Document: ECS 231C (Spring 2012)
Professor: Bai
Latest Update: June 6, 2012
Author: Jeff Irion
http://www.math.ucdavis.edu/~jlirion

Contents

1 4-2-12 **2**
1.1 Algorithms . . . 2
1.2 Introductory Handout . . . 3

2 4-4-12 **6**
2.1 Forward and Backward Error . . . 6
2.2 Frequently Used Matrix Factorizations . . . 6
2.3 Matrix Decomposition Handout . . . 8

3 4-6-12 **11**
3.1 Matrix Factorizations (Continued) . . . 11

4 4-9-12 **13**
4.1 Matrix Norms Handout . . . 13

5 4-11-12 **15**
5.1 IEEE Floating Point Arithmetic . . . 15
5.2 Floating Point Handout . . . 16

6 4-18-12 **23**

7 4-20-12 **24**
7.1 Segment 1 Recap . . . 24

8 4-23-12 **25**
8.1 Large Scale Linear Systems . . . 25
8.2 Subspace Projection Methods . . . 25
8.3 Linear Solvers Handout 1 . . . 26

9 4-25-12 **30**
9.1 Large Scale Linear Systems . . . 30
9.1.1 One-Dimensional Situation . . . 30
9.2 Linear Solvers Handout 2 . . . 31

10 4-27-12 **35**
10.1 Framework . . . 35
10.2 Implementation . . . 35
10.3 GMRES (version 0) . . . 36

11 4-30-12 **37**
11.1 Convergence Test/Stopping Criterion . . . 37
11.2 GMRES (version 0) . . . 37
11.3 The CG Method . . . 37
11.4 Linear Solvers Handout 3 . . . 39

12 5-2-12 **45**
12.1 Linear Systems . . . 45
12.2 Preconditioning . . . 46
12.3 Linear Solvers Handout 4 . . . 47

13 5-4-12 **54**
13.1 Large Scale Eigenvalue Computations . . . 54
13.2 The Power Method . . . 54
13.3 Large Scale Eigenvalue Computations Handout 1 . . . 55

14 5-11-12 **59**
14.1 Large Scale Eigenvalue Computations . . . 59
14.1.1 Method 1: The Power Method . . . 59

15 5-14-12 **60**
15.1 Large Scale Eigenvalue Computations . . . 60
15.1.1 (Krylov) Subspace Projection Methods . . . 60
15.2 Large Scale Eigenvalue Computations Handout 2 . . . 62

16 5-16-12 **67**
16.1 Large Scale Eigenvalue Computations . . . 67
16.2 Large Scale Eigenvalue Computations Handout 3 . . . 68

17 5-23-12 **71**

18 6-1-12 **72**
18.1 Fast Solvers . . . 72
18.1.1 Poisson's Equation . . . 72
18.2 Kronecker ("Tensor") Product of Matrices . . . 72
18.3 Vectorization . . . 72
18.4 Poisson Equation Handout . . . 73

19 6-4-12 **79**
19.1 Fast Solvers . . . 79
19.2 Block Cyclic Reduction . . . 79
19.3 Fast Solvers for Poisson Equation Handout . . . 80

1 4-2-12

1.1 Algorithms

Integer

- GF(2)
- Symbolic
- Numerical ← finite precision arithmetic

1. Scientific computing and computational science

   Scientific computing is about the design and analysis of numerical algorithms and engineering software for solving mathematical problems in finite precision arithmetic.

   Computational science involves innovative and essential use of high performance computation, and/or the development of computational technologies to advance knowledge or capabilities in scientific and engineering disciplines. A necessary element in computational science is a strong, close tie to an application discipline. Research in computation is inherently multidisciplinary and includes, for example, environmental modeling, simulation of complex physical systems that generate energy, semiconductor design, modeling DNA sequences and protein structure, and the simulation and analysis of flow through geologic structures (Ref: DOE's computational science graduate fellowship program).

2. Algorithms as a technology, computational simulation as the *third pillar* of science.

3. Algorithm general strategy: *to* **replace/reduce** *a difficult problem with an easier one that has the same solution or a closely related solution.*

   Example. Consider solving the linear equations $Ax = b$ for x. If we can write $A = LU$, where L and U are lower and upper triangular matrices, respectively, then it is equivalent to solve triangular linear equations $Ly = b$ for y and $Ux = y$ for x. The triangular linear equations can be computed straightforward by forward and back substitutions.

4. Approximation and error (*not mistake!*) are the facts of life.

   Sources of errors:

   measurement and data uncertainty,
   modeling,
   truncation (discretization),
   rounding in finite precision arithmetic

   For example, consider $f : \mathbf{R} \longrightarrow \mathbf{R}$

   $$x \longrightarrow f(x)$$

   We have an inexact input $\widehat{x}$, and approximate function $\widehat{f}$ constructued by some algorithm, then

   $$\begin{aligned} \text{total error} &= \widehat{f}(\widehat{x}) - f(x) \\ &= [\widehat{f}(\widehat{x}) - f(\widehat{x})] + [f(\widehat{x}) - f(x)] \\ &= \underbrace{\text{computational errors}}_{\text{trunction/rounding}} + \underbrace{\text{propagated data errors}}_{\text{(conditioning)}\times\text{(data error)}} \end{aligned}$$

5. Two error measurements: absolute error and relative error

   Let $\widehat{x}$ be an approximation of x. Then the *absolute error* is defined by

   $$\text{abserr}(x) = |\widehat{x} - x|,$$

and the *relative error* (assume that x is a nonzero number) is defined by

$$\text{relerr}(x) = |\rho| := \frac{|\widehat{x} - x|}{|x|}.$$

Remarks:

- The relative error is independent of scaling.
- $\widehat{x} = x(1+\rho)$, where $|\rho|$ is the relative error .
- *Rule of Thumb*: if $|\rho| = O(10^{-d})$, then x and $\widehat{x}$ agree to about d significant digits, and conversely.

6. Forward error analysis and backward error analysis

   Suppose that an approximation $\widehat{y}$ to $y = f(x)$ is computed. How should we measure the "*quality*" of $\widehat{y}$?

   Ideally, we would like to have the *forward error*

   $$\text{relerr}(y) = \frac{|y - \widehat{y}|}{|y|} = \text{"tiny"}.$$

   However, we don't know y! Instead, we ask "for what set of data have we actually solved our problem?" That is, for what Δx, do we have

   $$\widehat{y} = f(x + \Delta x)?$$

   $|\Delta x|$ (or $\min |\Delta x|$ if there are many such Δx) is called *backward error*.

   Two main motiviations for using backward error analysis:

   - interprets errors as being equivalent to perturbations in the data,
   - reduces the question of bounding or estimating the forward error to perturbation theory, for which many problems is well understood (and only has to be developed once, for the given problem, and not for each method.)

7. An algorithm for computing $y = f(x)$ is called *(backward) stable* if, for any x, it produces a computed $\widehat{y}$ with a small backward error, that is, $\widehat{y} = f(x + \Delta x)$ for some small Δx.

8. Conditioning of problems: the relationship between forward and backward errors for a problem is governed by the *conditioning* of the problem, that is, the sensitivity of the solution to perturbation in the data.

   Example: compute $y = f(x)$. Let the computed results in terms of backward error $\widehat{y} = f(x + \Delta x)$. Then the absolute error is

   $$\widehat{y} - y = f(x + \Delta x) - f(x) = f'(x)\Delta x + O\left((\Delta x)^2\right).$$

   Correspondingly, the relative error is given by

   $$\frac{\widehat{y} - y}{y} = \frac{x \cdot f'(x)}{f(x)} \left(\frac{\Delta x}{x}\right) + O((\Delta)^2).$$

where

$$\kappa_f(x) = \left| \frac{x \cdot f'(x)}{f(x)} \right|$$

The quantity $\kappa_f(x)$ is called the *condition number of f at x*. It measures approximately how much the relative backward error in x is magnified by evaluating of f at x.

Rule of Thumb:

$$|\text{forward error}| \leq (\text{condition number}) \times |\text{backward error}|.$$

The computed solution to an ill-conditioned (i.e., large condition number) problem can have a large forward error, even for small backward error!

2 4-4-12

2.1 Forward and Backward Error

Example 2.1.

$$f : \mathbb{R} \to \mathbb{R}$$

Ideally, we have $x \to y = f(x)$. In practice, $\hat{x} \xrightarrow{\hat{f}} \hat{y}$. The forward error is

$$\frac{|\hat{y} - y|}{|y|}.$$

The backward error (A. Turing, J. von Numan) is

$$\hat{y} = f(x + \underbrace{\Delta x}_{\substack{\text{backward} \\ \text{error}}}).$$

We say the algorithm is *stable* if it has a small $|\Delta x|$.

$$\text{Forward error} \leq \text{condition number} \times \underbrace{\text{backward error}}_{\frac{|\Delta x|}{|x|}}$$

The condition number is intrinsic to the problem, while the backward error is from the algorithm.

Matrices are our fundamental structure.

Simple Matrices:

- I = the identity matrix
- D = diagonal matrix
- L = lower triangular matrix
- R = upper triangular matrix

2.2 Frequently Used Matrix Factorizations

1. LU factorization. For any square matrix A, we can write $PA = LU$, where P is a permutation matrix, L is lower triangular, and U is upper triangular. Applications:
   - Solving a linear system.

$$\begin{aligned} Ax &= b \\ PAx &= Pb \\ L\underbrace{(Ux)}_{y} &= \tilde{b} \\ Ly &= \tilde{b} \\ Ux &= y \end{aligned}$$

- Finding a determinant.

$$\begin{aligned} \det(PA) &= \det(LU) \\ \underbrace{\det(P)}_{=\pm 1} \det(A) &= \det(L)\det(U) \\ \det(A) &= \pm(l_{11}\cdots l_{nn})(u_{11}\cdots u_{nn}) \end{aligned}$$

- Inverting a matrix.

$$\begin{aligned} (PA)^{-1} &= (LU)^{-1} \\ A^{-1}P^T &= U^{-1}L^{-1} \\ A^{-1} &= U^{-1}L^{-1}P \end{aligned}$$

2. QR decomposition. For any $n \times m$ matrix A, we can write

$$\underbrace{A}_{n\times m} = \underbrace{Q}_{n\times n}\underbrace{R}_{n\times m},$$

where R is upper triangular and Q is an orthogonal matrix: $Q^TQ = I \;\Leftrightarrow\; Q^{-1} = Q^T$. Applications:

- We can use QR factorization to perform the Gram-Schmidt orthogonalization process.

$$\begin{aligned} A = QR &= \begin{bmatrix} \underbrace{Q_1}_{n\times m} & \underbrace{Q_2}_{n\times n-m} \end{bmatrix} \begin{bmatrix} R_1 \\ \mathbf{0} \end{bmatrix} \\ &= Q_1R_1 \end{aligned}$$

- Least squares, a.k.a. linear regression: $\min_x \|Ax - b\|_2$. This problem always has a solution.

1. **LU decomposition.** If A is a square nonsingular matrix, then there exist a permutation matrix P, a unit lower triangular matrix L, and a upper triangular matrix U such that

$$PA = LU.$$

Examples of applications:

- LU decomposition is *Gaussian Elimination* in matrix form
- Solve $Ax = b$.
- Compute $\det(A)$.
- Compute A^{-1}, if really necessary.

Special cases:

(a) Cholesky decomposition. A matrix A is symmetric positive definite *if and only if* there exists a unique nonsingular upper triangular matrix R, with positive diagonal entries, such that

$$A = R^T R.$$

(b) $\mathrm{LDL}^{\mathrm{T}}$ factorization If $A^T = A$ is nonsingular, then there exists a permutation P, a unit lower triangular matrix L, and a block diagonal matrix D with 1-by-1 and 2-by-2 blocks such that

$$PAP^T = LDL^T.$$

2. **QR decomposition.** Let A be m-by-n with $m \geq n$. Suppose that A has full column rank. Then there exist a unique m-by-n orthogonal matrix Q (i.e. $Q^T Q = I$) and a unique n-by-n upper triangular matrix R with positive diagonal $r_{ii} > 0$ such that

$$A = QR.$$

Examples of applications:

- Find an orthonormal basis of the subspace spanned by the columns of A (the Gram-Schmidt orthogonalization process)
- Solve the linear least squares problem $\min_x \|Ax - b\|_2$.
- ...

3. **Schur decomposition.** Let A be of order n. Then there is an $n \times n$ unitary matrix U (i.e. $U^H U = I$) such that

$$A = UTU^H,$$

where T is upper triangular.

Variant: Real Schur decomposition.

Examples of applications:

- The eigenvalues of A are the diagonal elements of T.
- *Eigenvalue decomposition, if exists*

$$A = X\Lambda X^{-1},$$

where Λ is a diagonal matrix.

- Compute matrix functions $f(A) = Uf(T)U^H$.
- ...

4. **Singular Value Decomposition (SVD).** Let A be an m-by-n matrix with $m \geq n$. Then we can write

$$A = U\Sigma V^T,$$

where U is m-by-m orthogonal matrix (i.e. $U^TU = I_m$) and V is n-by-n orthogonal matrix (i.e. $V^TV = I_n$), and $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$.

The columns $u_1, u_2, \ldots, u_n$ of U are called *left singular vectors* of A. The columns $v_1, v_2, \ldots, v_n$ of V are called *right singular vectors*. The $\sigma_1, \sigma_2, \ldots, \sigma_n$ are called *singular values*.

Examples of applications:

- Suppose that A is m-by-n with $m \geq n$ and has full rank, with $A = U\Sigma V^T$ being A's SVD. Then the pseudo-inverse can also be written as

$$A^\dagger \equiv (A^TA)^{-1}A^T = V\Sigma^{-1}U^T.$$

If $m < n$, then $A^\dagger = A^T(AA^T)^{-1}$.

- Suppose that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0,$$

Then the rank of A is r. The range space of A is $\text{span}(u_1, u_2, \cdots, u_r)$. and the null space of A is $\text{span}(v_{r+1}, v_{r+2}, \ldots, v_n)$.

- $\|A\|_2 = \sigma_1 (\equiv \sigma_{\max})$
- Let A be $m \times n$ with $m \geq n$. Then
  (a) eigenvalues of A^TA are σ_i^2, $i = 1, 2, \ldots, n$. The corresponding eigenvectors are the right singular vectors v_i, $i = 1, 2, \ldots, n$.
  (b) eigenvalues of AA^T are σ_i^2, $i = 1, 2, \ldots, n$ and $m - n$ zeros. The left singular vectors u_i, $i = 1, 2, \ldots, n$ are corresponding eigenvectors for the eigenvalues σ_i^2. One can take any $m - n$ other orthogonal vectors that are orthogonal to $u_1, u_2, \ldots, u_n$ as the eigenvectors for the eigenvalues 0.
- Principal components. The SVD of A can be rewritten as

$$A = E_1 + E_2 + \cdots + E_p$$

where $p = \min(m, n)$, and E_k is a rank-one matrix of the form

$$E_k = \sigma_k u_k v_k^T,$$

E_k are referred to as component matrices, and are orthogonal to each other in the sense that

$$E_j E_k^T = 0, \quad j \neq k.$$

Since $\|E_k\|_2 = \sigma_k$, the contribution each E_k makes to reproduce A is determined by the size of the singular value σ_k.

- Optimal rank-k approximation:

$$\min_{\substack{B : m \times n \\ \text{rank}(B) = k}} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1},$$

where

$$A_k = U \Sigma_k V^T, = E_1 + E_2 + \cdots + E_k,$$

and $\Sigma_k = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k, 0, \ldots, 0)$

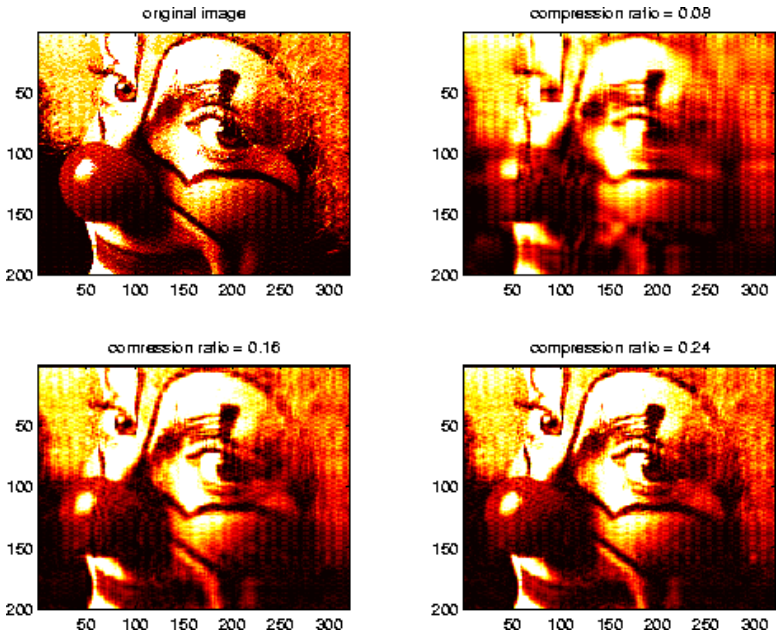
- Data compression. Note that the optimal rank-k approximation A_k can be written in a compact form as

$$A_k = U_k \widehat{\Sigma}_k V_k^T,$$

where U_k and V_k are the first k columns of U and V, respectively, $\widehat{\Sigma}_k = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k)$. Therefore, A_k is represented by $mk + k + nk = (m + n + 1)k$ elements, in contrast, A is represented by mn elements.

$$\text{compression ratio} = \frac{(m + n + 1)k}{mn}$$

The following plots show the original image, and three compressed ones with different compression ratios:



3 4-6-12

3.1 Matrix Factorizations (Continued)

$$\begin{aligned} f(x) = \|Ax-b\|_2^2 &= (Ax-b)^T(Ax-b) \\ &= (x^TA^T - b^T)(Ax-b) \\ &= x^TA^TAx - 2b^TAx + b^Tb \\ \nabla f(x) &= 0 \qquad \text{(condition at the minimum)} \\ \underbrace{A^TA}_{m\times m}x &= \underbrace{A^tb}_{m\times 1} \\ R_1^t \underbrace{Q_1^TQ_1}_{=I} R_1x &= R_1^TQ_1^Tb = \tilde{b} \\ R_1^Ty &= \tilde{b} \\ R_1x &= y \end{aligned}$$

3. Schur decomposition. For any $n \times n$ matrix A, there exist a unitary matrix U and an upper triangular matrix T such that
$$A = UTU^H.$$
Applications:
   - Finding the eigenvalues of A.
$$\begin{aligned} Ax &= \lambda x \\ UTU^Hx &= \lambda x \\ T\underbrace{U^Hx}_{} &= \lambda \underbrace{U^Hx}_{} \\ Ty &= \lambda y \qquad \Rightarrow \qquad \lambda_i = t_{ii} \end{aligned}$$
   - Functions of matrices. For example, e^A, $\sin(A), \ldots$
$$f(A) = f(UTU^H) = Uf(T)U^H \qquad \text{(by property of function definition)}$$

4. Singular Value Decomposition (SVD). "Swiss army knife of scientific computing." For any matrix A, we can write
$$\underbrace{A}_{n\times m} = \underbrace{U}_{n\times n}\underbrace{\Sigma}_{n\times m}\underbrace{V^T}_{m\times m},$$
where Σ is a diagonal matrix whose diagonal entries are the singular values of A: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m \geq 0$. (If A is complex, then we have V^H.) U is an $n \times n$ orthogonal matrix: $U^TU = I_n$. V is an $m \times m$ orthogonal matrix: $V^TV = I_m$. Applications:
   - rank(A) = # of positive (i.e., nonzero) singular values

- Principal component analysis (PCA)

$$
\begin{aligned}
A &= U\Sigma V^T \\
&= \begin{bmatrix} U_1 & U_2 & \cdots & U_n \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_m \\ & & \mathbf{0} & \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \\ \vdots \\ V_m^T \end{bmatrix} \\
&= \begin{bmatrix} \sigma_1 U_1 & \sigma_2 U_2 & \cdots & \sigma_m U_m \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \\ \vdots \\ V_m^T \end{bmatrix} \\
&= \sigma_1 \underbrace{U_1 V_1^T}_{n\times m} + \sigma_2 \underbrace{U_2 V_2^T}_{n\times m} + \cdots + \sigma_m \underbrace{U_m V_m^T}_{n\times m} \\
&= E_1 + E_2 + \cdots + E_m \\
E_i &= \sigma_i U_i V_i^T, \qquad \operatorname{rank}(E_i) = 1, \quad \|E_i\| = \sigma_i
\end{aligned}
$$

  These E_i are known as the *principal components* of the matrix. We have

$$
\begin{aligned}
\|E_1\| \geq \|E_2\| \geq \cdots \geq \|E_m\| \\
\min_{\operatorname{rank}(B)=k} \|A - B\|_{2 \text{ or } \mathrm{F}} = \|A - A_k\|_{2 \text{ or } \mathrm{F}},
\end{aligned}
$$

  where A_k is the sum of the first k principal components of A. This is known as *dimension reduction*.

- Generalized inverse. $A^+ = V\Sigma^+ U^T$. The solution to $\min_x \|Ax - b\|_2$ is given by $x = A^+ b$.

ECS231 Handout **April 9, 2011**

Norms are an indispensable tool to provide vectors and matrices with measures of size, length and distance.

I. Vector norms

1. A *vector norm* on $\mathcal{C}^n$ is a mapping that maps each $x \in \mathcal{C}^n$ to a real number $\|x\|$, satisfying
   (a) $\|x\| > 0$ for $x \neq 0$, and $\|0\| = 0$ (positive definite property)
   (b) $\|\alpha x\| = |\alpha| \, \|x\|$ for $\alpha \in \mathcal{C}$ (absolute homogeneity)
   (c) $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality)
2. Vector p-norm:
$$\|x\|_p \stackrel{\text{def}}{=} \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p},$$
   where $1 \leq p \leq \infty$.
3. Commonly used vector norms:
$$\begin{aligned} \|x\|_1 &= \sum_{i=1}^{n} |x_i|, \quad \text{“Manhattan” or “taxi cab” norm} \\ \|x\|_2 &= \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2} = \sqrt{x^H x}, \quad \text{Euclidean length} \\ \|x\|_\infty &= \max_{1 \leq i \leq n} |x_i|. \end{aligned}$$
4. The geometry of the closed unit “ball”: $\{x \in \mathcal{C}^2 : \|x\|_p \leq 1\}$ for $p = 1, 2, \infty$.
5. Norm equivalence: Let $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ be any two vector norms. There are constants $c_1, c_2 > 0$ such that
$$c_1 \| \cdot \|_\alpha \leq \| \cdot \|_\beta \leq c_2 \| \cdot \|_\alpha$$
   For examples, it can be easily shown that
$$\begin{aligned} \|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \|x\|_\infty \\ \|x\|_2 \leq \|x\|_1 \leq \sqrt{n} \|x\|_2 \\ \|x\|_\infty \leq \|x\|_1 \leq n \|x\|_\infty \end{aligned}$$
6. Cauchy-Schwarz inequality:
$$|x^H y| \leq \|x\|_2 \|y\|_2$$
   with equality if and only if x and y are linearly dependent.
   In general, we have Hölder inequality:
$$|x^H y| \leq \|x\|_p \|y\|_q$$
   for $1 \leq p, q < \infty$ and $\frac{1}{p} + \frac{1}{q} = 1$.

II. Matrix norms

1. A *matrix norm* on $\mathcal{C}^{m\times n}$ is a mapping that maps each $A \in \mathcal{C}^{m\times n}$ to a real number $\|A\|$, satisfying

   (a) $\|A\| > 0$ for $A \neq 0$, and $\|0\| = 0$ (positive definite property)

   (b) $\|\alpha A\| = |\alpha|\,\|A\|$ for $\alpha \in \mathcal{C}$ (absolute homogeneity)

   (c) $\|A + B\| \leq \|A\| + \|B\|$ (triangle inequality)

2. Example: for $A = (a_{ij}) \in \mathcal{C}^{m\times n}$, the Frobenius norm $\|A\|_{\mathrm{F}}$ is defined by

$$\|A\|_{\mathrm{F}} \stackrel{\mathrm{def}}{=} \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2} = \sqrt{\mathrm{tr}(A^H A)}.$$

3. The *induced matrix norm* $\|\cdot\|$:

   A vector norm $\|\cdot\|$ induces a matrix norm, denoted by the same notation:

$$\|A\| \stackrel{\mathrm{def}}{=} \max_{x\neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

   (Exercise. verify that $\|A\|$ is indeed a norm on $\mathcal{C}^{m\times n}$)

4. Useful property: $\|Ax\| \leq \|A\|\,\|x\|$. Therefore, $\|A\|$ is the maximal factor by which A can "strech" a vector.

