САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОЙ ТЕОРИИ МОДЕЛИРОВАНИЯ СИСТЕМ УПРАВЛЕНИЯ

Литвинцев Игорь Сергеевич

Выпускная квалификационная работа бакалавра

СРАВНЕНИЕ ГЛАДКИХ И НЕГЛАДКИХ МЕТОДОВ РЕШЕНИЯ ЗАДАЧИ О НАХОЖДЕНИИ ПРОСТРАНСТВЕННОЙ СТРУКТУРЫ МОЛЕКУЛЫ

Направление 010400

Прикладная математика, фундаментальная информатика
и основы программирования

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Тамасян Г. Ш.

Санкт-Петербург
2016

Содержание

Введение

Многие исследования в биологии сосредоточены на активности клеток, которые в первую очередь состоят из белка. Особенности белковой структуры таких клеток могут дать точное представление о ее функциях в организме.

Структура белка может быть определена экспериментально посредством спектроскопии ядерного магнитного резонанса или рентгеновской кристаллографии. В результате таких экспериментов становятся частично или полностью известны расстояния между атомами. Следовательно, возникает задача определения структуры молекулы по полученным расстояниям.

В работах [1]–[3] эта задача решается с помощью разнообразных подходов. Среди них имеются подходы, основанные на оптимизации некоторых гладких функционалов, минимумы которых совпадают с корректной конфигурацией молекулы.

В данной работе разработан подход основанный на оптимизации негладких функционалов, а также приведен сравнительный анализ результатов работы гладкого и негладких методов.

1 Постановка задачи

Задача о нахождении пространственной структуры молекулы может быть сформулирована, как задача поиска декартовых координат атомов $X_1, \ldots, X_m$ молекулы таких, что

$$\|X_i - X_j\| = d_{ij} \qquad ([i,j] \in D), \tag{1}$$

где D набор пар атомов $[i,j]$, между которыми известно евклидово расстояние d_{ij}. Множество D можно представить, как симметрическую матрицу, в которой элемент d_{ij} — расстояние между i-ым и j-ым атомами. Если расстояние между некоторой парой атомов неизвестно, на соответствующей позиции в матрице D поставим ноль.

В случае если точные расстояния между всеми парами атомов заданы, решение задачи, согласно статье [1], находится за линейное время. Однако на практике зачастую не только не все межатомные расстояния известны, но и не всегда существует возможность измерить расстояние между некоторыми парами атомов.

Есть множество различных подходов к решению задачи с разреженной матрицей расстояний. Один из них основан на том, что корректная конфигурация молекулы должна совпадать с минимумом некоторой функции невязок. Введем три различные функции невязок

$$F_1(X_1, \ldots, X_m) = \sum_{[i,j]\in D} (\|X_i - X_j\|^2 - d_{ij}^2)^2, \tag{2}$$

$$F_2(X_1, \ldots, X_m) = \sum_{[i,j]\in D} |\|X_i - X_j\|^2 - d_{ij}^2|, \tag{3}$$

$$F_3(X_1, \ldots, X_m) = \max_{[i,j]\in D} |\|X_i - X_j\|^2 - d_{ij}^2|, \tag{4}$$

где m – число атомов в молекуле.

Заметим, что первая функция является гладкой, а две другие нет. Реализация и сравнение алгоритмов минимизации приведенных функционалов составляют цель данной работы.

2 Вспомогательные сведения

Все нижеизложенные сведения данной главы взяты из [4]–[8].

2.1 Определения и обозначения

Определение 2.1. Выпуклой оболочкой точек $X_1, \ldots, X_m$ называется множество

$$\operatorname{co}\{X_1, \ldots, X_m\} = \left\{ X = \sum_{k=1}^{m} \alpha_k X_k \;\middle|\; \alpha_k \geqslant 0,\ \sum_{k=1}^{m} \alpha_k = 1 \right\}.$$

Определение 2.2. Функция f, заданная и конечная на открытом множестве $X \subset \mathbb{R}^n$, гиподифференцируема в точке $x \in X$, если существует такой выпуклый компакт $\underline{d}f(x) \subset \mathbb{R}^{n+1}$, что

$$f(x+\Delta) = f(x) + \max_{[a,v]\in \underline{d}f(x)} [a + \langle v, \Delta\rangle] + o_x(\Delta),$$

где

$$\lim_{\alpha \downarrow 0} \frac{o_x(\alpha, \Delta)}{\alpha} = 0 \qquad \forall \Delta \in \mathbb{R}^n.$$

Здесь $a \in \mathbb{R}^1;\ v \in \mathbb{R}^n$. Множество $\underline{d}f(x)$ называется гиподифференциалом функции f в точке x.

Определение 2.3. Пусть $A, B \subset \mathbb{R}^n$. Число

$$\rho(A,B) = \sup_{w\in A} \inf_{v\in B} \|v - w\| + \sup_{v\in B} \inf_{w\in A} \|v - w\|$$

называется расстоянием между множествами в метрике Хаусдорфа.

Определение 2.4. Отображение $A(x)$ называется непрерывным по Хаусдорфу в точке x_0, если

$$\lim_{x\to x_0} \rho(A(x), A(x_0)) = 0$$

Определение 2.5. Функция f называется непрерывно гиподифференцируемой в точке x, если она гиподифференцируема в некоторой окрестности точки x и если существует непрерывное (по Хаусдорфу) в этой точке гиподифференциальное отображение $\underline{d}f$.

2.2 Элементы гиподифференциального исчисления

Везде ниже все рассматриваемые функции полагаются заданными и конечными на открытом множестве $X \subset \mathbb{R}^n$. А суммирование выпуклых компактов производится по Минковскому.

Лемма 2.1. Гиподифференциал гладкой функции $f(x)$ в точке $x \in X$ равен

$$\underline{d}f(x) = \left\{ \begin{pmatrix} 0 \\ \nabla f(x) \end{pmatrix} \right\}. \tag{5}$$

Лемма 2.2. Если $f_1, \ldots, f_m$ — гиподифференцируемые в точке $x \in X$ функции, то и функция $f = \sum\limits_{i=1}^{m} c_i f_i$, где $c_i \in \mathbb{R}^1_+$, тоже гиподифференцируема в этой точке, при этом

$$\underline{d}f(x) = \sum_{i=1}^{m} c_i \underline{d}f_i(x). \tag{6}$$

Лемма 2.3. Если f_i, при $i \in 1:m$ — гиподифференцируемые в точке x функции, то и функция $f(x) = \max\limits_{i \in 1:m} f_i(x)$ тоже гиподифференцируема в этой точке, при этом

$$\underline{d}f(x) = \operatorname{co} \left\{ \underline{d}f_i(x) + \begin{pmatrix} f_i(x) - f(x) \\ 0_n \end{pmatrix} \;\middle|\; i \in 1:m \right\}. \tag{7}$$

2.3 МДМ-метод

Пусть в пространстве $\mathbb{R}^n$ заданы m точек, $H = \{a_i\}_{i=1}^m$. Обозначим через G выпуклую оболочку множества H. Задача: найти точку из G, ближайшую к началу координат или, что эквивалентно, решить следующую задачу

$$\|v\|^2 \to \min_{v \in G}. \tag{8}$$

В силу единственности точки минимума выпуклой функции на выпуклом множестве, задача (8) имеет единственное решение. Обозначим его v_*.

