September, 1998

PHY307F/407F - Computational Physics
Background Material for the Exercise - Solving Systems of Linear Equations

David Harrison

This document discusses techniques to solve systems of linear equations. We write the equations as:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{1}$$

or:

$$A\vec{x} = \vec{b} \tag{2}$$

Usually A and $\vec{b}$ are known and we are solving for $\vec{x}$.

This document is organised as follows:

I. Gaussian Elimination: introduces the major concepts and algorithms used to solve systems of linear equations.

II. LU decomposition: a generalisation of Gaussian elimination.

III. Interchanges: how to deal with possibly catastrophic rounding errors that can lead to totally wrong answers.

IV. Conditioning: how to quantify the degree to which the equations being solved are orthogonal with each other.

V. References: for further study.

VI. Code Listing: a listing of the code for the *Mathematica* package you will be using in the exercise.

I. GAUSSIAN ELIMINATION

Multiply the first of Equation (1) by a_{21}/a_{11} and subtract from the second equation, replacing the second equation with the result of the subtraction. Similarly multiply the first of Equation (2) by a_{31}/a_{11} and subtract from the third equation, replacing the third equation with the result of this subtraction. Continue for the fourth, fifth, ... , and n-th equation. The new set of equations will be:

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}xx_n &= b_1 \\
a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n &= b_2^{(1)} \\
&\vdots \\
a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n &= b_n^{(1)}
\end{aligned}
\tag{3}
$$

where:

$$a_{ij}^{(1)} = a_{ij} - a_{1j}a_{i1}/a_{11}, \quad b_i^{(1)} = b_i - b_1 a_{i1}/a_{11}, \quad i,j = 2,\ldots,n$$

Note that we have eliminated a_{11} from the second and subsequent equations.

The same procedure can eliminate a_{22} from the third and subsequent equations, and so on until we end up with:

$$
\begin{bmatrix}
a_{11} & a_{12} & \cdot & a_{1n} \\
 & a_{22}^{(1)} & \cdot & a_{2n}^{(1)} \\
 & & \cdot & \cdot \\
 & & & a_{nn}^{(n-1)}
\end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ x_n \end{bmatrix}
=
\begin{bmatrix} b_1 \\ b_2^{(1)} \\ \cdot \\ b_n^{(n-1)} \end{bmatrix}
\tag{4}
$$

where:

$$a_{ij}^{(k)} = a_{ij} - a_{kj}a_{ik}/a_{kk}, \quad b_i^{(k)} = b_i - b_k a_{ik}/a_{kk}, \quad i,j = k+1,\ldots,n$$

This procedure is called *forward reduction*. The upper triangular matrix in Equation (4) will be given the symbol U in the next section.

Note that the last equation of Equation (4) has only one unknown, x_n, which can then be calculated. The next to last equation has two unknowns, x_n and x_{n-1}, but x_n has just been found, so we can solve for x_{n-1}. Thus, through repeated *back substitution* we end solving for all the values of $\vec{x}$.

These two parts, *forward reduction* and *back substitution* together form the *Gaussian elimination* method of solving the system of equations.

It will be useful later to write down these two steps in "pseudo code", which is a sort of shorthand for the way the method would be coded in a procedural language such as C or FORTRAN.

Forward Reduction

```
For k = 1,...,n-1
     For i = k+1,...,n
          l(i,k) = a(i,k)/a(k,k)
          For j = k+1,...,n
```

```
          a(i,j) = a(i,j) - l(i,k)a(k,j)
     b(i) = b(i) - l(i,k)b(k)
```

Back Substitution

```
For k = n,n-1,...,1
     x(k) = b(k)
     For j = k+1,...,n
          x(k) = x(k) - a(k,j)x(j)
     x(k) = x(k) / a(k,k)
```

If you read the above carefully, you will see that we are overwriting the *a* matrix and the *b* vector as we proceed.

II. LU DECOMPOSITION

It is fairly trivial to show that the original matrix *A* of Equation (1) can be written as:

$$A = LU \tag{5}$$

where *U* is the upper triangular matrix defined by Equation (4) and *L* is a unit lower triangular matrix whose diagonal elements are all 1 and whose subdiagonal elements are all the *l(i,k)* elements defined in the pseudo code for forward reduction.

Any good computer mathematics library will include routines to carry out this *LU decomposition*, carefully optimised for speed and accuracy.

