Unit I

Linear Algebra

Linear algebra

- direct solution methods
  - Gaussian and Gauss-Jordan elimination with pivoting
  - matrix factorizations (LU, Cholesky & QR)
  - quantifying inaccuracy
  - conditioning
- iterative solution methods
  - Jacobi & Gauss-Siedel
  - iterative improvement
- over-determined systems
  - singular value decomposition
- finding eigenvalues

Tasks of computational linear algebra

Solution of a linear system $A \cdot x = b$
- A is square coefficient matrix
- b is a known vector of constants (or multi-vectors)
- x is an unknown solution vector

Inverse of A^{-1}
- equivalent of first task with n unit basis vectors as b's

Eigenvalues $A \cdot x = \lambda x$
- extremely valuable for many applications

Determinant det(A)
- rarely needed or useful

Linear systems: mathematical facts

- m equations and n unknowns
- n=m has a unique solution if
  - no row is a linear combination of the others (row degeneracy), or
  - no column is a linear combination of the others (column degeneracy)
- non-singular systems have a unique solution
- mathematically these statements are exact
- but numerically.....

Linear system: numerical issue

If a system is too close to linear dependence ....
- an alogorithm may fail altogether to get a solution
- round off errors can produce apparent linear dependence at some point in the solution process
- the numerical procedure will fail totally

Linear systems: numerical issue

If a system is too close to linear dependence ....
- an algorithm may still work but produce nonsense
- accumulated roundoff errors can swamp the solution
- very close cancellations occur in close-to-singular systems
- particularly if n large
- not algorithmic failure, but answer is (wildly) incorrect
- can confirm error by direct substitution in original equations

When is sophistication necessary?

- sophisticated methods can detect and correct numerical pathologies
- rough guide for a not-too-singular n×n system
  - n < 20...50 single precision
  - n < 200...300 double precision
  - n = 1000 ok if equations are sparse (special technique takes advantage of sparseness)
- close-to-singular can be a problem even for very small systems

Under-determined system

- m<n, or m=n with degenerate equations
- fewer equations than unknowns
- may be no solution, or
- may be an infinite number of solutions
  - particular solution (x_p) + linear combination of n-m vectors in the nullspace of A (i.e. $A \cdot x=0$)
  - this becomes an optimization problem

Over-determined system

- m > n and not degenerate
- inconsistent (no solution)
- may be derived from large experimental datasets
  - experimental errors
- the best compromise solution might be required
  - closest to satisfying all equations
  - requires quantification of 'closeness' to correct solution
  - sum of squares of differences between left and right hand sides is minimized (linear least squares problem)
  - singular value decomposition is a powerful technique

Solution techniques for linear systems

- direct methods
  - predictable number of steps
- iterative methods
  - converge in as many steps as necessary
  - useful when the battle against loss of significance is being lost (n large and/or close to singular)
- combination
  - direct solution then improved by iteration
  - useful for close-to-singular systems

Back and forward substitution

- an *upper triangular* system Ux = b has $u_{ij} = 0$ i>j
  - easily solved by back substitution
  - $x_n, x_{n-1}, \ldots, x_1$ successively
- a *lower triangular* system Lx = b has $l_{ij} = 0$ j>i
  - easily solved by forward substitution
  - $x_1, x_2, \ldots, x_n$ successively
- triangular systems are numerically straightforward

Direct methods

- tackle a general system Ax=b by ....
- ....transforming it to a triangular system or some combination of triangular systems
- numerical issues can occur in the transformation steps

Matlab methods

- Matlab has the handy-dandy backslash operator
- to solve a linear system Ax=b write x = A\b
- looks like a matrix inverse but it isn't
- various approaches are used intelligently according to characteristics of the system
- Matlab recognizes
  - a triangular system and applies a simple substitution algorithm
  - a permuted triangular system and unpermutes it first
  - specialized types of systems
  - potential numerical problems

Gauss-Jordan elimination: Ax=b

- PRO
  - efficient method for matrix inversion
  - produces both the solution(s), for (multiple) b_j, and the inverse A^{-1}
  - numerically stable if *pivoting* is used
  - straightforward, understandable method
- CON
  - all b_js must be stored and manipulated simultaneously
  - three times slower than alternatives when inverse is not required
  - inverse matrix prone to roundoff error

Row operations vs....