Обозначим через A матрицу со столбцами $a_1, \ldots, a_m$. Тогда любой вектор $v \in G$ допускает представление

$$v = Ap, \quad p \geq 0_m, \quad \sum_{i=1}^{m} p^{(i)} = 1. \tag{9}$$

Обозначим $M_+(p)$ носитель вектора p

$$M_+(p) = \{i \in 1\!:\!m \mid p^{(i)} > 0\}.$$

Введем величину

$$\Delta(p) = \max_{i \in M_+(p)} \langle a_i, v \rangle - \min_{i \in 1:m} \langle a_i, v \rangle ,$$

где $v = Ap$. Вектор p удовлетворяет условиям в (9). Множество таких векторов обозначим P.

Алгоритм

1) Выберем любое $v_0 \in \mathbb{R}^n$.

2) Пусть уже построено $v_k = Ap_k,\ p_k \in P$. Найдем индексы $i'_k \in M_+(p_k)$ и $i''_k \in 1\!:\!m$, такие, что

$$\max_{i \in M_+(p_k)} \langle a_i, v_k \rangle = \left\langle a_{i'_k}, v_k \right\rangle ,$$

$$\min_{i \in 1:m} \langle a_i, v_k \rangle = \left\langle a_{i''_k}, v_k \right\rangle .$$

Обозначим

$$a_{i'_k} = a'_k, \quad a_{i''_k} = a''_k.$$

Найдем

$$\Delta_k := \Delta(p_k) = \langle a'_k - a''_k, v_k \rangle .$$

Если $\Delta_k = 0$, то v_k — решение задачи (8). Процесс закончен.

3) Пусть $\Delta_k > 0$. Следующее приближение ищем, минимизируя выражение

$$v_k(t) = v_k - t p'_k (a'_k - a''_k), \quad t \in [0, 1],$$

где $p'_k = p_k^{(i'_k)}$.

Выберем t_k из условия

$$\|v_k(t_k)\|^2 = \min_{t \in [0,1]} \|v_k(t)\|^2.$$

Укажем явную формулу для t_k

$$t_k = \begin{cases} \widehat{t}_k, & \text{при } \widehat{t}_k < 1; \\ 1, & \text{при } \widehat{t}_k \geq 1; \end{cases}$$

где

$$\widehat{t}_k = \frac{p'_k \Delta_k}{\|\widehat{v}_k - v_k\|^2} = \frac{\Delta_k}{p'_k \|a'_k - a''_k\|^2}.$$

4) Вектор p_{k+1} находим по формуле

$$p_{k+1}^{(i)} = \begin{cases} p_k^{(i)}, & \text{при } i \neq i'_k,\ i \neq i''_k, \\ (1 - t_k) p_k^{(i'_k)}, & \text{при } i = i'_k, \\ p_k^{(i''_k)} + t_k p_k^{(i'_k)}, & \text{при } i = i''_k. \end{cases}$$

5) Положим $v_{k+1} = v_k(t_k)$.

Описание МДМ-метода завершено. Последовательность $\{v_k\}$ сходится к v_* — решению задачи (8).

2.4 Алгоритмы минимизации

Метод градиентного спуска

Пусть функция f определена и дифференцируема в пространстве $\mathbb{R}^n$. Будем осуществлять минимизацию в направлении наискорейшего спуска, т.е. в направлении антиградиента $-\nabla f$.

Алгоритм

1) Выберем любое $x_0 \in \mathbb{R}^n$.

2) Пусть уже построено $x_k \in \mathbb{R}^n$. Тогда

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \tag{10}$$

где α_k — находим, решая задачу одномерной минимизации

$$\alpha_k = \underset{\alpha \in [0,\infty]}{\operatorname{argmin}} f(x_k - \alpha \nabla f(x_k)). \tag{11}$$

3) Если для заданного ε выполнено одно из условий

$$\|x_{k+1} - x_k\| < \varepsilon; \tag{12}$$

$$\|f(x_{k+1}) - f(x_k)\| < \varepsilon; \tag{13}$$

$$\|\nabla f(x_{k+1})\| < \varepsilon. \tag{14}$$

то $x^* = x_{k+1}$ — точка минимума и алгоритм завершен. Иначе переходим к следующей итерации.

Метод гиподифференциального спуска

Пусть функция f задана, липшицева и непрерывно гиподифференцируема на $\mathbb{R}^n$.

Лемма 2.4. Необходимое условие минимума гиподифференцируемой функции f в точке x имеет вид

$$0_{n+1} \in \underline{d}f(x). \tag{15}$$

Алгоритм

1) Выберем любое $x_0 \in \mathbb{R}^n$.

2) Пусть уже построено $x_k \in \mathbb{R}^n$. Если в точке x_k выполнено (15), то x_k — точка минимума, и процесс останавливается.

3) Иначе найдем

$$\min_{z \in \underline{d}f(x_k)} \|z\| = \bar{z}_k. \tag{16}$$

Из того, что x_k не является точкой минимума следует, что

$$\bar{z}_k = [\eta_k, z_k] \neq 0_{n+1},$$

где $\eta_k \in \mathbb{R}^1,\ z_k \in \mathbb{R}^n$.

Направление наискорейшего спуска находится по формуле

$$g_k = -\frac{z_k}{\|z_k\|}.$$

4) Решая задачу одномерной минимизации, найдем шаг α_k

$$\alpha_k = \operatorname*{argmin}_{\alpha > 0} f(x_k + \alpha g_k). \tag{17}$$

5) Следующую точку положим

$$x_{k+1} = x_k + \alpha_k g_k.$$

Далее продолжаем аналогично. В результате получаем последовательность $\{x_k\}$ такую, что

$$f(x_{k+1}) < f(x_k).$$

На практике условие (15) будем проверять следующим образом. Если для заданного ε выполнено одно из условий

$$\|x_{k+1} - x_k\| < \varepsilon; \tag{18}$$

$$\|f(x_{k+1}) - f(x_k)\| < \varepsilon; \tag{19}$$

$$\|\bar{z}_k\| < \varepsilon; \tag{20}$$

то $x^* = x_{k+1}$ — точка минимума и алгоритм завершен. Иначе переходим к следующей итерации.

3 Решение задачи

Обозначим координаты точки $X_i \in \mathbb{R}^3$, как $(x_i,\ y_i,\ z_i)$, где $i \in 1:m$ (случай $X_i \in \mathbb{R}^2$ аналогичен). Введем вектор переменных

$$x = (X_1, \ldots, X_m) = (x_1, y_1, z_1, \ldots, x_m, y_m, z_m).$$

Без ограничения общности первый атом можно связать с началом координат. Таким образом, в дальнейшем при реализации алгоритмов оптимизации будем исключать координаты первого атома $X_m = (0, 0, 0)$ из искомых переменных. Следовательно, количество переменных равняется $N = n(m-1)$, где n – размерность пространства (везде далее будем полагать $n = 3$).

3.1 Минимизация гладкого функционала

Для минимизации функционала (2) применим метод наискорейшего спуска.