We now do some simple matrix algebra:

$$\begin{gathered} A\vec{x} = \vec{b} \\ LU\vec{x} = \vec{b} \\ L^{-1}LU\vec{x} = L^{-1}\vec{b} \\ U\vec{x} = L^{-1}\vec{b} \end{gathered}$$

We shall set the last of the equations equal to a new vector $\vec{z}$. Then:

$$\begin{gathered} L^{-1}\vec{b} = \vec{z} \\ L\vec{z} = \vec{b} \\ U\vec{x} = \vec{z} \end{gathered}$$

Thus, we can solve the original system of equation in three steps:

1. Factor $A = LU$.

2. Solve $L\vec{z} = \vec{b}$ for $\vec{z}$.

3. Solve $U\vec{x} = \vec{z}$ for $\vec{x}$.

This method doesn't offer large advantages over straight Gaussian elimination. It eliminates the need to form the b_i^k terms needed in Gaussian elimination, but requires us to first solve for $\vec{z}$ and then for $\vec{x}$. The fact that good routines to do the LU factorisation exist in all good math libraries can be useful. In addition, for ill-conditioned systems of equations LU decomposition can be more stable. Finally, LU decomposition forms the basis for some computational variations on the elimination process.

LU decomposition is also useful for calculating the determinant since:

$$Det[\ A\ ] = Det[\ LU\ ] = Det[\ L\ ] * Det[\ U\ ]$$

and the determinant of L is one. Thus the determinant of A equals the determinant of U; this calculation is trivial:

$$Det[U] = u_{11} u_{22} \cdots u_{nn}$$

III. INTERCHANGES

When we solve a system of linear equations using either Gaussian elimination or LU decomposition, we must calculate `l(i,k) = a(i,k)/a(k,k)` terms. Of course, this is possible only if `a(k,k)` is non-zero. If we find an `a(k,k)` that is zero we must search for an m-th equation ($m>k$) for which $a(m,m)$ is not zero. Then we interchange equations k and m, and continue with forward reduction. If no such equation m exists the system of equations is *singular* and can not be solved.

Interchanges can be necessary even if none of the diagonal terms of the A matrix are zero. We shall illustrate with a very simple set of equations:

$$\begin{bmatrix} -10^{-5} & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6}$$

The exact solution is:

$$x_1 = -0.4999975 \cdots, \quad x_2 = 0.999995 \cdots$$

Now imagine that we have a *decimal* computer that stores floating point numbers as (a,b,c,d,e) where the value being stored is:

$$0.abcd \times 10^e$$

The multiplier is:

$$\texttt{l(2,1)} \equiv l_{21} = -0.2\times10^{1}/0.1\times10^{-4} = -0.2\times10^{6}$$

The new a_{22} is:

$$a_{22}^{(1)} = 0.1\times10^{1}-(-0.2\times10^{6})(0.1\times10^{1}) = 0.1\times10^{1}+0.2\times10^{6} = 0.2\times10^{6}$$

Note that small rounding error in the above calculation.

The new b_2 is:

$$b_2^{(1)} = -(0.2\times10^{6})(0.1\times10^{1)} = 0.2\times10^{6}$$

This calculation is exact.

Now we calculate the answer:

$$x_2 = b_2^{(1)}/a_{22}^{(1)} = 0.2\times10^{6}/0.2\times10^{6} = 0.1\times10^{1} = 1$$
$$x_1=(b_1-a_{12}x_2)/a_{11} = (0.1\times10^{1}-0.1\times10^{1})/-0.1\times10^{-4} = 0$$

The value for x_2 is reasonable, but the value for x_1 is ridiculous.

Why did this happen? The problem is that the multiplier l_{21} is too large, which led the the rounding error in the calculation of $a_{22}^{(1)}$. This in turn arose because a_{11} is small compared to a_{21}. In fact, you can verify that solving:

$$\begin{bmatrix} 2 & 1 \\ -10^{-5} & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

on this same hypothetical four digit computer gives:

$$x_2 = 0.1, \quad x_1 = -0.5$$

which agrees very well with the exact solution.

Thus, in the forward reduction step, we must implement a *partial pivoting* algorithm so that at each step of the reduction the value of a_{kk} that is being reduced is numerically larger than any of the a_{mk} (m>k). The row *m* with the largest numerical value of a_{mk} is interchanged with row *k*. Also elements b_k and b_m are interchanged in $\vec{b}$.

There is yet one more subtlety. Consider

$$\begin{bmatrix} 10 & -10^{6} \\ 2 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -10^{6} \\ 0 \end{bmatrix} \tag{7}$$

According to our partial pivoting strategy, no interchanges are called for. However Equation (7) is identical to Equation (6) except the first equation has been multiplied by -10^6. Thus our hypothetical four digit computer will give exactly the same ridiculous solution to Equation (7) as to Equation (6).

