Notes

- Assignment 1 is out (questions?)

Linear Algebra

- Last class:
  we reduced the problem of “optimally” interpolating scattered data to solving a system of linear equations
- Typical: often almost all of the computational work in a scientific computing code is linear algebra operations

Basic Definitions

- Matrix/vector notation
- Dot product, outer product
- Vector norms
- Matrix norms

BLAS

- Many common matrix/vector operations have been standardized into an API called the BLAS (Basic Linear Algebra Subroutines)
  - Level 1: vector operations
    copy, scale, dot, add, norms, …
  - Level 2: matrix-vector operations
    multiply, triangular solve, …
  - Level 3: matrix-matrix operations
    multiply, triangular solve, …
- FORTRAN bias, but callable from other langs
- Goals:
  - As fast as possible, but still safe/accurate
- www.netlib.org/blas

Speed in BLAS

- In each level:
  multithreading, prefetching, vectorization, loop unrolling, etc.
- In level 2, especially in level 3: blocking
  - Operate on sub-blocks of the matrix that fit the memory architecture well
- General goal:
  if it’s easy to phrase an operation in terms of BLAS, get speed+safety for free
  - The higher the level better

LAPACK

- The BLAS only solves triangular systems
  - Forward or backward substitution
- LAPACK is a higher level API for matrix operations:
  - Solving linear systems
  - Solving linear least squares problems
  - Solving eigenvalue problems
- Built on the BLAS, with blocking in mind to keep high performance
- Biggest advantage: safety
  - Designed to handle difficult problems gracefully
- www.netlib.org/lapack

Specializations

- When solving a linear system, first question to ask: what sort of system?
- Many properties to consider:
  - Single precision or double?
  - Real or complex?
  - Invertible or (nearly) singular?
  - Symmetric/Hermitian?
  - Definite or Indefinite?
  - Dense or sparse or specially structured?
  - Multiple right-hand sides?
- LAPACK/BLAS take advantage of many of these (sparse matrices the big exception…)

Accuracy

- Before jumping into algorithms, how accurate can we hope to be in solving a linear system?
- Key idea: **backward error analysis**
- Assume calculated answer is the **exact solution** of a **perturbed problem**.

- The condition number of a problem: how much errors in data get amplified in solution

Condition Number

- Sometimes we can estimate the condition number of a matrix a priori
- Special case: for a symmetric matrix, 2-norm condition number is ratio of extreme eigenvalues

- LAPACK also provides cheap estimates
  - Try to construct a vector $\|x\|$ that comes close to maximizing $\|A^{-1}x\|$

Gaussian Elimination

- Let's start with the simplest unspecialized algorithm: Gaussian Elimination
- Assume the matrix is invertible, but otherwise nothing special known about it
- GE simply is row-reduction to upper triangular form, followed by backwards substitution
  - Permuting rows if we run into a zero

LU Factorization

- Each step of row reduction is multiplication by an elementary matrix
- Gathering these together, we find GE is essentially a matrix factorization:

  $$A=LU$$

  where
  L is lower triangular (and unit diagonal),
  U is upper triangular
- Solving Ax=b by GE is then

  $$Ly=b$$
  $$Ux=y$$

Block Approach to LU

- Rather than get bogged down in details of GE (hard to see forest for trees)
- Partition the equation A=LU
- Gives natural formulas for algorithms
- Extends to block algorithms

Cholesky Factorization

- If A is symmetric positive definite, can cut work in half: $A=LL^T$
  - L is lower triangular
- If A is symmetric but indefinite, possibly still have the Modified Cholesky factorization: $A=LDL^T$
  - L is unit lower triangular
  - D is diagonal