GFRD IMPLEMENTATION

ParticleKind

ParticleKind defines a class of particles and gathers the information which is specific to one type of particles, but not to each particular particle.

Attributes

- Diffusion constant double m_dc;
- Diameter double m_d;
- Specie string m_species;

Methods

- The constructors
  - ParticleKind();
    By default and with no variables, this constructor is never used and does not do anything. It is just here for unable us to use some structures like map, multimap...
  - ParticleKind(double dc, double d, string species);
    This constructor, that we use in this algorithm, takes two doubles and a string as variable to set up the diffusion constant, the diameter and the name of the particle.

- The access functions : to access the attributes that are private, they simply return the corresponding attribute.
  - double getDC();
  - double getD();
  - string getSpecies();

Scheme

Scheme defines the possible reactions that can occur in the system.

Attributes

- List of reactants list<ParticleKind> m_reactants;
As the reactants should not be ordered, I could have used a set rather than a list. But I had problems
with the set that wanted me to establish a comparator for the class ParticleKind. That is why I used list.

- List of products
  ```
  list<ParticleKind> m_products;
  ```
  The same remarks holds here.

- Constant of the reaction
  ```
  double m_k;
  ```

- Probabilities table: Those probabilities table are used to draw the time steps (for the S-Table) and to move the particles later on (for the F- and G- Table).
  ```
  map<double, map<double,double>> m_STable;
  map<double, map<double, map<double,double>>> m_FTable;
  map< double, map<double, map<double, map<double,double>>>> m_GTable;
  ```
  I have used a map for those table because it's an ordered data structure. It is then easy to access the element of the table. Also I didn't use multimap because each value of r0, deltaT... appears only once (I am computing them by iteration).

**Methods**

- The constructors
  ```
  Scheme();
  ```
  By default and with no variables, this constructor is never used and does not do anything. It is just here for enabling us to use some structures like map, multimap...
  ```
  Scheme(list<ParticleKind> reactants, list<ParticleKind> products, double k);
  ```
  This constructor takes as variable two lists of ParticleKind and a double, to set up the reactants' and products' lists and the reaction constant. It creates the table of probabilities by computation using the functions that are in functions.cpp. This file, function.cpp, should actually be part of the Scheme class, but for the sake of clarity, I wrote all the mathematical functions in a separated file.
  ```
  Scheme(string fileName, list<ParticleKind> reactants, list<ParticleKind> products, double k);
  ```
  This constructor takes a string as a supplementary variable, so that the probabilities tables are constructed from a file, which is much faster. This is particularly useful when one wants to run the program several times with the same values for the parameters.

- The access functions: to access the attributes that are private, they simply return the corresponding attribute.
  ```
  list<ParticleKind> getReactants();
  list<ParticleKind> getProducts();
  double getK();
  map<double, map<double,double>> getSTable();
  map<double, map<double, map<double,double>>> getFTable();
  map< double, map<double, map<double, map<double,double>>>> getGTable();
  ```
set<string> getReactantsStrings();
This method is very similar to an access function given that it just returns some information about the scheme. In this case, it returns the list of the names of the reactants.

Particles

The particles are the objects that we are going to move.

Attributes

- their coordinates
  - double m_x;
  - double m_y;
- their kind ParticleKind m_kind;
- a list of their neighbors multimap<double, int> m_neighbors;
  Here I used multimap because several neighbors can be at the same distance from the particle.
- an unique index int m_index;

Functions

- The constructors
  - Particle();
    By default and with no variables, this constructor is never used and does not do anything. It is just here for unable us to use some structures like map, multimap...
  - Particle(gsl_rng* rng, ParticleKind kind, int index);
    This constructor takes as variable a random number generator, a ParticleKind and an index. It creates a particle with the corresponding index and kind, and with random coordinates.
  - Particle(double x, double y, ParticleKind kind, int index);
    This constructor takes as variable two doubles, for the coordinates of the particle, a ParticleKind for its kind and an integer for its index.
- The access functions : to access the attributes that are private, they simply return the corresponding attribute.
  - double getX();
  - double getY();
  - multimap<double, int> getNeighbors();
  - int getIndex();
  - double getDiameter();
    This method is very similar to an access function given that it just returns some information about the particle that is actually hidden is the ParticleKind attribute.
- **double** `getDC()`;
This method is very similar to an access function given that it just returns some information about the particle that is actually hidden is the `ParticleKind` attribute.

