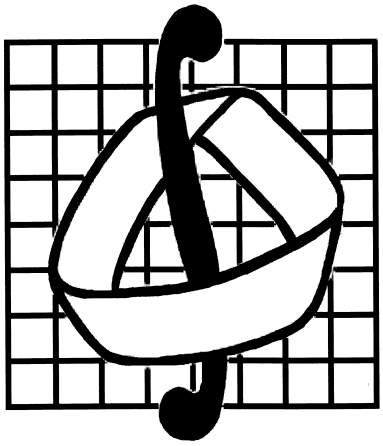
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М. В. ЛОМОНОСОВА
Механико-математический факультет



Курсовая работа.

Аппроксимация многогранника по набору пространственных отрезков. Построение многогранника по набору полупространств.

Научный руководитель: Валединский В.Д.

Студент: Ковальков М.Н.

Содержание

1. Введение

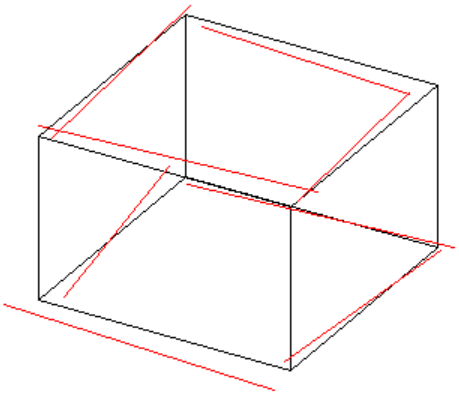
1.1. Происхождение задачи.

Данная работа возникла из задачи оценки качества огранки драгоценных камней.

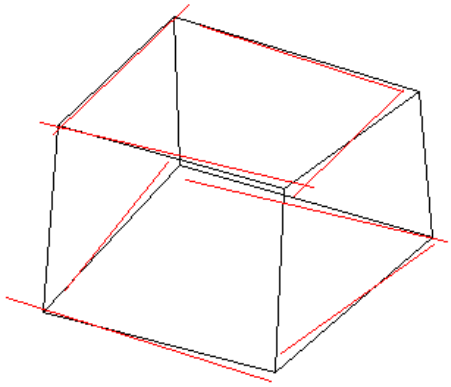
2. Задача аппроксимации многогранника по набору пространственных отрезков.

2.1. Постановка задачи.

Имеется многогранник и набор целевых ребер. Предполагается, что многогранник построен неточно, а целевые ребра-точны. Требуется изменить многогранник так, чтобы он наилчшим образом соответствовал целевым ребрам.



Исходный многогранник



Измененный многогранник

2.2. Математическая формализация задачи. Построение минимализируемого функционала

Имеется многогранник M, заданный наборами своих граней, ребер и вершин.

$$\begin{cases} p_i = (x_i, y_i, z_i) - & \text{Координаты вершин многогранника, } i = \overline{1, n_1} \\ e_j = (a_{j,1}, a_{j,2}) - & \text{Номера вершин, которые являются концами ребра } e_j,\ j = \overline{1, n_2} \\ f_k = [b_{k,1}, \ldots, b_{k,m_k}] - & \text{Номера ребер, входящих в грань } f_k,\ k = \overline{1, n_3} \end{cases}$$

Имеется список целевых ребер, которые считаются точными и которые поставлены в соответствие исходным ребрам многогранника.

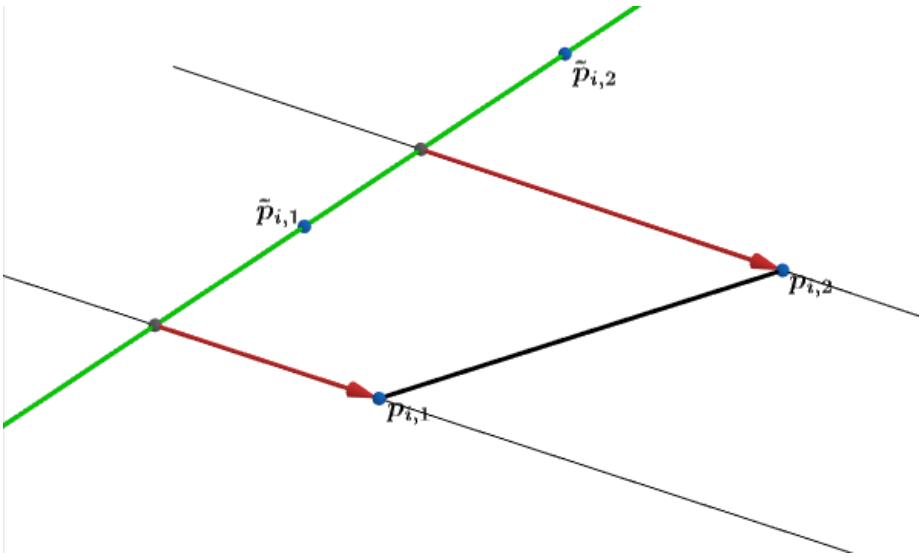
$$\widetilde{e_i} = \begin{cases} (a_{i,x}, a_{i,y}, a_{i,z}), (b_{i,x}, b_{i,y}, b_{i,z}) - \text{координаты двух концов данного целевого ребра.} \\ ind_i - \text{номер исходного ребра, которому соответствует i-е целевое ребро.} \end{cases}$$

