Numerical Analysis: Quick reference

vishvAs vAsuki

July 17, 2011

Contents

Part I

Introduction

1 Themes

Here, we are concerned with algorithms for the problems of continuous mathematics, especially solving them on a computer, and doing so in a way so as to tolerate round-off errors (which are inevitable due to limited availability of memory and the floating point architecture).

Using insights about stability of algorithms and conditioning of problems: The general theme may be summarized as: Finding fast and stable algorithms for well conditioned continuous mathematics problems: Eg: solving Ax = lx and Ax = b problems, finding SVD.

2 Characterization of research effort

2.1 Identify problems

Talk to scientists; identify clean problems faced in scientific calculations; develop geometric intuition, Formulate a problem; see if it is well conditioned.

2.2 Algorithm design

This is specialized and extended algorithms research: so general comments described there apply.

2.2.1 Common techniques

A frequently used trick in algorithm design is: Divide and conquer.
Try to parallellize it.
Keep up with new techniques.

2.2.2 Prototyping

Prototyping is very important. Play with a fast prototyping language (eg: matlab). Try it out on toy problems and note actual performance; implement it and let it have an impact on science.

2.3 Algorithm analysis

Often, besides evaluating evaluate time and space complexity, after having proved that the problem is well conditioned, we prove that iterative algorithms have a fast convergence rate, that the algorithm is stable.

2.3.1 Common tricks

2.3.1.1 Time space analysis

Use Integration to count flops. Counting flops of an alg using geometry: Convert the loop into a solid and find its volume.

2.3.2 Error analysis

(Problem condition and algorithm Stability analysis.) Show error term goes to 0.

3 Abused notation

$y = O(\epsilon) \implies \exists c = f(m,n), \forall x \lim_{\epsilon_M \to 0} y \leq c\epsilon_M$.
Extra definition: If $y = \frac{a(x)}{b(x)}$, at $b(x) = 0$, $y = O(\epsilon)$ means $a(x) = O(\epsilon b(x))$.

Part II

Low error floating point computation

4 Round off errors

See the computer architecture survey's chapters on number storage where representation accuracy and arithmetic accuracy are described. Tolerance of the problem and of the algorithm to representation error and error in basic arithmetic computations is considered later.

4.1 Underflow avoidance

Sometimes, in case of computations involving multiplication and division with very small numbers (eg: probability calculation), there is the chance of underflow: a small non-0 number x may be stored as 0; which would then lead to 0 and ∞ results during later multiplication and division. To avoid this, one can use the logrithmic representation: x would be stored as $\log x$, and (xy) would be computed using $\log x + \log y$.

5 Condition of a problem

5.1 Notation and motivation

Problem (at x)= Vector Function $f : X \to Y$ (at x). Is $\Delta f(x)$ big wrt Δx? Eg: Perturbing ball on peak vs is trough.

5.1.1 Use

Round off error analysis while solving on a computer!

5.2 Absolute condition number

$\hat{k} = \lim_{\Delta x \to 0} \sup_{\Delta x \leq \Delta x'} \left\| \frac{\Delta f(x)}{\Delta x} \right\|$
$= \frac{\|J_f(x)\Delta x\|}{\|\Delta x\|} = \|J_f\|$ (Induced p or ∞ or 2 norm of Jacobian).

5.3 Relative condition number

Aka Cond Num.
$k = \lim_{\Delta x' \to 0} \sup_{\Delta x \leq \Delta x'} \frac{\|\Delta f(x)\|/\|f(x)\|}{\|\Delta x\|/\|x\|} = \frac{\|J(x)\|}{\|f(x)\|/\|x\|}$. Well conditioned prob; Eg: $f(x) = \sqrt{x}$.

Ill conditioned if k near $(\epsilon^{-0.5}, \epsilon^{-1})$. Loose log k digits of accuracy: can't distinguish between f(x) closer than k. Eg: Roots of polynomial p(x): f(p(x))= x; so also Eigenvalue prob: $det(A - lI)$.

6 Stability of algorithm

6.1 Notation and motivation

6.1.1 Algorithm modeled as an approximate function

Vector fn $\tilde{f}$: an algorithm which approximates problem f. $\hat{f}(x)$: the value found by $\hat{f}$. This is not just the computer storage approximation fl(f(x)) of f(x), but includes error introduced because of $\tilde{f}$ being an approximation.

6.1.2 Relative accuracy of algorithm

Aka forward stability. [**Check**] $\frac{\left\|\tilde{f}(x)-f(x)\right\|}{\|f(x)\|} \leq O(\epsilon)$.

6.1.2.1 Limitations

Not useful if f() is ill conditioned at x: x rounded off to $\tilde{x}$, $\frac{\|f(\tilde{x})-f(x)\|}{\|\tilde{x}-x\|}$ huge; so relative error huge.

6.1.3 Conditioning/ stability separation

6.1.3.1 Assume good conditioning

First, don't try to solve an ill-conditioned problem; any algorithm will yield poor results merely because cannot $f(fl(x))$ is computed instead of f(x), due to limitations of computer storage.

6.1.3.2 A stable algorithm

Once you have a well conditioned problem, if you have an algorithm which computes some $\tilde{f}(x') \approx f(x')$ for $x' \approx x$, you are guaranteed a solution close to f(x).

6.1.3.3 Stability of algorithm: definition

For nearly right question, get nearly right ans:
$\forall x, \exists \tilde{x} : \frac{\|\tilde{x}-x\|}{\|x\|} \leq O(\epsilon) : \frac{\left\|\tilde{f}(x)-f(\tilde{x})\right\|}{\|f(\tilde{x})\|} \leq O(\epsilon)$. Independent of choice of norm.

6.2 Backward stability

Like Stability, but $\tilde{f}(x) = f(\tilde{x})$: Exactly right ans to nearly right question.

Implies stability; but simpler to analyze. Found useful in practice, with few exceptions; actual $\tilde{f}(x) - f(x)$: backward error or residual is small.

6.2.1 Accuracy

$\frac{\|\tilde{f}(x)-f(x)\|}{\|f(x)\|} = \frac{\|f(\tilde{x})-f(x)\|}{\|f(x)\|} \leq k \|\delta x\| / \|x\| \leq O(k\epsilon)$. So if alg stable, stability reflects condition number. So, useless if problem ill conditioned.

6.2.2 Show backward stability

Show $\tilde{f}(x) = f(x)(1+\epsilon) = f(\tilde{x}) = f(x(1+\epsilon))$: Start with $\tilde{f}(x)$, end up with $f(\tilde{x})$.

