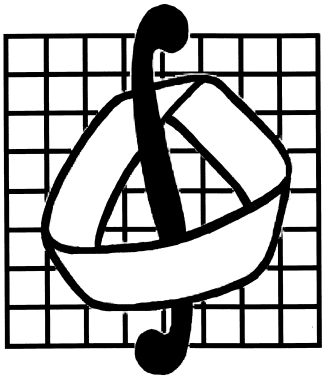
Moscow STATE UNIVERSITY
named after M. V. LOMONOSOV
Faculty of mechanics and mathematics



Coursework.

Approximation of a polyhedron by a set of spatial segments. Building a polyhedron from a set of half-spaces.

Supervisor: V. D. Valedinsky

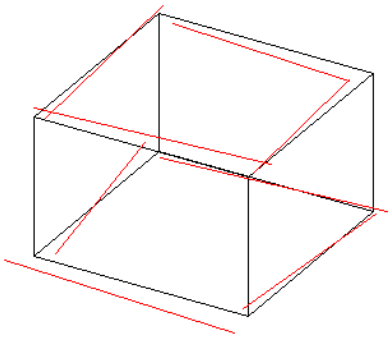
Student: M. N. Kovalkov

Содержание

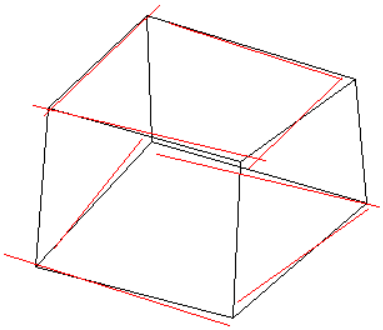
1. The problem of approximation of a polyhedron by a set of spatial segments.

1.1. Problem statement.

There is a polyhedron and a set of target edges. It is assumed that the polyhedron is constructed inaccurately, and the target edges are accurate. You need to change the polyhedron so that it best matches the target edges.



The original polyhedron



The modified polyhedron

1.2. Mathematical formalization of the problem. Build minimaliseren functionality

There is a polyhedron M defined by sets of its faces, edges, and vertices.

$$\begin{cases} p_i = (x_i, y_i, z_i) - & \text{Coordinates of the polyhedron vertices,} i = \overline{1, n_1} \\ e_j = (a_{j,1}, a_{j,2}) - & \text{Numbers of vertexes that are edge ends } e_j, j = \overline{1, n_2} \\ f_k = [b_{k,1}, \ldots, b_{k,m_k}] - & \text{Numbers of edges included in the face } f_k, k = \overline{1, n_3} \end{cases}$$

There is a list of target edges that are considered accurate and that correspond to the original edges of the polyhedron.

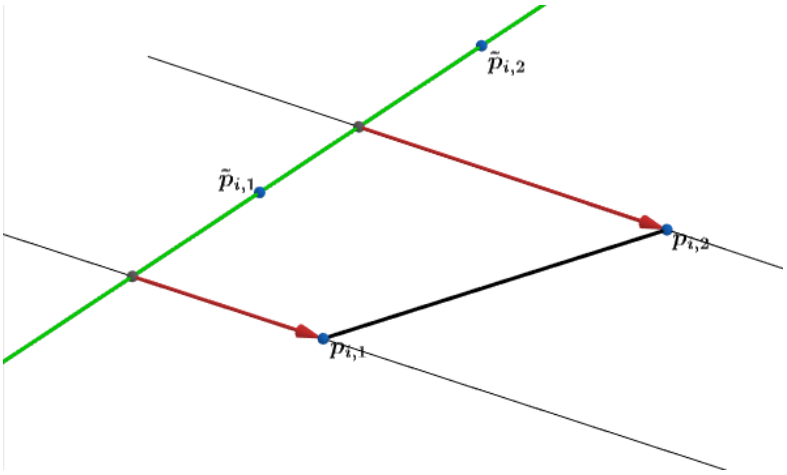
$$\widetilde{e_i} = \begin{cases} (a_{i,x}, a_{i,y}, a_{i,z}), (b_{i,x}, b_{i,y}, b_{i,z}) - \text{coordinates of the two ends of this target edge.} \\ ind_i - \text{number of the source edge, which corresponds to the I-th target edge.} \end{cases}$$

If there is no element for the current edge in the list of target edges, we assume that it is a target for itself. However, the weight with which it is included in the functional will be equal to $\omega_i = 0.1$, while for a full target edge $\omega_i = 1$. After expanding the list of target edges in the specified way, we assume that $\widetilde{e_i}$is the target edge corresponding to the original edge e_i of the polyhedron M.

Do the edges of a polyhedron "movable for this we introduce variables $\tilde{p}_{i,1} = (\tilde{x}_{i,1}, \tilde{y}_{i,1}, \tilde{z}_{i,1}), \tilde{p}_{i,1} = (\tilde{x}_{i,2}, \tilde{y}_{i,2}, \tilde{z}_{i,2})$ corresponding to ends of "moving"edges. We want to find a configuration so that these edges are the least distant from their corresponding targets. To do this, we must deliver a minimum of the following functionality:

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

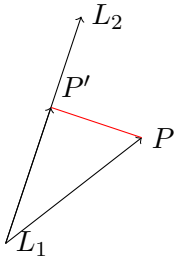
$\rho(\tilde{p}_{i,j}, \widetilde{e_i})$-the distance between the ends of the "movable"edge and the line defined by the corresponding target edge.



1.3. Calculating the distance from a point to a straight line.

To calculate the distance, use the formula for calculating the projection of the vector $\overrightarrow{a}$ on the vector $\overrightarrow{b}$:

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



(* ,*) – standard scalar product:

$$(\overrightarrow{a}, \overrightarrow{b}) = a_x b_x + a_y b_y + a_z b_z$$

Then we use the Pythagorean theorem to calculate the distance:

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

1.4. Proof of convexity of the functional.

For correct application of the inner point method, the convexity of the minimalized functional is required. We prove that the functional we are considering is convex. Let's consider one summand of the sum that forms our functional:

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

We introduce the notation: $c_1 := (b_x-a_x), c_2 := (b_y-a_y), c_3 := (b_z-a_z), c := c_1^2+c_2^2+c_3^2$. And write the matrix of second partial derivatives F:

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

Let's check the matrix for nonnegative certainty using the Sylvester Criterion. To do this, we calculate the determinants of major minors.