Выпишем формулу, по которой будем вычислять градиент в заданной точке. Для начала перепишем функционал в векторном виде

$$F_1(X_1, \ldots, X_m) = \sum_{\substack{1 \le i < j \le m \\ d_{ij} \neq 0}} \left((X_i - X_j)^T (X_i - X_j) - d_{ij}^2\right)^2,$$

Перегруппируем слагаемые следующим образом

$$\begin{aligned} F_1(X_1, \ldots, X_m) = &\sum_{\substack{j=2 \\ d_{1j} \neq 0}}^{m} \left((X_1 - X_j)^T (X_1 - X_j) - d_{1j}^2\right)^2 + \\ &+ \sum_{\substack{j=3 \\ d_{2j} \neq 0}}^{m} \left((X_2 - X_j)^T (X_2 - X_j) - d_{2j}^2\right)^2 + \cdots + \\ &+ \sum_{\substack{j=m-1 \\ d_{m-2,j} \neq 0}}^{m} \left((X_{m-2} - X_j)^T (X_{m-2} - X_j) - d_{m-2,j}^2\right)^2 + \\ &+ \sum_{\substack{j=m \\ d_{m-1,j} \neq 0}}^{m} \left((X_{m-1} - X_j)^T (X_{m-1} - X_j) - d_{m-1,j}^2\right)^2. \end{aligned}$$

Обозначим

$$\phi_i(X_i,\dots,X_m)=\sum_{\substack{j=i+1\\ d_{ij}\neq 0}}^{m}\big((X_i-X_j)^T(X_i-X_j)-d_{ij}^2\big)^2,\qquad \forall i\in 1{:}m{-}1.$$

Отметим, что каждое ϕ_i при $i\in 1:m-1$ зависит только от точек $X_i,\dots,X_m$.

Обозначим за $\frac{\partial}{\partial X_k}$ вектор частных производных по переменным x_k, y_k, z_k

$$\frac{\partial}{\partial X_k}=\left(\frac{\partial}{\partial x_k},\ \frac{\partial}{\partial y_k},\ \frac{\partial}{\partial z_k}\right).$$

Тогда для $\frac{\partial\phi_i}{\partial X_k}$ справедливы формулы

$$\frac{\partial\phi_i}{\partial X_k}=0_n,\qquad \forall i\in 2:m-1,\ k\in 1:i-1,$$

$$\frac{\partial\phi_i}{\partial X_i}=4\sum_{\substack{j=i+1\\ d_{ij}\neq 0}}^{m}\big((X_i-X_j)^T(X_i-X_j)-d_{ij}^2\big)(X_i-X_j),\quad \forall i\in 1{:}m{-}1,$$

$$\frac{\partial\phi_i}{\partial X_k}=-4\big((X_i-X_k)^T(X_i-X_k)-d_{ik}^2\big)(X_i-X_k),$$

$$\forall i\in 1{:}m{-}1,\ k\in i{+}1{:}m,\ d_{ik}\neq 0.$$

Наконец, компоненты градиента функции $F_1(X_1,\dots,X_m)$ находятся по следующей формуле

$$\frac{\partial F_1}{\partial X_k}=\sum_{i=1}^{m-1}\frac{\partial\phi_i}{\partial X_k}\qquad \forall k\in 1{:}m.$$

Учитывая, что точку X_1 мы зафиксировали в начале координат, в получившемся градиенте отбросим первые n компонент, соответствующие частным производным по компонентам точки X_1. Таким образом, ∇F_1 будет иметь размерность $N=n(m-1)$.

Задачу одномерной минимизации (11) будем решать, используя встроенную в MATLAB функцию *fmincon*, реализующую алгоритм внутренней точки. Отрезок минимизации формируем следующим образом: на первой итерации он равен фиксированному отрезку $[0,\ 10d_{max}]$, где d_{max} — максимальный элемент матрицы расстояний D. На k-й итерации $(k>1)$ отрезок

становится равен $[0,\ 5\alpha^*_{k-1}]$, где α^*_{k-1} — минимум задачи (11) на $(k-1)$-й итерации.

В качестве правила останова используем каждый из критериев (12)–(14) по очереди и сведем в единую таблицу количество итераций прошедших до выхода.

3.2 Минимизация негладких функционалов

Для минимизации негладких функционалов (3) и (4) будем использовать метод гиподифференциального спуска.

Минимизация функционала $F_2(x)$

Для начала опишем процесс вычисления гиподифференциала функции F_2 в произвольно заданной точке x.

Перепишем модули под знаком суммы в (3), как максимум двух функций

$$F_2(x) = \sum_{k_{ij}=1}^{M} f_{k_{ij}}(x) = \sum_{k_{ij}=1}^{M} \max\{h_{k_{ij}}(x), -h_{k_{ij}}(x)\},$$

где

$$f_{k_{ij}}(x) = |(X_i - X_j)^T(X_i - X_j) - d_{ij}^2|,$$

$$h_{k_{ij}}(x) = (X_i - X_j)^T(X_i - X_j) - d_{ij}^2.$$

Здесь k_{ij} — порядковый номер пары (i, j) такой, что $1 \le i < j \le m$ и $d_{ij} \neq 0$, а M — количество таких пар.

Выпишем в явном виде формулы для вычисления компонент градиентов функций $h_{k_{ij}}$ при $k_{ij} \in 1\!:\!M$

$$\frac{\partial h_{k_{ij}}}{\partial X_p} = 0_n, \qquad \forall p \in 1\!:\!m,\ p \neq i,\ p \neq j,$$

$$\frac{\partial h_{k_{ij}}}{\partial X_i} = 2(X_i - X_j), \qquad \frac{\partial h_{k_{ij}}}{\partial X_j} = -2(X_i - X_j).$$

Аналогично предыдущему случаю, во всех градиентах $\nabla h_{k_{ij}}$ отбросим первые n компонент, соответствующие частным производным по компонентам точки X_1. Таким образом, $\nabla h_{k_{ij}}$ будет иметь размерность $N = n(m-1)$.

Теперь, зная градиенты гладких функций, не составит труда вычислить гиподифференциал функции $F_2(x)$ пользуясь формулами гиподифференциального исчисления (5)–(7).

Важно отметить, что при увеличении числа атомов m количество элементов в $\underline{d}F_2(x)$ будет расти экспоненциально. Это будет происходить в силу того, что максимальное число слагаемых в функционале F_2 равняется числу ненулевых элементов d_{ij} при $1 \leq i < j \leq m$, и оценивается сверху числом $(m-1)m$. Тогда согласно (6), чтобы сформировать гиподифференциал, потребуется порядка $O(2^{(m-1)m})$ операций. Очевидно, что по возможности нужно избежать этой ситуации.

Согласно алгоритму гиподифференциального спуска, гиподифференциал $\underline{d}F_2(x)$ понадобится лишь на этапе проверки необходимого условия минимума и поиска направления спуска, что сводится к нахождению минимального по норме гипоградиента. Однако, в силу устройства функции F_2, задачу минимизации (16) можно свести к задаче квадратичного программирования. Покажем это.

Гиподифференциалы гладких функций $h_{k_{ij}}$ и $-h_{k_{ij}}$ (для краткости вместо k_{ij} далее будем писать просто k) при $k \in 1\!:\!M$ равны

$$\underline{d}h_k(x) = \left\{ \begin{pmatrix} 0 \\ \nabla h_k(x) \end{pmatrix} \right\}, \qquad \underline{d}(-h_k(x)) = \left\{ \begin{pmatrix} 0 \\ -\nabla h_k(x) \end{pmatrix} \right\}.$$

Отсюда, согласно гиподифференциальному исчислению, получаем гиподифференциал функции $f_k(x)$, $k \in 1\!:\!M$

$$\underline{d}f_k(x) = \operatorname{co}\{A_k, B_k\}, \tag{21}$$

где

$$A_k = \left\{ \underline{d}h_k(x) + \begin{pmatrix} h_k(x) - f_k(x) \\ 0_N \end{pmatrix} \right\} = \left\{ \begin{pmatrix} h_k(x) - f_k(x) \\ \nabla h_k(x) \end{pmatrix} \right\},$$

$$B_k = \left\{ -\underline{d}h_k(x) + \begin{pmatrix} -h_k(x) - f_k(x) \\ 0_N \end{pmatrix} \right\} = \left\{ \begin{pmatrix} -h_k(x) - f_k(x) \\ -\nabla h_k(x) \end{pmatrix} \right\}.$$

Так как гиподифференциал $\underline{d}f_k(x)$ это выпуклое множество, любую его точку $v_k \in \underline{d}f_k(x)$ можно представить в виде выпуклой комбинации

$$v_k = \alpha_k A_k + (1 - \alpha_k) B_k,$$

где $\alpha_k \in [0, 1]$.