6.2.3 To show backward instability

Show for some x, $\tilde{f}(x) = f(\tilde{x})$ means $\frac{\|\delta x\|}{\|x\|}$ is huge.

6.3 Stability without backward stability

Outer product $f(x, y) = xy^*$ stable, but not backward stable: cant write as $(x+\delta x)(y+\delta y)^*$. If $f(x, y)$ has higher dimensions than x, y; backward stability often not possible.

$x \oplus 1$ stable but not backward stable: Take x so small that $fl(1 + x) = 1$, then $\tilde{x} = 0$ for $1 + \tilde{x} = 1$: then $\frac{\tilde{x}-x}{x} = 1 \neq O(\epsilon)$. Similarly, $fl(x + y - x)$ is not backward stable: when x and y are so far apart that $x + y = x$. But, x+y+z is backward stable.

6.4 Backward stable linear algebra ops

6.4.1 Scalar Arithmatic

$f(x, y) = fl(x) \odot fl(y)$ backward stable for $\odot = + - */$: as $fl(x) \odot fl(y) = (x(1 + \epsilon_1) + y(1 + \epsilon_2))(1 + \epsilon_3)$.

6.4.1.1 Catastrophic cancellation

$a = fl(1 + \epsilon_M) = 1$; so, $fl(1 - a) = 0$. Catastrophic error when small nums are calculated from large nums. Soln: modify alg to avoid it; Eg: reflecting on plane in Householder triangularization $\perp$ $v = -x - \|x\| e_1$, not $v = x - \|x\| e_1$.

6.4.2 Inner product

$fl(x^*y) = fl(\sum_{i=1}^{d} \overline{x_i} y_i) = \sum_{i=1}^{d} \overline{x_i} y_i (1+\delta_i), |\delta_i| \leq d\epsilon$. This suffices, as usually $d << \epsilon$ for dense x, y. Useful in backward error analysis.

6.4.3 Ab

So, $Ab = (A + \delta A)b, \delta A_{i,j} = nA_{i,j}\epsilon + O(\epsilon^2)$. Useful in backward error analysis.

6.5 Error analysis

Estimate accuracy of alg.

6.5.1 Backward error analysis

Analyze conditioning of problem; then show backward stability of all steps of algorithm; then show backward stability of the entire alg; then easily compute accuracy.

QR is stable in the backward sense, but not in the forward sense.

6.5.2 Forward error analysis

Introduce rounding errors at each step; see how they accumulate to form 'forward errors'; thence find relative error. Tougher than backward analysis. $O(\epsilon^2)$ terms ignored.

Part III

Common problems

7 Interpolation

See approximation theory ref.

8 Root finding

Find $x : f(x) = 0$.

8.1 Newton's method

Find the root iteratively. Take a local linear approximation $g(x) = f(x_n) + \nabla f(x_n)^T(x - x_n)$ for f, find its root; and use it as the next guess for the root.

$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}$. For vector function f: $x_{n+1} = x_n + (\nabla(f(x_n)))^{-1} f(x_n)$.

This becomes the Newton's method in optimization, when applied to solving for the optimality condition $\nabla f(x) = 0$.

9 Find numerical gradients

If f is a functional, do $\frac{(f(x+\Delta x_i e_i) - f(x-\Delta x_i e_i))}{2\Delta x_i}$. Useful in optimization program design to verify algebra used in gradient calculation.

10 Partial differential equations

Strategy: discretize it: approximate it by equations with finite unknowns. A source of large and sparse matrix problems.

10.1 Elliptic operator problems

Take bounded open set O in R^2: visualize. (Poisson) $\Delta u = f$ for x in O. Special case: (Laplace) $f = 0 = \Delta u$ with harmonica functions as solutions.

Could model steady state heat distribution in O from constant heat source f.

10.1.1 Boundary conditions

Satisfied on boundary of O:
(Dirichlet) $u(x) = \phi(x)$. Cauchy: $\frac{\partial u}{\partial n}(x) + a(x)u(x) = g(x)$. Mixed: different conditions at diff boundaries.

Could model heat loss at boundary.

10.2 Finite Difference method

Approximate f at discretized points by low order Taylor series expansions. n: number of discretization points. Fast solvers solve it in $O(n \log n)$.

10.2.1 Fibonacci sequence: A difference equation

$u_{k+1} = \begin{bmatrix} F_{k+2} \\ F_{k+1} \end{bmatrix}$
$= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} F_{k+1} \\ F_k \end{bmatrix} = Au_k$. So, $u_k = A^k u_0 = SL^k S^{-1} u_0 = SL^k c$.

10.3 Initial value problem

Dynamical system, time t, position vector fn v(t): $\frac{dv_1(t)}{dt} = 4v_1(t)+4v_2(t)$, $\frac{dv_2}{dt} = 12v_1(t) + 4v_2(t)$, $v_1(0) = 8, v_2(0) = 4$; find v(t) for larger t's: find solutions of form $v(t) = e^{lt}y$; use in orig eqns to get eigenvalue problem $4y_1 + 4y_2 = ly_1$, $12y_1 + 4y_2 = ly_2$ to find l's, y's and thence $v(t) = c_1e^{lt}y + c_2e^{l't}y'$, solve c using v(0).

10.4 Finite element method

Original fn replaced by fn which is globally smooth, but locally piecewise polynomial in cells such as triangles/ rectangles.

11 Non negative matrix factorization

$A \approx BC; A : m \times nB : m \times k, C : k \times n$, with $k < n, B \geq 0, C \geq 0$. Aka +ve matrix factorization. So, write each a_i as conic combination $a_i \approx \sum b_k c_{k,i}$. Approximation measured by various metrics/ distortions.

Least squares NNMA: $\min_{B,C\geq 0} \|A - BC\|_F^2 /2$; can add regularization parameters $\lambda \|B\|_F^2 + \mu \|C\|_F^2$. $\|A - BC\|_F^2 = tr((A - BC)^T(A - BC))$ convex in B individually as tr() is increasing and $(A - BC)^T(A - BC)$ is linear fn of B + quadratic fn of B; similarly convex in C. Not convex in BC [**Find proof**].

[**Incomplete**]

12 n-body motion problem

Got n planets affected by each others' gravity, or n molecules affected by electrostatic forces. Want to compute new position after some time step.

Naively requires $O(n^2)$ ops: must consider all pairs of interactions.

12.1 Fast multipole algorithms

Need only O(N) ops. Uses multipole expansions, like net charge etc to approximate effects of distant clusters over a body. A hierarchical decomposition of space is used.

Part IV

Dense matrix algebra

13 Decompositional approach

Easy to analyze stability. Can reuse decomposition to solve multiple instances of the problem: eg consider A = LU.