- **string** `getSpecies()`;
This method is very similar to an access function given that it just returns some information about the particle that is actually hidden is the `ParticleKind` attribute.

- **void** `move(double x, double y);`
This method permits to move the particle of a vector \((x,y)\).

- **void** `moveto(double x, double y);`
This method permits to move the particle to a new location \((x,y)\).

- **void** `updateNeighbors(map<int, Particle> particles);`
This method permits to update the neighbor list of a particle \(A\). It goes through every particle and for each particle \(B\), it computes the distance from \(A\) to \(B\) and place it in the neighbor list of the particle \(A\). In \(O(n^2)\) this function could be optimized.

### AdaptativeQuad, Quad and HeaderFiles

Those files have been taken from the solution of some exercises of the course «Advanced Computational Science» given by the Professor Koumoutsakos.

They are used to compute the integral of different functions in the `functions.cpp` file. Previously I used GSL to do that, but for slowly convergent function I had problems which were relevant for my algorithm. So I decided to use another implementation, which solved the problem.

However it takes a lot of time to compute these slowly convergent integrals.

This function could also be optimized by a better utilization of the GSL library.

### Main function

Framework

We create first the framework : a random generator, the different types of possible particles (`ParticleKind`) and the possible reactions (Scheme) of the system that we are modeling.

```cpp
// Setting up the random generator
gsl_rng_env_setup();
gsl_rng* rng = gsl_rng_alloc(gsl_rng_mt19937);
gsl_rng_set(rng, time(NULL));

//Creation of the different possible particles in the system
ParticleKind A = ParticleKind(10,0.1, "A"); // (diffusion constant,diameter,species)
ParticleKind B = ParticleKind(10,0.05, "B");
ParticleKind C = ParticleKind(10,0.05, "C");

// Creation of a list of the possible reactions (Schemes).
list<ParticleKind> prod;
prod.push_back(A);
```
Initialization

Then we create as much particles as necessary. As a first simple example, we create one unique particle C in the middle of the domain [0,20] * [0,20].

```
int nc = 1; //nc of particles of type C
map<int,Particle> particles;
for (int i=0; i < nc; i++) {
    particles[index+i] = Particle(10, 10, C, index+i);
} index += nc;
```

We initialize the time.

```
double t = 0;
```

And the main loop on the time t begins:

```
while (t<2) {
```

Update of the neighbor list

For each particle:

```
updateNeighbors(particles);
```

Computation of the deltaTmax

For each particle

```
multimap<double,int> neighbors = (*it).second.getNeighbors();
```

If the neighbor list contains more than 1 element, we take the second neighbor and we compute the corresponding deltaT.

```
multimap<double,int>::iterator itt = neighbors.begin();
```
// Iterator to the second neighbor.
double deltaT = pow((*itt).first/H,2)/(6*(*it).second.getDC());

We keep the smallest deltaT.

if (deltaT < deltaTmax) {
    deltaTmax = deltaT;
    deltaRmax = (*itt).first;
}

If the neighbor list contains only one element, we take the distance to the closest neighbor to compute deltaT.

If the particle has no neighbor, deltaTmax is set at an initial default value, which is big because in the other cases, we want to find the minimum.

double deltaTmax = 100;

Corresponding to deltaTmax, we compute also deltaRmax, the distance from the particle to its second neighbor which one can find easily in the neighbor list given that it is the key of the map.

deltaRmax = (*itt).first;

Creation of a list of possible reactions

For each particle, we look at its closest neighbor (the only one which it can react with). For this couple, we go through all schemes.

for (list<Scheme>::iterator itt = Rs.begin(); itt != Rs.end(); itt++) {
    set<string> reactants;
    set<int> reactantsIndex;
    reactants.insert((*it).second.getSpecies());
    reactantsIndex.insert((*it).first);

    if ((*itt).getReactantsStrings() == reactants) {
        reactions[reactantsIndex] = (*itt);
    }

    if the couple of particles is away from a distance smaller than deltaRmax and is a couple a reactants of one of the schemes, which means that they could react, we add the indexes and the corresponding scheme in the list of possible reactions.

