**1. Описание исследуемой системы**

Система представляет из себя N частиц в 3D-пространстве, с потенциалом взаимодействия:

(1)

Здесь α и β – размерные константы, характеризующие параметры системы.

С помощью несложных преобразований можно перейти к новым, безразмерным переменным:

<описать процесс обезразмеривания>:

(1\*)

Первая сумма соответствует внешнему полю с параболическим потенциалом, «прижимающему» частицы к центру; вторая сумма – отталкивание между частицами порядка .

В присутствии только второй суммы частицы, очевидно, просто разлетелись бы. При наличии внешнего, сдерживающего потенциала (первая сумма в (1\*)), частицы образуют некоторое устойчивое образование конечных размеров (кластер).

В первой сумме суммирование производится по индексу k, и в сумме всего N членов. Во второй сумме суммирование производится по всем различным парам (i,k), где индексы суммируют одинаковые частицы, т.е. (i,k) == (k,i). Каждый индекс также пробегает значения от 1 до N, и всего в сумме N\*(N-1)/2 слагаемых. Двойка нужна для того, чтобы не посчитать кажду пару два раза, т.к. в простой сумме с N\*(N-1) слагаемых каждая пара посчитана дважды (например, пара (3,2) и (2,3)). Для исключения недоразумений вторую сумму в (1\*) можно также записать как:

**2. Исследовательский шаг №1. Поиск минимумов потенциальной энергии.**

Поиск минимума в выражении (1\*) сам по себе даёт уже неплохие представления о структуре системы.

Мы собираемся, для начала, ввести компьютерное описание для системы; затем найти минимумы функции (1\*) методом градиентного спуска.

**Компьютерное описание системы.**

На данном этапе будем описывать систему с помощью 3N переменных конечной точности. <указать точность здесь>: N троек координат x, y, z.

Тогда:

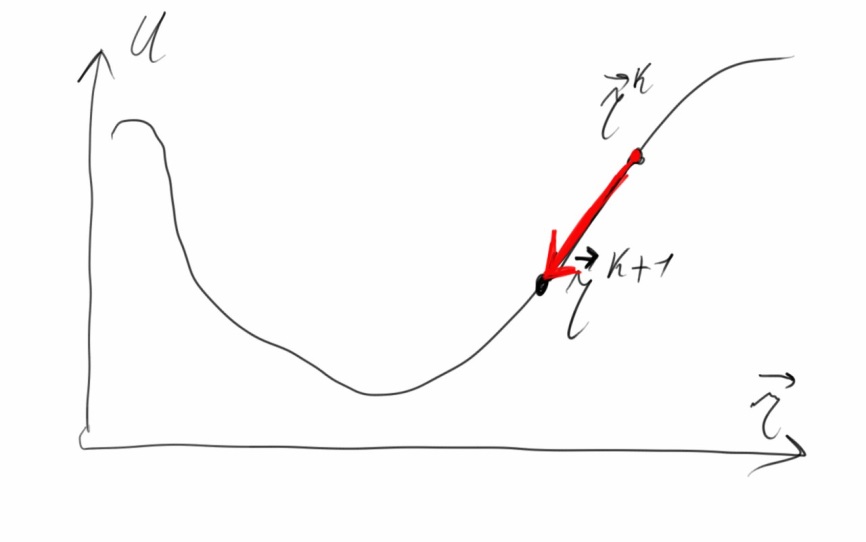
- длина радиус-вектора частицы;

– расстояние между i-й и k-й частицей.

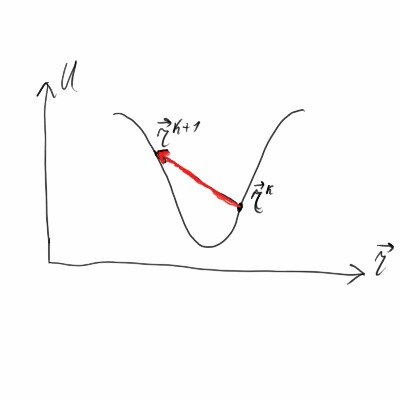
**Метод градиентного спуска** коротко можно описать так: вначале задаётся (произвольное) начальное значение всех переменных; затем запускается итерационный процесс, на каждом шаге которого переменным присваивается новое значение:

Здесь , а верхние индексы нумеруют номер итерации в итерационном процессе. λ – константа, которая характеризует «скорость спуска», и может, в общем случае, зависеть как от номера k итерации, так и от номера частицы\переменной, т.е. λ может быть векторной величиной размерности 3N.

Очевидно, что при таком процессе указывает в сторону минимума функции U:



При слишком больших λ возможны, однако, и такие ситуации:



Т.е. направление движения итерационного процесса правильное, но слишком большая амплитуда, в результате градиентный спуск «перескакивает» через положение минимума, и приводит только к увеличению потенциальной энергии. Поскольку нельзя заранее знать характер самой функции U, понятие «малости» параметра λ тоже неопределено, поэтому заранее знать, какой параметр выбрать, нельзя.