5. The vector p-norms induce the matrix p-norms, in particular, for $p = 1, 2, \infty$, we have

$$\|A\|_1 = \max_{x\neq 0} \frac{\|Ax\|_1}{\|x\|_1} = \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}| = \text{ max absolute column sum,}$$

$$\|A\|_2 = \max_{x\neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sqrt{\text{the largest eigenvalue of } A^*A} = \text{the largest singular value of } A,$$

$$\|A\|_\infty = \max_{x\neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} = \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}| = \text{ max absolute row sum.}$$

6. Some useful properties:

   - $\|Ax\| \leq \|A\|\,\|x\|$. Therefore, $\|A\|$ is the maximal factor by which A can "strech" a vector.
   - $\|A\|_2^2 \leq \|A\|_1 \|A\|_\infty$.
   - Norm equivalence

5 4-11-12

5.1 IEEE Floating Point Arithmetic

The floating point representation of a nonzero binary number x is

$$x = \pm b_0.b_1 b_2 \cdots b_{p-1} \times 2^E.$$

(a) It is *normalized*, i.e. $b_0 = 1$ (the *hidden bit*)

(b) *Precision* $(= p)$ is the number of bits in the significand (mantissa), including the hidden bit.

(c) *Machine epsilon* $\epsilon = 2^{-(p-1)}$, the gap between the number 1 and the smallest floating point number that is greater than 1.

Special numbers: $0, \infty, -\infty,$ `NaN`.

ECS231 Handout **April 11, 2012**

Part I: Floating-point numbers and representations

1. Floating-point representation of numbers (scientific notation), for example,

$$\underset{\text{sign}}{-} \quad \underset{\text{significand}}{3.1416} \times \underset{\text{base}}{10}^{\underset{\text{exponent}}{1}}$$

2. The floating-point representation of a nonzero **binary** number x is of the form

$$x = \pm\, b_0.b_1b_2\cdots b_{p-1} \times 2^E. \tag{1}$$

   (a) It is *normalized*, i.e., $b_0 = 1$ (the hidden bit)

   (b) *Precision* ($= p$) is the number of bits in the significand (mantissa) (including the hidden bit).

   (c) *Machine epsilon* $\epsilon = 2^{-(p-1)}$, the gap between the number 1 and the smallest floating-point number that is greater than 1.

   (d) The *unit in the last place*, $\text{ulp}(x) = 2^{-(p-1)} \times 2^E = \epsilon \times 2^E$. If $x > 0$, then $\text{ulp}(x)$ is the gap between x and the next larger floating-point number. If $x < 0$, then $\text{ulp}(x)$ is the gap between x and the smaller floating-point number (larger in absolute value).

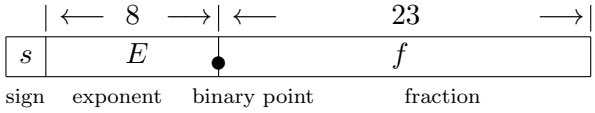
3. Special numbers: 0, -0, ∞, $-\infty$, NaN = "Not a Number".

4. All computers designed since 1985 use the ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic, represent each number as a binary number and use binary arithmetic.

   Essentials of IEEE 754-1985:

   - consistent representation of floating-point numbers by all machines adopting the standard;
   - correctly rounded floating-point operations, using various rounding modes;
   - consistent treatment of exceptional situation such as division by zero.

5. IEEE **single** format takes 32 bits long (= 4 bytes):



   It represents the number

$$(-1)^s \cdot (1.f) \times 2^{E-127}$$

   Note that the leading 1 in the fraction need not be stored explicitly, because it is always 1. This hidden bit accounts for the "1." here. The "$E - 127$" in the exponent is to avoid the need for storage of a sign bit.

   E is a normalized number, and $E_{\min} = (00000001)_2 = (1)_{10}$, $E_{\max} = (11111110)_2 = (254)_{10}$.

The range of positive normalized numbers is from

$$N_{\min} = 1.00\cdots0 \times 2^{E_{\min}-127} = 2^{-126} \approx 1.2 \times 10^{-38}$$

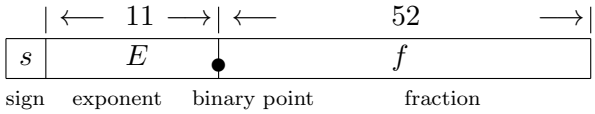
to

$$N_{\max} = 1.11\cdots1 \times 2^{E_{\max}-127} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}.$$

Special repsentations for 0, $\pm\infty$ and NaN:

zero	=	$\pm$	00000000	00000000000000000000000
$\pm\infty$	=	$\pm$	11111111	00000000000000000000000
NaN	=	$\pm$	11111111	otherwise

6. IEEE **double** format takes 64 bits long (=8 bytes):



It represents the numer

$$(-1)^s \cdot (1.f) \times 2^{E-1023}$$

The range of positive normalized numbers is from

$$N_{\min} = 2^{-1022} \approx 2.2 \times 10^{-308}$$

to

$$N_{\max} = 1.11\cdots1 \times 2^{1023} \approx 2^{1024} \approx 1.8 \times 10^{308}.$$

Special repsentations for 0, $\pm\infty$ and NaN.

7. IEEE **extended** format, with at least 15 bits available for the exponent and at least 63 bits for the fractional part of the significant. (Pentium has 80-bit extended format)

8. Precision and machine epsilon of the IEEE formats

Format	Precision p	Machine epsilon $\epsilon = 2^{-p-1}$
single	24	$\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$
double	53	$\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$
extended	64	$\epsilon = 2^{-63} \approx 1.1 \times 10^{-19}$

9. Rounding

Let a positive real number x is in the normalized range, i.e., $N_{\min} \le x \le N_{\max}$, and write in the normalized form

$$x = (1.b_1b_2\cdots b_{p-1}b_pb_{p+1}\ldots) \times 2^E,$$

Then the closest floating-point number less than or equal to x is

$$x_- = 1.b_1b_2\cdots b_{p-1} \times 2^E,$$

i.e., x_- is obtained by *truncating*. The next floating-point number bigger than x_- is

$$x_+ = ((1.b_1b_2\cdots b_{p-1}) + (0.00\cdots 01)) \times 2^E,$$

therefore, also the next one that bigger than x.

If x is negative, the situtation is reversed.

Correctly rounding modes:

- *round down*: $\text{round}(x) = x_-$;
- *round up*: $\text{round}(x) = x_+$;
- *round towards zero*: $\text{round}(x) = x_-$ of $x \geq 0$; $\text{round}(x) = x_+$ of $x \leq 0$;
- *round to nearest*: $\text{round}(x) = x_-$ or x_+, whichever is nearer to x, except that if $x > N_{\max}$, $\text{round}(x) = \infty$, and if $x < -N_{\max}$, $\text{round}(x) = -\infty$. In the case of tie, i.e., x_- and x_+ are the same distance from x, the one with its least significant bit equal to zero is chosen

When the *round to nearest* (IEEE default rounding mode) is in effect,

$$\text{abserr}(x) = |\text{round}(x) - x| \leq \frac{1}{2}\text{ulp}(x)$$

and

$$\text{relerr}(x) = \frac{|\text{round}(x) - x|}{|x|} \leq \frac{1}{2}\epsilon.$$

Therefore, we have

$$\text{the max. rel. representation error} = \begin{cases} \frac{1}{2} \cdot 2^{1-24} = 2^{-24} \approx 5.96 \cdot 10^{-8} \\ \\ \frac{1}{2} \cdot 2^{-52} \approx 1.11 \times 10^{-16}. \end{cases}$$

Part II: Floating point arithmetic

1. IEEE rules for correctly rounded floating-point operations:

   if x and y are correctly rounded floating-point numbers, then

   $$\begin{aligned} \text{fl}(x+y) &= \text{round}(x+y) = (x+y)(1+\delta) \\ \text{fl}(x-y) &= \text{round}(x-y) = (x-y)(1+\delta) \\ \text{fl}(x\times y) &= \text{round}(x\times y) = (x\times y)(1+\delta) \\ \text{fl}(x/y) &= \text{round}(x/y) = (x/y)(1+\delta) \end{aligned}$$

   where $|\delta| \leq \frac{1}{2}\epsilon$ for the *round to nearest*,

   IEEE standard also requires that correctly rounded remainder and square root operations be provided.

2. IEEE standard response to exceptions

Event	Example	Set result to
Invalid operation	$0/0$, $0 \times \infty$	NaN
Division by zero	Finite nonzero/0	$\pm\infty$
Overflow	$\|x\| > N_{\max}$	$\pm\infty$ or $\pm N_{\max}$
underflow	$x \neq 0, \|x\| < N_{\min}$	± 0, $\pm N_{\min}$ or subnormal
Inexact	whenever $\mathrm{fl}(x \circ y) \neq x \circ y$	correctly rounded value

3. Let $\hat{x}$ and $\hat{y}$ be the floating-point numbers and that

$$\hat{x} = x(1+\tau_1) \quad \text{and} \quad \hat{y} = y(1+\tau_2), \quad \text{for } |\tau_i| \leq \tau \ll 1$$

where τ_i could be the relative errors in the process of "collecting/getting" the data from the original source or the previous operations.

Question: how do the four basic arithmetic operations behave?

4. Addition and subtraction

$$\begin{aligned}
\mathrm{fl}(\hat{x}+\hat{y}) &= (\hat{x}+\hat{y})(1+\delta), \qquad |\delta| \leq \frac{1}{2}\epsilon \\
&= x(1+\tau_1)(1+\delta) + y(1+\tau_2)(1+\delta) \\
&= x + y + x(\tau_1 + \delta + O(\tau\epsilon)) + y(\tau_2 + \delta + O(\tau\epsilon)) \\
&= (x+y)\left(1 + \frac{x}{x+y}(\tau_1 + \delta + O(\tau\epsilon)) + \frac{y}{x+y}(\tau_2 + \delta + O(\tau\epsilon))\right) \\
&\equiv (x+y)(1+\hat{\delta}),
\end{aligned}$$

where $\hat{\delta}$ can be bounded as follows:

$$|\hat{\delta}| \leq \frac{|x|+|y|}{|x+y|}\left(\tau + \frac{1}{2}\epsilon + O(\tau\epsilon)\right).$$

Three possible cases:

- If x and y have the same sign, i.e., $xy > 0$, then $|x+y| = |x| + |y|$; this implies

$$|\hat{\delta}| \leq \tau + \frac{1}{2}\epsilon + O(\tau\epsilon) \ll 1.$$

  Thus $\mathrm{fl}(\hat{x}+\hat{y})$ approximates $x+y$ well.
- If $x \approx -y \Rightarrow |x+y| \approx 0$, then $(|x|+|y|)/|x+y| \gg 1$; this implies that $|\hat{\delta}|$ could be nearly or much bigger than 1. Thus $\mathrm{fl}(\hat{x}+\hat{y})$ may turn out to have nothing to do with the true $x+y$. This is so called *catastrophic cancellation* which happens when a floating-point number is subtracted from another nearly equal floating-point number. Cancellation causes relative errors or uncertainties already presented in $\hat{x}$ and $\hat{y}$ to be magnified.
- In general, if $(|x|+|y|)/|x+y|$ is not too big, $\mathrm{fl}(\hat{x}+\hat{y})$ provides a good approximation to $x+y$.

5. Multiplication and Division are very well-behaved.

$$\begin{aligned}
\mathrm{fl}(\hat{x} * \hat{y}) &= (\hat{x} \times \hat{y})(1+\delta) = xy(1+\tau_1)(1+\tau_2)(1+\delta) \equiv xy(1+\hat{\delta}_\times), \\
\mathrm{fl}(\hat{x}/\hat{y}) &= (\hat{x}/\hat{y})(1+\delta) = (x/y)(1+\tau_1)(1+\tau_2)^{-1}(1+\delta) \equiv xy(1+\hat{\delta}_\div),
\end{aligned}$$

where

$$\hat{\delta}_\times = \tau_1 + \tau_2 + \delta + O(\tau\epsilon), \quad \hat{\delta}_\div = \tau_1 - \tau_2 + \delta + O(\tau\epsilon).$$

Thus $|\hat{\delta}_\times| \leq 2\tau + \frac{1}{2}\epsilon + O(\tau\epsilon)$ and $|\hat{\delta}_\div| \leq 2\tau + \frac{1}{2}\epsilon + O(\tau\epsilon)$.

6. Examples of catastrophic cancellation

EXAMPLE 1. Computing $\sqrt{n+1}-\sqrt{n}$ straightforward causes substantial loss of significant digits for large n

n	$\mathrm{fl}(\sqrt{n+1})$	$\mathrm{fl}(\sqrt{n})$	$\mathrm{fl}(\mathrm{fl}(\sqrt{n+1})-\mathrm{fl}(\sqrt{n})$
1.00e+10	1.00000000004999994e+05	1.00000000000000000e+05	4.99999441672116518e-06
1.00e+11	3.16227766018419061e+05	3.16227766016837908e+05	1.58115290105342865e-06
1.00e+12	1.00000000000050000e+06	1.00000000000000000e+06	5.00003807246685028e-07
1.00e+13	3.16227766016853740e+06	3.16227766016837955e+06	1.57859176397323608e-07
1.00e+14	1.00000000000000503e+07	1.00000000000000000e+07	5.02914190292358398e-08
1.00e+15	3.16227766016838104e+07	3.16227766016837917e+07	1.86264514923095703e-08
1.00e+16	1.00000000000000000e+08	1.00000000000000000e+08	0.00000000000000000e+00

Catastrophic cancellation can sometimes be avoided if a formula is properly reformulated. In the present case, one can compute $\sqrt{n+1}-\sqrt{n}$ almost to full precision by using the equality

$$\sqrt{n+1}-\sqrt{n}=\frac{1}{\sqrt{n+1}+\sqrt{n}}.$$

Consequently, the computed results are

n	$\mathrm{fl}(1/(\sqrt{n+1}+\sqrt{n}))$
1.00e+10	4.999999999875000e-06
1.00e+11	1.581138830080237e-06
1.00e+12	4.999999999998749e-07
1.00e+13	1.581138830084150e-07
1.00e+14	4.999999999999987e-08
1.00e+15	1.581138830084189e-08
1.00e+16	5.000000000000000e-09

In fact, one can show that $\mathrm{fl}(1/(\sqrt{n+1}+\sqrt{n}))=(\sqrt{n+1}-\sqrt{n})(1+\delta)$, where $|\delta|\le 5\epsilon+O(\epsilon^2)$ (try it!)

EXAMPLE 2. Consider the function

$$f(x)=\frac{1-\cos x}{x^2}=\frac{1}{2}\left(\frac{\sin(x/2)}{x/2}\right)^2.$$

Note that

$$0\le f(x)<1/2 \quad \text{for all } x\neq 0.$$

Compare the computed values for $x=1.2\times 10^{-5}$ using the above two expressions (assume that the value of $\cos x$ rounded to 10 significant figures).

Part III: Floating point error analysis

1. Forward and backward error analysis

We illustrate the basic idea through a simple example. Consider the computation of an inner product of two vector $x,\ y\in\mathcal{R}^3$

$$x^T y \stackrel{\text{def}}{=} x_1y_1+x_2y_2+x_3y_3,$$

assuming already x_i's and y_j's are floating-point numbers. It is likely that $\mathrm{fl}(x \cdot y)$ is computed in the following order.

$$\mathrm{fl}(x^T y) = \mathrm{fl}(\ \mathrm{fl}(\mathrm{fl}(x_1 y_1) + \mathrm{fl}(x_2 y_2)) + \mathrm{fl}(x_3 y_3)\ ).$$

Adopting the floating-point arithmetic model, we have

$$\begin{aligned}
\mathrm{fl}(x^T y) &= \mathrm{fl}(\ \mathrm{fl}(x_1 y_1(1+\epsilon_1) + x_2 y_2(1+\epsilon_2)) + x_3 y_3(1+\epsilon_3)\ ) \\
&= \mathrm{fl}(\ (x_1 y_1(1+\epsilon_1) + x_2 y_2(1+\epsilon_2))(1+\delta_1) + x_3 y_3(1+\epsilon_3)\ ) \\
&= (\ (x_1 y_1(1+\epsilon_1) + x_2 y_2(1+\epsilon_2))(1+\delta_1) + x_3 y_3(1+\epsilon_3)\ )(1+\delta_2) \\
&= x_1 y_1(1+\epsilon_1)(1+\delta_1)(1+\delta_2) + x_2 y_2(1+\epsilon_2)(1+\delta_1)(1+\delta_2) \\
&\quad + x_3 y_3(1+\epsilon_3)(1+\delta_2),
\end{aligned}$$

where $|\epsilon_i| \le \frac{1}{2}\epsilon$ and $|\delta_j| \le \frac{1}{2}\epsilon$.

Now there are two ways to interpret the errors in the computed $\mathrm{fl}(x^T y)$:

(a) We have

$$\mathrm{fl}(x^T y) = x^T y + E,$$

where $E = x_1 y_1(\epsilon_1 + \delta_1 + \delta_2) + x_2 y_2(\epsilon_2 + \delta_1 + \delta_2) + x_3 y_3(\epsilon_3 + \delta_2) + O(\epsilon^2)$. It implies that

$$|E| \le \frac{1}{2}\epsilon(3|x_1 y_1| + 3|x_2 y_2| + 2|x_3 y_3|) + O(\epsilon^2) \le \frac{3}{2}\epsilon \cdot |x|^T |y| + O(\epsilon^2).$$

This bound on E tells the worst case difference between the exact $x^T y$ and its computed value. Such an error analysis is so-called *Forward Error Analysis.*

(b) We can also write

$$\mathrm{fl}(x^T y) = \hat{x}^T \hat{y} = (x + \Delta x)^T (y + \Delta y),$$

where[1]

$$\begin{array}{lllllll}
\hat{x}_1 = x_1(1+\epsilon_1), & & \hat{y}_1 = y_1(1+\delta_1)(1+\delta_2) & \equiv & y_1(1+\hat{\delta}_1), \\
\hat{x}_2 = x_2(1+\epsilon_2), & & \hat{y}_2 = y_2(1+\delta_1)(1+\delta_2) & \equiv & y_2(1+\hat{\delta}_2), \\
\hat{x}_3 = x_3(1+\epsilon_3), & & \hat{y}_3 = y_3(1+\delta_2) & \equiv & y_3(1+\hat{\delta}_3).
\end{array}$$

It can be seen that $|\hat{\delta}_1| = |\hat{\delta}_2| \le \epsilon + O(\epsilon^2)$ and $|\hat{\delta}_3| \le \frac{1}{2}\epsilon$. This says the computed value $\mathrm{fl}(x^T y)$ is the *exact* inner product of a slightly perturbed $\hat{x}$ and $\hat{y}$. Such an error analysis is so-called *Backward Error Analysis.*

Part IV: Further reading

1. The following article based on lecture notes of Prof. W. Kahan of the University of California at Berkeley provides an excellent review of IEEE float point arithmetics.

   D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 18(1):5–48, 1991.

[1] There are many ways to distribute factors $(1+\epsilon_i)$ and $(1+\delta_j)$ to x_i and y_j. In this case it is even possible to make either $\hat{x} \equiv x$ or $\hat{y} \equiv y$.

2. The following book gives a broad overview of numerical computing, with special focus on the IEEE standard for binary floating-point arithmetic.

   M. Overton. Numerical computing with IEEE floating-point arithemetic. SIAM, Philadelphia, 2001. ISBN 0-89871-482-6. Student price $20.00 directly from `www.siam.org`.

3. A classical book on error analysis, where the notion of backward error analysis is invented, is

   J.H. Wilkinson. Rounding Errors in Algebraic Process. Prentice-Hall, Englewood, NJ, 1964. Reprinted by Dover, New York, 1994.

4. A contemporary treatment of error analysis and its applications to numerical analysis is

   N.J. Higham, Accuracy and stability of Numerical Algorithms. second edition, SIAM, Philadelphia, 2002.

5. Websites for discussion of numerical disasters:

   - D. Arnold, Some disasters attributable to bad numerical computing
     http://www.ima.umn.edu/∼arnold/disasters/disasters.html
   - K. Vuik, Some disasters caused by numerical errors
     http://ta.twi.tudelft.nl/nw/users/vuik/wi211/disasters.html
   - T. Huckle, Collection of software bugs
     http://www5.in.tum.de/∼huckle/bugse.html

6 4-18-12

4 types of multiplications:

1. ~~scalar-scalar~~
2. **Level 1 BLAS:** vector-vector, $x + y \to z$
3. **Level 2 BLAS:** matrix-vector, $Ax \to y$
4. **Level 3 BLAS:** matrix-matrix, $AB \to C$

7 4-20-12

7.1 Segment 1 Recap

1. **Basic concepts and terminology.** Scientific computing vs. computational science: these are different. We look at the forward and backward error here, and relate that to stability and conditioning.

2. **Frequently used matrix decompositions.** The idea is to reduce to a subspace that we can project our problems onto; these are smaller problems with the same or similar structure. The purpose of this is to refamiliarize ourselves with matrix/vector multiplication.

   We have four decompositions:

   (a) LU-Factorization: $PA = LU$. This certainly works for square nonsingular matrices, but we can extend this to singular or non-square matrices.

   (b) $A = QR$, where Q is orthogonal and R is upper triangular. Here we don't place any requirements on the structure of A; this is naturally defined for non-square or singular matrices.

   (c) Schur decomposition: $A = UTU^H$. Here A is again square and (I think) nonsingular. T is upper triangular, with the eigenvalues of A on the diagonal. U is unitary/orthogonal.

   (d) Principle value decomposition: $A = U\Sigma V^T$, where U, V are orthogonal and Σ is diagonal (though not necessarily square), with elements $(\sigma_1, \ldots, \sigma_m)$ such that $\sigma_1 \geq \ldots \geq \sigma_m \geq 0$. Note here that A need not be square or nonsingular.

3. **Vector and Matrix Norms.**

4. **Accuracy/Speed.**

   (a) Floating-point representation

   (b) Floating-point arithmetic. The big point here is catastrophic cancellation. Know this!

   (c) Floating-point error analysis. (We did not cover this.)

5. Block matrix multiplication using BLAS. We use blocking/tiling, which leads to high-performance computing. Later on we'll talk about multicore and GPU implementations.

8 4-23-12

8.1 Large Scale Linear Systems

$$\underbrace{A}_{n\times n}\underbrace{x}_{n\times 1} = \underbrace{b}_{n\times 1}$$

1. n is large: $n = 10^4,\ 10^5$, unknown
2. A is sparse, either in terms of its elements or its data (rank and/or structure)
3. "Matrix-free" $\Rightarrow$ A is a black box. We don't know how it is generated, we just know its action: $u \xrightarrow{A} v = Au$.

8.2 Subspace Projection Methods

"Dimension Reduction." Consider a problem $Ax = b$, where $x \in \mathbb{R}^n$. Let $\mathcal{V} \subseteq \mathbb{R}^n$ be a subspace. Given an initial guess x_0, generate

$$x \approx \tilde{x} = x_0 + v, \qquad v \in \mathcal{V},$$

subject to $b - A\tilde{x} \perp \mathcal{V}$ (Galerkin condition).

0. Select/given x_0
1. Pick (compute) $\underbrace{V_k}_{n\times k},\ W_k$
2. "Form" $\underbrace{A_k}_{k\times k} = W_k^T A V_k,\ b_k = W_k^T b$
3. Solve $A_k z = b_k \leftarrow$ solvable?
4. "Form" $\tilde{x} = x_0 + V_k z$
5. Iterate if necessary

ECS231 Handout **April 23, 2012**

1. The landscape of solvers for linear systems of equations

$$Ax = b,$$

where A is an $n \times n$ matrix and b is an n-vector, $x \in \mathcal{R}^n$ is the unknown.

more robust $\leftarrow - - - - \rightarrow$ less storage

	Direct	Iterative ($u = Av$)
Nonsymmetric A	LU	GMRES
Symmetric positive definite A	Cholesky	CG

more general
$\uparrow$
$\downarrow$
more robust

2. Subspace projection methods: framework

The basic idea of subspace projection technique is to extract an approximate solution $\widetilde{x}$ from a subspace of $\mathcal{R}^n$. It is a technique of *dimension reduction*.

