INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA CONTIENA 11 - TELEPHONE: 2240-1
CABLE: CENTRATOM - TELEX 460392-1

SECOND SCHOOL ON ADVANCED TECHNIQUES
IN COMPUTATIONAL PHYSICS
(18 January - 12 February 1988)

SMR.282/ 26

LINEAR ALGEBRA

J. Du Croz

Numerical Algorithms Group LTD, Oxford, U.K.

LINEAR ALGEBRA

Jeremy Du Croz
Numerical Algorithms Group Ltd
256 Banbury Road
Oxford OX2 7DE
England

1. Linear Equations
2. Eigenvalue Problems
3. Sparse Problems

Some recommended reading:

G. W. Stewart,
Introduction to Matrix Computations,
Academic Press, New York, 1973.

G. H. Golub and C.F. van Loan,
Matrix Computations,
The Johns Hopkins University Press,
Baltimore, 1983.

C.L. Lawson and R.J. Hanson,
Solving Least Squares Problems,
Prentice-Hall, Englewood Cliffs, 1974.

B.N. Parlett,
The Symmetric Eigenvalue Problem,
Prentice Hall, Englewood Cliffs, 1980.

I.S. Duff, A.M. Erisman and J.K. Reid,
Direct Methods for Sparse Matrices,
Clarendon Press, Oxford, 1986.

Software for linear algebra:

All commercial libraries
- NAG
- IMSL
- . . .

Public domain software
- Linpack
- Eispack
- . . .

For sparse problems
- Harwell subroutine library
- Sparspak

For further details see lectures on program libraries.

LINEAR ALGEBRA 1:

LINEAR EQUATIONS

(including Linear Least Squares problems)

This lecture will concentrate on methods for dense matrices: see Linear Algebra 3 for sparse matrices.

Systems of Linear Equations

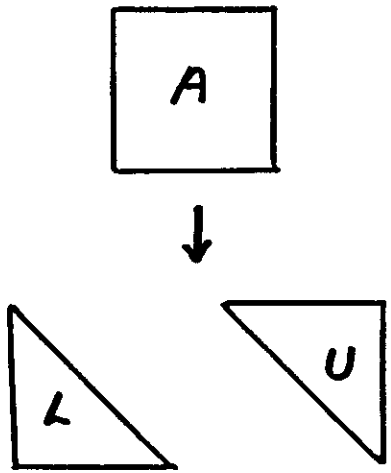
$$Ax = b$$

If A is upper or lower triangular, i.e.

$$Ux = b \quad \text{or} \quad Lx = b,$$

then x can be computed directly by forward or backward substitution.

For general A, we first factorize A as a product of triangular matrices.



$Ly = b$ and $Ux = y \Rightarrow LUx = b.$

Unsymmetric Systems

The LU factorization may not be defined, and is certainly not numerically stable, unless we incorporate row interchanges (or column interchanges, or both).

The result can be written

$$PA = LU$$

where P is a permutation matrix

Symmetric Positive-definite Systems

Stability is guaranteed without any interchanges:

$$A = LL^T \quad (\text{or } U^TU)$$

(Cholesky factorization)

Alternatively

$$A = LDL^T$$

where D is diagonal, and the diagonal elements of L are 1.

Symmetric Indefinite Systems:

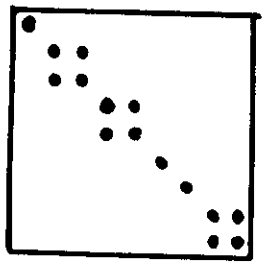
To achieve numerical stability:

use symmetric row-and-column interchanges

use 2×2 diagonal blocks if necessary

$$PAP^T = LDL^T$$

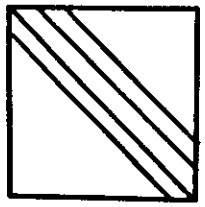
where D is block-diagonal with blocks of order 1 or 2:



(Bunch and Parlett)

Banded Systems

Suppose the elements of A are zero except for m_L subdiagonals and m_U superdiagonals, e.g. if $m_L = 1$ and $m_U = 2$:



We can organize the computation so that we do not store or compute with zero elements outside the band.

With unsymmetric systems, row interchanges introduce m_L extra superdiagonals.

Storage scheme:

			*	*	*	*
		1,3	2,4	3,5	4,6	5,7
	1,2	2,3	3,4	4,5	5,6	6,7
1,1	2,2	3,3	4,4	5,5	6,6	7,7
2,1	3,2	4,3	5,4	6,5	7,6	

Computational cost

measured in 'flops'

1 flop is the work typically involved in computing

$$a_{ij} \leftarrow a_{ij} + x * a_{ik}$$

	Flops	Storage
Unsymmetric	$\frac{1}{3}n^3$	n^2
Symmetric	$\frac{1}{6}n^3$	$\frac{1}{2}n^2$
Unsymm. band *	$< 2nm^2$	$3nm$
Symm. pos. def. * band	$\frac{1}{2}nm^2$	nm

* assuming $m_L = m_U = m \ll n$.

Note the benefits from taking account of the structure of the problem.

Perturbation Analysis

Suppose

$$(A + \delta A)(x + \delta x) = b$$

then:

$$\frac{\|\delta x\|}{\|x\|} \lessapprox \kappa(A) \cdot \frac{\|\delta A\|}{\|A\|}$$

provided $\|\delta A\|$ is small compared with $\|A\|$.

where the condition number

$$\kappa(A) = \|A\| \, \|A^{-1}\|$$

If $(A + \delta A)(x + \delta x) = (b + \delta b)$

then:

$$\frac{\|\delta x\|}{\|x\|} \lessapprox \kappa(A)\left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|}\right)$$

This shows the effect of uncertainty in A or b on the solution x, even if we compute an exact solution.

Effects of rounding errors

The effects of rounding errors in LU factorization are equivalent to introducing <u>additional uncertainty</u> into A, i.e.

the <u>computed</u> solution $\hat{x}$ is the <u>exact</u> solution of a nearby problem

$$(A+E)\hat{x} = b$$

(backward error analysis)

A theoretical bound:

$$\frac{\|E\|}{\|A\|} \leq 8n^3 \rho \varepsilon$$

where ε is the relative precision of the computer

ρ is the relative 'growth factor' for elements computed during the LU factorization.

Effects of rounding errors (continued)

The effect of row interchanges is to limit the growth factor ρ.

In practice $\rho < 100$ usually.

often $\rho \approx 1$

(especially for ill-conditioned matrices)

If A is symm. pos. def., $\rho = 1$.

Often the error E due to rounding errors is less than the uncertainty in the original data.

But the effect of E on the computed solution depends also on the condition number $\kappa(A)$

Error Bounds and Estimates

"A priori bounds are not, in general, quantities that should be used in practice. Practical error bounds should usually be determined by some form of a posteriori error analysis, since this takes full advantage of the statistical distribution of rounding errors and of any special features, such as sparseness in the matrix."

J. H. Wilkinson

Iterative Improvement:

let $\hat{x}$ be the computed solution:

compute the residual

$$r = b - A\hat{x}$$

in extended precision

compute the correction d by solving

$$Ad = r$$

Iterative Improvement (continued)

then $\hat{x} + d$ is a more accurate solution, and d gives valuable information about the accuracy of $\hat{x}$.

Often it is sufficient to perform just this one step of 'iterative' refinement. It is possible to repeat the process and compute a solution with maximum possible accuracy:

if the computer has t digit precision,

$\hat{x}$ has k correct digits $(k \geq 1)$,

and s extra digits are used to compute r,

then the maximum accuracy is:

$$\min(k+s, t)$$

Iterative Improvement (continued)

The cost is $O(n^2)$ flops for each iteration,

and an extra n^2 elements of storage for keeping the original matrix.

The main disadvantage is the need to compute residuals in extended precision

$\Rightarrow$ machine-dependent software.

Alternatives

1. 'Do-it-yourself': perturb your data, solve the problem again, and compare the two solutions (more expensive — need to factorize the perturbed matrix)
2. Estimate the condition-number (LINPACK). Requires $O(n^2)$ flops. (But does not give so much information).

Linear Least Squares Problems

given

$$b = Ax + r$$

where A is rectangular m-by-n ($m \geq n$) and r is a 'noise' or 'error' vector.

Problem: find x to

minimize $\|r\|_2 = \|b - Ax\|_2$

('overdetermined' system of 'equations').

Assume A has rank n (i.e. is not rank-deficient).

then the problem has a unique solution, denoted $\hat{x}$,

with $\hat{r} = b - A\hat{x}$.

Perturbation Analysis

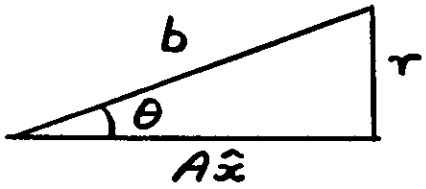
Suppose A is perturbed by δA (without reducing its rank) and b is perturbed by δb:

and hence the solution $\hat{x}$ is perturbed by $\delta\hat{x}$:

then

$$\frac{\|\delta\hat{x}\|}{\|\hat{x}\|} \leq \max\left(\frac{\|\delta A\|}{\|A\|}, \frac{\|\delta b\|}{\|b\|}\right) \times \left(2\sec\theta.\,\kappa(A) + \tan\theta.\,\kappa(A)^2\right)$$

where θ is defined by:



if θ is small ('small residual problems'),
the condition number $\propto \kappa(A)$

if θ is large ('large residual problems'),
the condition number $\propto \kappa(A)^2$

Method of Normal Equations

Compute $\hat{x}$ as the solution of

$$A^TA\hat{x} = A^Tb$$

(A^TA is symm. pos. def., so we can use Cholesky factorization)

Cost:

$\frac{1}{2}mn^2$ flops to form A^TA

$\frac{1}{6}n^3$ flops to factorize A^TA

But the condition-number for solving the system of equations is:

$$\kappa(A^TA) = \kappa(A)^2$$

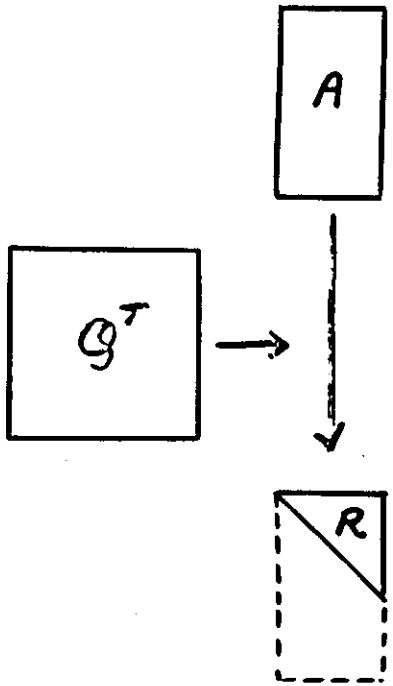
this indicates potential instability for small-residual problems.