Это является **первой сложностью** расчёта минимума потенциальной энергии. Существуют разные способы преодоления этой сложности: выбор λ в качестве вектора, динамическое изменение параметра λ с каждой операцией, метод «наискорейшего спуска», при котором параметр λ подбирается таким, чтобы изменение потенциальной энергии за итерацию было наибольшим (и отрицательным!).

В нашей работе используется следующий способ: до вычисления k+1’го шага итерации мы уменьшаем параметр λ (деля его на всё большее число), пока полученная после итерации потенциальная энергия не станет меньше исходной. Таким образом, наш метод градиентного спуска **всегда** приводит к уменьшению потенциальной энергии (или, точнее, к её неувеличению).

Остаются следующие проблемы:

1. Для некоторых значений количества частиц N исходные данные для градиентного спуска приводят к колебанию системы между двумя симметричными состояниями с одинаковыми значениями потенциальной энергии (Возможно, программный баг)
2. Таким способом можно найти только **локальный минимум** потенциальной энергии.

Проблема 2 является фундаментальной проблемой для любого вычислительного алгоритма. Мне не известно алгоритмов, которые могли бы с уверенностью опеределить, является ли данный минимум абсолютным или всего лишь одним из локальных минимумов. В дальнейшем планируется дополнительно проверить полученные результаты на «абсолютность». Алгоритм этому я пока ещё не придумал.

Известны случаи, когда для похожих систем (нанокластеры в двухмерном пространстве с отталкиванием 1-го и 3-го порядка) для некоторых значений N «симметричные» картины для потенциальных минимумов соответствовали не самому маленькому локальному минимуму. Самый маленький же (из известных) обладал несколько меньшей симметрией. Таким образом, опираясь лишь на «красивость» получаемых результатов, нельзя судить об абсолютности данного минимума. <Здесь нужно вставить ссылку на соотв. работу Лозовика>

Программная часть

Все вычисления я производил на платформе R.

Ниже будут приведены описания основных функций (вместе с, собственно, кодом) и процесса получения результатов.

1. **Структура данных.**

В текущем разделе я использовал следующие структуры для хранения данных:

Вся информация о системе хранится в матрице r размерами 3\*N. Количество частиц системы хранится в ней же, и его можно легко достать с помощью команды R:

N<-ncol(r)

Три строки матрицы соответствуют координатам x,y,z, а столбцы – разным частицам.

Вычисленные потенциальные минимумы (а точнее, всё те же матрицы r, из которых можно посчитать потенциальную энергию в данной точке) хранятся в той же форме, что и r, на диске, в папке R working directory/data\_init, в файлах вида N.csv

1. **Функция init()**

init<-function(N){ ##initializes matrix 3 times N with correct rownames and random values

r<-rbind(rnorm(N),rnorm(N),rnorm(N))

rownames(r)<-(c("x","y","z"))

r

}

Функция init() принимает в качестве аргумента только целое число N – количество частиц.

Функция генерирует рандомный массив из 3N чисел и возвращает их в виде матрицы 3\*N.

r<-rbind(rnorm(N),rnorm(N),rnorm(N)): команда rbind связывает несколько векторов в матрицу путём склеивания по горизонтали, и записывает в переменную r.

rnorm(N) – возвращает вектор из N (псевдо-)рандомных чисел. Вставить описание процесса получения случайных чисел.

rownames(r)<-(c("x","y","z")): сугубо косметическая правка. Добавляет имена к строкам матрицы r.

1. **Функция rad()**

rad<-function(r){ ##calculates the distance from particle k to the beginning of the coordinates

N<-ncol(r)

rad<-rep(0,times=N)

for(k in 1:N){

rad[k]<-(r[1,k]^2+r[2,k]^2+r[3,k]^2)^(0.5)

}

rad

}

rad() принимает на вход матрицу r, и высчитывает вектор длин радиус-векторов каждой частицы. Функция возвращает вектор rad длиной R, k-й элемент которого соответствует расстоянию от k-й частицы до начала координат:

N<-ncol(r) – просто записываем в переменную N количество частиц

rad<-rep(0,times=N) – задаём начальное значение для rad в виде вектора из N нулей

for(k in 1:N){

rad[k]<-(r[1,k]^2+r[2,k]^2+r[3,k]^2)^(0.5)

}

rad – поэлементно присваиваем каждому элементу вектора rad значение расстояния.

1. **Функция rki()**

rki<-function(r,k,i){##calculates the distance between particle i and particle k;

rki<-NULL

if(k==i){print("rki error! k=i!")}

else{

rki<-((r[1,k]-r[1,i])^2+(r[2,k]-r[2,i])^2+(r[3,k]-r[3,i])^2)^0.5

}

names(rki)<-"Distance"

rki

}

Функция rki() возвращает расстояние между i-й и k-й частицей. Возвращает вектор длины 1.

rki<-NULL – эта строка не нужна?

if(k==i){print("rki error! k=i!")} –Проверяет, чтобы i и k были различны. В противном случае выводит ошибку.