Let $\mathcal{W}$ and $\mathcal{V}$ be two m-dimensional subspaces of $\mathcal{R}^n$, and x_0 is an initial guess of the solution, then the *subspace projection technique* is to

$$\text{find } \widetilde{x} \in x_0 + \mathcal{W} \text{ such that } b - A\widetilde{x} \perp \mathcal{V}. \tag{1}$$

Write $\widetilde{x} = x_0 + z$, $z \in \mathcal{W}$ and define initial residual $r_0 = b - Ax_0$. Notice that $b - A\widetilde{x} = b - A(x_0 + z) = r_0 - Az$. Then the formulation (1) is equivalent to

$$\text{find } z \in \mathcal{W} \text{ such that } r_0 - Az \perp \mathcal{V}. \tag{1a}$$

If $\mathcal{W} = \mathcal{V}$, then it is called an *orthogonal projection* method and the corresponding orthogonality constraints in (1a) is known as the Galerkin condition. Otherwise, if $\mathcal{W} \neq \mathcal{V}$, it is called an *oblique projection* method and the corresponding orthogonality constraints in (1a) is known as the Petrov-Galerkin condition.

Remark: What we described is a basic one projection step. Most implementations use a succession of such projections. Typically, a new projection step uses a "new" pair of subspaces $\mathcal{W}$ and $\mathcal{V}$ (updated from the previous step) and an initial guess x_0 equal to the most recent approximation. This leads to an iterative (refinement) procedure, and is a common approach in numerical computing.

3. In matrix notation, let $V = [v_1, v_2, \ldots, v_m]$ be an $n \times m$ matrix whose columns form a basis of $\mathcal{V}$, and similarly $W = [w_1, w_2, \ldots, w_m]$ an $n \times m$ matrix whose columns form a basis of $\mathcal{W}$. Then any approximation solutions in $x_0 + \mathcal{W}$ can be written as

$$\widetilde{x} = x_0 + z = x_0 + Wy, \quad \text{i.e.,} \quad z = Wy,$$

and the orthogonality condition (1a) implies

$$V^T(r_0 - Az) = 0.$$

Thus we have

$$V^T AWy = V^T r_0$$

Consequently,

$$y = (V^T AW)^{-1} V^T r_0,$$

provided $V^T AW$ is invertible. Putting it all together, we have

$$\widetilde{x} = x_0 + W(V^T AW)^{-1} V^T r_0.$$

4. Now, we have a *prototype subspace projection technique*:

    0. Let x_0 be an initial approximation
    1. Iterate until convergence:
    2. Select a pair of subspaces $\mathcal{V}$ and $\mathcal{W}$
    3. Generate basis matrices V and W for $\mathcal{V}$ and $\mathcal{W}$
    4. $r_0 \leftarrow b - Ax_0$
    5. $y \leftarrow (V^T AW)^{-1} V^T r_0$
    6. $x_0 \leftarrow x_0 + Wy$

    Two remarks are in order:

    1. In many practical algorithms, the matrix $V^T AW$ does not have to be formed explicitly. It is available as a by-product of Steps 2 and 3.
    2. The method is defined only when $V^T AW$ is nonsingular, which is not guaranteed to be true even when A is nonsingular. There are two important special cases where the nonsingularity of $V^T AW$ is guaranteed:
        (a) If A is symmetric positive definite (SPD) and $\mathcal{W} = \mathcal{V}$, then $V^T AW = W^T AW$ is nonsingular.
        (b) If A is nonsingular, and $\mathcal{V} = A\mathcal{W}$, then then $V^T AW = W^T A^T AW$ is nonsingular.

5. A one-dimensional subspace projection process is defined when

$$\mathcal{W} = \text{span}\{w\} \quad \text{and} \quad \mathcal{V} = \text{span}\{v\},$$

where w and v are two vectors. In this case, the new approximation takes form

$$x_0 \leftarrow x_0 + z = x_0 + \alpha w$$

and the orthogonality condition (1a) implies $v^T(r_0 - Az) = v^T(r_0 - \alpha Aw) = 0$, and thus

$$\alpha = \frac{v^T r_0}{v^T Aw}.$$

6. Steepest Descent (SD) method

   When A is SPD, at each step, let

$$v = w = r_0 = b - Ax_0$$

   This yields

   1. Pick an initial guess x_0
   2. For $k = 0, 1, 2, \ldots$ until convergence do
   3. $\quad r_k = b - Ax_k$
   4. $\quad \alpha_k = \frac{r_k^T r_k}{r_k^T A r_k}$
   5. $\quad x_{k+1} = x_k + \alpha_k r_k$

   Remarks: (1) Since A is SPD, $r_k^T A r_k > 0$ except $r_k = 0$. Therefore, SD does not breakdown.

   (2) We can view that each step of the SD iteration minimizes

$$f(x) \stackrel{\text{def}}{=} \frac{1}{2}\|x_* - x\|_A^2 = \frac{1}{2}(x_* - x)^T A(x_* - x),$$

   over all vectors of the form $x - \alpha(\nabla f(x))$, where $\nabla f(x) = b - Ax$ is the gradient of f at x. Recall that from the Calculus, we learned that the negative of the gradient direction is locally the direction that yields the fastest rate of decrease for f.

7. Minimal Residual (MR) Iteration.

   For a general nonsingular matrix A, at each step, let

$$w = r_0 \quad \text{and} \quad v = Ar_0,$$

   It yields

   1. Pick an initial guess x_0
   2. For $k = 0, 1, 2, \ldots$ until convergence do
   3. $\quad r_k = b - Ax_k$
   4. $\quad \alpha_k = \frac{r_k^T A r_k}{r_k^T A^T A r_k}$
   5. $\quad x_{k+1} = x_k + \alpha_k r_k$

   Remark: each step of the MR iteration minimizes

$$f(x) \stackrel{\text{def}}{=} \|r\|_2^2 = \|b - Ax\|_2^2$$

   over all vectors of the form $x - \alpha r$, i.e., solve the least squares problem $\min_x \|b - Ax\|_2$

Further reading

1. Optimality for orthogonal projection. Assume that A is SPD and that $\mathcal{V} = \mathcal{W}$. Then a vector $\widetilde{x}$ is the result of (1) if and only if

$$\|x_* - \widetilde{x}\|_A = \min_{x \in x_0 + \mathcal{W}} \|x_* - x\|_A,$$

where $\|x_* - x\|_A = \sqrt{(x_* - x)^T A(x_* - x)}$, and x_* is the exact solution to $Ax = b$.

PROOF: Notice that $(A(\cdot), \cdot)$ is an inner product on $\mathcal{R}^n$. Thus $\|x_* - x\|_A$ over all possible $x \in x_0 + \mathcal{W}$ is minimized at $\widetilde{x}$ if and only if $x_* - \widetilde{x} \perp_A \mathcal{W}$, i.e.,

$$(A(x_* - \widetilde{x}), w) = (b - A\widetilde{x}, w) = 0 \quad \text{for any } w \in \mathcal{W} = \mathcal{V}.$$

This is (1). ∎

Remark: The *steepest descent (SD) method* and *conjugate gradient (CG) method* are the corresponding implementations for solving large scale symmetric definite linear system of equations.

2. Optimality for oblique projection. Let A be an arbitrary square matrix and assume $\mathcal{V} = A\mathcal{W}$. Then a vector $\widetilde{x}$ is the result of (1) if and only if

$$\|b - A\widetilde{x}\|_2 = \min_{x \in x_0 + \mathcal{W}} \|b - Ax\|_2.$$

PROOF: $\|b - Ax\|_2$ over all possible $x \in x_0 + \mathcal{W}$ is minimized at $\widetilde{x}$ if and only if $b - A\widetilde{x} \perp A\mathcal{W}$, i.e.,

$$(b - A\widetilde{x}, v) = 0 \quad \text{for any } v \in A\mathcal{W} = \mathcal{V}.$$

This is (1). ∎

Remark: The *minimal residual residual (MR) method* and *generalized minimal residual (GMRES) method* are the corresponding implementations for solving large scale nonsymmetric linear systems of equations.

3. Convergence theorem of SD algorithm: Let A be SPD, and let $\lambda_{\min}$ and $\lambda_{\max}$ be its smallest and largest eigenvalues respectively. Then for the SD ALGORITHM

$$\|x_* - x_{k+1}\|_A \le \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}}\right) \|x_* - x_k\|_A = \left(\frac{\kappa(A) - 1}{\kappa(A) + 1}\right) \|x_* - x_k\|_A,$$

where x_* is the exact solution to $Ax = b$. $\kappa(A) = \lambda_{\max}/\lambda_{\min}$ is the condition number of A.

For a proof, see [Y. Saad, Iterative methods for sparse linear systems, Second Edition, SIAM, 2003]

Remark: The SD converges for any initial guess. However, if $\kappa(A)$ is large and $\frac{\kappa(A)-1}{\kappa(A)+1} \approx 1$, the convergence could be extremely slow. The simple SD becomes impractical.

4. Convergence theorem of MR algorithm: Assume that $A + A^T$ is *SPD*,[1], and let $\mu = \lambda_{\min}\left(\frac{A+A^T}{2}\right)$, and $\sigma = \|A\|_2$. Then for the MR iteration

$$\|r_{k+1}\|_2 \le \left(1 - \frac{\mu^2}{\sigma^2}\right)^{1/2} \|r_k\|_2.$$

For a proof, see [Y. Saad, Iterative methods for sparse linear systems, Second Edition, SIAM, 2003]

Remark: For any positive definite (not necessarily symmetric) linear systems, the MR iteration converges for any initial guess. However, if $\frac{\mu}{\sigma} \approx 0$, the convergence becomes extremely slow and the MR is not a practical method.

[1]This is equivalent to say that A is positive definite. A real matrix A said to be positive definite if $u^T Au > 0$ for any $0 \neq u \in \mathcal{R}$. It can be shown that if A is real positive definite, then A is nonsingular, in addition, $u^T Au \ge \lambda_{\min}\left(\frac{1}{2}(A + A^T)\right) u^T u$.

9 4-25-12

9.1 Large Scale Linear Systems

Dimension reduction:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ \vdots \\ x_n \end{bmatrix} \approx \begin{bmatrix} \\ V_m \\ \\ \\ \\ \end{bmatrix} \begin{bmatrix} \\ z \\ \\ \end{bmatrix} = \tilde{x}$$

We want

1. V_m to be a "high-quality" base
2. the $\approx$ to be "optimal"
    - by subspace projection:

$$\begin{aligned} b - A\tilde{x} &\perp V_m \\ V_m^T(b - A\tilde{x}) &= 0 \\ V_m^t A V_m z &= V_m^T b \end{aligned}$$

9.1.1 One-Dimensional Situation

Let $m = 1$.

$$\begin{aligned} x &\approx \begin{bmatrix} \\ V_1 \\ \\ \end{bmatrix} z x_0 \\ x_1 &= x_0 + v_1 z \\ (V_1^t A V_1) z &= V_1^T b \\ Z &= \frac{V_1^T b}{V_1^T A V_1}, \qquad V_1^T A V_1 \neq 0 \text{ (no breakdown)} \end{aligned}$$

	$A^T = A > 0$	A general
1-D subspace projection	Steepest Descent (SD)	Minimal Residual (MR)
m-D subspace projection	Conjugate Gradient (CG)	GMRES

ECS231 Handout **April 25, 2012**

1. *Krylov subspace* is defined as

$$\mathcal{K}_m(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{m-1}v\},$$

where A is an $n \times n$ matrix, and v is a column vector of length n.

Note that if $x \in \mathcal{K}_m(A, v)$, then $x = p(A)v$, where $p(A)$ is a polynomial of degree not exceeding $m - 1$.

2. *Arnoldi procedure* is an algorithm for building an orthogonal basis $\{v_1, v_2, \ldots, v_m\}$ of the Krylov subspace $\mathcal{K}_m(A, v)$ using a modified Gram-Schmidt orthogonalization process.[1]

$[V_{m+1}, \widehat{H}_m] = \text{arnoldi}(A, v, m)$
1. $v_1 = v/\|v\|_2$
2. for $j = 1, 2, \ldots, m$
3. compute $w = Av_j$
4. for $i = 1, 2, \ldots, j$
5. $h_{ij} = v_i^T w$
6. $w := w - h_{ij}v_i$
7. end for
8. $h_{j+1,j} = \|w\|_2$
9. If $h_{j+1,j} = 0$, *stop*
10. $v_{j+1} = w_j/h_{j+1,j}$
11. endfor

Proposition 1 *Assume that the Arnoldi procedure does not stop before the m-th step. Then the vectors* $\{v_1, v_2, \ldots, v_m\}$ *form an orthonormal basis of the Krylov subspace* $\mathcal{K}_m(A, v)$*:*

$$span\{v_1, v_2, \ldots, v_m\} = \mathcal{K}_m(A, v).$$

3. Arnoldi decomposition. Let

$$V_m = [v_1, v_2, \ldots, v_m] \quad \text{and} \quad H_m = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1,m-1} & h_{1m} \\ h_{21} & h_{22} & \cdots & h_{2,m-1} & h_{2m} \\ & h_{32} & \ddots & h_{3,m-1} & h_{3m} \\ & & \ddots & \vdots & \vdots \\ & & & h_{m,m-1} & h_{m,m} \end{bmatrix},$$

where H_m is called an upper Hessenberg matrix, then in the matrix form, the Arnoldi procedure can be expressed in the following governing relations:

$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T$$

[1] Alternatively, we can also view the Arnoldi procedure as a partial implementation for computing a Hessenberg decomposition of A. The Hessenberg decompositoin is defined as the following: for any real square matrix A, there exists an orthogonal matrix V (i.e., $V^TV = I$), such that $A = VHV^T$, where H is an upper Hessenberg matrix, $H = (h_{ij})$ with $h_{ij} = 0$ for $i > j + 1$.

and $V_m^T V_m = I_m$ and $V_m^T A V_m = H_m$. This is referred to as an *order-m Arnoldi decomposition.* If we denote

$$V_{m+1} = [V_m, v_{m+1}] \quad \textit{and} \quad \widehat{H}_m = \begin{bmatrix} H_m \\ h_{m+1,m} e_m^T \end{bmatrix},$$

where $\widehat{H}_m$ is a $m+1$ by m upper triangular matrix, then an order-m Arnoldi decomposition can also be written in the following compact form

$$AV_m = V_{m+1}\widehat{H}_m.$$

4. *Remarks:*

   - Note that the matrix A is only referenced via the matrix-vector multiplication Av_j. Therefore, it is ideal for large sparse or dense structure matrices. Any sparsity or structure of a matrix can be exploited in the matrix-vector multiplication.
   - The main storage requirement is $n(m+1)$ for storing Arnoldi vectors $\{v_i\}$ plus the storage requirement for the matrix A in question or the required matrix-vector multiplication.
   - The primary arithmetic cost of the procedure is the cost of m matrix-vector products plus $2m^2n$ for the rest. It is common that the matrix-vector multiplication is the dominant cost.
   - The Arnoldi procedure breaks down when $h_{j+1,j} = 0$ for some j. It is easy to see that if the Arnoldi procedure breaks down at step j (i.e. $h_{j+1,j} = 0$), we have

     $$AV_j = V_j H_j.$$

     This indicates that $\mathcal{K}_j$ is an invariant subspace of A.
   - Some care must be taken to insure that the vectors v_j remain orthogonal to working accuracy in the presence of rounding error. The usual technique is called *reorthogonalization.*

5. The Generalized Minimum Residual (GMRES) method[2] is a generalization of the one-dimensional MR iteration. It uses a pare of Krylov subspaces as pair of projection subspaces:

   $$\mathcal{W} = \mathcal{K}_m(A, r_0) \quad \text{and} \quad \mathcal{V} = A\mathcal{W} = A\mathcal{K}_m(A, r_0).$$

   The GMRES method can then be derived under the framework of the subspace projection technique (shown in the class).

6. We can also derive the GMRES method by exploitng the optimality property. Note that any vector x in $x_0 + \mathcal{K}_m$ can be written as $x = x_0 + V_m y$, where y is an m-vector. Define

   $$J(y) = \|b - Ax\|_2 = \|b - A(x_0 + V_m y)\|_2 \tag{1}$$

   Then using the Arnoldi decomposition, we have

   $$\begin{aligned} b - Ax &= b - A(x_0 + V_m y) = r_0 - AV_m y \\ &= \beta v_1 - V_{m+1}\widehat{H}_m y = V_{m+1}(\beta e_1 - \widehat{H}_m y). \end{aligned}$$

[2] Y. Saad and M. H. Schultz. GMRES: a Generalized Minimal RESidual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing, Vol.7, pp.856–869, 1986.

Since the column vectors of V_{m+1} are orthonormal, then

$$J(y) = \|b - A(x_0 + V_m y)\|_2 = \|\beta e_1 - \widehat{H}_m y\|_2.$$

Therefore, the GMRES approximation x_m is the unique vector

$$x_m = x_0 + V_m y,$$

where y the solution of the least squares problem

$$\min_y \|\beta e_1 - \widehat{H}_m y\|_2.$$

This least squares problem is inexpensive to compute since m is typically small.

7. Restarting GMRES method. As m increases, the computational cost increases at least as $O(m^2 n)$. The memory cost increases as $O(mn)$. For large n this limits the largest value of m that can be used. The popular remedy is to restart the algorithm periodically for a fixed m.

   RESTARTED GMRES:
   1. compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$
   2. call Arnoldi procedure with A, v_1 and m
   3. solve $\min_y \|\beta e_1 - \widehat{H}_m y\|_2$
   4. $x_m = x_0 + V_m y_m$
   5. test for convergence, if satisfied, then *stop*
   6. set $x_0 := x_m$ and go to 1.

8. Breakdown of GMRES: Since the least squares problem always has solution, the only possibility of the breakdown of the GMRES is in the Arnoldi procedure when $h_{j+1,j}$ at some step j. However, in this case, the residual norm of x_j is zero, $b - Ax_j = 0$. x_j is the exact solution of the linear system $Ax = b$. This is called *lucky breakdown*. In fact, we have

   Proposition 2 *Let A be a nonsingular matrix. Then the GMRES algorithm breaks down at step j, i.e., $h_{j+1,j} = 0$, if and only if x_j is an exact solution of $Ax = b$.*

Further reading

1. Convergence of GMRES. We wish to establish a result to provide an upper bound on the convergence rate of the GMRES iterates. Unfortunately, because of the complication of non-Hermitian matrices and their spectral distribution, it is not possible to prove a simple result, but can get pretty close for practical use. First, we have the following lemma to characterize the approximate solution by the GMRES method:

   Lemma 1 *Let x_m be the approximate solution obtained from the m-th step of the GMRES algorithm, and let $r_m = b - Ax_m$. Then x_m is of the form*

   $$x_m = x_0 + q_m(A) r_0$$

   and

   $$\|r_m\|_2 = \|(I - Aq_m(A))r_0\|_2 = \min_{q \in \mathcal{P}_{m-1}} \|(I - Aq(A))r_0\|_2.$$

PROOF: This is true because x_m minimizes the 2-norm of the residual in the affine subspace $x_0 + \mathcal{K}_m$, the optimality property of the projection technique. Recall that $\mathcal{K}_m$ is the set of all vectors of the form $x_0 + q(A)r_0$, where q is a polynomial of degree $\leq m-1$. ∎

Proposition 3 *Assume that A is diagonalizable matrix and let $A = V\Lambda V^{-1}$ where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the diagonal matrix of eigenvalues. Define*

$$\epsilon^{(m)} = \min_{p \in \mathcal{P}_m, p(0)=1} \max_{1 \leq i \leq n} |p(\lambda_i)|.$$

Then the residual norm satisfies the inequality

$$\|r_m\|_2 \leq \kappa_2(V)\epsilon^{(m)}\|r_0\|_2.$$

where $\kappa_2(V) = \|V\|_2\|V^{-1}\|_2$.

PROOF: see [Y. Saad, Iterative methods for sparse linear systems, Second Edition, SIAM, 2003]

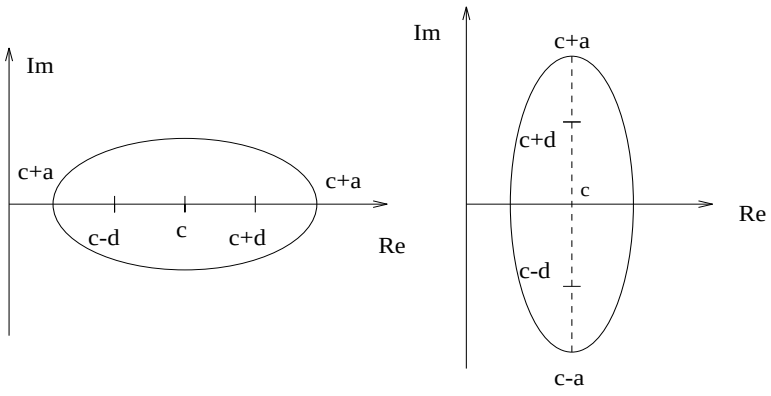
The results of approximation theory on near-optimal Chebyshev polynomials in the complex plane can now be used to obtain an upper bound for $\epsilon^{(m)}$. This is stated in the following corollary.

Corollary 1 *Assume that all the eigenvalues of A are located in the ellipse $E(c, d, a)$ which excludes the origin. Then*

$$\|r_m\|_2 \lesssim \kappa_2(V) \left(\frac{a + \sqrt{a^2 - d^2}}{c + \sqrt{c^2 - d^2}} \right)^m \|r_0\|_2.$$

PROOF: see [Y. Saad, Iterative methods for sparse linear systems, Second Edition, SIAM, 2003]

The follow plots show the spectrum of A is contained in the ellipses $E(c, d, a)$ with center c, focal distance d and major semi axis a. The left plot is for the case of real d and the right plot is for the case of purely imaginary d.



Since the condition number $\kappa_2(V)$ is typically not known and can be very large, results are of limited practical interest. They can be useful one when it is known that the matrix is nearly normal, in which case, $\kappa_2(V) \approx 1$.

10 4-27-12

10.1 Framework

1. V_m = Kyrlov subspace,
$$V_m = \text{span}\left\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\right\},$$
where $r_0 = b - Ax_0$

2. Optimal approximation: projection

$$\begin{aligned} b - A(x_0 + V_m z) &\perp V_m \qquad (A \text{ SPD}) \\ V_m^T(b - A(x_0 + V_m z)) &= 0 \\ V_m^T A V_m z &= V_m^T r_0 \\ &\text{OR} \\ b - A(x_0 + V_m z) &\perp AV_m \qquad (A \text{ general}) \\ V_m^T A^T(b - A(x_0 + V_m a)) &= 0 \\ V_m^T A^T A V_m z &= V_m^T A^T r_0 \end{aligned}$$

10.2 Implementation

1. V_m = Kyrlov subspace = $\begin{bmatrix} v_1 & v_2 & \cdots & v_m \end{bmatrix}$, orthonormal. By exploting the structure of the Kyrlov subspace, we can make the Gram-Schmidt process more efficient $\Rightarrow$ Arnoldi procedure. In matrix form,
$$AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T$$
where H_m is an upper Hessenberg matrix (almost triangular) and $h_{m+1,m}$ is a scalar. $V_m^T V_m = I$, $V_m^T v_{m+1} = 0$. In shorthand format:
$$\begin{aligned} AV_m &= \left[\begin{array}{c|c} V_m & v_{m+1} \end{array}\right] \left[\begin{array}{c} H_m \\ \hline h_{m+1,m} v_{m+1} e_m^T \end{array}\right] \\ &= V_{m+1} \hat{H}_m \end{aligned}$$

2. GMRES.
$$V_m^T A^T A V_m = \hat{H}_m^T \underbrace{V_{m+1}^T V_{m+1}}_{I} \hat{H}_m = \hat{H}_m^T \hat{H}_m, \qquad v_1 = \frac{r_0}{\|r_0\|}$$
$$\begin{aligned} V_m^T A^T r_0 &= \hat{H}_m^T V_{m+1}^T r_0 \\ &= \hat{H}_m^T \|r_0\| \cdot V_{m+1}^T v_1 \\ &= \|r_0\| \cdot \hat{H}_m^T e_1 \end{aligned}$$
Reduced linear system:
$$\hat{H}_m^T \hat{H}_m z = \|r_0\| \cdot \hat{H}_m^T e_1$$
This is the normal equation. It is equivalent to
$$\min_z \|\hat{H}_m z - \|r_0\| e_1\|_2$$

10.3 GMRES (version 0)

- Given x_0 = initial approximation
- $r_0 = b - Ax_0$ ("restarting point")
- Use the Arnoldi procedure to generate (V_{m+1}) $\hat{H}_m$
- Solve $\min_z \|\hat{H}_m z - \|r_0\| e_1\|_2$. Equivalently, $\hat{H}_m^T \hat{H}_m z = \hat{H}_m(\|r_0\| e_1)$.
- Test for convergence of $x_1 = x_0 + V_m z \Rightarrow$ stopping criterion
- If it has not converged, you can either:
  - restart if m is fixed
  - expand $m + 1$

11 4-30-12

11.1 Convergence Test/Stopping Criterion

$$\begin{aligned} x_1 &= x_0 + V_m z \\ r_1 &= b - Ax_1 = b - A(x_0 + V_m z) \\ &= r_0 - Av_m z \\ &= r_0 - V_{m+1}\hat{H}_m z \\ &= r_0 - (V_m H_m + h_{m+1,m} v_{m+1} e_m^T) z \\ &= \underbrace{r_0 - V_m H_m z}_{=0} - h_{m+1,m} v_{m+1} e_m^T z \\ \|r_1\| &= |h_{m+1,m}| \cdot |e_m^T z| \end{aligned}$$

11.2 GMRES (version 0)

- Given x_0 = initial approximation
- $r_0 = b - Ax_0$ ("restarting point")
- Use the Arnoldi procedure to generate (V_{m+1}) $\hat{H}_m$
- Sovel $\hat{H}_m^T \hat{H}_m z = \hat{H}_m^T (\|r_0\| e_1)$
- Test $\|r_1\| = |h_{m+1,m}| \cdot |e_m^T z|$
- Iterate...

