Initial exercise E0

Charles S. Bos

August, 2012

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 `mC` is

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

   What would its value be after line 176? And what would be the value of `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`

```
/*
**  e0_elim.ox
**
**  Purpose:
**    ???
**
**  Date:
**    23/7/09
**
**  Author:
**    Charles Bos
*/
#include <oxstd.h>

/*
**  TransRow(const amC, const i, const j)
**
**  Purpose:
**    Clean row j of element in column i
**
**  Inputs:
**    amC   address of existing iK x iM matrix
**    i     integer, number of pivot element
**    j     integer, number of row to sweep
**
**  Output:
**    amC   address of iK x iM matrix, with a zero at location j,i
**          as a result of the sweeping
**
**  Return value:
**    ir    TRUE if all well
*/
TransRow(const amC, const i, const j)
{
  decl dF;

  if (amC[0][i][i] == 0)
    return FALSE;
  // Find factor multiplying row i
  dF= amC[0][j][i] / amC[0][i][i];

  // Subtract dF times row i from row j
```

```
  amC[0][j][i:]-= dF * amC[0][i][i:];

  return TRUE;
}

/*
**  ElimColumn(const amC, const i)
**
**  Purpose:
**    Eliminate the elements of column i, below the pivot
**
**  Inputs:
**    amC   address of existing iK x iM matrix
**    i     integer, number of pivot element
**
**  Output:
**    amC   address of iK x iM matrix, with zeros below location i,i
**          as a result of the sweeping
**
**  Return value:
**    ir    TRUE if all well
*/
ElimColumn(const amC, const i)
{
  decl ir, j, iK;

  ir= TRUE;
  iK= rows(amC[0]);
  for (j= i+1; j < iK; ++j)
    ir= ir && TransRow(amC, i, j);

  return ir;
}

/*
**  ElimGauss(const amC)
**
**  Purpose:
**    Eliminate the lower diagonal of the matrix
**
**  Inputs:
**    amC   address of existing iK x iM matrix
**
**  Output:
**    amC   address of iK x iM matrix, with zero at lower diagonal,
**          as a result of the sweeping
**
**  Return value:
**    ir    TRUE if all well
*/
ElimGauss(const amC)
{
  decl iK, ir, i;

  iK= rows(amC[0]);
  ir= TRUE;
  for (i= 0; i < iK-1; ++i)
```

```
    {
      println ("Starting iteration ", i);
      ir= ir && ElimColumn(amC, i);
      println ("resulting in ", amC[0]);
    }

  return ir;
}

/*
**  BackSubst(const mU, const vC)
**
**  Purpose:
**    Substitute back, given a upper diagonal matrix and a right-hand
**    side. This solves mU vX = vC for vX.
**
**  Inputs:
**    mU        iK x iK upper diagonal matrix
**    vC        iK x 1 right-hand side
**
**  Return value:
**    vX        iK x 1 solution
*/
BackSubst(const mU, const vC)
{
  decl vX, j, k, iN;

  iN= sizerc(vC);

  // Start with solution set at zero
// Old solution, using slower loops
//   vX= zeros(vC);
//   for (k= iN-1; k >= 0; --k)
//     {
//       vX[k]= vC[k];
//       for (j= k+1; j < iN; ++j)
//         vX[k]-= mU[k][j] * vX[j];
//       vX[k]/= mU[k][k];
//     }

  // Start with solution set at zero
  vX= zeros(vC);
  vX[iN-1]= vC[iN-1]/mU[iN-1][iN-1];
  for (k= iN-2; k >= 0; --k)
    vX[k]= (vC[k] - mU[k][k+1:] * vX[k+1:])/mU[k][k];

  return vX;
}


main()
{
  decl mX, vY, mA, vB, mC, ir, iN, vX;

  // Inputs
  mX= <  1,   1,   3;
```

```
           1,   -1,   -3;
           1,   -4,   -1;
           1,    1,   -1;
           1,    0,    2;
           1,    1,   -2;
           1,    2,    3;
           1,    1,   -2;
           1,   -5,    1;
           1,   -3,   -4>;
  vY= <4;   -2;   12;   -4;    5;
       -6;    1;   -6;   19;    1>;

  // Transform inputs
  mA= mX'mX;
  vB= mX'vY;
  mC= mA~vB;

  print ("Initial matrix [A | b]: ", mC);

  // Eliminate the mC matrix, resulting in [ mU | vC ]
  ir= ElimGauss(&mC);
  println ("ElimGauss returns ", ir ? "TRUE" : "FALSE",
           " with mC= ", mC);

  // Find the solution
  iN= rows(mC);
  vX= BackSubst(mC[][:iN-1], mC[][iN]);

  print ("Solution: ", "%c", {"Elim+Subst"}, vX);
}
```

References

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