14 Condition number of a matrix

14.1 Condition number of Ax when x perturbed

$k = \sup_{\Delta x} \frac{\|A\Delta x\|}{\|\Delta x\|} \frac{\|x\|}{\|Ax\|} = \|A\| \frac{\|x\|}{\|Ax\|} \leq \|A\| \left\|A^{-1}\right\|$ or $\leq \|A\| \|A^{+}\|$ if A not square.

So, condition of $A^{-1}b$ when b perturbed, $k = \left\|A^{-1}\right\| \frac{\|Ax\|}{\|x\|} \leq \|A\| \left\|A^{-1}\right\|$.

14.2 Condition number of matrix wrt norm

$k(A) = \|A\| \left\|A^{-1}\right\|$.

2 norm condition number: $\frac{\sigma_1}{\sigma_n}$. So, here, $k(A) = k(A^T)$.

Ill conditioned matreces. Loose $\log k(A)$ digits of accuracy: change 1 bit in $\frac{\|\Delta x\|}{\|x\|}$, change in $\frac{\|A\Delta x\|}{\|Ax\|}$ is worth $\log k(A)$ bits.

14.3 Condition of $x = A^{-1}b$ when A perturbed

$(A + \Delta A)(x + \Delta x) = b$. So, ignoring $\Delta A \Delta x$, $\Delta x = -A^{-1}(\Delta A)x$. By Cauchy Schwartz, $\|\Delta x\| = \left\|A^{-1}\right\| \|(\Delta A)\| \|x\|$. So, $k = sup_{\Delta A} \frac{\|\Delta x\|\|A\|}{\|\Delta A\|\|x\|} \leq \|A\| \left\|A^{-1}\right\|$.

15 Solving Ax=b for x

15.1 Assumption

$b \in range(A)$.

15.2 Stability and expense

Using SVD is most reliable, but expensive. LU is cheapest, QR is more expensive, but more reliable. Usually, look at condition number k(A) and decide method.

15.3 Triangularization by row elimination

15.3.1 Using PA=LU

Make augmented matrix $X = [\, A \; b \,]$ corresponding to problem Ax = b; Use row operations to reduce X to $[\, U \; L^{-1}b \,]$: thence get the problem $Ux = L^{-1}b$; do back substitution: to get : $[\, I \; UL^{-1}b \,]$ corresponding to problem $Ix = UL^{-1}b$. Thus got $x_r \in range(A^T)$.

Then, to get set of all solutions: $\{x_r + x_n : x_n \in N(A)\}$.

15.3.2 Cost

If you have L,U: Solve LUx = b by solving Ly = b: forward substitution: $O(m^2 flops)$; then solving Ux=y :back substitution: $O(m^2 flops)$.

15.3.3 Stability

Back substitution is stable.

15.4 With A=QR

15.4.1 Using Householder reflections

Get A=QR; get $y = Q^*b$; get $x = R^{-1}y$. For stability, householder reflections used to get A=QR.

Note: $y = Q^*b$ implicitly found as part of using householder reflections to get $R = Q^*A$ by instead doing $[\, R \; y \,] = Q^* [\, A \; b \,]$.

15.4.2 Backward stability

For $\frac{\|\Delta A\|}{\|A\|} = O(\epsilon)$, $(A + \Delta A)\hat{x} = b$: use $A + \delta A = \tilde{Q}\tilde{R}$ from backward stability of Householder; $(\tilde{Q} + \delta Q)y = b$; $(\tilde{R} + \delta R)\hat{x} = y$; set $\Delta A = \delta A + \delta Q\tilde{R} + \tilde{Q}\delta R$.

So accuracy: $\frac{\|\delta x\|}{\|x\|} = k(A)\epsilon$.

15.5 Use SVD

Take $Ax = U\Sigma V^*x = b$, each of these is easy to invert. Costly, but very accurate: Σ reveals numerical rank, in case of very small sw_i, drop the corresponding u_i or v_i and get a well conditioned problem.

15.6 Use determinants (Impractical)

(Cramer) Let C be cofactor matrix : $C_{i,j}$ multiple of $A_{i,j}$ in $|A|$ formula; $C^TAx = C^Tb$; so $x = \frac{C^Tb}{det(A)}$; so for x_j, find jth row of C^T (= cofactors of jth col of A) times b = det (A where b replaces jth col).

15.7 Find the inverse

Find $A^{-1}b$: A change of basis operation, saying b in terms of C(A) rather than e_i's.

15.8 For Hermitian +ve semidefinite $X = A^*A$

15.8.1 Use LU/ Cholesky

Use $X = R^*R$. Time needed to solve two triangular systems: $O(m^2)$; time needed to make R: $O(m^3/3)$.

15.8.1.1 Stability

Diagonal heavy if $X \succ 0$: so no pivoting required. [**Find proof**]Cholesky alg to make R backward stable. So, $(A + \delta A)\tilde{x} = b$ for relatively small δA.

15.8.2 Avoiding $X = A^*A$ if A known

In doing A^*A, you square the condition number. So, if A is ill conditioned, you get an even more ill conditioned problem.

So, don't do this. Instead, replace A with LU or QR or $U\Sigma V^*$ and solve.

15.8.3 Use A=QR, if A known

Get $A = \hat{Q}\hat{R}$ ($2mn^2 - \frac{2}{3}n^3$ flops), solve $Rx = \hat{Q}^*b$. More numerically stable than Cholesky [**Find proof**].

Get $A = \hat{Q}\hat{R}$, so $A^*A = \hat{R}^*\hat{R}$ (Cholesky: LU for A^*A); get $\hat{R}^*\hat{R}x = A^*b$. Implementation: solve $\hat{R}^*w = A^*b$, then $Rx = w$. Finding A^*A + Cholesky = $mn^2 + \frac{n^3}{3}$ flops.

15.8.4 Diagonalize and solve

Find thin SVD ($2mn^2 + 11n^3$ flops): $A = \hat{U}\hat{\Sigma}V^*$: $P = A(A^*A)^{-1}A^* = \hat{U}\hat{U}^*$; so $\hat{U}\hat{\Sigma}x = \hat{U}\hat{U}^*b$, $\hat{\Sigma}V^*x = \hat{U}^*b$; very dependable.

16 Iteratively solve Ax=b

16.1 Get x one component at a time

Take A = D + L + U, with D diagonal, L/ U strictly lower/ upper triangular.

16.1.1 Using updated components immediately

Aka Gauss Siedel.