Hence numerical analysts recommend using QR factorization, but:

"the subject is tricky, and informed men of good will can disagree, even in a specific application"

Linpack Users' Guide

Method of QR factorization



Cost:
$mn^2 - \frac{1}{3}n^3$
flops

R is m-by-n upper-triangular

where Q is <u>orthogonal</u>

i.e. $Q^TQ = QQ^T = I$

so $A = QR$

also $A^TA = R^TQ^TQR = R^TR$

so R is the (upper triangular) Cholesky factor of A^TA

Method of QR factorization (contd)

Advantages of orthogonal matrices:

1. leave 2-norms invariant:

$$\|Qx\|_2 = \|x\|_2$$

2. hence leave corresponding condition-numbers invariant:

$$\kappa_2(QA) = \kappa_2(A)$$

3. do not amplify rounding errors
⇒ numerically stable algorithms.

Linear least squares problem:

$$\begin{aligned}\|b - Ax\|_2 &= \|Q^Tb - Q^TAx\|_2 \\ &= \|Q^Tb - Rx\|_2 \\ &= \left\| \begin{bmatrix} c \\ d \end{bmatrix} - \begin{bmatrix} R' \\ 0 \end{bmatrix} x \right\|_2 \\ &= \left\| \begin{bmatrix} c - R'x \\ d \end{bmatrix} \right\|_2\end{aligned}$$

Hence $R'\hat{x} = c$ and $\|\hat{r}\|_2 = \|d\|_2$

Orthogonal matrices

in computation they are formed as products of

either plane rotations (Givens)

or reflections (Householder)

They are nearly always generated so as to set some element or elements of QA to zero, while preserving, as far as possible, zeros which already exist in A.

Reflections are used to set part of a column (or row) to zero, all at once.

Rotations are used to set one element to zero at a time.

Reflections are usually more efficient when they are appropriate.

Reflection: example

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & I-\alpha uu^T & \\ & & & \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & \times & \times & \times & \times \end{bmatrix}$$

$$= \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \underline{\times} & \underline{\times} & \underline{\times} & \underline{\times} \\ & & 0 & \underline{\times} & \underline{\times} & \underline{\times} \\ & & 0 & \underline{\times} & \underline{\times} & \underline{\times} \\ & & 0 & \underline{\times} & \underline{\times} & \underline{\times} \end{bmatrix}$$

Plane rotations: example

$$\begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & c & s & \\ & & -s & c & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

$$= \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \underline{\times} & \underline{\times} & \underline{\times} & \underline{\times} \\ & & 0 & \underline{\times} & \underline{\times} & \underline{\times} \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

Rank-deficiency

(or near-deficiency)

Now assume A has rank $< n$:

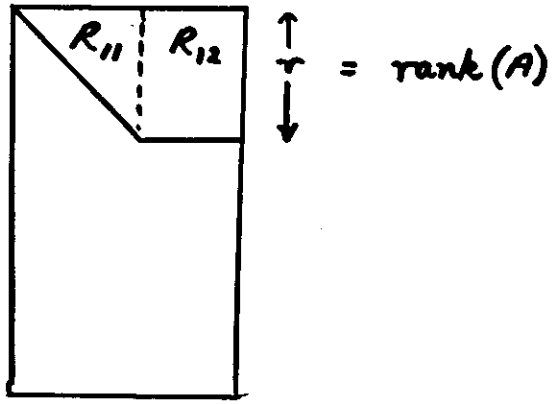
the linear least squares problem has an infinite number of solutions, but a unique solution of <u>minimum norm</u>

R is <u>singular</u> (or very <u>ill-conditioned</u>)

QR-factorization with column interchanges

$$AP = QR$$

where P is a permutation matrix and R has the form



Rank-deficiency (cont'd)

But it is not always possible to determine rank reliably by this method (though it usually works well in practice)

Singular Value Decomposition (SVD)

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

where U and V are orthogonal and Σ is diagonal

(for computing the SVD see lecture 2)

The diagonal elements of Σ are the singular values and can be arranged in descending order

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = 0$$

where r is the rank of A.

This is the most reliable approach to determining the numerical rank: even so, deciding when to regard a computed singular value as zero is not always straightforward: it depends on the accuracy of the data.

Rank-deficiency (continued)

The minimum-norm solution to a rank-deficient linear least squares problem is given by:

$$\hat{x} = V \Sigma^+ U^T b$$

where

$$\Sigma^+ = \mathrm{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0\right)$$

Note: we do not need to compute U, only $U^T b$.

Note also:

the pseudo-inverse of A is given by

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

the condition number of A (for the 2-norm) is given by

$$\kappa_2(A) = \sigma_1 / \sigma_n$$

Summary for linear least Squares

method	cost in flops
normal equations	$\frac{1}{2}mn^2 + \frac{1}{6}n^3$
QR factorization	$mn^2 - \frac{1}{3}n^3$
SVD	$mn^2 + \frac{17}{3}n^3$

A possible strategy for coping with the possibility of near rank-deficiency is:

1. compute QR factorization of A
2. estimate condition number of R

if R is not too ill-conditioned, then:

3. solve by method of QR-factorization

otherwise:

4. solve by method of SVD (we only need to compute SVD of R)

See NAG routine F04JGF.

Modifying matrix factorizations

In solving $Ax = b$,
the cost of factorizing A is $O(n^3)$,
the subsequent cost of computing x is $O(n^2)$.

If we wish to solve for a new r.h.s, $Ax' = b'$, we do not need to repeat the factorization; the extra cost is $O(n^2)$.

Suppose we make a 'small' change to A: in certain cases we can modify the factorization of A with cost $O(n^2)$, e.g.

adding a row (or deleting)

adding a column (or deleting)

rank-1 change:

$$\tilde{A} = A + xy^T$$

$$\tilde{A} = A + xx^T \text{ (symmetric)}$$

Example: updating Cholesky

Given: $A = LL^T$

$\tilde{A} = A + xx^T$

to find: $\tilde{L}$ such that $\tilde{A} = \tilde{L}\tilde{L}^T$.

Solution: compute orthogonal Q such that

$$[L \vdots x]Q = [\tilde{L} \vdots 0]$$

then
$$LL^T + xx^T = [L \vdots x]\begin{bmatrix} L^T \\ x^T \end{bmatrix}$$
$$= [L \vdots x]QQ^T\begin{bmatrix} L^T \\ x^T \end{bmatrix}$$
$$= [\tilde{L} \vdots 0]\begin{bmatrix} \tilde{L}^T \\ 0 \end{bmatrix}$$
$$= \tilde{L}\tilde{L}^T$$

Q is formed as a product of n plane rotations, but we do not need to form Q explicitly, only need to compute $\tilde{L}$.

LINEAR ALGEBRA 2:

EIGENVALUE PROBLEMS

This lecture will concentrate on methods for dense matrices: see Linear Algebra 3 for sparse matrices.

Standard Eigenvalue Problem

$$Ax = \lambda x \qquad x \neq 0$$

the QR-algorithm is the best method for computing all the eigenvalues of a matrix that is not too large:

$$A = QR$$

$$A' = RQ = Q^T A Q$$

. . .

shifts are incorporated to accelerate the convergence of a_{nn} to an eigenvalue:

$$A - \sigma I = QR$$

$$A' = RQ + \sigma I = Q^T A Q$$

. . .

for efficiency, the QR algorithm is applied after A has been reduced to a condensed form (tridiagonal, Hessenberg).