С другой стороны гиподифференциал $\underline{d}F_2(x) = \sum\limits_{k=1}^{M} \underline{d}f_k(x)$ тоже выпуклое множество и любую его точку $v \in \underline{d}F_2(x)$ можно представить как

$$v = \sum_{k=1}^{M} v_k = \sum_{k=1}^{M} \big(\alpha_k A_k + (1 - \alpha_k) B_k\big),$$

где $\alpha_k \in [0, 1]$.

Таким образом, задачу (16) свели к задаче квадратичного программирования

$$\frac{1}{2}\|v\|^2 \to \min_{v \in \underline{d}F_2(x)}.$$

Везде в программе, где необходимо решить задачу квадратичного программирования, использовали встроенную в MATLAB функцию *quadprog*.

Задачу одномерной минимизации будем решать аналогично предыдущему случаю (см. пункт 3.1), используя функцию *fmincon*.

В качестве правила останова используем каждый из критериев (18) - (20) по очереди и сведем в единую таблицу количество итераций прошедших до выхода.

Минимизация функционала $F_3(x)$

Гиподифференциал $\underline{d}F_3(x)$ легко вычисляется по формулам (5) и (7) и равен

$$\underline{d}F_3(x) = \operatorname{co}\left\{ \underline{d}f_k(x) + \begin{pmatrix} f_k(x) - F_3(x) \\ 0_N \end{pmatrix} \;\middle|\; k \in 1\!:\!M \right\}, \qquad (22)$$

где $\underline{d}f_k(x)$ находятся из (21).

Заметим, что на этот раз число элементов в $\underline{d}F_3(x)$ не превышает $(m-1)m$. Перепишем формулу (22) в виде

$$\underline{d}F_3(x) = \operatorname{co}\{Z_1, \ldots, Z_s\}, \quad s = 2M,$$

где $Z_i \in \mathbb{R}^{N+1}$, при $N = n(m-1)$, и находятся по формулам

$$Z_{2k-1} = \begin{pmatrix} h_k(x) - F_3(x) \\ \nabla h_k(x) \end{pmatrix}, \quad Z_{2k} = \begin{pmatrix} -h_k(x) - F_3(x) \\ -\nabla h_k(x) \end{pmatrix}, \; k \in 1\!:\!M.$$

Согласно алгоритму гиподифференциального спуска, после того, как найден гиподифференциал $\underline{d}F_3(x)$ необходимо решить задачу минимизации (16). На этот раз для ее решения эффективно воспользоваться МДМ-методом, алгоритм которого описан в главе 2.

Заметим, что МДМ-метод плохо сходится, когда начало координат принадлежит $\underline{d}F_3(x)$. В связи с этим в [5] рекомендуется делать следующее.

Вначале решим задачу линейного программирования в пространстве векторов $W = (\alpha_1, \ldots, \alpha_s, u)$: найдем

$$\min u$$

при условиях

$$\sum_{i=1}^{s} \alpha_i z_k^{(i)} - u \le 0, \quad k \in 1\!:\!N+1,$$

$$-\sum_{i=1}^{s} \alpha_i z_k^{(i)} - u \le 0, \quad k \in 1\!:\!N+1,$$

$$\sum_{i=1}^{s} \alpha_i = 1; \; \alpha_i \ge 0, \; i \in 1\!:\!s; \; u \ge 0,$$

где $Z_i = (z_1^{(i)}, \ldots, z_{N+1}^{(i)})$ — точки из $\underline{d}F_3(x)$.

Обозначим через $W_0 = (\alpha_1^{(0)}, \ldots, \alpha_s^{(0)}, u^{(0)})$ решение этой задачи. Если $u^{(0)} = 0$, то начало координат содержится в гиподифференциале, и алгоритм гиподифференциального спуска завершен. Если же $u^{(0)} > 0$, то для нахождения ближайшей к началу координат точки гиподифференциала, можно воспользоваться МДМ-методом, взяв в качестве начального приближения точку

$$Z_0 = \sum_{i=1}^{s} \alpha_i^{(0)} Z_i,$$

где вектор $(\alpha_1^{(0)}, \ldots, \alpha_s^{(0)})$ составлен из первых s компонент вектора W_0.

На практике условие останова (20) заменим эквивалентным условием

$$|u^{(0)}| < \varepsilon, \tag{23}$$

а задачу линейного программирования решим, воспользовавшись функцией *linprog*.

Задачу одномерной минимизации решаем аналогично предыдущим случаям (см. пункт 3.1).

В качестве правила останова, помимо критерия (23), будем использовать критерии (18) и (19), и для каждого критерия сведем в единую таблицу количество итераций прошедших до выхода.

4 Численные эксперименты

Все рассмотренные выше алгоритмы реализованы на языке MATLAB.

Для каждого из функционалов F_1, F_2, F_3 были написаны сценарии, реализующие соответствующие алгоритмы минимизации. На вход каждой функции передаются четыре параметра: матрица расстояний D, начальная точка x_0, точность вычислений ε и максимальное количество итераций $NumIter$. На выходе получаем точку минимума соответствующего функционала.

Рассмотрим несколько примеров с разными входными параметрами.

Во всех примерах положим $NumIter = 200$ и $\varepsilon = 10^{-4}$. В критериях (12), (13), (18) и (19) будем использовать непосредственно $\varepsilon = 10^{-4}$, а в критериях (14), (20) и (23) используем $\varepsilon = 10^{-2}$ (в связи со спецификой критериев).

Заметим, что произвольно сформированная матрица D не подойдет для нашей задачи, т.к. в общем случае по ней нельзя восстановить корректную конфигурацию молекулы. Будем формировать матрицу D на основе координат атомов реально существующей молекулы.

Выберем произвольную молекулу из Protein Data Bank (банк данных 3-D структур белков и нуклеиновых кислот). Пусть это будет молекула с идентификатором 1PTQ, состоящая из 404 атомов. Во избежание больших размерностей в примерах будем использовать не все атомы молекулы, а лишь некоторые из них. По известным координатам этих атомов сформируем полную матрицу расстояний D. В качестве начальной точки x_0 выберем координаты $(x_i + \delta_x, y_i + \delta_y, z_i + \delta_z)$, где $i \in 1:m$, (x_i, y_i, z_i) — известные координаты атомов, а $\delta_x, \delta_y, \delta_z$ — случайные величины, подчиняющиеся стандартному нормальному распределению. Тем самым мы смоделировали ситуацию, когда известна полная матрица расстояний и некоторая начальная точка близкая к x_* — оптимальной точке, доставляющей минимум всем трем функционалам.

Пример 4.1. Случай полной матрицы, $m = 10$.