- Finally: $x_1 = x_0 + V_m z$

11.3 The CG Method

Arnoldi process:

$$\begin{aligned} AV_m &= V_m H_m h_{m+1,m} v_{m+1} e_m^T \\ V_m^T A V_m &= H_m \end{aligned}$$

If $A^T = A$, then (upper Hessenberg) $H_m = H_m^T$ (lower Hessenberg), so H_m is tridiagonal. We call this matrix T_m, and the Lanczos process generates T_m directly.

Subspace Projection:

$$\begin{aligned} x_1 &= x_0 + V_m z \\ V_m^T \underbrace{A}_{\text{SPD}} V_m z &= V_m^T b \\ T_m z &= \tilde{b} \\ x_1 &= x_0 + V_m T_m^{-1} \tilde{b} \end{aligned}$$

Observations:

1. T_m is the product of lower and upper bidiagonal matrices

2. (For the first step, $x_0 \to x_j,\ x_1 \to x_{j+1}$)

$$\begin{aligned} x_{j+1} &= x_j + \alpha_j p_j \\ r_{j+1} &= r_j + \alpha_j A p_j = b - A x_{j+1} \\ p_{j+1} &= r_{j+1} + \beta_j p_j, \end{aligned}$$

   where p_j is the direction and α is the step size

So we just need to know how to get α_j and β_j.

1. If $r_1, r_2, r_3, \ldots$ are basis vectors for the Krylov subspace, then

$$\begin{aligned} r_{j+1}^T r_j &= 0 \\ (r_j^T + \alpha_j p_j^T A) r_j &= 0 \\ \alpha_j &= -\frac{r_j^T r_j}{p_j^T A r_j} \end{aligned}$$

2.

$$\begin{aligned} p_{j+1}^T A p_j &= 0 \\ \beta_j &= \end{aligned}$$

<u>The CG Method</u>

- $x_0,\ r_0 = b - A x_0,\ p_0 = r_0$
- **for** $j = 0, 1, 2, \ldots$
  - Compute α_j

$$\begin{aligned} x_{j+1} &= x_j + \alpha_j p_j \\ r_{j+1} &= r_j + \alpha_j A p_j \end{aligned}$$

  - Compute β_j

$$p_{j+1} = r_j + \beta_j p_j$$

- **end** (j)

1. The *symmetric Lanczos procedure* can be regarded as a simplification of Arnoldi's procedure when A is symmetric.

   By an order-m Arnoldi decomposition, we know that

   $$H_m = V_m^T A V_m.$$

   If A is symmetric, then H_m becomes symmetric tridiagonal. This simple observation leads to the following procedure to compute an orthonormal basis Q_m of Krylov subspace $\mathcal{K}_m(A, v)$ when A is symmetric[1]:

   $[V_{m+1}, \widehat{T}_m] = \text{Lanczos}(A, v, m)$
   1. $v_1 = v/\|v\|_2$, set $\beta_1 = 0$, $v_0 = 0$
   2. for $j = 1, 2, \ldots, m$
   3. $\quad w = Av_j - \beta_j v_{j-1}$
   4. $\quad \alpha_j = v_j^T w$
   5. $\quad w := w - \alpha_j v_j$
   8. $\quad \beta_{j+1} = \|w\|_2$
   9. $\quad$ If $\beta_{j+1} = 0$, then *stop*
   10. $\quad v_{j+1} = w/\beta_{j+1}$
   11. endfor

   Remarks:

   - Only three vectors must be saved in the inner loop of the procedure. This is referred as a *three-term recurrence*.
   - The computed Lanczos vectors $\{v_i\}$ are orthogonal in exact arithmetic. In the presence of finite precision, it starts losing such orthogonality rapidly with the increase of j. (The same phenomenon is also observed in the Arnoldi procedure, but it's not as severe as in the Lanczos procedure). There has been much research devoted to understanding the effect of loss of the orthogonality, and finding ways to either recover the orthogonality, or to at last diminish its effects. An excellent reference on the subject is [B. N. Parlett, The Symmetric Eigenvalue Problem, SIAM Press, 1998].

2. In the matrix form, the Lanczos procedure can be expressed in the following governing equations, referred to as an *order−m Lanczos decomposition*:

   $$\begin{aligned} AV_m &= V_m T_m + \beta_{m+1} v_{m+1} e_m^T \\ &= V_{m+1} \widehat{T}_m \end{aligned}$$

   where $V_m = [v_1, v_2, \ldots, v_m]$, $V_{m+1} = [V_m, v_{m+1}]$, and

   $$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \ddots & & \\ & \beta_3 & \ddots & \beta_{m-1} & \\ & & \ddots & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{bmatrix} \equiv \text{tridiag}(\beta_j, \alpha_j, \beta_{j+1}) \quad \text{and} \quad \widehat{T}_m = \begin{bmatrix} T_m \\ \beta_{m+1} e_m^T \end{bmatrix}.$$

[1] Note that we change the notation $\alpha_j = h_{jj}$ and $\beta_{j+1} = h_{j-1,j}$, comparing with the Arnoldi procedure.

By the orthogonlity properties $V_m^T V_m = I$ and $V_m^T v_{m+1} = 0$, we have

$$V_m^T A V_m = T_m = \text{tridiag}(\beta_j, \alpha_j, \beta_{j+1}).$$

3. The *Conjugate Gradient (CG) method* is the best known iterative techniques for solving sparse SPD linear system, $Ax = b$, first published in 1952 by Hestenes and Stiefel.[2] There are several ways to derive the CG method. In terms of our familiar subspace projection technique, we can describe the CG method in one sentence:

   > *The CG method is a realization of an orthogonal projection technique onto the Krylov subspace $\mathcal{K}_m(A, r_0)$, where A is symmetric positive definite and $r_0 = b - Ax_0$ with initial guess x_0.*

   In the following, we provide a derivation of the CG method under this algorithmic framework.

4. Before we derive the CG method, we first derive a so-called *direct Lanczos method*. With an initial guess x_0, the approximate solution obtained from an orthogonal projection method onto $x_0 + \mathcal{K}_m(A, r_0)$ is given by

$$x_m = x_0 + V_m y_m \tag{1}$$

   where y_m is the solution of the tridiagonal system

$$T_m y_m = \beta e_1, \quad \text{where} \quad \beta = \|r_0\|_2.$$

   Now, let's try to compute the solution of the tridiagonal system *progressively* along with the Lanczos procedure. For doing so, let's write the LU factorization of T_m as

$$T_m = L_m U_m,$$

   i.e. the Gaussian elimination without pivoting:

$$T_m = L_m U_m = \begin{bmatrix} 1 & & & & \\ \lambda_2 & 1 & & & \\ & \lambda_3 & 1 & & \\ & & \ddots & \ddots & \\ & & & \lambda_m & 1 \end{bmatrix} \begin{bmatrix} \eta_1 & \beta_2 & & & \\ & \eta_2 & \beta_3 & & \\ & & \ddots & \ddots & \\ & & & \eta_{m-1} & \beta_m \\ & & & & \eta_m \end{bmatrix},$$

   where $\eta_1 = \alpha_1$, and for $j = 2, 3, \ldots, m$,

$$\lambda_j = \beta_j / \eta_{j-1}, \qquad \eta_j = \alpha_j - \lambda_j \beta_j.$$

   Then x_m is given by

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1} (\beta e_1)$$

   Let $P_m = V_m U_m^{-1}$ and $z_m = L_m^{-1}(\beta e_1)$, then

$$x_m = x_0 + P_m z_m.$$

   The following two observations connect P_m and z_m of the mth step with P_{m-1} and z_{m-1} of the previous step.

[2] M. R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bur. Standards, 49:409–436, 1952.

(a) Let us write $P_m = [P_{m-1}\ p_m]$, where p_m is the last column of P_m, then we have

$$\begin{aligned} P_m &= V_m U_m^{-1} = \left[\begin{array}{cc} V_{m-1} & v_m \end{array}\right] \left[\begin{array}{cc} U_{m-1} & \beta_m e_{m-1} \\ & \eta_m \end{array}\right]^{-1} \\ &= \left[\begin{array}{cc} V_{m-1} & v_m \end{array}\right] \left[\begin{array}{c|c} U_{m-1}^{-1} & -U_{m-1}^{-1}(\beta_m e_{m-1})\eta_m^{-1} \\ \hline & \eta_m^{-1} \end{array}\right] \\ &= \left[\begin{array}{cc} V_{m-1}U_{m-1}^{-1} & -V_{m-1}U_{m-1}^{-1}(\beta_m e_{m-1})\eta_m^{-1} + v_m\eta_m^{-1} \end{array}\right] \\ &= \left[\begin{array}{cc} P_{m-1} & -P_{m-1}(\beta_m e_{m-1})\eta_m^{-1} + v_m\eta_m^{-1} \end{array}\right] \\ &= \left[\begin{array}{cc} P_{m-1} & \eta_m^{-1}(v_m - \beta_m p_{m-1}) \end{array}\right] \end{aligned}$$

Therefore, we see that the vector p_m can be computed from previous p_{m-1} and v_m by the simple update

$$p_m = \eta_m^{-1}(v_m - \beta_m p_{m-1}), \tag{2}$$

(b) By the definition of the vector z_m, we have

$$\begin{aligned} z_m &= L_m^{-1}(\beta e_1) = \left[\begin{array}{c|c} L_{m-1}^{-1} & \\ \hline -\lambda_m e_{m-1}^T L_{m-1}^{-1} & 1 \end{array}\right] \left[\begin{array}{c} \beta e_1 \\ 0 \end{array}\right] \\ &= \left[\begin{array}{c} L_{m-1}^{-1}(\beta e_1) \\ -\lambda_m e_{m-1}^T L_{m-1}^{-1}(\beta e_1) \end{array}\right] \equiv \left[\begin{array}{c} z_{m-1} \\ \zeta_m \end{array}\right] \end{aligned}$$

where $\zeta_m = -\lambda_m \zeta_{m-1}$.

As a result of these two observations, x_m can be written in an updated form

$$\begin{aligned} x_m &= x_0 + [P_{m-1},\ p_m] \left[\begin{array}{c} z_{m-1} \\ \zeta_m \end{array}\right] \\ &= x_0 + P_{m-1}z_{m-1} + \zeta_m p_m \\ &= x_{m-1} + \zeta_m p_m. \end{aligned}$$

This gives the following direct Lanczos algorithm:

DIRECT LANCZOS METHOD

1. compute $r_0 = b - Ax_0$, $\beta := \zeta_1 := \|r_0\|_2$, and $v_= r_0/\beta$,
2. set $\lambda_1 = \beta_1 = 0$, $p_0 = 0$
3. for $m = 1, 2, \ldots,$
4. $\quad w := Av_m - \beta_m v_{m-1}$ and $\alpha_m = v_m^T w$
5. $\quad$ If $m > 1$ then compute $\lambda_m = \beta_m/\eta_{m-1}$ and $\zeta_m = -\lambda_m\zeta_{m-1}$
6. $\quad \eta_m = \alpha_m - \lambda_m\beta_m$
7. $\quad p_m = \eta_m^{-1}(v_m - \beta_m p_{m-1})$
8. $\quad x_m = x_{m-1} + \zeta_m p_m$
9. $\quad$ If x_m has converged, then Stop
10. $\quad w := w - \alpha_m v_m$
11. $\quad \beta_{m+1} = \|w\|_2$ and $v_{m+1} = w/\beta_{m+1}$
12. endfor

5. Now let us examine the residual vector r_m of the approximate solution x_m,

$$\begin{aligned} r_m &= b - Ax_m = b - A(x_0 + V_m y_m) = r_0 - AV_m y_m \\ &= r_0 - (V_m T_m + \beta_{m+1} v_{m+1} e_m^T) y_m \\ &= r_0 - V_m T_m y_m - \beta_{m+1} v_{m+1} (e_m^T y_m) \\ &= -\beta_{m+1} v_{m+1} (e_m^T y_m). \end{aligned}$$

Therefore, we see that the residual vector r_m is in the direction of v_{m+1}. Since $\{v_i\}$ are orthogonal, we conclude that

$$\text{the residual vectors } \{r_i\} \text{ are orthogonal,i.e., } r_j^T r_i = 0 \text{ for } i \neq j. \tag{3}$$

Next we can show that

$$\text{the vectors } \{p_i\} \text{ are } A\text{-conjugate, i.e., } p_j^T A p_i = 0 \text{ for } i \neq j. \tag{4}$$

To show this, we just need to show that $P_m^T A P_m$ is a diagonal matrix. In fact,

$$P_m^T A P_m = U_m^{-T} V_m^T A V_m U_m^{-1} = U_m^{-T} T_m U_m^{-1} = U_m^{-T} L_m U_m U_m^{-1} = U_m^{-T} L_m.$$

Note that $U^{-T} L_m$ is a lower triangular which is also symmetric. Therefore it must be a diagonal matrix.

A consequence of the orthogonality condition (3) and conjugacy condition (4) is that a version of the algorithm can be derived by directly imposing the conditions (3) and (4). This gives the Conjugate Gradient (CG) algorithm.

We now drive this. Let express the vector x_{j+1} as [3]

$$x_{j+1} = x_j + \alpha_j p_j$$

Therefore, the residual vectors must satisfy the recurrence

$$r_{j+1} = b - Ax_{j+1} = b - A(x_j + \alpha_j p_j) = r_j - \alpha_j A p_j. \tag{5}$$

Since the r_j's are orthogonal, i.e., $r_j^T r_{j+1} = 0$, then it gives

$$\alpha_j = \frac{r_j^T r_j}{r_j^T A p_j}$$

By (2), it is known that the next search direction p_{j+1} is a linear combination of r_{j+1} and p_j, and with proper rescaling the p vectors approximately, it can be written as

$$p_{j+1} = r_{j+1} + \beta_j p_j.$$

Thus a first consequence of the above relation is that

$$r_j^T A p_j = (p_j - \beta_{j-1} p_{j-1})^T A p_j = p_j^T A p_j.$$

[3] Note that the scalars α_j and β_j here are different from those of the direct Lanczos method.

i.e.,

$$\alpha_j = \frac{r_j^T r_j}{p_j^T A p_j}.$$

By imposing A-conjugacy $p_{j+1}^T A p_j = 0$, we have

$$\beta_j = -\frac{p_j^T A r_{j+1}}{p_j^T A p_j}$$

Note that from (5), $Ap_j = -\frac{1}{\alpha_j}(r_{j+1} - r_j)$ and therefore

$$\beta_j = \frac{1}{\alpha_j}\frac{(r_{j+1} - r_j)^T r_{j+1}}{p_j^T A p_j} = \frac{r_{j+1}^T r_{j+1}}{r_j^T r_j}$$

Putting these relations together gives the following CG algorithm

Conjugate Gradient (CG) Method
1. compute $r_0 = b - Ax_0$ and $p_0 := r_0$
2. for $j = 0, 1, 2, \ldots,$ until convergence do
3. $\quad \alpha_j = r_j^T r_j / (p_j^T A p_j)$
4. $\quad x_{j+1} = x_j + \alpha_j p_j$
5. $\quad r_{j+1} = r_j - \alpha_j A p_j$
6. $\quad \beta_j = r_{j+1}^T r_{j+1} / (r_j^T r_j)$
7. $\quad p_{j+1} = r_{j+1} + \beta_j p_j$
8. endfor

Note that in addition to the matrix A, four vectors of storage are required: x, p, Ap and r.

Further reading

1. There are many different derivations of the CG method, for example, see the following paper (pdf file is available at the class website)

   Jonathan Shewchuk, *An Introduction to Conjugate Gradient Method Without the Agonizing Pain.* 1994 (64 pages)

2. Convergence analysis of the CG method

   (a) From the optimality of the projection technique, we know that the approximate solution obtained from the m-th step of the CG algorithm minimizes the A-norm of the error in the affine subspace $x_0 + \mathcal{K}_m(A, r_0)$. Since $\mathcal{K}_m$ is the set of all vectors of the form $x_0 + q(A)r_0$, where q is a polynomial of degree $\leq m - 1$, we conclude the following lemma which characterizes the approximate solution x_m:

   Lemma 1 *Let x_m be the approximate solution obtained from the m-th step of the CG algorithm, and let $d_m = x_* - x_m$ where x_* is the exact solution of $Ax = b$. Then x_m is of the form*

   $$x_m = x_0 + q_m(A)r_0$$

   where q_m is a polynomial of degree $m - 1$ such that

   $$\|(I - Aq_m(A))d_0\|_A = \min_{q \in \mathcal{P}_{m-1}} \|(I - Aq(A))d_0\|_A$$

(b) From Lemma 1, we have the following theorem.

Theorem 1 *Let x_m be the approximate solution obtained from the m-th step of the CG algorithm, and x_* is the exact solution of $Ax = b$. Then,*

$$\|x_* - x_m\|_A \leq \frac{1}{T_m(1+2\eta)} \|x_* - x_0\|_A, \tag{6}$$

where T_m is the Chebyshev polynomial of degree m, and $\eta = \lambda_{\min}/(\lambda_{\max} - \lambda_{\min})$. $\lambda_{\max}$ and $\lambda_{\min}$ are the largest and smallest eigenvalues of A.

A slightly different formulation of inequality can be derived. Using the relation

$$T_m(t) = \frac{1}{2}\left[\left(t + \sqrt{t^2-1}\right)^m + \left(t + \sqrt{t^2-1}\right)^{-m}\right] \geq \frac{1}{2}\left(t + \sqrt{t^2-1}\right)^m.$$

Then

$$T_m(1+2\eta) \geq \frac{1}{2}\left(1 + 2\eta + \sqrt{(1+2\eta)^2 - 1}\right)^m = \frac{1}{2}\left(1 + 2\eta + 2\sqrt{\eta(\eta+1)}\right)^m.$$

Now notice that

$$\begin{aligned} 1 + 2\eta + 2\sqrt{\eta(\eta+1)} &= (\sqrt{\eta} + \sqrt{\eta+1})^2 = \frac{(\sqrt{\lambda_{\min}} + \sqrt{\lambda_{\max}})^2}{\lambda_{\max} - \lambda_{\min}} \\ &= \frac{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}}{\sqrt{\lambda_{\max}} - \sqrt{\lambda_{\min}}} = \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1} \end{aligned}$$

where κ is the condition number of A, $\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$. Substituting into the inequality (6) yields

$$\|x_* - x_m\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m \|x_* - x_0\|_A.$$

This bound is similar to that of the steepest descent algorithm except that the condition number of A is now replaced by its square root. CG method could be of order of magnitudes faster than the steepest descent algorithm. For example, let $\kappa = 10^3$, if one wants

$$\left(\frac{k-1}{k+1}\right)^{m_1} = \left(\frac{\sqrt{k}-1}{\sqrt{k}+1}\right)^{m_2} = 10^{-2}$$

then it means that the steepest descent algorithm needs to take $m_1 \approx 2300$ iterations to reach the same level of accuracy as $m_2 \approx 73$ iterations of the CG method.

(c) The above analysis using the condition number may not explain all the convergence behavior of CG. In fact, the entire distribution of eigenvalues of A is important, not just the ratio of the largest to the smallest one. If the largest and smallest eigenvalues of A are few in number (or clustered closely together), then CG will converge much more quickly than the above analysis based just on A's condition number would indicate. Any important fact is that the behavior of CG in floating point arithmetic can differ significantly from its behavior in exact arithmetic[4].

[4] A. Greenbaum, Iterative Methods for Solving Linear Systems, SIAM, Philadelphia, 1997.
H. van der Vorst, Iterative methods for large linear systems, Cambridge University Press, 2003

12 5-2-12

12.1 Linear Systems

CG Kernels

$$
\begin{aligned}
x_1 &= x_0 + \alpha_0 p_0 & p_0 &= r_0 = b - Ax_0 \\
r_1 &= r_0 + \alpha_0 A p_0 & \text{by } r_1^T r_0 &= 0 \\
& & \alpha_0 &= -\frac{r_0^T r_0}{r_0^T A p_0} \\
p_1 &= r_1 + \beta_0 p_0 & p_1^T A p_0 &= 0 \\
& & \beta_0 &= -\frac{p_0^T A r_1}{p_0^T A p_0}
\end{aligned}
$$

Alternative derivation: "geometric way" ← J. Shewchuk

Linear system $Ax = b \Leftrightarrow$ Minimization problem $\min_x \frac{1}{2} x^T Ax - x^T b$

Implementation Issues

1. Matrix-vector multiplication: $p \to \textcircled{A} \to q = Ap$, where A is usualy large and sparse ← "Matrix Market" (NIST)
2. Stopping criterion. x_j is the jth approximation of x_* (the exact solution).

$$
\begin{aligned}
Ax_* &= b \\
r_j &= b - Ax_j \\
\|r_j\| \le \text{ tol } &= 10^{-16}? \quad \Rightarrow \quad \|x_j - x_*\| \le \ ? \\
r_j &= b - Ax_j \\
Ax_j &= b - r_j = b + \Delta b \qquad \text{"backward error"} \\
Ax_j + r_j &= b \\
\left(A + \frac{r_j x_j^T}{x_j^T x_j} \right) x_j &= b \\
(A + \Delta A) x_j &= b
\end{aligned}
$$

Computed $\hat{x}$ satisfies

$$(A + \Delta A)\hat{x} = b + \Delta b, \qquad \|\Delta A\|, \|\Delta b\| \approx \|r_j\|$$

So we have a backward stable algorithm. Forward error:

$$\begin{aligned}
\frac{\|\hat{x}-x_*\|}{\|x_*\|} &\leq \ ? \\
Ax_* &= b \\
A\hat{x}+\Delta A\hat{x} &= b+\Delta b \\
A(\hat{x}-x_*)+\Delta A\hat{x} &= \Delta b \\
\hat{x}-x_* &= A^{-1}(\Delta b-\Delta A\hat{x}) \\
\|\hat{x}-x_*\| &\leq \|A^{-1}\|(\|\Delta b\|+\|\Delta A\|\|\hat{x}\|) \\
\frac{\|\hat{x}-x_*\|}{\|x_*\|} &\leq \frac{\|A^{-1}\|}{\|x_*\|}(\|\Delta b\|+\|\Delta A\|\|\hat{x}\|) \\
&\leq \underbrace{\|A^{-1}\|\|A\|}_{\text{condition number}} \underbrace{\left(\frac{\|\Delta b\|}{\|b\|}+\frac{\|\Delta A\|}{\|A\|}\frac{\|\hat{x}\|}{\|b\|}\|A\|\right)}_{\text{relative backward error}}
\end{aligned}$$

Rule of Thumb

$$\text{relative forward error} \lesssim (\text{condition \#}) \times (\text{relative backward error})$$

The condition number is an intrinsic property of the problem, whereas the relative backward error is user controlled.

12.2 Preconditioning

The problem $Ax = b$ has condition number $\kappa(A) = \|A\|\|A^{-1}\|$. The equivalent problem $MAx = Mb$ has condition number $\kappa(MA)$, which is hopefully good. The idea of preconditioning is to modify the system in this way to obtain a good condition number.