    We make a special case if the particle has no neighbor, because we cannot access the first neighbor.

Computation of deltaTnu for every possible reaction
We go through the list of possible reactions. For each reaction, we draw a random number: \( \zeta \) in the monomolecular case and \( Z \) in the case of the bimolecular reactions.

Given this random number, we can find \( \Delta T_{\nu} \), thanks a formula in the monomolecular case, and thanks the \( \text{STable} \) for the bimolecular case.

```c++
multimap<double, set<int>> > deltaTnu;

for (map<set<int>, Scheme>::iterator it=reactions.begin(); it!=reactions.end(); it++) {
    set<int> Rnu = (*it).first;
    Scheme scheme = (*it).second;

    // Monomolecular case
    if (Rnu.size() == 1) {
        double \zeta = gsl_rng_uniform(rng);
        double deltaT = -\log(1-\zeta)/scheme.getK();
        deltaTnu.insert(pair<double, set<int>>>(deltaT, Rnu));
    }

    // Bimolecular case
    else {
        Particle reactant1 = (*particles.find(*(Rnu.begin()))).second;
        Particle reactant2 = (*particles.find(*(Rnu.begin()+1))).second;
        multimap<double, int> neighbor = reactant1.getNeighbors();
        double \( r_0 \) = (*(neighbor.begin())).first;

        map<double, map<double, double>> STable = scheme.getSTable();
        Parameters p;
        p.k = scheme.getK();
        p.D = reactant1.getDC() + reactant2.getDC();
        p.sigma = (reactant1.getDiameter()+reactant2.getDiameter())/2;
        p.r0 = \( r_0 \);

        double \( Z \) = gsl_rng_uniform(rng);
        double test = 1-S(100000, \&p);
        double deltaT;
        map<double, double> Ztmap;
        if (Z <= test) {
            if (STable.lower_bound(r0) == STable.end()) {
                Ztmap = (*STable.rbegin()).second;
            } else {
                Ztmap = (*STable.lower_bound(r0)).second;
            }
            if (Ztmap.lower_bound(Z) == Ztmap.end()) {
                deltaT = (*Ztmap.rbegin()).second;
            } else {
                deltaT = (*Ztmap.lower_bound(Z)).second;
            }
        }
        deltaTnu.insert(pair<double, set<int>>>(deltaT, Rnu));
    }
}
```
**Final deltaT and firing reaction**

We can compute the final deltaT as the minimum of the \( \delta T_{nu} \) and \( \delta T_{max} \). The firing reaction is the one, if it exists, that corresponds to the \( \delta T_{nu} \).

```cpp
double deltaT = min((*deltaTnu.begin()).first, deltaTmax);
set<int> firingReaction;
if (deltaT == (*deltaTnu.begin()).first) {
    firingReaction = (*deltaTnu.begin()).second;
}
```

**Moving and reacting**

For each possible reaction, we move the particles and create and delete the particles involve in the firing reaction.

```cpp
set<int> Rnu = (*it).first;
Scheme scheme = (*it).second;

// monomolecular case
if (Rnu.size() == 1) {
    Particle reactant = ((*particles.find(*Rnu.begin())).second);
    double DC = reactant.getDC();
    double zetaX = gsl_ran_gaussian(rng, 1);
    double zetaY = gsl_ran_gaussian(rng, 1);
    if (Rnu == firingReaction) {
        list<ParticleKind> products = scheme.getProducts();
        for (list<ParticleKind>::iterator it = products.begin(); it != products.end(); it++) {
            double r = (*it).getD()/2;
            double x = reactant.getX() + sqrt(2*DC*deltaT)*zetaX;
            double xnew = x + (((index%2)-0.5)/0.5)*r/sqrt(2);
            if (xnew < 0) {
                xnew += 20;
            }
            if (xnew > 20) {
                xnew -= 20;
            }
            double y = reactant.getY() + sqrt(2*DC*deltaT)*zetaY;
            double ynew = y + (((index%2)-0.5)/0.5)*r/sqrt(2);
            if (ynew < 0) {
                ynew += 20;
            }
            if (ynew > 20) {
                ynew -= 20;
            }
            particles.insert(pair<int, Particle> (index, Particle(xnew, ynew, *it, index)));
            index++;
        }
        particles.erase(*Rnu.begin());
    } else {
        (*particles.find(*Rnu.begin())).second.move(sqrt(2*DC*deltaT)*zetaX, sqrt(2*DC*deltaT)*zetaY);
    }
}
```

