Molecular mechanics

Author: Martijn Oele

GitHub: Martijn-075

VUID:

Table of Contents

[1. Modules 3](#_Toc130757264)

[1.1. Molecule module 3](#_Toc130757265)

[1.2. Energy module 3](#_Toc130757266)

[1.3. Metropolis module 3](#_Toc130757267)

[1.4. Math module 3](#_Toc130757268)

[1.5. Constants module 3](#_Toc130757269)

[2. Data types 2](#_Toc130757270)

[2.1. Atom 2](#_Toc130757271)

[2.2. Bond 2](#_Toc130757272)

[2.3. Bonds\_angle 2](#_Toc130757273)

[2.4. Torsion 2](#_Toc130757274)

[2.5. Molecule 3](#_Toc130757275)

[3. Subroutines and functions 4](#_Toc130757276)

[3.1. Molecule subroutines 4](#_Toc130757277)

[3.1.1. Read atom(mol, filename) and Write atom(mol, filename) 4](#_Toc130757278)

[3.1.2. Bonds atom(mol) 4](#_Toc130757279)

[3.1.3. Angle bonds(mol) 5](#_Toc130757280)

[3.1.4. Angle torsion(mol) 5](#_Toc130757281)

[3.1.5. Create molecule(mol) 6](#_Toc130757282)

[3.1.6. Delete molecule(mol) 6](#_Toc130757283)

[3.2. Energy functions 6](#_Toc130757284)

[3.2.1. Stretch energy(mol) 6](#_Toc130757285)

[3.2.2. Bending energy(mol) 7](#_Toc130757286)

[3.2.3. Torsion energy(mol) 7](#_Toc130757287)

[3.2.4. Electrostatic energy(mol) 8](#_Toc130757288)

[3.2.5. Van der Waals energy(mol) 8](#_Toc130757289)

[3.2.6. Forcefield energy(mol) 8](#_Toc130757290)

[3.3. Metropolis subroutines 9](#_Toc130757291)

[3.3.1. Random atom metropolis(mol, r) 9](#_Toc130757292)

[3.3.2. Metropolis(mol, T, r) 9](#_Toc130757293)

[3.3.3. Minimize energy 10](#_Toc130757294)

[3.4. Supporting math functions 11](#_Toc130757295)

[3.4.1. Deg to rad(deg) and rad to deg(rad) 11](#_Toc130757296)

[3.4.2. Cross product(vec1, vec2) 11](#_Toc130757297)

[3.4.3. Vector length(vector) 11](#_Toc130757298)

[4. Interdependencies (flow diagram) 12](#_Toc130757299)

[5. In and output file format 13](#_Toc130757300)

[5.1. Ch4.xyz 13](#_Toc130757301)

[5.2. C4h10.xyz 13](#_Toc130757302)

[5.3. C6h12.xyz 14](#_Toc130757303)

[5.4. C2h6.xyz 14](#_Toc130757304)

# Modules

## Molecule module

The molecule module contains the data types used to simplify argument passing in future subroutines and functions. The main data type molecule is split into several data types such as bond, bond angle, and torsion. This module also contains the subroutines associated with calculating the relevant parameters of the molecule such as the bonds, bonds angles, and torsion angles

## Energy module

The energy module contains all the function for calculating the different energies of the molecule. This includes the stretch, bending, torsion, electrostatic, and van der Waals energies. All these energies are combined into the forcefield energy.

## Metropolis module

The metropolis module contains the metropolis algorithm which minimizes the energy of the molecule by randomly sampling and changing the atoms position within a defined radius and checks if the new forcefield energy is lower than the previous iteration. If the energy is higher there is still a nonzero probability that the position of the new atom is accepted based on the Boltzmann distribution. This is repeated until the new atoms' positions are rejected a certain amount of time in a row. The energy of the molecule is then believed to be at a (local) minimum.

## Math module

Not all basic math functions used in this program were available as Fortran functions so these simple functions were implemented. This includes degree and radian conversion, 3D cross-product, and vector length functions.

## Constants module

The constants module contains all the constants that are used throughout the program. It contains the number of bytes for all real numbers and constants for the molecule, energy, and metropolis module.

# Data types

For real number realkind = 8 by default.