Take $x^{(k)}$, get $x^{(k+1)}$ by this:
find $x_j^{(k+1)}$ using $A, b, x_{1:j-1}^{(k+1)}, x_{j+1:n}^{(k)}$. So, $Dx^{k+1} = b - Lx^{k+1} - Ux^k$; $(D + L)x^{k+1} = b - Ux^k$.

Not guaranteed to converge.

16.1.2 Use only old guesses of x

Aka Jacobi iteration.

Take $x^{(k)}$, get $x^{(k+1)}$ by doing this: find $x_j^{(k+1)}$ using $A, b, x_{i \neq j}^{(k)}$: so using old iterates of x uniformally rather than some old and some new iterates in finding $x_j^{(k+1)}$. So, $Dx^{k+1} = b - Lx^k - Ux^k$.

Guaranteed to converge.

16.2 Using $(I - A)^{-1}$ series for square A

(Neumann) See linear algebra ref. Can be used in solving $A'x = b$ where $A' = (I - A)$.

17 Overdetermined system of equations: Least squares solution

17.1 Problem

Solve $\min_x \|Ax - b\|$, given $m > n$, b not in C(A). Error vector $e = A\hat{x} - b$. We want to $\min_x \|e\|^2$.

For versions with regularizers, weights etc.. see optimization ref.

17.1.1 Importance

Solving this for the case where 2 norm is used in the problem specification $\equiv$ orthogonal projection.

Useful also in linear regression; in that context also the maximum likelihood solution for data generated by a linear fn in the presense of Gaussian noise: see statistics, optimization ref.

17.2 Solution

17.2.1 Projection + inverse operation

Two steps: First, project b to $range(A)$; that is, find
$b' = \arg\min_{v \in ran(A)} \|b - v\|$. Then, solve $Ax = b'$.

When error magnitude is measured wrt some other norms, other, perhaps oblique projections may be needed.

17.3 Solution form: 2-norm minimization

$\|Ax-b\|^2 = (Ax-b)^*(Ax-b)$, set $\nabla f(x) = 0$ to find minimum. This is equivalent to solving
$A^*(b-A\hat{x}) = 0$ or $A^*A\hat{x} = A^*b$ (Normal equations).

17.3.1 As orthogonal projection + inverse

Note that the condition $A^*(b-A\hat{x}) = 0$ matches the condition from geometric intuition that the error vector $e \perp ran(A)$.

17.3.2 Solution algorithm

The projection and inversion can either be done together or separately. In the former case, projection can be accomplished by finding an orthogonal basis for $ran(A)$ using either QR or SVD.

17.3.2.1 Non-singular A

$(A^*A)^{-1}$ invertible. $\hat{x} = (A^*A)^{-1}A^*b, p = A\hat{x}$.
$P = A(A^*A)^{-1}A^*$. $(A^*A)^{-1} = A^{-1}(A^*)^{-1}$ iff A is square and both exist.

Pseudoinverse for non-singular A See linear algebra ref.

17.3.2.2 Rank deficient A

$(A^*A)^{-1}$ not invertible, rank deficient. [**Proof**]: $\square N(A^*A) = N(A)$: $A^*Ax = 0 \implies x^*A^*Ax = \|Ax\| = 0$. I-P projects to $N(A^T)$ (not N(A)): $(I-P)b = e$.

Solution $A^*A\hat{x} = A^*b$ has many solutions for x. This is equivalent to the solution obtained by projecting b to $ran(A)$ to get b' and then solving $Ax = b'$.

18 Triangularization by row elimination

Aka Gaussian elimination. Get PA=LDU where L is unit lower triangular, D is diagonal, U is unit upper triangular. This is unique: see linear algebra ref.

18.1 Algorithm

In step i, subtract multiples of row $A_{i,:}$ from rows $A_{i+1:m,:}$ so as to make $A_{i+1:m,i}$ subcolumn 0. These row operations correspond to doing $L^{-1}A = U$ to get U from A.

18.1.1 Pivoting

But, maybe $A_{i,i} = 0$. In this case, we need to bring row k with $Ak, i \neq 0$ in place of row i for the algorithm to proceed.

Pivoting is also needed for stability of the algorithm.

18.1.2 Cost

Find L, U ($\frac{2m^3}{3}$ flops).

18.1.3 Various Formulations

Reordering the loops: ijk version puts 0's column-wise. ikj version puts 0's row-wise, has good storage properties [**Find proof**].

18.1.3.1 Reducing memory usage

Overwrite A with L and U.

18.1.3.2 Outer product formulation

$A_{2:m,2:m} - \frac{A_{2:m,1}A^*_{1,2:m}}{a_{1,1}}$.

18.1.3.3 Reducing memory access

Let $l_k = 0 + A_{k+1:m,k}$ after k-1 steps of triangularization; then k+1th step of traing is to make $L_k^{-1} = I - l_k e_k^*$; $A = L_k^{-1}A$. Reducing A to U is to reduce it to $\begin{bmatrix} U_{1,1} & U_{1,2} \\ 0 & U_{2,2} \end{bmatrix}$; can do this repeatedly. So make yer alg use this decomposition, and block $\times$ as basic ops, to avoid having to get each col to cache.

18.2 Instablity when no pivoting

$A = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}$ yields $\tilde{L} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}$ and $\tilde{U} = \begin{bmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{bmatrix}$ with $A - \tilde{L}\tilde{U}$ big: so stable, but not backward stable (considering functions of m or A in error bound).

Instability when, resulting from pivot very small wrt other elements in A, element t in $\tilde{L}$ or $\tilde{U}$ huge: relative error $O(\epsilon)$ but abs error $O(\epsilon t)$.

$|\Delta A| \leq 3n\epsilon|L||U|$. [**Find proof**]$\tilde{L}\tilde{U} = A + \Delta A = LU + \Delta A$, $\frac{\|\Delta A\|}{\|L\|\|U\|} = O(\epsilon)$.

18.3 Partial pivoting (GEPP)

At step k, pivot selected as biggest element in $A_{k:m,k}$. Take $\prod L_i P_i A = U$ where L_i are atomic unit lower triangular, use $PLP^* = L'$ (row exchange in L = col exchange in L'), get PA = LU.

18.3.1 Stability of GEPP

$L_{i,j} \leq 1; \|L\| = O(1)$; let Growth Factor $\rho = \frac{\max|U_{i,j}|}{\max|A_{i,j}|}$; $\|U\| = O(\rho\|A\|)$; so $\tilde{L}\tilde{U} = \tilde{P}A + \Delta A$, $\frac{\|\Delta A\|}{\|A\|} = O(\rho\epsilon)$.