// bimolecular case
else {
    Particle reactant1 = (*(particles.find(*(Rnu.begin())))).second;
    Particle reactant2 = (*(particles.find(*(Rnu.rbegin())))).second;
    double DC1 = reactant1.getDC();
    double DC2 = reactant2.getDC();
    double x1 = reactant1.getX();
    double x2 = reactant2.getX();
    double y1 = reactant1.getY();
    double y2 = reactant2.getY();
    double xR = x1*sqrt(DC2/DC1) + x2*sqrt(DC1/DC2);
    double yR = y1*sqrt(DC2/DC1) + y2*sqrt(DC1/DC2);
    double zetaX = gsl_ran_gaussian(rng, 1);
    double zetaY = gsl_ran_gaussian(rng, 1);
    double xRnew = xR + sqrt(2*(DC1+DC2)*deltaT)*zetaX;
    if (xRnew < 0) {
        xRnew += 20;
    }
    if (xRnew > 20) {
        xRnew -= 20;
    }
    double yRnew = yR + sqrt(2*(DC1+DC2)*deltaT)*zetaY;
    if (yRnew < 0) {
        yRnew += 20;
    }
    if (yRnew > 20) {
        yRnew -= 20;
    }
}

if (Rnu == firingReaction) {
    ParticleKind product = *(scheme.getProducts()).begin();
    particles.insert(pair<int, Particle> (index, Particle(xRnew, yRnew, product, index)));
    index++;
    particles.erase(*Rnu.begin());
    particles.erase(*Rnu.rbegin());
} else {
    double R = gsl_rng_uniform(rng);
    double Theta = gsl_rng_uniform(rng);
    double r0 = sqrt(pow(x1-x2, 2)+pow(y1-y2, 2));
    double theta0 = asin((y2-y1)/r0);
    map<double, map<double, map<double, double, double> > > > FTable = scheme.getFTable();
    map<double, map<double, map<double, double, double> > > > GTable = scheme.getGTable();
    map<double, double> r0Table1;
    map<double, double> deltaTTable1;
    double r;
    if (FTable.lower_bound(r0) == FTable.end()) {
        r0Table1 = (*FTable.rbegin()).second;
    } else {
        r0Table1 = (*FTable.lower_bound(r0)).second;
    }
    if (r0Table1.lower_bound(deltaT) == r0Table1.end()) {
        deltaTTable1 = (*r0Table1.rbegin()).second;
    } else {
        deltaTTable1 = (*r0Table1.lower_bound(deltaT)).second;
    }
    if (deltaTTable1.lower_bound(R) == deltaTTable1.end()) {
        r = (*deltaTTable1.rbegin()).second;
    }
```cpp
else {
    r = (*deltaTT.tlower_bound(R)).second;
}

map<double, map<double, double> > r0Table;
map<double, double> rTable;
double theta;
if (GTable.tlower_bound(r0) == GTable.end()) {
    r0Table = (*GTable.rbegin()).second;
} else {
    r0Table = (*GTable.tlower_bound(r0)).second;
}
if (r0Table.tlower_bound(Theta) == r0Table.end()) {
    theta = (*r0Table.rbegin()).second;
} else {
    theta = (*rTable.tlower_bound(Theta)).second;
}

double deltaX = r*cos(theta0+theta);
double deltaY = r*sin(theta0+theta);

double x2new = 0.5*(deltaX + sqrt(DC2/DC1)*xRnew);
if (x2new < 0) {
    x2new += 20;
} if (x2new > 20) {
    x2new -= 20;
}
double y2new = 0.5*(deltaY + sqrt(DC2/DC1)*yRnew);
if (y2new < 0) {
    y2new += 20;
} if (y2new > 20) {
    y2new -= 20;
}

double x1new = 0.5*(-deltaX + sqrt(DC1/DC2)*xRnew);
if (x1new < 0) {
    x1new += 20;
} if (x1new > 20) {
    x1new -= 20;
}
double y1new = 0.5*(-deltaY + sqrt(DC1/DC2)*yRnew);
if (y1new < 0) {
    y1new += 20;
} if (y1new > 20) {
    y1new -= 20;
}

((*(particles.find(*Rnu.begin())))).second.moveto(x1new,y1new);
```
When a particle is going away from the domain, we put it back on the other side of the domain, using a modulo on both coordinates.

