БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

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.

Минск, 2012

Входные и выходные данные.

Входные данные

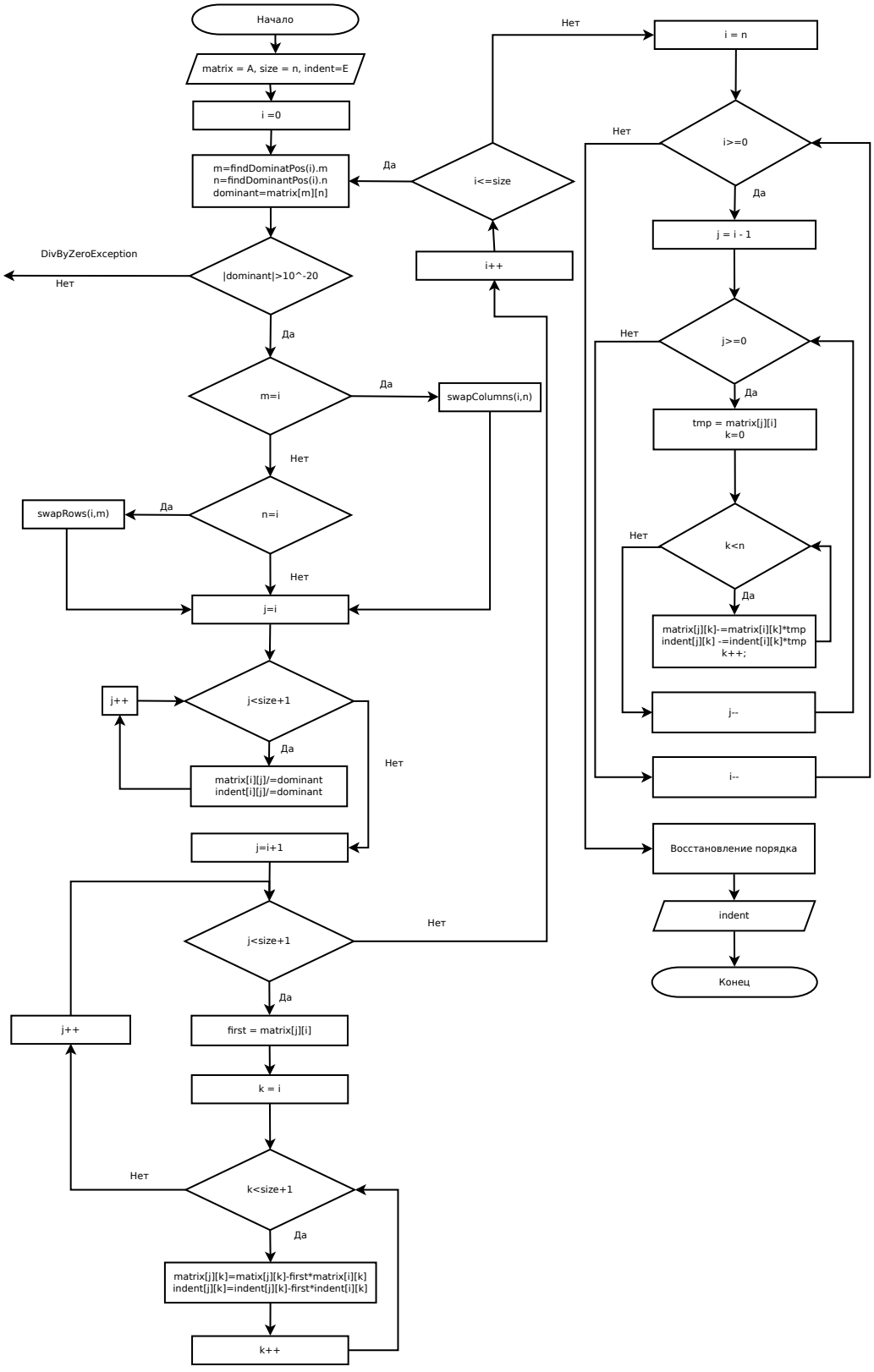
На вход программа принимает текстовый файл в котором первой строкой стоит целое неотрицательное число n, показывающее размерность матрицы A. Следующие n строк содержат матрицу A.