Symmetric Eigenvalue Problem

$$AQ = Q\Lambda$$

$\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ – eigenvalues

Q is orthogonal – eigenvectors

hence $A = Q\Lambda Q^T$

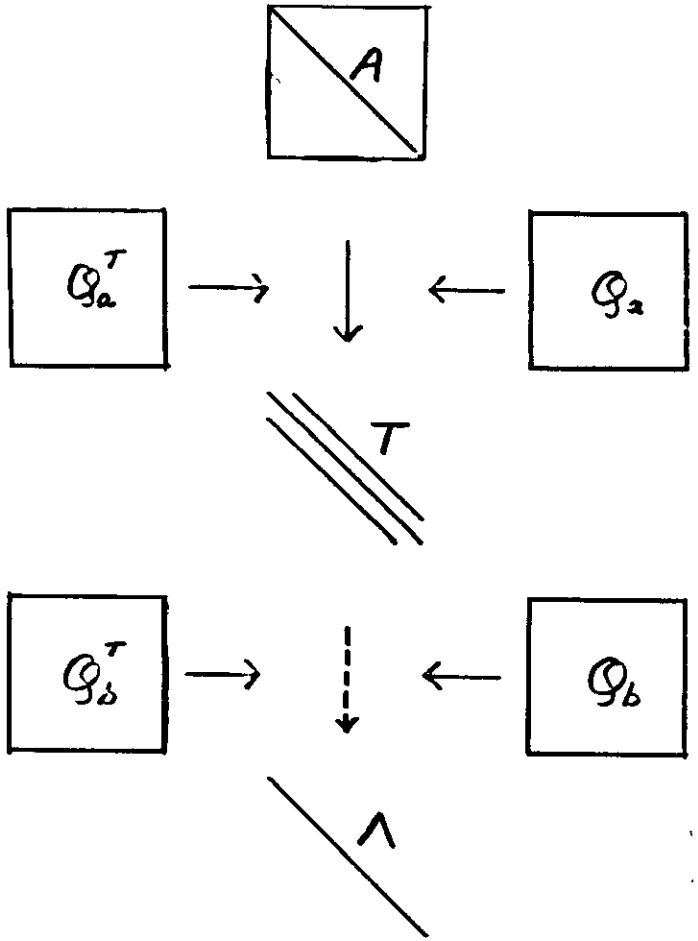
or $Q^T A Q = \Lambda$

eigenvalues always well-conditioned

orthogonal transformations $\Rightarrow$ numerical stability

QR-algorithm (with shifts) is globally and rapidly convergent

Parlett (1980). The Symmetric Eigenvalue Problem



$$Q_b^T Q_a^T A Q_a Q_b = \Lambda$$

To compute a few <u>selected</u> <u>eigenvectors</u>:

use <u>inverse</u> '<u>iteration</u>'

choose x_0

compute x_1 by solving $(A-\lambda I)x_1 = x_0$

test x_1 for acceptability

(usually x_1 is acceptable; if it isn't, it may be best to choose a new x_0)

It does not matter that $A-\lambda I$ is extremely ill-conditioned: we are only interested in the <u>direction</u> of x_1.

N.B. practical algorithms – and software – for inverse iteration contain many tricky details.

There may be a loss of orthogonality between computed eigenvectors

To compute a few <u>selected</u> <u>eigenvalues</u> use <u>bisection</u> combined with <u>spectrum slicing</u>:

if $A - \sigma I = L D L^T$,

then: no. of eigenvalues of A which are $< \sigma$ = no. of negative diagonal elements of D.

<u>Singular Value Decomposition (SVD)</u>

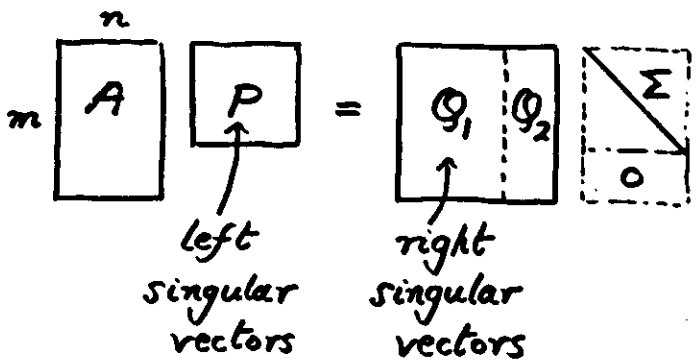
$$AP = Q \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$$

$\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n)$ – the singular values

Q and P orthogonal

hence $$A = Q \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} P^T$$

or $$Q^T A P = \begin{bmatrix} \Sigma \\ 0 \end{bmatrix}$$



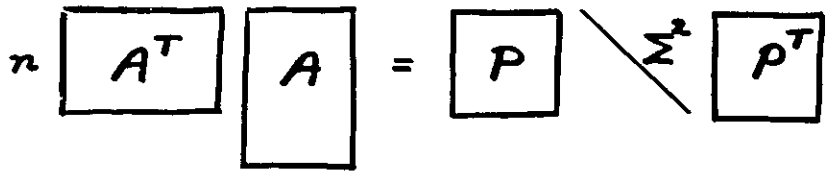
$$A p_i = \sigma_i q_i$$

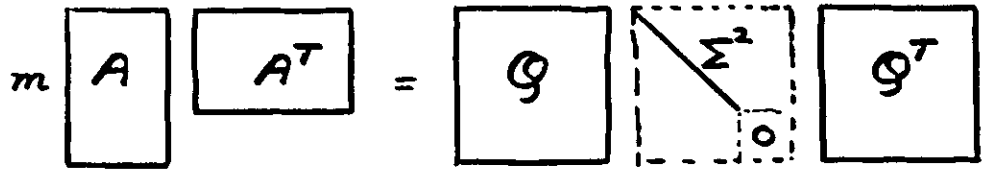
$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$$

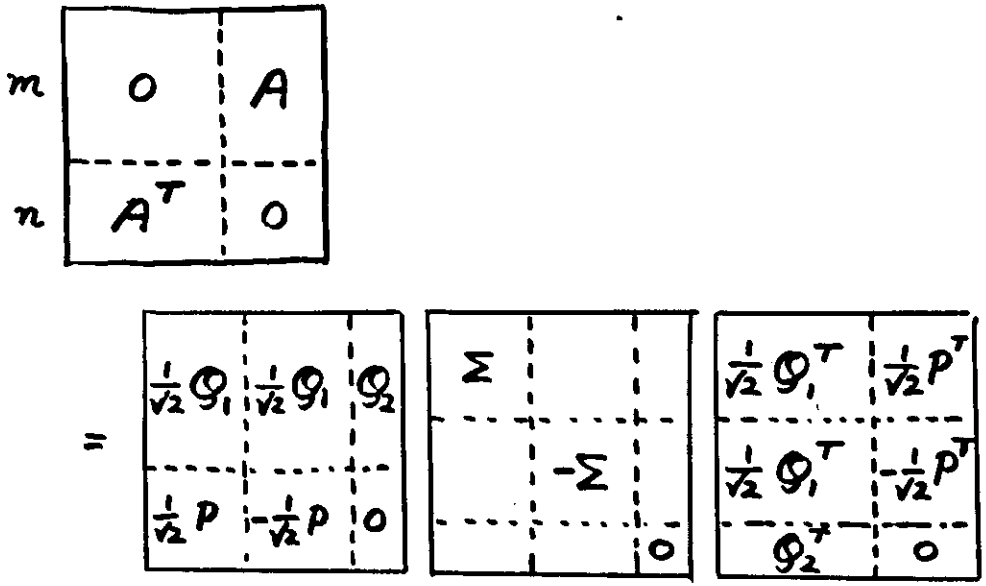
(assume, for illustration, $m \geq n$)

(often $m \gg n$)

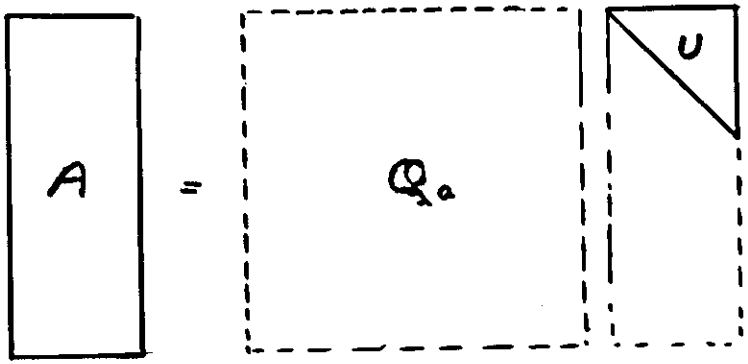
Associated symmetric eigenvalue problems:







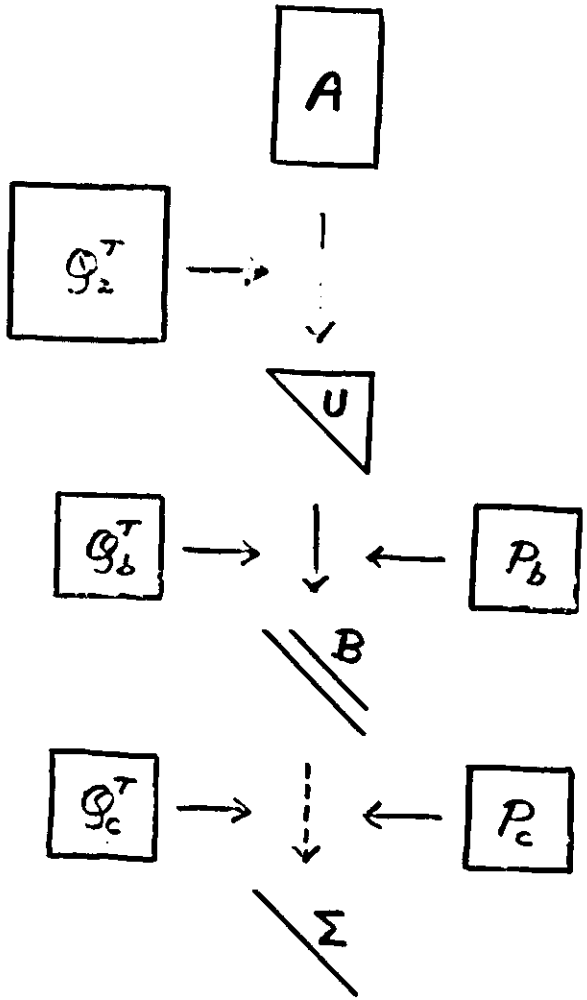
Preliminary QU-factorization is especially advantageous if $m \gg n$:



also useful if we can take advantage of any special structure in A (e.g. bandedness), or if A is generated one row at a time.

