Inveritble Matrix Program

Sava Spasojevic
savaspasojevic@g.ucla.edu

January 2017

Abstract

I wrote this program after having taken my first course in C++ in the fall semester of 2016 at Santa Monica College. During that semester I was heavily preoccupied with two physics course, i.e. Electricity & Magnetism and Fluids, Waves, Thermodynamics, and Optics, as well as Linear Algebra. Because of my heavy course load, my focus was not so explicitly centered on C++, although I performed well in the class. After the semester was over, I self-studied the language and coded this program in effort to test my newly acquired knowledge. I also wrote it so that I may never again have to do the mundane calculations involved in finding the inverse of a matrix.

Invertible Matrix Implementation

The program contains what I call the "default matrix", namely

$$D := \begin{bmatrix} 1 & 1 & 2 \\ 3 & 1 & 0 \\ -2 & 0 & 3 \end{bmatrix}$$

which is an invertible matrix with inverse

$$D^{-1} := \begin{bmatrix} -\frac{3}{2} & \frac{3}{2} & 1 \\ \frac{9}{2} & -\frac{7}{2} & -3 \\ -1 & 1 & 1 \end{bmatrix}$$

I added the class Rationals in order to avoid having decimal entries. This is not a problem, but when calculating matrix inverse for homework, the entries are expected to be in fractional form. There is a piece of code which deals with the problem of singular matrices. Of course, if a matrix is singular it has a linearly dependent column or row, and this is accounted for by considering pivots of value 0. The program makes use of the procedure