## Atom

|  |  |
| --- | --- |
| Data type | Description |
| character(1) :: element | The element of the atom (in this project can only be C for carbon or H for hydrogen). |
| real(realkind) :: cords(3) | The 3D cartesian coordinates of the atom. |

## Bond

|  |  |
| --- | --- |
| Data type | Description |
| integer :: link(2) | The atom indices of the atoms that are forming a bond. |
| character(2) :: type | The type of bond (in this project can only be CC for carbon carbon bond of CH for carbon hydrogen bond. |
| real(realkind) :: length | The length of the bond extracted from the distance array (see 2.5. Molecule). |
| real(realkind) :: vector(3) | The 3D vector of the bond calculated using the boding atoms coordinates. |

## Bonds\_angle

|  |  |
| --- | --- |
| Data type | Description |
| type (bond) :: bonds(2) | The 2 connecting bonds that form a angle.  See 2.2. Bonds. |
| real(realkind) :: angle | The angle of the bonds in degrees. |

## Torsion

|  |  |
| --- | --- |
| Data type | Description |
| type (bond) :: end\_bonds(2) | The 2 end bonds that have an angle in the Newman projection.  See 2.2. Bond. |
| type (bond) :: center\_bond | The central (CC) bond that connects the end bonds.  See 2.2. Bond. |
| real(realkind) :: angle | The angle of the two end bond from Newman projection. |

## Molecule

|  |  |
| --- | --- |
| Data type | Description |
| type (atom), allocatable :: atoms(:) | The atoms of the molecule.  See 2.1. atom. |
| type (bond), allocatable :: bonds(:) | The bonds of the molecule.  See 2.2. Bonds. |
| type (bond\_angle), allocatable :: angles(:) | The angles between 2 connecting bonds in the molecule.  See 2.3. Bonds\_angle. |
| type (torsion), allocatable :: torsion\_angles(:) | The torsion angle between 3 connecting bonds. The molecule must contain at least 1 CC bond otherwise not used.  See 2.4.Torsion. |
| real(realkind), allocatable :: distance(:,:) | The array of all the distances between the atoms in the molecules. Is used to determine if atoms are boding and for nonbonding interactions. |
| logical, allocatable :: bonding(:,:) | Supports the distance array and stores if atoms are bonding. |

# Subroutines and functions

## Molecule subroutines

### Read atom(mol, filename) and Write atom(mol, filename)

Parameters:  
Atoms is a derived atom type (type (bond)) see 2.1. atom. (In practice the sub-type atoms from the derived type molecule is passed in this function (molecule%atoms))  
Filename is a character string containing the filename (preferably ending in .xyz) of the file to read or write to.

These subroutines read and write the coordinates and elements of the atoms.

Operation (read atom):

1. The file is opened
2. The number of atoms is used to allocate the atoms vector in the molecule type
3. For every line (number of atoms) the element (character(1)) and the x, y, z coordinates 3 (real numbers) are read in
4. The file is closed

Operation (write atom)

1. The file is opened
2. The number of atoms is written
3. An empty line is written
4. The element (character(1) and the x, y, z coordinates are written on one line per atom
5. The file is closed

The to-be-read or written file has the format specified in section 5.

### Bonds atom(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

The bonds atom subroutine calculates the distances between all atoms (later on used for nonbonding interactions as well) and determines if two atoms are bonding based on the elements of the atoms and their distance. The bonds are stored in the bonds type (molecule%bonds).

Operation:

1. The bonds vector inside the molecule type is allocated at the same size as the atoms vector (number of atoms)
2. The distance and bonding arrays that are part of the molecule type are allocated as a square matrix with the same size as the atoms vector (and bonds) in both dimensions
3. The distances between all atoms are calculated and stored in the distance array (molecule%distance)
4. All unique atom pairs are checked if they are bonding by checking:
   1. For carbon-carbon bonds:
      1. Check if both atoms are carbon (element == C)
      2. Check if the distance is within 10% of the optimal carbon-carbon single bond
   2. For carbon hydrogen:
      1. Check if the first atom is carbon and the second a hydrogen atom
      2. Check if the distance is within 10% of the optimal carbon-hydrogen bond
5. If one of these criteria is met:
   1. The indices of the atoms are stored in the bond%link
   2. The distance (now bond length) is stored in the bond%length
   3. The bond vector is calculated and stored in bond%vector
   4. In the bonding array, the position is set to true
   5. The bond type (CC or CH) is stored in bond%type

### Angle bonds(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

The angle bonds subroutine calculates the angle of two connecting bonds. First, the indices of the carbon atoms are extracted. This is done because all bonds that are connected are connected via a carbon atom. Then the four bonds that that carbon atom makes is extracted for every carbon atom. With these four bonds, six unique connected bond pairs can be made per carbon. The angles between all unique connected bond pairs are calculated using the dot product of the two bond vectors calculated in 3.1.2. bonds atom.

