Numerical Linear Algebra

- Probably the simplest kind of problem.
- Occurs in many contexts, often as part of larger problem.
- Symbolic manipulation packages can do linear algebra "analytically" (e.g. Mathematica, Maple).
- Numerical methods needed when:
  - Number of equations very large
  - Coefficients all numerical

Linear Systems

- Write linear system as:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

  - This system has *n* unknowns and *m* equations.
  - If *n* = *m*, system is closed.
  - If any equation is a linear combination of any others, equations are degenerate and system is singular.*

*see Singular Value Decomposition (SVD), NRiC 2.6.

Numerical Constraints

- Numerical methods also have problems when:
  1) Equations are degenerate "within round-off error".
  2) Accumulated round-off errors swamp solution (magnitude of *a*'s and *x*'s varies wildly).
- For n,m < 50, single precision usually OK.
- For n,m < 200, double precision usually OK.
- For 200 < n,m < few thousand, solutions possible only for sparse systems (lots of *a*'s zero).

Matrix Form

- Write system in matrix form:

$$A\,\mathbf{x} = \mathbf{b}$$

where:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{mn} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Rows

Columns

Matrix Data Representation

- Recall, C stores data in row-major form:

  $a_{11}, a_{12}, \ldots, a_{1n}; a_{21}, a_{22}, \ldots, a_{2n}; \ldots; a_{m1}, a_{m2}, \ldots, a_{mn}$

- If using "pointer to array of pointers to rows" scheme in C, can reference entire rows by first index, e.g. 3rd row = `a[2]`.
  - ✗ Recall in C array indices start at zero!!
- FORTRAN stores data in column-major form:

  $a_{11}, a_{21}, \ldots, a_{m1}; a_{12}, a_{22}, \ldots, a_{m2}; \ldots; a_{1n}, a_{2n}, \ldots, a_{mn}$

Note on Numerical Recipes in C

- The canned routines in NRiC make use of special functions defined in `nrutil.c` (header `nrutil.h`).
  - In particular, arrays and matrices are allocated dynamically with indices starting at 1, not 0.
  - If you want to interface with the NRiC routines, but prefer the C array index convention, pass arrays by subtracting 1 from the pointer address (i.e. pass `p-1` instead of `p`) and pass matrices by using the functions `convert_matrix()` and `free_convert_matrix()` in `nrutil.c` (see NRiC 1.2 for more information).

Tasks of Linear Algebra

- We will consider the following tasks:

  1) Solve $A\mathbf{x} = \mathbf{b}$, given A and $\mathbf{b}$.

  2) Solve $A\mathbf{x}_i = \mathbf{b}_i$ for multiple $\mathbf{b}_i$'s.

  3) Calculate A^{-1}, where $A^{-1}A = I$, the identity matrix.

  4) Calculate determinant of A, det(A).

- Large packages of routines available for these tasks, e.g. LINPACK, LAPACK (public domain); IMSL, NAG libraries (commercial).
- We will look at methods assuming $n = m$.

The Augmented Matrix

- The equation $A\mathbf{x} = \mathbf{b}$ can be generalized to a form better suited to efficient manipulation:

$$(A|\,\mathbf{b}) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & | & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & | & b_2 \\ \vdots & \vdots & & \vdots & | & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & | & b_n \end{pmatrix}$$

- The system can be solved by performing operations on the augmented matrix.
- The $\mathbf{x}_i$'s are placeholders that can be omitted until the end of the computation.

Elementary Row Operations

- The following row operations can be performed on an augmented matrix without changing the solution of the underlying system of equations:
  I. Interchange two rows.
  II. Multiply a row by a nonzero real number.
  III. Add a multiple of one row to another row.
- The idea is to apply these operations in sequence until the system of equations is trivially solved.

The Generalized Matrix Equation

- Consider the generalized linear matrix equation:

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}}_{\text{coefficients}} \underbrace{\begin{pmatrix} x_{11} & | & x_{12} & | & x_{13} & | & y_{11} & y_{12} & y_{13} & y_{14} \\ x_{21} & | & x_{22} & | & x_{23} & | & y_{21} & y_{22} & y_{23} & y_{24} \\ x_{31} & | & x_{32} & | & x_{33} & | & y_{31} & y_{32} & y_{33} & y_{34} \\ x_{41} & | & x_{42} & | & x_{43} & | & y_{41} & y_{42} & y_{43} & y_{44} \end{pmatrix}}_{\text{solutions and inverse}} = \underbrace{\begin{pmatrix} b_{11} & | & b_{12} & | & b_{13} & | & 1 & 0 & 0 & 0 \\ b_{21} & | & b_{22} & | & b_{23} & | & 0 & 1 & 0 & 0 \\ b_{31} & | & b_{32} & | & b_{33} & | & 0 & 0 & 1 & 0 \\ b_{41} & | & b_{42} & | & b_{43} & | & 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{RHS and identity}}$$

- Its solution simultaneously solves the linear sets:

$$A\mathbf{x}_1 = \mathbf{b}_1,\ A\mathbf{x}_2 = \mathbf{b}_2,\ A\mathbf{x}_3 = \mathbf{b}_3, \text{ and } AY = I,$$

where the $\mathbf{x}_i$'s and $\mathbf{b}_i$'s are column vectors.