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) $|\Delta_3| = \begin{vmatrix} 2(1-\frac{c_1^2}{c}) & -\frac{c_1c_2}{c} & -\frac{c_1c_3}{c} \\ -\frac{c_1c_2}{c} & 2(1-\frac{c_2^2}{c}) & -\frac{c_2c_3}{c} \\ -\frac{c_1c_3}{c} & -\frac{c_2c_3}{c} & 2(1-\frac{c_3^2}{c}) \end{vmatrix} = 8(1-\frac{c_1^2}{c})(1-\frac{c_2^2}{c})(1-\frac{c_3^2}{c})-2\frac{c_1^2c_2^2c_3^2}{c^3}-$

$-2\frac{c_1^2c_3^2}{c^2}(1-\frac{c_2^2}{c})-2\frac{c_2^2c_3^2}{c^2}(1-\frac{c_1^2}{c})-\frac{c_1^2c_2^2}{c^2}(1-\frac{c_3^2}{c}) = 8-4\frac{c_1^2c_2^2c_3^2}{c^3}+6(\frac{c_1^2c_2^2}{c^2}+\frac{c_1^2c_3^2}{c^2}+\frac{c_2^2c_3^2}{c^2})-8\frac{c_1^2+c_2^2+c_3^2}{c} =$

$= 6(\frac{c_1^2c_2^2}{c^2}+\frac{c_1^2c_3^2}{c^2})+2(\frac{c_2^2c_3^2}{c^2})+4+\frac{c_2^2c_3^2}{c^2}(1-\frac{c_1^2}{c}) = 6(\frac{c_1^2c_2^2}{c^2}+\frac{c_1^2c_3^2}{c^2})+2(\frac{c_2^2c_3^2}{c^2})+4+\frac{c_2^2c_3^2}{c^2}(\frac{c_2^2+c_3^2}{c}) \geq 0$

Thus, we have shown that the matrix of the second partial derivatives of the function is non-negative, hence this function is convex. It remains to remember that the sum of convex functions is also convex (a Direct consequence of the definition via Jensen's inequality).

Thus, the convexity of the problem and, as a result, the correctness of using the inner point method for its solution is shown.

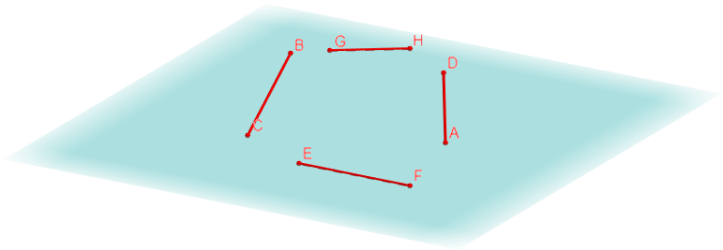
1.5. Conditions imposed on the functionality.

A number of restrictions must be imposed on the functionality, the main of which is the following: if the original edges lay in the same face, then the "moving"edges must lie in the same face. In other words:

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

Mathematically, you can use the determinant to write the condition that 4 points belong to the same plane:

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



Accordingly, for n points, this condition will look like this:

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

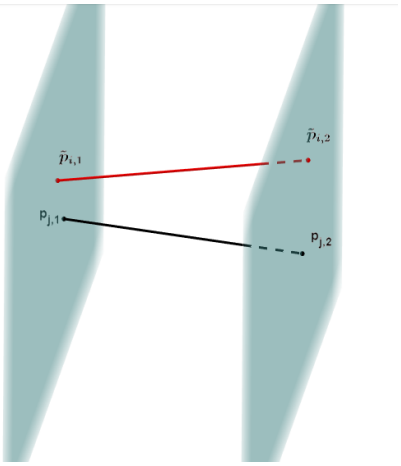
However, this method of writing is quite cumbersome and, in addition, gives "cubic"conditions imposed on the problem. On the other hand, you can enter new variables $A_k, B_k, C_k, D_k, \forall k = \overline{1, n_3}$-coefficients from the equations of the planes. They do not participate in writing the minimalized functional, but they can be used to write the conditions of belonging to a single face quite simply:

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

Equally important are the conditions for the ends of the "moving"edges to belong to the normal planes of the original edges of the polyhedron (non-target). They can be written as follows:

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

where $N(p, e)$is a plane normal to the vector e and passing through the point p.



These conditions are necessary to prevent the mobile edges from running away.

2. Building a polyhedron from a set of half-spaces.

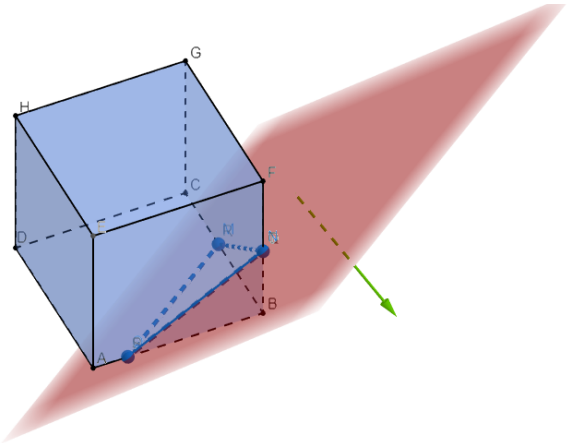
After solving the optimization problem, we have sets of moving edges and the planes in which they lie. However, we would like to use this data to construct a polyhedron. The algorithm described below will help you solve this problem.

To begin with, we get half-spaces from a set of planes. To do this, we calculate the center of mass of the original polyhedron and substitute it in the equations of planes. if we get a positive number, then the plane is given by the internal normal, so we multiply the coefficients of the plane by -1. Now we have a set of half-spaces whose intersection is the desired polyhedron.

Let's go to the algorithm for constructing a polyhedron based on a set of half-spaces:

```
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
```

Initially, we create an enclosing cube, from which we will cut off half-spaces. Let's describe the procedure for clipping a plane from a cube. We go through all the faces of the enclosing cube and in each face we look at the intersection of all the edges of this face with the plane being cut off. After getting the intersection points, we change the edges (taking only the part of the edge that lies in the desired half-space) and add the edge to the face. Edges that lie entirely in the opposite half-space are deleted. By memorizing the intersection edges, we form a new face of the enclosing cube (now it is no longer a cube, but an enclosing polyhedron) and add it to its structure. By doing this with the entire list of half-spaces, we get a polyhedron. However,



the original enclosing cube may have been too small, so if a part of the face of the original enclosing cube remains in the polyhedron after the intersection of the half-spaces, then we double its size and start the process again.

It should be noted that the algorithm requires consideration of a large number of extreme cases: intersections along a vertex, edge, or plane.

3. Tools for software implementation of the task.

3.1. IPOPT

The inner point method is implemented in the IPOPT library. Let's describe the installation process for this library.

1) Download the library from the repository:

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

2) Go to the folder with the IPOPT distribution:

```
$ cd CoinIpopt
```

3) Download the third-party libraries needed to build IPOPT:

```
$ cd /ThirdParty/Blas
$ ./get.Blas
$ cd ..
$cd Lapack
$ ./get.Lapack
$ cd ..
$cd ASL
$ ./get.ASL
etc for all folders in ThirdPart
```

4) Creating the build directory:

```
$ mkdir build
```

and go to it:

```
$ cd build
```

5) Run the configure script:

```
$ ../configure
```

6) Collect:

```
$ make
```

7) Run a short test to check that the compilation was successful:

```
$ make test
```

8) Setting IPOPT:

```
$ make install
```

After that, the ipopt executable file should appear in the bin directory.

9) Go to the root folder:

```
$ cd
```

And then in the bin:

```
$ cd bin
```

Save the ipopt file to this folder.

3.2. C++ Interface

The main class to work with is MyNLP, which is inherited from the standard Ipopt::TNLP class. It must define the following methods of the class:

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

Sets General information about the problem: the number of variables, the number of constraints, the number of non-zero elements of the Jacobi matrix for constraints, and the number of non-zero elements of the Hess matrix for the Lagrange function.

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

Sets restrictions on variables and conditions. If there are no restrictions, then

```
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)`

Sets the starting point of the algorithm. It is important that it satisfies all the conditions of g. the Algorithm is called the inner point algorithm for a reason.

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

   Here you need to specify minimaliseren functionality.

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

   Calculate the gradient from the functional.

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

   In this function, we set restrictions on the functionality.

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)
   ```

   Here we define the structure of non-zero elements of the Jacobi matrix.

