465.3845
87	1,148,400	-472.0982	10377.668				-472.0982
88	149,600	-477.4990	10305.951				-479.0326
89	391,600	-483.1937	10431.854				-486.0539
90	540,000	-492.4339	10795.432				-492.4339
91	309,400	-498.8111	10927.070		no conv		-498.8111

92	147,200	-505.1853	11058.703				-505.1853
93	269,700	-510.8777	11185.455				-510.8777
94	206,800	-514.5697	11061.667				-517.2641
95	199,500	-523.6402	11448.829				-523.6402
96	355,200	-529.8791	11582.651		no conv		-529.8791
97	194,000	-536.6814	11721.193				-536.6814
98	245,000	-543.6430	11868.226				-543.6654
99	158,400	-550.6665	12005.412				-550.6665
100	460,000	-557.0398	12136.608				-557.0398
101	181,800	-563.4113	12267.907	702,101	-560.6457	12308.666	-563.4113
102	183,600	-568.3888	12440.567				-569.3637
103	298,700	-575.6589	12527.885				-575.7661
104	457,600	-582.0384	12659.319				-582.0866
105	273,000	-588.2665	12792.905				-588.2665

C. Programmer's Guide

C.1. Main Program and Core Task Routines

The following routines implement the principal tasks of the SAGA program:

`program saga`
Main SAGA program.

`subroutine defalt(natoms)`
Set program defaults.

`subroutine init(c,ct,rt,idx,list,natoms,maxatm,igeom)`
Generate an initial icosahedral, ccp, hcp, or bcc configuration.

`subroutine force(c,d,natoms)`
Evaluate and print forces.

`function opt(c,natoms)`
Perform local optimization of configuration `c(3,natoms)`, and return the resulting energy.

`function anneal(c,ct,cg,natoms)`
Perform global, simulated annealing optimization of configuration `c(3,natoms)`, and return the resulting energy.

`function genetic(c,cg,ch,t1,t2,ep,p,ct,scr,idx,list,natoms, maxatm)`
Perform global optimization of configuration `c(3,natoms)` using the genetic algorithm. Return the resulting energy.

`function zpe(c,h,u,v,x,l,natoms)`
Evaluate vibrational frequencies and report the zero-point energy of configuration `c(3,natoms)`.

`subroutine vbeanl(c,ct,e,rav,list,nb,natoms)`
Perform vertical binding energy analysis of configuration `c(3,natoms)`.

C.2. Input Parsing

The following routines parse the groups of the input file.

`subroutine coninp(natoms)`
Parse the $control group.

`subroutine vinp()`
Parse the $potential group.

`subroutine optinp()`
Parse the $opt group.

`subroutine anninp()`
Parse the $anneal group.

`subroutine geninp()`
Parse the $genetic group.

`subroutine xyzinp(c,ct,rsq,list,natoms,maxatm)`
Parse the $xyzinp group.

If the number of atoms listed in the $xyz group exceeds natoms, the atoms closest to the center-of-mass are retained; all others are discarded.

If the number of atoms listed in the $xyz group is less than natoms, atoms are added to the configuration using the move subroutine described elsewhere.

C.3. Local Optimization

SAGA implements two conjugate-gradient methods for local optimizations of cluster configurations. These are the Powell method and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method. BFGS relies on the analytic evaluation of energy first-derivatives, whereas Powell is entirely numerical.

The routines implemented here are based on those of Numerical Recipes.

`function powell(c,ct,cit,ctt,tt,ci,natoms)`
Implementation of the Powell conjugate-gradient method.

`function bfgs(c,ct,d,dd,xi,hdd,scr,hess,natoms)`
Implementation of the BFGS conjugate-gradient method.

`subroutine mnbrak(ax,bx,cx,c,s,scr,n)`
Bracket the energy minimum along the search vector s(n).

`subroutine linmin(f,c,s,scr,n)`
Locate the energy minimum along search vector s(n).

`subroutine dlinmn(f,c,s,scr,n)`
Locate the energy minimum along search vector s(n) using analytic derivatives.

`function brent(xlam,ax,bx,cx,c,s,scr,n)`
Use the Brent method to refine the energy minimum.

`function brent(xlam,ax,bx,cx,c,s,scr,n)`
Use the Brent method, with analytic derivatives, to refine the energy minimum.

`function f1dim(x,c,s,ct,n)`
Return the energy at point x along the search vector s(n).

`function df1dim(x,c,s,ct,n)`
Return the first derivative of the energy at point x along the search vector s(n).