- inverse matrix A^{-1} and solutions x_j can be built up in the storage locations of A and b_j respectively
- elementary row operations correspond to *pre-multiplication* by elementary matrices:

$$A \cdot x = b$$
$$(...R_3 \cdot R_2 \cdot R_1 \cdot A) \cdot x = ...R_3 \cdot R_2 \cdot R_1 \cdot b$$
$$(I_n) \cdot x = ...R_3 \cdot R_2 \cdot R_1 \cdot b$$
$$x = ...R_3 \cdot R_2 \cdot R_1 \ b$$

- x can be built-up in stages since the R matrices are multiplied in the order of acquisition

... column operations

- elementary column operations correspond to *post-multiplication* by elementary matrices:

$$A \cdot x = b$$
$$A \cdot C_1 \cdot C_1^{-1} \cdot x = b$$
$$A \cdot C_1 \cdot C_2 \cdot C_2^{-1} \cdot C_1^{-1} \cdot x = b$$
$$(A \cdot C_1 \cdot C_2 \cdot C_3 ...) \cdot (...C_3^{-1} \cdot C_2^{-1} \cdot C_1^{-1}) \cdot x = b$$
$$(I_n) \cdot (...C_3^{-1} \cdot C_2^{-1} \cdot C_1^{-1}) \cdot x = b$$
$$x = C_1 \cdot C_2 \cdot C_3 ... \cdot b$$

- the C matrices must be stored until the last step
- applied to b in the reverse order of acquisition
- a fundamental computational difference between elementary row and column operations

Gauss-Jordan elimination

- augmented matrix $A' = [\ A \mid b_1 \mid \cdots \mid b_j \mid I_n\ ]$
- operations which do not change the solutions
  1. Replace a row of A′ by a linear combination of itself and any other row(s).
  2. Interchange two rows of A′.
  3. Interchange two columns of A and corresponding rows of b_j and x.
- basic G-J elimination uses only operation #1 but ....

The need for pivoting

- .... fails mathematically when a zero pivot is encountered
- and fails numerically with a too-close-to-zero pivot
- the fix is *partial pivoting*
  - use operation #2 to place a desirable pivot entry in the current row
  - usually sufficient for stability
- using operation #3 as well gives *full pivoting*

Encountering a zero pivot

- pivoting is essential to avoid total failure of the algorithm if you run into a zero pivot
- try Ax=b with

$$A = \begin{bmatrix} 2 & 4 & -2 & -2 \\ 1 & 2 & 4 & -3 \\ -3 & -3 & 8 & -2 \\ -1 & 1 & 6 & -3 \end{bmatrix} \qquad b = \begin{bmatrix} -4 \\ 5 \\ 7 \\ 7 \end{bmatrix}$$

Choosing the pivot entry

- can choose from elements that are both:
  - on rows below (or on) the one that is being normalized
  - on columns to the right (or on) the one that is about to be eliminated
- partial pivoting
  - restricts the pivot choices to the column being eliminated
  - easier than full pivoting because permutations of the vector elements don't need to be recorded
  - almost as good numerically as full pivoting

Example: the value of pivoting

- pivoting can be essential to avoid inaccuracy
- illustrate using the toy computer with 4 significant digits to exaggerate the effect
- try Ax=b with

$$A = \begin{bmatrix} 0.0001 & 0.5 \\ 0.4 & -0.3 \end{bmatrix} \qquad b = \begin{bmatrix} 0.5 \\ 0.1 \end{bmatrix}$$

A desirable pivot choice

- select the entry with the largest absolute value
- in theory this depends on the original scaling of the equations
- experimental data may need to be pre-processed
- scale the original equations so that the largest coefficient (abs val) in each row is one

Conditioning

- a measure of the sensitivity to perturbations in parameters, e.g. due to
  - data collection, or caused by .....
  - roundoff error
- a function of the problem itself
- independent of the algorithm used to solve it
- determines the limits to attainable accuracy

Stability

- the property of not ampliying errors
- a function of the algorithm used to solve a problem
- a stable algorithm + a well-conditioned problem
  → the right answer

BUT what does *well-conditioned* mean?

