

Лабораторная работа  
по вычислительным методам алгебры на тему:

Вычисление определителя и обратной матрицы с помощью метода  
Гаусса

Выполнил:  
Архангельский И.А.

Проверил:  
Кондратюк А.П.

# Входные и выходные данные.

## Входные данные

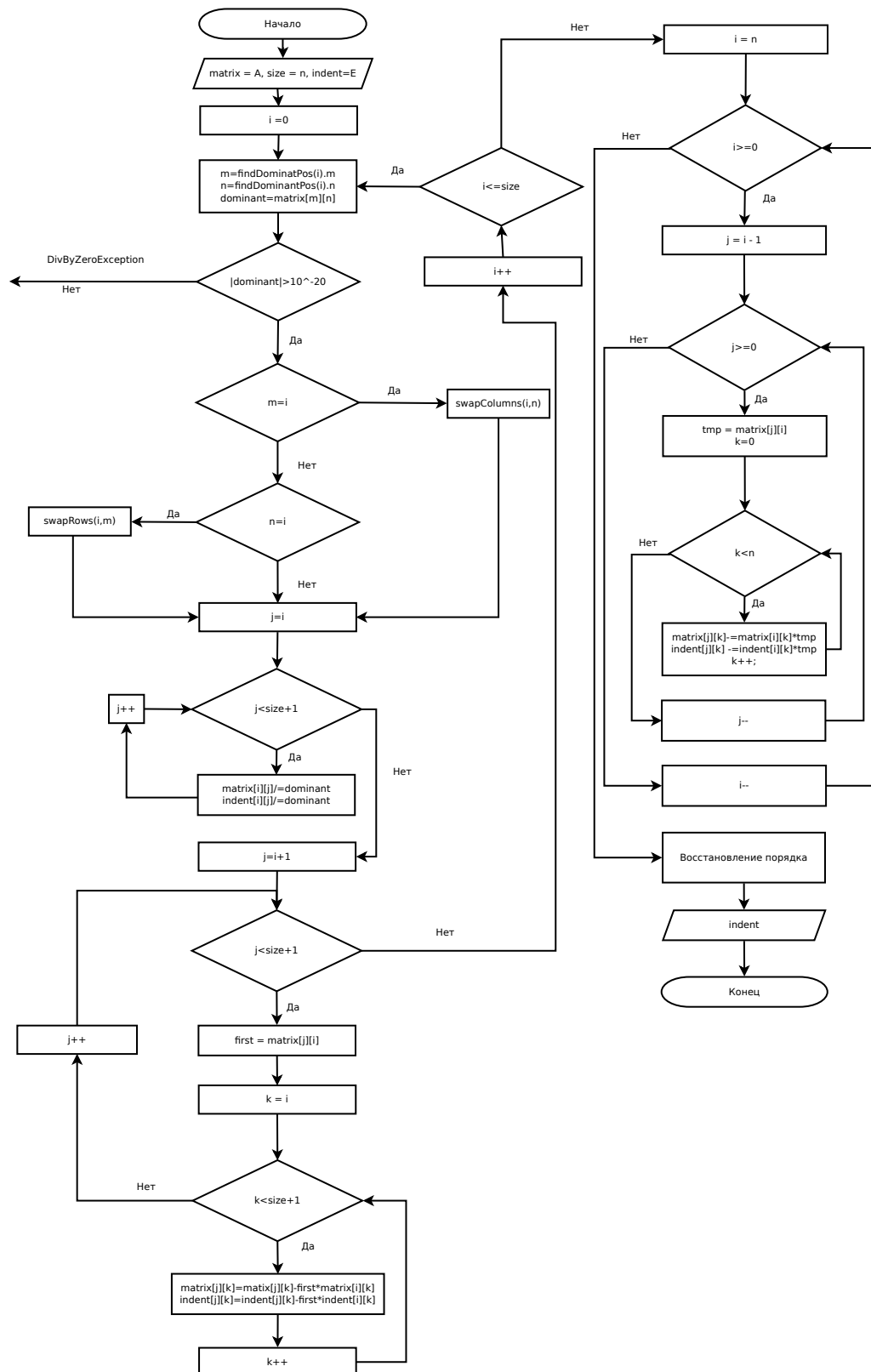
На вход программа принимает текстовый файл в котором первой строкой стоит целое неотрицательное число  $n$ , показывающее размерность матрицы  $A$ . Следующие  $n$  строк содержат матрицу  $A$ .