C.4. Global Optimization

The following routines support SAGA's implementation of the simulated annealing (see anneal) and genetic algorithm (see genetic) methods for global optimization.

`subroutine adjust(c,ct,natoms)`
Select a center at random from configuration c(3,natoms) and move it to a new position to generate a new configuration. The original configuration is temporarily stored in ct(3,natoms).

`subroutine select(ith,nb,natoms)`
Select, at random, the *i*th center from a list of atoms. nb(natoms) is the number of neighboring centers for each atom. The algorithm implemented is most likely to select a surface atom, that is an atom with small nb value.

`subroutine move(ith,c,natoms,r)`
Move the *i*th atom to a new position on the surface of the configuration c(3,natoms). The algorithm implemented either (i) moves the atom to an arbitrary position on the surface or (ii) moves the atom onto the principal axis having the largest moment of inertia.

`function metrop(etst,eref,temp)`
Use the Metropolis algorithm to test etst. If etst is less than eref, set metrop>1. If etst is greater than eref, select a random number between 0.0 and 1.0. If the random number is less than exp(-(etst-eref)/temp), set metrop>1. Otherwise, set metrop=0.

`subroutine restor(c,ct,natoms)`
Restore configuration c(3,natoms) from ct(3,natoms).

`subroutine mate(c,p1,p2,t1,t2,scr,list,natoms)`
Construct a child configuration c(3,natoms) from parent configurations p1(3,natoms) and p2(3,natoms). The parent configurations are randomly rotated in three dimensions. The child is then constructed from the upper half of p1 and the lower half of p2. Atoms at the interface of these halves are moved to the surface as needed (using move) to ensure that atoms are not strongly overlapping.

C.5. Energies and Energy Derivatives

The following routines are used to evaluate the energies and energy derivatives of the Lennard-Jones (LJ) and extended Lennard-Jones (ELJ) potentials:

`function enrg(c,natoms)`
Return the potential energy of the configuration c(3,natoms), where c lists the Cartesian coordinates of the configuration in Angstrom units. The energy is

$$V = \sum_{i<j}^{n} V_{ij}(r_{ij}) \quad \text{where} \quad r_{ij} = \left((x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2\right)^{1/2}$$

and the pairwise V_{ij} contributions are those of the LJ or ELJ potentials as determined by the value of itype in common/cints/.

`real*8 function lj(eps,sigma,rsq)`
Return the value of the Lennard-Jones potential evaluated at the square distance rsq. The LJ pair potential is defined as:

$$V_{ij} = 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]$$

where ε is the binding energy of pair and σ is associated with the equilibrium pair separation:

$$r_{eq} = \sqrt[6]{\sigma}$$

`function elj(p,n,rsq)`
Return the value of the extended Lennard-Jones potential evaluated at the square distance rsq. p(n) lists the n parameters of this potential. The ELJ pair potential is defined as:

$$V_{ij} = \sum_{\alpha=1}^{n} \frac{p_\alpha}{(r_{ij}^2)^{\alpha+2}}$$

`subroutine denrg(d,c,natoms)`
Evaluate the energy first-derivatives d(3,natoms) of the configuration c(3,natoms). For the LJ potential, these derivatives are:

$$\frac{dV}{dx_i} = 24\varepsilon \sum_{j \neq i} \frac{x_i - x_j}{r_{ij}^2}\left[\left(\frac{\sigma}{r_{ij}}\right)^{6} - 2\left(\frac{\sigma}{r_{ij}}\right)^{12}\right]$$

For ELJ, the first derivatives are:

$$\frac{dV}{dx_i} = -\sum_{j\neq i}\frac{x_i - x_j}{r_{ij}^2}\sum_{\alpha}\alpha\frac{p_\alpha}{r_{ij}^\alpha}$$