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

   And then the value of this element

   ```
   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)
   ```

   Calculating The Hessian from the Lagrangian of the functional. You can skip it by writing a special command in the compilation options.

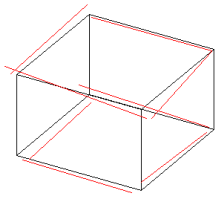
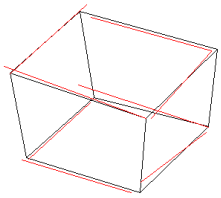
9) ```
   void MyNLP::finalize_solution(SolverReturn status,
   Index n, const Number* x, const Number* z_L, const Number* z_U,
   Index m, const Number* g, const Number* lambda,
   Number obj_value,
   const IpoptData* ip_data,
   IpoptCalculatedQuantities* ip_cq)
   ```

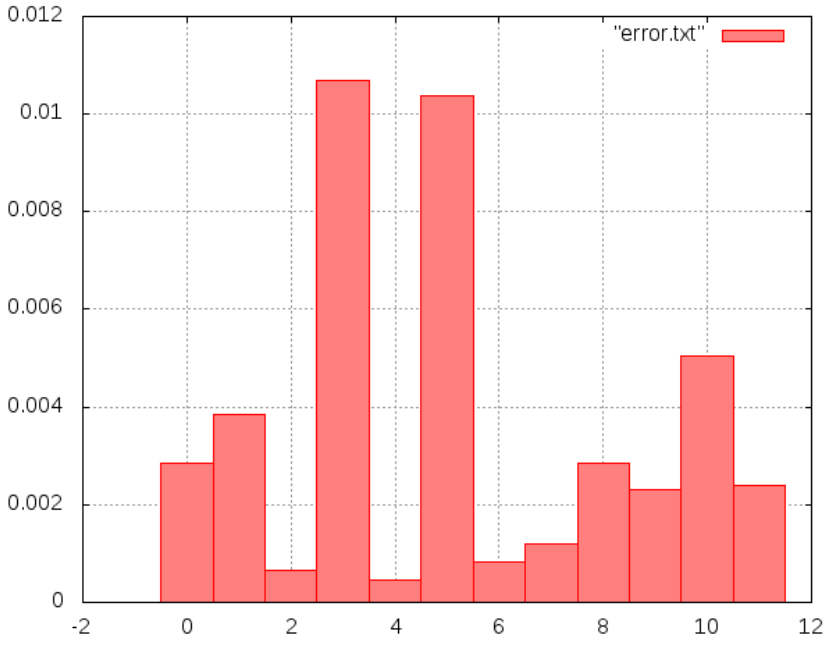
   What will be performed when the search for the optimum is completed. Here I start building a polyhedron from a set of half-spaces and then drawing it.

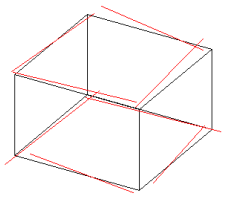
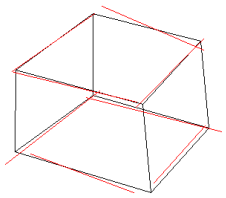
4. Results of the program implementation of the model.

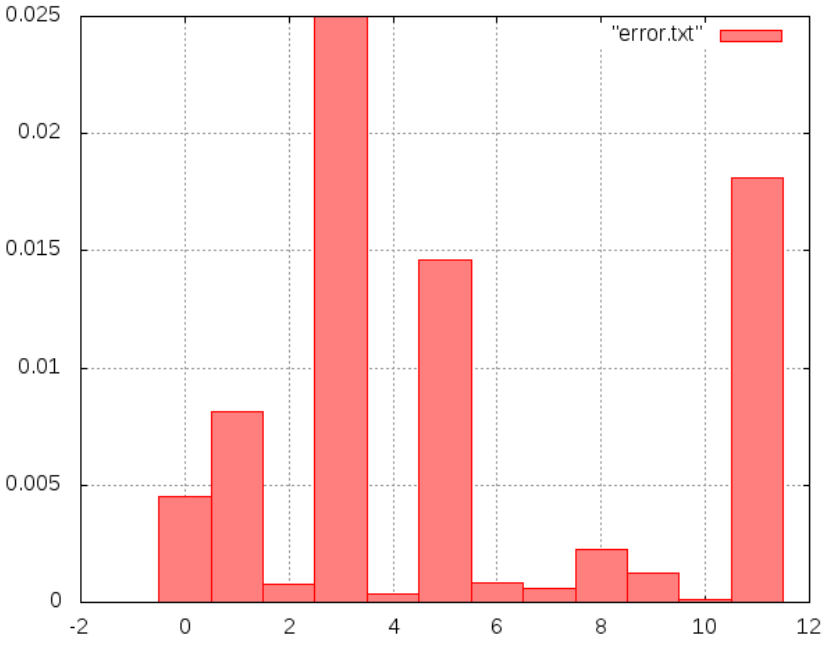
4.1. Test model of the cube

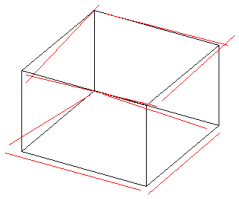
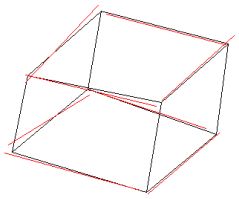
4.1.1. Small declinations $|\Delta x_i| < 0.1, |\Delta y_i| < 0.1, |\Delta z_i| < 0.1$
   ```

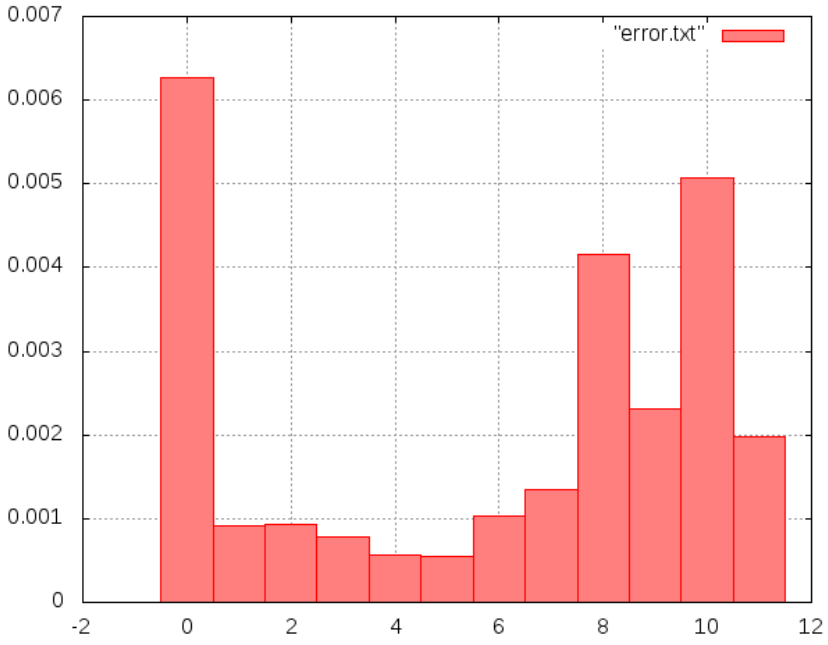
The original polyhedron	Rebuilt polyhedron
	