**Update the time**

At the end, we increase the time $t$ of $\Delta T$.

$$t += \Delta T;$$

**Example**

In this example, I have chosen to put initially a single particle of type C in the middle of the domain.

We can see that the particle dissociate in new particles A and B.

Then those particle diffuse a bit, until the reaction time $\Delta T_{nu}$ becomes smaller than the $\Delta T_{max}$. At this moment, they react, giving birth to a new particle of type C.

\[
\begin{align*}
\text{t} &= 0 \\
\text{---------} \\
\text{Particle 0 of type C at position } 10, 10 \\
\Delta T_{max} &= 100 \\
\Delta R_{max} &= 20 \\
\text{reactions contains 1 elements :} \\
0 &\text{ C} \\
\Delta T_{nu} &= 0.364795 \\
\Delta T &= 0.364795 \\
\text{reaction between :0} \\
\text{t} &= 0.364795 \\
\text{---------} \\
\text{Particle 1 of type A at position } 7.90761, 9.05345 \\
2 &\text{ 0.075} \\
\text{Particle 2 of type B at position } 7.85458, 9.00042 \\
\end{align*}
\]
\begin{verbatim}
1 0.075

deltaTmax = 1.04167e-05
deltaRmax = 0.075

reactions contains 1 elements :
  1 2 A B

deltaTnu
0.04

deltaT
1.04167e-05

reaction between :

  t = 0.364806

-------------

Particle 1 of type A at position 7.35359, 8.9171
2 1.075

Particle 2 of type B at position 8.40393, 9.14603
1 1.075

deltaTmax = 0.00214005
deltaRmax = 1.075

reactions contains 1 elements :
  1 2 A B

deltaTnu
0.04

deltaT
0.00214005

reaction between :

  t = 0.366946

-------------

Particle 1 of type A at position 7.63355, 8.04822
2 1.075
\end{verbatim}
Particle 2 of type B at position 8.00842, 9.05574
\[ \text{deltaTmax} = 0.00214005 \]
\[ \text{deltaRmax} = 1.075 \]

reactions contains 1 elements:
\[ 1 \, 2 \, A \, B \]

\[ \text{deltaTnu} \]
0.02

\[ \text{deltaT} \]
0.00214005

reaction between:

\[ t = 0.369086 \]

-------------

Particle 1 of type A at position 8.16215, 9.04797
\[ \text{deltaTmax} = 0.00214005 \]
\[ \text{deltaRmax} = 1.075 \]

reactions contains 1 elements:
\[ 1 \, 2 \, A \, B \]

\[ \text{deltaTnu} \]
0.04

\[ \text{deltaT} \]
0.00214005

reaction between:

\[ t = 0.371226 \]

-------------

Particle 1 of type A at position 7.14447, 8.97564
\[ \text{deltaTmax} = 0.00214005 \]
\[ \text{deltaRmax} = 1.075 \]

reactions contains 1 elements:
\[ 1 \, 2 \, A \, B \]

\[ \text{deltaTnu} \]
0.04

\[ \text{deltaT} \]
0.00214005

reaction between:

\[ t = 0.371226 \]