Digression: vector and matrix norms

- how to compare 'closeness' of two vectors x & y?
- look at

$$\frac{||x-y||}{||x||} \leq \delta$$

- need concept of length or magnitude or *norm* ||x|| of a vector x....
- vector norm properties:
  (1) ||x||>0 all x≠0 (2) ||ax|| = |a| ||x|| (3) ||x+y||≤||x||+||y||
- let the vector be n-dimensional x = (x_1, ... ,x_n)

Digression: vector and matrix norms

- the vector *p-norms* (*L_p norms*) are defined by

$$||x||_1 = \sum_{i=1}^{n} |x_i|$$

$$||x||_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$$

$$||x||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

$$||x||_\infty = \max(|x_1|, \cdots, |x_n|)$$

- $||x||_2$ is the usual *Euclidean* norm
- $||x||_2 = \sqrt{x^T x}$ is another way of viewing the 2-norm

Digression: vector and matrix norms

- Matlab has a built-in vector p-norm function: *norm(x,p)*
- convergence of a vector sequence is independent of which p-norm is used to check
  - see Matlab example in *normcompare*
- L_2 norm is most often used
- $||x||_\infty \leq ||x||_2 \leq ||x||_1$
- L_∞ norm is useful when computationally challenged
- what about matrix norms?

Digression: vector and matrix norms

- y=Ax transforms vector x into y
  - A rotates and/or stretches x
- consider and compare the effect of A on a unit vector x [i.e. x so that $||x||_2 = 1$]
- the 'largest' Ax value is a measure of the geometric effect of the transformation A
- the L_2 norm of A is $||A||_2 = \max_{||x||_2=1} ||Ax||_2$
- $||A||_2$ is not easy to calculate
- also called the *spectral norm* of A because $||A||_2 = \sqrt{\max(\lambda_i)}$ where λ_i is an eigenvalue of $A^T A$

Digression: vector and matrix norms

- two other useful and easier-to-calculate matrix norms...
- $||A||_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$ *column sum norm*
- $||A||_\infty = \max_{1 \leq i \leq m} \sum_{i=1}^{n} |a_{ij}|$ *row sum norm*
- Matlab has built-in matrix norm function *norm(A,p)*
- ||A|| satisfies vector norm properties PLUS...
  $||AB|| \leq ||A||||B||$ and, in particular, $||Ax|| \leq ||A||||x||$

Quantifying inaccuracy: the residual

- $\hat{x}$ is a numerical solution to Ax = b
- define *residual* r to represent the error $r = b - A\hat{x}$
- then relative error $E_{\text{rel}} = \frac{||x - \hat{x}||}{||x||} \ll 1$
  implies the relative residual $\frac{||r||}{||b||} \ll 1$ also
- B U T not the converse.....

The residual

$$\begin{aligned} r &= b - A\hat{x} \\ &= b - A\hat{x} + (Ax - b) \quad \text{..... the added term = zero} \\ &= A(x - \hat{x}) \end{aligned}$$

So $x - \hat{x} = A^{-1}r$

And so $||x - \hat{x}|| \leq ||A^{-1}||||r||$ [eqn 1]

Also Ax = b gives $||b|| \leq ||A||||x||$

$$\frac{1}{||x||} \leq \frac{||A||}{||b||} \quad \text{[eqn 2]}$$

The residual

Combining eqns 1 & 2:

$$\frac{||x - \hat{x}||}{||x||} \leq \frac{||A^{-1}||||r||||A||}{||b||}$$

$$E_{\text{rel}} \leq \frac{k(A)||r||}{||b||}$$

- k(A) = ||A|| $||A^{-1}||$ is called the *condition number* of A
- it is possible for the relative error to be large even when r « 1, e.g. when k(A) » 1
- an *ill-conditioned* problem has a large condition number
- so converse statement is false, i.e. a small residual does not guarantee accuracy for an ill-conditioned problem

Condition number

- k(A) is a measure of the sensitivity of x^ to perturbations in A or b
- $1 \leq k(A) \leq \infty$
- k(A) can be measured with any p-norm
- k(A) is a mathematical property of the coefficient matrix A
- in exact math a singular matrix has $k(A) = \infty$
- k(A) is an indication of how close a matrix is to being *numerically* singular
- *any* algorithm will produce a solution that is sensitive to perturbations in A or b if k(A) is large
- large k(A).....bad problem