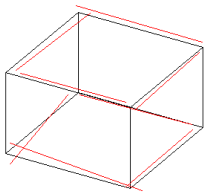
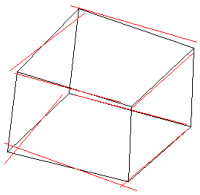


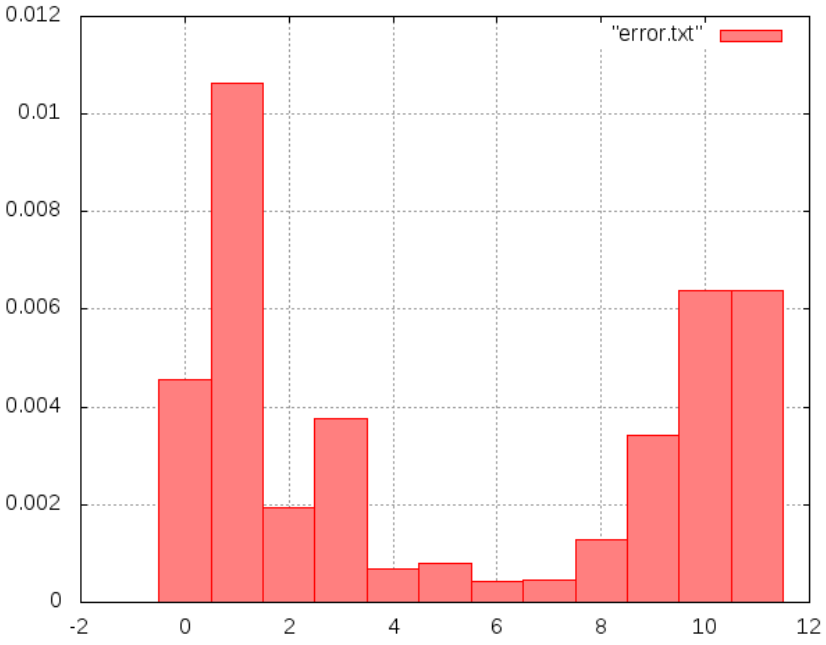
The original polyhedron	Rebuilt polyhedron
	

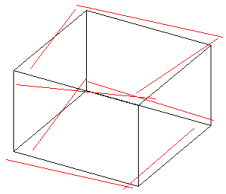
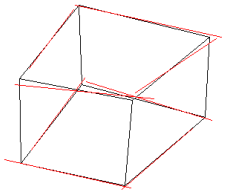


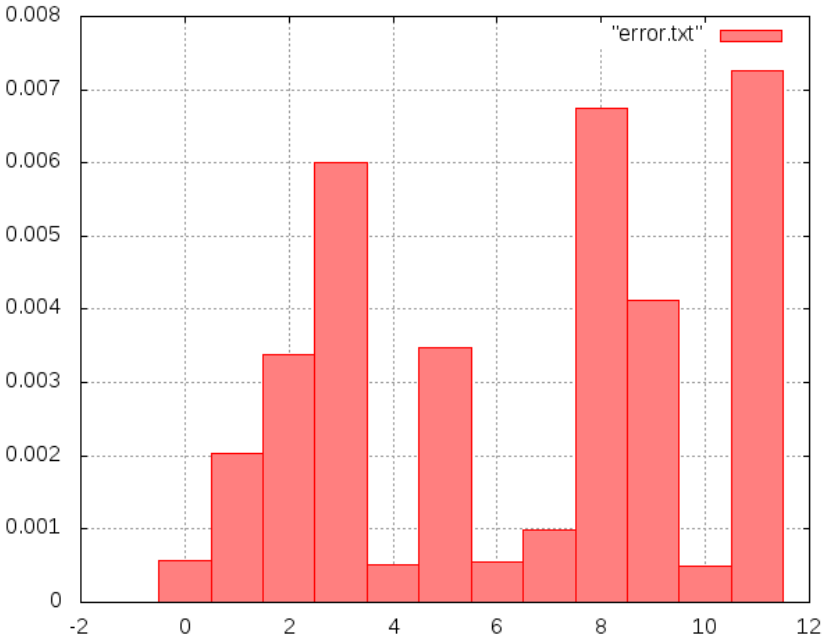
The original polyhedron	Rebuilt polyhedron
	



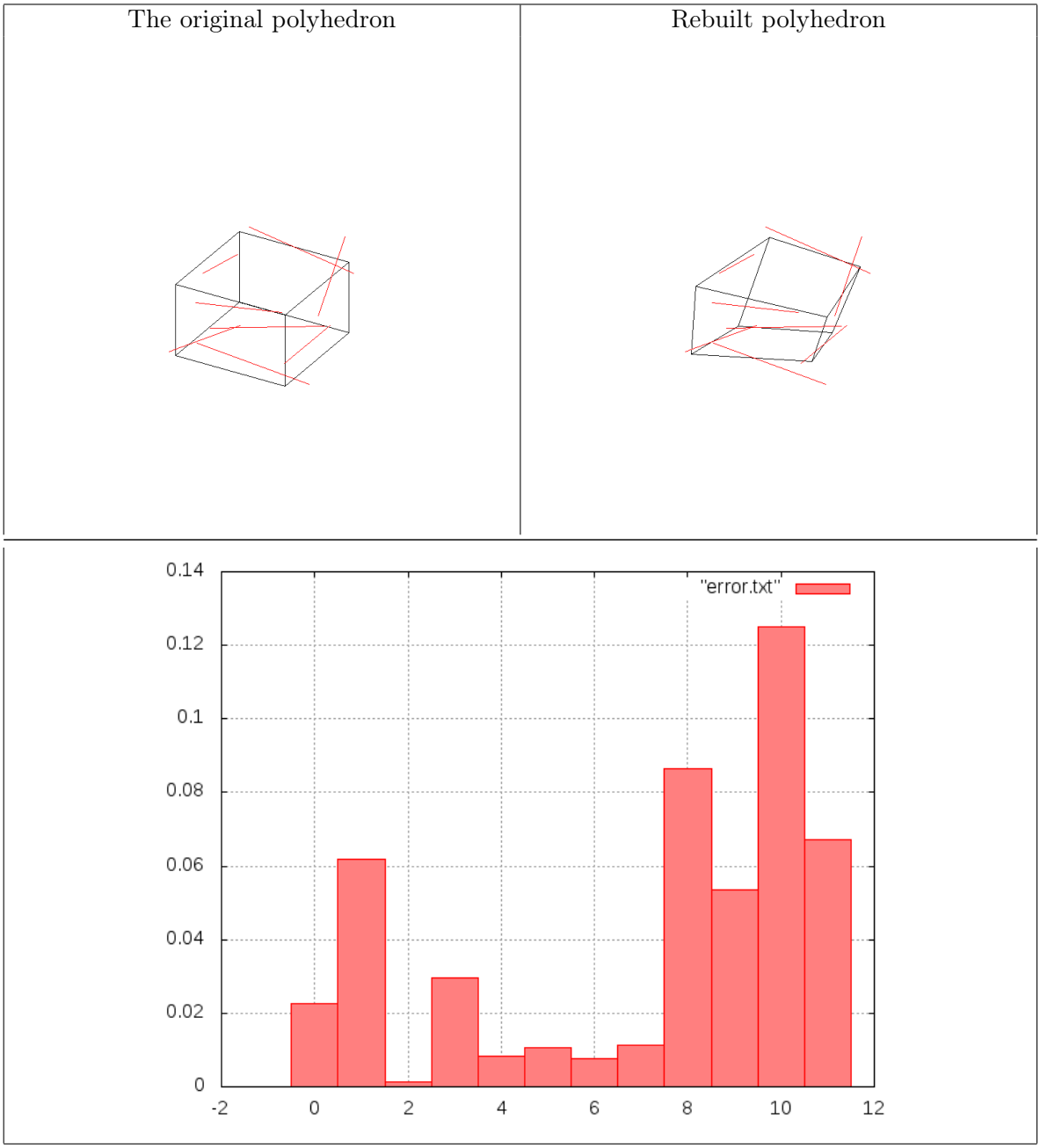
The original polyhedron	Rebuilt polyhedron
	