1. By the convergence analysis of CG and GMRES algorithms, we learn that the convergence rate strongly depends on the condition number of the coefficient matrix A of the linear system $Ax = b$, and the distribution of A's eigenvalues. Other Krylov subspace methods share the similar property.

2. *Preconditioning* means replacing the system $Ax = b$ with the modified systems

$$M^{-1}Ax = M^{-1}b. \tag{1}$$

or

$$AM^{-1}\hat{x} = b, \quad x = M^{-1}\hat{x}. \tag{2}$$

These are referred to as left and right preconditioning, respectively.

If the preconditioner M is SPD, then one can precondition symmetrically and solve the modified linear system

$$L^{-1}AL^{-T}y = L^{-1}b, \quad x = L^{-T}y, \tag{3}$$

where $M = LL^T$. The matrix L could be the Cholesky factor of M or any other matrix satisfying $M = LL^T$.

3. The desired *preconditioner* M should be chosen so that

   (a) $M^{-1}A$ or $L^{-1}AL^{-T}$ is "well-conditioned" or approximates "the identity matrix",

   (b) linear systems with coefficient matrix M are easy to solve.

   A careful choice of M can often make the condition number of the modified system much smaller than the condition number of the original one, and thus accelerate convergence dramatically. Indeed, a good preconditioner is often necessary for an iterative method to converge at all, and much past and current research in iterative methods is directed at finding better preconditioners.

4. We now show that a preconditioner can be easily incorporated into the CG method, and lead to a Preconditioned Conjugate Gradient method, PCG for short.

   If the CG algorithm is applied directly to the symmetric preconditioned system (3), the iterative kernels satisfy

$$\begin{aligned} y_{j+1} &= y_j + \hat{\alpha}_j \hat{p}_j \\ \hat{r}_{j+1} &= \hat{r}_j - \hat{\alpha}_j L^{-1}AL^{-T}\hat{p}_j \\ \hat{p}_{j+1} &= \hat{r}_{j+1} + \hat{\beta}_j \hat{p}_j \end{aligned}$$

   with

$$\hat{\alpha}_j = \frac{\hat{r}_j^T \hat{r}_j}{\hat{p}_j^T L^{-1}AL^{-T}\hat{p}_j} \quad \text{and} \quad \hat{\beta}_j = \frac{\hat{r}_{j+1}^T \hat{r}_{j+1}}{\hat{r}_j^T \hat{r}_j}.$$

   Defining

$$x_j = L^{-T}y_j, \quad r_j = L\hat{r}_j, \quad p_j = L^{-T}\hat{p}_j.$$

   The iterative kernels become

$$\begin{aligned} x_{j+1} &= x_j + \alpha_j p_j \\ r_{j+1} &= r_j - \alpha_j A p_j \\ p_{j+1} &= M^{-1}r_{j+1} + \beta_j p_j \end{aligned}$$

with

$$\alpha_j = \frac{r_j^T M^{-1} r_j}{p_j^T A p_j} \quad \text{and} \quad \beta_j = \frac{r_{j+1}^T M^{-1} r_{j+1}}{r_j^T M^{-1} r_j}.$$

We obtained the following preconditioned CG algorithm for solving $Ax = b$ using the preconditioner $M = LL^T$.

PRECONDITIONED CONJUGATE GRADIENT (PCG)
1. compute $r_0 = b - Ax_0$, solve $Mz_0 = r_0$ and $p_0 := z_0$
2. for $j = 0, 1, 2, \ldots,$ until convergence do
3. $\quad \alpha_j = (r_j^T z_j)/(p_j^T A p_j)$
4. $\quad x_{j+1} = x_j + \alpha_j p_j$
5. $\quad r_{j+1} = r_j - \alpha_j A p_j$
6. $\quad$ solve $Mz_{j+1} = r_{j+1}$
7. $\quad \beta_j = (r_{j+1}^T z_{j+1})/(r_j^T z_j)$
8. $\quad p_{j+1} = z_{j+1} + \beta_j p_j$
9. endfor

5. Similarly, a preconditioner can be easily incorporated into the GMRES method, and lead to a Preconditioned GMRES method, PGMRES for short.

PRECONDITIONED GMRES
1. compute $r_0 = M^{-1}(b - Ax_0)$, $\beta = \|r_0\|_2$ and $v_1 := r_0/\beta$
2. for $j = 0, 1, 2, \ldots, m$ do
3. $\quad$ solve $Mw = Av_j$
4. $\quad$ for $i = 1, 2, \ldots, j$ do
5. $\quad\quad h_{ij} = v_i^T w$
6. $\quad\quad w := w - h_{ij} w_i$
7. $\quad$ end do
8. $\quad$ compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$
9. end do
10. let y_m be the solution of $\min_y \|\beta e_1 - \widehat{H}_m y\|_2$
11. $x_m = x_0 + V_m y_m$
12. If satisfied, **Stop**, else set $x_0 := x_m$ and GOTO 1.

Note that in the above algorithm, $V_m = [v_1, v_2, \ldots, v_m]$ and $\widehat{H}_m$ is a $(m+1) \times m$ upper triangular matrix with the entries h_{ij} computed at steps 4 and 8.

6. Preconditioning Techniques.

The reliability and robustness of iterative techniques, when dealing with various applications, often depends much more on the quality of the preconditioner than on the particular Krylov subspace methods used. Finding a good preconditioner to solve a given sparse linear system is oftne viewed as a combination of art and science. Preconditioners can be divided roughly into three categories:

I. Preconditioners designed for general classes of matrices; e.g. Jacobi, Gauss-Seidel, SOR, incomplete LU factorization, incomplete Cholesky decomposition, approximate inverse.

II. Preconditioners designed for broad classes of underlying problems; e.g. elliptic partial differential equations (such as Poisson equation). Examples are multigrid and domain decomposition preconditioners.

III. Preconditioners designed for a specific matrix or underlying problem; e.g. for the transport equation.

The best choice of a preconditioner is generally application problem-dependent, and also depends on the iterative method being used.

- For CG and related methods to solve a symmetric positive definite system, one would like the condition number of the symmetrically preconditioned matrix $L^{-1}AL^{-T}$ to be close to one, in order for the error bound based on the Chebyshev polynomial to be small, or alternatively, has few extreme eigenvalues.
- For GMRES, a preconditioned matrix that is close to normal and whose eigenvalues are tightly clustered around some point away from the origin would be good, but other properties might also suffice to define a good preconditioner.

7. ILU Factorization Preconditioners.

Except for diagonal matrices, the solution of the linear system with coefficient matrix M requires that we have a suitable decomposition of M. In many instances this will be an LU decomposition. The idea of an incomplete LU preconditioner is to perform an abbreviated (sparse) form of Gaussian elimination of A and to declare the production of the resulting factors to be M. Since M is by construction already factorized, system involving M will be easy to solve.

Let us first introduce a *sparsity set* $\mathcal{Z}$ to control the patterns of zeros. Specifically, let $\mathcal{Z}$ be a set of ordered pairs of integers from $\{1, 2, \ldots, n\}$ containing no pairs of the form (i, i). An incomplete LU factorization of A is a decomposition of the form

$$A = LU + E, \tag{4}$$

where L is unit lower triangular, and U is upper triangular, and L, U and E have the following properties

(a) If $(i, j) \in \mathcal{Z}$ with $i > j$, then $\ell_{ij} = 0$,

(b) If $(i, j) \in \mathcal{Z}$ with $i < j$, then $u_{ij} = 0$,

(c) If $(i, j) \notin \mathcal{Z}$, then $e_{ij} = 0$.

In other words, the elements of L and U are zero on the sparsity set $\mathcal{Z}$, and off the sparsity set the decomposition reproduces A.

It is instructive to consider two extreme cases. (1) If the sparsity $\mathcal{Z}$ set is empty, we get the LU decomposition of A, i.e., we are using A as a preconditioner. (2) If $\mathcal{Z}$ is everything except diagonal pairs of the form (i, i), then we are effectively using the diagonal of A as a preconditioner.

Let us consider an ILU algorithm to generate L and U rowwise. Suppose we have computed the first $k-1$ rows of L and U, and we wish to compute the kth row. Write the first k rows of (4) in the form

$$\begin{bmatrix} A_{11} & A_{1k} \\ a_{k1}^T & a_{kk}^T \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ l_{1k}^T & 1 \end{bmatrix} \begin{bmatrix} U_{11} & U_{1k} \\ 0 & u_{kk}^T \end{bmatrix} + \begin{bmatrix} E_{11} & E_{1k} \\ e_{k1}^T & e_{kk}^T \end{bmatrix}.$$

we need to compute l_{1k}^T and u_{kk}^T. Multiplying out, we find that

$$l_{1k}^T U_{11} + e_{k1}^T = a_{k1}^T \tag{5}$$

and

$$u_{kk}^T + e_{kk}^T = a_{kk}^T - l_{1k}^T U_{1k}$$

We then can solve these two systems in order:

$$\underbrace{\ell_{k1}, \ell_{k2}, \ldots, \ell_{k,k-1}}_{l_{1k}^T}, \underbrace{\nu_{kk}, \nu_{k,k+1}, \ldots, \nu_{k,n}}_{u_{kk}^T}.$$

Suppose that we have computed $\ell_{k1}, \ell_{k2}, \ldots, \ell_{k,j-1}$. If $(k,j) \in \mathcal{Z}$, then set $\ell_{kj} = 0$. If $(k,j) \notin \mathcal{Z}$, then $e_{kj} = 0$, and the equation (5) gives

$$\alpha_{kj} = \sum_{i=1}^{k-1} \ell_{ki}\nu_{ij} + \ell_{kj}\nu_{jj},$$

from which we get

$$\ell_{kj} = \frac{\alpha_{kj} - \sum_{i=1}^{k-1} \ell_{ki}\nu_{ij}}{\nu_{jj}}.$$

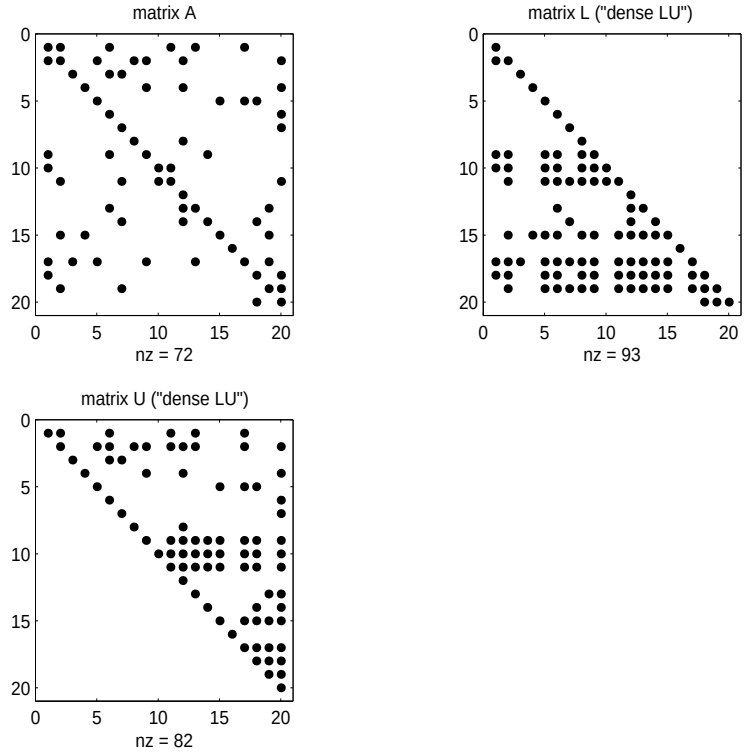
The key observation here is that it does not matter how the values of the preceding ℓ's and ν's were determined. If ℓ_{kj} is defined in this way, then when we compute LU, its (k,j)-element will be α_{kj}. Thus we set ℓ's and ν's to zero on the sparsity set without interfering with the values of LU off the sparsity set. A similar procedure applies to the determination of $\nu_{kk}, \nu_{k,k+1}, \ldots, \nu_{k,n}$.

```
INCOMPLETE_LU_FACTORIZATION(A, Z)
1.  for k = 1 to n
2.      for j = 1 to k − 1
3.          if ((k, j) ∈ Z)
4.              L(k, j) = 0
5.          else
6.              L(k, j) = (A(k, j) − L(k, 1 : j − 1) ∗ U(1 : j − 1, j))/U(j, j)
7.          end if
8.      end for j
9.      for j = k to n
10.         if ((k, j) ∈ Z)
11.             U(k, j) = 0
12.         else
13.             U(k, j) = (A(k, j) − L(k, 1 : k − 1) ∗ U(1 : k − 1, j)
14.         end if
15.     end for j
16. end for k
```

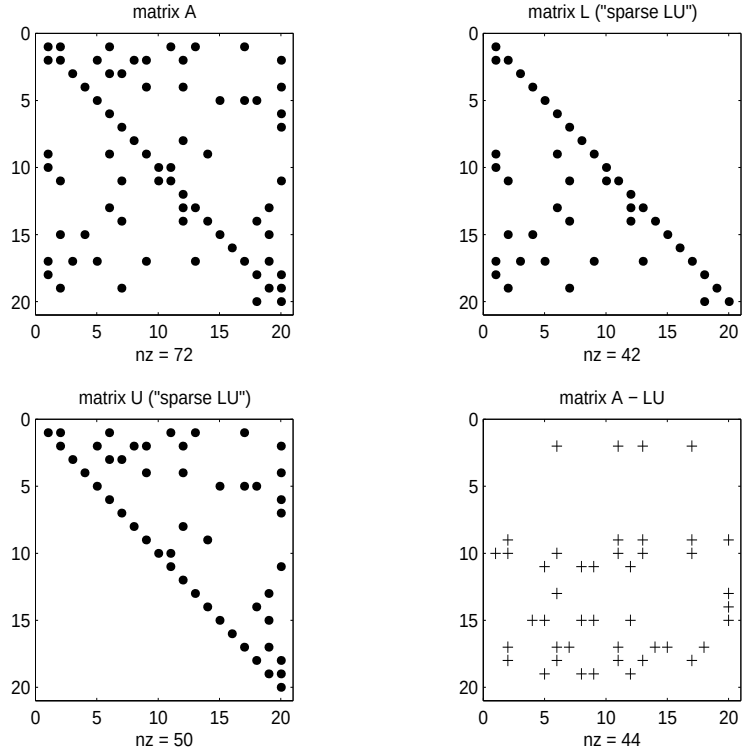
The algorithm can be carried to completion provided the quantities $U(j,j)$ are all nonzero, in which case the decomposition is unique. Whether or not the $U(j,j)$ are nonzero will depend on the matrix in question.

The following figure compares the sparsity of LU and ILU factorizations of a sparse 20 by 20 matrix:

The sparsity of LU



The sparsity of ILU and E-factor



8. Not all matrices can have an ILU factorization. The following two classes of matrices, the algorithm always works.

   (a) If A is nonsingular diagonally dominant matrix, then A has an incomplete LU factoriza-

tion for any sparsity set $\mathcal{Z}$.

Note: A matrix A of order n is *diagonally dominant* if

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}|, \qquad \text{for } i = 1, 2, \ldots, n.$$

It is strictly diagonally dominant if strictly inequality holds for all j.

It can be shown that *A strictly diagonally dominant matrix is nonsingular*. Be aware that diagonal dominance alone does not imply either nonsingularity or singularity. For examples, let

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Then A is nonsingular. On the other hand, B is singular.

(b) The incomplete LU factorization also exists for any M-matrix.

Note: A matrix is said to be an M-matrix if it satisfies the following properties:

(1) $a_{ii} > 0$ for $i = 1, \ldots n$,

(2) $a_{ij} \leq 0$ for $i \neq j, i, j = 1, \ldots n$,

(3) A is nonsingular and

(4) A^{-1} is a nonnegative matrix (all entries are nonnegative).

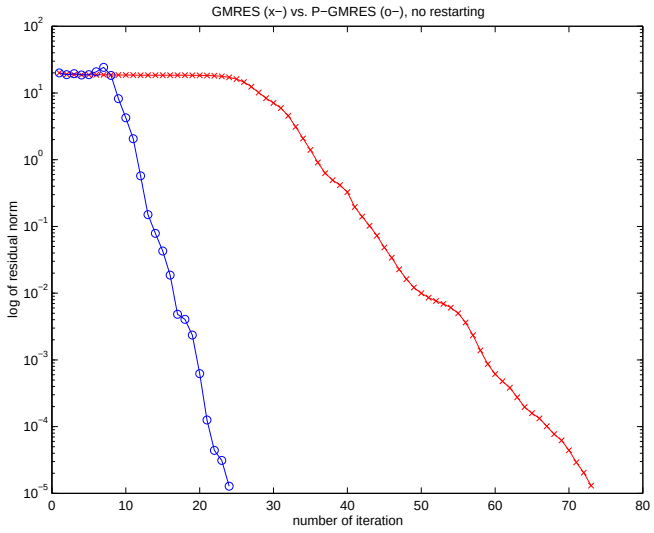
9. Block preconditioner is a popular technique for block-tridiagonal matrices arising from the discretization of elliptic problems, such as Poisson's equation. It can be also be generalized to other sparse matrices. For example, the matrix arises in the solution of 2D Poisson's equation has the form

$$A = \begin{pmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & T & -I \\ & & & -I & T \end{pmatrix}$$

where T is a symmetric tridiagonal matrix, with diagonal entres all 4, and off diagonal entries all -1. In this case, a natural preconditioner is

$$M = \text{diag}(T, T, \ldots, T).$$

10. The following figure shows the convergence history of GMRES with and without preconditioning for solving a linear system of equations arising from a discretization of a model convection-diffusion equation. The preconditioner used here is ILU(0), i.e., ILU factorization with the same sparsity pattern of A.



11. Iterative methods in Matlab

functions	methods
`pcg`	Preconditioned Conjugate Gradients Method.
`gmres`	Generalized Minimum Residual Method.
`bicg`	BiConjugate Gradients Method.
`bicgstab`	BiConjugate Gradients Stabilized Method.
`cgs`	Conjugate Gradients Squared Method.
`minres`	Minimum Residual Method.
`qmr`	Quasi-Minimal Residual Method.
`symmlq`	Symmetric LQ Method.

Preconditioners

functions	preconditioners
`luinc`	Incomplete LU factorization.
`cholinc`	Incomplete Cholesky factorization.

12. Further Reading

- Yousef Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, SIAM, 2003
- H. van der Vorst, Iterative Krylov Methods for Large Linear Systems, Cambridge Univ. Press, 2003
- R. Barrett *et al*, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, 1994

13 5-4-12

13.1 Large Scale Eigenvalue Computations

Given an $n \times n$ matrix A.

$$Ax = \lambda x$$

We call λ an eigenvalue and x an eigenvector. Together, they are an eigenpair.

Methods

- single-vector (one-dimensional search) The Power Method
- Subspace projection methods: Arnoldi method, Lanczos method, Steepest Descent/Conjugate gradient method

13.2 The Power Method

Note that A has n-eigenpairs (λ_i, s_i) where (λ_1, s_1) is the dominant pair. We make an initial guess x_0. We know we can represent

$$x_0 = \gamma_1 s_1 + \gamma_s s_2 + \cdots + \gamma_n s_n.$$

Applying A to both sides yields

$$Ax_0 = \gamma_1 As_1 + \cdots + \gamma_n As_n = \gamma_1 \lambda_1 s_1 + \cdots + \gamma_n \lambda_n s_n.$$

Do this repeatedly. Then for large k, since λ_1 is dominant, we see

$$A^k x_0 \approx \gamma_1 \lambda_1^k s_1.$$

This is a simple idea, but there are lots of problems:

- Computation and normalization (dealing with overflow)
- Only get convergence in theory
- Stopping criteria = ?
- "Spectral transformation" (how do I find the second largest eigenvalue?
- Effect of rounding errors.

Ways to fix this:

1. $\hat{u}_{i+1} = Au_i$; $u_{i+1} = \hat{u}_{i+1}/\|u_{i+1}\|$. This fixes overflow.
2. To fix computation, notice that

$$\begin{aligned} u_i &= \frac{Au_{i-1}}{\|Au_{i-1}} \\ &= \frac{A^2 u_{i-2}}{\|A^2 u_{i-2}\|} \\ &= \frac{A^i x_0}{\|A^i x_0\|} \\ &= \frac{\gamma_1 \lambda_1^i s_1 + \cdots}{\|\gamma_1 \lambda_1^k s_1 + \cdots\|} \end{aligned}$$

We see for large i, this converges to $\pm s_1/\|s_1\|$, or the normalized first eigenvector and prevents overflow/underflow issues.

ECS231 Handout **May 4, 2012**

Theory

1. Let $A \in \mathcal{C}^{n\times n}$.

   (a) A scalar λ is an *eigenvalue* of an $n \times n$ A and a nonzero vector $x \in \mathcal{C}^n$ is a corresponding *right eigenvector* if
   $$Ax = \lambda x.$$
   A nonzero vector y such that $y^H A = \lambda y^H$ is a *left eigenvector*.

   (b) $\mathcal{L}_{A,\lambda} \stackrel{\text{def}}{=} \{x : Ax = \lambda x\}$ is an *eigenspace* of A corresponding to the eigenvalue λ.

   (c) The set $\lambda(A)$ of all eigenvalues of A is called the *spectrum* of A.

   (d) $p_A(\lambda) \stackrel{\text{def}}{=} \det(\lambda I - A)$, a polynomial of degree n, is called *characteristic polynomial* of A.

2. The following is a list of basic properties straightforwardly from the definition

   (a) λ is A's eigenvalue $\Leftrightarrow \lambda I - A$ is singular $\Leftrightarrow \det(\lambda I - A) = 0 \Leftrightarrow p_A(\lambda) = 0$.

   (b) There is at least one eigenvector x associated with A's eigenvalue λ; in the other word, the dimension $\dim(\mathcal{L}_{A,\lambda}) \geq 1$.

   (c) $\mathcal{L}_{A,\lambda}$ is a subspace, i.e., it has the following two properties:

   (1) $x \in \mathcal{L}_{A,\lambda} \Rightarrow \alpha x \in \mathcal{L}_{A,\lambda}$ for all $\alpha \in \mathcal{C}$.

   (2) $x_1, x_2 \in \mathcal{L}_{A,\lambda} \Rightarrow x_1 + x_2 \in \mathcal{L}_{A,\lambda}$.

   (d) Suppose A is real. λ is A's eigenvalue $\Leftrightarrow$ conjugate $\bar{\lambda}$ is also A's eigenvalue.

   (e) A is singular $\Leftrightarrow 0$ is A's eigenvalue.

   (f) If A is upper (or lower) triangular, then its eigenvalues consist of its diagonal entries.

3. $A \in \mathcal{C}^{n\times n}$ is *simple* if it has n linearly independent eigenvectors; otherwise it is *defective*.

   Examples

   (a) I and any diagonal matrices is simple. $e_1, e_2, \ldots, e_n$ *are* n linearly independent eigenvectors.

   (b) $\begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$ is simple. It has two different eigenvalues -1 and 5. By the fact that each eigenvalue corresponds to at least one eigenvector, it must have 2 linearly independent eigenvectors.

   (c) If $A \in \mathcal{C}^{n\times n}$ has n different eigenvalues, then A is simple.

   (d) $\begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}$ is defective. It has two repeated eigenvalues 2, but only one eigenvector $e_1 = (1, 0)^T$.

4. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of A, and $x_1, x_2, \ldots, x_n$ be a set of corresponding eigenvectors, then

$$AX = X\Lambda$$

where $X = [x_1, x_2, \ldots, x_n]$ and $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$.

If A is simple, namely the eigenvectors are linearly independent, then X^{-1} exists and

$$A = X\Lambda X^{-1}$$

This is known as the *eigenvalue decomposition* of the matrix A.

5. An *invariant subspace* of A is a subspace $\mathcal{V}$ of $\mathcal{R}^n$, with the property that $v \in \mathcal{V}$ implies that $Av \in \mathcal{V}$. We also write this as $A\mathcal{V} \subseteq \mathcal{V}$.

Examples:

(1) The simplest, one-dimensional invariant subspace is the set $\mathrm{span}(x)$ of all scalar multiples of an eigenvector x.

(2) Let $x_1, x_2, \ldots, x_m$ be any set of independent eigenvectors with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$. Then $\mathcal{X} = \mathrm{span}(\{x_1, x_2, \ldots, x_m\})$ is an invariant subspace.