## Выходные данные

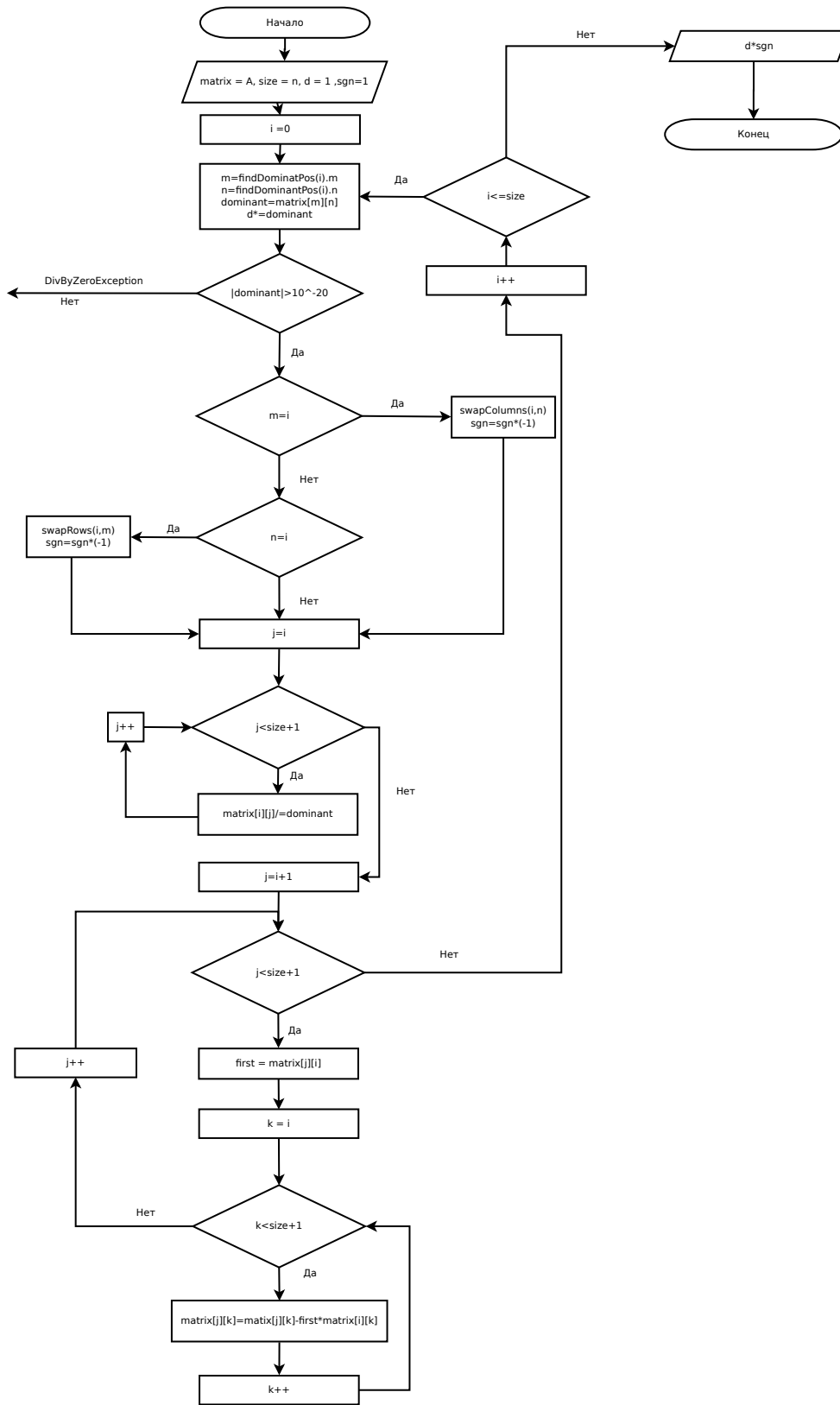
На выход в `stdout` выводится значение определителя, и матрица вида  $\det(A)A^{-1}$ .