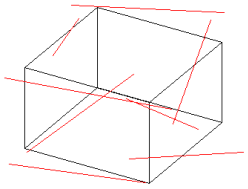
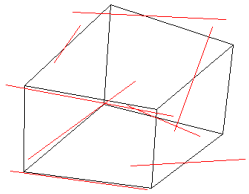


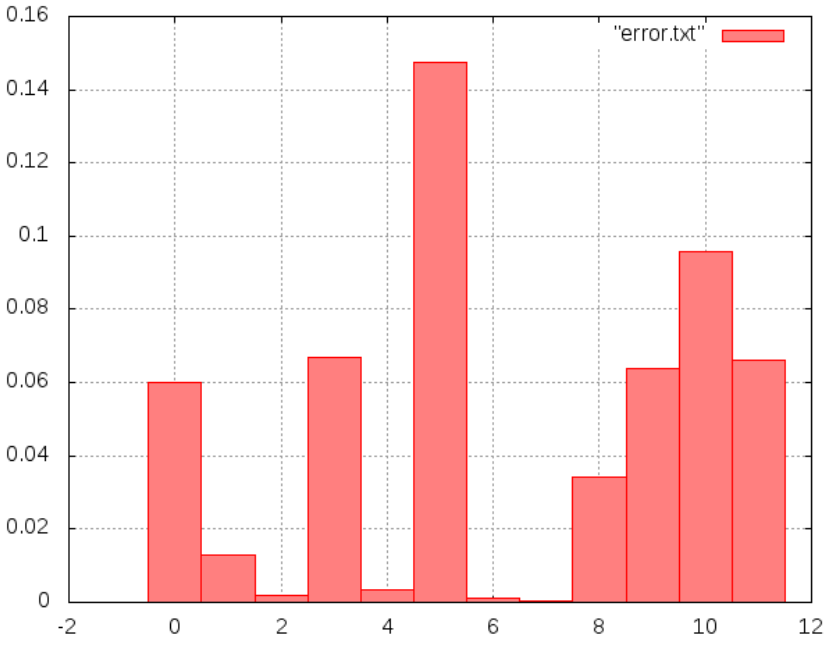
The original polyhedron	Rebuilt polyhedron
	

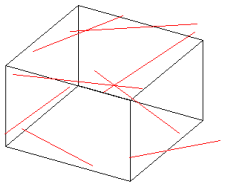
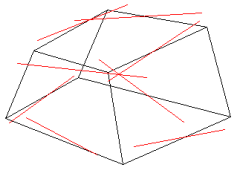


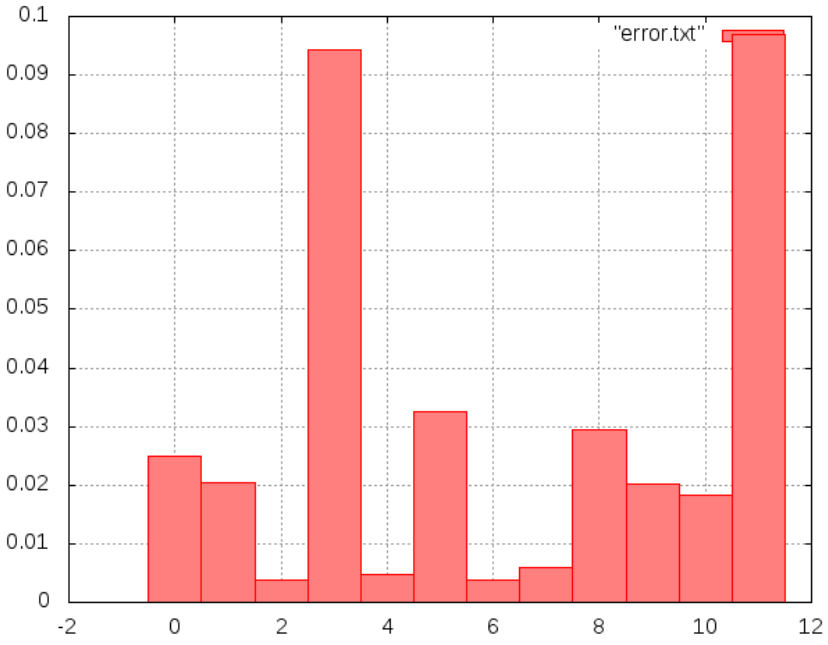
4.1.2. Average deviations $|\Delta x_i| < 0.25, |\Delta y_i| < 0.25, |\Delta z_i| < 0.25$

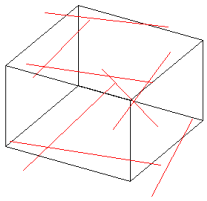
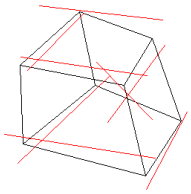


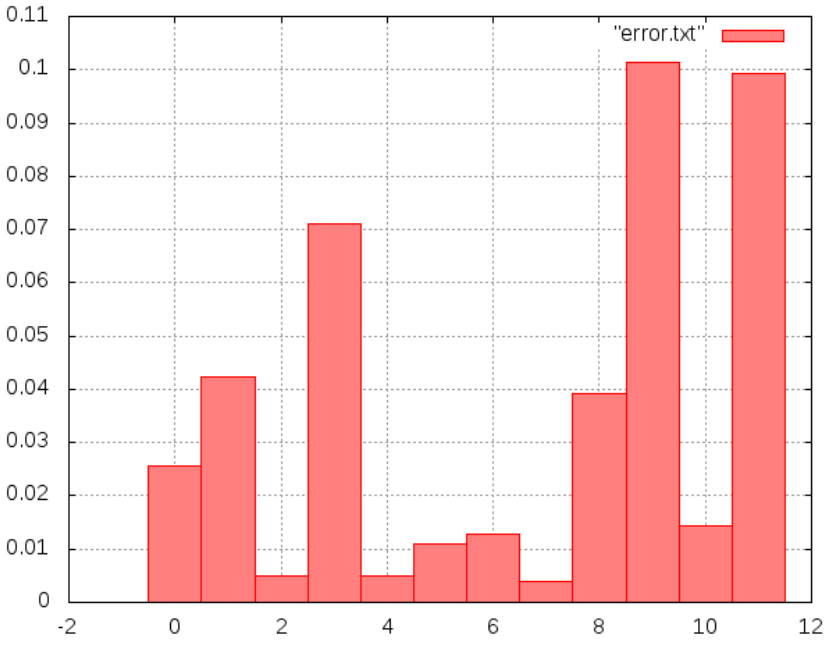
The original polyhedron	Rebuilt polyhedron
	