Если в списке целевых ребер нет элемента для текущего ребра, то считаем, что оно является целевым само для себя. Однако вес, с которым оно входит в функционал будет равен $\omega_i = 0.1$, в то время, как для полноценного целевого ребра $\omega_i = 1$. После расширения списка целевых ребер указанным способом, будем считать, что $\widetilde{e_i}$-целевое ребро, соответствующее исходному ребру e_i многогранника M.

Сделаем ребра многогранника "подвижными для этого введём переменные $\tilde{p}_{i,1} = (\tilde{x}_{i,1}, \tilde{y}_{i,1}, \tilde{z}_{i,1}), \tilde{p}_{i,1} = (\tilde{x}_{i,2}, \tilde{y}_{i,2}, \tilde{z}_{i,2})$, соответствующие концам "подвижных"ребер. Мы хотим найти такую конфигурацию, чтобы эти ребра были наименее удалены от соответствующим им целевых. Для этого мы должны доставить минимум следующего функционала:

$$\sum_{i=1}^{n_2} \omega_i(\rho^2(\tilde{p}_{i,1}, \widetilde{e_i}) + \rho^2(\tilde{p}_{i,2}, \widetilde{e_i})) \to \min, \text{ где}$$

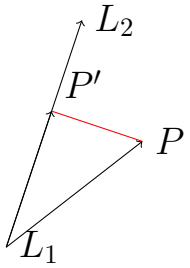
$\rho(\tilde{p}_{i,j}, \widetilde{e_i})$-расстояние между концами "движимого"ребра и прямой, заданной соответствующим целевым ребром.



2.3. Вычисление расстояния от точки до прямой.

Для вычисления расстояние воспользуемся формулой вычисления проекции вектора $\overrightarrow{a}$ на вектор $\overrightarrow{b}$:

$$pr_{\overrightarrow{L_1L_2}}\overrightarrow{L_1P} = \frac{(\overrightarrow{L_1P}, \overrightarrow{L_1L_2})}{(\overrightarrow{L_1L_2}, \overrightarrow{L_1L_2})}\overrightarrow{L_1L_2}, \text{ где}$$



(*,*) – стандартное скалярное произведение:

$$(\overrightarrow{a}, \overrightarrow{b}) = a_xb_x + a_yb_y + a_zb_z$$

Далее пользуемся теоремой Пифагора для вычисления расстояния:

$$\rho^2(P, L_1L_2) = |\overrightarrow{L_1P}|^2 - |pr_{\overrightarrow{L_1L_2}}\overrightarrow{L_1P}|^2 = \frac{(\overrightarrow{L_1P}, \overrightarrow{L_1P})(\overrightarrow{L_1L_2}, \overrightarrow{L_1L_2}) - (\overrightarrow{L_1P}, \overrightarrow{L_1L_2})(\overrightarrow{L_1P}, \overrightarrow{L_1L_2})}{(\overrightarrow{L_1L_2}, \overrightarrow{L_1L_2})}$$

2.4. Доказательство выпуклости функционала.

Для корректности применения метода внутренней точки требуется выпуклость минимализируемого функционала. Докажем, что рассматриваемый нами функционал-выпуклый. Рассмотрим одно слагаемое суммы, образующей наш функционал:

$$F = (x-a_x)^2+(y-a_y)^2+(z-a_z)^2-\frac{((x-a_x)(b_x-a_x)+(x-a_y)(b_y-a_y)+(x-a_z)(b_z-a_z))^2}{(b_x-a_x)^2+(b_y-a_y)^2+(b_z-a_z)^2}$$

Введем обозначения: $c_1 := (b_x-a_x), c_2 := (b_y-a_y), c_3 := (b_z-a_z), c := c_1^2+c_2^2+c_3^2$.
И запишем матрицу вторых частных производных F:

$$\begin{pmatrix} \frac{\partial^2 F}{\partial x^2} & \frac{\partial^2 F}{\partial x \partial y} & \frac{\partial^2 F}{\partial x \partial z} \\ \frac{\partial^2 F}{\partial y \partial x} & \frac{\partial^2 F}{\partial y^2} & \frac{\partial^2 F}{\partial y \partial z} \\ \frac{\partial^2 F}{\partial z \partial x} & \frac{\partial^2 F}{\partial z \partial y} & \frac{\partial^2 F}{\partial z^2} \end{pmatrix} = \begin{pmatrix} 2(1-\frac{c_1^2}{c}) & -\frac{c_1c_2}{c} & -\frac{c_1c_3}{c} \\ -\frac{c_1c_2}{c} & 2(1-\frac{c_2^2}{c}) & -\frac{c_2c_3}{c} \\ -\frac{c_1c_3}{c} & -\frac{c_2c_3}{c} & 2(1-\frac{c_3^2}{c}) \end{pmatrix}$$