Результат работы алгоритмов минимизации функционалов F_1, F_2 и F_3 приведен в таблицах 1, 2 и 3 соответственно.

k	$f(x_{k+1})$	$\lvert f(x_k) - f(x_{k+1})\rvert$	$\lVert x_k - x_{k+1}\rVert$	$\lVert \nabla f(x_k)\rVert$
1	8260.2707511790	8260.2707511790	13.2544287096	6040.3248898236
5	453.9828111224	198.4693130093	0.6888006248	603.8107615152
20	47.5342117678	4.9514857255	0.0774235571	127.1388025029
50	2.2129246930	0.2605837340	0.0172937807	30.1074781694
100	0.0411799823	0.0028448104	0.0018859290	3.0164233547
150	0.0015528047	0.0000992959	0.0003559610	0.5590147346
200	0.0001296652	0.0000057771	0.0000953830	0.1213709544

Таблица 1: минимизация F_1 в случае полной матрицы, $m = 10$

k	$f(x_{k+1})$	$\lvert f(x_k) - f(x_{k+1})\rvert$	$\lVert x_k - x_{k+1}\rVert$	$\lVert \bar{z}_k\rVert$
1	144.2262555770	144.2262555770	2.5913798097	127.6774733645
5	3.8337048298	15.2395353420	0.8233148493	12.3558759441
10	0.0001334643	0.0007932857	0.0000277705	0.0009267500
17	0.0000100027	0.0000097699	0.0000003087	0.0000002328

Таблица 2: минимизация F_2 в случае полной матрицы, $m = 10$

k	$f(x_{k+1})$	$\lvert f(x_k) - f(x_{k+1})\rvert$	$\lVert x_k - x_{k+1}\rVert$	$\lVert u^{(k)}\rVert$
1	12.4168327175	12.4168327175	2.9441990362;	4.8797177542
5	2.1041535953	0.7673412804	0.5271780222	0.4223796526
10	1.0855159106	0.3827402898	0.9943154442	0.1381279400
13	0.0064073252	0.0000000105	0.0000002778	0.0063155860

Таблица 3: минимизация F_3 в случае полной матрицы, $m = 10$

Видим, что метод градиентного спуска довольно медленно сходился и превысил максимальное число итераций. Метод гиподифференцильного спуска, напротив, сошелся быстро: за 17 и 13 итераций для F_2 и F_3 соответственно. Максимальной точности по значению функции в точке минимума достиг функционал F_2. Время работы градиентного спуска составило 33.66с., гиподифференциального — 5.93с. и 8.60с. при минимизации F_1 и F_2 соответственно.

По всем параметрам наилучшим методом оказался метод гиподифференциального спуска минимизирующий функционал F_1.

Пример 4.2. Случай неполной матрицы, $m=6$.

Обнулив некоторые случайно выбранные компоненты корректной матрицы, получим разреженную матрицу

$$D = \begin{pmatrix} 0 & 1,464 & 0 & 0 & 2,458 & 0 \\ 1,464 & 0 & 1,525 & 2,411 & 0 & 2,591 \\ 0 & 1,525 & 0 & 1,232 & 2,459 & 0 \\ 0 & 2,411 & 1,232 & 0 & 0 & 4,620 \\ 2,458 & 0 & 2,459 & 0 & 0 & 1,519 \\ 0 & 2,591 & 0 & 4,620 & 1,519 & 0 \end{pmatrix}.$$

Результат работы алгоритмов приведен в таблицах 4, 5 и 6.

Аналогично предыдущему примеру медленней всех отработал метод градиентного спуска — за 13.8с. Время работы алгоритмов гиподифференциального спуска при минимизации F_1 и F_2 составило 1.57с. и 2.55с. соответственно.

В случае разреженной матрицы расстояний алгоритм оптимизирующий функционал F_2 показал лучшие результаты. Алгоритм градиентного спуска — худшие.

k	$f(x_{k+1})$	$\lvert f(x_k) - f(x_{k+1})\rvert$	$\lVert x_k - x_{k+1}\rVert$	$\lVert \nabla f(x_k)\rVert$
1	0.3295250778	0.3295250778	0.1699607237	72.6172899711
20	0.0168285291	0.0018310064	0.0036965003	0.9901334297
50	0.0007277416	0.0000776693	0.0007665760	0.2031338062
100	0.0000133201	0.0000010312	0.0000824726	0.0258778964
111	0.0000069185	0.0000003078	0.0000943951	0.0096069820

Таблица 4: минимизация F_1 в случае неполной матрицы, $m = 6$

k	$f(x_{k+1})$	$\lvert f(x_k) - f(x_{k+1})\rvert$	$\lVert x_k - x_{k+1}\rVert$	$\lVert \bar{z}_k\rVert$
1	0.2397530326	0.2397530326	0.2831620837	4.5378906335
4	0.0000007361	0.0000005537	0.0000003076	0.0000012898

Таблица 5: минимизация F_2 в случае неполной матрицы, $m = 6$

k	$f(x_{k+1})$	$\lvert f(x_k) - f(x_{k+1})\rvert$	$\lVert x_k - x_{k+1}\rVert$	$\lVert u^{(k)}\rVert$
1	0.0714320502	0.0714320502	0.2924567705	1.4933117648
4	0.0014927833	0.0000003999	0.0000023039	0.0014862652

Таблица 6: минимизация F_3 в случае неполной матрицы, $m = 6$

Ориентируясь на приведенные примеры, можно сделать вывод, что метод гиподифференциального спуска, в сравнении с методом градиентного спуска, сходится за гораздо меньшее количество итераций, работает быстрее, а также, в случае функционала F_2, показывает наилучшую точность.

Заключение

В работе решена задача о нахождении пространственной структуры молекулы по заданным расстояниям между атомами. В процессе решения задачи были разработаны оптимальные алгоритмы минимизации гладкого и негладких функционалов, а также приведен сравнительный анализ работы этих алгоритмов.

На языке Matlab написана программная реализация всех необходимых методов.

ЛИТЕРАТУРА

[1] Dong Q., Wu Z. A linear time algorithm for solving the molecular distance geometry problem with exact inter-atomic distances. // Journal of Global Optimization 22: pp. 365-375, 2002

[2] Grosso A., Locatelli M., Schoen F. Solving molecular distanse geometry problem by global optimization algorithms // Springer Science, 2007. pp. 23-37

[3] Thi L., An H. Solving large scale molecular distance geometry problems by a smoothing technique via the gaussian transform and D.C. programming // Journal of global optimization 27: 375-397, 2003.

[4] Глебов Н. И., Кочетов Ю. А., Плясунов А. В. Методы оптимизации. // Новосиб. ун-т. Новосибирск, 2000. 105с.

[5] Демьянов В. Ф., Малозёмов В. Н. Введение в минимакс. // Издательство «Наука», 1972. 368с.

[6] Демьянов В. Ф., Васильев Л. В. Недифференцируемая оптимизация. // М.: Наука, 1981. 384с.

[7] Демьянов В. Ф., Рубинов А. М. Основы негладкого анализа и квазидифференциальное исчисление. // М.: Наука, 1990. 432с.

[8] Малозёмов В. Н. МДМ-методу — 40 лет. // Семинар «DHA& CAGD». Избранные доклады. 10 декабря 2011.