If $A = Q_a \begin{bmatrix} U \\ 0 \end{bmatrix}$ and $U = Q_b \Sigma P^T$,

then $A = Q_a \begin{bmatrix} Q_b & \\ & I \end{bmatrix} \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} P^T$



$$Q_c^T Q_b^T Q_a^T A P_b P_c = \Sigma$$

Unsymmetric Eigenvalue Problem

Schur factorization

$$A = Q U Q^H$$

or $$Q^H A Q = U$$

or $$A Q = Q U$$

U is upper triangular

Q is unitary - Schur vectors

eigenvalues are u_{ii}

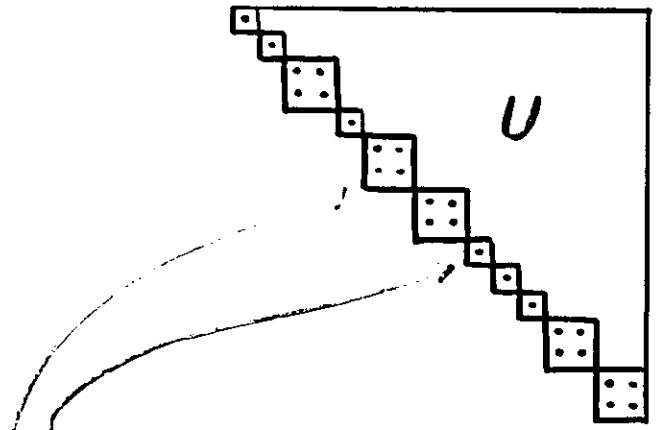
If A is real, then in real arithmetic:

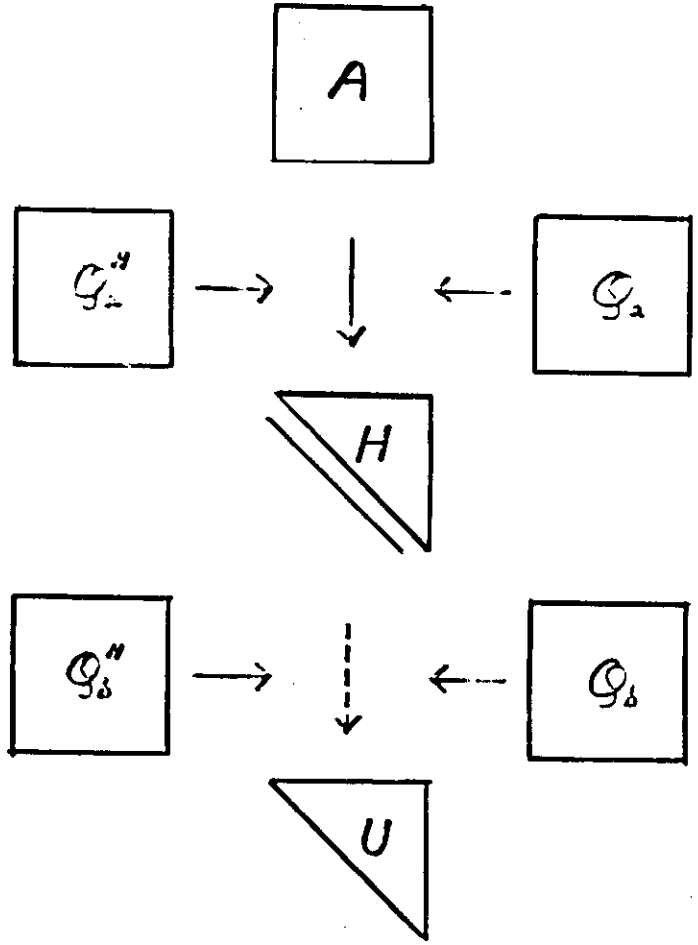
$$A = Q U Q^T$$

U is <u>block</u> upper triangular

Q is orthogonal

Block upper triangular form





$$Q_b^H Q_a^H A Q_a Q_b = U$$

If eigenvectors of A are wanted, compute eigenvectors of U by back-substitution:

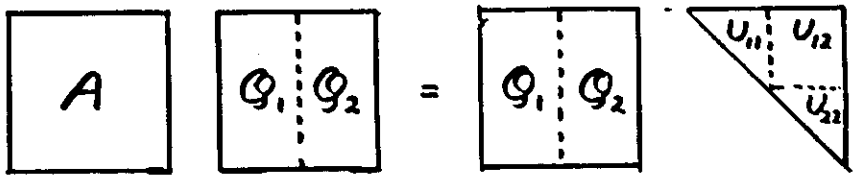
$$U X = X \Lambda$$

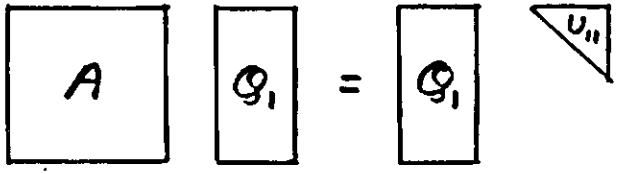
(non-unitary transformations!)

and then

$$A(QX) = (QX)\Lambda$$

Invariant Subspaces





The columns of Q_1 form an orthonormal basis of the invariant subspace corresponding to those eigenvalues which lie on the diagonal of U_{11}

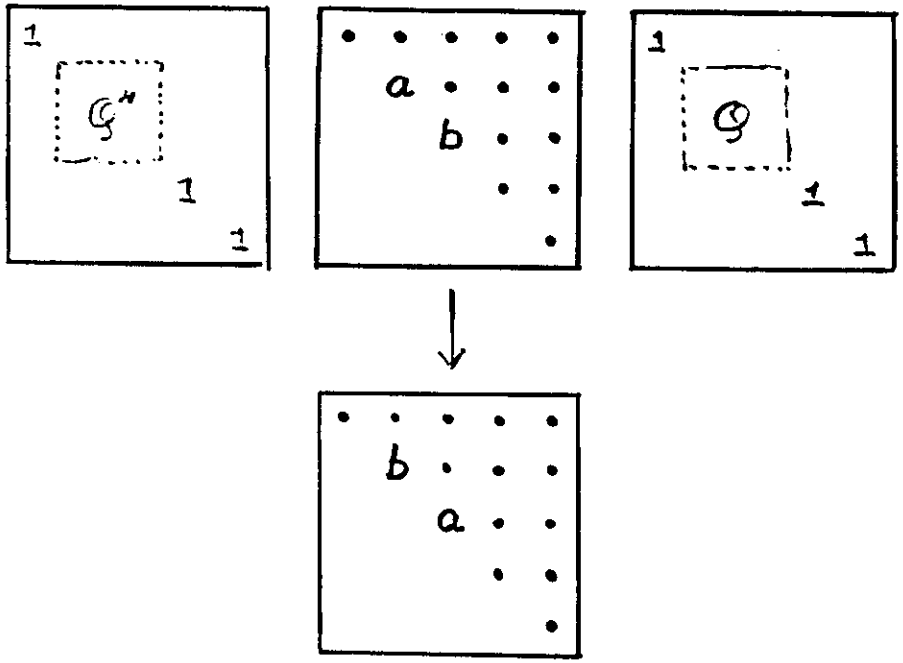
Hence we need to be able to re-order the Schur factorization, to bring the required eigenvalues up to the top of the diagonal of U

Re-ordering the Schur factorization

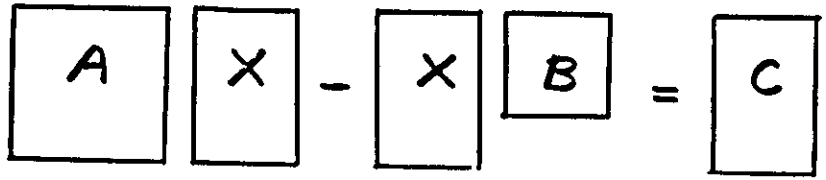
must maintain upper triangular form of U

can be done by unitary transformations

Algorithm EXCHNG (Stewart, 1976)

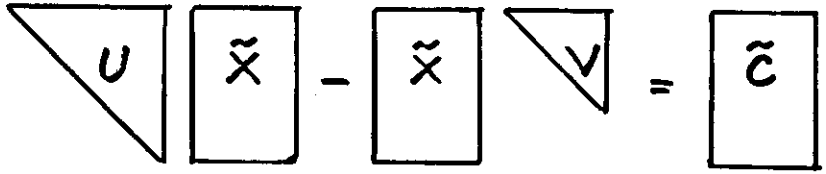


Sylvester equation



$$(QUQ^H)X - X(RVR^H) = C$$

$$U(Q^HXR) - (Q^HXR)V = Q^HCR$$



this can be solved by back-substitution provided that U and V have no eigenvalues in common. (Bartels & Stewart, 1972)

(But a faster algorithm is possible in which A is only reduced to upper Hessenberg form. (Golub, Nash & Van Loan, 1979))

Reduction to block-diagonal form

requires non-unitary transformations

$$\begin{bmatrix} I & -X \\ & I \end{bmatrix}\begin{bmatrix} U & C \\ & V \end{bmatrix}\begin{bmatrix} I & X \\ & I \end{bmatrix} = \begin{bmatrix} U & \\ & V \end{bmatrix}$$

if $UX - XV = -C$

if the elements of X are large, the transformation is ill-conditioned.

Bavely and Stewart (1979): compute Y such that

$$Y^{-1}UY = \begin{bmatrix} U_{11} & & & \\ & U_{22} & & \\ & & \ddots & \\ & & & U_{rr} \end{bmatrix}$$

with the blocks as small as possible provided that Y is not too ill-conditioned. This form can be useful for computing functions of matrices.

Condition Numbers

if λ is a simple eigenvalue of A,

with $Ax = \lambda x$

and $y^H A = \lambda y^H$ (or $A^H y = \bar{\lambda} y$)

then

$$\text{cond}(\lambda) = \frac{\|y\|\,\|x\|}{|y^H x|}$$

this is invariant under unitary transformations, so can be computed from U in the Schur factorization of A.