Проверим матрицу на неотрицательную определенность с помощью Критерия Сильвестра. Для этого вычислим определители главных миноров.
1)$|\Delta_1| = 2\frac{c_2^2+c_3^2}{c} \geq 0$
2)$|\Delta_2| = \begin{vmatrix} 2(1-\frac{c_1^2}{c}) & -\frac{c_1c_2}{c} \\ -\frac{c_1c_2}{c} & 2(1-\frac{c_2^2}{c}) \end{vmatrix} = 4 - 4\frac{c_1^2+c_2^2}{c} + 4\frac{c_1^2c_2^2}{c^2} - \frac{c_1^2c_2^2}{c^2} = 4\frac{c_3^2}{c} + 3\frac{c_1^2c_2^2}{c^2} \geq 0$
3) $|\Delta_3| = \begin{vmatrix} 2(1-\frac{c_1^2}{c}) & -\frac{c_1c_2}{c} & -\frac{c_1c_3}{c} \\ -\frac{c_1c_2}{c} & 2(1-\frac{c_2^2}{c}) & -\frac{c_2c_3}{c} \\ -\frac{c_1c_3}{c} & -\frac{c_2c_3}{c} & 2(1-\frac{c_3^2}{c}) \end{vmatrix} = 8(1-\frac{c_1^2}{c})(1-\frac{c_2^2}{c})(1-\frac{c_3^2}{c}) - 2\frac{c_1^2c_2^2c_3^2}{c^3} -$
$- 2\frac{c_1^2c_3^2}{c^2}(1-\frac{c_2^2}{c}) - 2\frac{c_2^2c_3^2}{c^2}(1-\frac{c_1^2}{c}) - \frac{c_1^2c_2^2}{c^2}(1-\frac{c_3^2}{c}) = 8 - 4\frac{c_1^2c_2^2c_3^2}{c^3} + 6(\frac{c_1^2c_2^2}{c^2} + \frac{c_1^2c_3^2}{c^2} + \frac{c_2^2c_3^2}{c^2}) -$
$8\frac{c_1^2+c_2^2+c_3^2}{c} =$

$= 6(\frac{c_1^2c_2^2}{c^2} + \frac{c_1^2c_3^2}{c^2}) + 2(\frac{c_2^2c_3^2}{c^2}) + 4 + \frac{c_2^2c_3^2}{c^2}(1 - \frac{c_1^2}{c}) = 6(\frac{c_1^2c_2^2}{c^2} + \frac{c_1^2c_3^2}{c^2}) + 2(\frac{c_2^2c_3^2}{c^2}) + 4 + \frac{c_2^2c_3^2}{c^2}(\frac{c_2^2+c_3^2}{c}) \geq 0$
Тем самым мы показали, что матрица вторых частных производных функции неотрицательна, следовательно эта функция является выпуклой. Осталось вспомнить, что сумма выпуклых функций также является выпуклой (Прямое следствие определения через неравенство Йенсена).

Таким образом, показана выпуклость задачи и, как следствие, корректность применения метода внутренней точки для её решения.

2.5. Условия, налагаемые на функционал.

На функционал нужно наложить ряд ограничений, основным из которых является следующее: если исходные ребра лежали в одной грани, то и "подвижные"ребра должны лежать в одной грани. Иными словами:

$$\forall k = \overline{1, n_3} \ \exists \text{ плоскость } \pi_k : \forall j \in f_k, \overline{\tilde{p}_{j,1}, \tilde{p}_{j,2}} \in \pi_k$$

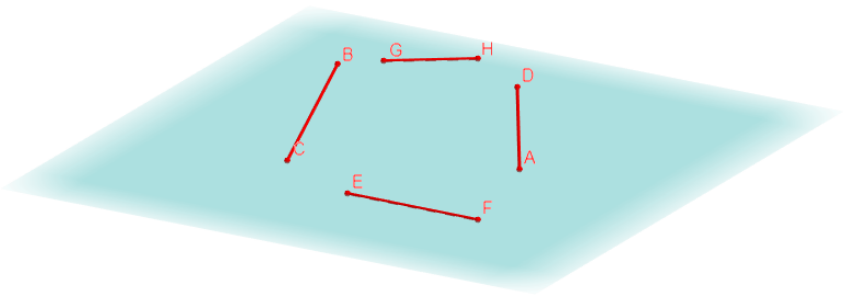
Математически записать условие принадлежности 4-х точек одной плоскости можно с помощью определителя:

$$\begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_4 - x_1 & y_4 - y_1 & z_4 - z_1 \end{vmatrix} = 0$$

