Gauss-Jordan Inversion of a Matrix

To invert a square matrix, the simplest program, though not likely the fastest nor the most accurate on some machines, is based upon Gauss-Jordan Elimination, a process that resembles Gaussian elimination but goes beyond it to perform the elimination process upon the rows above as well as below the pivotal row. An example of such a program, written in BASIC for an IBM PC, is appended to this note. Its purpose is twofold; first to explain how Gauss-Jordan Elimination works when pivotal exchanges are included; second, to exhibit two features that belong in any elimination-based scheme that solves linear equations or inverts matrices. The two features pertain to nearly singular matrices and to the possibility of extremely large growth of intermediate results (the *Schur Complements*) during elimination.

To benefit from this note, the reader should translate the BASIC program into whatever programming language she favors.

Gauss-Jordan Elimination without frills is performed by lines 680 to 720 and 790 to 950 of the program, which is explained thus: Given an n-by-n matrix A , attach the identity matrix to it to produce a n-by-2n matrix B = [I, A] . We shall operate upon B to convert it into $H = [A^{-1}, I]$ by n successive elementary row operations whose product is tantamount to premultiplication by A^{-1} ; that is, $H = A^{-1}B$. After performing only k of those operations, where $1 \le k \le n$, the array stored in memory could be KB = [K, KA] where K is the product of the first k row operations. However, two technicalities complicate the picture. First, the first k columns of KA turn out to be the same as the first k columns of the identity matrix, not worth storing explicitly. Second, only the first k columns of K are worth storing explicitly since the rest are a permutation, reflecting pivotal exchanges, of the last n-k rows of the identity. Hence the arrays in storage will be X and P , where P records the pivotal exchanges and X contains only the interesting columns of KB , namely the first k columns of KP^{-1} and the last n–k of KA . At the end, $X = KP^{-1}$ is unswapped to obtain $A^{-1} = XP$.

Dealing with the permutations P is the part of the program that novices most often get wrong.

Lines 690 to 720 search for the largest element on or below the diagonal in the k^{th} column to serve as a pivot; its reciprocal will be needed later. But what if this pivot vanishes? Then A must lie no farther than about $2n^3/3$ rounding errors from a singular matrix. These are the same rounding errors as would be committed during the triangular factorization of PA = LU by Gaussian elimination. In fact, the triangular factors L and U actually computed will satisfy $LU = P(A+\Delta A)$ for a perturbation ΔA about which we can know only that $||\Delta A|| < g\, n^2 \varepsilon\, ||A||$ where ε denotes the roundoff level of floating-point arithmetic (computed in statement 590), ||...|| denotes some simple matrix norm, and growth factor $g := \max_j (\max_i |u_{ij}| / \max_i |a_{ij}|)$ is underestimated in statement 740. This g tells by how much a column of some Schur complement has grown bigger than the same column of A . Pivotal exchanges (P) keep g from getting too big except in very rare instances, which will be caught in statements 750 to

780 . Statements like these ought to appear (but hardly ever do) in every conscientiously written program that uses Gaussian elimination without both row- and column- pivoting or some other scheme that keeps $g < 8n$ (say).

When a pivot vanishes, statement 730 replaces it by something tiny, comparable with roundoff in the biggest element of A in the same column, but not so tiny that its reciprocal overflows. This replacement is tantamount to adding another rounding error to the perturbation $\Delta A = P^{-1}LU - A$; but it prevents a (nearly) singular matrix A from being exposed by a vanishing pivot. Is this bad? No. Although a relatively tiny pivot does imply that A is singular or nearly so, the converse is untrue unless n is small. The larger is n , the larger is the probability that a nearly singular A will not be betrayed by any noticeably small pivot. Therefore, the smallness of pivots (or determinants) is generally an unreliable way to detect nearness to singularity. The size of the condition number $\|A\|\,\|A^{-1}\|$ is much better because its logarithm (base 10) indicates roughly how many sig. dec. can be lost to roundoff during the computation by elimination etc. of some approximation X to A^{-1} . Even if $\|A\|\,\|X\|$ is so huge that none of the figures in X can be trusted, yet $\|I - AX\|$ and/or $\|I - XA\|$ will always be far tinier than that, justifying the use of X for preconditioning and similar purposes.

The Program

```
 ...
500 ' Gauss-Jordan Matrix Inversion  X = A^(-1)  in  IBM PC BASIC
510 ' including checks for excessive growth despite row-pivoting,
520 '      and  adjustments for zero pivots to avoid  .../0 .
530 ' DIM A(N,N), X(N,N), P(N)  ...  are assumed.

540  DEFINT  I-N  '  ...  integer variables;  the rest are REAL.
550 '
560 ' First determine levels of roundoff and over/underflow.
570  UFL = 5.9E-39 ' ... = max{ under, 1/over}flow thresholds.
580     G=4 :  G=G/3 :  G=G-1    ' ... = 1/3 + roundoff in 4/3
590  EPS = ABS( ((G+G) - 1) + G ) ' ... = roundoff level.
600  G = 1  ' ...  will record pivot-growth factor
610 '
620 ' Copy  A  to  X  and record each column's biggest element.
630  FOR  J=1 TO N :  P(J)=0
640       FOR  I=1 TO N :  T = A(I,J) :  X(I,J) = T :  T = ABS(T)
650            IF  T > P(J)  THEN  P(J) = T
660            NEXT I :  NEXT J
670 '
680  FOR  K=1 TO N  :' ...  perform elimination upon column  K .
690       Q=0 :  J=K :  ' ...  search for  Kth pivot  ...
700       FOR  I=K TO N
710            T=ABS(X(I,K)) :  IF  T>Q  THEN  Q=T :  J=I
720            NEXT I
730       IF  Q=0  THEN  Q = EPS*P(K) + UFL :  X(K,K)=Q
740       IF  P(K)>0  THEN  Q=Q/P(K) :  IF  Q>G  THEN  G=Q
750       IF  G<=8*K  THEN  GOTO  790
```

```
760    PRINT "Growth factor  g = ";G;"  exceeds ";8*K;" ;  try"
770    PRINT "moving  A's  column ";K;" to col. 1  to reduce  g ."
780          STOP  '  ...  or go back to re-order  A's  columns.

790       P(K)=J  '  ...  record pivotal row exchange,  if any.
800       IF  J=K  THEN  GOTO  830  '  ...  Don't bother to swap.
810           FOR  L=1 TO N :  Q=X(J,L) :  X(J,L)=X(K,L)
820                            X(K,L)=Q :  NEXT L

830       Q = X(K,K) :  X(K,K) = 1
840       FOR  J=1 TO N :  X(K,J) = X(K,J)/Q :  NEXT J
850       FOR  I=1 TO N :  IF  I=K  THEN  GOTO  890
860            Q = X(I,K) :  X(I,K) = 0
870            FOR  J=1 TO N
880                 X(I,J) = X(I,J) - X(K,J)*Q :  NEXT J
890            NEXT I :  NEXT K
900 '
910  FOR  K=N-1 TO 1  STEP -1  ' ...  unswap columns of  X
920       J=P(K) :  IF  J=K  THEN  GOTO  950
930       FOR  I=1 TO N :  Q=X(I,K) :  X(I,K)=X(I,J)
940                        X(I,J)=Q :  NEXT I
950       NEXT K
960  RETURN
```