Introduction & Systems of Linear Equations

Paul Heckbert

Computer Science Department
Carnegie Mellon University

State of the Art in Scientific Computing

- aerospace: simulate subsonic & supersonic air flow around full aircraft, no wind tunnel
  - divide space into millions of tetrahedra or parallelepipeds, solve sparse linear or nonlinear PDE
- nuclear: simulate nuclear explosion!
  - especially important because of nuclear test bans
- weather prediction: entire planet, 2 weeks into future
- astrophysics: galactic collisions
- automotive: simulate car crash
- biology: simulate protein folding – drug design

Strategies for Simplifying Problems

- replace infinite process with finite process
  - e.g. integrals to sums
- replace general matrices with simple matrices
  - e.g. diagonal
- replace complex functions with simple ones
  - e.g. polynomials
- replace nonlinear problems with linear problems
- replace differential equations with algebraic equations
  - e.g. linear systems
- replace high-order systems with low-order systems
- replace infinite-dimensional spaces with finite-dim. ones
  - e.g. all real functions on [0,1] with samples on *n*-element grid

Sources of Error

error type	***example: car crash simulation***
modeling	*approximate car geometry*
empirical measurements	*incorrect tire friction coeff.*
previous computations	*error in initial speed of car*
truncation or discretization	*numerical solution to dif.eq.*
rounding	*used floats, not doubles*

Each step introduces some error, but magnitudes may differ greatly.

Look for the largest source of error – the *weak link in the chain.*

Quantifying Error

(absolute error) = (approximate value) – (true value)

$$\text{(relative error)} = \frac{\text{(absolute error)}}{\text{(true value)}} = \frac{\text{(approximate value)}}{\text{(true value)}} - 1$$

Fundamental difficulty with measuring error:

For many problems we cannot compute the exact answer, we can only approximate it!

Often, the best we can do is *estimate* the error!

Significant Digits

```
main() {
    float f = 1./3.;
    printf("%.20f\n", f);  // print to 20 digits
}
```

```
0.33333334326744080000
```

we get 7 *significant digits*; the rest is *junk*!

When reporting results, only show the significant digits!

IEEE Floating Point Format

- very widely used standard for floating point
- **C float** is 4 bytes: 24 bit mantissa, 8 bit exponent
  - about 7 significant digits
  - smallest pos. no: 1.3e-38, largest: 3.4e+38
- **C double** is 8 bytes: 53 bit mantissa, 11 bit exponent
  - about 16 significant digits
  - smallest pos.: 2.3e-308, largest: 1.7e+308
- special values
  - **`Inf`** - infinity (e.g. 1/0)
  - **`NaN`** - "not a number", undefined (e.g. 0/0)

C program to test floating point

```
#include <math.h>

main() {
  int i;
  float f;
  double d;

  for (i=0; i<55; i++) {
   f = 1. + pow(.5, i);
   d = 1. + pow(.5, i);
   printf("%2d  %.9f  %.18f\n", i, f, d);
  }
}
```

Output: precision of float & double

```
 i  float         double   1+2^(-i)

 0  2.000000000  2.00000000000000000000
 1  1.500000000  1.50000000000000000000
 2  1.250000000  1.25000000000000000000
 3  1.125000000  1.12500000000000000000
 4  1.062500000  1.06250000000000000000

21  1.000000477  1.00000047683715820000
22  1.000000238  1.00000023841857910000
23  1.000000119  1.00000011920928960000
24  1.000000000  1.00000005960464480000

50  1.000000000  1.00000000000000090000
51  1.000000000  1.00000000000000040000
52  1.000000000  1.00000000000000020000
53  1.000000000  1.00000000000000000000
```

Condition Number of a Problem

- some problems are harder to solve accurately than others
- The *condition number* is a measure of how sensitive a problem is to changes in its input

$$\text{Cond} = \frac{\left|\text{relative change in output}\right|}{\left|\text{relative change in input}\right|} = \frac{\left|\left[f(\hat{x}) - f(x)\right]/f(x)\right|}{\left|\left[\hat{x} - x\right]/x\right|} = \left|\frac{f'(x)}{f(x)}\,x\right|$$

where $f(x)$ represents the exact solution to problem with input x

- Cond<1 or so   problem is well-conditioned
- Cond>>1   problem is ill-conditioned

Condition Number -- Examples

well conditioned	**ill-conditioned**
taking a step on level ground	step near cliff
tan(x) near x=45°, say	tan(x) near x=90° (because *f*’ infinite)
cos(x) not near x=90°	cos(x) near x=90° (because *f* zero)

Systems of Linear Equations

- Solve Ax=b for x
- A is $n \times n$ matrix
- x and b are *n*-vectors (column matrices)

- Later we’ll look at overdetermined and underdetermined systems, where the matrix is not square (#equations not equal to #unknowns)

Matrix Properties

For a square, $n \times n$ matrix:

- rank is the max. no. of linearly independent rows or columns
- full rank = rank is n
- rank-deficient = rank is less than n
- singular matrix = determinant zero = no inverse = linearly dependent = rank-deficient = (Ax=0 for some nonzero x)