Соответственно для n точек это условие будет выглядеть следующим образом:

$$\begin{cases} \begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_4 - x_1 & y_4 - y_1 & z_4 - z_1 \end{vmatrix} = 0 \\ \\ \begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_5 - x_1 & y_5 - y_1 & z_5 - z_1 \end{vmatrix} = 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_n - x_1 & y_n - y_1 & z_n - z_1 \end{vmatrix} = 0 \end{cases}$$

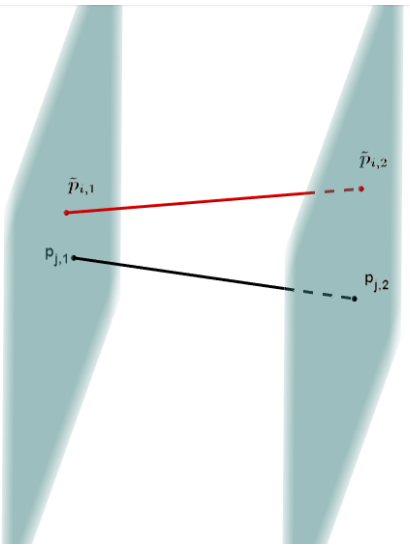
Однако такой способ записи довольно громоздкий и, помимо этого, дает "кубические"условия, налагаемые на задачу. С другой стороны можно ввести новые переменные $A_k, B_k, C_k, D_k, \forall k = \overline{1, n_3}$-коэффициенты из уравнений плоскостей.



Они не участвуют в записи минимализируемого функционала, однако с помощью них условия принадлежности одной грани можно записать достаточно просто:

$$\begin{cases} A_k\tilde{x}_{j,1} + B_k\tilde{y}_{j,1} + C_k\tilde{z}_{j,1} + D_k = 0, \forall j \in f_k \\ A_k\tilde{x}_{j,2} + B_k\tilde{y}_{j,2} + C_k\tilde{z}_{j,2} + D_k = 0, \forall j \in f_k \\ A_k^2 + B_k^2 + C_k^2 = 1 - \text{ нормировка коэффициентов} \end{cases}$$

Не менее важными являются условия принадлежности концов "подвижных"ребер нормальным плоскостям изначальных ребер многогранника (нецелевым). Их можно записать следующим образом:



$$\begin{cases} \tilde{p}_{j,1} \in N(p_{e_j[1]}, e_j) \\ \tilde{p}_{j,2} \in N(p_{e_j[2]}, e_j) \end{cases}$$

где $N(p, e)$-плоскость, нормальная к вектору e и проходящая через точку p.

Данные условия нужны для предотвращения разбегания подвижных ребер.

3. Построение многогранника по набору полупространств.

После решения задачи оптимизации мы имеем наборы подвижных ребер и плоскостей, в которых они лежат. Однако нам хотелось бы по этим данным построить многогранник. Решить эту задачу поможет алгоритм, описанный ниже.

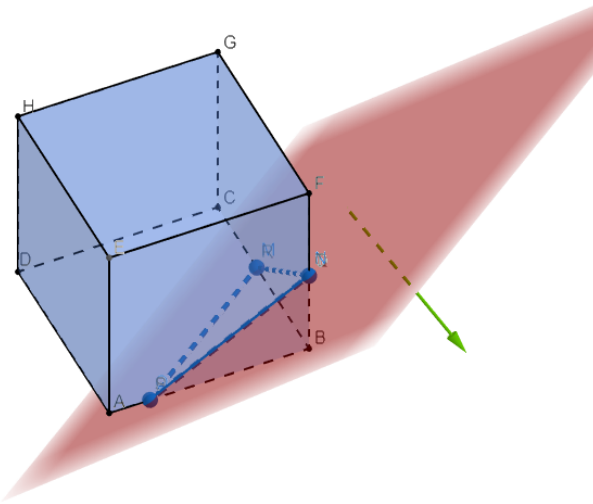
Для начала получим из набора плоскостей-полупространства. Для этого вычислим центр масс исходного многогранника и подставим его в уравнения плоскостей, если получаем положительное число, то плоскость задана внутренней нормалью, поэтому умножим коэффициенты плоскости на -1. Теперь мы имеем

множество полупространств, пересечением которых является искомый многогранник.

Перейдем к алгоритму построения многогранника по набору полупространств:

```
x=1;
cub=createCub(x);
while True do
    oldFacets=cub.facets;
    for plane ∈ planes do
        newFacet=∅;
        newEdges=∅;
        for facet ∈ cub.facets do
            newPoints=∅;
            for edge ∈ facet.edges do
                if edge ∩ plane ≠ ∅ then
                    newPoints=newPoints ∪ (edge ∩ plane);
                end
                reconstructEdge(edge);
            end
            reconstructFacet(facet);
            newEdges=newEdges ∪ edgeConstruct(newPoints)
        end
        newFacet=facetConstruct(newEdges);
        cub.facets=cub.facets ∪ newFacet;
    end
    if cub.facets ∩ oldFacets =∅ then
        break;
    else
        x=2x;
        cub=createCub(x);
    end
end
```