`subroutine hess(h,c,natoms)`
Evaluate the energy second-derivatives h(3*natoms,3*natoms) of the configuration c(3,natoms). For the LJ potential, these are:

off-diagonal blocks (for atoms *i* and *j*):

$$\frac{d^2V}{dx_i dx_j} = 24\varepsilon\left[\frac{1}{r_{ij}^2}\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] - \frac{(x_i - x_j)^2}{r_{ij}^4}\left[28\left(\frac{\sigma}{r_{ij}}\right)^{12} - 8\left(\frac{\sigma}{r_{ij}}\right)^{6}\right]\right]$$

$$\frac{d^2V}{dx_i dy_j} = -96\varepsilon\frac{(x_i - x_j)(y_i - y_j)}{r_{ij}^4}\left[7\left(\frac{\sigma}{r_{ij}}\right)^{12} - 2\left(\frac{\sigma}{r_{ij}}\right)^{6}\right]$$

diagonal blocks (for atom *i*):

$$\frac{d^2V}{dx_i^2} = -\sum_{j\neq i}\frac{d^2V}{dx_i dx_j}$$

$$\frac{d^2V}{dx_i dy_j} = -\sum_{j\neq i}\frac{d^2V}{dx_i dy_j}$$

For the ELJ potential, the second derivatives are:

off-diagonal blocks (for atoms *i* and *j*):

$$\frac{d^2V}{dx_i dx_j} = \frac{1}{r_{ij}^2}\sum_{\alpha}\alpha\frac{p_\alpha}{r_{ij}^\alpha} - \frac{(x_i - x_j)^2}{r_{ij}^4}\sum_{\alpha}\alpha(\alpha+2)\frac{p_\alpha}{r_{ij}^\alpha}$$

$$\frac{d^2V}{dx_i dy_j} = -\frac{(x_i - x_j)(y_i - y_j)}{r_{ij}^4}\sum_{\alpha}\alpha(\alpha+2)\frac{p_\alpha}{r_{ij}^\alpha}$$

diagonal blocks (for atom *i*):

$$\frac{d^2V}{dx_i^2} = -\sum_{j\neq i}\frac{d^2V}{dx_i dx_j}$$

$$\frac{d^2V}{dx_i dy_j} = -\sum_{j \neq i} \frac{d^2V}{dx_i dy_j}$$

C.6. String Query and Manipulation

String operations are performed by the following routines:

`subroutine czero(a,n)`
Fill string a with n blanks.

`logical function equal(word1,len1,word2,len2,len,case)`
Test the first len characters of word1(1:len1) and word2(1:len2) to determine whether these strings are equivalent (equal=.true.) or not (equal=.false.). The test is case sensitive (case=.true.) or not (case=.false.).

`subroutine in2ch(in,ch,nc)`
Construct string representation ch(1:nc) of integer in. On exit, nc is the number of characters required to represent in.

`function lentrm(string)`
Return the position of the last non-blank character in string.

`function lenwrd(string)`
Return the number of characters in the first word of string.

`subroutine re2ch(re,nd,ch,nc)`
Construct string representation ch(1:nc) of real number re. nd is the number of decimal places to include in ch. If nd<0, as many places as possible are included. On exit, nc is the number of characters required to represent re.

`function ucase(word,len)`
Replace all lowercase characters in word(1:len) by uppercase characters.

C.7. Memory Management

SAGA is configured to use a large scratch array for most of its significant memory needs. Portions of this array are dynamically allocated (and released and reallocated) during the course of a calculation. The amount of scratch memory available is determined during program installation by setting the mbytes parameter of the mbytes.fh include file:

```
parameter(mbytes = 1024)
parameter(mwords = 131072 * mbytes)
```

By default, SAGA is configured to use a 1GB (1024MB) scratch array. mbytes can be set to any value desired, up to 2047MB. SAGA calculations will halt if insufficient scratch memory is available.

The scratch array is defined in common/mem/ of the mem.fh include file (and in meminit.f).

```
common/mem/a(mwords)
```

common/scr/ of the scr.fh include file is used to track the number of words (ipt) currently allocated in the scratch array and the maximum number of words (imx) used during the calculation (since meminit was last called).

```
common/scr/ipt,imx
```

The following routines manage memory for SAGA:

`subroutine meminit()`
Initialize the scratch memory array.

`function memalli(n)`
Allocate n integer*4 elements. memalli returns an integer pointer identifying the first element of the scratch array associated with these n elements. Calls memall.

`function memallr(n)`
Allocate n real*8 elements. memallr returns an integer pointer identifying the first element of the scratch array associated with these n elements. Calls memall.

`function memall(n,type)`
Allocate n elements of type type, where type = 'c' (character*1), 'l' (logical*1), 'i' (integer*4), or 'r' (real*8). memall returns an integer pointer identifying the first element of the scratch array associated with these n elements.

`subroutine memrel(ip)`
Release all memory in the scratch array including and after element (pointer) ip.

`function memleft()`
Returns the number of unused words remaining in the scratch array.

`function memused()`
Returns the maximum number of words used since the scratch array was last initialized by meminit.