Operation:

1. The bonds vector inside the molecule type is allocated (6 x number of carbon atoms)
2. all indices of the carbon atoms are stored in a carbon indices vector
3. For each carbon atom, the four bonds are stored in a bonds holder array (4 x number of carbons) (type bond)
4. For all carbon atoms with the four bonds six unique connected bond pairs are created
   1. The two bonds making the angel are stored in molecule%angles%bonds vector (type bond)
   2. The angle is calculated via the dot product of the two bonds vectors and is stored in molecule%angles%angle
5. If an angle is less than 90 degrees the angle is changed to the correct angle by subtracting it from 180 degrees

### Angle torsion(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

The angle torsion subroutine calculates the torsion (dihedral) angle over three consecutive bonds. All torsion angles have carbon-carbon as their central bond. First, it is checked if there is a CC bond in the molecule if there isn’t one the subroutine is skipped and no torsion calculations will be made. This will not give an error but just skip this subroutine. If there is or are cc bonds these bonds are copied to a bonds holding array. For each of these bonds on both sides, the three remaining bonds are extracted and stored in end bonds holding array. With these three bonds on each side of the CC bond nine unique bond pairs are made and stored in molecule%torsion\_angels%end\_bonds together with the central CC bond in molecule%torsion\_angle%central\_bond. With these bonds, the torsion angle is calculated by first calculating the two cross products of the central CC bond and one end bond on each side of the central bond vectors. With these new vectors, the angle is calculated using the dot product.

Operation:

1. The torsion angle vector is allocated based on the number of CC bonds ( 9 x the number of CC bonds)
2. All CC bonds are stored in a CC bond holder vector (type bond)
3. For each CC bond, the 3 other bonds on each side of the bonds is extracted and stored in an end bonds holding array (type bond)
4. From these central and end bonds for each CC bond, 9 unique consecutive bond pairs are made
   1. The central CC bond is stored in molecule%torsion\_angels%central\_bond
   2. The two end bonds are stored in molecule%torsion\_angle%end\_bonds
5. With these unique bond combinations, the torsion angle is calculated
   1. Two new vectors are computed by the cross product of the central bond vector and one of the end bond vector
   2. With these new vectors, the torsion angle is calculated using the dot product

### Create molecule(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

Create molecule calls the bond atoms, angle bonds, and angle torsion subroutines described above. This will calculate all the molecule parameters needed for all the energy calculations.

Operation:

1. The bond atoms subroutine is called
2. The angle bonds subroutine is called
3. The angle torsion subroutine is called

### Delete molecule(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

The delete molecule subroutine deallocates all allocated data types in the molecule data type except the atoms. This effectively deletes all calculated data calculated with the create molecule subroutine above. This makes it possible to recalculate the molecule parameters and ultimately the forcefield energy.

Operation:

1. It is checked if there were any CC bonds
   1. If so the molecule%torsion\_angle vector is deallocated
2. The molecule%bonds vector is deallocated
3. The molecule%bonding vector is deallocated
4. The molecule%distance vector is deallocated
5. The molecule%angels vector is deallocated

## Energy functions

### Stretch energy(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

Where A and B are atoms that are bonding, is a constant that depends on the type of bond (CC or CH), is the bond length and is the optimal bond length and depends on the type of bond.

Before the stretch energy function can be called the bonds atoms subroutine (3.1.2.) must be called in advance, otherwise, this function will give an error!

Operation:

1. Based on the type of bond the stretch energy of the bond is calculated
2. The stretch energy of all bonds is summed up and returned

### Bending energy(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule

Where A and B are bonds that are connected with a central atom, is a constant that depends on the type bond types, is the angle in degrees between the two connected bonds and is the optimal angle of the bonds (in this project only SP3 hybridized carbons are used so the angel will be the same for all bond combinations)

Before the bending energy function can be called the angle bond subroutine (3.1.3.) must be called in advance, otherwise, this function will give an error!

Operation:

1. For all connecting bond pairs, the two bond types are checked
2. The bending energy per connected bond pair is calculated
3. The energies of all bending bond pairs are summed and returned

### Torsion energy(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

Where A, B, and C are 3 connecting bonds where B must be a CC bond (in this project), is a constant dependent on the central bond and thus is the same for all torsion angles, n is a constant based on the possible number of torsion angles for the central bond, is the torsion (dihedral) angle in degrees, is a correction of the torsion angle

Before the torsion angle energy function can be called the angle torsion subroutine (3.1.4.) must be called in advance, otherwise, this function will give an error!