Gauss-Jordan Elimination

- GJE uses one or more elementary row operations to reduce matrix A to the identity matrix.
- The RHS of the generalized equation becomes the solution set and Y becomes A^{-1}.
- Disadvantages:
  1) Requires all $\mathbf{b}_i$'s to be stored and manipulated at same time $\Rightarrow$ memory hog.
  2) Don't always need A^{-1}.
- Other methods more efficient, but good backup.

Gauss-Jordan Elimination: Procedure

- Start with simple augmented matrix as example:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & | & b_1 \\ a_{21} & a_{22} & a_{23} & | & b_2 \\ a_{31} & a_{32} & a_{33} & | & b_3 \end{pmatrix}$$

← Row $\mathbf{a}_1|\mathbf{b}_1$

- Divide first row ($\mathbf{a}_1|\mathbf{b}_1$) by first element a_{11}.
- Subtract a_{i1} ($\mathbf{a}_1|\mathbf{b}_1$) from all other rows:

$$\begin{pmatrix} 1 & a_{12}/a_{11} & a_{13}/a_{11} & | & b_1/a_{11} \\ 0 & a_{22}-a_{21}(a_{12}/a_{11}) & a_{23}-a_{21}(a_{13}/a_{11}) & | & b_2-a_{21}(b_1/a_{11}) \\ 0 & a_{32}-a_{31}(a_{12}/a_{11}) & a_{33}-a_{31}(a_{13}/a_{11}) & | & b_3-a_{31}(b_1/a_{11}) \end{pmatrix}$$

← Pivot row

↑ First column of identity matrix

- Continue process for 2^{nd} row, etc.

GJE Procedure, Cont'd

- Problem occurs if leading diagonal element ever becomes zero.
- Also, procedure is numerically unstable!
- Solution: use "pivoting" - rearrange remaining rows (partial pivoting) or rows & columns (full pivoting - requires permutation!) so largest coefficient is in diagonal position.
- Best to "normalize" equations (implicit pivoting).

Gaussian Elimination with Backsubstitution

- If, during GJE, only subtract rows below pivot, will be left with a triangular matrix:

"Gaussian Elimination"

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

  - Solution for x_3 is then trivial: $x_3 = b_3'/a_{33}'$.
  - Substitute into 2nd row to get x_2.
  - Substitute x_3 & x_2 into 1st row to get x_1.
- Faster than GJE, but still memory hog.

LU Decomposition

- Suppose we can write A as a product of two matrices: $A = LU$, where L is lower triangular and U is upper triangular:

$$L = \begin{pmatrix} \times & 0 & 0 \\ \times & \times & 0 \\ \times & \times & \times \end{pmatrix} \qquad U = \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix}$$

- Then $A\mathbf{x} = (LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b}$, i.e. must solve,

  (1) $L\mathbf{y} = \mathbf{b}$; (2) $U\mathbf{x} = \mathbf{y}$

- Can reuse L & U for subsequent calculations.

LU Decomposition, Cont'd

- Why is this better?
  - Solving triangular matrices is easy: just use forward substitution for (1), backsubstitution for (2).
- Problem is, how to decompose A into L and U?
  - Expand matrix multiplication LU to get n^2 equations for $n^2 + n$ unknowns (elements of L and U plus n extras because diagonal elements counted twice).
  - Get an extra n equations by choosing $L_{ii} = 1$ ($i = 1,n$).
  - Then use Crout's algorithm for finding solution to these $n^2 + n$ equations "trivially" (NRiC 2.3).

LU Decomposition in NRiC

- The routines `ludcmp()` and `lubksb()` perform *LU* decomposition and backsubstitution respectively.
- Can easily compute A^{-1} (solve for the identity matrix column by column) and det(*A*) (find the product of the diagonal elements of the *LU* decomposed matrix) - see NRiC 2.3.
- <u>WARNING</u>: for large matrices, computing det(*A*) can overflow or underflow the computer's floating-point dynamic range.