If $U = \begin{bmatrix} U_{11} & u & U_{13} \\ & \lambda & v^H \\ & & U_{33} \end{bmatrix}$, solve:

$$(U_{11} - \lambda)\underline{s} = -u$$

$$(U_{33} - \lambda)^H \underline{t} = -v$$

then $x = \begin{bmatrix} s \\ 1 \\ 0 \end{bmatrix}$ $y = \begin{bmatrix} 0 \\ 1 \\ t \end{bmatrix}$

so $\text{cond}(\lambda) = \sqrt{(1 + s^H s)(1 + t^H t)}$

Algorithm CONDIT: Chan, Feldman & Parlett (1977)

Sensitivity of invariant subspaces:

if $\begin{bmatrix} A & \end{bmatrix}\left[\begin{array}{c|c} Q_1 & Q_2 \end{array}\right] = \left[\begin{array}{c|c} Q_1 & Q_2 \end{array}\right]\begin{bmatrix} U_{11} & U_{12} \\ & U_{22} \end{bmatrix}$,

and T is the linear operator defined by

$$T(X) = U_{11}X - XU_{22}$$

then the sensitivity of the subspace spanned by the columns of Q_1 is

$$\|T^{-1}\| \equiv \frac{1}{\underline{sep}(U_{11}, U_{22})}$$

If U_{11} reduces to a single eigenvalue λ,

i.e. $\begin{bmatrix} A & \end{bmatrix}\left[\begin{array}{c|c} x & * \end{array}\right] = \left[\begin{array}{c|c} x & * \end{array}\right]\begin{bmatrix} \lambda & * \\ & B \end{bmatrix}$

then $\text{cond}(x) = \|(B-\lambda I)^{-1}\|$

how to compute or estimate this efficiently?

Alternatively, use iterative refinement:

Algorithm SICEDR: Dongarra (1982)

Generalized Eigenvalue Problem

$$Ax = \lambda Bx, \quad x \neq 0$$

matrix pencil $A - \lambda B$

(assume A and B square)

If B is non-singular,

generalized problem is theoretically equivalent to the standard problem

$$B^{-1}Ax = \lambda x$$

<u>but</u> computing $B^{-1}A$ is:

undesirable if B is ill-conditioned

impossible if B is singular.

Generalized problems with singular B:

eigenvalues may be <u>infinite</u>
(these correspond to zero eigenvalues μ of the problem $\mu Ax = Bx$)

<u>or</u>

eigenvalues may be <u>undefined</u>
(e.g. if $Ax = 0$ and $Bx = 0$ for some $x \neq 0$, then $Ax = \lambda Bx$ for <u>all</u> values of λ)
in general this happens if
$\det(A - \lambda B) \equiv 0$ (for all λ);
such pencils are called <u>singular</u>

for the moment assume $A - \lambda B$ is <u>regular</u>

<u>Symmetric-definite generalized problems</u>

A and B symmetric
B positive-definite
(and not too ill-conditioned)

$\exists$ non-singular X such that

$$\left.\begin{aligned} X^T A X &= \operatorname{diag}(\alpha_1, \alpha_2, \ldots, \alpha_n) \\ X^T B X &= \operatorname{diag}(\beta_1, \beta_2, \ldots, \beta_n) \end{aligned}\right\}$$

so $\lambda_i = \alpha_i / \beta_i$ (real)

but X is not orthogonal.

<u>Cannot</u> preserve <u>symmetry</u> and use orthogonal transformations

Reduce symmetric-definite generalized problems to symmetric standard problems:

1. If $B = U^T U$ (Cholesky factorization),

$$Ax = \lambda Bx$$

$$\updownarrow$$

$$(U^{-T} A U^{-1})(Ux) = \lambda (Ux) \qquad \text{(EISPACK, NAG, Crawford, 1973)}$$

$$C \quad y = \lambda \quad y$$

2. If $B = Q \Delta Q^T$ (eigenvalues and eigenvectors)

$$Ax = \lambda Bx$$

$$\updownarrow$$

$$(\Delta^{-1/2} Q^T A Q \Delta^{-1/2})(\Delta^{1/2} Q^T x) = \lambda (\Delta^{1/2} Q^T x)$$

$$C \quad y = \lambda \quad y$$

Both methods are equivalent to forming $B^{-1}A$ and involve non-orthogonal transformations of A — but they work.

Unsymmetric Generalized Eigenvalue Problem

generalized Schur — or Stewart-factorization

$$\left.\begin{aligned} A &= Q U Z^H \\ B &= Q V Z^H \end{aligned}\right\}$$

$$\left.\begin{aligned} Q^H A Z &= U \\ Q^H B Z &= V \end{aligned}\right\} \quad \text{or}$$

$$\left.\begin{aligned} A Z &= Q U \\ B Z &= Q V \end{aligned}\right\} \quad \text{or}$$

U and V are upper triangular

Q and Z are unitary

eigenvalues are u_{ii}/v_{ii}

If A and B are real, then in real arithmetic

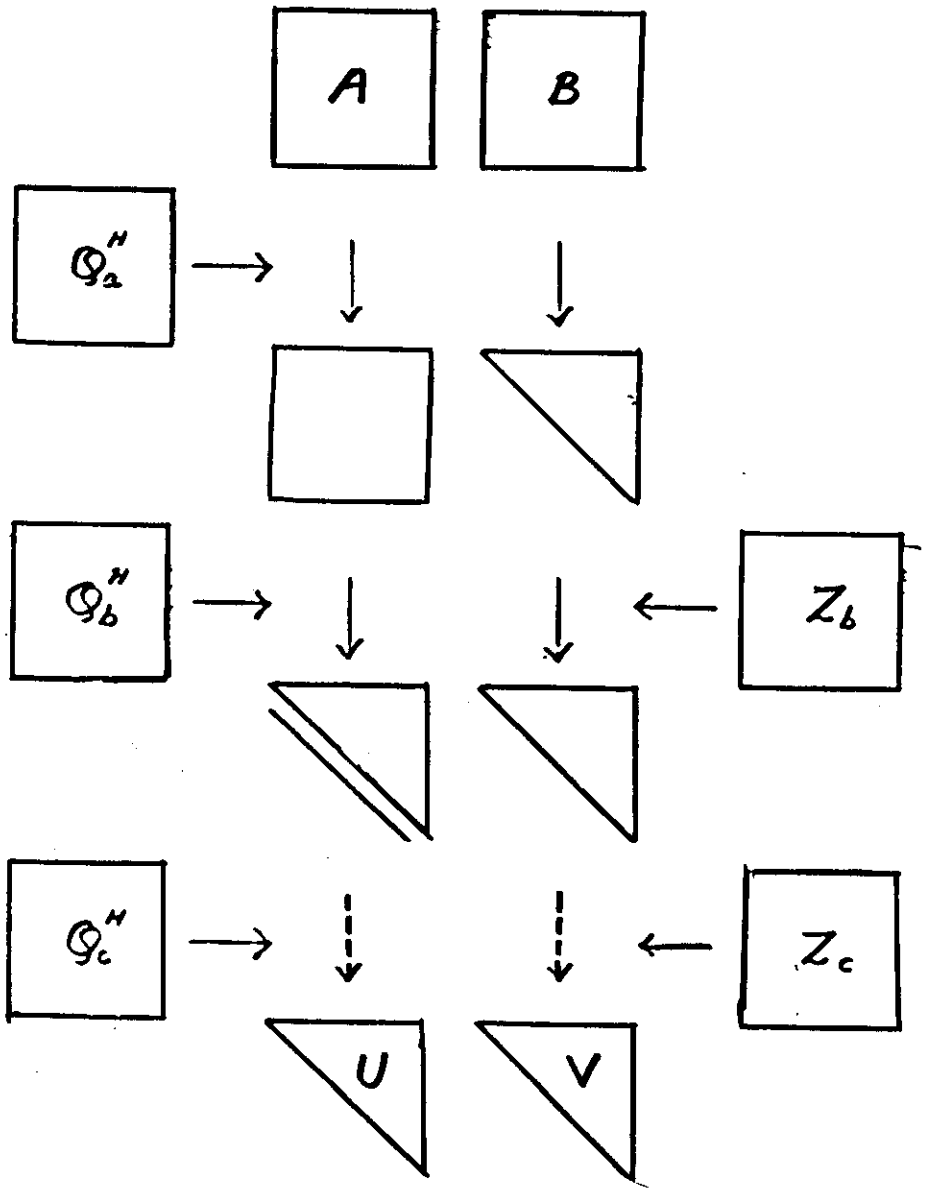
$$A = Q U Z^T$$
$$B = Q V Z^T$$

U is <u>block</u> upper triangular

V is upper triangular

Q and Z are orthogonal

QZ-algorithm (Moler and Stewart, 1973)



$$Q_c^H Q_b^H Q_a^H A Z_b Z_c = U$$
$$Q_c^H Q_b^H Q_a^H B Z_b Z_c = V$$

Computations associated with QZ-algorithm and the generalized Schur factorization:

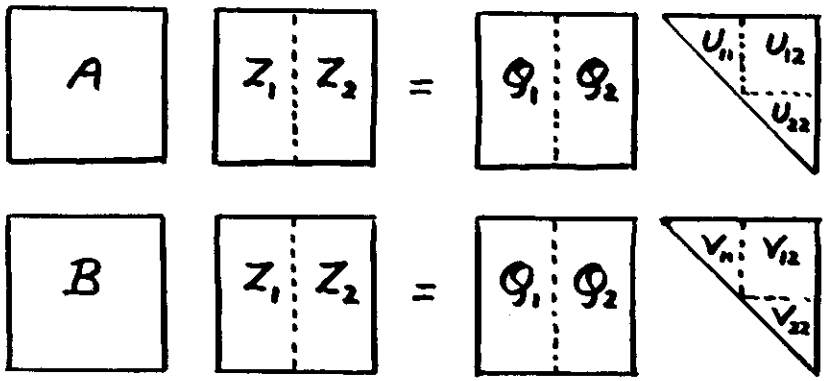
1. <u>balancing</u> the original problem (to improve accuracy of computed eigenvalues) (Ward, 1981)
2. <u>re-ordering</u> (by unitary transformations) Algorithm EXCHQZ (Van Dooren, 1982)
3. computing orthonormal bases for <u>deflating subspaces</u> (next slide)
4. solving <u>Riccati</u>-type equations (Van Dooren, 1981a)
5. stable reduction of problems to manageable form, e.g. <u>generalized Sylvester</u> equation (Epton, 1980)

$$AXD - EXB = C$$

Deflating Subspaces

A p-dimensional subspace $\mathcal{U}$ is a deflating subspace for $A - \lambda B$ if

$$\dim(A\mathcal{U} + B\mathcal{U}) \leq p$$



$$AZ_1 = Q_1 U_{11}$$

$$BZ_1 = Q_1 V_{11}$$

The columns of Z_1 form a basis for a deflating subspace of $A - \lambda B$ since they are mapped by both A and B onto the space spanned by the columns of Q_1.