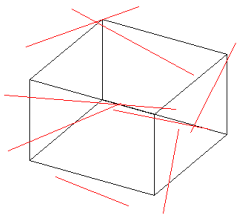
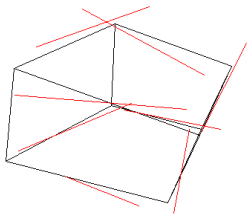


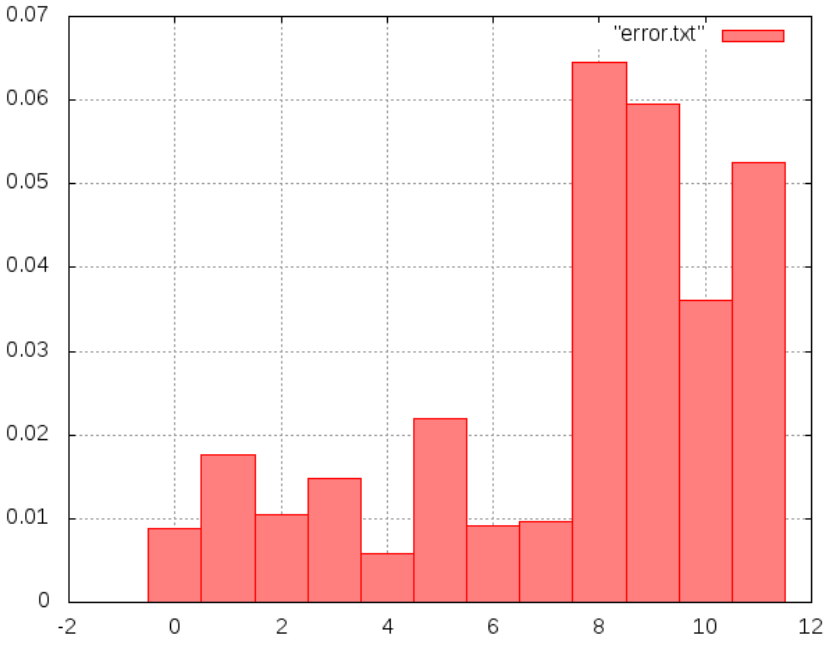
The original polyhedron	Rebuilt polyhedron
	



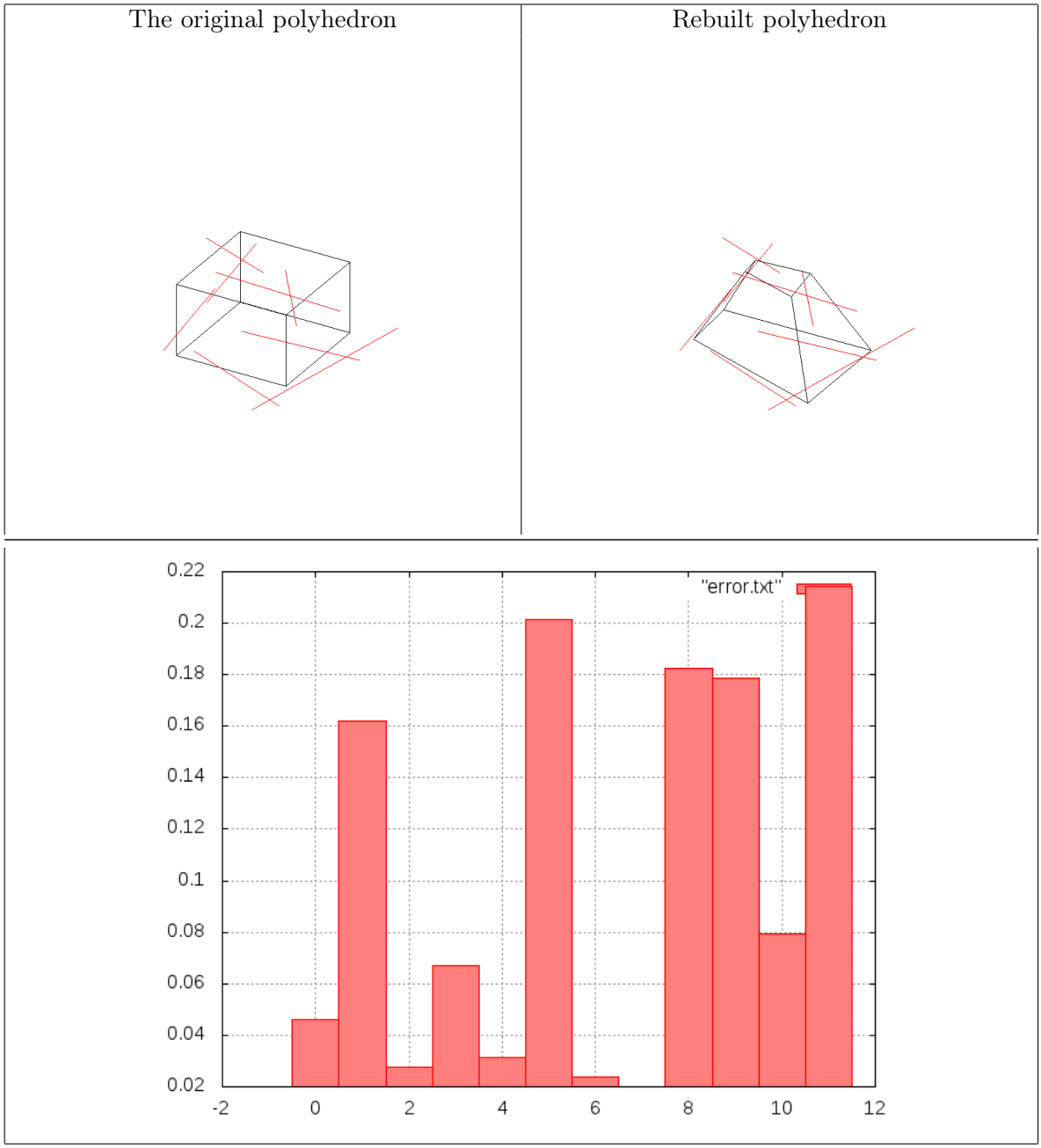
The original polyhedron	Rebuilt polyhedron
	

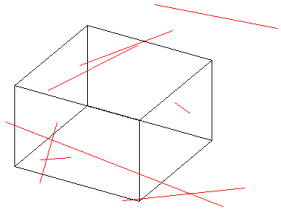
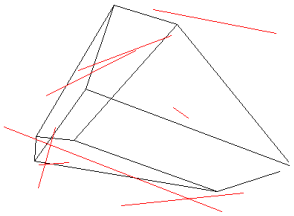