Приложение

Основной файл сценарий.

```
close all; clc;
getpdb('1ptq', 'ToFile', '1ptq_file.pdb'); % скачиваем молекулу 1PTQ
pdbstruct = pdbread('1ptq_file.pdb'); % считываем файл в структуру
atom = pdbstruct.Model.Atom; % извлекаем поле атом
S = 10; % число считываемых атомов
XYZ = zeros(S,3); % массив координат
for i =1:S
    XYZ(i,1) = atom(i).X;
    XYZ(i,2) = atom(i).Y;
    XYZ(i,3) = atom(i).Z;
end
[m, n] = size(XYZ);
x0 = zeros(m,n);
shift = XYZ(1,:); % сдвиг координат в ноль
 for i=1:m
     x0(i,:) = XYZ(i,:) - shift;
 end
D = zeros(m, m);
for i=1:m-1
    for j=i+1:m
        D(i,j) = sqrt((x0(i,:)-x0(j,:))*(x0(i,:)-x0(j,:))');
    end
end
 y0 = x0;
% всем координатам добавляем случайную погрешность
for i=2:length(y0)
    y0(i,1) = y0(i,1)+ randn;
    y0(i,2) = y0(i,2)+ randn;
    y0(i,3) = y0(i,3)+ randn;
end
eps = 0.0001; iter = 200; % точность и  число итераций
x1 = GradientDescentF1( D, y0, eps, iter );
D1 = CheckAnswer(x1, D); % градиентный спуск для F1
title('Молекула найденная минимизацией F1');
figure
x2 = HypodifferentialDescentF2( D, y0, eps, iter );
D2 = CheckAnswer(x2, D); % гиподифференциальный спуск для F2
title('Молекула найденная минимизацией F2');
figure;
x3 = HypodifferentialDescentF3( D, y0, eps, iter );
D3 = CheckAnswer(x3, D); % гиподифференциальный спуск для F3
title('Молекула найденная минимизацией F3');
```

Минимизация $F_1(x)$ методом градиентного спуска.

```
function  y = GradientDescentF1( D, x0, eps1, IterNum )
% Минимизация функционала F1(x) методом градиентного спуска.
% D - матрица расстояний, x0 - начальная точка,
% eps1 - точность, IterNum - число итераций
[m, n] = size(x0);
dia = max(max(D))*10; % 10-ти кратный "диаметр" молекулы
xk = x0; iter = 0;
% флаги критериев останова 1, 2, 3
flag_g = 0; flag_y = 0; flag_x = 0;
f = zeros(2,1); % значение функции на смежных итерациях
while iter < IterNum
    iter = iter + 1;
    gk = GradF1( xk, D );
    fhnd_Func1 = @(t) Func1(D, xk, gk, t); % handle функция
    opts = optimset('Display','none');  % задача одномерной минимизации
    [tk, f(mod(iter,2)+1)] = fmincon(fhnd_Func1, dia/2, [],[],[],[],0, ...
        dia,[], opts);
    dia = tk*10;
    prev_xk = xk; % предыдущая точка
    xk(2:m, :) = xk(2:m, :) - gk*tk; % следующее приближение
    fk = f(mod(iter,2)+1);
    nfk = abs(f(1) - f(2));
    if nfk < eps1 && flag_y ~= 1
        disp(['1) Ответ по норме приращения функции. Итерация № ' ...
            num2str(iter) '. F1(x*) = ' num2str(fk)]);
        flag_y = 1;
    end
    ngk = Norma(gk); % норма градиента
    if ngk < eps1*100 && flag_g ~=1
        disp(['3) Ответ по норме градиента. Итерация № ' ...
            num2str(iter) '. F1(x*) = ' num2str(fk)]);
        flag_g = 1;
    end
    Δ = prev_xk-xk;
    nxk = Norma(Δ);
    if  nxk < eps1 && flag_x~=1
        disp(['2) Ответ по норме приращения аргумента. Итерация № ' ...
            num2str(iter) '. F1(x*) = ' num2str(fk)]);
        flag_x = 1;
    end
    if flag_x*flag_y*flag_g == 1
        break; % если все критерии сработали - выход
    end
end
```

```
41   if iter ≥ IterNum
42           disp(['Алгоритм превысил ', num2str(IterNum), ' итераций. ...
                F1(x*) = ' num2str(min(f))]);
43   end
44  y = xk;
45  end
```

Минимизация $F_2(x)$ методом гиподифференциального спуска.

```
1   function  y = HypodifferentialDescentF2( D, x0, eps1, IterNum )
2   % Минимизация функционала F2(x) методом гиподифференциального спуска.
3   % D - матрица расстояний, x0 - начальная точка,
4   % eps1 - точность, IterNum - число итераций
5   [m, n] = size(x0); % m число атомов, n размерность
6   N =  n*(m - 1);%число неизвестных
7   dia = max(max(D))*2;
8   M = nnz(D); %количество ненулевых элементов в D
9   xk = x0; iter = 0;
10  % флаги критериев останова 1, 2, 3
11  flag_g = 0; flag_y = 0; flag_x = 0;
12  f = zeros(2,1);
13  while iter < IterNum
14      iter = iter+1; k=1;
15  hi = zeros(M, 1);
16  hi_grad = zeros(M, N);
17  for i=1:m-1 % вычисляем градиенты
18      for j=i+1:m
19          if  D(i,j)≠0
20              hi(k) = hij(xk(i,:), xk(j,:), D(i,j));
21              hi_grad(k, :) = Gradhi(xk(i,:), xk(j,:), n*(i-1)-(n-1), ...
                    n*(j-1)-(n-1), N);
22              k=k+1;
23          end
24      end
25  end
26  h = max(hi, -hi); % формируем гиподифференциал
27  A = [hi-h, hi_grad]; B = [-hi-h, -hi_grad];
28  % находим ближайшее расстояние до начала координат
29  gipogradient = DistanceToZero(A, B)';
30  ngk = norm(gipogradient); % норма гипоградиента
31  z = gipogradient(2:N+1);
32  g = -z/norm(z); % направление спуска
33  gk = zeros(m-1, n);
34  k=1;
35  for i=1:m-1
36      for j=1:n
```

```
37          gk(i, j) = g(k);
38          k=k+1;
39      end
40  end
41  fhnd_Func2 = @(t) Func2(D, xk, gk, t, M); % handle функция
42  % одномерная минимизация
43  opts = optimset('Display','none');
44  [tk, f(mod(iter,2)+1)] = fmincon(fhnd_Func2, dia/2, [],[],[],[],0, ...
        dia,[], opts);
45  dia = tk*5;
46  fk =  f(mod(iter,2)+1);
47  prev_xk = xk; % предыдущее приближение
48  xk(2:m, :) = xk(2:m, :) + gk*tk; % следующее приближение
49      nyk = abs(f(1) - f(2));
50      if nyk < eps1 && flag_y ~= 1
51          disp(['1) Ответ по норме приращения функции. Итерация № ' ...
                num2str(iter) '. F2(x*) = ' num2str(fk)]);
52          flag_y = 1;
53      end
54      if ngk < eps1*100 && flag_g ~=1
55            disp(['3) Ответ по норме гипоградиента. Итерация № ' ...
                  num2str(iter) '. F2(x*) = ' num2str(fk)]);
56            flag_g = 1;
57      end
58      nxk = Norma(prev_xk-xk);
59      if  nxk < eps1 && flag_x~=1
60          disp(['2) Ответ по норме приращения аргумента. Итерация № ' ...
                num2str(iter) '. F2(x*) = ' num2str(fk)]);
61          flag_x = 1;
62      end
63      if flag_x*flag_y*flag_g == 1
64          break; % если все критерии сработали - выход
65      end
66  end
67  if iter >= IterNum
68       disp(['Алгоритм превысил ', num2str(iter), ' итераций. F2(x*) = ' ...
             num2str(fk)]);
69  end
70  fclose(fid);
71  y = xk;
72  end
```