# Блок-схема

## Вычисление обратной матрицы



# Вычисление определителя



# Реализация

Листинг 1: gauss.h

```
1  #ifndef GAUSS_H
2  #define GAUSS_H
3
4  #include <cstdlib>
5  #include <stdio.h>
6  #include <math.h>
7
8  using namespace std;
9
10 struct DivByZeroException {};
11 class Gauss
12 {
13 private:
14     double** matrix;
15     unsigned int size;
16     void copy ( double** source, double** &dest );
17     double** getIndent (int sz);
18     int *permutationColumns;
19     int *permutationRows;
20
21     struct Position
22     {
23         unsigned int n;
24         unsigned int m;
25     };
26
27     void permutate (double* &x);
28     Position findDominant (uint k, double** &matr);
29     void swapRows (int x, int y, double** &matr, bool chVector);
30     void swapColumns (int x, int y, double** &matr, bool chVector);
31 public:
32     Gauss (char* filename);
33     int getSize() {return size;}
34     double det ();
35     double** invert ();
36     void printMatrix (double ** matrix, int size);
37     ~Gauss();
38 };
39
40 #endif // GAUSS_H
```

```

1  #include "gauss.h"
2
3  Gauss::Gauss (char *filename)
4  {
5      FILE* fin = fopen (filename, "r");
6      fscanf (fin, "%d\n", &size);
7      matrix = new double* [size];
8      for (uint i=0; i<size; i++)
9      {
10         matrix[i] = new double [size];
11         for (uint j=0; j<size; j++)
12         {
13             fscanf(fin, "%lf", &matrix[i][j]);
14         }
15     }
16     permutationColumns = new int [size];
17     permutationRows = new int [size];
18     for (uint i=0; i<size; i++)
19     {
20         permutationColumns[i] = i;
21         permutationRows[i] = i;
22     }
23     fclose (fin);
24 }
25
26 Gauss::~~Gauss()
27 {
28     for (uint i=0; i<size; i++)
29     {
30         delete matrix[i];
31     }
32     delete matrix;
33     delete permutationColumns;
34 }
35
36
37 void Gauss::copy (double** source, double** &dest)
38 {
39     double** tmp = new double* [this->size];
40     for (uint i=0; i<this->size; i++)
41     {
42         tmp[i] = new double [size];
43         for (uint j=0; j<this->size; j++)
44         {
45             tmp[i][j] = source[i][j];
46         }
47     }
48     dest = tmp;
49 }
50
51 double** Gauss::getIndent (int sz)
52 {
53     double** tmp = new double* [sz];
54     for (int i=0; i<sz; i++)
55     {
56         tmp[i] = new double [sz];
57         for (int j=0; j<sz; j++)
58         {
59             if (i==j)
60             {
61                 tmp[i][j] = 1;
62             }
63             else
64             {
65                 tmp[i][j] = 0;
66             }
67         }
68     }
69 }

```

```

68     }
69     return tmp;
70 }
71
72 double Gauss::det()
73 {
74
75     double** matrcopy = NULL;
76     copy(this->matrix, matrcopy);
77     double d = 1;
78     double sgn = 1;
79
80     for (uint i=0; i<size; i++)
81     {
82
83         Position dominantPos = findDominant(i, matrcopy);
84         double dominant = matrcopy[dominantPos.m][dominantPos.n];
85         if (fabs(dominant)<pow(10,-20)) throw DivByZeroException();
86         if (dominantPos.m==i && dominantPos.n!=i)
87         {
88             swapColumns(i, dominantPos.n, matrcopy, true);
89             sgn*=-1;
90         }
91         if (dominantPos.n==i && dominantPos.m!=i)
92         {
93             swapRows(i, dominantPos.m, matrcopy, true);
94             sgn*=-1;
95         }
96         d *= dominant;
97         for (uint j=i; j<size; j++)
98         {
99             matrcopy[i][j]/=dominant;
100         }
101
102         for (uint j=i+1; j<size; j++)
103         {
104             double first = matrcopy[j][i];
105             for (uint k=i; k<size; k++)
106             {
107                 matrcopy[j][k]=matrcopy[j][k] - first*matrcopy[i][k];
108             }
109         }
110     }
111
112     for (uint i=0; i<size; i++)
113     {
114         delete[] matrcopy[i];
115     }
116     delete[] matrcopy;
117     return d*sgn;
118 }
119 Gauss::Position Gauss::findDominant(uint k, double **&matr)
120 {
121     Position pos;
122     pos.m = k;
123     pos.n = k;
124     for (uint i=k; i<size; i++)
125     {
126         if (fabs(matr[i][k])>fabs(matr[pos.m][pos.n]))
127         {
128             pos.m = i;
129             pos.n = k;
130         }
131         if (fabs(matr[k][i])>fabs(matr[pos.m][pos.n]))
132         {
133             pos.m = k;
134             pos.n = i;
135         }
136     }