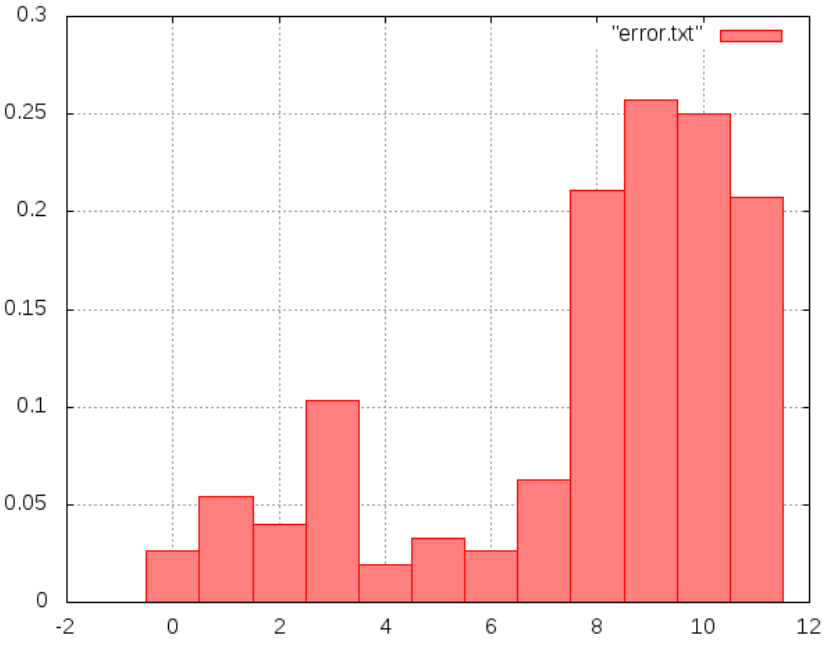
The original polyhedron	Rebuilt polyhedron
	

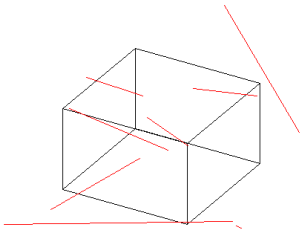
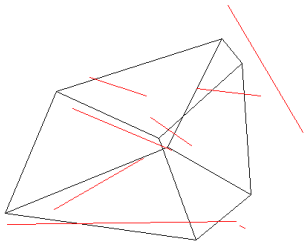


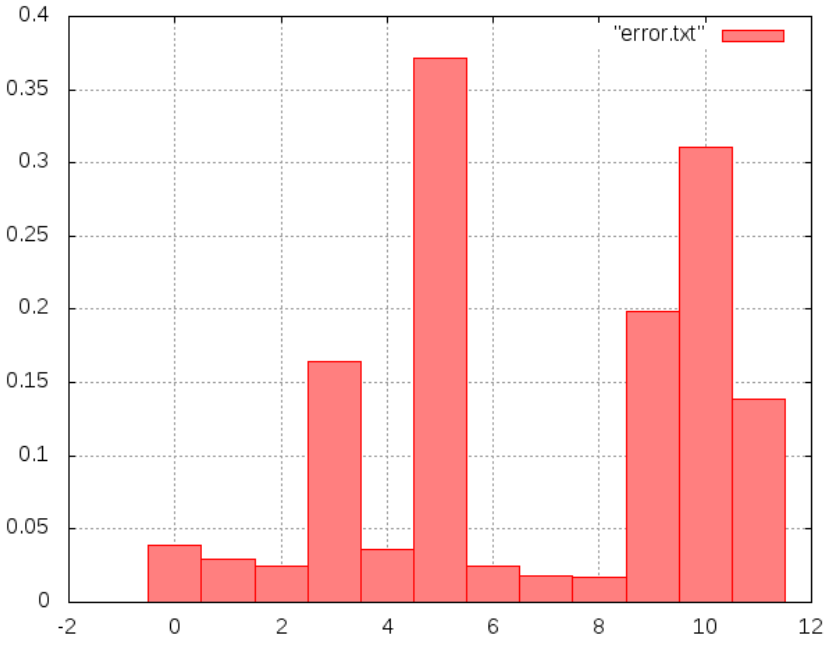
4.1.3. Large deviations $|\Delta x_i| < 0.5, |\Delta y_i| < 0.5, |\Delta z_i| < 0.5$



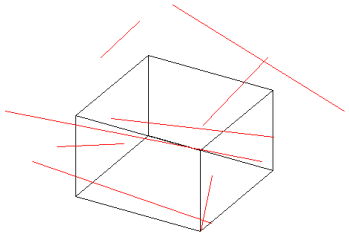
The original polyhedron	Rebuilt polyhedron
	



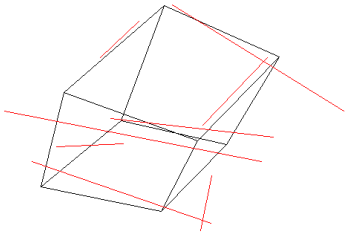
The original polyhedron	Rebuilt polyhedron
	