Maximal instability: $\rho = 2^{m-1}$: Eg: $A = \left[\begin{smallmatrix} 1 & 1 \\ -1 & 1 \end{smallmatrix}\right]$; m-1 digits of accuracy lost. Unstability occurs very rarely; usually $\rho \leq m^{-0.5}$. Used in Matlab op.

18.4 Complete pivoting

At step k, pivot selected as biggest element in $A_{k:m,k:m}$. $O(m^3) = \sum i^2$ flops: expensive. $PAP' = LU$. Stable.

18.5 Avoidance of pivoting

If X is +ve definite, no pivoting required: As all principle submatrices are +ve definite - so non-singular, X = LU exists.

Also, if X is diagonally dominant, diagonal dominance is preserved during triangularization. So, it does not require pivoting.

18.6 Formula for pivots

Let A_k be submatrix of first k*k elements; then from block multiplication, $P_kA_k = L_kD_kU_k$ holds; so pivots can also be found by $\frac{|D_k|}{|D_{k-1}|} = \frac{|A_k|}{|A_{k-1}|}$.

18.7 Symmetric Elimination Algorithm for spd A

Do Gaussian elimination + extra column ops to diagonalize/ maintain symmetry at each step.

$A = \left[\begin{smallmatrix} a_{1,1} & A_{2,1}^* \\ A_{2,1} & A_{2,2} \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 0 \\ \frac{A_{2,1}}{a_{1,1}} & I \end{smallmatrix}\right] \left[\begin{smallmatrix} a_{1,1} & 0 \\ 0 & A_{2,2} - \frac{A_{2,1}A_{2,1}^*}{a_{1,1}} \end{smallmatrix}\right] \left[\begin{smallmatrix} 1 & \frac{A_{2,1}}{a_{1,1}} \\ 0 & I \end{smallmatrix}\right] = LDL^*$. Get R^*R by doing $LD^{1/2}$ at each step.

18.7.1 Code and Opcount

R=A; Repeat: do symmetric elimination on submatrix $R_{i+1,i+1}$; do $R_i^*/\sqrt{r_{i,i}}$. Only Upper part of R stored.

Opcount: $\sum_{k=1}^{m}\sum_{j=k+1}^{m} 2(m-j) \approx \frac{m^3}{3}$ flops.

18.7.2 Stability

By SVD: $\|R\|_2 = \|R^*\|_2 = \|A\|_2^{1/2}$; *so* $\|R\| \leq \sqrt{m}\|A\|$ [**Check**]. So, R never grows large. So, backward stable : get $\hat{R} * \hat{R}$ for perturbed A. Forward error

in R large; but R and R^* diabolically correlated.

19 A=QR

19.1 Triangular orthonormalization

Take the m*n matrix A; arrive at the matrix $Q_n = A\hat{R}$. At step j, you have $Q_{:,1:j-1} = Q_{j-1}$ find the direction of q_j by removing the component of a_j in the subspace spanned by Q_{j-1}.

Thence, you arrive at reduced QR: $A = Q_n\hat{R}$, where Q_n is a m*n matrix. Can extend Q thence to be a square matrix: this is full QR.

So, QR $\exists\forall A$. If sign$(R_{i,i})$ is fixed to be +ve, QR unique.

19.1.1 Gram-Schmidt classical

At step j: Take $a_j - \hat{Q}_{j-1}\hat{Q^*}_{j-1}a_j$ and normalize it to get q_j, with $\hat{Q}_{j-1} = [q_1..q_j]$.

$2mn^2$ flops.

19.1.1.1 Instability

Even by the time it calculates q_{20}, error becomes unbearable.

19.1.1.2 Double gram-schmidt

Get A = QR, then do Q = Q'R' to re-orthogonalize Q. A surer way of getting orthogonal basis for range(A).

19.1.2 Gram-Schmidt Modified (MGS)

At step j: Remove the component of a_j first in the subspace $\langle q_1\rangle$, then remove the component of the residue from $\langle q_2\rangle$ and so on. Algebraically same as classical version: $Q_{j-1} = \sum_1^{j-1} q_i q_i^*$. Computational cost same as classical version.

19.1.2.1 Round off error

very small angle twixt q_1, a_2; so $q_2' = a_2 - q_1 q_1 * a_2$ very small, with smaller error (maybe 10^{-15} in $q'_{2,j}$ wrt $q'_{2,k}$); err amplified maybe 10^{10} times when q_2 made after normalization. MGS has lesser roundoff error: [**Find proof**].

19.2 Orthogonal triangularization

(Householder) Do $Q^*A = \hat{R}$, not $A\hat{R}^{-1} = \hat{Q}$. Init: R=A; $Q^* = Q_n..Q_1$; $Q_k = \left[\begin{smallmatrix} I_{k-1} & 0 \\ 0 & F \end{smallmatrix}\right]$ leaves $r_1..r_{k-1}$ and $r_1^*..r_{k-1}^*$ alone, Householder reflector F reflects x (last m-k+1 entries in r_k) to $\|x\| e_1$; last entries in $r_{k+1}..r_n$ elsewhere.

F reflects accross plane $\perp v = x - \|x\| e_1$ or $\perp v' = -x - \|x\| e_1$; so by geometry, $\frac{vv^*}{\|v\|}x$ is projection on v; $\|x\| e_1 = Fx = (I - 2\frac{vv^*}{\|v\|})x$.

To avoid catastrophic cancellation when $x - \|x\| e_1$ very small, choose $v = -sign(x_1)x - \|x\| e_1$. $F^* = F, soQ_k^* = Q$. Needs: $2mn^2 - \frac{2}{3}n^3$ flops.

19.2.1 Stability of finding Q, R

Calculated $\hat{Q}$ and $\hat{R}$ can have large forward errors; but they're diabolically correlated; backward error or residual $A - \hat{Q}\hat{R}$ very small. $A + \Delta A = \hat{Q}\hat{R}$ for $\frac{\|\Delta A\|}{\|A\|} = O(\epsilon)$. So, backward error analysis best way to proceed.

20 Find eigenvalues

20.1 Hardness

Can't get directly for $m \geq 5$: thm from Galois theory that roots can't be expressed as radicals etc.. So, iterative part necessary in alg. 2 phases: Preliminary reduction to structured form; then iterative part.

20.1.1 Usual approach

Easily reduce to almost-triangular form using similarity transformations. Then, use more similarity transformations to iteratively get close to triangular form.

20.1.2 Finding a single ew

(Dhillon) If you have an idea about the approximate size of the ew, you can find it in $O(n^2)$. Else, if ye need kth ew, it takes $O(kn^2)$ time.