-------------

Particle 1 of type A at position 7.14447, 8.97564
\[ \text{deltaTmax} = 0.00214005 \]
\[ \text{deltaRmax} = 1.575 \]
Particle 2 of type B at position 8.4679, 8.12173
\[ \Delta T_{\text{max}} = 0.00459375 \]
\[ \Delta R_{\text{max}} = 1.575 \]

reactions contains 1 elements:
1 2 A B

deltaTnu
0.01

deltaT
0.00459375

reaction between:

t = 0.37582
-------------

Particle 1 of type A at position 6.88189, 7.99003
\[ \Delta T_{\text{max}} = 0.00797338 \]
\[ \Delta R_{\text{max}} = 2.075 \]

reactions contains 1 elements:
1 2 A B

deltaTnu
0.01

deltaT
0.00797338

reaction between:

t = 0.383793
-------------

Particle 1 of type A at position 7.57977, 6.73948
Particle 2 of type B at position 7.94879, 9.2879

\[ \delta T_{\text{max}} = 0.0122789 \]
\[ \delta R_{\text{max}} = 2.575 \]

Reactions contain 1 element:

\[ 1 \ 2 \ A \ B \]

\[ \delta T_{\text{nu}} = 0.04 \]
\[ \delta T = 0.0122789 \]

Reaction between:

\[ t = 0.396072 \]

Particle 1 of type A at position 9.05059, 6.3305

Particle 2 of type B at position 6.72788, 8.3456

\[ \delta T_{\text{max}} = 0.0175104 \]
\[ \delta R_{\text{max}} = 3.075 \]

Reactions contain 1 element:

\[ 1 \ 2 \ A \ B \]

\[ \delta T_{\text{nu}} = 0.09 \]
\[ \delta T = 0.0175104 \]

Reaction between:

\[ t = 0.413582 \]
Particle 1 of type A at position 7.62191, 9.3568
2 3.075

Particle 2 of type B at position 9.10211, 6.6615
1 3.075

deltaTmax = 0.0175104
deltaRmax = 3.075

reactions contains 1 elements:
1 2 A B

deltaTnu
0.08

deltaT
0.0175104

reaction between:

t = 0.431093
----------

Particle 1 of type A at position 6.91361, 7.08365
2 3.575

Particle 2 of type B at position 10.1611, 8.57856
1 3.575

deltaTmax = 0.0236678
deltaRmax = 3.575

reactions contains 1 elements:
1 2 A B

deltaTnu
0.09

deltaT
0.0236678

reaction between:

t = 0.454761
----------
Particle 1 of type A at position 7.35139, 7.89708
2 3.075

Particle 2 of type B at position 10.2381, 6.83756
1 3.075

deltaTmax = 0.0175104
deltaRmax = 3.075

reactions contains 1 elements:
1 2 A B

deltaTnu
0.09

deltaT
0.0175104

reaction between:

t = 0.472271

-------------------

Particle 1 of type A at position 9.76608, 5.57202
2 3.075

Particle 2 of type B at position 8.08751, 8.14846
1 3.075

deltaTmax = 0.0175104
deltaRmax = 3.075

reactions contains 1 elements:
1 2 A B

deltaTnu
0.08

deltaT
0.0175104

reaction between:

t = 0.489781
Particle 1 of type A at position 10.9934, 6.4278, 3.575
Particle 2 of type B at position 7.45763, 6.95578, 3.575

\[ \Delta T_{\text{max}} = 0.0236678 \]
\[ \Delta R_{\text{max}} = 3.575 \]

reactions contains 1 elements:
1 2 A B

\[ \Delta T_{\nu} \]
0.09

\[ \Delta T \]
0.0236678

reaction between:

\[ t = 0.513449 \]

------------

Particle 1 of type A at position 7.52299, 4.62744, 3.575
Particle 2 of type B at position 8.9891, 7.88798, 3.575

\[ \Delta T_{\text{max}} = 0.0236678 \]
\[ \Delta R_{\text{max}} = 3.575 \]

reactions contains 1 elements:
1 2 A B

\[ \Delta T_{\nu} \]
0.15

\[ \Delta T \]
0.0236678

reaction between:

------------
t = 0.537117
-------------

Particle 1 of type A at position 6.46983, 5.52741
2 4.075

Particle 2 of type B at position 10.2763, 6.98216
1 4.075

deltaT_{max} = 0.0307512
deltaR_{max} = 4.075

reactions contains 1 elements:
1 2 A B

deltaT_{nu}
0.11

deltaT
0.0307512

reaction between:

-------------

Particle 1 of type A at position 8.35467, 3.7838
2 4.575

Particle 2 of type B at position 7.02782, 8.16216
1 4.575

deltaT_{max} = 0.0387604
deltaR_{max} = 4.575

reactions contains 1 elements:
1 2 A B

deltaT_{nu}
0.11

deltaT
0.0387604

reaction between:
\[ t = 0.606629 \]

-------------------

Particle 1 of type A at position 8.42127, 4.69358
2 4.075

Particle 2 of type B at position 5.77821, 7.79517
1 4.075

deltaT_{\text{max}} = 0.0307512
\text{deltaR}_{\text{max}} = 4.075

reactions contains 1 elements:
1 2 A B

deltaT_{\text{n}}
0.15

deltaT
0.0307512

reaction between:

-------------------

\[ t = 0.63738 \]

-------------------

Particle 1 of type A at position 7.49911, 4.14977
2 4.075

Particle 2 of type B at position 6.31727, 8.04963
1 4.075

deltaT_{\text{max}} = 0.0307512
\text{deltaR}_{\text{max}} = 4.075

reactions contains 1 elements:
1 2 A B

deltaT_{\text{n}}
0.11

deltaT
0.0307512

reaction between:
t = 0.668131
--------------
Particle 1 of type A at position 4.43965, 7.11624
2 4.575

Particle 2 of type B at position 8.43994, 4.89626
1 4.575

deltaTmax = 0.0387604
deltaRmax = 4.575

reactions contains 1 elements :
1 2 A B

deltaTnu
0.01
deltaT
0.01

reaction between : 1 2

t = 0.678131
--------------

Particle 3 of type C at position 12.4378, 12.5966

deltaTmax = 100
deltaRmax = 20

reactions contains 1 elements :
3 C

deltaTnu
0.67226
deltaT
0.67226

reaction between : 3

t = 1.35039
--------------
Particle 4 of type A at position 16.8872, 9.05369
5 0.075

Particle 5 of type B at position 16.9402, 9.10672
4 0.075

deltaTmax = 1.04167e-05
deltaRmax = 0.075

reactions contains 1 elements:
4 5 A B

deltaTnu
0.04

deltaT
1.04167e-05

reaction between:

t = 1.3504

-------------

Particle 4 of type A at position 6.96373, 8.79578
5 0.575

Particle 5 of type B at position 6.84128, 9.35759
4 0.575

deltaTmax = 0.000612269
deltaRmax = 0.575

reactions contains 1 elements:
4 5 A B

deltaTnu
0.04

deltaT
0.000612269

reaction between:

t = 1.35101

-------------
Particle 4 of type A at position 7.21777, 8.24964  
5 1.575

Particle 5 of type B at position 6.77435, 9.76093  
4 1.575

deltaTmax = 0.00459375  
deltaRmax = 1.575

reactions contains 1 elements :
4 5 A B

deltaTnu  
0.01

deltaT  
0.00459375

reaction between :

t = 1.35561  
-----------

Particle 4 of type A at position 6.34028, 9.13446  
5 1.075

Particle 5 of type B at position 6.9231, 8.23116  
4 1.075

deltaTmax = 0.00214005  
deltaRmax = 1.075

reactions contains 1 elements :
4 5 A B

deltaTnu  
0.02

deltaT  
0.00214005

reaction between :

t = 1.35775
Particle 4 of type A at position 7.43241, 8.52031
5 1.575

Particle 5 of type B at position 5.87269, 8.73912
4 1.575

deltaTmax = 0.00459375
deltaRmax = 1.575

Reactions contains 1 elements:
4 5 A B

deltaTnu
0.01

deltaT
0.00459375

reaction between:

----------

t = 1.36234
----------

Particle 4 of type A at position 5.95888, 8.2895
5 2.075

Particle 5 of type B at position 7.624, 9.52765
4 2.075

deltaTmax = 0.00797338
deltaRmax = 2.075

Reactions contains 1 elements:
4 5 A B

deltaTnu
0.04

deltaT
0.00797338

reaction between:
\[ t = 1.37031 \]

-------------

Particle 4 of type A at position 7.88191, 9.14373
5 2.575

Particle 5 of type B at position 5.30692, 9.14943
4 2.575

deltaTmax = 0.0122789
deltaRmax = 2.575

reactions contains 1 elements:
4 5 A B

deltaTnu
0.08

deltaT
0.0122789

reaction between:

\[ t = 1.38259 \]

-------------

Particle 4 of type A at position 5.88944, 6.92723
5 3.575

Particle 5 of type B at position 7.81436, 9.93976
4 3.575

deltaTmax = 0.0236678
deltaRmax = 3.575

reactions contains 1 elements:
4 5 A B

deltaTnu
0.15

deltaT
0.0236678

reaction between:
\( t = 1.40626 \)

-------------------

Particle 4 of type A at position 5.69676, 8.18519
5 3.075

Particle 5 of type B at position 8.6983, 8.85333
4 3.075

\( \delta T_{max} = 0.0175104 \)
\( \delta R_{max} = 3.075 \)

reactions contains 1 elements :
4 5 A B

\( \delta T_{nu} \)
0.08

\( \delta T \)
0.0175104

reaction between :

\( t = 1.42377 \)

-------------------

Particle 4 of type A at position 5.78125, 9.46675
5 3.075

Particle 5 of type B at position 7.27337, 6.77803
4 3.075

\( \delta T_{max} = 0.0175104 \)
\( \delta R_{max} = 3.075 \)

reactions contains 1 elements :
4 5 A B

\( \delta T_{nu} \)
0.08

\( \delta T \)
0.0175104

reaction between :
\[ t = 1.44128 \]

Particle 4 of type A at position 7.62459, 7.59619, 3.075

Particle 5 of type B at position 5.28413, 9.59065, 4.3.075

\[ \text{deltaTmax} = 0.0175104 \]
\[ \text{deltaRmax} = 3.075 \]

reactions contains 1 elements:
\[ 4 \quad 5 \quad A \quad B \]

\[ \text{deltaTnu} \]
\[ 0.09 \]

\[ \text{deltaT} \]
\[ 0.0175104 \]

reaction between:

\[ t = 1.45879 \]

Particle 4 of type A at position 8.38198, 8.56044, 4.075

Particle 5 of type B at position 4.31536, 8.29917, 4.4.075

\[ \text{deltaTmax} = 0.0307512 \]
\[ \text{deltaRmax} = 4.075 \]

reactions contains 1 elements:
\[ 4 \quad 5 \quad A \quad B \]

\[ \text{deltaTnu} \]
\[ 0.15 \]

\[ \text{deltaT} \]
\[ 0.0307512 \]
reaction between:

t = 1.48954
-----------

Particle 4 of type A at position 3.78801 , 6.90985 4 4.575

Particle 5 of type B at position 7.93532 , 8.84125 4 4.575

deltaTmax = 0.0387604
deltaRmax = 4.575

reactions contains 1 elements:
4 5 A B
deltaTnu
0.01
deltaT
0.01

reaction between : 4 5

t = 1.49954
-----------

Particle 6 of type C at position 11.3372 , 16.0779

deltaTmax = 100
deltaRmax = 20

reactions contains 1 elements:
6 C
deltaTnu
0.508316
deltaT
0.508316

reaction between : 6

t = 2.00786
-----------
Particle 7 of type A at position 11.5826, 16.4957

Particle 8 of type B at position 11.5295, 16.4426