```

```

137
138     return pos;
139 }
140
141 void Gauss::swapRows(int x, int y, double **&matr, bool chVector)
142 {
143     for (uint i=0;i<size;i++)
144     {
145         double tmp = matr[x][i];
146         matr [x][i] = matr[y][i];
147         matr [y][i] = tmp;
148     }
149     if (chVector)
150     {
151         int tmp = permutationRows[x];
152         permutationRows [x] = permutationRows[y];
153         permutationRows [y] = tmp;
154     }
155 }
156
157 void Gauss::swapColumns(int x, int y, double **&matr, bool chVector)
158 {
159     for (uint i=0;i<size;i++)
160     {
161         double tmp = matr[i][x];
162         matr [i][x] = matr[i][y];
163         matr [i][y] = tmp;
164     }
165     if (chVector)
166     {
167         int tmp = permutationColumns[x];
168         permutationColumns [x] = permutationColumns[y];
169         permutationColumns [y] = tmp;
170     }
171 }
172 void Gauss::permute(double *&x)
173 {
174     double* res = new double [size];
175     for (uint i=0;i<size;i++)
176     {
177         res[permutationColumns[i]]=x[i];
178     }
179     for (uint i=0;i<size;i++)
180     {
181         x[i]=res[i];
182     }
183     delete res;
184 }
185
186 double** Gauss::invert()
187 {
188     for (int i=0;i<size;i++)
189     {
190         permutationColumns[i]=i;
191         permutationRows[i]=i;
192     }
193     double** matrCopy = NULL;
194     copy(this->matrix, matrCopy);
195     double** indent = getIndent(this->size);
196     for (uint i=0;i<size;i++)
197     {
198         Position dominantPos = findDominant(i, matrCopy);
199         double dominant = matrCopy[dominantPos.m][dominantPos.n];
200         if (fabs(dominant)<pow(10,-20)) throw DivByZeroException ();
201         if (dominantPos.m==i && dominantPos.n!=i)
202         {
203             swapColumns(i, dominantPos.n, matrCopy, true);
204             swapColumns(i, dominantPos.n, indent, false);
205

```



```

206     }
207     if (dominantPos.n==i && dominantPos.m!=i)
208     {
209         swapRows(i, dominantPos.m, matrCopy, true);
210         swapRows(i, dominantPos.m, indent, false);
211     }
212     for (uint j=0; j<size; j++)
213     {
214         matrCopy[i][j]/=dominant;
215         indent[i][j]/=dominant;
216     }
217
218     for (uint j=i+1; j<size; j++)
219     {
220         double first = matrCopy[j][i];
221         for (uint k=0; k<size; k++)
222         {
223             matrCopy[j][k]=matrCopy[j][k] - first*matrCopy[i][k];
224             indent[j][k] = indent[j][k]-first*indent[i][k];
225         }
226     }
227 }
228
229 for (int i=size-1; i>=0; i--)
230 {
231     for (int j=i-1; j>=0; j--)
232     {
233         double tmp = matrCopy[j][i];
234         for (int k=0; k<size; k++)
235         {
236             matrCopy[j][k] = matrCopy[j][k]-matrCopy[i][k]*tmp;
237             indent[j][k] = indent[j][k]-indent[i][k]*tmp;
238         }
239     }
240 }
241 }
242
243 for (int i=0; i<size; i++)
244 {
245     swapColumns(permutationColumns[i], i, indent, false);
246     swapColumns(permutationColumns[i], i, matrCopy, true);
247 }
248
249 for (int i=0; i<size; i++)
250 {
251     for (int j=0; j<size; j++)
252     {
253         if (matrCopy[i][j]==1)
254         {
255             permutationRows[i]=j;
256         }
257     }
258 }
259 for (int i=0; i<size; i++)
260 {
261     swapRows(permutationRows[i], i, indent, true);
262 }
263 for (uint i=0; i<size; i++)
264 {
265     delete[] matrCopy[i];
266 }
267
268 delete[] matrCopy;
269 return indent;
270 }