Operation:

1. Checked if there is a CC bond otherwise, there are no torsion angles and energy
2. For all torsion angles the torsion energy is calculated
3. The torsion energies are summed and returned

### Electrostatic energy(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

Where i and j are nonbonding atom pairs, are the charges of both atoms depending on the atom element and is the coulomb constate in (Kcal/mol) \* A / e2

Before the electrostatic energy function can be called the bonds atoms subroutine (3.1.2.) must be called in advance, otherwise, this function will give an error!

Operation:

1. For all unique atom pairs, it is checked if they are nonbonding by checking the bonding array
2. Depending on both elements of the atoms the electrostatic energy is calculated
3. The electrostatic energies are summed and returned

### Van der Waals energy(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

Where i and j are nonbonding atom pairs, are van der waals interaction correction factor and is the distance between the nonbonding atom pair in Armstrong (A).

Before the van der Waals energy function can be called the bonds atoms subroutine (3.1.2.) must be called in advance, otherwise, this function will give an error!

Operation:

1. For all unique atom pairs, it is checked if they are nonbonding by checking the bonding array
2. Depending on both elements of the atoms the van der Waals energy is calculated
3. The van der Waals energies are summed and returned

### Forcefield energy(mol)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

Function:

Before the forcefield energy function can be called the create molecule subroutine (3.1.5.) must be called in advance, otherwise, the function will give an error

Operation:

1. All the separate energies are calculated
2. The different energies are summed and the total forcefield energy is returned

## Metropolis subroutines

### Random atom metropolis(mol, r)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule and r is an optional argument that can set the (maximum) radius an atom can be moved in.

The metropolis algorithm works by changing all the atoms coordinates within a certain radius (r) according to:

Where are the new vectors for all atoms, are the old vectors for the atoms, is a random matrix of size atoms X 3 (X, Y, Z) between -1,1 and r is the (maximum) radius the atom movement can take on.

The new atom coordinates can then be used by the metropolis subroutine to calculate the new forcefield energy and compare it to the old conformation of the molecule (atoms).

The atoms of the molecule must be read (or already present) before this subroutine can be called otherwise an error will occur!

Operation:

1. Check if r is passed as an argument
   1. If so r is set to the argument value of r
   2. If not r is set to its default value of 0.0001
2. The matrix q (number of atoms x 3 (xyz)) is filled with random numbers between 0 and 1
   1. This is converted to random, numbers between -1 and 1 by q = (q – 0.5) \* 2
3. The random matrix q is multiplied by r to set the maximum movement of the atoms
4. The new random values are added to the coordinates (xyz) of the atoms

### Metropolis(mol, T, r)

Parameters:  
Mol is a derived molecule type (type (molecule)) see 2.5. Molecule.

The metropolis algorithm works by moving all the atoms at random (within a set radius) and then recalculates the forcefield energy. If the new energy is lower the new coordinates will become the old coordinates and the atoms are again moved at random and the forcefield energy is calculated. If the new forcefield energy is higher there is a change according to the Boltzmann distribution that the new coordinates be accepted. For this calculation, an temperature is needed which can be passed as an argument or left at its default value. If the new coordinates are rejected a set number of times in a row the forcefield energy is believed to be in a (local) minimum. This energy is then printed together with the reduction of the energy compared to the original energy. if r is present it is passed on to the random atom metropolis subroutine.

The total forcefield energy of the new atoms position is compared with the old positions of the energy by:

If the energy difference is lower than 0 () (the new position has lower energy) the new atoms coordinates are accepted and used as the new atom coordinates. If the energy is higher than 0 this means that the new energy is higher than the old energy the new coordinates are accepted or rejected based on the Boltzmann distribution based on the probability:

Where is the probability that the new atom coordinates will be accepted is calculated above, is the Boltzmann constant in Kcal/(mol \* K) and T is the absolute temperature in kelvin (K).

Operation:

1. Check if the temperature (T) is passed as an argument
   1. If so T is set to the passed argument
   2. If not T is set to its default of 293 K
2. The create molecule subroutine is called
3. The forcefield energy of the first conformation is stored in old energy and stating energy (used to compare the new minimum energy)
4. The metropolis algorithm:
   1. The old molecule (including atoms) is stored in an old mol holding variable (type molecule)
   2. The delete molecule subroutine is called
   3. The random atom metropolis subroutine is called
   4. The create molecule subroutine is called
   5. The delta energy is calculated
   6. If the delta energy is negative
      1. The new atom coordinates are accepted
      2. The (new) energy is now stored in the old energy variable and the metropolis algorithm is run again
   7. If the delta energy is greater than 0 (new energy is higher than the old energy)
      1. A random number between 0 and 1 is called and stored in the P
      2. With this, Pa is calculated
      3. If P is smaller than Pa then
         1. The new atoms coordinates are accepted
         2. The (new) energy is now stored in the old energy variable and the metropolis algorithm is run again
      4. If P is bigger than Pa then
         1. The new atom coordinates are rejected
         2. The molecule is set back to the old molecule (previous iteration)
         3. The energy is set back to the old energy (previous iteration)
   8. This is repeated until the new atom coordinates are rejected 1000 in a row
5. The minimized energy and the reduction in energy and energy % are printed

### Minimize energy

The minimize energy subroutine has no arguments as it will read in the atom data. Call metropolis subroutine with or without r and T as specified by the user and can write the optimized conformation of atoms to a file.