Выходные данные

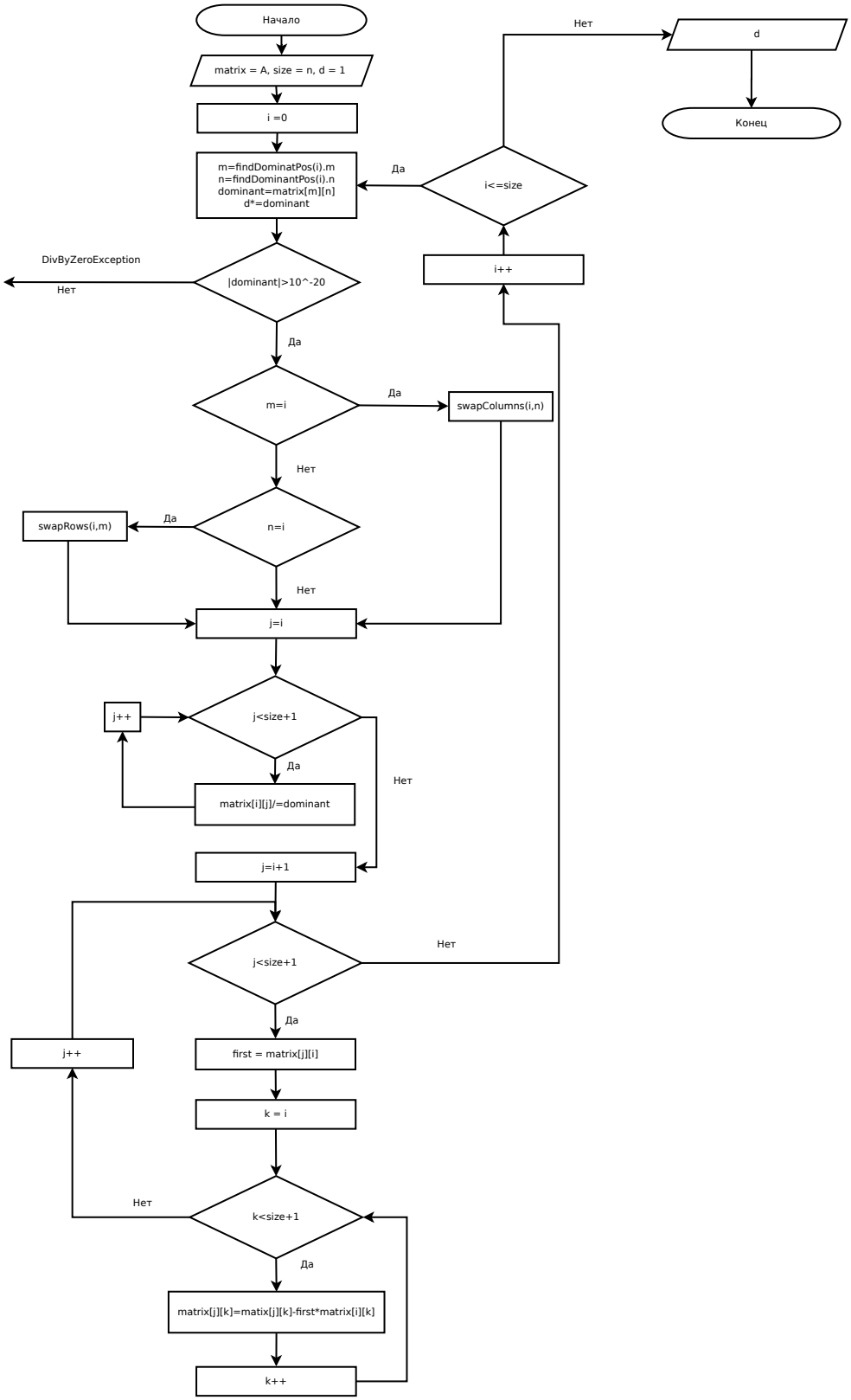
На выход в stdout выводится значение определителя, и матрица вида $det(A)A^{-}1$.