Using scratch memory in a program is generally implemented as follows:

```
      include 'mem.fh'

c  Initialize memory:

      call meminit()

c  Allocate two real*8 arrays and one integer array:

      i1 = memallr(3*natoms)
      i2 = memallr(natoms*natoms)
      j1 = memalli(natoms)

c  Perform some task:

      call task1(a(i1),a(i2),a(j1))

c  Release memory to pointer i2 (retaining i1):

      call memrel(i2)

c  Reallocate memory:

      i2 = memallr(natoms)
      i3 = memallr(3*natoms)
      j1 = memalli(9*natoms)

c  Another task:

      call task2(a(i1),a(i2),a(i3),a(j1))

c  Release all memory (to i1) and print usage:

      call memrel(i1)
      write(6,*) 'Total memory used = ',memused()
```

C.8. Matrix and Vector Operations

The following are utility routines that perform a variety of matrix and vector functions:

`subroutine icopy(ia,ib,n)`
Copy integer vector ia(n) to ib(n).

`subroutine izero(ia,n)`
Form the null integer vector ia(n).

`subroutine mcopy(a,b,m,nr,nc)`
Copy matrix a(nr,nc) to b(nr,nc). m is the defined row dimension of a and b.

`subroutine mdiag(a,b,m,n)`
Copy the n diagonal elements of a(m,m) into b(n).

`subroutine mmult(a,b,c,v,m,n)`
Multiply matrices a(n,n) and b(n,n), storing result in c(n,n) and using v(n,n) as scratch storage. m is the defined row dimension of each matrix.

`subroutine mout(a,mr,nr,nc)`
Print matrix a(nr,nc) to standard out. m is the defined row dimension of a.

`subroutine mrot(a,natoms,r)`
Rotate in three dimensions the Cartesian coordinates in a(3,natoms) by r(3,3).

`subroutine msimtr(a,b,x,m,n)`
Similarity transform matrix a(n,n) by b(n,n) using x(n,n) as scratch storage. m is the defined row dimension of each matrix.

`subroutine mtrnsp(a,m,n)`
Transpose matrix a(n,n). m is the defined row dimension of a.

`subroutine munit(a,m,n)`
Form the unit matrix in a(n,n). m is the defined row dimension of a.

`subroutine mzero(a,m,nr,nc)`
Form the null matrix in a(nr,nc). m is the defined row dimension of a.

`subroutine vcopy(a,b,n)`
Copy vector a(n) to b(n).

`subroutine vdiff(a,b,d,n)`
Evaluate the vector difference d(n) = a(n) – b(n).

`function vdot(a,b,n)`

Return the dot product of vectors a(n) and b(n).

`function vlen(a,n)`
Return the length of vector a(n).

`subroutine vnorm(a,n)`
Normalize vector a(n) to unit length.

`subroutine vscal(a,n,scale)`
Scale vector a(n) by the factor scale.

`subroutine vsum(a,b,s,n)`
Evaluate the vector sum s(n) = a(n) + b(n).

`subroutine vzero(a,n)`
Form the null vector a(n).

C.9. General Input

SAGA includes a set of general, free-format input routines designed to conveniently parse the input file. The following guidelines apply:

- An input line can consist of up to 256 characters. Any additional characters are ignored.
- Commas (,) and equal signs (=) are treated as blank characters.
- Exclamation pointx (!) delimit comments. The exclamation point and any characters that follow it are ignored.
- The following are examples of fields that are recognized as real numbers. The first three can also be read as integers.

```
204         -2.03d2     e6          2.345       5.5E-03
```

Input is generally handled by the following five routines, one to initiate input and force lines of the input file to be read (deck), three routines to parse character, real, and integer fields (cfld, rfld, ifld), and one to return a field if intending to re-read it (rtnfld). The logicals error and eof respectively signal (when .true.) that an error has been encountered parsing input or that the end-of-file has been reached.

`subroutine deck(lfn,eof)`
Initiate input from unit lfn and read the first line of input (or the next line of input if lfn is already open).

`subroutine cfld(string,nc,eof)`
Read the next word in the input file. The result is returned in string(1:nc).

`function rfld(error,eof)`
Return the real*8 value of the next word in the input file. error is .true. when the next word cannot be interpreted as a real number.

`function ifld(error,eof)`
Return the integer value of the next word in the input file. error is .true. when the next word cannot be interpreted as an integer.

`subroutine rtnfld()`
Return (reuse) the last word for next call to cfld, rfld, or ifld.