Singular (or almost singular) Pencils

If $A - \lambda B$ is singular, then in the generalized Schur factorization we would have in exact arithmetic:

$$u_{ii} = v_{ii} = 0$$

hence $\lambda \left(= u_{ii}/v_{ii}\right)$ is undefined.

Numerically, if $A - \lambda B$ is almost singular, then in the results of the QZ algorithm we have:

u_{ii} and v_{ii} both very small

When this happens, the <u>other</u> ratios u_{jj}/v_{jj} can assume <u>arbitrary</u> values (Wilkinson, 1979)!

Algorithmic principles:

1. use only orthogonal (or unitary) transformations
2. apply transformations to original data
3. preserve structure of the problem

to be observed as far as possible

LINEAR ALGEBRA 3:

SPARSE PROBLEMS

for linear equations and eigenvalue problems

Sparse Systems of Linear Equations

- Direct methods
  - use triangular factorizations, like method for dense systems
  - non-zero elements of A and of its triangular factors are stored explicitly in elaborate data structures
  - want to minimize "fill-in"
- Iterative methods
  - ask user to compute Ay for given y (or By where B is a matrix derived from A)
  - want to minimize number of iterations

Direct methods for symmetric positive-definite systems

for dense matrices, use the factorization

$$A = LL^T$$

for sparse matrices, use

$$PAP^T = LL^T$$

where P is a permutation matrix chosen so as to "minimize" fill-in

Example no permutations

Figure 1.1.1 Nonzero pattern of a 35 by 35 matrix A.

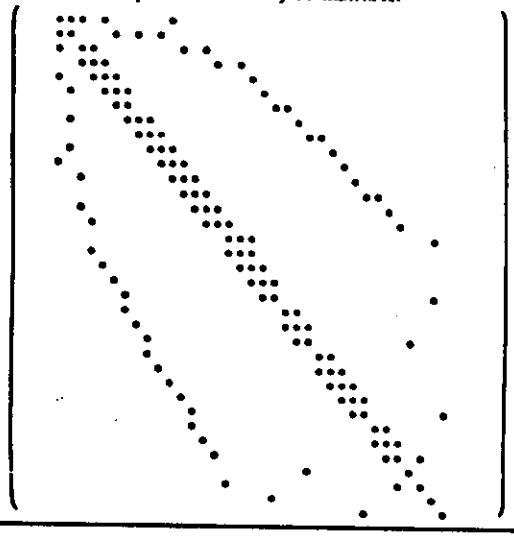
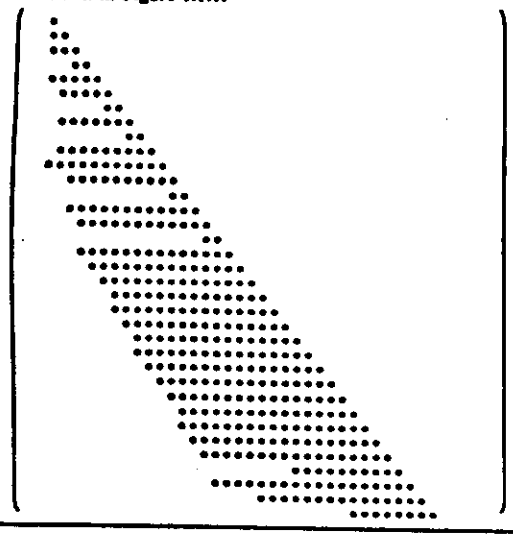


Figure 1.1.2 Nonzero pattern of the Cholesky factor L for the matrix whose structure is shown in Figure 1.1.1.



Example P chosen to minimize bandwidth

Figure 1.1.3 The structure of A', a symmetric permutation of the matrix A whose structure is shown in Figure 1.1.1.

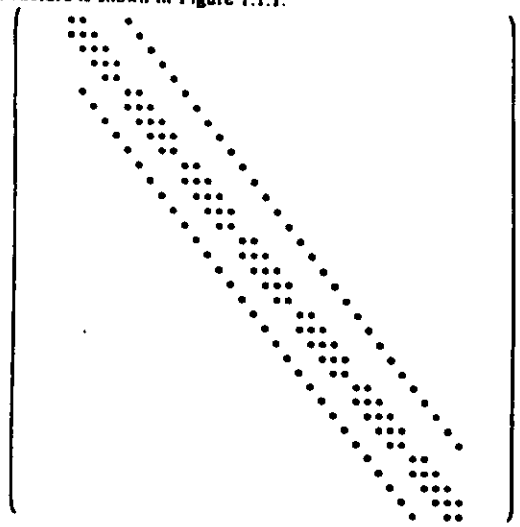
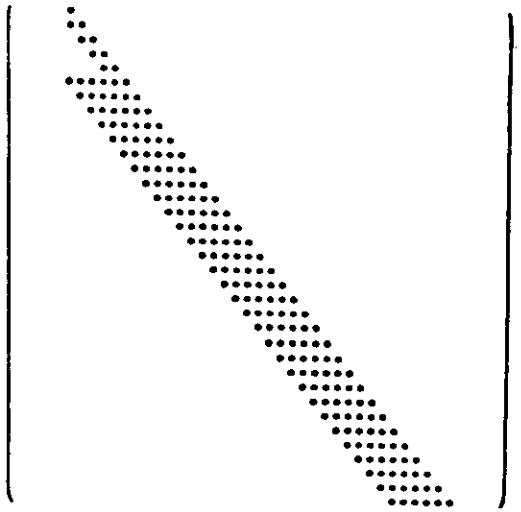


Figure 1.1.4 The structure of L', the Cholesky factor of A', whose structure is shown in Figure 1.1.3.



Example P chosen to minimize fill-in

Figure 1.1.5 The structure of A'', a symmetric permutation of the matrix A whose structure is shown in Figure 1.1.1.

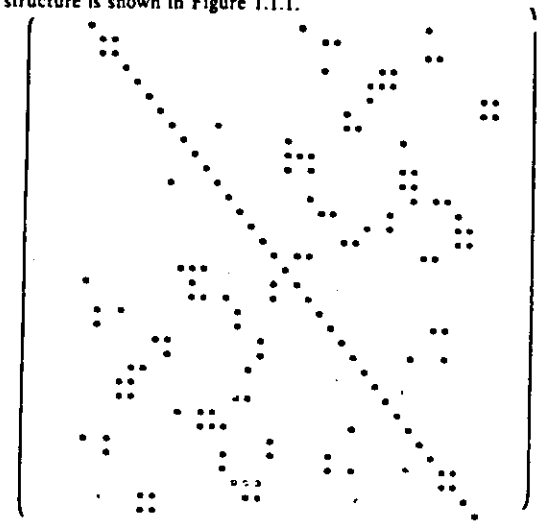
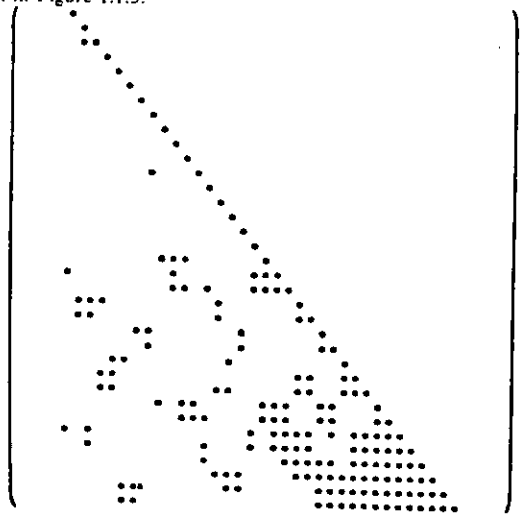


Figure 1.1.6 The structure of L'', the Cholesky factor of A'' whose structure is shown in Figure 1.1.5.



Features of a sparse matrix code (for a direct method):

Phase	Features
ANALYSE	• Transfer data to internal data structures
	• Determine good pivotal sequence
	• Prepare data structures for later phases
FACTORIZE	• Transfer new numerical data to internal data structures
	• Factorize, using pre-determined pivotal sequence
	• Monitor stability, if necessary
SOLVE	• Apply permutations and forward and backward substitutions to new right hand side

Direct methods for unsymmetric systems

for dense matrices, use the factorization

$$PA = LU$$

where P is a permutation matrix chosen to maintain numerical stability

for sparse matrices, use

$$PAQ = LU$$

where P and Q are permutation matrices, chosen so as to:

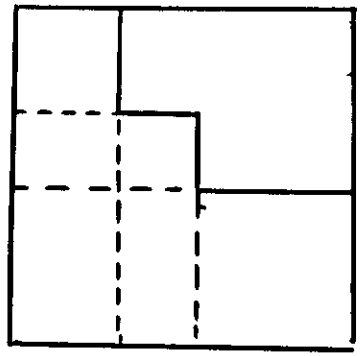
- "minimize" fill-in
- maintain numerical stability

(in practice use Markowitz criterion)

hence ANALYSE and FACTORIZE phases must be combined

Refinements (which may be provided as options in good software)

A. permute to block triangular form



only the diagonal blocks need be factorized

B. switch to full-matrix code when reduced matrix is sufficiently dense

C. use a 'drop tolerance': set elements of L and U to zero if they are less than a threshold value. This gives an approximate, but sparser, factorisation. Then use iterative refinement to get accurate solution.

