Solutions to Computational Economics Exercises using Stata and Mata

ABG Analytics

Solutions to exercises from Chapter 2 of *Applied Computational Economics and Finance* by Mario Miranda and Paul Fackler (MIT press).

Exercise 2.1.

This question asks us to solve the linear equation *Ax = b* using a) L-U decomposition, and then with two iterative methods: b) Gauss-Jacobi iteration and c) Gauss-Seidel iteration.

a. Mata can solve for *x* using L-U decomposition as follows:

```
. mata
: A = (54, 14, -11, 2 \ 14, 50, -4, 29 \ -11, -4, 55, 22 \ 2, 29, 22, 95)
: b = (1, 1, 1, 1)'
: x = lusolve(A, b)
: x
                  1
    +----------------+
  1 |   .0189344097  |
  2 |   .0168050792  |
  3 |   .0233552294  |
  4 |  -.0004108543  |
    +----------------+
: end
```

b. Our Stata program to perform Gauss-Jacobi iteration converged to a row sum norm of less than 0.0001 in 18 iterations.

c. Our Mata program to perform Gauss-Seidel iteration converged to a row sum norm of less than 0.0001 in 10 iterations.

Exercise 2.2.

This problem asks us to solve the linear equation *Ax = b* in three ways, and to record their execution time. Elements of matrix *A* and vector *b* are generated from a *normal(0,1)* distribution. This exercise is designed to help us understand the tradeoffs in computational time if we are solving a problem or its variants once or multiple times. Consider when *A* is fixed and *b* varies across situations. Then computing the inverse of *A* once and computing *x = inv(A)*b* may take less time than applying a solve function for each instance of *b*.

Mata has solver functions for particular situations and we make use of the `lusolve` function instead of Matlab's general matrix solve function (*x=A\a*). We also use Mata's `timer` function instead of tic and toc in Matlab.

In our translation of the problem to the Stata/Mata setting, we:

a) use `lusolve(A,b)` to compute *x* using LU decomposition repeatedly in each of 50 replications,

b) use `lud(A,L,U,p)` to compute once the lower triangular matrix L and the upper triangular matrix U and permutation vector p, but then invert L and U within each replication, and

c) compute the inverse of *A* once and use it to directly compute *x* within each replication.

We make *A* have dimension 300 x 300 rather than 100 x 100 as the book suggests, so it takes a measurable amount of time on a current computer.

```
. mata
: size=300
: A = rnormal(size, size, 0, 1)
: b = rnormal(size, 1, 0, 1)
: reps = 1
: timer_clear()
: timer_on(1)
: for (i=1; i<=reps; i++) {
>         x = lusolve(A, b)
> }
: timer_off(1)
: L=U=p=y=.
: timer_on(2)
: lud(A, L, U, p)
: for (i=1; i<=reps; i++) {
>         Linv=luinv(L)
>         Uinv=luinv(U)
>         P=I(size)[p,.]
>         x = Uinv*Linv*P'*b
> }
: timer_off(2)
: timer_on(3)
: Ainv=luinv(A)
: for (i=1; i<=reps; i++) {
>         x = Ainv*b
> }
: timer_off(3)
: timer()
timer report
  1.       .031 /        1 =      .031
  2.       .063 /        1 =      .063
  3.       .031 /        1 =      .031
: end
```

With only one replication, the timer report suggests methods a) and c) take about the same amount of time (three hundredths of a second each), with method b) taking twice as long. With reps=50, we get the following timer report:

```
timer report
  1.       .578 /        1 =      .578
  2.       2.67 /        1 =     2.668
  3.       .047 /        1 =      .047
```

In this case, method b) takes more than 2.5 seconds compare with less than a second for method a) and less than .05 seconds for method c).

The key lessons here are 1) calls to solvers/inverters take time and should be economized when possible, 2) breaking A into parts L and U that are individually simpler to invert did not make up for the time cost of doubling of the calls needed to inverter routines, and 3) computing matrix products is very fast.

Exercise 2.3.

Here we are asked to prove that Gauss-Jacobi iteration applied to the linear equation $Ax = b$ must converge if A is diagonally dominant. We need to use the Contraction Mapping Theorem and the result that $\|My\| <= \|M\| \|y\|$ for any square matrix M and conformable vector y.

Recall that square matrix A is (strictly) diagonally dominant if the absolute value of the diagonal elements in a row is greater than the sum of the absolute values of the nondiagonal elements of the row. One thing that implies is that A is nonsingular and has an inverse.

The Gauss-Jacobi method sets Q equal to the diagonal matrix formed from the diagonal entries of A in the following iteration rule: $x(k+1) \Leftarrow Q^{-1}b + (I - Q^{-1}A)x(k)$. This rule is a mapping from the vector space containing x onto itself e.g., $x' = F(x)$. We need to show that this mapping is a contraction mapping – once we do that, convergence is assured via the Contraction Mapping Theorem.

Consider any two conformable vectors x and y. We need to show that $d(F(x), F(y)) <= cd(x, y)$, where c is a constant between 0 and 1 and $d()$ is the distance function, i.e., the norm of the difference between x and y.