Изначально мы создаем объемлющий куб, от которого будем отсекать полупространства. Опишем процедуру отсечения плоскости от куба. Мы проходим по всем граням объемлющего куба и в каждой грани смотрим на пересечение всех ребер этой грани с отсекаемой плоскостью. После получения точек пересечения мы изменяем ребра (берем только ту часть ребра, которая лежит в нужном полупространстве) и добавляем к грани ребро



пересечения. Ребра, целиком лежащие в противоположном полупространстве-удаляются. Запоминая ребра пересечения мы формируем из них новую грань объемлющего куба (теперь это уже не куб, а объемлющий многогранник) и добавляем ее к его структуре. Проделывая так со всем списком полупространств мы получаем некий многогранник. Однако исходно объемлющий куб мог быть слишком мал, поэтому если после пересечения полупространств в многограннике осталась часть грани исходного объемлющего куба, то мы увеличиваем его размер вдвое и запускаем процесс заново.

Следует отметить, что алгоритм требует рассмотрения большого числа крайних случаев: пересечение по вершине, ребру, плоскости.

4. Средства программной реализации задачи.

4.1. IPOPT

Метод внутренней точки реализован в библиотеке IPOPT. Опишем процесс установки этой библиотеки.

1) Скачаем библиотеку из репозитория:

```
$ svn co https://projects.coin-or.org/svn/Ipopt/stable/3.12 CoinIpopt
```

2) Перейдем в папку с дистрибутивом IPOPT:

```
$ cd CoinIpopt
```

3) Скачаем сторонние библиотеки, необходимые для сборки IPOPT:

```
$ cd /ThirdParty/Blas
$ ./get.Blas
$ cd ..
$cd Lapack
$ ./get.Lapack
$ cd ..
$cd ASL
$ ./get.ASL
и т.д. для всех папок в ThirdPart
```

4) Создаём каталог build:

```
$ mkdir build
```

и переходим в него:

```
$ cd build
```

5) Запускаем скрипт configure:

```
$ ../configure
```

6) Собираем:

```
$ make
```

7) Запускаем короткий тест, чтобы проверить, что компиляция прошла успешно:

```
$ make test
```

8) Устанавливаем IPOPT:

```
$ make install
```

После этого в каталоге bin должен появиться исполняемый файл ipopt.

9) Переходим в корневую папку:

```
$ cd
```

И затем в bin:

```
$ cd bin
```

Сохраняем в эту папку файл ipopt.

4.2. C++ Интерфейс

Основным классом, с которым предстоит работать является MyNLP, унаследованный от стандартного класса Ipopt::TNLP. В нем должны быть определены следующие методы класса:

1) `bool MyNLP::get_nlp_info(Index& n, Index& m, Index& nnz_jac_g, Index& nnz_h_lag, IndexStyleEnum& index_style)`

   Задает общую информацию о задаче: число переменных, число ограничений, число ненулевых элементов матрицы Якоби для ограничений, число ненулевых элементов матрицы Гесса для функции Лагранжа.

2) `bool MyNLP::get_bounds_info(Index n, Number* x_l, Number* x_u, Index m, Number* g_l, Number* g_u)`

   Задает ограничения на переменные и на условия. Если ограничений нет, то

   ```
   x_l[i] = -1.0e19;
   x_u[i] = 1.0e19;
   ```

3) `bool MyNLP::get_starting_point(Index n, bool init_x, Number* x, bool init_z, Number* z_L, Number* z_U, Index m, bool init_lambda, Number* lambda)`

   Задает стартовую точку алгоритма. Важно, чтобы она удовлетворяла всем условиям g. Алгоритм не просто так называется алгоритмом внутренней точки.

4) `bool MyNLP::eval_f(Index n, const Number* x, bool new_x, Number& obj_v`

   Здесь нужно задать минимализируемый функционал.

5) `bool MyNLP::eval_grad_f(Index n, const Number* x, bool new_x, Number*`

   Посчитать градиент от функционала.

6) `bool MyNLP::eval_g(Index n, const Number* x, bool new_x, Index m, Numb`

   В этой функции мы задаем ограничения на функционал.