Examples of the benefits of these refinements:

Note: we give just one favourable example of each. The actual benefits vary from one sparse matrix to another.

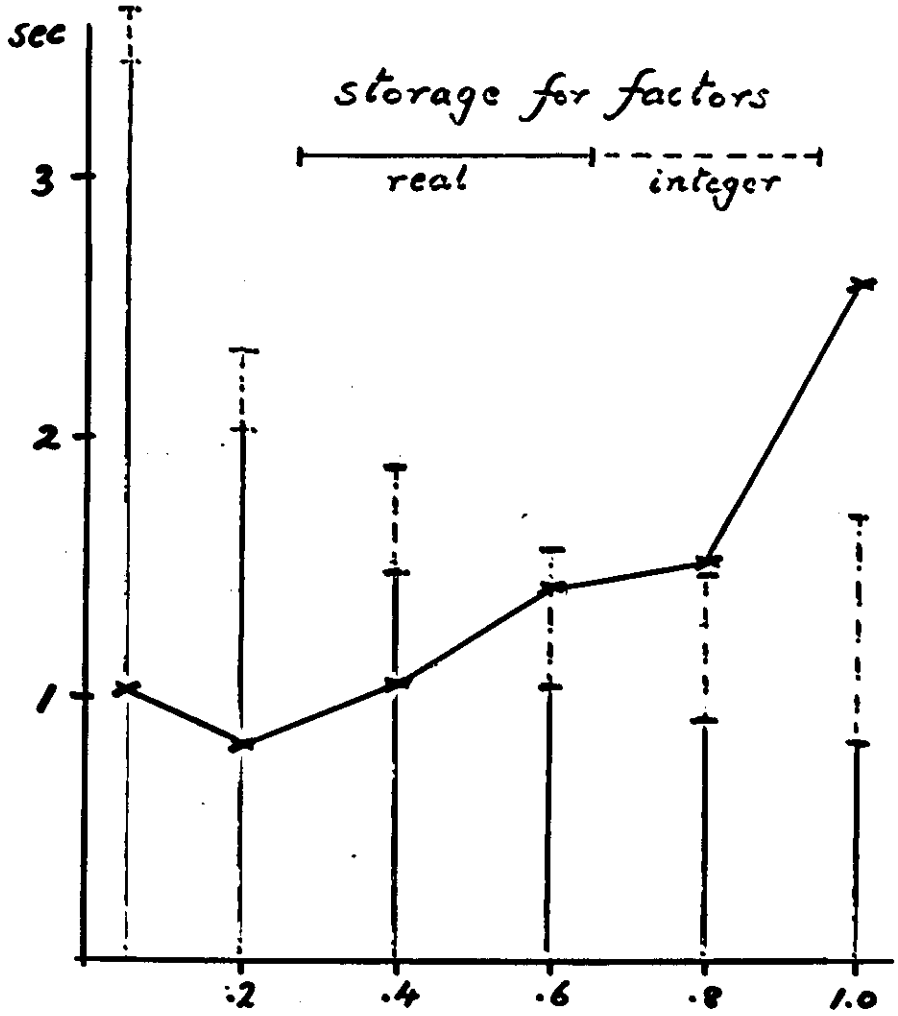
A. ($n = 822$, density .007)

	Time (sec)
Block triangularization	.25
ANALYSE-FACTORIZE *after* block triangularization	.94
ANALYSE-FACTORIZE *without* block triangularization	1.76

(450 blocks, largest block of order 217)

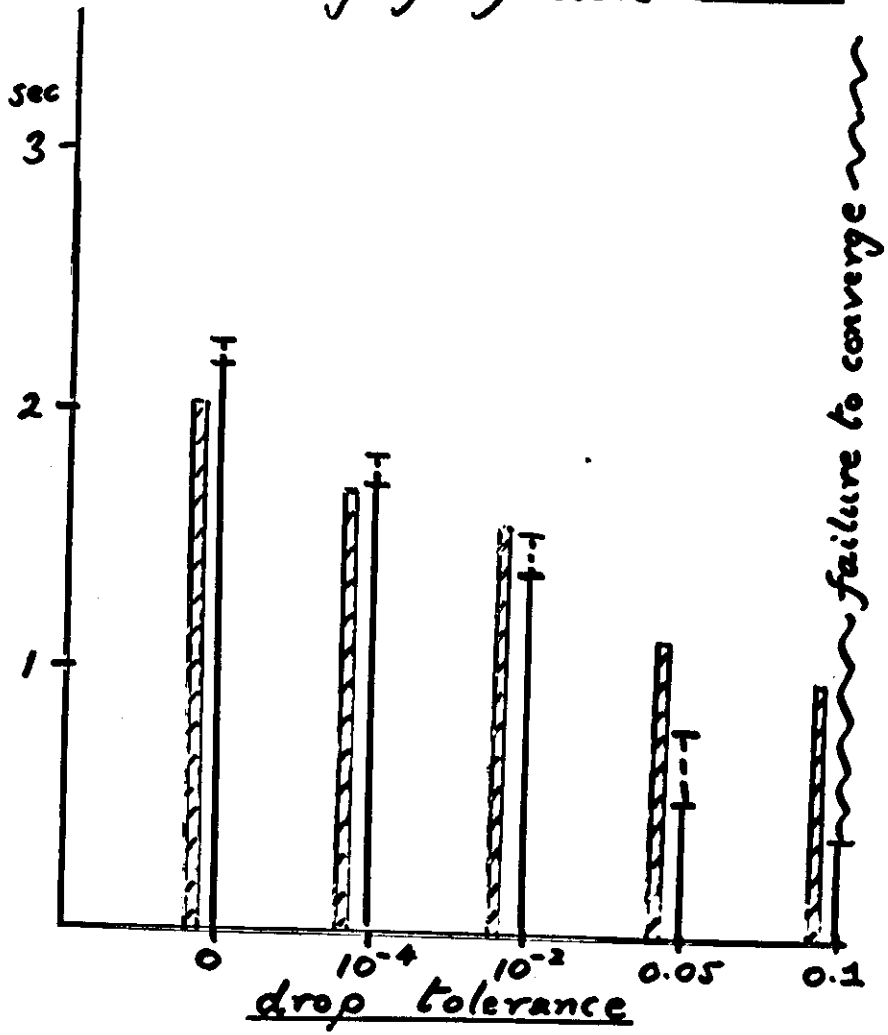
B.

Time for ANALYSE-FACTORIZE
(Cray-1, $n = 1176$, density .01)



C. Time for ANALYSE-FACTORIZE (——)
and SOLVE (- - - -)

storage for factors ▭



Sparse Eigenvalue Problems

- applying unitary transformations would cause a large amount of fill-in
- may not want, or be able, to store the non-zero elements of A explicitly
- may only want to compute a few eigenvalues, possibly with their eigenvectors.

hence, need quite different algorithms

Do not need explicit access to non-zero elements of A;
ask user to compute Ax for given x.

We shall discuss two types of algorithm for <u>symmetric</u> problems:

<u>Simultaneous iteration:</u>

- for finding a few of the largest eigenvalues (in absolute value) and their eigenvectors
- well understood
- software available

<u>Lanczos methods</u>

- finds extreme eigenvalues first, but can be used to find all eigenvalues
- potentially much faster than simultaneous iteration
- development of good general purpose algorithms is still a subject of research
- software not so easily accessible

<u>Simultaneous iteration</u>

if U is a p-dimensional subspace of $\mathbb{R}^n$,

$$U,\ AU,\ A^2U,\ A^3U,\ \ldots$$

converges to the p-dimensional invariant subspace corresponding to the p largest eigenvalues (under suitable conditions)

(compare the power method:

if v is an arbitrary vector,

$$v,\ Av,\ A^2v,\ A^3v,\ \ldots$$

converges to the eigenvector corresponding to the largest eigenvalue (under suitable conditions))

Power method:

v_0

$y_1 = Av_0$

$v_1 = y_1 / \|y_1\|$

$y_2 = Av_1$

⋮

convergence factor:

$|\lambda_2 / \lambda_1|$

Basic subspace iteration:

V_0

$Y_1 = AV_0$

$V_1 = Q_1 \quad (Y_1 = Q_1 R_1)$

$Y_2 = AV_1$

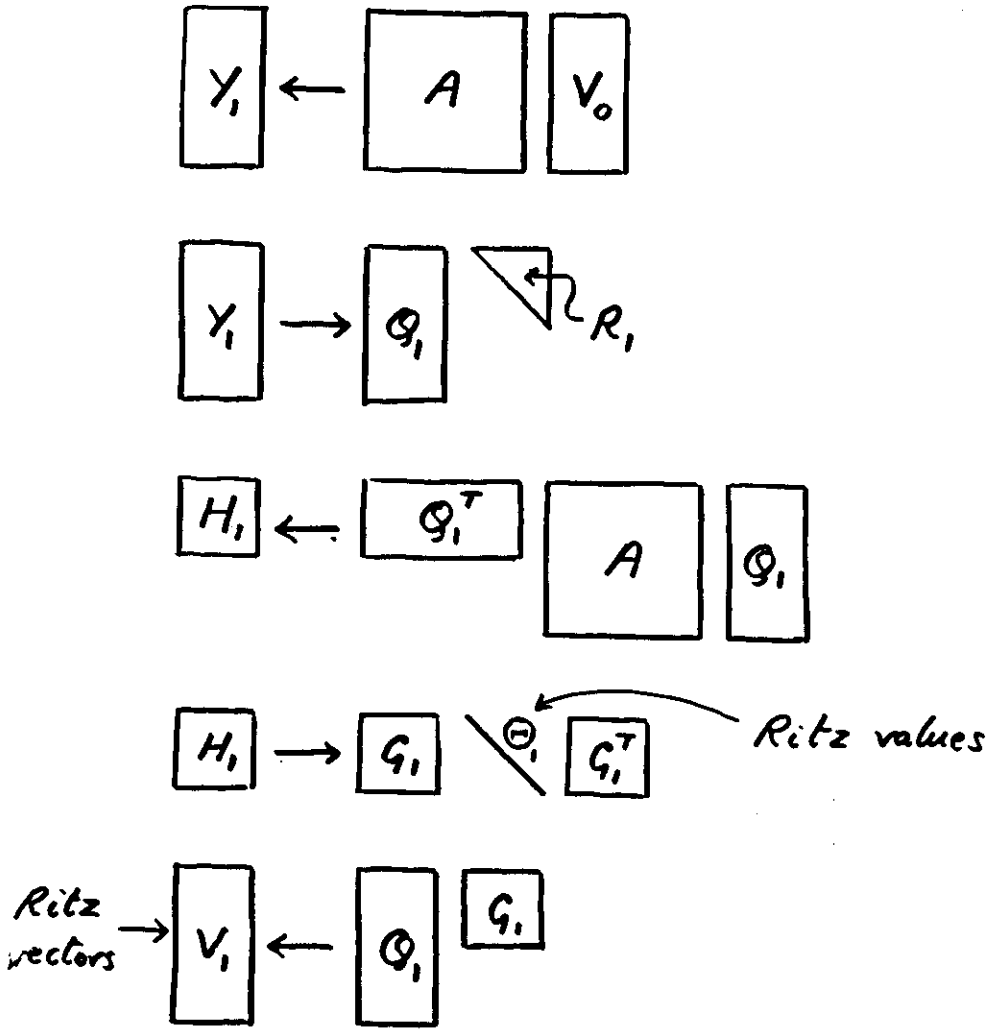
⋮

convergence factor:

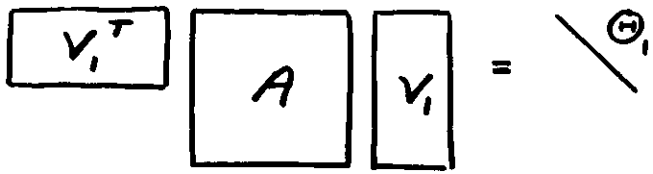
$|\lambda_{p+1} / \lambda_p|$

where $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_p| > |\lambda_{p+1}| \geq \ldots$

Subspace iteration with Rayleigh Ritz procedure:



hence



(compare Rayleigh quotient $\theta = v^T A v$ which minimises $\|Av - \theta v\|$)

The Ritz values and Ritz vectors are in a certain sense the best approximations to p eigenvalues and eigenvectors from the subspace AV.

$$\theta_j^{(1)}, \theta_j^{(2)}, \theta_j^{(3)}, \ldots \longrightarrow \lambda_j$$

with convergence factor $|\lambda_{p+1}/\lambda_j|$ and the Ritz rector converges to the eigenvector.

Practical algorithms include many refinements, e.g.

- to reduce cost of Rayleigh-Ritz procedure
- to accelerate convergence

and so on

RITZIT: (Rutishauser, 1970) in Algol 60

EA12: (Harwell)
SIMITZ: (Nikolai, 1979)
F02FJF: (NAG, Mark 11)
} in Fortran

Lanczos methods

work with a sequence of subspaces of *increasing* dimension:

$$\mathcal{K}_1 = \{q\}$$

$$\mathcal{K}_2 = \{q, Aq\}$$

$$\mathcal{K}_3 = \{q, Aq, A^2q\}$$

$$\vdots$$

(*Krylov* subspaces)

We need an orthonormal basis of each $\mathcal{K}_i$; if we orthogonalise $q, Aq, A^2q, \ldots$ in natural order, we obtain Q_i such that

$Q_i^T A Q_i$ is *tridiagonal* $(= T_i)$

hence Rayleigh-Ritz procedure is economical.

Basic iteration:

1. compute Q_i and T_i
   (see next slide)
2. compute eigenvalues and eigenvectors of T_i:
   $$T_i = S_i \Theta_i S_i^T$$
3. compute Ritz vectors
   $$Y_i = Q_i S_i$$
4. test for convergence

In fact Ritz vectors need not be computed at each iteration because accuracy of Ritz pair (θ_j, y_j) is given by

$$\| A y_j^{(i)} - \theta_j^{(i)} y_j^{(i)} \| = \beta_i | s_{ij}^{(i)} |$$

to compute Q_i and T_i use 3-term recurrence:

$$\beta_{i+1} q_{i+1} = A q_i - \alpha_i q_i - \beta_i q_{i-1}$$

where $\alpha_i = q_i^T A q_i$

and β_{i+1} is a normalizing factor.

$$Q_3 = \left[\begin{array}{c|c|c} q_1 & q_2 & q_3 \end{array}\right] \qquad T_3 = \begin{bmatrix} \alpha_1 & \beta_2 & \\ \beta_2 & \alpha_2 & \beta_3 \\ & \beta_3 & \alpha_3 \end{bmatrix}$$

(N.B. it is essential to compute this recurrence in a stable manner)

In <u>exact</u> arithmetic the iteration terminates when $\beta_{i+1} = 0$ with

$$A Q_i = Q_i T_i$$

In <u>finite-precision</u> arithmetic:

- computed quantities deviate markedly from theoretical values
- $\{q_j\}$ lose orthogonality

<u>but</u>

this starts to happen just when a Ritz pair (θ_j, y_j) converges to an eigenpair of A.

If we carry on, new Ritz pairs emerge which converge to the same eigenpair of A.

Understanding of algorithm is largely due to Paige.

Practical Lanczos algorithms:

(complete re-orthogonalization: ensures that $\{q_j\}$ are always orthogonal; effective, but very expensive)

selective re-orthogonalization: occasionally (when?) re-orthogonalize q_{i+1} against selected Ritz vectors (which?); effective, can become expensive for several eigenvalues, multiple eigenvalues not detected (Parlett and Scott, 1979)

block Lanczos: basic recurrence works with a block of p orthonormal vectors:

$$Q_{i+1} B_{i+1} = A Q_i - Q_i A_i - Q_{i-1} B_i^T$$

satisfactory for eigenvalues with multiplicity $\leq p$; can be combined with selective orthogonalization.

Parlett and Reid (1981): keep a record of intervals on the real line within which eigenvalues of A are known to lie, and attempt to refine them at each iteration; efficient (no re-orthogonalization, no computation of eigenvalues of T_i), but cannot detect multiplicity of any eigenvalue (EA15, Harwell)

Sparse SVD

Apply Lanczos method to $\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$

taking advantage of special structure.

This yields tridiagonal matrices T_i which have zero diagonals.

Computing the eigenvalues and eigenvector of such a tridiagonal matrix is equivalent (via a permutation) to doing SVD of a bidiagonal matrix of half the size.

In fact the whole process can be regarded as a Lanczos-type approach to generating a sequence of bidiagona matrices B_i whose singular values approximate those of A.

If

$$A^T Q = P \begin{bmatrix} \alpha_1 & \beta_2 & & \\ & \alpha_2 & \beta_3 & \\ & & \ddots & \ddots \\ & & & \ddots \end{bmatrix}$$

and

$$A P = Q \begin{bmatrix} \alpha_1 & & & \\ \beta_1 & \alpha_2 & & \\ & \beta_2 & \ddots & \\ & & \ddots & \ddots \end{bmatrix}$$

then:

$$A^T q_i = \alpha_i p_i + \beta_i q_{i-1}$$
$$A p_i = \alpha_i q_i + \beta_{i+1} q_{i+1}$$

hence we can compute $\alpha_i, \beta_i, p_i, q_i$ by a Lanczos-type recurrence:

$$\alpha_i p_i = A^T q_i - \beta_i q_{i-1}$$
$$\beta_{i+1} q_{i+1} = A p_i - \alpha_i q_i$$

where α_i and β_{i+1} are normalizing factors.

$$\begin{aligned} \alpha_j v_j &= A^T u_j - \beta_j v_{j-1} \\ \beta_{j+1} u_{j+1} &= A v_j - \alpha_j u_j \end{aligned} \qquad (9.4)$$

Sparse symmetric-definite generalized problems

1. implicitly form $C = B^{-1}A$:
   when algorithm requires $z = Cv$,
   compute z by solving $Bz = Av$.

2. implicitly form $C = U^{-T}AU^{-1}$
   (assuming sparse Cholesky factorization of B):
   when algorithm requires $z = Cv$,
   compute w by solving $Uw = v$
   and then z by solving $U^Tz = Aw$.

3. implicitly apply Lanczos recurrence to $B^{-1}A$:

$$\beta_{i+1}Bq_{i+1} = Aq_i - \alpha_i Bq_i - \beta_i Bq_{i-1}$$

(NAG routine F02FJF uses 1. in simultaneous iteration)

Iterative methods for linear equations

The Lanczos algorithm is also the basis of iterative algorithms for solving systems of linear equations:

the conjugate gradient method for symm. pos. def. A

the algorithm SYMMLQ (Paige & Saunders) for symm. indefinite A

the algorithm LSQR (Paige & Saunders) for linear least squares problems

These algorithms are sufficiently well understood to be implemented in robust software (SYMMLQ and LSQR are included in NAG Library).