Noting the common term in $F(x)$ and $F(y)$ of $Q^{-1}b$ drops out when taking the difference, we have $d(F(x),F(y)) = \| (I - Q^{-1}A)x - (I-Q^{-1}A)y \|$
$= \| (I - Q^{-1}A)(x-y) \| <= \| (I - Q^{-1}A) \| \| (x-y) \|$. What remains to be shown then is that $\| (I - Q^{-1}A) \| < 1$. Indeed, this is the convergence criterion described on page 17.

We can readily see this in the case that A is a 2 x 2 matrix. Suppose $A = [a\ b,\ c\ d]$. Then Q^{-1} in the case of Gauss-Jacobi iteration is $[1/a\ 0,\ 0\ 1/d]$. $Q^{-1}A = [1\ b/a,\ c/d\ 1]$, and so $I - Q^{-1}A = [0\ -b/a,\ -c/d,\ 0]$. The Contraction Mapping Theorem will hold if its conditions hold for any matrix norm. The most convenient matrix norm for us to use here is the infinity norm, which for some matrix A is the maximum absolute value over all elements of A. As a direct consequence of diagonal dominance, $|b/a|$ and $|c/d|$ will both be less than one, so the matrix norm of $I - Q^{-1}A = [0\ -b/a,\ -c/d,\ 0]$ is less than 1 and the conditions of the Contraction Mapping Theorem are met and the iterates of the Gauss-Jacobi method must converge.

For a general diagonally dominant square matrix A, with Q^{-1} defined per Gauss-Jacobi iteration, all of the off-diagonal elements of $I - Q^{-1}A$ will be less than one and so $\|I - Q^{-1}A\|$ will be less than 1.

Exercise 2.4.

This question asks us to consider the numerical accuracy of alternative ways of expressing the roots of the quadratic equation $ax^2 + bx + c$. Under what circumstances will each produce inaccurate results? Here we will program the different forms into Stata and consider cases in which the results may be inaccurate.

First we consider a case with known roots: $(x-3)(x-1) = x2 - 4x + 3$. So $a=1$, $b=-4$, and $c=3$. The two roots are 3 and 1. We define and run the following program in Stata for this example, using each of 4 formulas for the two roots provided in the text.

```
capture program drop quadratic
program define quadratic
args a b c
scalar a=`a'
scalar b=`b'
scalar c=`c'
disp "Discriminant = " b^2 - 4*a*c
scalar x11 = (-b + sqrt(b^2 - 4*a*c)) / (2*a)
scalar x12 = (-b - sqrt(b^2 - 4*a*c)) / (2*a)

scalar x21 = -2*c / (b + sqrt(b^2 - 4*a*c))
scalar x22 = -2*c / (b - sqrt(b^2 - 4*a*c))

scalar x31 = (b/(2*a)) * (-1 - sqrt(1 - 4*a*c/b^2))
scalar x32 = (b/(2*a)) * (-1 + sqrt(1 - 4*a*c/b^2))

scalar x41 = -2*c / (b*(1 - sqrt(1 - 4*a*c/b^2)))
scalar x42 = -2*c / (b*(1 + sqrt(1 - 4*a*c/b^2)))

disp %12.0g x11, %12.0g x12
disp %12.0g x21, %12.0g x22
disp %12.0g x31, %12.0g x32
disp %12.0g x41, %12.0g x42
end

. quadratic 1 -4 3

Discriminant = 4
           3            1
           3            1
           3            1
           3            1
```

We see that four formulas return the same two roots in Stata, which are the true roots.

With a small perturbation to give us non-integer roots (*a=1, b=-4, and c=3.1*), we see that the roots are all the same up to 9 decimals.

```
. quadratic 1.1 -4 3

Discriminant = 2.8

 2.578781842  1.057581794

 2.578781842  1.057581794

 2.578781842  1.057581794

 2.578781842  1.057581794
```

Trouble can occur when we subtract two similar large numbers. This can happen in the quadratic formulas if b is a large positive or a large negative number as in the following example.

```
quadratic 1 -100000000 3

Discriminant = 1.000e+16

   100000000  2.98023e-08

   100663296  3.00000e-08

   100000000  3.33067e-08

 90071992.55  3.00000e-08
```

Here, each method gives somewhat different solutions. In the first method, the second root involves adding *-b* and *sqrt(b^2 - 4*a*c)*. With *b* so large, the second term is approximately equal to b and adding

-*b* results in a cancellation of the significant digits (so called "catastrophic cancellation"). We can see that 2.98023 is not the root. The closer approximation is 3.0.

```
. disp a*x12^2 + b*x12 + c

.01976776

. disp a*x22^2 + b*x22 + c

0
```

In the second method, there is no catastrophic cancellation because the denominator involves subtracting a large negative number from a large negative number. Likewise, we can see that the first root in methods 2 and 4 and the second root of method 3 involve catastrophic cancellation and contain substantial error.

An approach to safeguard against numerical inaccuracy would be to condition on the sign of *b* and use the first root using method 1 and the second root using method 2 if *b* is negative. If *b* is positive use the second root using method 1 and the first root using method 2.

See more chapter exercises on our Applied Computational Economics and Finance in Stata main page.

Additional resources

Good discussion on vector and matrix norms related to iterative solutions to linear systems

Recommended Economics Books Financial Economics Books

Excellent description of the Contraction Mapping Theorem

The Penultimate Guide to Precision in Stata

Numerically Stable Method for Solving Quadratic Equations

ABG-Analytics