The following are utility routines that support the five routines listed above.

`subroutine card(eof)`
Read the next input line.

`subroutine word(len,eof)`
Parse next word on input line.

C.10. Utility Routines

The following routines are of general utility:

`function augmnt(c,natoms)`
Augment the configuration by a single atom, placing the atom on the cluster surface near the principal axis of highest moment of inertia. Perform conjugate-gradient optimization of the resulting configuration, returning its energy and equilibrium coordinates, the latter in c(3,natoms).

`subroutine com(x,c,natoms)`
Determine the center of mass, x(3), of the configuration c(3,natoms). All centers are assumed to be of equal mass.

`function dist(a,b)`
Return the distance between points a(3) and b(3).

`function distsq(a,b)`
Return the square of the distance between points a(3) and b(3).

`subroutine euler(rot,alpha,beta,gamma)`
Construct the rotation matrix rot(3,3) based on the Euler angles alpha, beta, and gamma using the x-y-z convention with right-handed positive rotation.

`subroutine frqout(f,nf,ifrq)`
Print the frequencies in f(nf), grouping frequencies that differ by less ±1 in the ifrqth decimal place.

`function getr()`
Return the equilibrium pair separation of the defined LJ or ELJ potential. For LJ, this value is determined analytically. For ELJ, the value is obtained through the conjugate-gradient optimization of the dimer.

`subroutine halt(message)`
Halt program execution, writing message and reporting memory usage and timing information.

`subroutine ihsops(sop)`
Generate in sop(3,3,120) the 120 symmetry operators of the I_h group.

`function intrn(ilo,ihi)`
Return a random integer in the inclusive range ilo to ihi.

`subroutine itensr(xi,c,natoms)`
Return in xi(3,3) the moment of inertia tensor for configuration c(3,natoms).

`subroutine jobtim(elap,tot)`

Return the elapsed time elap (since last call) and total run time tot. Timing information is acquired by calls to the Fortran 3F dtime function, which may not be portable.

`subroutine mackay(n,c)`
Generate in c(3,*) the coordinates for the idealized Mackay icosahedral configuration consisting of n shells.

`subroutine naybor(nb,rav,c,natoms,r)`
Return in nb(natoms) and rav(natoms), respectively, the number of neighboring atoms and the average distance to these neighbors for each of the atoms in configuration c(3,natoms). r is the threshold distance used to determine whether a pair of atoms are neighbors or not.

`function nctrs(nshell)`
Return the number of centers in the Mackey icosahedral configuration that consists of nshell complete shells.

`function nshell(natoms)`
Return the number of shells in the smallest Mackey icosahedral configuration that consists of at least natoms atoms.

`function pythag(a,b)`
Return the length of the hypotenuse given the lengths a and b of the sides of a right triangle.

`subroutine rank(a,n,l)`
Rank (from lowest to highest, and to within $\pm 10^{-6}$) the values in a(n). l(n) reports the original location of each element of the sorted a values.

`function ran3()`
Return a uniform random deviate in the inclusive range 0 to 1.

`subroutine ranrot(c,natoms)`
Perform a random three-dimensional rotation of configuration c(3,natoms).

`function realrn(xlo,xhi)`
Return a random real value in the inclusive range xlo to xhi.

`subroutine shell(n,c,natoms,sops)`
Generate in c(3,natoms) the coordinates for all atoms residing in the nth shell of the Mackay icosahedral configuration. Uses the I_h symmetry operators of sops(3,3,120).

`subroutine svdcmp(a,m,n,mp,np,w,v,xs)`
Use singular value decomposition to evaluate the eigenvalues w(n) and eigenvectors v(n,n) of matrix a(m,n). Defined dimensions of the matrices and vectors are a(mp,np), w(np), v(np,np), and xs(n).

`subroutine trans(x,c,natoms)`

Translate the configuration c(3,natoms) so that the center of mass coincides with x(3).

`subroutine vdef()`
Load default potential parameters into common/creals/.

`function wrgeom(c,natoms,idx)`
Shift the center of mass of configuration c(3,natoms) to the origin, write the configuration in XMol format to unit 7, and return the energy to the calling program. The title line of the XMol formatted file reports the energy and the arbitrary geometry index idx.

`subroutine xshift(c,natoms)`
Translate the configuration c(3,natoms) so that the center of mass coincides with the origin of the coordinate system.

`function xmass(atsym)`
Return the atomic mass (in amu) of the most abundant isotope of atomic symbol atsym.