For sparse A, it is much cheaper. [**Incomplete**]

20.2 Use characteristic polynomial

Find roots using rootfinder. So, every ew has one non 0 ev. Ill conditioned. Eg: If coefficients in $x^2 + 2x - 1$ change by ϵ, x changes by $O(\sqrt{\epsilon})$.

20.3 Reduction to Upper Hessenberg matrix H

0's below first sub diagonal. If $A = A^*$, get Tridiagonal matrix. Use Householder reflections:
$H = (\prod_i Q_{m-1-i}^*)A(\prod_{i=1}^{m-2} Q_i)$.

These are similarity transformations, so $\lambda(H) = \lambda(A)$.

For large sparse matreces: Use Arnoldi iteration.

20.3.1 Op count

Row ops, at 4 flops per num: $\frac{4m^3}{3}$; Col ops, at 4 flops per num: $2m^3$; total: $\frac{10m^3}{3}$. Reduced work if $A = A^*$: $\frac{4m^3}{3}$.

20.3.2 Stability

$\tilde{Q}\tilde{H}\tilde{Q}^* = A + \delta A$ for relatively small δA.

20.4 Approach Eigenvalue revealing factorizations

$A = QUQ^*$;
$U = ..Q_2^*Q_1^*AQ_1Q_2...$ If $A = A^*$, this leads to unitary diagonalization.

20.5 Power iteration for real symmetric A

The series $v^{(i)} = \frac{A^i x}{\|A^i x\|}$ and $l^{(i)} = r(v^{(i)})$ converge to eigenpair corresponding to largest ew λ_1, q_1: as $x = \sum a_i q_i$.

So, Applying A repeatedly takes x to dominant ev.

20.5.1 Convergence

Linear convergence of ev. $\left\|v^{(i)} - \pm q_1\right\| = O(|\frac{\lambda_2}{\lambda_1}|^i)$,
$\left\|\lambda^{(i)} - \pm\lambda_1\right\| = \left\|v^{(i)} - \pm q_1\right\|^2$.

20.6 Inverse iteration

ev of A and $(A - pI)^{-1}$ same, ew λ_i shifted and inverted to get ew $(\lambda_i - p)^{-1}$.
If p near λ_j, using power iteration on $(A - pI)^{-1}$ gives fast convergence.

Good for finding ev if ew already known.

20.6.1 Convergence

Linear convergence of ev.
$\left\|v^{(i)} - \pm q_j\right\| = O(|\frac{p-\lambda_j}{p-\lambda_k}|^i)$, $\left\|\lambda^{(i)} - \pm\lambda_1\right\| = \left\|v^{(i)} - \pm q_j\right\|^2$.

20.6.2 Alg

Solve $(A - pI)w = v^{(k-1)}$; normalize to get $v^{(k)}$.

20.7 Rayleigh quotient iteration

Inverse iteration, where $\lambda^{(i)} = R(v^{(i)})$ used as p (ew estimate).

20.7.1 Convergence

Cubic convergence of ev and ew. If $\left\|v^{(k)} - q_j\right\| \leq eps$ when $|\lambda^{(k)} - \lambda_j| \leq O(\epsilon^2)$. So $\left\|v^{(k+1)} - q_j\right\| = O(|\lambda^{(k)} - \lambda_j| \left\|v^{(k)} - q_j\right\|) =$
$O(\left\|v^{(k)} - (\pm q_j)\right\|^3)$. $|\lambda^{(k+1)} - (\lambda_j)| = O(\left\|v^{(k+1)} - q_j\right\|^2) = O(|\lambda^{(k)} - (\pm q_j)|^3)$.

Gain 3 digits of accuracy in each iteration.

20.8 Simultaneous iteration for real symmetric A

Aka Block power itern. (v_i) linearly independent; their matrix $V^{(0)}$. (q_i) orth ev of A; cols of $\tilde{Q}$.

Unstable. [**Find proof**]

20.8.1 Convergence

If $|\lambda_1| > .. > |\lambda_n| \geq |\lambda_{n+1}|..$, Orth basis of $\langle A^k v_1^{(0)}, ..A^k v_n^{(0)}\rangle$ converges to $\langle q_1, ..q_n\rangle$: take $v_i = \sum_j a_j q_j$, do power iteration.

20.8.2 Alg

Take some $Q^0 = I$ or other orth cols, get $Z = AQ^{(k-1)}$; get $Q^{(k)}R^{(k)} = Z$. Defn: $A^{(k)} = (Q^{(k)})^T AQ^{(k)}$, $R'^{(k)} = \prod R^{(k)}$.

$A^k = Q^{(k)}R'^{(k)}$: By induction: $A^k = AQ^{(k-1)}R'^{(k-1)} = Q^{(k)}R'^{(k)}$.

20.9 QR iteration

Not QR factorization. Get $Q^{(k)}R^{(k)} = A^{(k-1)}$;
$A^{(k)} = R^{(k)}Q^{(k)} = (Q^{(k)})^T A^{(k-1)}Q^{(k)}$: Similarity transformation. Works for all A with distinct $|\lambda_i|$; easy analysis for $A = A^T$.

Defn: $R'^{(k)} = \prod R^{(k)}$, $Q'^{(k)} = \prod_k Q^{(k)}$: same as $Q^{(k)}$ in Simult itern alg.

Stable - finding λ now routine!

20.9.1 Convergence for real symmetric A

Same as Simultaneous iteration starting with I.
$A^k = Q^{(k)}R'^{(k)}$: So, finds orth bases for A^k.

$A^{(k)} = (Q'^{(k)})^T AQ'^{(k)}$; $A_{i,i}^{(k)}$ are $R(Q_i'^{(k)})$; as $Q_i'^{(k)}$ converges, $A_{i,i}^{(k)} \to \lambda_i$, off diagonal entries tend to 0; so approaches Schur factorization.

Linear convergence rate: $max_j \frac{\lambda_{j+1}}{\lambda_j}$.

20.10 Modified QR alg

Tridiagonalize: $(Q^{(0)})^T A^{(0)} Q^{(0)} = A$. Pick shift $p^{(k)}$; get $Q^{(k)}R^{(k)} = A^{(k-1)} - p^{(k)}I$; get $A^{(k)} = R^{(k)}Q^{(k)} + p^{(k)}I$; if any off diagonal element is close to 0, take $\left[\begin{smallmatrix} A_1 & 0 \\ 0 & A_2 \end{smallmatrix}\right]$, deflate and apply QR.

Needs O(m) iterations costing $O(m^2)$ iterations each. [**Find proof**]

20.11 Stability

Aka Eigenvalue perturbation theory.