Iterative Improvement

- For large sets of linear equations $A\mathbf{x} = \mathbf{b}$, roundoff error may become a problem.
- We want to know $\mathbf{x}$ but we only have $\mathbf{x} + \delta\mathbf{x}$, which is an exact solution to $A(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$.
- Subtract the exact solution and eliminate $\delta\mathbf{b}$:

$$A\delta\mathbf{x} = A(\mathbf{x} + \delta\mathbf{x}) - \mathbf{b}$$

- The RHS is known, hence can solve for $\delta\mathbf{x}$. Subtract this from the wrong solution to get an improved solution (make sure to use `doubles`!).

Tridiagonal Matrices

- Many systems can be written as (or reduced to):

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \qquad\qquad i = 1,n$$

i.e. a tridiagonal matrix:

$$\begin{bmatrix} b_1 & c_1 & & & & 0 \;\; s \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 \;\; s & & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

Here a_1 and c_n are associated with "boundary conditions" (i.e. x_0 and x_{n+1}).

Sparse Matrices

- *LU* decomposition and backsubstitution is very efficient for tri-di systems: $O(n)$ operations as opposed to $O(n^3)$ in general case.
- Operations on sparse systems can be optimized.

  e.g. Tridiagonal

  Band diagonal with bandwidth M

  Block diagonal

  Banded
- See NRiC 2.7 for various systems & techniques.

Iterative Methods

- For very large systems, direct solution methods (e.g. *LU* decomposition) are slow and RE prone.
- Often iterative methods much more efficient:
  1. Guess a trial solution $\mathbf{x}^0$
  2. Compute a correction $\mathbf{x}^1 = \mathbf{x}^0 + \delta\mathbf{x}$
  3. Iterate procedure until convergence, i.e. $|\delta\mathbf{x}| < \Delta$
- e.g. Congugate gradient method for sparse systems (NRiC 2.7).

Singular Value Decomposition

- Can diagnose or (nearly) solve singular or near-singular systems.
- Used for solving linear least-squares problems.
- Theorem: any $m \times n$ matrix A can be written:

$$A = UWV^T$$

where U $(m \times n)$ & V $(n \times n)$ are orthogonal and W $(n \times n)$ is a diagonal matrix.

- Proof: buy a good linear algebra textbook.

SVD, Cont'd

- The values W_i are zero or positive and are called the "singular values".
- The NRiC routine `svdcmp()` returns U, V, & W given A. You have to trust it (or test it yourself!).
  - Uses Householder reduction, QR diagonalization, etc.
- If A is square then we know:

  $$A^{-1} = V\,[\mathrm{diag}(1/W_i)]\,U^T$$

- This is fine so long as no W_i is too small (or 0).

Definitions

- Condition number cond(A) = (max W_i)/(min W_i).
  - If cond(A) = ∞, A is singular.
  - If cond(A) very large (10^6, 10^{12}), A is ill-conditioned.
- Consider $A\mathbf{x} = \mathbf{b}$. If A is singular, there is some subspace of $\mathbf{x}$ (the nullspace) such that $A\mathbf{x} = 0$.
- The nullity is the dimension of the nullspace.
- The subspace of $\mathbf{b}$ such that $A\mathbf{x} = \mathbf{b}$ is the range.
- The rank of A is the dimension of the range.

The Homogeneous Equation

- SVD constructs orthonormal bases for the nullspace and range of a matrix.
- Columns of U with corresponding non-zero W_i are an orthonormal basis for the range.
- Columns of V with corresponding zero W_i are an orthonormal basis for the nullspace.
- Hence immediately have solution for $A\mathbf{x} = 0$, i.e. the columns of V with corresponding zero W_i.

Residuals

- If **b** ($\neq 0$) lies in the range of A, then the singular equations do in fact have a solution.
- Even if **b** is outside the range of A, can get solution which minimizes residual $r = |A\mathbf{x} - \mathbf{b}|$.
- Trick: replace $1/W_i$ by 0 if $W_i = 0$ and compute

$$\mathbf{x} = V\,[\mathrm{diag}\,(1/W_i)]\,(U^T\,\mathbf{b})$$

- Similarly, can set $1/W_i = 0$ if W_i very small.

Approximation of Matrices

- Can write $A = UWV^T$ as:

$$A_{ij} = \sum_{k=1}^{N} W_k U_{ik} V_{jk}$$

- If most of the singular values W_k are small, then A is well-approximated by only a few terms in the sum (strategy: sort W_k's in descending order).
- For large memory savings, just store the columns of U and V corresponding to non-negligible W_k's.
- Useful technique for digital image processing.