Operation:

1. The user is asked for a file ending in .xyz
2. The read atom subroutine is called with the given filename
3. The user is asked if it wants to give specific values for r and T
   1. If y (yes) is answered
      1. The values for r and T are asked
      2. The metropolis subroutine is called with r and T passed as arguments
   2. If n (no) is answered the metropolis subroutine is called without r and T
4. The user is asked if it wants to save the new atom coordinates to a file ending in .xyz
   1. If y (yes) is answered
      1. The file name is asked
      2. The write atoms subroutine is called
      3. The program will stop
   2. If n (no) is answered the program will stop

## Supporting math functions

### Deg to rad(deg) and rad to deg(rad)

Parameters:  
Deg and rad are real numbers

Deg to rad converts degrees to radian. Rad to deg converts radian to degrees. These functions are needed as Fortran uses radian and the used energy function use degrees.

### Cross product(vec1, vec2)

Parameters:  
Vec1 and Vec2 are 3D real number vectors

The cross product has no standard Fortran function so one for 3D vectors needed to be implemented following the function below.

Function:

A and B are 3D vectors.

### Vector length(vector)

Parameters:  
Vector is a 3D vector

To save repeating the same function a function to calculate the length of a vector was implemented using the function below (square root of the dot product).

Function:

A is a 3D vector.

# Interdependencies (flow diagram)

# 

# In and output file format

The in and output file has the .xyz extension. The file contains the number of atoms in the molecule and the element and xyz coordinates of the atoms in Armstrong (A).

The in and output file has the following format:

Number of atoms

(Empty line)

Element x coordinate y coordinate z coordinate

…

## Ch4.xyz

5

C -0.964831852126 -0.691462827481 -0.000000000062

H -0.782365337661 -1.372751578831 0.856342065276

H -1.767889039149 -1.109936546833 -0.640897047230

H -1.275127188280 0.303959353789 0.378974418443

H -0.033945843010 -0.587122537140 -0.594419436427

## C4h10.xyz

14

C -1.232759595443 -1.345606134088 0.014003682233

C -1.419304186087 0.163267645731 -0.115553876451

C -0.098373185034 0.905778676358 0.113621120470

C -0.285908157631 2.414706548084 -0.012559314040

H 0.685718222927 2.925312550872 0.155600623199

H -1.013978600154 2.777408653398 0.744095711972

H -0.652360693659 2.674358918812 -1.028657046622

H -0.501013687135 -1.710139732522 -0.738225444171

H -2.203742162219 -1.856080017810 -0.158223647764

H -0.871572899402 -1.603596997480 1.032400638565

H -2.171961803403 0.497874769384 0.631465269997

H -1.806332754843 0.391689445530 -1.132664723267

H 0.653425145916 0.573099937721 -0.635136706349

H 0.289976528509 0.675578207129 1.129833712228

## C6h12.xyz

18

C -1.3763 0.1297 -0.4173

C -0.6538 1.2401 0.3544

C 0.8504 1.2665 0.0365

H -1.1062 2.2207 0.1202

H -0.7993 1.0861 1.4411

C 1.4054 -0.1227 -0.3217

H 1.0456 1.9600 -0.8024

H 1.3963 1.6791 0.9050

C 0.6302 -1.2503 0.3714

H 1.3611 -0.2678 -1.4182

H 2.4758 -0.1792 -0.0510

C -0.8567 -1.2627 -0.0238

H 1.0900 -2.2260 0.1305

H 0.7183 -1.1325 1.4687

H -1.0218 -1.9616 -0.8644

H -1.4518 -1.6595 0.8199

H -2.4652 0.1900 -0.2391

H -1.2362 0.2856 -1.5043

## C2h6.xyz

8

C 0.269747 0.370902 -0.595646

H 1.386678 0.370902 -0.595646

H -0.073180 1.433897 -0.595646

H -0.072168 -0.099373 -1.549437

C -0.269784 -0.370807 0.595692

H -0.889295 0.299999 1.238952

H -0.911095 -1.228155 0.277520

H 0.559298 -0.777750 1.223884