This difficulty is called *equilibration* or *balancing* of the matrix, and arises because the matrix in Equation (7) is no properly scaled. Unfortunately, there is no known general procedure to properly scale a set of equations, although often careful inspection will indicate that scaling is necessary.

Although the make-believe computer we have been using is not very realistic, the same catastrophic rounding errors can arise with *any* computer since all such machines use a finite number of digits to store floating point numbers.

IV. CONDITIONING

Imagine a system of two equations in two unknowns:

$$a_{11}x_1 + a_{12}x_2 = b_1$$
$$a_{21}x_1 + a_{22}x_2 = b_2$$

For each of these equations you will make plots of x_2 versus x_1. The solution of the set of equations is the point where these lines intersect.

If the two lines are parallel, then the equations can not be solved.

If the two lines are nearly but not quite parallel, we say they are *ill conditioned*. A good measure of conditioning is:

$$V = |\det(A)| / (\alpha_1 \alpha_2 \cdots \alpha_n) \qquad (8)$$

where:

$$\alpha_i \equiv \sqrt{a_{i1}^2 + a_{i2}^2 + \cdots + a_{in}^2}$$

and $a_{\text{row column}}$ is an element of the *A* matrix.

If V = 1 the equations are perfectly conditioned; if V = 0 the *A* matrix is singular and the equations can not be solved. *V* is the volume of the *n*-dimensional unit parallelepiped circumscribed by the lines defined the the rows of the *A* matrix.

V. REFERENCES

- Gene H. Golub and James M. Ortega, **Scientific Computing and Differential Equations** (Academic Press, 1992), § 4.2 - 4.4.
- William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, **Numerical Recipes: The Art of Scientific Computing or Numerical Recipes in C: The Art of Scientific Computing** (Cambridge Univ. Press), § 2.0 - 2.3.

