Class 6. Numerical Linear Algebra, Part 1

- 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, etc.).
- Numerical methods needed when:
  - Number of equations very large.
  - One or more coefficients numerical.

Linear Systems

- Write linear system as:

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

  - This system has n unknowns and m equations.
  - If $n = m$, system is closed.
  - If $m \leq n$ and any equation is a linear combination of any others, equations are <u>degenerate</u> and system is <u>singular</u>.

Numerical Constraints

- Numerical methods have their own problems when:
  1. Equations are degenerate "within round-off error."
  2. Accumulated round-off errors swamp solution (magnitudes of a's and x's vary wildly).
- For $n, m < 50$, single precision usually OK (but why bother?).
- 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:

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

  where:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}.$$

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., 3$^{\text{rd}}$ 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 normal 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 $\mathbf{A}\mathbf{x} = \mathbf{b}$, given $\mathbf{A}$ and $\mathbf{b}$.
  2. Solve $\mathbf{A}\mathbf{x}_i = \mathbf{b}_i$ for multiple $\mathbf{b}_i$'s.
  3. Calculate $\mathbf{A}^{-1}$, where $\mathbf{A}^{-1}\mathbf{A} = \mathbf{1}$, the identity matrix.
  4. Calculate the determinant of $\mathbf{A}$, $\det(\mathbf{A})$.

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

The Augmented Matrix

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

$$(\mathbf{A}|\mathbf{b}) = \left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right).$$

- 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:

  1. Interchange two rows.
  2. Multiply a row by a nonzero real number.
  3. 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:

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

- Its solution simultaneously solves the linear sets:

  $\mathbf{Ax}_1 = \mathbf{b}_1$, $\mathbf{Ax}_2 = \mathbf{b}_2$, $\mathbf{Ax}_3 = \mathbf{b}_3$, and $\mathbf{AY} = \mathbf{1}$,

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

Procedure

- Start with simple augmented matrix as example:

$$\left(\begin{array}{ccc|c} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array}\right)$$

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

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

- Continue process for 2^{nd} row, etc.
- Problem occurs if leading diagonal element ever becomes <u>zero</u>.
- Also, procedure is numerically unstable (in presence of RE)!
- Solution: use "pivoting"—rearrange remaining rows (partial pivoting) or rows *and* columns (full pivoting—requires permutation!) so largest coefficient is in diagonal position.
- Best to "normalize" equations (implicit pivoting) so largest coefficient in each row is exactly unity before starting the procedure.

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 2$^{\text{nd}}$ row to get x_2.
  - Substitute x_3 and x_2 into 1$^{\text{st}}$ row to get x_1.

- Faster than GJE, but still memory hog.