7) `bool MyNLP::eval_jac_g(Index n, const Number* x, bool new_x, Index m, Index nele_jac, Index* iRow, Index *jCol, Number* values)`

   Здесь задается структура ненулевых элементов матрицы Якоби.

```
iRow[i]=rowNumber;
jCol[i] = colNumber;
```

А затем значение этого элемента

```
values[i]=value;
```

8) `bool MyNLP::eval_h(Index n, const Number* x, bool new_x, Number obj_factor, Index m, const Number* lambda, bool new_lambda, Index nele_hess, Index* iRow, Index* jCol, Number* values)`

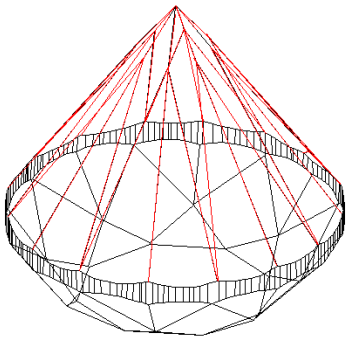
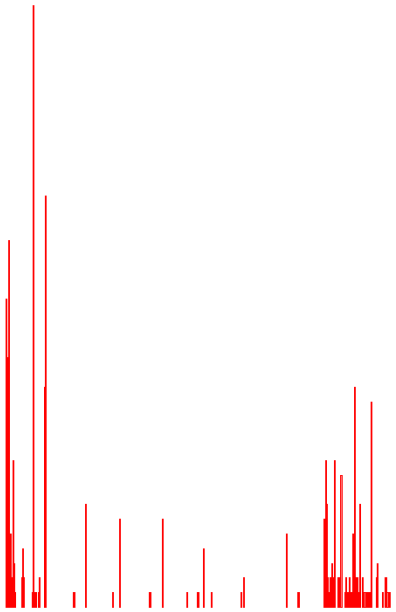
   Вычисление Гессиана от Лагранжиана функционала. Можно пропустить, написав специальную команду в опциях компиляции.

9) `void MyNLP::finalize_solution(SolverReturn status, Index n, const Number* x, const Number* z_L, const Number* z_U, Index m, const Number* g, const Number* lambda, Number obj_value, const IpoptData* ip_data, IpoptCalculatedQuantities* ip_cq)`

   То, что будет выполнено по завершению поиска оптимума. Здесь я запускаю построение многогранника по набору полупространств и его последующую отрисовку.

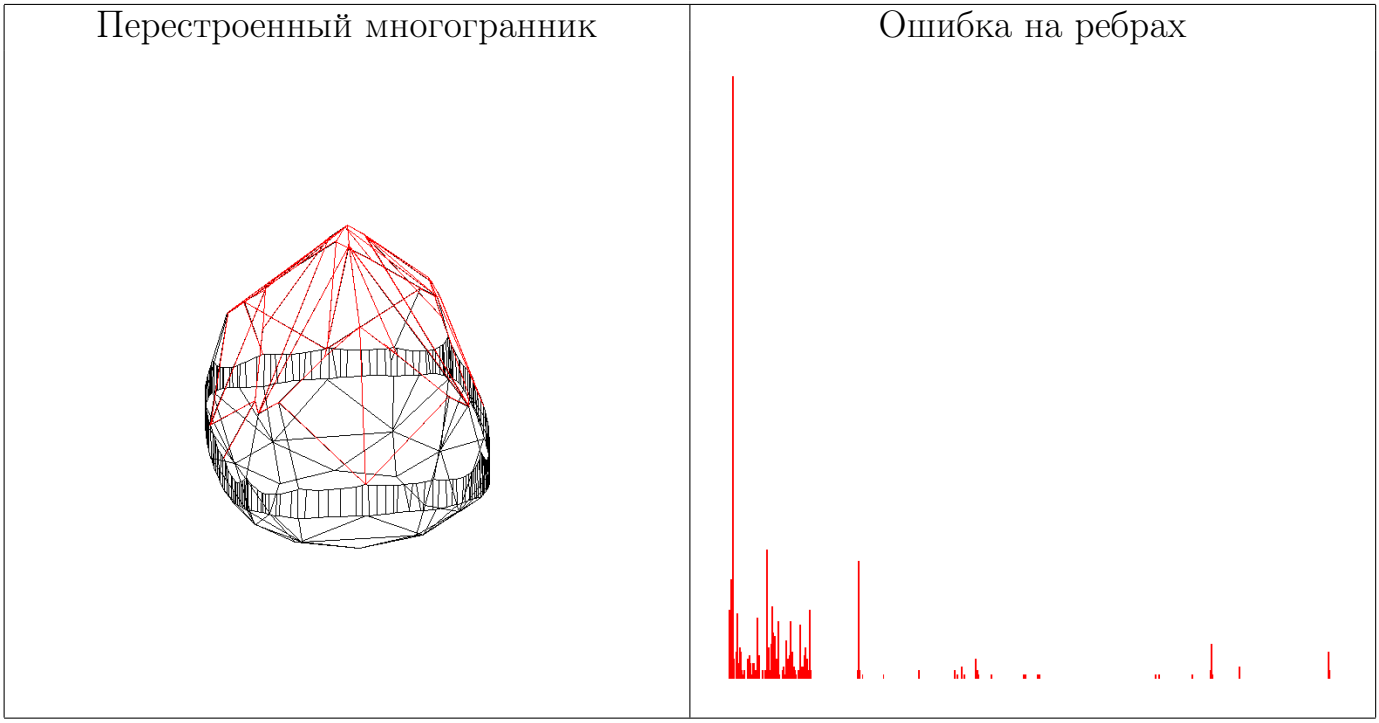
5. Результаты работы программной реализации модели.