**else{**

**rki<-((r[1,k]-r[1,i])^2+(r[2,k]-r[2,i])^2+(r[3,k]-r[3,i])^2)^0.5**

**}**

В случае же, если k и i различны, присваевает в rki расстояние между частицами:

**names(rki)<-"Distance"**

**rki**

Присваивает имя в rki (исключительно для удобства пользования) имя distance, и возвращает сам rki.

**5. Функция U()**

Функция считает потенциальную энергию в соответствии с выражением (1\*).

(1\*)

U<-function(r){##returns a value of the potential energy of the system

N<-ncol(r)

U<-0

for(i in 1:N){

sum<-0

if(i<N){

for(k in (i+1):N){

sum<-sum+rki(r,k,i)^(-6)

}

}

U<-U+sum+r[1,i]^2+r[2,i]^2+r[3,i]^2

}

names(U)<-"Potential energy"

U

}

Разбираем пошагово.

Функция принимает на вход матрицу r размерами 3\*N координат частиц.

N<-ncol(r)

Основываясь на размеры матрицы, узнаём, сколько частиц было в системе

U<-0

Объявляем переменную U (в будущем – потенциальная энергия) и устанавливаем значение по умолчанию – 0.

for(i in 1:N){

sum<-0

if(i<N){

for(k in (i+1):N){

sum<-sum+rki(r,k,i)^(-6)

}

}

U<-U+sum+r[1,i]^2+r[2,i]^2+r[3,i]^2

}

Здесь написаны 2 вложенных друг в друга цикла for. Их предназначение – посчитать суммы в слагаемом (1\*). Для упрощения понимания перепишем слагаемое (1\*) в аналогичном виде:

(1\*)

Начнём разбор изнутри суммы.

sum<-0

промежуточная переменная, обозначающая значение второй вложенной суммы

if(i<N){

Это условие нужно, чтобы не посчитать энергию для «лишних» пар. При i=N-1, k=N уже посчитана энергия для пары (N-1,N), следовательно, считать вторую сумму для i=N уже нет необходимости.

for(k in (i+1):N){

sum<-sum+rki(r,k,i)^(-6)

}

На каждой итерации прибавляем к переменной sum значение .

U<-U+sum+r[1,i]^2+r[2,i]^2+r[3,i]^2

Для каждого i складываем внутреннюю сумму с членом . Данные действия повторяем для каждого i.

**6. Функция grad.U()**

Эта функция считает градиент энергии, а точнее, одну из компонент вектора градиента по конкретной переменной для конкретной частицы. Например, градиент полной энергии по считается так:

.

*.*

В первой сумме выживает только один член, содержащий . Во второй сумме таких членов N-1: суммирование производиться по всем i неравным k. Соответственно, для расчёта суммы нам понадобится только один цикл for.

grad.U<-function(r,var,k){##takes numeric vectors x,y,z, and number of particles N

##var is the variable, for which the gradient is calculated.var=1 - x, var=2 - y, var=3 - z

##k is the index of the variable in vector. (i.e., number of particle)

##so for var="x" and k=3 the gradient U by x[3] will be calculated.

N<-ncol(r) ##number of particles

if(k>N){print("grad.U: invalid k. k is more then number of particles")}

sum<-0

for(i in 1:N){

if(i!=k){

sum<-sum+6\*(r[var,k]-r[var,i])\*(rki(r,k,i)^(-8))

}

}

sum<-2\*r[var,k]-sum

sum

}

**7. Функция delta()**

Эта функция возвращает значение матрицы r2 следующей итерации градиентного спуска. На вход принимает r как матрицу с координатами частиц и alfa как параметр градиентного спуска. Осуществление итерации может привести к увеличению энергии при больших alfa; при достаточно малых alfa итерация приведёт к уменьшению потенциальной энергии градиентный спуск будет сходиться к минимуму потенциальной энергии.

delta<-function(r,alfa=1){##calculate vector of difference (one gradient descent iteration)

r2<-r

N<-ncol(r)

for(k in 1:N){

for(var in 1:3){

r2[var,k]<-r[var,k]-alfa\*grad.U(r,var,k)

}

}

r2

}

Здесь k обозначает номер частицы, var координату (var=1 соответствует x, var=2 – y, var=3 – z)

**8. Функция gradient.descent()**

gradient.descent<-function(N,r,alfa=1,K=10,print=FALSE){

Эта функция принимает на вход либо количество частиц N, либо матрицу координат r, а также параметры: alfa – константа градиентного спуска, K – количество итераций, print – логический параметр, характеризующий необходимость вывода потенциальной энергии после каждой итерации.

## Расчёт минимумов потенциальной энергии и оболочечной структуры.

1. Пример алгоритма для N=27

1.0 Придумать, как охарактеризовать, насколько близко мы находимся к локальному минимуму.Например, разность потенциальных энергий за последние 5000 итераций

Potential energy

-4.462208e-12

1.1. Убедиться, что найденный локальный минимум обладает наименьшей энергией (посчитать град. спуск несколько раз для разных случайных распределений)

1.2. Пользуясь rad(), посмотреть распределение частиц по радиусам, разделить на оболочки.

2. Построить таблицу распределения частиц по оболочкам для N=1:40