

Initial exercise E0

Charles S. Bos

August, 2011

1 Introduction

In mid-August you will be following the course on *Advanced Programming in Quantitative Economics*. In order to get a bit of a headstart, you'll find attached a first complete program, written in the Ox Doornik (2009) programming language.

During the course, we will study the concepts in detail. In order to streamline the discussions, you are asked to prepare for yourself.

2 Preparation

Section 3 contains the listing of the program. Read this program through, don't use a computer at all at this stage (maybe use a pocket calculator if you really want to).

Ask yourself e.g. the following questions:

1. Where would execution of the program start?
2. What lines are comments, which are code?
3. What is the system in the naming of the variables?
4. After line 171, the value of \mathbf{mC} is

$$\mathbf{mC} = \begin{pmatrix} 10 & -7 & -4 & 28 \\ -7 & 59 & 18 & -145 \\ -4 & 18 & 58 & 56 \end{pmatrix}$$

What would its value be after line 176? And what would be the value of \mathbf{vX} in the end?

5. Matrices are indexed throughout the program. How does this work? Where does the index start?
6. The program prints some final solution. In standard econometrics, what is the problem that is solved? What would you have written on line 5?

7. This same program could have been written in some 40 lines of code (of which roughly half initialisation of `vY` and `mX`). What would be possible advantages of the present, rather extensive program, using 185 lines instead?

Think about these questions before the first class; you are not supposed to answer them all precisely and correctly, that should be easy at the end of the course. You could write for yourself some basic answers, or list your doubt where you don't see the answer. During the course, tick-off the doubts that are solved, or raise the questions with the instructors.

3 Program `e0_elim.ox`

```
1  /*
2  **  e0_elim.ox
3  **
4  **  Purpose:
5  **    ???
6  **
7  **  Date:
8  **    23/7/09
9  **
10 **  Author:
11 **    Charles Bos
12 **
13 #include <oxstd.h>
14
15 /*
16 **  TransRow(const amC, const i, const j)
17 **
18 **  Purpose:
19 **    Clean row j of element in column i
20 **
21 **  Inputs:
22 **    amC    address of existing iK x iM matrix
23 **    i      integer, number of pivot element
24 **    j      integer, number of row to sweep
25 **
26 **  Output:
27 **    amC    address of iK x iM matrix, with a zero at location j,i
28 **           as a result of the sweeping
29 **
30 **  Return value:
31 **    ir     TRUE if all well
32 **
33 TransRow(const amC, const i, const j)
34 {
35     decl dF;
36
37     if (amC[0][i][i] == 0)
38         return FALSE;
39     // Find factor multiplying row i
40     dF= amC[0][j][i] / amC[0][i][i];
41
42     // Subtract dF times row i from row j
```

```

43  amC[0][j][i:]-= dF * amC[0][i][i:];
44
45  return TRUE;
46 }
47
48 /*
49 ** ElimColumn(const amC, const i)
50 **
51 ** Purpose:
52 **   Eliminate the elements of column i, below the pivot
53 **
54 ** Inputs:
55 **   amC   address of existing iK x iM matrix
56 **   i     integer, number of pivot element
57 **
58 ** Output:
59 **   amC   address of iK x iM matrix, with zeros below location i,i
60 **         as a result of the sweeping
61 **
62 ** Return value:
63 **   ir    TRUE if all well
64 */
65 ElimColumn(const amC, const i)
66 {
67   decl ir, j, iK;
68
69   ir= TRUE;
70   iK= rows(amC[0]);
71   for (j= i+1; j < iK; ++j)
72     ir= ir && TransRow(amC, i, j);
73
74   return ir;
75 }
76
77 /*
78 ** ElimGauss(const amC)
79 **
80 ** Purpose:
81 **   Eliminate the lower diagonal of the matrix
82 **
83 ** Inputs:
84 **   amC   address of existing iK x iM matrix
85 **
86 ** Output:
87 **   amC   address of iK x iM matrix, with zero at lower diagonal,
88 **         as a result of the sweeping
89 **
90 ** Return value:
91 **   ir    TRUE if all well
92 */
93 ElimGauss(const amC)
94 {
95   decl iK, ir, i;
96
97   iK= rows(amC[0]);
98   ir= TRUE;
99   for (i= 0; i < iK-1; ++i)

```

```

100     {
101         println ("Starting iteration ", i);
102         ir= ir && ElimColumn(amC, i);
103         println ("resulting in ", amC[0]);
104     }
105
106     return ir;
107 }
108
109 /*
110 ** BackSubst(const mU, const vC)
111 **
112 ** Purpose:
113 **     Substitute back, given a upper diagonal matrix and a right-hand
114 **     side. This solves mU vX = vC for vX.
115 **
116 ** Inputs:
117 **     mU           iK x iK upper diagonal matrix
118 **     vC           iK x 1 right-hand side
119 **
120 ** Return value:
121 **     vX           iK x 1 solution
122 */
123 BackSubst(const mU, const vC)
124 {
125     decl vX, j, k, iN;
126
127     iN= sizerc(vC);
128
129     // Start with solution set at zero
130     // Old solution, using slower loops
131     //   vX= zeros(vC);
132     //   for (k= iN-1; k >= 0; --k)
133     //       {
134     //           vX[k]= vC[k];
135     //           for (j= k+1; j < iN; ++j)
136     //               vX[k]-= mU[k][j] * vX[j];
137     //           vX[k]/= mU[k][k];
138     //       }
139
140     // Start with solution set at zero
141     vX= zeros(vC);
142     vX[iN-1]= vC[iN-1]/mU[iN-1][iN-1];
143     for (k= iN-2; k >= 0; --k)
144         vX[k]= (vC[k] - mU[k][k+1:] * vX[k+1:])/mU[k][k];
145
146     return vX;
147 }
148
149
150 main()
151 {
152     decl mX, vY, mA, vB, mC, ir, iN, vX;
153
154     // Inputs
155     mX= < 1, 1, 3;

```

```

156         1,  -1,  -3;
157         1,  -4,  -1;
158         1,   1,  -1;
159         1,   0,   2;
160         1,   1,  -2;
161         1,   2,   3;
162         1,   1,  -2;
163         1,  -5,   1;
164         1,  -3,  -4>;
165     vY= <4;  -2;  12;  -4;   5;
166         -6;   1;  -6;  19;   1>;
167
168     // Transform inputs
169     mA= mX~mX;
170     vB= mX~vY;
171     mC= mA~vB;
172
173     print ("Initial matrix [A | b]: ", mC);
174
175     // Eliminate the mC matrix, resulting in [ mU | vC ]
176     ir= ElimGauss(&mC);
177     println ("ElimGauss returns ", ir ? "TRUE" : "FALSE",
178             " with mC= ", mC);
179
180     // Find the solution
181     iN= rows(mC);
182     vX= BackSubst(mC[][iN-1], mC[][iN]);
183
184     print ("Solution: ", "%c", {"Elim+Subst"}, vX);
185 }

```

References

Doornik, J. A. (2009). *Ox6: An Object-Oriented Matrix Programming Language*. London: Timberlake Consultants Ltd.