2.4 SOLVING SYSTEMS OF LINEAR EQUATIONS

L and U Matrices

- Lower Triangular Matrix

$$[L]=\begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{34} & l_{44} \end{bmatrix}$$

- Upper Triangular Matrix

$$[U]=\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{13} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

LU Decomposition for Ax=b

- LU **decomposition / factorization**

  $[A]\{x\} = [L][U]\{x\} = \{b\}$
- **Forward substitution**

  $[L]\{d\} = \{b\}$
- **Back substitution**

  $[U]\{x\} = \{d\}$
- **Q:Why might I do this instead of Gaussian elimination?**

Complexity of 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

Cholesky LU Factorization

- **If [A] is symmetric and positive definite, it is convenient to use Cholesky decomposition.**

$$[A] = [L][L]^T = [U]^T[U]$$

- **No pivoting or scaling needed if [A] is symmetric and positive definite (all eigenvalues are positive)**
- **If [A] is not positive definite, the procedure may encounter the square root of a negative number**
- **Complexity is ½ that of LU (due to symmetry exploitation)**

Cholesky LU Factorization

- $[A] = [U]^T[U]$
- **Recurrence relations**

$$u_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2}$$

$$u_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj}}{u_{ii}} \quad for \;\; j = i+1, \ldots, n$$

Pivoting in LU Decomposition

- Still need pivoting in LU decomposition (why?)
- Messes up order of [L]
- What to do?
- Need to pivot both [L] and a permutation matrix [P]
- Initialize [P] as identity matrix and pivot when [A] is pivoted. Also pivot [L]

LU Decomposition with Pivoting

- **Permutation matrix [P]**
  - permutation of identity matrix [I]
- **Permutation matrix performs "bookkeeping" associated with the row exchanges**
- **Permuted matrix [P] [A]**
- **LU factorization of the permuted matrix**

$$[P][A] = [L][U]$$

- **Solution**

$$[L][U]\{x\} = [P]\{b\}$$

LU-factorization for real symmetric Indefinite matrix A (constrained optimization has saddle points)

LU – factorization $$A=\left(\begin{array}{c|c} E & c^T \\ \hline c & B \end{array}\right)=\left(\begin{array}{c|c} I & \\ \hline cE^{-1} & I \end{array}\right)\left(\begin{array}{c|c} E & c^T \\ \hline & B-cE^{-1}c^T \end{array}\right)$$

LDL^T – factorization $$A=\left(\begin{array}{c|c} E & c^T \\ \hline c & B \end{array}\right)=\left(\begin{array}{c|c} I & \\ \hline cE^{-1} & I \end{array}\right)\left(\begin{array}{c|c} E & \\ \hline & B-cE^{-1}c^T \end{array}\right)\left(\begin{array}{c|c} I & E^{-1}c^T \\ \hline & I \end{array}\right)$$

where $$L=\left(\begin{array}{c|c} I & \\ \hline cE^{-1} & I \end{array}\right)$$ and $$L^T=\left(\begin{array}{c|c} I & E^{-T}c^T \\ \hline & I \end{array}\right)=\left(\begin{array}{c|c} I & E^{-1}c^T \\ \hline & I \end{array}\right)$$

Question: 1) If A is not singular, can I be guaranteed to find a nonsingular principal block E after pivoting? Of what size?

2) Why not *LU*-decomposition?

History of LDL' decomposition: 1x1, 2x2 pivoting

- diagonal pivoting method with complete pivoting:
  Bunch-Parlett, "Direct methods fro solving symmetric indefinite systems of linear equations," SIAM J. Numer. Anal., v. 8, 1971, pp. 639-655

- diagonal pivoting method with partial pivoting:
  Bunch-Kaufman, "Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems," Mathematics of Computation, volume 31, number 137, January 1977, page 163-179

- DEMOS

2.4 COMPLEXITY OF LINEAR ALGEBRA; SPARSITY

Complexity of 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

Complexity of linear algebra

lesson:

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

Cholesky factorization -- cost $n^3/3$ flops

LDL' factorization -- cost $n^3/3$ flops

Q: What is the cost of Cramer's rule (roughly)?

Sparse Linear Algebra

- Suppose you are applying matrix-vector multiply and the matrix has lots of zero elements
  - Computation cost? Space requirements?
- General sparse matrix representation concepts
  - Primarily only represent the nonzero data values (nnz)
  - Auxiliary data structures describe placement of nonzeros in "dense matrix"
- And *MAYBE* LU or Cholesky can be done in O(nnz), so not as bad as (O(n^3)); since very oftentimes nnz=O(n)

Sparse Linear Algebra

- Because of its phenomenal computational and storage savings potential, sparse linear algebra is a huge research topic.
- VERY difficult to develop.
- Matlab implements sparse linear algebra based on i,j,s format.
- DEMO
- Conclusion: Maybe I can SCALE well … Solve O(10^12) problems in O(10^12).

SUMMARY SECTION 2

- The heaviest components of numerical software are Numerical differentiation (AD/DIVDIFF) and linear algebra.
- Factorization is always preferable to direct (Gaussian) elimination.
- Keeping track of sparsity in linear algebra can enormously improve performance.