Part V

Sparse, large matrix algebra

21 Iterative Linear algebra methods

Unlike direct methods. To solve Ax=b and Ax=lx. Eg: Conjugate gradient, Lanczos, Arnoldi.

21.1 Exploiting sparsity: Use only Ax

Use black box access to methods which find Ax and A^*x; minimize these calls. Thus, take advantage of sparsity.

21.1.1 Flops

Dense computations: O(m) steps, $O(m^2)$ ops per step, total flops $O(m^3)$: Also worst case for Iterative methods. But generally O(m) or $O(m^2)$.

21.1.2 Accuracy

Converge geometrically to ϵ_{mach}; direct methods make no progress until all $O(m^3)$ are completed.

21.2 Krylov sequences and subspaces of A and b

$b, Ab, A^2b...$ Krylov subspaces $K_r(A, b)$ are spanned by successively larger groups of these. Can also form Krylov matrix. Orthogonalization (perhaps Gram Schmidt style) used between iterations in order to avoid erroneous linear dependence.

Convergence rate depends on spectral properties of A.

Analysis closely related to approximation of f(x) by polynomials of on subsets of the complex plane. [**Find proof**]

21.3 Projection of A to Krylov subspaces

Reduces the problem to problems in $1, 2..$ dimensions. Look at the action of A in $Ax = b$ or $Ax = lx$, restricted to Krylov subspace. Approximation gets closer as number of iterations $n \to m$.

21.4 Iteratively find ortho-basis of K(A, b)

21.4.1 Triangular orthogonalization

(Arnoldi). Direct method used Orthogonal triangularization (Householder); Now use Triangular Orthogonalization (Gram Schmidt) : Can be stopped in middle for partial solution Q_n. Also, use a trick: Take current orthogonal basis Q, set next column q_{i+1} = the normalized component of $Aq_i \perp (q_1..q_i)$. Thus, $q_{i+1} \perp [q_0\ Aq_0\ ..\ A^{i-1}q_0]$ also.

21.4.2 Description using H

Start with arbit b; normalize to get q_1; for any n, for $j \leq n$ get: $h_{j,n} = \langle q_j, Aq_n \rangle$; get: $h_{n+1,n} = \|v\| = \left\| Aq_n - \sum_{j=1}^{n} h_{j,n} q_j \right\|$: Do the subtraction mgs style when each $h_{j,n}$ is found; find $q_{n+1} = \frac{v}{h_{n+1,n}}$. $Aq_n = Q_{n+1} h_n = \sum_{i=1}^{n+1} q_i h_{i,n}$.

So, you have almost upper triangular/ upper hessenberg $(n+1) \times n : \hat{H}_n$. Get $AQ_n = Q_{n+1}\hat{H}_n$.

21.4.3 Square H

Take $\hat{H}$ cut to $n \times n : H_n$, the Ritz matrix; get $H_n = Q_n^* A Q_n$; Also: $H_n = Q_n^* Q_{n+1} \hat{H}_n$.

As if going towards Similarity transformation of A to $H = QAQ^*$***!*** Maybe $m \to \infty$: So maybe only solving for first $n < m$ cols of Q in $Q^*AQ = H$.

21.4.3.1 H_n as Projection of A to Krylov subspaces

Representation in basis $\{q_1..q_n\}$ of the orthogonal projection of operator A onto K_n: The shadow of operation of A in K_n. Consider operator $K_n \to K_n$: Take $v = Q_n x \in K_n$: so x in basis Q_n; apply A: $AQ_n x$; project Av back to K_n: $Q_n^* A Q_n x = H_n x$.

Can't get operator A back from H_n.

21.4.4 Hermitian case: tridiagonal form

(Lanczos): Arnoldi iteration for $A = A^*$. H_n becomes tridiagonal T_n with $\{a_i\}$ in diagonal and $\{b_i\}$ in 1st super and sub diagonals. 3 term recurrance: $Aq_n = b_{n-1}q_{n-1} + a_n q_n + b_n q_{n+1}$.

21.4.5 Reduction of arbit A to tridiagonal form

If A symmetric, $A = QTQ^*$ for tridiagonal T; if A assymetric gotto give up orthogonality or tridiagnoalness: so find $A = VTV^{-1} = VTW^T$ for tridiagonal but non symmetric T with diagonals $\{d_2, ..\}, \{a_1, ..\}, \{b_2, ..\}$. Solve AV = VT and $W^T A = TW^T$: 3 term recurrances.

22 Eigenvalue estimation

Aka Rayleigh Ritz procedure. $(H_n)_{i,j} = q_i^* A q_j$. As $Aq_j \in K_{j+1}$, for $i > j + 1$, $(H_n)_{i,j} = 0$. $(H_n)_{j,j} = r(q_j) = q_j^* A q_j$.

ew of H_n called Arnoldi ew estimates of A, Ritz values wrt K_n of A. ev of H_n are the stationary points of r(x) restricted to K_n: $r(x) = r(Q_n y) = \frac{y^T Q_n^T A Q_n y}{y^T y} = \frac{y^T H_n y}{y^T y}$. $\nabla r(Q_n y) = r'(y) = 0$ iff y is ev and $r'(y)$ is ew of H_n.

23 Directly solve Ax = b

Can use triangular triangularization: but sparsity could be spoilt due to row operations: so try to order elimination steps to minimize the number of non zeros added.

23.1 For symmetric +ve definite A

A can be interpreted to be the precision matrix of a gaussian distribution $Pr(x) \propto e^{-2^{-1} x^T A x - \mu^T A x}$, whose graphical model G is sparse. Then, solving $A\mu = b$ is same as finding the mode of Pr(x). Can use loopy belief propogation to solve Ax = b. This is exact when G is a tree; the updates correspond to triangular triangularization.

24 Iteratively solve Ax=b

24.1 The optimization view

Error $e_n = \|x - x_n\|$ vs residual $r_n = \|Ax_n - b\|$.

View solving Ax=b as an optimization problem: minimize something like this at every iteration.

24.2 For dense A

See linear algebra ref.

24.3 For SPD A

See later section.

24.4 Generalized minimum residuals (GMRES)

24.4.1 Residue minimization in a subspace

At step n, approximate x by $x_n \in K_n(A, r_0)$ which minimizes residual $\|.\|_2$ of $r_n = b - Ax_n$: minimize $AQ_n y - b = Q_{n+1}\hat{H}_n y - b$: $\left\|Q_{n+1}\hat{H}_n y - b\right\|_2 = \left\|\hat{H}_n y - Q_{n+1}^* b\right\|_2 = \left\|\hat{H}_n y - \|b\| e_1\right\|$. So oblique projection of problem.