6. Let A be n-by-n, let $V = [v_1, v_2, \ldots, v_m]$ be any n-by-m matrix with linearly independent columns, and let $\mathcal{V} = \mathrm{span}(V)$, the m-dimensional space spanned by the columns of V. Then $\mathcal{V}$ is an invariant subspace if and only if there is an m-by-m matrix B such that

$$AV = VB.$$

In this case the m eigenvalues of B are also eigenvalues of A.

7. Similarity transformations: $n \times n$ matrices A and B are *similar* if there is an $n \times n$ nonsingular matrix P such that $B = P^{-1}AP$. We also say A is *similar* to B, and likewise B is similar to A; P is a *similarity transformation*. A is *unitarily similar* to B if P is unitary.

8. Suppose that A and B are similar: $B = P^{-1}AP$.

(a) A and B have the same eigenvalues. In fact $p_A(\lambda) \equiv p_B(\lambda)$.

(b) $Ax = \lambda x \Rightarrow B(P^{-1}x) = \lambda(P^{-1}x)$.

(c) $Bw = \lambda w \Rightarrow A(Pw) = \lambda(Pw)$.

9. Schur decomposition. Let A be of order n. Then there is an $n \times n$ unitary matrix U ($U^H U = I$) such that

$$A = UTU^H,$$

where T is upper triangular. By appropriate choice of U, the eigenvalues of A, which are the diagonal elements of T, may be made to appear in any order.

10. Real Schur Decomposition. If A is real, there is an orthogonal matrix Q such that

$$A = QTQ^T,$$

where T is block triangular with 1×1 and 2×2 blocks on its diagonal. The 1×1 blocks contain the real eigenvalues of A, and the eigenvalues of the 2×2 blocks are pairs of complex conjugate eigenvalues.

The power method

1. The power method is based on the following simple analysis:

   Assume that $A = X\Lambda X^{-1}$ with $X = [x_1, x_2, \ldots, x_n]$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, and eigenvalues λ_j are ordered such that $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|$.

   Let u_0 be a vector such that $u_0 = \gamma_1 x_1 + \gamma_2 x_2 + \cdots + \gamma_n x_n$ and $\gamma_1 \neq 0$. Then we can show that

   (a) $u_j = \frac{A^j u_0}{\|A^j u_0\|} \to \pm \frac{x_1}{\|x_1\|}$ as $j \to \infty$.

   (b) $\theta_j = u_j^H A u_j \to \lambda_1$ as $j \to \infty$.

   (c) $\frac{|\lambda_2|}{|\lambda_1|}$ is the rate of convergence.

2. Pseudocode:

   Given an initial vector u_0,
   for $j = 1, 2, \ldots$ until convergence
   $w = Au_{j-1}$
   $u_j = w/\|w\|_2$
   $\theta_j = u_j^H A u_j$

3. Example. Let

$$A = \begin{bmatrix} -261 & 209 & -49 \\ -530 & 422 & -98 \\ -800 & 631 & -144 \end{bmatrix}$$

   Then $\lambda(A) = \{\lambda_1, \lambda_2, \lambda_3\} = \{10, 4, 3\}$. Let $u_0 = e_1$, by the power method, we have

i	1	2	3	$\cdots$	10
θ_i	994.49	13.0606	10.07191	$\cdots$	10.0002

4. The drawback of the power method is that if $\frac{|\lambda_2|}{|\lambda_1|}$ is close to 1, then the power method could be very slow convergent or doesn't converge at all.

The method of inverse iteration

1. The method of **Inverse iteration** has two purposes:

   (a) overcome the drawbacks of the power method (slow convergence).

   (b) find an eigenvalue closest to a particular given number σ, referred to as a *shift*).

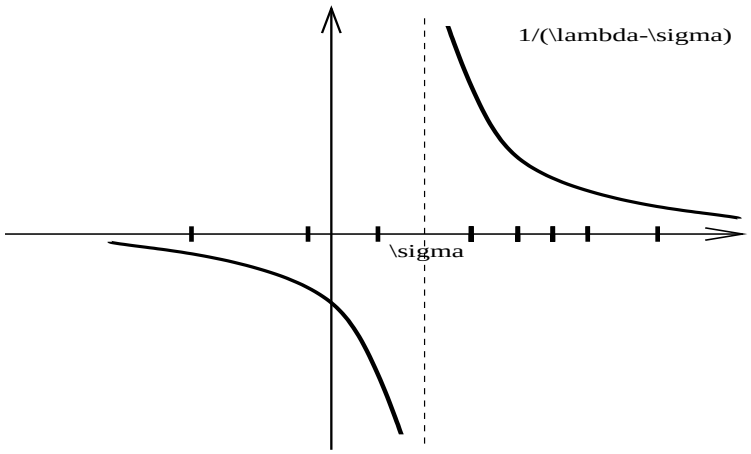
2. Spectral transformation: if λ is an eigenvalue of A, then

   (a) $\lambda - \sigma$ is an eigenvalue of $A - \sigma I$,

   (b) $\frac{1}{\lambda - \sigma}$ is an eigenvalue of $(A - \sigma I)^{-1}$.

   This is referred to as *shift-and-invert spectral transformation*.

The following plot illustrates the transformation of eigenvalues:



3. By applying the power method to the shift-and-invert eigenvalue problem

$$(A - \sigma)^{-1}x = \mu x,$$

we derive the follow algorithm, which is referred to as the *inverse iteration*:

Given an initial vector u_0 and a shift σ
for $j = 1, 2, \ldots$ until convergence
$w = (A - \sigma I)^{-1}u_{j-1}$
$u_j = \frac{w}{\|w\|_2}$ (approximate eigenvector)
$\mu_j = u_j^H A u_j$ (approximate eigenvalue)
end for
Returen approximate eigenpair of A; $(\theta_j, \sigma + \frac{1}{\mu_j})$

4. Assume λ_k is the eigenvalue cloest to the shift σ. It can be shown that

   (a) u_j converges to $x_k/\|x_k\|$, where $s_k = Se_k$ $j \to \infty$.

   (b) θ_j converges to λ_k as $j \to \infty$.

   (c) $\max_{j \neq k} \frac{|\lambda_k - \sigma|}{|\lambda_j - \sigma|}$ is the convergence rate.

5. The advantages of inverse iteration over the power method is the ability to converge to any desired eigenvalue (the one nearest to the shift σ). By choosing σ very close to a desired eigenvalue, the method converges very quickly and thus not be as limited by the proximity of nearby eigenvalues as is the power method. The method is particularly effective when we have a good approximation to an eigenvalue and want only its corresponding eigenvector.

   However, the inverse iteration is expensive in general. It requires solving $(A - \sigma I)w = j_j$ for u. One (sparse) LU factorization of $A - \sigma I$ is required, which could be very expensive in memory requirements.

14 5-11-12

14.1 Large Scale Eigenvalue Computations

Rule: A is available through matrix-vector multiplication only!

14.1.1 Method 1: The Power Method

Idea:

$$u_0 = \gamma_1 x_1 + \gamma_2 x_2 + \cdots + \gamma_n x_n$$
$$A^k u_0 = \gamma_1 \lambda_1^k x_1 + \gamma_2 \lambda_2^k x_2 + \cdots + \gamma_n \lambda_n^k x_n \xrightarrow{k\to\infty} \gamma_1 \lambda_1^k x_1$$

Eigenpairs (λ_i, x_i), with

$$|\lambda_1| > |\lambda_2| \geq \cdots$$

So λ_1 is the dominant eigenvalue.

Practical algorithm:

$$u_0$$
$$\begin{cases} \hat{u}_1 = Au_0 \\ u_1 = \frac{\hat{u}_1}{\|u_1\|} \\ \theta_1 = u_1^T A u_1 \end{cases}$$

Facts:

1. $u_k \to \pm \frac{x_1}{\|x_1\|}$ as $k \to \infty$, $\theta_k \to \lambda_1$
2. Rate: $r = \frac{|\lambda_1|}{|\lambda_1|} < 1$

"Open" issues:

1. What if we need to compute more than one eigenpairs? What if $r \approx 1 \Rightarrow$ slow convergence?
2. What if I am interested in an eigenvalue λ, which is closest to σ (a user-specified point)?

15 5-14-12

15.1 Large Scale Eigenvalue Computations

Methods:

1. The power method
2. Spectrum transformation $\rightarrow$ inverse iteration (inner-outer loops)
3. Simultaneous iterations $\Leftarrow$ slow convergence

15.1.1 (Krylov) Subspace Projection Methods

Let $\mathcal{K} \subseteq \mathbb{R}^n$. Find $\tilde{x} \in \mathcal{K}$ and $\tilde{\lambda} \in \mathbb{C}$ such that $A\tilde{x} - \tilde{\lambda}\tilde{x} \perp \mathcal{K}$. Let $V = \begin{bmatrix} v_1 & \cdots & v_m \end{bmatrix}$ be a basis of $\mathcal{K}$. So

$$\begin{aligned} \tilde{x} &= Vz \\ V^T(A\tilde{x} - \tilde{\lambda}\tilde{x}) &= 0 \\ \underbrace{V^t}_{m\times n}\underbrace{A}_{n\times n}\underbrace{V}_{)n\times m} z &= \tilde{\lambda}\underbrace{V^TV}_{=I}z \\ V^TAVz &= \tilde{\lambda}z \end{aligned}$$

This is the reduced eigenproblem.

Algorithm: (Rayleigh-Ritz procedure)

1. Select/construct V
2. Solve $V^TAVz = \tilde{\lambda}z$
3. Approximate eigenpairs $(\tilde{\lambda}, Vz) \rightarrow$ a Rayleigh-Ritz pair
4. Test for convergence

Lanczos method $\rightarrow$ a realization of the Rayleigh-Ritz subspace projection method for a symmetric matrix.

$$\begin{aligned} \mathcal{K} &= \text{ Kyrlov subspace} \\ &= \left\{x_0, Ax_0, A^2x_0, \ldots, A^{m-1}x_0\right\} \\ &\downarrow \\ &= V = \begin{bmatrix} v_1 & v_2 & \cdots & v_m \end{bmatrix} \qquad \text{(orthogonal)} \\ AV_j &= V_jT_j + \beta_{j+1}v_{j+1}e_j^T \\ V_j &= \begin{bmatrix} v_1 & v_2 & \cdots & v_j \end{bmatrix} \\ T_j &= \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_j \\ & & \beta_j & \alpha_j \end{bmatrix} \\ V_j^TAV_j &= V_j^T(V_jT_j + \beta_{j+1}v_{j+1}e_j^T) \\ &= T_j + \cancel{\beta_{j+1}V_j^Tv_{j+1}e_j^T} = T_j \end{aligned}$$

Solve

$$T_jz = \tilde{\lambda}z \rightarrow (\tilde{\lambda}, z)$$

The approximate eigenvalue is $\tilde{\lambda}$, and the approximate eigenvector is $\tilde{x} = V_j z$.

$$\begin{aligned}
\text{residual} &= A\tilde{x} - \tilde{\lambda}\tilde{x} = AV_j z - \tilde{\lambda} v_j z \\
&= (V_j T_j + \beta_{j+1} v_{j+1} e_j^T) z - \tilde{\lambda} V_j z \\
&= V_j \underbrace{T_j z}_{=\tilde{\lambda} z} + \beta_{j+1} v_{j+1} e_j^T z - \tilde{\lambda} V_j z = \beta_{j+1} v_{j+1} (e_j^T z) \\
\|\text{residual}\| &= |e_j^T z| \cdot |\beta_{j+1}| \quad \leftarrow \text{ free by-product of Lanczos process}
\end{aligned}$$

ECS231 Handout **May 14, 2012**

Rayleight-Ritz procedure

1. Rayleight-Ritz procedure is a framework of the orthogonal projection methods for solving large scale eigenvalue problems

   Let A be an $n \times n$ real matrix and $\mathcal{K}$ be an m-dimensional subspace of $\mathcal{R}^n$. An orthogonal projection technique seeks an approximate eigenpair

   $$(\tilde{\lambda}, \tilde{u}) \quad \text{with} \quad \tilde{\lambda} \in \mathcal{C} \text{ and } \tilde{u} \in \mathcal{K}.$$

   by imposing the following so-called Galerkin condition:

   $$A\tilde{u} - \tilde{\lambda}\tilde{u} \perp \mathcal{K} \ , \tag{1}$$

   or, equivalently,

   $$v^T(A\tilde{u} - \tilde{\lambda}\tilde{u}) = 0, \ \ \forall \ v \in \mathcal{K}. \tag{2}$$

   To translate this into a matrix problem, assume that an orthonormal basis $\{v_1, v_2, \ldots, v_m\}$ of $\mathcal{K}$ is available. Denote $V = [v_1, v_2, \ldots, v_m]$, and let $\tilde{u} = Vy$. Then, equation (2) becomes

   $$V^T(AVy - \tilde{\lambda}Vy) = 0$$

   Therefore, y and $\tilde{\lambda}$ must satisfy the following reduced eigenvalue problem:

   $$B_m y = \tilde{\lambda} y \tag{3}$$

   with $B_m = V^H AV$. The eigenvalues $\tilde{\lambda}_i$ of B_m are called *Ritz value* values, and the vectors Vy_i are called *Ritz vector*.

2. This procedure is known as the *Rayleigh-Ritz procedure*:

   Rayleigh-Ritz Procedure

   (a) Compute an orthonormal basis $\{v_i\}_{i=1:m}$ of the subspace $\mathcal{K}$;
   (b) Compute $B_m = V^T AV$, where $V = [v_1, v_2, \ldots, v_m]$;
   (c) Compute the eigenvalues of B_m and select the k desired ones $\tilde{\lambda}_i, i = 1 : k$, where $k \leq m$.
   (d) Compute the eigenvectors y_i of B_m associated with $\tilde{\lambda}_i$.
   (e) return $(\lambda_i, \tilde{u}_i = Vy_i)$ as approximate eigenvectors of A.

   The numerical solution of the $m \times m$ eigenvalue problem in steps (c) and (d) can be treated by standard algorithms for solving small dense eigenvalue problems. An important note is that in step (d) one can replace eigenvectors by Schur vectors to get approximate Schur vectors $\tilde{u}_i$ instead of approximate eigenvectors. Schur vectors y_i can be obtained in a numerically stable way and, in general, eigenvectors are more sensitive to rounding errors than are Schur vectors.

Further reading

3. Optimality. Consider the case where A is real and symmetric. Let $Q = [Q_k, Q_u]$ be any n-by-n orthogonal matrix, where Q_k is n-by-k, and Q_u is n-by-$(n-k)$. Let

$$T = Q^T A Q = [Q_k, Q_u]^T A [Q_k, Q_u] = \begin{bmatrix} Q_k^T A Q_k & Q_k^T A Q_u \\ Q_u^T A Q_k & Q_u^T A Q_u \end{bmatrix} \equiv \begin{bmatrix} T_k & T_{uk} \\ T_{ku} & T_u \end{bmatrix}$$

When $k = 1$, T_k is just called the Rayleigh quotient. So far $k > 1$, T_k is called a generalization of the Rayleigh quotient.

The *Rayleigh-Ritz procedure* is to approximate the eigenvalues of A by the eigenvalues of $T_k = Q_k^T A Q_k$. These approximations are called the *Ritz values*. Let $T_k = V \Lambda V^T$ be the eigendecomposition of T_k. The corresponding eigenvector approximations are the columns of $Q_k V$ and are called *Ritz vectors*.

The Ritz values and Ritz vectors are considered *optimal* approximations to the eigenvalues and eigenvectors of A as justified by the following theorem.

Theorem. The minimum of $\|AQ_k - Q_k R\|_2$ over all k-by-k symmetric matrices R is attained by $R = T_k$, in which case, $\|AQ_k - Q_k T_k\|_2 = \|T_{ku}\|_2$.

PROOF: Let $R = T_k + Z$, to proof the theorem, we just want to show that $\|AQ_k - Q_k R\|_2$ is minimized when $Z = 0$. This is shown by the following sequence of derivation:

$$\begin{aligned}
\|AQ_k - Q_k R\|_2^2 &= \lambda_{\max}\left[(AQ_k - Q_k R)^T (AQ_k - Q_k R)\right] \\
&= \lambda_{\max}\left[(AQ_k - Q_k (T_k + Z))^T (AQ_k - Q_k (T_k + Z))\right] \\
&= \lambda_{\max}\Big[(AQ_k - Q_k T_k)^T (AQ_k - Q_k T_k) - ((AQ_k - Q_k T_k)^T (Q_k Z) \\
&\qquad -(Q_k Z)^T (AQ_k - Q_k T_k) + (Q_k Z)^T (Q_k Z)\Big] \\
&= \lambda_{\max}\Big[(AQ_k - Q_k T_k)^T (AQ_k - Q_k T_k) - (Q_k^T A Q_k - T_k) Z \\
&\qquad -Z^T (Q_k^T A Q_k - T_k) + Z^T Z\Big] \\
&= \lambda_{\max}\left[(AQ_k - Q_k T_k)^T (AQ_k - Q_k T_k) + Z^T Z\right] \\
&\geq \lambda_{\max}\left[(AQ_k - Q_k T_k)^T (AQ_k - Q_k T_k)\right] \\
&= \|AQ_k - Q_k T_k\|_2^2
\end{aligned}$$

Furthermore, it is easy to compute the minimum value

$$\|AQ_k - Q_k T_k\|_2 = \|(Q_k T_k + Q_u T_{ku}) - Q_k T_k\|_2 = \|Q_u T_{ku}\|_2 = \|T_{ku}\|_2. \qquad \blacksquare$$

Corollary. Let $T_k = Y \Lambda Y^T$ be the eigendecomposition of T_k. The minimum of $\|AP_k - P_k D\|$ over all n-by-k orthogonal matrices P_k where $\text{span}(P_k) = \text{span}(Q_k)$ and over all diagonal D is also $\|T_{ku}\|_2$ and is attained by $P_k = Q_k Y$ and $D = \Lambda$.

PROOF: If we replace Q_k with $Q_k U$ in the above proof, where U is another orthogonal matrix, then the columns of Q_k and $Q_k U$ span the same space, and

$$\|AQ_k - Q_k R\|_2 = \|AQ_k U - Q_k R U\|_2 = \|A(Q_k U) - (Q_k U)(U^T R U)\|_2.$$

These quantities are still minimized when $R = T_k$, and by choosing $U = Y$ so that $U^T T_k U$ is diagonal. $\blacksquare$

Lanczos algorithm

1. The Lanczos algorithm combines the Lanczos process for building a Krylov subspace with the Raleigh-Ritz procedure for for finding a few eigenpairs of a symmetric matrix A. First, let us recall that the Lanczos process will generate an orthonormal basis of a Krylov subspace:

$$\mathcal{K}_k(A, v) \stackrel{\text{def}}{=} \text{span}\{v, Av, \ldots, A^{k-1}v\} = \text{span}\{q_1, q_2, \ldots, q_k\},$$

and yield a fundamental relation

$$AQ_k = Q_k T_k + f_k e_k^T, \quad f_k = \beta_k q_{k+1} \tag{4}$$

where $T_k = Q_k^T A Q_k = \text{tridiag}(\beta_j, \alpha_j, \beta_{j+1})$. Let μ be an eigenvalue of T_k and y be a corresponding eigenvector y, i.e.,

$$T_k y = \mu y, \quad \|y\|_2 = 1.$$

Apply y to the right of (4) to get

$$A(Q_k y) = Q_k T_k y + f_k(e_k^T y) = \mu(Q_k y) + f_k(e_k^T y).$$

$\{\mu\}$ are *Ritz values*, and $\{Q_k y\}$ are *Ritz vectors*.

2. Convergence

- If $f_k(e_k^T y) = 0$ for some k, then the associated Ritz value μ is an eigenvalue of A with the corresponding eigenvector $Q_k y$.
- In general, it is unlikely that $f_k(e_k^T y) = 0$, but we hope that the residual norm $\|f_k(e_k^T y)\|_2$ may be small; and when this happens we expect that μ is going to be a good approximate to A's eigenvalue. Indeed, we have

  Lemma 1 *Let H be (real) symmetric, and $Hz - \mu z = r$ and $z \neq 0$. Then*

$$\min_{\lambda \in \lambda(H)} |\lambda - \mu| \leq \|r\|_2 / \|z\|_2.$$

  PROOF: Let $H = U \Lambda U^T$ be the eigen-decomposition of H. Then $Hz - \mu z = r$ yields

$$(H - \mu I)z = r \quad \Rightarrow \quad U(\Lambda - \mu I)U^T z = r \quad \Rightarrow \quad (\Lambda - \mu I)(U^T z) = U^T r.$$

  Notice that $\Lambda - \mu I$ is diagonal. Thus

$$\|r\|_2 = \|U^T r\|_2 = \|(\Lambda - \mu I)(U^T z)\|_2 \geq \min_{\lambda \in \lambda(H)} |\lambda - \mu| \, \|U^T z\|_2 = \min_{\lambda \in \lambda(H)} |\lambda - \mu| \|z\|_2,$$

  as expected. ∎

  The following corollary is a consequence of above Lemma 1.

  Corollary 1 *There is an eigenvalue λ of A such that*

$$|\lambda - \mu| \leq \|f_k(e_k^T y)\|_2 = |\beta_k| \cdot |e_k^T y|.$$

3. In summary, we have the following Lanczos algorithm in the simplest form:

   LANCZOS ALGORITHM for finding eigenvalues and eigenvectors of $A = A^T$:

   1. $q_1 = v/\|v\|_2$, $\beta_0 = 0$; $q_0 = 0$;
   2. for $j = 1$ to k, do
   3. $\quad w = Aq_j$;
   4. $\quad \alpha_j = q_j^T w$;
   5. $\quad w = w - \alpha_j q_j - \beta_{j-1} q_{j-1}$;
   6. $\quad \beta_j = \|w\|_2$;
   7. $\quad$ if $\beta_j = 0$, quit;
   8. $\quad q_{j+1} = w/\beta_j$;
   9. $\quad$ Compute eigenvalues and eigenvectors of T_j
   10. $\quad$ Test for convergence
   11. EndDo

   Caveat: All the discussion in this lecture is under the assumption of exact arithmetic. In the presence of finite precision arithmetic, the numerical behaviors of the Lanczos algorithm could be significantly different. For example, in finite precision arithmetic, the orthogonality of the computed Lanczos vectors $\{q_j\}$ is lost when j is as small as 10 or 20. The simplest remedy (and also the most expensive one) is to implement the the full reorthogonalization, namely after the step 5, do

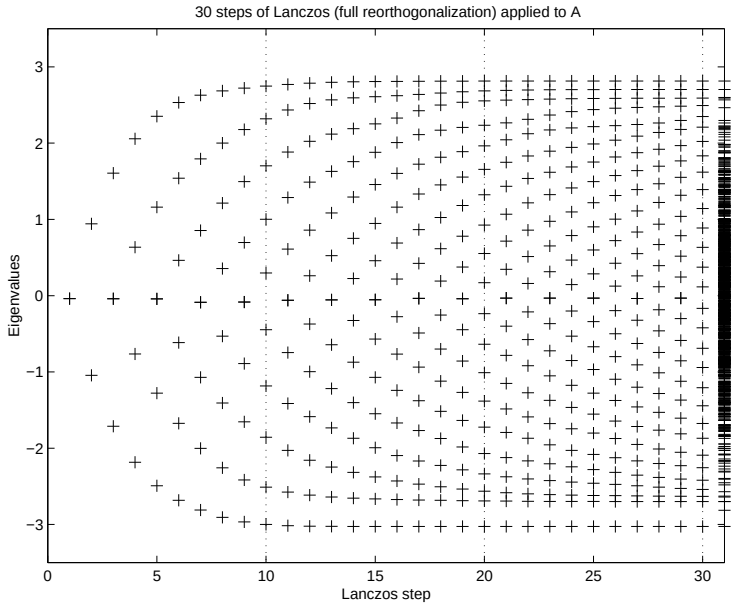
$$w = w - \sum_{i=1}^{j-1} (w^T q_i) q_i.$$

   This is called the Lanczos algorithm with full reorthogonalization. (Sometimes, it may be needed to execute *twice*). A more elaborate scheme, necessary when convergence is slow and several eigenvalues are sought, is to use the selective orthogonalization.

4. Example. We illustrate the Lanczos algorithm by a running an example, a 1000-by-1000 diagonal matrix A, most of whose eigenvalues were chosen randomly from a normal Gaussian distribution. To make the plot easy to understand, we have also sorted the diagonal entries of A from largest to smallest, so $\lambda_i(A) = a_{ii}$ with the corresponding eigenvector e_i. There are a few extreme eigenvalues, and the rest cluster near the center of the spectrum. The starting Lanczos vector v has all equal entries.

