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

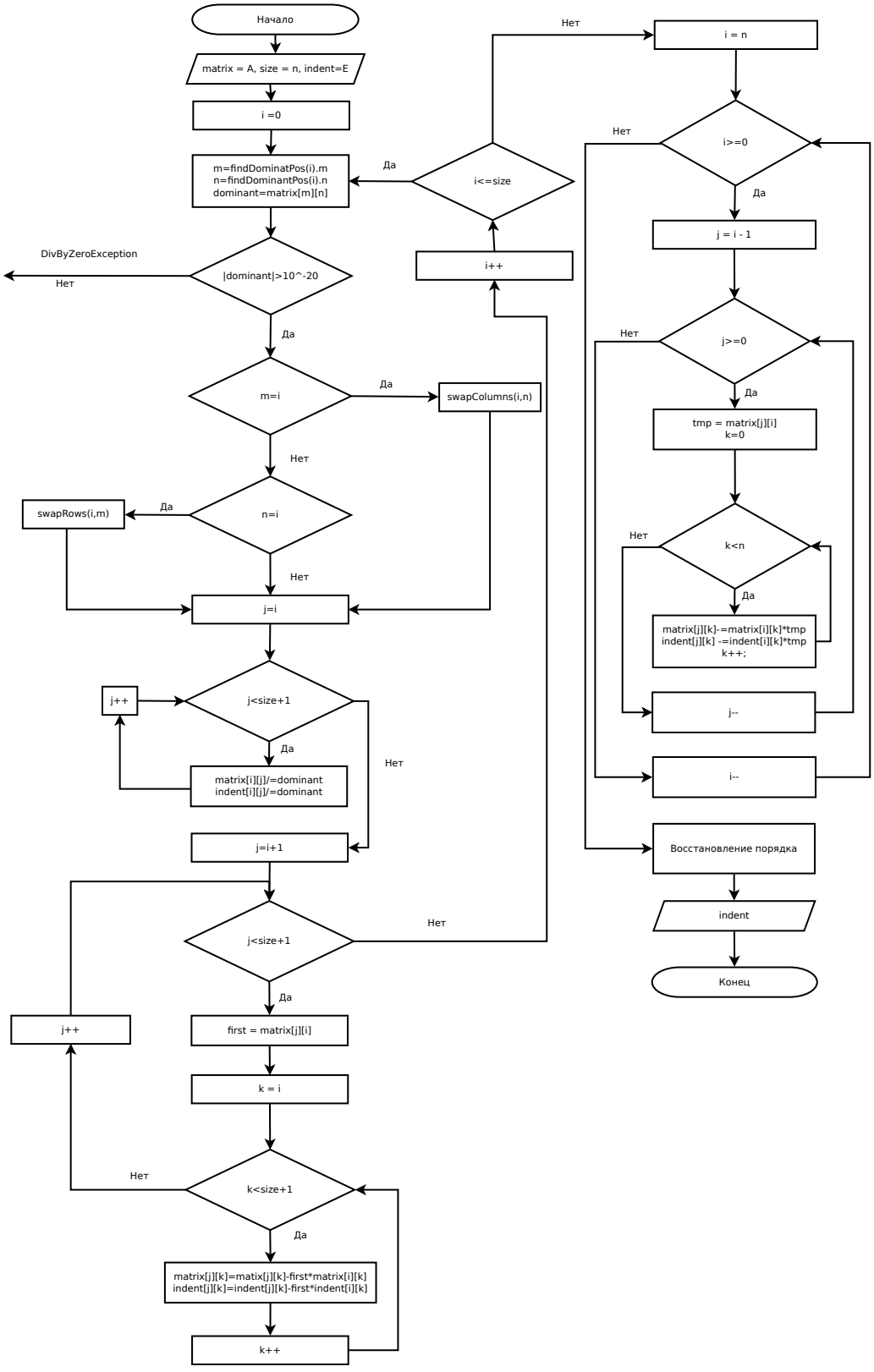
На вход программа принимает текстовый файл в котором первой строкой стоит целое неотрицательное число n, показывающее размерность матрицы A. Следующие n строк содержат матрицу A.