5.1. Первый тестовый многогранник.

Перестроенный многогранник	Ошибка на ребрах
	

Значение функционала: $3.9 * 10^{-4}$
Время работы: 405 секунд
Число итераций: 478
Число вершин: 442
Число ребер: 663
Число граней: 223
Число целевых ребер: 35
Число переменных: 4870
Число ненулевых элементов в Якобиане: 9951

5.2. Второй тестовый многогранник.



Значение функционала: $7.6 * 10^{-4}$
Время работы: 635 секунд
Число итераций: 526
Число вершин: 536
Число ребер: 804
Число граней: 270
Число целевых ребер: 68
Число переменных: 5904
Число ненулевых элементов в Якобиане: 12066

6. Ускорение работы на больших моделях.

К сожалению, на больших моделях алгоритм работает долго. Однако из гистограмм ошибок можно заметить, что на более чем половине ребер удаление от целевых ребер-нулевое. Это ребра нижней части кристалла, для которых изначально нет целевых ребер и они притягиваются сами к себе. В результате этого они являются статичными и их движения не происходит, однако обсчет их движения происходит. Будем теперь включать в функционал только ребра, лежащие в гранях, в которые входит хотя бы одно целевое ребро. Такая модель несколько сложнее программируется, но, даст ускорение в десятки раз. Так

первый кристалл приближается за 10 секунд после 37 итераций, а второй-за 20 секунд после 41 итерации.

7. Сведение задачи к задаче безусловной оптимизации

Будем теперь считать неизвестными уравнения плоскостей, в которых лежат грани кристалла. Для дальнейших рассуждений введем следующие обозначения:

$$\begin{cases} e_i = (p_i^0, p_i^1) \text{ - ребра многогранника} \\ \widetilde{e_i} = (\widetilde{p_i^0}, \widetilde{p_i^1}) \text{ - соотвествующие целевые ребра} \\ \left.\begin{array}{l} \Pi_i^0 : A_i^0 x + B_i^0 y + C_i^0 z + D_i^0 = 0 \\ \Pi_i^1 : A_i^1 x + B_i^1 y + C_i^1 z + D_i^1 = 0 \end{array}\right\} \Pi_i^0 \cap \Pi_i^1 = e_i \\ N_i^0 : a_i^0 x + b_i^0 y + c_i^0 z + d_i^0 = 0 : N_i^0 \perp e_i, p_i^0 \in N_i^0 \\ N_i^1 : a_i^1 x + b_i^1 y + c_i^1 z + d_i^1 = 0 : N_i^1 \perp e_i, p_i^1 \in N_i^1 \end{cases}$$

Из этих обозначений напряую следует, что вершины ребер

$$p_i^0 = (x_i^0, y_i^0, z_i^0)$$

$$p_i^1 = (x_i^1, y_i^1, z_i^1)$$

являются соответственно решениями систем:

$$\begin{cases} A_i^0 x_i^0 + B_i^0 y_i^0 + C_i^0 z_i^0 + D_i^0 = 0 \\ A_i^1 x_i^0 + B_i^1 y_i^0 + C_i^1 z_i^0 + D_i^1 = 0 \\ a_i^0 x_i^0 + b_i^0 y_i^0 + c_i^0 z_i^0 + d_i^0 = 0 \end{cases} \tag{1}$$

$$\begin{cases} A_i^0 x_i^1 + B_i^0 y_i^1 + C_i^0 z_i^1 + D_i^0 = 0 \\ A_i^1 x_i^1 + B_i^1 y_i^1 + C_i^1 z_i^1 + D_i^1 = 0 \\ a_i^1 x_i^1 + b_i^1 y_i^1 + c_i^1 z_i^1 + d_i^1 = 0 \end{cases} \tag{2}$$

Откуда следуют зависимости:

$$x_i^0 = x_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)$$

$$y_i^0 = y_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)$$

$$z_i^0 = z_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)$$

$$x_i^1 = x_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)$$

$$y_i^1 = y_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)$$
$$z_i^1 = z_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)$$

Введем теперь функционал, который будем минимализировать:

$$\Phi = \sum_{i=0}^{n} \rho^2(\widetilde{p_i^0}, p_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)) + \rho^2(\widetilde{p_i^1}, p_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)) \to \min \quad (3)$$

$$\Phi = \sum_{i=0}^{n} (\widetilde{x_i^0} - x_i^0)^2 + (\widetilde{y_i^0} - y_i^0)^2 + (\widetilde{z_i^0} - z_i^0)^2 + (\widetilde{x_i^1} - x_i^1)^2 + (\widetilde{y_i^1} - y_i^1)^2 + (\widetilde{z_i^1} - z_i^1)^2 \to \min \quad (4)$$