Минимизация $F_3(x)$ методом гиподифференциального спуска.

```
function  y = HypodifferentialDescentF3( D, x0, eps1, IterNum )
% Минимизация функционала F3(x) методом гиподифференциального спуска.
% D - матрица расстояний, x0 - начальная точка,
% eps1 - точность, IterNum - число итераций
[m, n] = size(x0); % m число атомов, n размерность
M = nnz(D); % количество ненулевых элементов в D
N =  n*(m - 1); dia = max(max(D));
xk = x0; iter = 0;
% флаги критериев останова 1, 2, 3
flag_g = 0; flag_y = 0; flag_x = 0;
f = zeros(2,1);
while iter < IterNum
    iter = iter + 1; k=1;
hi = zeros(M, 1); hi_grad = zeros(M, N);
% вычисляем градиенты гладких функций
for i=1:m-1
    for j=i+1:m
        if  D(i,j)≠0
            hi(k) = hij(xk(i,:), xk(j,:), D(i,j));
            hi_grad(k, :) = Gradhi(xk(i,:), xk(j,:), n*(i-1)-(n-1), ...
                n*(j-1)-(n-1), N);
            k=k+1;
        end
    end
end
F3 = max(max(hi, -hi)); % формируем гиподифференциал
A = [hi-F3, hi_grad]; B = [-hi-F3, -hi_grad];
Gipo = [A; B]';
% проверяем лежит ли ноль в гиподифференциале
[FlagNullInside, z0, u] = ZeroInside(Gipo, 10*eps1);
if FlagNullInside == 1
    disp(['3) Ответ по норме z0 найден на  ' num2str(iter) ' итерации.']);
    if iter < 2
        f(1) = Func2(D, xk, xk(2:m,:), 0, M);
    end
    disp(['F3(x*) = ' num2str(min(f))]);
end
gipogradient = MDM_method(Gipo, z0); % МДМ-метод
ngk = norm(gipogradient); % норма гипоградиента
z = gipogradient(2:N+1);
g = -z/norm(z); % направление спуска
gk = zeros(m-1, n);
k=1;
for i=1:m-1
```

```
44      for j=1:n
45          gk(i, j) = g(k);
46          k=k+1;
47      end
48  end
49  % задача одномерной оптимизации
50  fhnd_Func3 = @(t) Func3(D, xk, gk, t, M); % handle функция
51  opts = optimset('Display','none');
52  [tk, f(mod(iter,2)+1)] = ...
        fmincon(fhnd_Func3,dia/2,[],[],[],[],0,dia,[],opts);
53  dia = tk*5;
54  fk =  f(mod(iter,2)+1);
55  prev_xk = xk; % точка на предыдущей итерации
56  xk(2:m, :) = xk(2:m, :) + gk*tk; % следующее приближение
57      nyk = abs(f(1) - f(2));
58      if nyk < eps1 && flag_y ~= 1
59          disp(['1) Ответ по норме приращения функции. Итерация № ' ...
                num2str(iter) '. F3(x*) = ' num2str(fk)]);
60          flag_y = 1;
61      end
62      if ngk < eps1*100 && flag_g ~=1
63            disp(['3) Ответ по норме гипоградиента. Итерация № ' ...
                  num2str(iter) '. F3(x*) = ' num2str(fk)]);
64            flag_g = 1;
65      end
66      Δ = prev_xk-xk;
67      nxk = Norma(Δ);
68      if  nxk < eps1 && flag_x~=1
69          disp(['2) Ответ по норме приращения аргумента. Итерация № ' ...
                num2str(iter) '. F3(x*) = ' num2str(fk)]);
70          flag_x = 1;
71      end
72      if flag_x*flag_y*flag_g == 1
73          break; % если все критерии сработали - выход
74      end
75
76  end
77  if iter ≥ IterNum
78       disp(['Алгоритм превысил ', num2str(iter), ' итераций. F3(x*) = ' ...
            num2str(fk)]);
79  end
80  y = xk;
81  end
```

Функция $F_1(x)$.

```
function y = Func1( D, x0, g, t)
% функционал F1(x). D - матрица расстояний,
% x0 - начальная точка, g - направление спуска,
% t - параметр одномерной минимизации
[m, n] = size(x0);
x = zeros(m, n);
x(2:m, :) = x0(2:m, :) - t*g;
fi = zeros(m-1, 1);
k=1;
for i=1:m-1
    for j=i+1:m
        if D(i,j)≠0
            fi(k) = ((x(i,:)-x(j,:))*(x(i,:)-x(j,:))' - D(i,j)^2)^2;
            k=k+1;
        end
    end
end
y = sum(fi);
end
```

Функция $F_2(x)$.

```
function y = Func2( D, x0, g, t, M)
% функционал F2(x). D - матрица расстояний,
% x0 - начальная точка, g - направление спуска,
% t - параметр одномерной минимизации,
% M - число ненулевых элементов в матрице D
[m, n] = size(x0);
x = zeros(m, n);
x(2:m, :) = x0(2:m, :) + t*g;
fi = zeros(M, 1);
k=1;
for i=1:m-1
    for j=i+1:m
        if D(i,j)≠0
            fi(k) = ((x(i,:)-x(j,:))*(x(i,:)-x(j,:))' - D(i,j)^2);
            k = k+1;
        end
    end
end
y = sum(max(fi, -fi));
end
```

Функция $F_3(x)$.

```
function y = Func3( D, x0, g, t, M)
% функционал F3(x). D - матрица расстояний,
% x0 - начальная точка, g - направление спуска,
% t - параметр одномерной минимизации,
% M - число ненулевых элементов в матрице D
[m, n] = size(x0);
x = zeros(m, n);
x(2:m, :) = x0(2:m, :) + t*g;
fi = zeros(M, 1);
k=1;
for i=1:m-1
    for j=i+1:m
        if D(i,j)≠0
            fi(k) = ((x(i,:)-x(j,:))*(x(i,:)-x(j,:))' - D(i,j)^2);
            k = k+1;
        end
    end
end
y = max(max(fi, -fi));
end
```

Градиент функции $F_1(x)$.

```
function y = GradF1( x0, D )
% функция вычисляющая градиент функции F1(x)
% x0 - точка, D - матрица расстояний
[m, n] = size(x0);
Phi = zeros(n*m, m-1); % матрица частных производных функций
for i=1:m-1
    for j=i+1:m
        if D(i,j)≠0
            buf = ((x0(i,:)-x0(j,:))*(x0(i,:)-x0(j,:))' - D(i,j)^2)*( ...
                x0(i,:)-x0(j,:) )';
            % формируем диагональные элементы матрицы
            Phi(n*i-(n-1):n*i, i) = Phi(n*i-(n-1):n*i, i) + 4*buf;
            % формируем элементы под диагональю
            Phi(n*j-(n-1):n*j, i) = -4*buf;
        end
    end
end
y = zeros(m-1, n);
% первые n строк нас не интересуют, т.к. они отвечают частным ...
    производным по
% компонентам 1-го атома, который мы зафиксировали в начале координат
```

```
k=n+1;
for i=2:m
    for j=1:n
        y(i-1,j) = sum(Phi(k,:));
        k=k+1;
    end
end
end
```

Функция $h_{k_{ij}}(x)$.

```
function y = hij( xi, xj, d )
% функция, вычисляющая значение функции hij в точках xi, xj.
% d - (i,j)-ый элемент матрицы D
y = (xi-xj)*(xi-xj)' - d^2;
end
```