   There is no loss in generality in experimenting with a diagonal matrix, since running the Lanczos algorithm on A with starting vector $q_1 = v/\|v\|_2$ is equivalent to running the Lanczos algorithm on $Q^T A Q$ with starting vector $Q^T q_1$.

   The following figure illustrates convergence of the Lanczos algorithm for computing the eigenvalues of A. In this figure, the eigenvalues of each T_k are shown plotted in column k, for $k = 1, 2, 3, \ldots, 30$, with the eigenvalues of A plotted in an extra column at the rightmost column. The column k has k "+"s, one marking each eigenvalues of T_k.



We observe that:

- Extreme eigenvalues, i.e., the largest and smallest ones, converge first, and the interior eigenvalues converge last.
- Convergence is monotonic, with the ith largest (smallest) eigenvalues of T_k increasing (decreasing) to the ith laregst (smallest) eigenvalue of A, provided that the Lanczos algorithm does not stop prematurely with some $\beta_k = 0$.

5. An excellent reference to study the observation in theory is the book by B. N. Parlett, "The Symmetric Eigenvalue Problem", reprinted by SIAM, 1998.

16 5-16-12

16.1 Large Scale Eigenvalue Computations

$$Ax = \lambda x,$$

A is a sparse $n \times n$ matrix. Dimension reduction/subspace projection $\Rightarrow$ find $\tilde{x} \in \mathcal{K} \subset \mathbb{R}^n$ and $\tilde{\lambda} \in \mathbb{C}$ such that $A\tilde{x} - \tilde{\lambda}\tilde{x} \perp \mathcal{K}$.

In practice:

$$\begin{aligned}\mathcal{K} = &\text{ Krylov subspace}\\ &= \{x_0, Ax_0, A^2x_0, \ldots, A^{m-1}x_0\}\\ &\downarrow\\ &= V_m = \begin{bmatrix} v_1 & v_2 & \cdots & v_m \end{bmatrix}, \qquad V_m^T V_m = I\end{aligned}$$

- Gram-Schmidt
  - Lanczos for $A^T = A$
  - Arnoldi for $A^T \neq A$

ECS231 Handout **May 16, 2012**

Arnoldi algorithm

1. The power method is the simplest algorithm suitable for computing just the largest eigenvalue in absolute value, along with its eigenvector. Starting with a given x_0, k iterations of the power method produce a sequence of vectors $x_0, x_1, x_2, \ldots, x_k$. It is easy to see that these vectors span a *Krylov Subspace*:

$$\text{span}\{x_0, x_1, x_2, \ldots, x_k\} = \mathcal{K}_{k+1}(A, x_0) = \text{span}\{x_0, Ax_0, A^2x_0, \ldots, A^kx_0\}.$$

   Now, rather than taking x_k as out approximate eigenvector, it is natural to ask for the "best" approximate eigenvector in $\mathcal{K}_{k+1}(A, x_0)$ using the Rayleigh-Ritz procedure. We will see that the best eigenvector (and eigenvalue) approximations from $\mathcal{K}_{k+1}(A, x_0)$ are much better than x_k alone.

2. The Arnoldi algorithm for finding a few eigenpairs of a general matrix A combines the Arnoldi process for building a Krylov subspace with the Raleigh-Ritz procedure.

   First, let us recall that the following Arnoldi process generates an orthonormal basis of a Krylov subspace $\mathcal{K}_k(A, v)$:

$[V_{m+1}, \widehat{H}_m] = \text{arnoldi}(A, v, k)$
1. $v_1 = v/\|v\|_2$
2. for $j = 1, 2, \ldots, k$
3. compute $w = Av_j$
4. for $i = 1, 2, \ldots, j$
5. $h_{ij} = v_i^T w$
6. $w := w - h_{ij}v_i$
7. end for
8. $h_{j+1,j} = \|w\|_2$
9. If $h_{j+1,j} = 0$, *stop*
10. $v_{j+1} = w_j/h_{j+1,j}$
11. endfor

   The Arnoldi process yields the fundamental relation, referred to as an Arnoldi decomposition of length k:

$$AV_k = V_kH_k + h_{k+1,k}v_{k+1}e_k^T, \tag{1}$$

   where H_k is Hessenberg, $V_k^HV_k = I$, and $V_k^Hv_{k+1} = 0$. If H_k is unreduced and $h_{k+1,k} \neq 0$, the decomposition is uniquely determined by the starting vector v (This is commonly called implicit Q-Theorem).

3. Since $V_k^Hv_{k+1} = 0$, we have

$$H_k = V_k^TAV_k.$$

   Let μ be an eigenvalue of H_k and y be a corresponding eigenvector y, i.e.,

$$H_ky = \mu y,$$

Then the corresponding Ritz pair is $(\mu, Q_k y)$. Applying y to the right of (1), the residual vector of $(\mu, V_k y)$ is given by

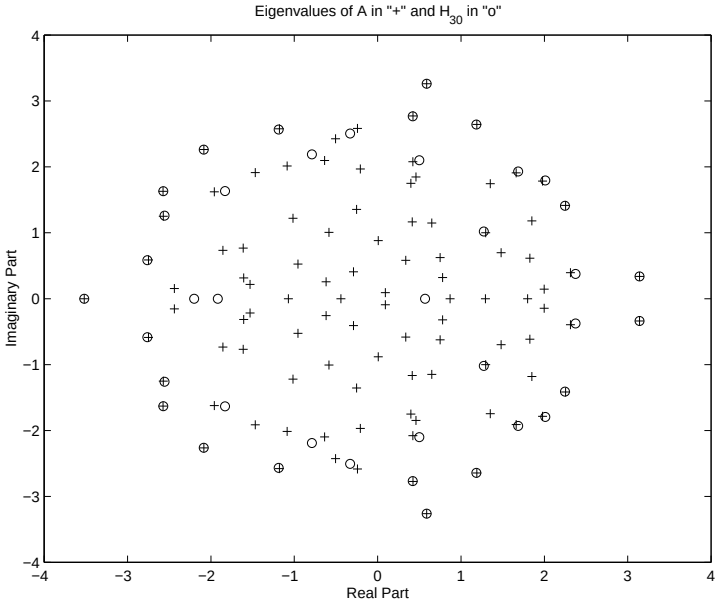
$$A(V_k y) - \mu(V_k y) = h_{k+1,k} v_{k+1}(e_k^T y).$$

Using the backward error interpretation, we know that $(\mu, V_k y)$ is an exact eigenpair of $A+E$, where $\|E\|_2 = |h_{k+1,k}| \cdot |e_k^T y|$.

4. This gives us a criterion for accepting the Ritz pair $(\mu, V_k y)$ as approximate eigenpair[1] of A.

ARNOLDI'S METHOD
1. Choose a starting vector v
2. Generate the Arnoldi decomposition of length k by the Arnoldi process
3. Compute the Ritz pairs and decide which are acceptable
4. If necessary, increase k and repeat

5. Example. We illustrate the above simplest Arnoldi algorithm by a running a 100-by-100 random sparse matrix A with approximately 1000 normally distributed nonzero entries, $A =$ `sprandn(100,100,0.1)`. All entries of the starting vector v are 1. The following figure illustrates typical convergence behavior of the Arnoldi algorithm for computing the eigenvalues. In the figure, "+" are the eigenvalues of matrix A (computed by `eig(full(A))`). and the "$\circ$" are the eigenvalues of upper Hessenberg matrix H_{30} (computed by `eig(`H_{30}`)`).



We observe that *Exterior eigenvalues converge first.* This is the typical convergence phenomenon of the Arnoldi algorithm (in fact, all Krylov subspace based methods). There is a general theory for the convergence analysis of the Arnoldi algorithm.

6. The Arnoldi algorithm has two nice aspects:

[1]Note that because of non-symmetry of A, we generally do not have the nice forward error estimation as discussed in the Lanczos algorithm for symmetric eigenproblem. But a similar error bound involving the condition number of the corresponding eigenvalue exists.

(a) The matrix H_k is already in Hessenberg form, so that we can immediately apply the QR algorithm to find its eigenvalues.

(b) After we increase k, say $k + p$, we only have to orthogonalize p vectors to compute the $(k+p)$th Arnoldi decomposition. The work we have done previously is not thrown away.

Unfortunately, the algorithm has its drawbacks:

- If A is large we cannot increase k indefinitely, since V_k requires $n \times k$ memory locations to store.
- We have little control over which eigenpairs the algorithm finds. In a given application, we will be interested in a certain set of eigenpairs. For example, eigenvalues lying near the imaginary axis. There is nothing in the algorithm to force desired eigenvectors into the subspace or the discard undesired ones.

These issues have been successfully addressed to some extent by a so-called *implicitly restart* scheme, see

- D. Sorensen, Implicit application of polynomial filters in a k-step Arnoldi method, SIAM J. Matrix Anal. Appl., Vol. 13, pp.357–385, 1992.
- Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, editors, Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide. SIAM, Philadelphia, 2000 Available at `http://www.cs.ucdavis.edu/~bai/ET/contents.html`

17 5-23-12

http://bayou.cs.ucdavis.edu

18 6-1-12

18.1 Fast Solvers

18.1.1 Poisson's Equation

1-D:

$$-\frac{d^2v}{dt^2} = f(t), \qquad 0 < t < 1$$
$$v(0) = v(1) = 1$$

2-D:

$$-\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2} = f(x,y)$$
$$-\Delta u = f$$

18.2 Kronecker ("Tensor") Product of Matrices

Given matrices A $(m \times n)$ and B $(p \times q)$, the Kronecker product is

$$\underbrace{A \otimes B}_{(mp)\times(nq)} = (a_{ij}B) = \texttt{kron(A,B)}$$

Properties

- $(A \otimes B) \cdot (C \otimes D) = (AC) \otimes (BD)$
- $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$
- $(A \otimes B)^T = A^T \otimes B^T$

18.3 Vectorization

$$\underbrace{A}_{m\times n} \to \texttt{vec(A)}$$

$$A = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix} \to \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}_{mn\times 1} = \texttt{vec(A)}$$

Properties

- $\texttt{vec(AX)} = (I \otimes A) \cdot \texttt{vec(X)}$
  This enables us to rewrite the matrix equation $AX = B$ as a vector equation: $(I \otimes A) \cdot \texttt{vec(X)} = \texttt{vec(B)}$
- $\texttt{vec(AB)} = (B^T \otimes I) \cdot \texttt{vec(X)}$

Applications

- Sylvester matrix equation.

$$\underbrace{A}_{n\times n}\underbrace{X}_{n\times m} + X\underbrace{B}_{m\times m} = C$$
$$\texttt{vec(AX+XB)} = \texttt{vec(C)}$$
$$(I \otimes A)\texttt{vec(X)} + (B^T \otimes I)\texttt{vec(X)} = \texttt{vec(C)}$$
$$\underbrace{(I \otimes A + B^T \otimes I)}_{\hat{A}}\underbrace{\texttt{vec(X)}}_{\hat{x}} = \underbrace{\texttt{vec(C)}}_{\hat{b}}$$

  A special case are Lyapunov equations, where $B = A^T$.

ECS231 Handout **June 1, 2012**

Kronecker product.

1. Let $A = (a_{ij})$ be $m \times n$ and $B = (b_{ij})$ be $p \times q$, then the *Kronecker product* of A and B are defined as

$$A \otimes B = (a_{ij}B) = \begin{pmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{pmatrix}.$$

   $C = A \otimes B$ is a $(mp) \times (nq)$ matrix.

2. Kronecker product has the following basic properties:

   - Assume AC and BD are well defined, then $(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D)$
   - If A and B are invertible, $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
   - $(A \otimes B)^T = A^T \otimes B^T$

3. Let $\mathrm{vec}(X)$ be defined to be a column vector of length $m \cdot n$ made of the columns of an $m \times n$ matrix X stacked atop one another from left to right, i.e.,

$$\mathrm{vec}(X) = \mathrm{vec}([x_1, x_2, \ldots, x_n]) = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix},$$

   then we have

   - $\mathrm{vec}(AX) = (I_n \otimes A) \cdot \mathrm{vec}(X)$
   - $\mathrm{vec}(XB) = (B^T \otimes I_m) \cdot \mathrm{vec}(X)$

Discretization of Poisson's equations by finite differences.

1. One-dimensional Poisson's equation takes the form

$$-\frac{d^2v(x)}{dx^2} = f(x), \qquad 0 < x < 1 \tag{1}$$

   with Dirichlet boundary conditions:

$$v(0) = v(1) = 0, \tag{2}$$

   where $f(x)$ is a given function and $v(x)$ is the unknown function to be computed. The Possion equation models the displacement of an elastic bar or cord in continuum mechanics, the temperature distribution in a heat conducting bar.

(a) Let us *discretize* Poisson's equation by trying to compute an approximate solution $N+2$ evenly spaced point x_i between 0 and 1:

$$0 = x_0 < x_1 < x_2 < \cdots < x_N < x_{N+1} = 1$$

and

$$x_i = x_0 + ih = ih, \qquad h = \frac{1}{N+1}.$$

The points x_0 and x_{N+1} are called boundary points, and are known. x_i for $i = 1, 2, \ldots, N$ are called interior points and are unknown.

Denoting $v_i = v(x_i)$ and $f_i = f(x_i)$ and using the 3-point centered difference approximation, at $x = x_i$, we have

$$-\frac{d^2v(x)}{dx^2} = \frac{-v_{i-1} + 2v_i - v_{i+1}}{h^2} + \tau_i,$$

where the truncation error $\tau_i = O(h^2)$ (assuming $v(x)$ is smooth enough). Therefore at $x = x_i$, $0 < i < N+1$, we have

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i + h^2 \tau_i$$

and $v_0 = v_{N+1} = 0$.

In matrix notation, let

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix}, \quad \bar{\tau} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_N \end{bmatrix} \quad \text{and} \quad T_N = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix},$$

then we have

$$T_N \, v = h^2 f + h^2 \bar{\tau} \tag{3}$$

To solve this equation, let us ignore $\bar{\tau}$, since it is expected to be small compared to f, then we have the linear system of equations

$$T_N \, \hat{v} = h^2 f, \tag{4}$$

where $\hat{v}$ is an approximation of v.

(b) The tridiagonal matrix $T_N = \text{tridiag}(-1, 2, -1)$ has the following explicit eigenvalue decomposition

$$T_N = Z_N \Lambda_N Z_N^T,$$

where $Z_N = [z_1, z_2, \ldots, z_N]$ and $\Lambda_N = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$,

- $\lambda_j = 2(1 - \cos\frac{\pi j}{N+1})$ for $j = 1, 2, \ldots, N$ are the eigenvalues of T_N.
- z_j are the eigenvectors for $j = 1, 2, \ldots, N$. The kth entry of z_j is given by $z_j(k) = \sqrt{\frac{2}{N+1}} \sin(\frac{\pi kj}{N+1})$ for $k = 1, 2, \ldots, N$.
- Since $\lambda_j > 0$ for all j, T_N is symmetric positive definite.
- Z is orthogonal.

The largest eigenvalue of T_N is λ_N and for large N,

$$\lambda_N = 2(1 - \cos\frac{N\pi}{N+1}) \approx 4$$

and the smallest eigenvalue is λ_1 and for large N,

$$\lambda_1 = 2(1 - \cos\frac{\pi}{N+1}) \approx 2\left(1 - \left(1 - \frac{\pi^2}{2(N+1)^2}\right)\right) = \frac{\pi^2}{(N+1)^2}.$$

Therefore, the condition number of T_N is

$$\text{cond}(T_N) = \|T_N\|_2\|T_N^{-1}\|_2 = \frac{\lambda_N}{\lambda_1} \approx \frac{4(N+1)^2}{\pi^2} = \mathcal{O}(h^{-2}) \quad \text{for large } N.$$

(c) Now we can bound the error $v - \hat{v}$, subtracting equation (4) from equation (3), we have

$$v - \hat{v} = h^2 T_N^{-1}\,\bar{\tau}.$$

By taking norm and assuming that v is sufficient smooth (the required derivatives are bounded), we have

$$\|v - \hat{v}\|_2 \le h^2\|T_N^{-1}\|_2\|\bar{\tau}\|_2 \approx h^2\frac{(N+1)^2}{\pi^2}\|\bar{\tau}\|_2 = O(\|\bar{\tau}\|_2) = O(h^2).$$

In the rest of this note, we will not distinguish between v and its approximation $\hat{v}$ and so will simplify notation by letting

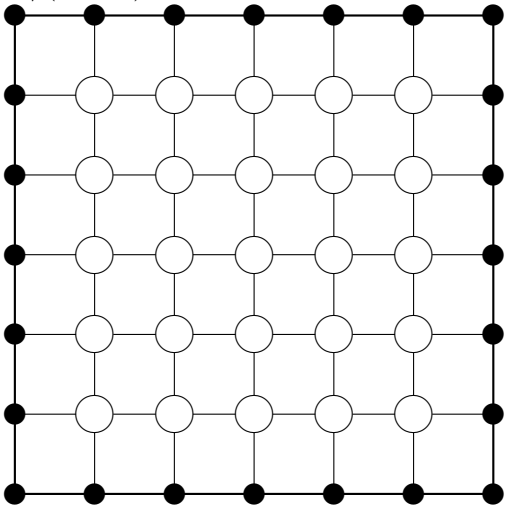
$$T_N\, v = h^2 f.$$

2. Two-dimensional Poisson's equation takes the form

$$\begin{aligned} -\nabla^2 v(x,y) = -\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) v(x,y) &= f(x,y) \quad \text{for } (x,y) \in \Omega, \\ v(x,y) &= \phi(x,y) \quad \text{for } (x,y) \in \partial\Omega, \end{aligned}$$

where $\Omega = (0,1) \times (0,1)$, the unit square and $\partial\Omega$ is its boundary.

(a) To discretize the differential equation, the domain Ω is covered with a grid of step size $h = 1/(N+1)$ as follows.



This is an example grid with $N = 5$. The values $v(x,y)$ at the boundary grid points ● is given by $\phi(x,y)$, and the values $v(x,y)$ at interior grid points ○ are to be sought.

Each grid point (x_i, y_j) have the representation

$$x_i = ih \quad \text{and} \quad y_j = jh \quad \text{for } i,\, j = 0, 1, \ldots, N+1.$$

Those points with one of i and j being $i = 0$ or $N+1$ are the *boundary grid points*; all other points are the *interior grid points*. We seek approximations to $v(x_i, y_j)$ for all the interior grid points. Write

$$v_{ij} = v(x_i, y_j), \quad f_{ij} = f(x_i, y_j), \quad \text{and} \qquad \phi_{ij} = \phi(x_i, y_j).$$

To this end, we do approximately at each interior grid point:

$$\begin{aligned} -\frac{\partial^2 v}{\partial x^2}\bigg|_{\text{at } (x_i, y_j)} &\approx \frac{-v_{i-1\,j} + 2v_{ij} - v_{i+1\,j}}{h^2}, \\ -\frac{\partial^2 v}{\partial y^2}\bigg|_{\text{at } (x_i, y_j)} &\approx \frac{-v_{i\,j-1} + 2v_{ij} - v_{i\,j+1}}{h^2}. \end{aligned}$$

Adding these approximations we have

$$-\frac{\partial^2 v}{\partial x^2} - \frac{\partial^2 v}{\partial y^2}\bigg|_{\text{at } (x_i, y_j)} = \frac{-v_{i-1\,j} - v_{i\,j-1} + 4v_{ij} - v_{i+1\,j} - v_{i\,j+1}}{h^2} + \tau_{ij}$$

where τ_{ij} is a truncation error. By Taylor expansion, it is easy to show that it is at the order of h^2, $O(h^2)$. Ignoring the truncation errors, we arrive at the linear equations in the unknowns v_{ij},

$$-v_{i-1\,j} - v_{i\,j-1} + 4v_{ij} - v_{i+1\,j} - v_{i\,j+1} = h^2 f_{ij}, \tag{5}$$

for $1 \le i, j \le N$. The left-hand side of which is 4 times the v at the point subtracting the v at the four neighbor points. This is called *5-point centered difference* or *5-point stencil*. Notice that the boundary points

$$v_{0j} = \phi_{0j}, \quad v_{0\,N+1} = \phi_{0\,N+1}, \quad v_{i0} = \phi_{i0}, \quad v_{i\,N+1} = \phi_{i\,N+1}$$

are known and the unknowns are for $0 < i, j < N+1$; so there are N^2 of them. By collecting all v_{ij} to form an $N \times N$ matrix V whose (i,j)th entry is v_{ij}:

$$V = (v_{ij})$$

and define an $N \times N$ matrix $\widetilde{F}$ by

$$h^2(\widetilde{F})_{ij} = \begin{cases} h^2 f_{ij}, & \text{for } 2 \le i, j \le N-1, \\ h^2 f_{ij} + \phi_{i\,j-1}, & \text{for } 2 \le i \le N-1 \text{ and } j = 1, \\ h^2 f_{ij} + \phi_{i\,j+1}, & \text{for } 2 \le i \le N-1 \text{ and } j = N, \\ h^2 f_{ij} + \phi_{i-1\,j}, & \text{for } i = 1 \text{ and } 2 \le j \le N-1, \\ h^2 f_{ij} + \phi_{i+1\,j}, & \text{for } i = N \text{ and } 2 \le j \le N-1, \\ h^2 f_{ij} + \phi_{i\,j-1} + \phi_{i-1\,j}, & \text{for } (i,j) = (1,1), \\ h^2 f_{ij} + \phi_{i\,j-1} + \phi_{i+1\,j}, & \text{for } (i,j) = (N,1), \\ h^2 f_{ij} + \phi_{i-1\,j} + \phi_{i\,j+1}, & \text{for } (i,j) = (1,N), \\ h^2 f_{ij} + \phi_{i\,j+1} + \phi_{i+1\,j}, & \text{for } (i,j) = (N,N). \end{cases}$$

then it can be verified that the (5) becomes

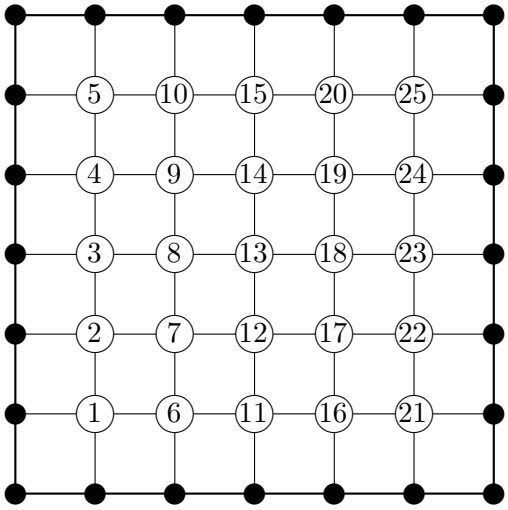
$$T_N \cdot V + V \cdot T_N = h^2 \widetilde{F}, \tag{6}$$

where $T_N = \text{tridiag}(-1, 2, -1)$. Note that care should be taken for the grid points that are neighbors of boundary grid points.

(b) Lexicographic (natural) ordering: the system (6) is not in the familiar form "$Ax = b$" of linear system of equations because all the unknowns are compactly stored into a matrix. To reorganize equations (5) in a way that leads to the $Ax = b$ form, we need to arrange v_{ij} into a column vector. A natural way would be arranging one column of V on top of another, i.e., defining a N^2-dimensional vector v as (in MATLAB-like notation)

$$v = [V(:,1); V(:,2); \ldots; V(:,N)] \equiv \mathrm{vec}(V).$$

Such an ordering of v_{ij} is best described by the following picture in the case of $N = 5$.