```

Листинг 3: main.cpp

```

1  #include <iostream>
2  #include "gauss.h"
3  using namespace std;
4
5
6
7  int main(int argc, char *argv[])
8  {
9      for (int i=1; i<argc; i++)
10     {
11         printf ("RUNNING ON TEST: %s\n", argv[i]);
12         Gauss gauss = Gauss(argv[i]);
13         double ** res = NULL;
14         try
15         {
16             res = gauss.invert();
17         }
18         catch (DivByZeroException e)
19         {
20             printf ("ERR::DIVIZION BY ZERO\n");
21             continue;
22         }
23         double det = gauss.det();
24         printf ("det (A) = %8.3lf\n", det);
25         printf ("det(A)*A^-1:\n");
26         for (int i=0; i<gauss.getSize(); i++)
27         {
28             for (int j=0; j<gauss.getSize(); j++) printf ("%8.2lf", det*res[i][j]);
29             printf ("\n");
30         }
31         for (int i=0; i<gauss.getSize(); i++) delete [] res[i];
32         delete [] res;
33     }
34     return 0;
35
36 }
```

# Тестовые данные

<p>test01.in</p> <pre> 5 2 3 0 0 0 4 5 0 0 0 1 7 1 1 1 -1 3 2 3 4 9 8 4 9 16 </pre>	<p>test01.out</p> <pre> RUNNING ON TEST: tests/test01.in det (A) = -4.000 det (A)*A^-1: 10.00 -6.00 -0.00 0.00 -0.00 -8.00 4.00 0.00 -0.00 0.00 144.00 -58.00 -24.00 14.00 -2.00 -138.00 46.00 32.00 -24.00 4.00 40.00 -10.00 -12.00 10.00 -2.00 </pre>
<p>test02.in</p> <pre> 6 1 2 0 0 0 0 3 8 0 0 0 0 -2 3 2 -1 0 0 7 2 3 2 0 0 5 -1 3 5 7 -5 2 3 7 2 2 -1 </pre>	<p>test02.out</p> <pre> RUNNING ON TEST: tests/test02.in det (A) = 42.000 det (A)*A^-1: 168.00 -42.00 -0.00 -0.00 -0.00 0.00 -63.00 21.00 0.00 0.00 0.00 -0.00 -0.00 -6.00 12.00 6.00 -0.00 -0.00 -525.00 135.00 -18.00 12.00 0.00 -0.00 931.00 -203.00 -98.00 -84.00 -14.00 70.00 959.00 -199.00 -148.00 -102.00 -28.00 98.00 </pre>
<p>test03.in</p> <pre> 5 1 2 3 1 5 0 1 0 5 1 2 1 2 3 2 0 3 0 1 3 3 2 1 3 4 </pre>	<p>test03.out</p> <pre> RUNNING ON TEST: tests/test03.in det (A) = 168.000 det (A)*A^-1: -44.00 -56.00 112.00 108.00 -56.00 -42.00 -42.00 42.00 0.00 42.00 -22.00 14.00 98.00 12.00 -70.00 36.00 0.00 0.00 -12.00 0.00 32.00 56.00 -112.00 -48.00 56.00 </pre>
<p>test04.in</p> <pre> 6 -1 5 0 0 0 0 0 0 3 -2 0 0 0 0 2 -5 0 0 -3 6 0 0 0 0 0 0 0 0 4 5 0 0 0 0 3 4 </pre>	<p>test04.out</p> <pre> RUNNING ON TEST: tests/test04.in det (A) = -99.000 det (A)*A^-1: 18.00 -45.00 -0.00 -0.00 -0.00 -0.00 -0.00 -0.00 -33.00 11.00 -0.00 -0.00 -0.00 -0.00 -66.00 55.00 -0.00 -0.00 27.00 -18.00 -0.00 -0.00 -0.00 -0.00 0.00 0.00 0.00 0.00 -396.00 495.00 -0.00 -0.00 -0.00 -0.00 297.00 -396.00 </pre>
<p>test05.in</p> <pre> 4 0 3 2 1 3 0 5 6 2 4 0 5 1 7 9 0 </pre>	<p>test05.out</p> <pre> RUNNING ON TEST: tests/test05.in det (A) = -242.000 det (A)*A^-1: 391.00 29.00 -113.00 -103.00 2.00 28.00 -34.00 -16.00 -45.00 -25.00 39.00 -3.00 -158.00 -34.00 24.00 54.00 </pre>