Осуществять поиск минимума будем методом градиентного спуска. Соответсвенно необходимо уметь считать градиент: $\nabla\Phi = (\frac{\partial\Phi}{\partial A_0^0}, \ldots, \frac{\partial\Phi}{\partial D_n^1})$
Для примера рассмотрим компоненту градиента, вычисленую по формуле сложной функции

$$\frac{\partial\Phi}{\partial A_i^0} = -2(\widetilde{x_i^0} - x_i^0)\frac{\partial x_i^0}{\partial A_i^0} - 2(\widetilde{y_i^0} - y_i^0)\frac{\partial y_i^0}{\partial A_i^0} - 2(\widetilde{z_i^0} - z_i^0)\frac{\partial z_i^0}{\partial A_i^0}$$
$$-2(\widetilde{x_i^1} - x_i^1)\frac{\partial x_i^1}{\partial A_i^0} - 2(\widetilde{y_i^1} - y_i^1)\frac{\partial y_i^1}{\partial A_i^0} - 2(\widetilde{z_i^1} - z_i^1)\frac{\partial z_i^1}{\partial A_i^0}$$

Остальные компоненты градиента имеют аналогичный вид. Отсюда следует, что для вычисления градиента $\nabla\Phi$ достаточно знать значения координат вершин $x_i^0, \ldots, z_i^1$ и их градиенты $\nabla x_i^0, \ldots, \nabla z_i^1$. Если для получения значений координат вершин достаточно решить системы 1 и 2, то для нахождения градиентов эти системы необходимо продифференцировать по $A_i^0, \ldots, D_i^1$, а затем решить относительно частных производных. Так, если продифференцировать 1 по A_i^0, то получим:

$$\begin{cases} x_i^0 + A_i^0\frac{\partial x_i^0}{\partial A_i^0} + B_i^0\frac{\partial y_i^0}{\partial A_i^0} + C_i^0\frac{\partial z_i^0}{\partial A_i^0} = 0 \\ A_i^1\frac{\partial x_i^0}{\partial A_i^0} + B_i^1\frac{\partial y_i^0}{\partial A_i^0} + C_i^1\frac{\partial z_i^0}{\partial A_i^0} = 0 \\ a_i^0\frac{\partial x_i^0}{\partial A_i^0} + b_i^0\frac{\partial y_i^0}{\partial A_i^0} + c_i^0\frac{\partial z_i^0}{\partial A_i^0} = 0 \end{cases} \Leftrightarrow \begin{cases} A_i^0\frac{\partial x_i^0}{\partial A_i^0} + B_i^0\frac{\partial y_i^0}{\partial A_i^0} + C_i^0\frac{\partial z_i^0}{\partial A_i^0} = -x_i^0 \\ A_i^1\frac{\partial x_i^0}{\partial A_i^0} + B_i^1\frac{\partial y_i^0}{\partial A_i^0} + C_i^1\frac{\partial z_i^0}{\partial A_i^0} = 0 \\ a_i^0\frac{\partial x_i^0}{\partial A_i^0} + b_i^0\frac{\partial y_i^0}{\partial A_i^0} + c_i^0\frac{\partial z_i^0}{\partial A_i^0} = 0 \end{cases}$$

$$\begin{pmatrix} A_i^0 & B_i^0 & C_i^0 \\ A_i^1 & B_i^1 & C_i^1 \\ a_i^0 & b_i^0 & c_i^0 \end{pmatrix} \begin{pmatrix} \frac{\partial x_i^0}{\partial A_i^0} \\ \frac{\partial y_i^0}{\partial A_i^0} \\ \frac{\partial z_i^0}{\partial A_i^0} \end{pmatrix} = \begin{pmatrix} -x_i^0 \\ 0 \\ 0 \end{pmatrix}$$

Решив систему получим $(\frac{\partial x_i^0}{\partial A_i^0}, \frac{\partial y_i^0}{\partial A_i^0}, \frac{\partial z_i^0}{\partial A_i^0})$. Решив остальные системы получим все значения $\nabla x_i^0, \ldots, \nabla z_i^1$, а следоательно и $\nabla \Phi$. Теперь можно применять градиентный спуск.

Список литературы

[1] Веселов А. П., Троицкий Е. В. Лекции по аналитической геометрии. Учебное пособие. — Изд. новое. — М.: МЦНМО, 2016. — 152 с.

[2] Алексеев В.М., Галеев Э.М., Тихомиров В.М. Сборник задач по оптимизации. Теория. Примеры. Задачи: Учеб. пособие. — 2-е изд. М.: ФИЗМАТЛИТ, 2005. — 256 с.

[3] О.А. Щербина КРАТКОЕ ВВЕДЕНИЕ В AMPL - СОВРЕМЕННЫЙ АЛГЕБРАИЧЕСКИЙ ЯЗЫК МОДЕЛИРОВАНИЯ (препринт), 2012. — 29 с.

[4] Попов А.В. GNUPLOT и его приложения. — М.: Издательство попечительского совета механико-математического факультета МГУ, 2015 , —240 с.

[5] Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt. July 20, 2016