Блок-схема

Вычисление обратной матрицы



Вычисление определителя



Реализация

Листинг 1: gauss.h

```
#ifndef GAUSS_H
#define GAUSS_H

#include <cstdlib>
#include <stdio.h>
#include <math.h>

using namespace std;

struct DivByZeroException {};
class Gauss
{
private:
    double** matrix;
    unsigned int size;
    void copy (  double** source, double** &dest);
    double** getIndent (int sz);
    int *permutationColumns;
    int *permutationRows;

    struct Position
    {
        unsigned int n;
        unsigned int m;
    };

    void permutate (double* &x);
    Position findDominant (uint k, double** &matr);
    void swapRows (int x, int y, double** &matr, bool chVector);
    void swapColumns (int x, int y, double** &matr, bool chVector);
public:
    Gauss (char* filename);
    int getSize() {return size;}
    double det ();
    double** invert ();
    void printMatrix (double ** matrix, int size);
    ~Gauss();
};

#endif // GAUSS_H
```

Листинг 2: gauss.cpp

```
#include "gauss.h"

Gauss::Gauss (char *filename)
{
    FILE* fin = fopen (filename,"r");
    fscanf (fin,"%d\n",&size);
    matrix = new double* [size];
    for (uint i=0;i<size;i++)
    {
        matrix[i]= new double [size];
        for (uint j=0;j<size;j++)
        {
            fscanf(fin,"%lf",&matrix[i][j]);
        }
    }
    permutationColumns = new int [size];
    permutationRows = new int [size];
    for (uint i=0;i<size;i++)
    {
        permutationColumns [i]= i;
        permutationRows[i]=i;
    }
    fclose (fin);
}

Gauss::~Gauss()
{
    for (uint i=0;i<size;i++)
    {
        delete matrix[i];
    }
    delete matrix;
    delete permutationColumns;
}


void  Gauss::copy (  double** source, double** &dest)
{
    double** tmp = new double* [this->size];
    for (uint i=0;i<this->size;i++)
    {
        tmp[i]=new double [size];
        for (uint j=0;j<this->size;j++)
        {
            tmp[i][j]=source[i][j];
        }
    }
    dest = tmp;
}

double** Gauss::getIndent(int sz)
{
    double** tmp = new double* [sz];
    for (int i=0;i<sz;i++)
    {
        tmp[i]=new double [sz];
        for (int j=0;j<sz;j++)
        {
            if (i==j)
            {
                tmp[i][j]=1;
            }
            else
            {
                tmp[i][j]=0;
            }
        }
```

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

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

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

Листинг 3: main.cpp

```
#include <iostream>
#include "gauss.h"
using namespace std;



int main(int argc, char *argv[])
{
    for (int i=1;i<argc;i++)
    {
        printf ("RUNNING ON TEST: %s\n",argv[i]);
        Gauss gauss =  Gauss(argv[i]);
        double ** res = NULL;
        try
        {
            res = gauss.invert();
        }
        catch (DivByZeroException e)
        {
            printf ("ERR::DIVIZION BY ZERO\n");
            continue;
        }
        double det = gauss.det();
        printf ("det (A) = %8.3lf\n",det);
        printf("det(A)*A^-1:\n");
        for (int i=0;i<gauss.getSize();i++)
        {
            for (int j=0;j<gauss.getSize();j++) printf ("%8.2lf",det*res[i][j]);
            printf ("\n");
        }
        for (int i=0;i<gauss.getSize();i++) delete [] res[i];
        delete [] res;
    }
    return 0;

}
```

Тестовые данные

test01.in

```
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
```

test01.out

```
RUNNING ON TEST: tests/test01.in
det (A) =    4.000
det*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
```

test02.in

```
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
```

test02.out

```
RUNNING ON TEST: tests/test02.in
det (A) =   42.000
det*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
```

test03.in

```
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
```

test03.out

```
RUNNING ON TEST: tests/test03.in
det (A) =  168.000
det*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
```

test04.in

```
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
```

test04.out

```
RUNNING ON TEST: tests/test04.in
det (A) =  -99.000
det*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
```

test05.in

```
4
0 3 2 1
3 0 5 6
2 4 0 5
1 7 9 0
```

test05.out

```
RUNNING ON TEST: tests/test05.in
det (A) =  242.000
det*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
```