$$[A|I] \Longrightarrow [I|A^{-1}]$$

by elementary row operations.

```
#include <iostream>
using namespace std;

class Rationals
{
public:
   Rationals(int num, int den);
   Rationals(int whole_number);
   Rationals();
   friend istream& operator >>(istream& ins, Rationals& rational);
   friend ostream& operator <<(ostream& outs, Rationals& rational);
   friend Rationals operator +(const Rationals& r1, const Rationals& r2);
   friend Rationals operator *(const Rationals& r1, const Rationals& r2);
   friend Rationals operator /(const Rationals& r1, const Rationals& r2);
   friend bool operator !=(const Rationals& r1, const Rationals& r2);
   friend bool operator ==(const Rationals& r1, const Rationals& r2);
private:
   int numerator;
   int denominator;
};

int GrCF(int num, int den);

class matrix {
public:
```

```
26    matrix();
27    matrix(char custom);
28    void output(bool invertible);
29    int get_m();
30    int get_n();
31    void create_pivot(int m, int n);
32    void gauss_jordan(int m, int n);
33 private:
34    void create_indentity();
35    int rows;
36    int cols;
37    Rationals **p;
38    Rationals **I;
39 };
40
41 void new_line();
42
43 void main()
44 {
45    char ans;
46
47    cout << "Inveritble Matrix Program\n";
48    cout << "-------------------------\n";
49
50    do {
51       matrix A;
52       cout << "Would you like to use the default matrix? ";
53       cin >> ans;
54       new_line();
55       if (ans == 'n')
56       {
57          A = matrix('n');
58       }
59       else
60       {
61          A = matrix();
62       }
63       A.output(false);
64       int row_number = 0, col_number = 0;
65       while (row_number < A.get_m()) {
66          A.create_pivot(row_number, col_number);
67          A.gauss_jordan(row_number, col_number);
68          row_number++;
69          col_number++;
70       }
71       A.output(true);
72       cout << "Would you like to try another matrix? ";
73       cin >> ans;
74       new_line();
75    } while (ans != 'n');
76    cout << "Exiting program...\n";
77    system("pause");
78 }
79
80 matrix::matrix()
81 {
82    rows = 3;
83    cols = 3;
84
85    p = new Rationals*[rows];
86
87    for (int i = 0; i < rows; i++)
88       p[i] = new Rationals[cols];
89    //default matrix
90    p[0][0] = Rationals(1);
91    p[0][1] = Rationals(1);
92    p[0][2] = Rationals(2);
93
94    p[1][0] = Rationals(3);
95    p[1][1] = Rationals(1);
96    p[1][2] = Rationals(0);
97
98    p[2][0] = Rationals(-2);
99    p[2][1] = Rationals(0);
100   p[2][2] = Rationals(3);
101
```

```
102    create_indentity();
103 }
104
105 matrix::matrix(char custom)
106 {
107    cout << "Enter the row and column dimensions of the matrix.\n";
108    cout << "Rows: ";
109    cin >> rows;
110    cout << "Cols: ";
111    cin >> cols;
112
113    p = new Rationals*[rows];
114
115    for (int i = 0; i < rows; i++)
116       p[i] = new Rationals[cols];
117
118    int num;
119
120    cout << "Enter " << rows << " rows of " << cols << " integers.\n";
121    for (int i = 0; i < rows; i++)
122       for (int j = 0; j < cols; j++) {
123          cin >> num;
124          p[i][j] = num;
125       }
126
127    create_indentity();
128 }
129
130 void matrix::output(bool invertible)
131 {
132    if (!invertible) {
133       cout << "Echoing typed matrix...\n";
134       for (int i = 0; i < rows; i++) {
135          for (int j = 0; j < cols; j++)
136             cout << p[i][j] << " ";
137          cout << endl;
138       }
139       cout << "Echoing identity matrix...\n";
140       for (int i = 0; i < rows; i++) {
141          for (int j = 0; j < cols; j++)
142             cout << I[i][j] << " ";
143          cout << endl;
144       }
145    }
146    else {
147       cout << "Echoing typed matrix...\n";
148       for (int i = 0; i < rows; i++) {
149          for (int j = 0; j < cols; j++)
150             cout << p[i][j] << " ";
151          cout << endl;
152       }
153       cout << "Echoing inverse matrix...\n";
154       for (int i = 0; i < rows; i++) {
155          for (int j = 0; j < cols; j++)
156             cout << I[i][j] << " ";
157          cout << endl;
158       }
159    }
160
161 }
162
163 void matrix::create_indentity()
164 {
165    I = new Rationals*[rows];
166
167    for (int i = 0; i < rows; i++)
168       I[i] = new Rationals[cols];
169
170    for (int i = 0; i < rows; i++) {
171       for (int j = 0; j < cols; j++) {
172          if (i != j)
173             I[i][j] = 0;
174          else
175             I[i][j] = 1;
176       }
177    }
```

```
}

int matrix::get_m()
{
   return rows;
}

int matrix::get_n()
{
   return cols;
}

void matrix::create_pivot(int m, int n)
{
   if (p[m][n] != Rationals(1) && p[m][n] != Rationals(0)) {
      Rationals temp = p[m][n];
      for (int j = 0; j < cols; j++) {
         p[m][j] = p[m][j] / temp;
         I[m][j] = I[m][j] / temp;
      }
   }
}

void matrix::gauss_jordan(int m, int n)
{
   Rationals pivot = p[m][n];
   if (pivot == 0) {
      cout << "This matrix does not have an inverse. Aborting Program.\n";