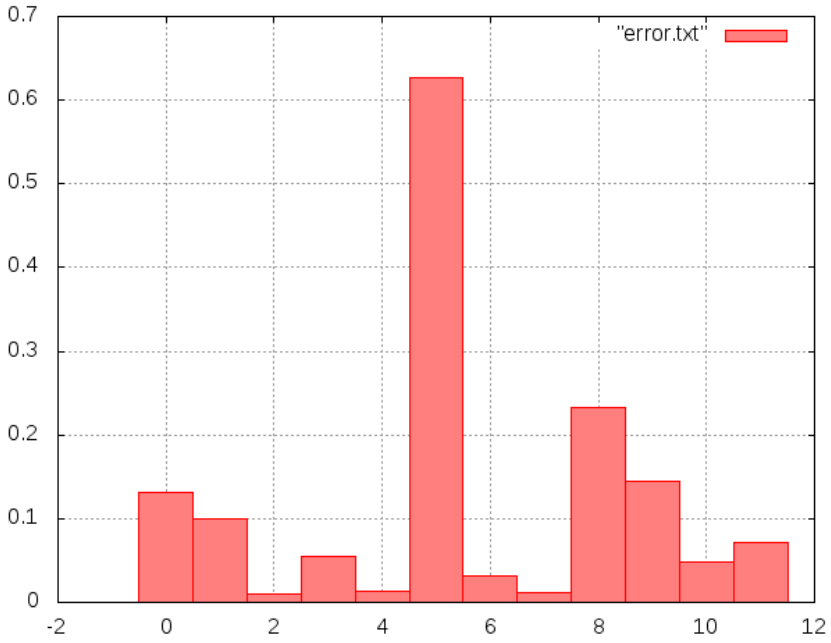


The original polyhedron

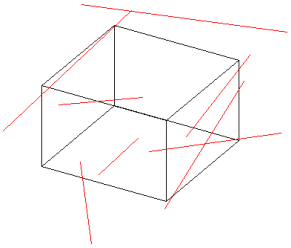


Rebuilt polyhedron

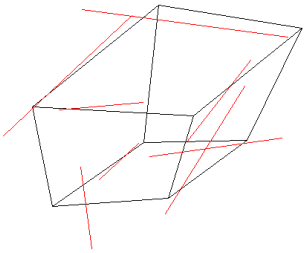


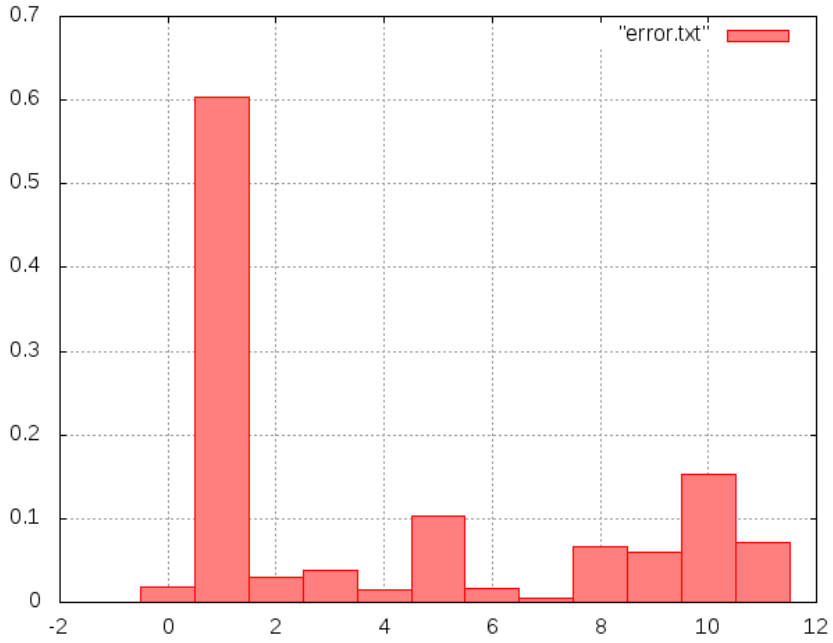


The original polyhedron

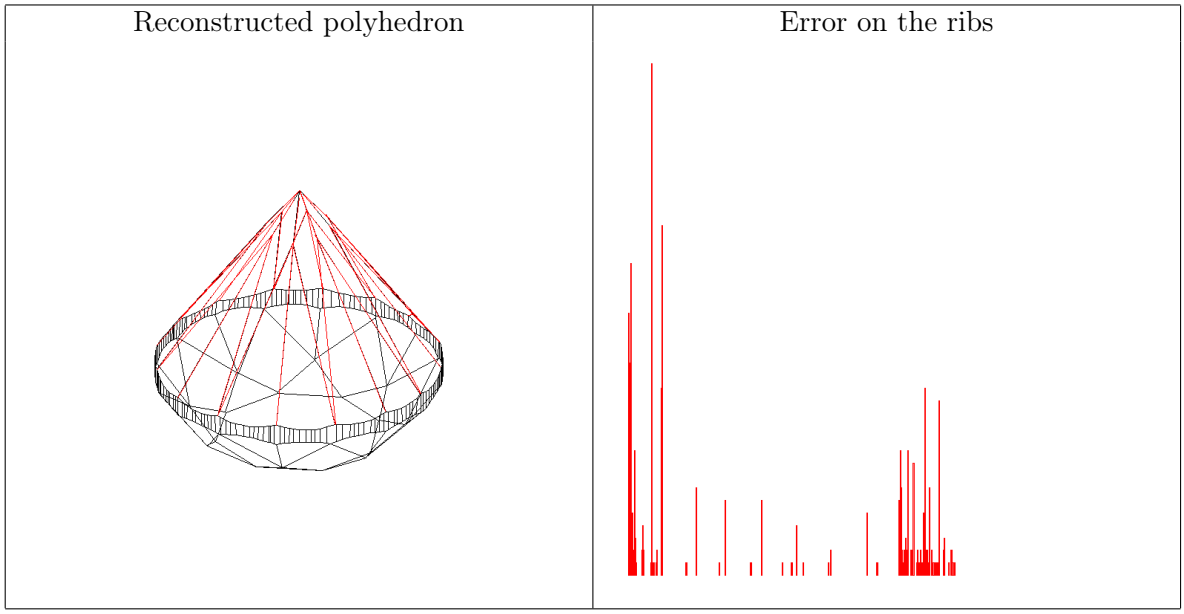


Rebuilt polyhedron



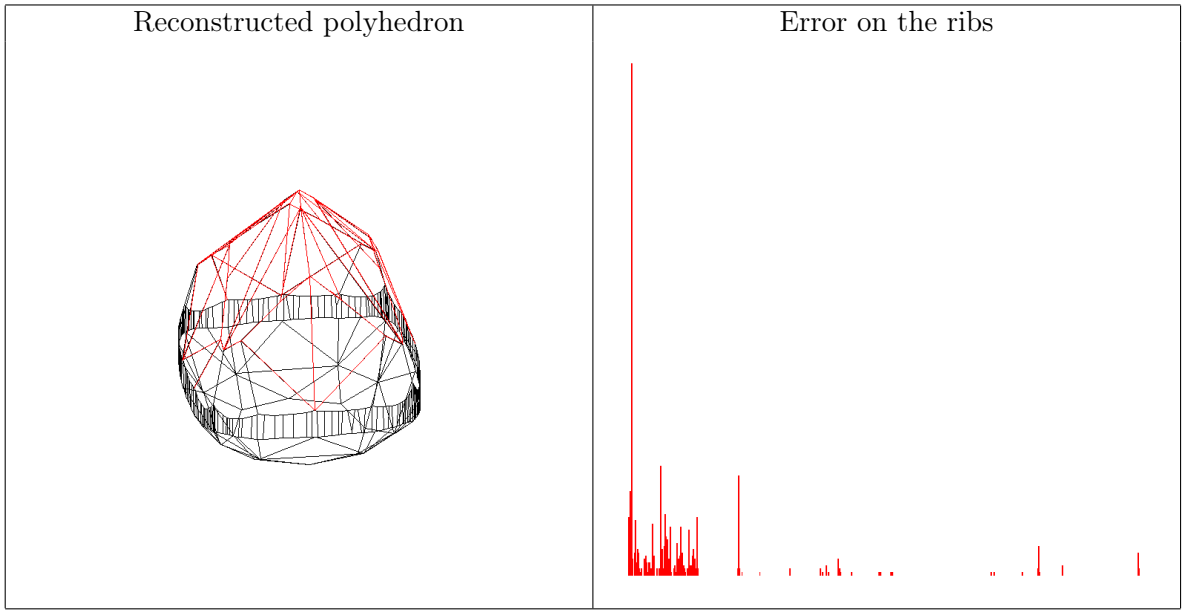


4.2. The first test of the polyhedron.



The value of the functional: $3.9 * 10^{-4}$
Operating time: 405 seconds
number of iterations: 478
number of vertexes: 442
the Number of edges: 663
the Number of faces: 223
Number of target edges: 35
the Number of variables: 4870
Number of non-zero elements in the Jacobian: 9951

4.3. The second test of the polyhedron.



The value of the functional: $7.6 * 10^{-4}$
Operating time: 635 seconds
number of iterations: 526
number of vertexes: 536
the Number of edges: 804
the Number of faces: 270
Number of target edges: 68
the Number of variables: 5904
Number of non-zero elements in the Jacobian: 12066

5. Speed up work on large models.

Unfortunately, the algorithm takes a long time to work on large models. However, from the error histograms, you can see that on more than half of the edges, the distance from the target edges is zero. These are the edges of the lower part of the crystal, for which there are initially no target edges and they are attracted to themselves. As a result, they are static and their movement does not occur, but their movement is calculated. We will now include only edges in the functional that lie in faces that include at least one target edge. This model is somewhat more difficult to program, but it will speed up ten times. So the first crystal approaches in 10 seconds after 37 iterations, and the second-in 20 seconds after 41 iterations.

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