Выходные данные

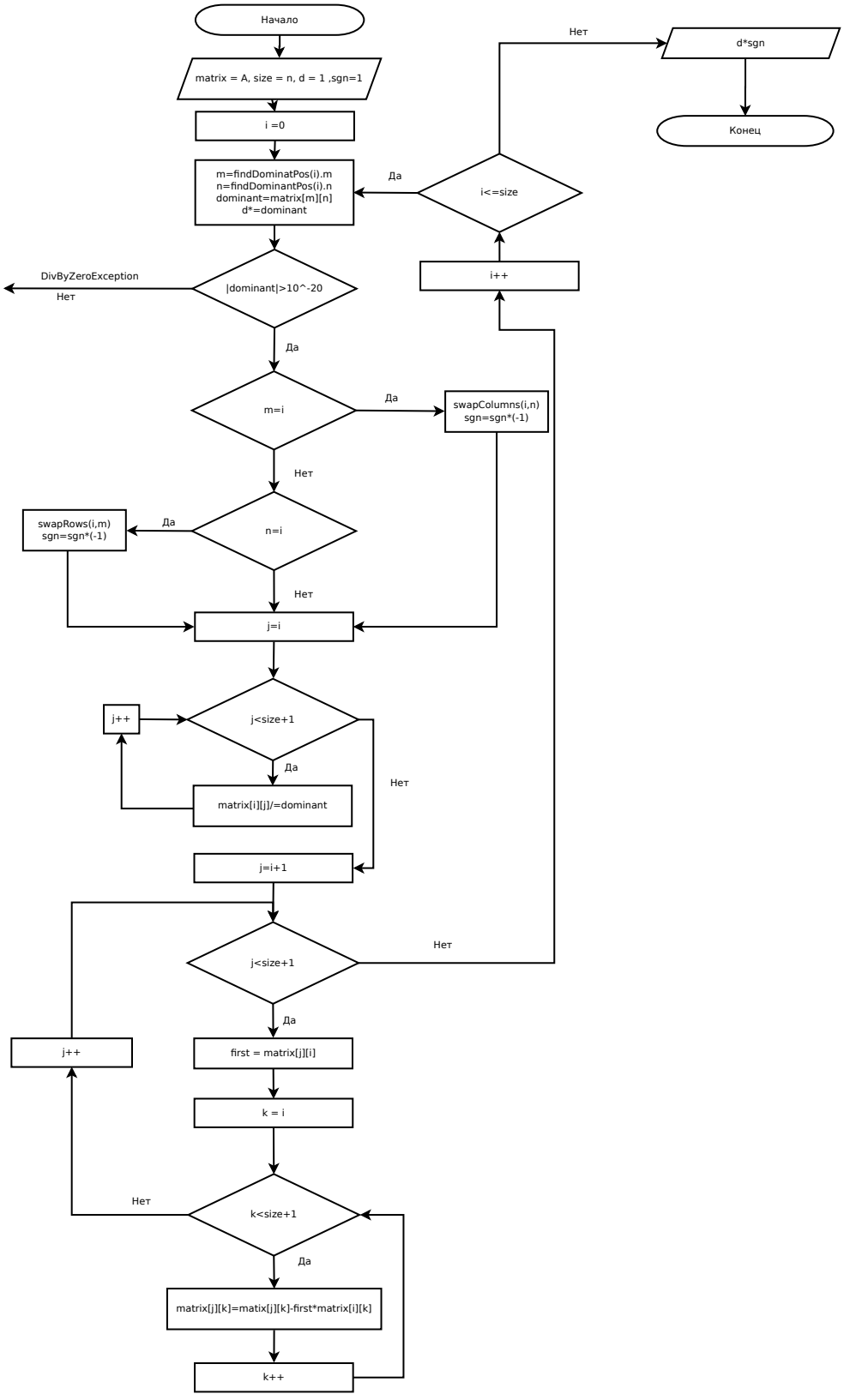
На выход в stdout выводится значение определителя, и матрица вида $det(A)A^{-}1$.

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      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;k<size;k++)
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::permutate(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

```
#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)*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)*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)*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)*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)*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
```