      exit(1);
   }
   else {
      int current_row = m + 1, current_col = n;
      Rationals constant_term;
      while (current_row < rows)
      {
         if (p[current_row][current_col] != 0) {
            constant_term = p[current_row][current_col];
            for (int j = 0; j < cols; j++) {
               p[current_row][j] = p[current_row][j] + ((-1) * constant_term)*p[m][j];
               I[current_row][j] = I[current_row][j] + ((-1)*constant_term)*I[m][j];
            }
         }
         current_row++;
      }
      current_row = m - 1;
      while (current_row >= 0)
      {
         if (p[current_row][current_col] != 0) {
            constant_term = p[current_row][current_col];
            for (int j = 0; j < cols; j++) {
               p[current_row][j] = p[current_row][j] + ((-1) * constant_term)*p[m][j];
               I[current_row][j] = I[current_row][j] + ((-1)*constant_term)*I[m][j];
            }
         }
         current_row--;
      }
   }
}

void new_line()
{
   char symbol;
   cin.get(symbol);
   while (symbol != '\n')
      cin.get(symbol);
}

Rationals::Rationals(int num, int den) : numerator(num), denominator(den)
{
   if (denominator < 0 && numerator > 0)
   {
      numerator *= -1;
      denominator *= -1;
   }
   else if (denominator < 0 && numerator < 0)
   {
```

```
254         numerator *= -1;
255         denominator *= -1;
256     }
257 }
258
259 Rationals::Rationals(int whole_number) : numerator(whole_number), denominator(1)
260 {
261
262 }
263
264 Rationals::Rationals() : numerator(0), denominator(1)
265 {
266
267 }
268
269 istream& operator >>(istream& ins, Rationals& rational)
270 {
271     char forward_slash;
272     ins >> rational.numerator >> forward_slash >> rational.denominator;
273     return ins;
274 }
275
276 ostream& operator <<(ostream& outs, Rationals& rational)
277 {
278
279     if (rational.denominator < 0 && rational.numerator > 0)
280     {
281         rational.numerator *= -1;
282         rational.denominator *= -1;
283     }
284     else if (rational.denominator < 0 && rational.numerator < 0)
285     {
286         rational.numerator *= -1;
287         rational.denominator *= -1;
288     }
289
290     if (GrCF(rational.numerator, rational.denominator) != 1)
291     {
292         int GCF = GrCF(rational.numerator, rational.denominator);
293         rational.numerator /= GCF;
294         rational.denominator /= GCF;
295     }
296
297     if ((rational.numerator % rational.denominator) == 0)
298         outs << rational.numerator / rational.denominator;
299     else {
300         outs << rational.numerator << "/" << rational.denominator;
301     }
302     return outs;
303 }
304
305 Rationals operator +(const Rationals& r1, const Rationals& r2)
306 {
307     Rationals temp;
308     Rationals r1_t, r2_t;
309     if (r1.denominator == r2.denominator) {
310         temp.denominator = r1.denominator;
311         temp.numerator = r1.numerator + r2.numerator;
312     }
313     else
314     {
315         r1_t.numerator = r1.numerator*r2.denominator;
316         r1_t.denominator = r1.denominator*r2.denominator;
317
318         r2_t.numerator = r2.numerator*r1.denominator;
319         r2_t.denominator = r1_t.denominator;
320
321         temp.numerator = r1_t.numerator + r2_t.numerator;
322         temp.denominator = r2_t.denominator;
323     }
324     return temp;
325 }
326
327 Rationals operator *(const Rationals& r1, const Rationals& r2)
328 {
329     Rationals temp;
```

```
330
331     temp.numerator = r1.numerator * r2.numerator;
332     temp.denominator = r1.denominator * r2.denominator;
333
334     return temp;
335 }
336
337 Rationals operator /(const Rationals& r1, const Rationals& r2)
338 {
339     Rationals temp;
340
341     temp.numerator = r1.numerator * r2.denominator;
342     temp.denominator = r1.denominator * r2.numerator;
343
344     return temp;
345 }
346
347 bool operator !=(const Rationals& r1, const Rationals& r2)
348 {
349     int cross1, cross2;
350     cross1 = r1.numerator * r2.denominator;
351     cross2 = r1.denominator * r2.numerator;
352     if (cross1 == cross2)
353        return false;
354     else
355        return true;
356 }
357
358 bool operator ==(const Rationals& r1, const Rationals& r2)
359 {
360     int cross1, cross2;
361     cross1 = r1.numerator * r2.denominator;
362     cross2 = r1.denominator * r2.numerator;
363     if (cross1 == cross2)
364        return true;
365     else
366        return false;
367 }
368
369 int GrCF(int num, int den)
370 {
371     int GCF = 0, one = 1;
372     bool num_GCF, den_GCF;
373     for (int i = 2; i < 100000; i++) {
374
375        if (num % i == 0)
376           num_GCF = true;
377        else
378           num_GCF = false;
379
380        if (den % i == 0)
381           den_GCF = true;
382        else
383           den_GCF = false;
384
385        if (num_GCF && den_GCF)
386           GCF = i;
387     }
388
389     if (GCF != 0)
390        return GCF;
391     else
392        return one;
393 }
```