Matrix Rank Examples

Rank 2

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 2 \\ -.001 & 0 \end{bmatrix}$$

Rank 1

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ -3 & -6 \end{bmatrix}$$

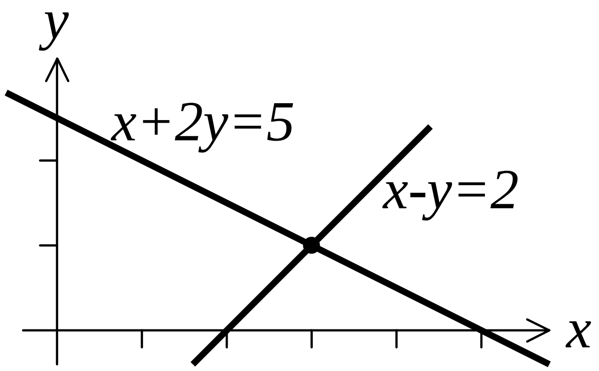
Rank 0

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

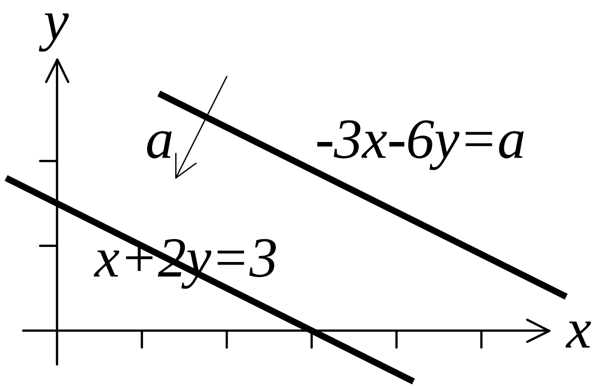
Geometric Interpretation - 2×2 System

intersection of 2 lines

$$\begin{bmatrix} 1 & 2 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix} \Leftrightarrow \begin{matrix} x + 2y = 5 \\ x - y = 2 \end{matrix}$$



$$\begin{bmatrix} 1 & 2 \\ -3 & -6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ a \end{bmatrix} \Leftrightarrow \begin{matrix} x + 2y = 3 \\ -3x - 6y = a \end{matrix}$$



Rank 1 matrix means lines are parallel.

For most *a*, lines non-coincident, so no solution.

For *a*=-9, lines coincident, one-dimensional subspace of solutions.

Gaussian Elimination and LU Decomposition

Gaussian Elimination on square matrix *A*

$$A = \begin{bmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{bmatrix} = LU$$

computes an *LU decomposition*

L is unit lower triangular (1’s on diagonal)

U is upper triangular

Gaussian Elimination - comments

- G.E. can be done on any square matrix
- if *A* singular then diagonal of *U* will contain zero(s)
- usually partial pivoting is used (swapping rows during elimination) to reduce errors

- G.E. is an example of an *explicit* method for solving linear systems – solve for solution in one sweep
- Other, more efficient algorithms can be used for specialized matrix types, as we'll see later

Solving Systems with LU Decomposition

to solve $Ax=b$:

- decompose A into LU -- cost $2n^3/3$ flops
- solve $Ly=b$ for y by forw. substitution -- cost n^2 flops
- solve $Ux=y$ for x by back substitution -- cost n^2 flops

slower alternative:

- compute A^{-1} -- cost $2n^3$ flops
- multiply $x=A^{-1}b$ -- cost $2n^2$ flops

this costs about 3 times as much as LU

lesson:

- if you see A^{-1} in a formula, read it as “solve a system”, not “invert a matrix”

Symmetric Positive Definite

Symmetric Positive Definite – an important matrix class

- symmetric: $A=A^T$
- positive definite: $x^TAx>0$ for $x\neq0$ $\Leftrightarrow$ all $\lambda_i>0$

if A is spd,

LU decomposition can be written $A=LL^T$,

where L is lower triangular (not unit)

this is the *Cholesky factorization* -- cost $n^3/3$ flops

no pivoting required

Cramer’s Rule

- A method for solving $n \times n$ linear systems
- What is its cost?

Vector Norms

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$ 1-norm, Manhattan norm

$$\|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}}$$ 2-norm, Euclidean norm

$$\|x\|_\infty = \max_i |x_i|$$ ∞-norm

norms differ by at most a constant factor, for fixed *n*

$$\|x\|_\infty \le \|x\|_2 \le \|x\|_1 \le \sqrt{n}\|x\|_2 \le n\|x\|_\infty$$

Matrix Norm

matrix norm defined in terms of vector norm:

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

geometric meaning: the maximum stretch resulting from application of this transformation

exact result depends on whether 1-, 2-, or ∞-norm is used

Condition Number of a Matrix

A measure of how close a matrix is to singular

$$\text{cond}(A) = \kappa(A) = \|A\| \cdot \|A^{-1}\|$$

$$= \frac{\text{maximum stretch}}{\text{maximum shrink}} = \frac{\max_i |\lambda_i|}{\min_i |\lambda_i|}$$

- cond(I) = 1
- cond(singular matrix) = ∞