Градиент функции $h_{k_{ij}}(x)$.

```
function g = Gradhi( xi, xj, i, j, N )
% функция, вычисляющая градиенты гладких функций hi(xi, xj)
% xi, xj - аргументы hi, i, j - индексы аргументов
% N - общее число неизвестных задачи

n = length(xi); % размерность
g = zeros(N,1);
if i<1 % В силу специфики функций hij-ых, на i-ую позицию ставим значения:
    g(j:j+n-1, 1) = 2*xj';
else
    g(i:i+n-1, 1) = 2*(xi - xj)';
    % на j-ую позицию ставим:
    g(j:j+n-1, 1) = 2*(xj - xi)';
end
% Остальные компоненты градиента g равны нулю
end
```

Поиск минимального расстояния от начала координат до выпуклого множества.

```
function vmin = DistanceToZero( A, B )
% функция поиска минимального расстояния от гиподифференциала
% до начала координат методом квадратичного программирования. [A, B] - ...
    гиподифференциал
 % задаем матрицу H для задачи квадратичного программирования (xT*H*x)
AB = A-B; H = AB*AB';
% формируем остальные параметры функции quadprog
Bs = sum(B); f = Bs*AB';
[M, N1] = size(A);
% верхние и нижние ограничения
lb = zeros(M, 1); ub = ones(M, 1);
opts = optimset('LargeScale','on', 'Display','none');
% рещаем задачу квадратичного программирования
[alpha]  = quadprog(H,f,[],[],[],[],lb,ub, [], opts);
vmin = Bs + alpha'*AB;
end
```

Проверка на принадлежность начала координат выпуклому множеству.

```
function [ NullInside, z0, u] = ZeroInside( A1, eps2 )
% функция, проверяющая принадлежность нуля выпуклому множеству A1
% если ноль принадлежит, то возвращает NullInside=1, z0=0.
% иначе NullInside=0, z0 - начальное приближение для МДМ метода.
% u - искомый параметр метода
    [size_m, size_n] = size(A1);
    % формируем последний столбец коэффициентов вектора u
    last_column = -ones(size_m*2, 1);
    b = zeros(size_m*2, 1); % столбец b
    % матрица A для ограничений неравенств
    A = [A1; -A1];
    A = [A, last_column];
    % вектор-столбец коэффициентов целевой функции (минимизируем только
    % последнюю компоненту u)
    f = zeros(size_n+1, 1);
    f(size_n+1, 1) = 1;
    % матрица Aeq для ограничений равенств
    Aeq = ones(1, size_n+1);
    Aeq(1, size_n+1) = 0;
    beq = 1;
    % ограничения снизу lb ≤ x ≤ ub
    lb = zeros(size_n+1, 1);
```

```
23      % линейное программирование. используем симплекс метод
24      options = optimset('Simplex','on', 'Display','none');
25      [alfa_u] = linprog(f, A, b, Aeq, beq, lb, [], [], options);
26      z0 = zeros(size_m,1);
27      u = abs(alfa_u(size_n+1));
28      if u < eps2
29          NullInside = 1;
30      else
31          NullInside = 0;
32          for i=1:size_n
33              z0 = z0 + alfa_u(i)*A1(:,i);
34          end
35      end
36  end
```

МДМ-метод.

```
1   function [ v_min ] = MDM( A, v0 )
2   % МДМ метод. Поиск ближайшей к нулю точки симплекса
3   % A - симплекс, v0 - начальное приближение
4     eps2 = 0.001; % точность МДМ-метода
5     vk = v0;
6     [N, M] = size(A);
7     % вектор из единиц для составления линейной системы с ограничениями
8     a = ones(1, M);
9     % решаем систему, учитывая, что p_1 + p_2 + ... + p_m = 1 & p_i ≥ 0
10    opts = optimset('TolX', 0.000001);
11     p = lsqnonneg([A; a], [vk; 1], [], opts);
12    N = 1000; % максимальное число итераций
13    for iter=1:N
14      M_plus= find(p); % формируем носитель вектора p
15      % ищем индекс i', на котором реалиуется max(a_i, v_k), по всем i из ...
            M_plus
16      max_ai_vk = vk'*A(:, M_plus);
17      min_ai_vk = vk'*A;
18      % находим максимум и запоминаем индекс i1
19      [max_i1, i1_with_hat] = max(max_ai_vk);
20      i1 = M_plus(i1_with_hat);
21      % находим минимум и запоминаем индекс i2
22      [min_i2, i2] = min(min_ai_vk);
23      % вычисляем дельта и проверяем условие останова
24      Δ = max_i1-min_i2;
25      if Δ < eps2
26          v_min = vk;
27          break;
28      end
```

```
29      Δ_a1a2 = A(:, i1)-A(:, i2);
30      v_hat = vk - p(i1)*Δ_a1a2;
31      norma_a1a2 = Δ_a1a2'*Δ_a1a2;
32      tk_hat = Δ/(p(i1)*norma_a1a2);
33      if tk_hat≥1
34          tk = 1;
35      else
36          tk = tk_hat;
37      end
38      vk = vk + tk*(v_hat - vk);  % следующее приближение
39      % вычисляем p
40      pi1 = p(i1); pi2 = p(i2);
41      for i=1:length(p)
42          if i ≠ i1 && i≠i2
43              p(i)=p(i);
44          else
45              if i == i1
46                  p(i) = (1 - tk)*pi1;
47              end
48              if i == i2
49                  p(i) = pi2 + tk*pi1;
50              end
51          end
52      end
53    end
54    v_min = vk;
55  end
```

Вычисление нормы Евклида.

```
1  function a = Norma( x )
2   % разворачиваем матрицу x([m, n]) в вектор и вычисляем его норму
3  [m, n] = size(x);
4  a = 0;
5  for i=1:m
6      for j=1:n
7          a = a + x(i,j)^2;
8      end
9  end
10 a = sqrt(a);
11 end
```

Восстановление матрицы D по заданному вектору x.

```
function D_ans = CheckAnswer( x, D )
% функция, восстановливающая матрицу D по заданному вектору x
% и рисующая молекулу.
[m, n] = size(x);
D_ans = zeros(m, m);
k=1;
for i=1:m-1
    for j=i+1:m
        D_ans(i,j) = sqrt((x(i,:)-x(j,:))*(x(i,:)-x(j,:))');
        if D(i,j)≠0
            bonds(k,:) = [i, j];
            k=k+1;
        end
    end
end
% рисуем ответ
if n == 2 % двумерный случай
    plot(x(:,1), x(:,2), 'o');
     for i=1:m
         text(x(i,1), x(i,2), ['  ' num2str(i)], 'color', 'red');
     end
    for k=1:length(bonds)
        line([x(bonds(k,1),1) x(bonds(k,2),1)], [x(bonds(k,1),2), ...
            x(bonds(k,2),2)]);
    end
     grid on;
end
if n==3  % трехмерный случай
    plot3(x(:,1), x(:,2), x(:,3), 'o');
     for i=1:m
         text(x(i,1), x(i,2), x(i,3), ['  ' num2str(i)], 'color', 'red');
     end
    for k=1:length(bonds)
        line([x(bonds(k,1),1), x(bonds(k,2),1) ], [x(bonds(k,1),2), ...
            x(bonds(k,2),2)], [x(bonds(k,1),3), x(bonds(k,2),3)]);
    end
    grid on;
end
end
```