Define also N^2-dimensional vector $\widetilde{f}$ from the matrix $\widetilde{F}$ analogously. The system (6) becomes

$$Av = h^2 \widetilde{f}, \tag{7}$$

where

$$A = \begin{pmatrix} T_N + 2I_N & -I_N & & & & \\ -I_N & T_N + 2I_N & -I_N & & & \\ & \ddots & \ddots & \ddots & & \\ & & -I_N & T_N + 2I_N & -I_N & \\ & & & -I_N & T_N + 2I_N & \end{pmatrix}.$$

In fact, A is the Kronecker products of T_N and I_N:

$$A = I_N \otimes T_N + T_N \otimes I_N \equiv T_{N\times N}.$$

(c) Using the Kronecker product and the eigenvalue decomposition of the tridiagonal matrix T_N, we immediately derive the eigenvalue decomposition of the matrix $T_{N\times N}$:

Let $T_N = Z_N \Lambda_N Z_N^T$ be the eigendecomposition of the tridiagonal matrix T_N. Then the eigendecomposition of $T_{N\times N}$ is given by

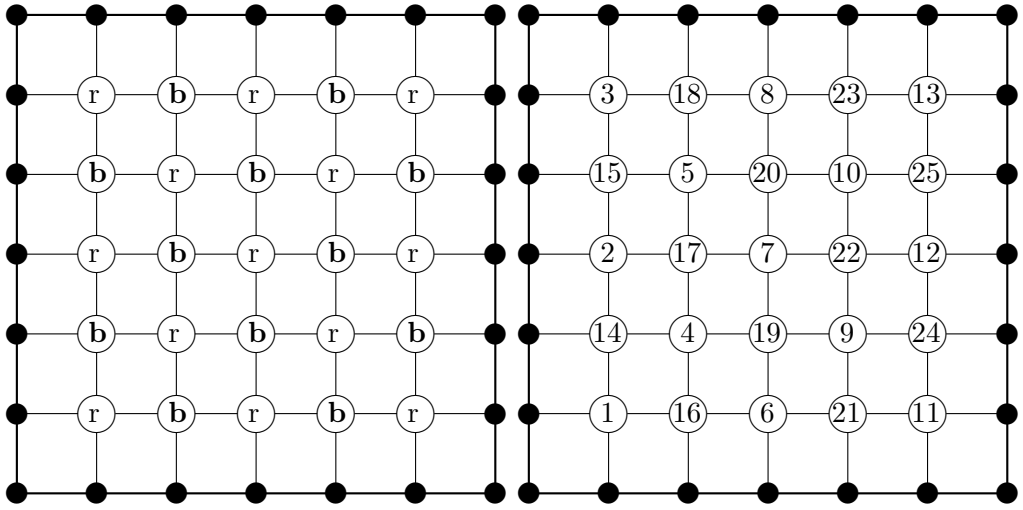
$$\begin{aligned} T_{N\times N} &= I_N \otimes T_N + T_N \otimes I_N \\ &= (Z_N \otimes Z_N)(I_N \otimes \Lambda_N + \Lambda_N \otimes I_N)(Z_N \otimes Z_N)^T. \end{aligned}$$

By the eigenvalue decomposition of $T_{N\times N}$, we know that eigenvalues λ_{ij} of the Poisson matrix $T_{N\times N}$ are given by

$$\lambda_{i\,j} \stackrel{\mathrm{def}}{=} \lambda_i + \lambda_j = 2(2 - \cos i\pi h - \cos j\pi h) \tag{8}$$

$i, j = 1, 2, \ldots, N$, where λ_i and λ_j are the eigenvalues of T_N. Note that $h = 1/(N+1)$.

(d) Red-black Ordering: first color all nodes by either *red* or *black* in such a way that no neighbor nodes share the same color; and then enumerate all nodes with one color and then all nodes with the other. Such an ordering of v_{ij} is best described by the following picture in the case of $N = 5$.



Let $v_{\rm rb}$ and $\widetilde{f}_{\rm rb}$ be the N^2-dimensional vectors obtained from V and $\widetilde{F}$ with this red-black ordering. The system (6) becomes

$$A_{\rm rb} v_{\rm rb} = h^2 \widetilde{f}_{\rm rb}, \quad A_{\rm rb} = \begin{pmatrix} D_r & B \\ B^T & D_b \end{pmatrix}, \tag{9}$$

both D_r and D_b are diagonal matrices with all diagonal entries being 4. B is a sparse matrix with nonzero entries -1 (the details of the structure of B is not important for us now).

Notice that $A_{\rm rb}$ is consistently ordered and has eigenvalues given by (8).

3. Three-dimensional Poisson's equations takes the form

$$\begin{aligned} -\nabla^2 v(x,y,z) = -\left(\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial y^2} - \frac{\partial^2}{\partial z^2}\right) v(x,y,z) &= f(x,y,z) \quad \text{for } (x,y,z) \in \Omega, \\ v(x,y,z) &= \phi(x,y,z) \quad \text{for } (x,y,z) \in \partial\Omega, \end{aligned}$$

where $\Omega = (0,1) \times (0,1) \times (0,1)$, the unit cubic and $\partial\Omega$ is its boundary

Using a 7-point centered finite difference on a cubic grid of step size $h = 1/(N+1)$, with natural ordering, it leads to the linear system of equations $Av = b$, where the coefficient matrix

$$A = T_{N\times N\times N} = T_N \otimes I_N \otimes I_N + I_N \otimes T_N \otimes I_N + I_N \otimes I_N \otimes T_N$$

It can be shown that A's eigenvalues are all possible triple sum of the eigenvalues of T_N and the eigenvector matrix is $Z_N \otimes Z_N \otimes Z_N$.

4. Poisson's equation in higher dimensions is represented analogously.

19 6-4-12

19.1 Fast Solvers

- $O(n^3)$
  - LU Factorization
    - If $A^T = A > 0$, use Cholesky
- $O(n^{3/2})$
  - CG/Krylov subspace iterative methods
    - Matrix-vector product: $5n$
- Fast solvers
  - Block Cyclic Reduction (BCR) $\Rightarrow O(n \log n)$
  - Fast Fourier Transform (FFT) $\Rightarrow O(n \log n)$
  - Multi-grid $\Rightarrow O(n)$

19.2 Block Cyclic Reduction

$$\begin{aligned}
\underbrace{A}_{N^2 \times N^2} &= I \otimes T + T \otimes I \\
&= \begin{pmatrix} T & & & \\ & T & & \\ & & \ddots & \\ & & & T \end{pmatrix} + \begin{pmatrix} 2I & -I & & \\ -I & 2I & -I & \\ & \ddots & \ddots & \ddots \\ & & & 2I \end{pmatrix} \\
&= \begin{pmatrix} T+2I & -I & & \\ -I & \ddots & \ddots & \\ & & & -I \\ & & -I & T+2I \end{pmatrix} \\
A \begin{pmatrix} \frac{x_1}{x_2} \\ \hline \vdots \\ \hline x_N \end{pmatrix} &= \begin{pmatrix} \frac{b_1}{b_2} \\ \hline \vdots \\ \hline b_N \end{pmatrix}
\end{aligned}$$

Let $N = q,\ C = T + 2I$. Then

$$A = \begin{pmatrix} C & -I & & \\ -I & C & -I & \\ & \ddots & \ddots & \ddots \\ & & -I & C & I \end{pmatrix}$$

BCR = divide and conquer

$$\begin{aligned}
&(1) \quad cx_1 - x_2 && = b_1 \\
&(2) \quad +)\ c \cdot (-x_1 + cx_2 - x_3 && = b_2) \\
&(3) \quad +)\ -x_2 + cx_3 - x_4 && = b + 3 \\
&\qquad (c^2 - 2I)x_2 - x_4 && = b_2 + cb_2 + b_3
\end{aligned}$$

Fast solvers for Poisson's equation

Block cyclic reduction.

1. Block cyclic reduction (BCR) is a fast method for the Poisson model problem. The fastest algorithms on vector and parallel computers are often a hybrid of block cyclic reduction and FFT. BCR has also be extendned to solve many other types of structured matrix computation problems. In this note, we describe a simple but numerically unstable version of the BCR algorithm. A stable implementation is given in the reference at the end of this note.

2. Recall 2-D Poisson's model problem is given by

$$(I_N \otimes T_N + T_N \otimes I_N \equiv T_{N\times N}) \cdot \mathrm{vec}(V) = \mathrm{vec}(h^2 F).$$

Write it as the standard form of the linear system of equations, we have

$$\begin{bmatrix} A & -I & & \\ -I & A & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix},$$

where $A = T_N + 2I$ and I is an $N \times N$ identity matrix. x_i and b_i are N-vectors.

For simplicity we assume that N is **odd**. We use block Gaussian elimination to combine three consecutive sets of equations

$$\begin{array}{rl} & [\ -x_{j-2} + Ax_{j-1} - x_j = b_{j-1}\ ], \\ +A & [\ -x_{j-1} + Ax_j - x_{j+1} = b_j\ ], \\ + & [\ -x_j + Ax_{j+1} - x_{j+2} = b_{j+1}\ ], \end{array}$$

Thus eliminating x_{j-1} and x_{j+1}

$$-x_{j-2} + (A^2 - 2I)x_j - x_{j+2} = b_{j-1} + Ab_j + b_{j+1}.$$

Doing this for every set of three consecutive equations yields two sets of equations:

- one for the x_j with j even

$$\begin{bmatrix} A^{(1)} & -I & & \\ -I & A^{(1)} & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_4 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} b_1 + Ab_2 + b_3 \\ b_3 + Ab_4 + b_5 \\ \vdots \\ b_{N-2} + Ab_{N-1} + b_N \end{bmatrix}, \tag{1}$$

where

$$A^{(1)} = A^2 - 2I \equiv \left(A^{(0)}\right)^2 - 2I,$$

- one set of equations for the x_j with j odd,

$$\begin{bmatrix} A & & & \\ & A & & \\ & & \ddots & \\ & & & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 + x_2 \\ b_3 + x_2 + x_4 \\ \vdots \\ b_N + x_{N-1} \end{bmatrix}. \tag{2}$$

This set of equations can be solved directly after solving the equation (1) for x_j with j even.

Note that equation (1) has the same form as the original problem, so we may repeat this process recursively. For example, at the next step we get

$$\begin{bmatrix} A^{(2)} & -I & & \\ -I & A^{(2)} & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A^{(2)} \end{bmatrix} \begin{bmatrix} x_4 \\ x_8 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}, \tag{3}$$

where

$$A^{(2)} = \left(A^{(1)}\right)^2 - 2I,$$

and

$$\begin{bmatrix} A^{(1)} & & & \\ & A^{(1)} & & \\ & & \ddots & \\ & & & A^{(1)} \end{bmatrix} \begin{bmatrix} x_2 \\ x_6 \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}. \tag{4}$$

We repeat this until only one equation is left, which we solve another way.

In summary, the BCR algorithm consists of three steps, where for simplicity, assume that $N = 2^{k+1} - 1$.

(a) Block reduction (see equations (1) and (3))
(b) Solve $A^{(k)}x^{(k)} = b^{(k)}$
(c) Back solve (see equations (2) and (4))

3. Complexlity: $O(N^2 \log_2 N)$

4. The simple BCR approach has two drawbacks:

   (a) It is numerically unstable because $A^{(r)}$ grows quickly:

   $$\|A^{(r)}\| \sim \|A^{(r-1)}\| \approx 4^{2^r},$$

   so in computing $b_j^{(r+1)}$, the $b_{2j\pm1}^{(r)}$ are lost in roundoff.

   (b) $A^{(r)}$ has bandwidth $2^r + 1$ if $A^{(0)} = A$ is tridiagonal, so it can be dense and thus more expensive to multiply or solve.

5. A numerically stable and efficient algorithm can be found in

   - B. Buzbee, G. Golub and C. Nielson , On the direct method for solving Poisson's equation, SIAM J. Numer. Anal. Vol. 7, pp.627–656, 1970.

- W. Gander and G.H. Golub, Cyclic Reduction - History and Applications, Proceedings of the Workshop on Scientific Computing: 10-12 March, 1997, edited by F. T. Luk, R. Plemmons. Springer Verlag, New York, 1997. Also appeared as Technical Report SCCM 97-02, Stanford University, 1997.

FFT (fast Fourier fransform) method.

1. Let us learn how to solve the 2D Poisson's model problem using the matrix-matrix multiplications involving the eigenvector matrix of T_N. A straightforward implementation of the matrix-matrix would cost $O(N^3)$. We will show how this multiplication can be implemented using the fast Fourier transform (FFT) in only $O(N^2 \log_2 N)$ operation. Note that if $N = 2^{20} = 1,048,576$, then $\log_2 N = 20$.

2. Recall that the formulation of the 2D Poisson's equation in the matrix equation form

$$T_N \cdot V + V \cdot T_N = h^2 F.$$

Let $T_N = Z \Lambda Z^T$ be the eigenvalue decomposition of T_N. Then the previous equation becomes

$$\Lambda \cdot \widetilde{V} + \widetilde{V} \cdot \Lambda = h^2 \widetilde{F}.$$

where $\widetilde{V} = Z^T V Z$ and $\widetilde{F} = Z^T F Z$. It is easy to see that the (j,k) entry of this equation is

$$\lambda_j \tilde{v}_{jk} + \tilde{v}_{jk} \lambda_k = h^2 \tilde{f}_{jk},$$

which can be solved for $\tilde{v}_{jk}$ to get

$$\tilde{v}_{jk} = \frac{h^2 \tilde{f}_{jk}}{\lambda_j + \lambda_k}.$$

This yields the first version of our algorithm:

(a) Compute $\widetilde{F} = Z^T F Z$

(b) For all j and k, compute $\tilde{v}_{jk} = \frac{h^2 \tilde{f}_{jk}}{\lambda_j + \lambda_k}$

(c) Compute $V = Z \widetilde{V} Z^T$

The cost of step (b) is $3N^2$, and the cost of steps (a) and (b) is four matrix-matrix multiplications by Z and $Z^T (= Z)$, which is $8N^3$ using a conventional algorithm. In the following, we show how multiplication by Z is essentially the same as computing a *discrete Fourier transform*, which can be done in $O(N^2 \log_2 N)$ operation.

3. Using the language of Kronecker product, we have

$$\begin{aligned} v = \mathrm{vec}(V) &= (T_{N \times N})^{-1} \cdot \mathrm{vec}(h^2 F) \\ &= \left((Z_N \otimes Z_N)(I_N \otimes \Lambda_N + \Lambda_N \otimes I_N)(Z_N \otimes Z_N)^T \right)^{-1} \cdot \mathrm{vec}(h^2 F) \\ &= (Z_N \otimes Z_N) \left(I_N \otimes \Lambda_N + \Lambda_N \otimes I_N \right)^{-1} (Z_N^T \otimes Z_N^T) \cdot \mathrm{vec}(h^2 F) \end{aligned}$$

It is easy to see that doing the indicated matrix-vector multiplications from right to left is mathematically the same as the algorithm described in Item 1. This also shows how to extend the algorithm to Poisson's equation in higher dimension.

4. The Discrete Fourier Transform (DFT) of an N-vector a is the vector

$$b = \Phi a,$$

where $\Phi = (\phi_{jk})$ is N-by-N matrix defined as follows:

$$\phi_{jk} = \omega^{j \times k}, \quad \text{for} \quad j, k = 0, 1, \ldots, N-1$$

where

$$\omega = \exp\left(\frac{-2\pi \mathtt{i}}{N}\right) = \cos\frac{2\pi}{N} - \mathtt{i}\sin\frac{2\pi}{N}, \quad \mathtt{i} = \sqrt{-1}$$

a principal Nth root of unity, $\omega^N = 1$.

The Inverse Discrete Fourier Transform (IDFT) of b is the vector

$$a = \Phi^{-1} b.$$

Therefore, both the DFT and IDFT are just matrix-vector multiplications and can be straightforwardly implemented in $2N^2$ operations. The DFT and IDFT are closely related the Fourier transform and its inverse in continuous case.

5. Properties of DFT:

   (a) $\frac{1}{\sqrt{N}}\Phi$ is a complex symmetric and unitary matrix, i.e.,

   $$\Phi^{-1} = \frac{1}{N}\Phi^H = \frac{1}{N}\bar{\Phi}.$$

   (Exercise: verify that $\Phi^H = (\bar{\Phi})^T = \bar{\Phi}$ and $\frac{1}{N}\Phi \cdot \Phi^H = I$.)

   (b) Let $a = [a_0, a_1, \ldots, a_{N-1}]$, then the kth component of the DFT $b = \Phi a$ is

   $$b_k = \sum_{j=0}^{N-1} a_j \omega^{kj}.$$

   This can be viewed as the value of the polynomial $p_a(x) = \sum_{j=0}^{N-1} a_j x^j$ at $x = \omega^k$:

   $$b_k = (\Phi a)_k = p_a(\omega^k).$$

   In other words,

   the DFT is polynomial evaluation at the points $\omega^0, \omega^1, \ldots, \omega^{N-1}$.

   Conversely,

   the IDFT is polynomial interpolation, producing the coefficients of a polynomial given its values at $\omega^0, \omega^1, \ldots, \omega^{N-1}$.

   (c) If $a = [a_0, \ldots, a_{N-1}, 0, \ldots, 0]^T$ and $b = [b_0, \ldots, b_{N-1}, 0, \ldots, 0]^T$ are $2N$-vectors, then the *discrete convolution* of a and b, denoted as $a * b$, is defined as

   $$a * b \equiv c = [c_0, c_1, \ldots, c_{2N-1}]^T,$$

   where $c_k = \sum_{j=0}^{k} a_j b_{k-j}$.

   To illustrate the use of the discrete convolution, consider the polynomial multiplication. Let $p_a(x) = \sum_{k=0}^{N-1} a_k x^k$ and $p_b(x) = \sum_{k=0}^{N-1} b_k x^k$ be degree-$(N-1)$ polynomials. Then their product

   $$p_a(x) \cdot p_b(x) = \sum_{k=0}^{2N-1} c_k x^k \equiv p_c(x),$$

   where the coefficients c_k are given by the discrete convolution.

   One purpose of the Fourier transform is to convert the convolution into multiplication.

Theorem 1 *Let $a = [a_0, \ldots, a_{N-1}, 0, \ldots, 0]^T$ and $b = [b_0, \ldots, b_{N-1}, 0, \ldots, 0]^T$ be vectors of dimension $2N$, and let $c = a * b = [c_0, \ldots, c_{2N-1}]^T$. Then*

$$(\Phi c)_k = (\Phi a)_k \cdot (\Phi b)_k.$$

PROOF. Recall the property (b), if $\widetilde{a} = \Phi a$, then the kth entries of $\widetilde{a}$ is $\widetilde{a}_k = \sum_{j=0}^{2N-1} a_j \omega^{kj}$, the value of the polynomial $p_a(x) = \sum_{j=0}^{N-1} a_j x^j$ at $x = \omega^k$, i.e.,

$$\widetilde{a}_k = p_a(\omega^k).$$

Similarly, $\widetilde{b} = \Phi b$ means that $\widetilde{b}_k = \sum_{j=0}^{N-1} b_j \omega^{kj} = p_b(\omega^k)$, and $\widetilde{c} = \Phi c$ means that $\widetilde{c}_k = \sum_{j=0}^{2N-1} c_j \omega^{kj} = p_c(\omega^k)$. Therefore

$$(\Phi a)_k \cdot (\Phi b)_k = \widetilde{a}_k \cdot \widetilde{b}_k = p_a(\omega^k) \cdot p_b(\omega^k) = p_c(\omega^k) = \widetilde{c}_k = (\Phi c)_k.$$

6. Fast Fourier Transform (FFT) is a fast way to multiply by Φ. Instead of $2N^2$, it will require only about $\frac{3N}{2} \cdot \log_2 N$ operations. We now derive the FFT via its interpretation as polynomial evaluation. Recall that the goal is to evaluate $p_a(x) = \sum_{k=0}^{N-1} a_k x^k$ at $x = \omega^j$ for $0 \le j \le N-1$. For simplicity we will assume $N = 2^m$. The FFT algorithm is based on the following two critical observations:

   (a) By writing

$$\begin{aligned} p_a(x) &= a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1} \\ &= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + (a_1 x + a_3 x^3 + a_5 x^5 + \cdots) \\ &= (a_0 + a_2 x^2 + a_4 x^4 + \cdots) + x(a_1 + a_3 x^2 + a_5 x^4 + \cdots) \\ &= p_{a_{\text{even}}}(x^2) + x p_{a_{\text{odd}}}(x^2), \end{aligned}$$

   we see that the evaluation of $p_a(x)$ is *divided* into evaluating two polynomials $p_{a_{\text{even}}}$ and $p_{a_{\text{odd}}}$ of degree $\frac{N}{2} - 1$ at $(\omega^j)^2$, $0 \le j \le N-1$.

   (b) Since $\omega^N = 1$,

$$\omega^{2j} = \omega^{2(j+\frac{N}{2})}.$$

   Therefore, there are really just $\frac{N}{2}$ points ω^{2j} for $j = 0, 1, \ldots, \frac{N}{2} - 1$.

In summary, evaluating a polynomial of degree $N-1 = 2^m - 1$ at all N points ω^j $(0 \le j \le N-1)$ is the same as evaluating two polynomials of degree $\frac{N}{2} - 1$ at all $\frac{N}{2}$ points,[1], and then combining the results with N multiplications and additions. This can be done *recursively* as shown by the following pseudo-code:

```
function ã = FFT(a, N)
    if N = 1
        return ã = a
    else
        ã_even = FFT(a_even, N/2)
        ã_odd = FFT(a_odd, N/2)
        ω = e^{-2πi/N}
        z = [ω^0, ω^1, ..., ω^{N/2-1}]
        return ã = [ã_even + z. * ã_odd,  ã_even − z. * ã_odd]
    end if
```

[1] those are the $\frac{N}{2}$th roots of the unity.

where $.*$ means componentwise multiplication of arrays (as in MATLAB), and have used the fact that $\omega^{j+N/2} = -\omega^j$.

7. Matlab script

```
function y = ffttx(x)
%FFTTX  Textbook Fast Finite Fourier Transform.
%    FFTTX(X) computes the same finite Fourier transform as FFT(X).
%    The code uses a recursive divide and conquer algorithm for
%    even order and straight matrix-vector multiplication otherwise.
%    If length(X) is m*p where m is odd and p is a power of 2, the
%    computational complexity of this approach is O(m^2)*O(p*log2(p)).

x = x(:);
n = length(x);
omega = exp(-2*pi*i/n);

if rem(n,2) == 0
   % Recursive divide and conquer
   u = ffttx(x(1:2:n-1));
   v = ffttx(x(2:2:n));
   k = (0:n/2-1)';
   w = omega .^ k;
   y = [u+w.*v; u-w.*v];
else
   % The Fourier matrix.
   j = 0:n-1;
   k = j';
   F = omega .^ (k*j);
   y = F*x;
end
```

8. Let the cost of this algorithm be denoted $C(N)$. Then we see that

$$C(N) = 2 \cdot C\left(\frac{N}{2}\right) + \frac{3N}{2},$$

assuming that the powers if ω are precomputed and stored in tables. This recurrence can be solved as the following:

$$\begin{aligned} C(N) &= 2 \cdot C\left(\frac{N}{2}\right) + \frac{3N}{2} \\ &= 2^2 \cdot C\left(\frac{N}{4}\right) + 2 \cdot \frac{2N}{2} \\ &= 2^3 \cdot C\left(\frac{N}{8}\right) + 3 \cdot \frac{2N}{2} \\ &= \cdots \\ &= (\log_2 N) \cdot \frac{3N}{2}. \end{aligned}$$

Note that $C(1) = 0$.

In conclusion, to compute the FFTs of columns (or rows) of an N-by-N matrix the total costs $N \cdot (\log_2 N) \cdot \frac{3N}{2} = \frac{3}{2}N^2 \log_2 N$, which is the complexity of the FFT method for solving the 2D Poisson's model problem.

9. We have seen that to solve the discrete Poisson's model problem by the eigenvalue decomposition of T_N requires the ability to multiply by the N-by-N matrix Z, whose the (j,k) entry is

$$z_{jk} = \sqrt{\frac{2}{N+1}} \sin\left(\frac{\pi(k+1)(j+1)}{N+1}\right),$$

where for the convenient of notation, we number rows and columns from 0 to $N-1$ starting now.

Now consider the $(2N+2)$-by-$(2N+2)$ DFT matrix Φ, whose j,k entry is

$$\exp\left(\frac{-2\pi \mathtt{i} jk}{2N+2}\right) = \exp\left(\frac{-\pi \mathtt{i} jk}{N+1}\right) = \cos\frac{\pi jk}{N+1} - \mathtt{i}\sin\frac{\pi jk}{N+1}.$$

Thus the N-by-N matrix Z consists of $-\sqrt{\frac{2}{N+1}}$ times the imaginary part of the second through $(N+1)$st rows and columns of Φ. So if we can multiply efficiently by Φ using the FFT, then we can multiply efficiently by Z. In practice, we can modify the FFT to multiply by Z directly. This is called the *Fast Sine Transform (FST)*.

10. References:

- C. Van Loan, Computational Framework for the Fast Fourier Transform, SIAM Press, 1992
- A. Edelman, P. McCorquodale, and S. Toledo. The future fast fourier transform? SIAM Journal on Scientific Computing, Vol.20, pp.1094-1114, 1999.

Index

dimension reduction, 12

principal component, 12

stable, 6