24.4.2 Alg

At each step of Arnoldi itern to find orthobasis of $K_n(A, r_0)$: Find $y_n = \arg\min_y \left\|\hat{H}_n y - \|b\| e_1\right\|$, $x_n = Q_n y$.

24.5 CGNR

Conjugate Gradients for minimizing residue using Normal eqns. Take Ax = b, get $A^*Ax = A^*b$; then apply CG. Relationship with old r: $\hat{r} = A^*r$. Ortho projection of problem under normal matrix.

But $k(A^*A) = k(A)^2$: Slower convergence.

24.6 CGNE

Take $A^T u = x$, get $AA^T u = b$. AA^T is SPD; now apply CG. Residue r same; new direction vector $\hat{p} = A^{-T}p$. Get monotonic decay in error vector. Ortho projection of problem under normal matrix.

25 Ax=b, SPD A

Many practical applications.

25.1 Reduction to quadratic programming

Consider $A \succeq 0$. Then, any $q(x) = (0.5)x^T Ax - b^T x + c$ is convex, has minimum when $\nabla q(x) = Ax - b = 0$.

So, can now use ideas from quadratic programming to solve Ax = b! Can try to minimize q(x) iteratively! Move in the direction of $\nabla q(x)$.

25.2 Conjugate gradients (CG) for SPD A

25.2.1 Error minimization in a subspace

Consider iteration n: restricted to subspace K_n. For $x_n \in K_n$: Take $e_n = x - x_n$; solve optimization problem: minimize $\|e_n\|_A = x_n^T Ax_n - 2x_n^T b + b^T b = 2f(x_n) + k$: note definition of f().

Find $x_n = argmin_{x_n \in K_n} f(x_n)$: so minimizing only the shadow of f in K_n. f is convex, so is K_n, so it has a unique minimum.

Actual problem was to minimize q(x). Here, solving the ortho projection of problem under A.

25.2.2 Iteration

$x_0 = 0, p_0 = r_0 = b$. $x_n = x_{n-1} + a_n p_{n-1}$.

Step size $a_n = \frac{r_{n-1}^T r_{n-1}}{p_{n-1}^T A p_{n-1}}$: $f(x_n)$ minimum when $\nabla f(x_n) = Ax_n - b = A(x_{n-1} + a_n p_{n-1}) - b = a_n A p_{n-1} - r_{n-1} = 0$.

Residual $r_n = b - Ax_n = r_{n-1} - a_n A p_{n-1}$; direction $p_n = r_n + c_n p_{n-1}$; improvement in current step $c_n = \frac{(r_n^T r_n)}{(r_{n-1}^T r_{n-1})}$.

From iteration: $K_n = \langle x_1, ..x_n \rangle = \langle p_0, ..p_{n-1} \rangle$.

$\|e_{n-1}\|_A \geq \|e_n\|_A$. [**Find proof**]

Search directions are A conjugate: $p_n^T A p_j = 0$. [**Find proof**]$r_n^T r_j = 0$. [**Find proof**]

25.3 Conjugate residues (CR) for symmetric A

Minimize $\|Ax - b\|_A$:
oblique projection of the problem under A.

[**Incomplete**]

25.4 Biconjugate gradients (BCG) method for non singular A

Do CG on 2 systems together: Ax = b and $A^* x^* = b^*$. If A is near-symmetric, get good convergence.

[**Incomplete**]

25.5 Preconditioning

25.5.1 Uses

Maybe k(A) very high in $Ax = b$; so, get an equivalent, better conditioned problem. Or, maybe want to turn it into an equivalent problem which is easier to solve.

25.5.2 Left preconditioning

Take $M \in S_{++}$; $M^{-1}Ax = M^{-1}b$.
If $M^{-1} \approx A^{-1}, k(M^{-1}A) \approx 1$.

25.5.3 Right preconditioning

Take $AM^{-1}Mx = AM^{-1}u = b$.

25.5.4 Shift preconditioning

Take $M_L^{-1}AM_R^{-1}M_Rx =$
$M_L^{-1}b$. Want $\left\|M_L^{-1}AM_R^{-1}\right\|$ small.

25.5.5 Traits of good preconditioners

Work involved is actually in solving $My = b$, so usually want M very sparse. But, at the same time, want M to be close to A, which may be quite dense.

25.5.5.1 Joint Condition Number of M and A

k(A,M)

25.6 Finding left preconditioning matrix M

25.6.1 Using A=LU

Get $A \approx LU$, get $A^{-1} \approx M = U^{-1}L^{-1}$.

Use Incomplete LU factorization (ILU) with 0 pattern P: $A^{-1} = U^{-1}L^{-1}$ can kill sparsity, so sacrifice accuracy for sparsity. Alter LU alg to keep sparsity.

ILU(0): P same as 0's in A.

ILU(p): Keep level of fill $l_{ij} : k : |a_{ij}| \approx \epsilon_k < 1$, drop items with $l_{ij} > k'$. Heuristic which doesn't need finding log: init value: $l_{ij} = 0$ if $a_{ib} \neq 0$, else ∞; updates while putting 0: $l_{i,j} := \min(l_{ij}, l_{ik} + l_{kj} + 1)$: $l_{ik} + l_{kj}$ corresponds to change in level. Same run for every matrix with same pattern.

ILU (Thresholding): No row op when wt is near 0, keep only p pre and post diagonal entries.

25.6.2 Use support graph theory

M is symmetric and +ve semidefinite.

[**Incomplete**]

25.6.3 Find sparse M to min $\|MA - I\|_F$

Start with random M, do gradient descent. Let $F(M) = \|MA - I\|_F$; Use Frechet derivative: $\frac{F(M+E)-F(M)}{\|E\|}$.

[**Incomplete**]

25.7 Using preconditioners

25.7.1 Preconditioned conjugate gradient (PCG) for SPD A

Left preconditioning: $M^{-1}A$ is SPD under $\langle ., . \rangle_M$. Can adapt CG to this: always use $\langle ., . \rangle_M$, residue $z = M^{-1}r$. Replace new vars with old vars to get PCG alg.

Use $M = LL^T$, $\hat{p}_j = p_j$ etc.. to see that split PCG is equivalent to PCG.

Right preconditioning: AM^{-1} is SPD under $\langle ., . \rangle_{M^{-1}}$. Get alg equivalent to PCG.

Similarly, can apply PCG on CGNR and CGNE.

25.7.2 Preconditioning GMRES

Simply solve the preconditioned form of the eqn using GMRES.

Left and right preconditioning not equivalent. Right preconditioning minimizes actual, unskewed residue.

Bibliography

[1] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. Siam, 1997.