VI. CODE LISTING

In this section we list the code for the *Mathematica* package you will be using in the Exercise. Not only is it usually a good idea to open up the "black box" of a computer package to get an idea of how it works, but for beginners looking at code written by more expert programmers is often valuable in learning how to write good programs yourself.

```
(* @(#)Exer.m     1.9 U of T Physics 09/12/96 *)

(*
 * Modified by David Harrison.  See LinearAlgebra/GaussianElimination
 * as shipped on the Mathematica tapes for the history of the original
 * version.  This version is identical except for the addition of
 * SimpleLUFactor, and a change to the internal lufactor[] to
 * disable pivoting.
 *)

(* :Copyright: Copyright 1990,  Wolfram Research, Inc.
      Permission is hereby granted to modify and/or make copies of
      this file for any purpose other than direct profit, or as part
      of a commercial product, provided this copyright notice is left
      intact.  Sale, other than for the cost of media, is prohibited.

      Permission is hereby granted to reproduce part or all of
      this file, provided that the source is acknowledged.
*)

BeginPackage["PHY307F‘Exer‘",
      "EDA‘Master‘"     (* For Exercise 3 *)
]

Unprotect[LUFactor, LUSolve, LU, SimpleLUFactor];

SimpleLUFactor::usage =
      "SimpleLUFactor[mat] gives the LU decomposition just as LUFactor[mat],
      except that partial pivoting is disabled.  SimpleLUFactor[mat,
      WorkingPrecision -> n] limits the working precision to n digits."

Options[SimpleLUFactor] = {WorkingPrecision -> $MachinePrecision}

LUFactor::usage =
```

```
      "LUFactor[mat] gives the LU decomposition along with a pivot list of the
      matrix mat.  The calculation is done using Gaussian elimination with
      partial pivoting.  The result is returned as a data object with a head of
      LU.  LUFactor[ ] may be used in the context of ComputerArithmetic.m or
      IntervalArithmetic.m.  LUFactor[mat, WorkingPrecision -> n] limits the
      working precision to n digits."

Options[LUFactor] = {WorkingPrecision -> $MachinePrecision}

LUSolve::usage =
      "LUSolve[lu, b] solves the linear system represented by lu (the matrix) and
      right-hand side b.  The \"matrix\" lu must be the data object with a head of
      LU resulting from LUFactor[mat] and b must be an ordinary list of the
      appropriate size.  LUSolve[ ] may be used in the context of
      ComputerArithmetic.m or IntervalArithmetic.m.  LUSolve[lu, b,
      WorkingPrecision -> n] limits the working precision to n digits."

Options[LUSolve] = {WorkingPrecision -> $MachinePrecision}

LU::usage =
      "LU[a, pivots] is the data object returned by LUFactor[ ] and is to be given
      as the first argument to LUSolve[ ].  The matrix a represents the lower and
      upper triangular matrices in the LU decomposition, and pivots is a record
      of the pivots used in the decomposition."


Begin["Private‘"]

SimpleLUFactor[aa_?MatrixQ,opts___] :=
    Module[{ans, prec = WorkingPrecision /. {opts} /. Options[SimpleLUFactor] },
            ans /; Head[ans = lufactor[aa,pivot = False,prec]] == LU] /;
            (Length[aa] == Length[aa[[1]]])

LUFactor[aa_?MatrixQ,opts___] :=
    Module[{ans, prec = WorkingPrecision /. {opts} /. Options[SimpleLUFactor] },
            ans /; Head[ans = lufactor[aa,pivot = True,prec]] == LU] /;
            (Length[aa] == Length[aa[[1]]])

LUSolve[ap_LU, bb_List,opts___] :=
    Module[{ans, prec = WorkingPrecision /. {opts} /. Options[SimpleLUFactor]},
      ans /; VectorQ[ans = lusolve[ap, bb, prec]]] /;
```

```
      (Length[ap[[1]]] == Length[bb])

divide[a_, b_] :=
    If[FreeQ[{a, b}, NumericalMath‘ComputerArithmetic‘ComputerNumber],
      a/b, NumericalMath‘ComputerArithmetic‘IdealDivide[a, b]];

lufactor[aa_, dopivot_,prec_] :=
    Module[{a = aa, pivot, ii, iip, i, ip, j, k, mpiv, m, n=Length[aa], tmp},
      pivot = Table[i, {i, n}];
      For[ii=1, ii<=n, ii++,  (* for each row do ... *)
          (* find a pivot *)
          mpiv = SetPrecision[ Abs[a[[pivot[[ii]], ii]]],prec];
          k = ii;
          For[i=ii+1, i<=n, i++,
            tmp = SetPrecision[ Abs[a[[pivot[[i]], ii]]],prec];
            If[ dopivot == True,
                  If[tmp > mpiv, mpiv = tmp; k = i]
            ];
          ];
          tmp = pivot[[ii]];
          pivot[[ii]] = iip = pivot[[k]];
          pivot[[k]] = tmp;
          mpiv = SetPrecision[ a[[iip, ii]],prec];
          (*
           * calculate and store the multipliers and reduce the
           * other elements.
           *)
          For[i=ii+1, i<=n, i++,
            ip = pivot[[i]];
            m = a[[ip, ii]] = SetPrecision[divide[a[[ip, ii]], mpiv],prec];
            For[j=ii+1, j<=n, j++, a[[ip, j]] -= SetPrecision[m*a[[iip, j]],prec]]
            ];
          ];
      LU[a, pivot]
      ]

lusolve[ap_, bb_, prec_] :=
    Module[{a, b = bb, pivot, ii, i, m, n, tmp},
      a = SetPrecision[ap[[1]],prec];
      pivot = SetPrecision[ap[[2]],prec];
      n = Length[a];
```

```
      (* forward elimination *)
      For[ii=1, ii<=n, ii++,  (* for each row do ... *)
          For[i=ii+1, i<=n, i++,
            b[[pivot[[i]]]] -= (SetPrecision[a[[pivot[[i]], ii]],prec]
                  * SetPrecision[b[[pivot[[ii]]]],prec])
            ];
          ];
      (* back substitution *)
      Clear[i];
      For[ii=n, ii>0, ii--,
          tmp = Sum[( SetPrecision[a[[pivot[[ii]], i]],prec] *
            SetPrecision[b[[pivot[[i]]]],prec]), {i, ii+1, n}];
          tmp = SetPrecision[b[[pivot[[ii]]]],prec] - SetPrecision[tmp,prec];
          b[[pivot[[ii]]]] = divide[tmp, a[[pivot[[ii]], ii]]]
          ];
      tmp = Table[,{n}];
      For[ii=1, ii<=n, ii++, tmp[[ii]] = SetPrecision[b[[pivot[[ii]]]],prec]];
      tmp
      ]

End[ ] (* "LinearAlgebra‘GaussianElimination‘Private‘" *)

Protect[LUFactor, LUSolve, LU, SimpleLUFactor];

EndPackage[ ] (* "LinearAlgebra‘GaussianElimination‘" *)
```