3.1 FAILURE OF NEWTON METHODS

Problem definition

$$min\ f(x)$$

$$f : R^n \rightarrow R$$

- continuously differentiable
- gradient is available
-Hessian is unavailable

Necessary optimality conditions: $\nabla f(x^*) = 0$

Sufficient optimality conditions: $\nabla^2 f(x^*) \succ 0$

DEMO

- Algorithm: Newton.
- Note: not only does the algorithm not converge, the function values go to infinity.
- So we should have known ahead of time we should have done something else earlier.

Ways of enforcing that thinks do not blow up or wander

- 1. Line-search methods.
  - Make a "guess" of a good direction.
  - Make good progress along that direction. At least know you will decrease f.
- 2. Trust region model.
  - Create a quadratic model of the function.
  - Define a region where we "believe"—"trust" the model and find a "good" point in that "region".
  - If at that point the model is far from f, less trust—smaller region, if not, more –larger region.

3.2 LINE SEARCH METHODS

3.2.1 LINE SEARCH METHODS: ESSENTIALS

Line Search Methods Idea:

- At the current point x_k find a "Newton-like" direction d_k
- Along that direction d_k do 1-dimensional minimization (simpler than over whole space)

$$x_{k+1} \approx \arg\min_{\alpha} f(x_k + \alpha d_k)$$

- Because the line search always decreases f, we will have an accumulation point (cannot diverge if bounded below) – unlike Newton proper

Descent Principle

- Descent Principle: Carry Out a one-Dimensional Search Along a Line where I will decrease the function.

$$g(\alpha) = f(x_k + \alpha p_k) \qquad for \quad \nabla f(x_k)' p_k < 0$$

- If this happens, there exists an alpha (why?) such that.

$$f(x_k + \alpha p_k) < f(x_k)$$

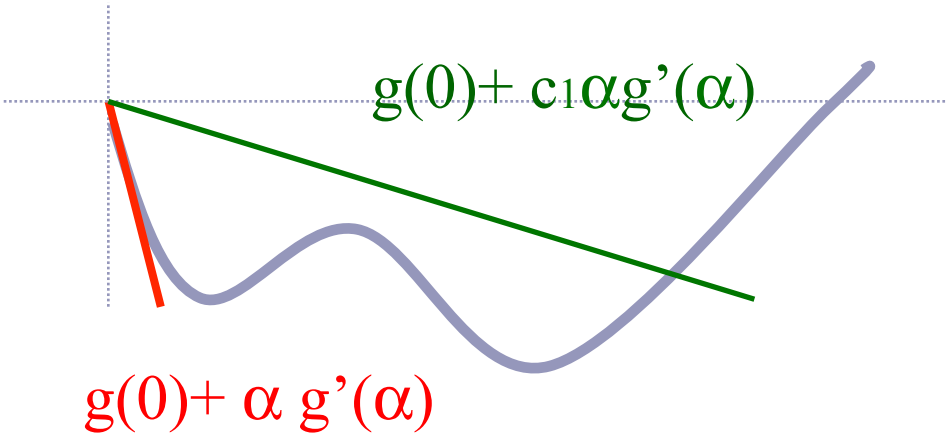
- So I will keep making progress.
- Typical choice (why)? $B_k p_k = -\nabla f(x_k); \quad B_k \succ 0$
- Newton may need to be modified (why?)

Line Search-Armijo

- I cannot accept just about ANY decrease, for I may NEVER converge (why , example of spurious convergence).
- IDEA: Accept only decreases PROPORTIONAL TO THE SQUARE OF GRADIENT. Then I have to converge (since process stops only when gradient is 0).
- Example: Armijo Rule. It uses the concept of BACKTRACKING.

$$f(x_k) - f(x_k + \beta^m \tau_k d_k) \geq -\rho \beta^m \tau_k \nabla f(x_k)^T d_k$$

$$\beta \in (0,1) \qquad \rho \in (0,1/2)$$



Some Theory

Global Convergence:

Let f be twice continuously differentiable on an open set $\mathcal{D}$, and assume that the starting point x_0 of Algorithm 3.2 is such that the level set $\mathcal{L} = \{x \in \mathcal{D} : f(x) \leq f(x_0)\}$ is compact. Then if the bounded modified factorization property holds, we have that

$$\lim_{k \to \infty} \nabla f(x_k) = 0.$$

$$\kappa(B_k) = \|B_k\| \, \|B_k^{-1}\| \leq C, \qquad \text{some } C > 0 \text{ and all } k = 0, 1, 2, \ldots.$$

Fast Convergence:

Newton is accepted by LS

Suppose that f is twice differentiable and that the Hessian $\nabla^2 f(x)$ is Lipschitz continuous (see (A.42)) in a neighborhood of a solution x^* at which the sufficient conditions (Theorem 2.4) are satisfied. Consider the iteration $x_{k+1} = x_k + p_k$, where p_k is given by (3.30). Then

(i) *if the starting point x_0 is sufficiently close to x^*, the sequence of iterates converges to x^*;*

(ii) *the rate of convergence of $\{x_k\}$ is quadratic; and*

(iii) *the sequence of gradient norms $\{\|\nabla f_k\|\}$ converges quadratically to zero.*

Extensions

- Line Search Refinements:
  - Use interpolation
  - Wolfe and Goldshtein rule
- Other optimization approaches
  - Steepest descent,
  - CG ….