Condition number

- when k(A) is large the residual r is useless to assess accuracy of x^
- when k(A)~1 the residual r is a good measure of the accuracy of x^

$$\frac{1}{k(A)}\frac{||r||}{||b||} \leq E_{\text{rel}} \leq k(A)\frac{||r||}{||b||}$$

- Matlab has a built-in function: *cond(A)*
- suppose the coefficients in A are not known exactly, so we're really solving (A+E)x=b

Condition number

- $\hat{A} = A + E$ is the perturbed A matrix
- the computed solution is $\hat{x}$ so that $\hat{A}\hat{x} = b$
- the exact solution is $Ax = b$
- we want to know how big is $x - \hat{x}$?
- $x = A^{-1}b = A^{-1}\hat{A}\hat{x} = A^{-1}(A + \hat{A} - A)\hat{x}$

  $= (I + A^{-1}(\hat{A} - A))\hat{x} = \hat{x} + A^{-1}(\hat{A} - A))\hat{x}$
- so $x - \hat{x} = A^{-1}E\hat{x}$ and taking norms

  $||x - \hat{x}|| \leq ||A^{-1}||||E||||\hat{x}|| = ||A^{-1}||||A||\frac{||E||}{||A||}||\hat{x}||$

Condition number

- the final equation: $\frac{||x - \hat{x}||}{||\hat{x}||} \leq k(A)\frac{||E||}{||A||}$
- condition number scales the largest relative error in the solution to relative error in the matrix coefficients
- e.g. if ||E|| / ||A|| ~10^{-4} and k(A)=1000 then x^ may have only one digit accuracy
- Matlab tries to warn when cond(A) is large

Factorization methods

- disadvantage of Gaussian elimination
  - all righthand side b_js must be known in advance
- LU decomposition keeps track of the steps in Gaussian elimination (book-keeping)
  - the result can be applied to any future b required
- A is decomposed or factorized as A=LU
  - L lower triangular
  - U upper triangular
- why do this?

LU factorization

- A = LU, L lower triangular, U upper triangular
- A x = b becomes LUx = b, or equivalent to ....
- .... Ly = b solved by forward-substitution, followed by ....
- .... Ux = y solved by back-substitution
- it isn't necessary to record the interim y vector
  - it's temporary

LU factorization

- U is what would have been obtained by Gaussian elimination and ...
- L records the information necessary to undo the elimination steps
- the two interim systems are trivial to solve since both are triangular
- work effort goes into the factorization steps to get L and U
- how to do this?

LU factorization: the book-keeping

- first assume no pivoting is required
- steps in Gaussian elimination involve pre-multiplication by elementary matrices
  - these are trivially invertible
  - no flops required, only inspection and book-keeping
- $A = (R_1^{-1} \cdot R_1) \cdot A = \ldots =$
  $= (R_1^{-1} \cdot R_2^{-1} \cdot R_3^{-1} \ldots \cdot \ldots R_3 \cdot R_2 \cdot R_1) \cdot A$
  $= (R_1^{-1} \cdot R_2^{-1} \cdot R_3^{-1} \ldots) \cdot (\ldots R_3 \cdot R_2 \cdot R_1 \cdot A)$
  $=$ L U
- L and U
  - can be built up by inspection
  - can occupy the same storage location as the original A (*in-situ*)

LU factorization: the book-keeping

- entries for U are simply obtained from the Gaussian elimination
- entries for L are the negatives of the multipliers in the row transformation for each step
  - if the step is $R_j \leftarrow R_j + aR_i$ then -a is the entry in L_{ij}
- LU is not unique
  - factors can be re-arranged between L & U
  - represents the effect of using different $R_j \leftarrow cR_j$ which puts a 1/c entry in the L_{jj}

LU factorization: example

Find an LU factorization [by hand] for $A = \begin{bmatrix} 1 & 2 & -3 \\ -3 & -4 & 13 \\ 2 & 1 & -5 \end{bmatrix}$

LU is not unique: example

Find two different LU factorizations [by hand] for $A = \begin{bmatrix} 2 & -1 & 3 \\ -4 & 5 & 0 \\ 4 & 2 & 18 \end{bmatrix}$

What about pivoting?

- LU may not exist at all
  - if there is a zero pivot demanding a row interchange
  - can factorize as $A = P^{-1}LU = P^{T}LU$
  - P records the effects of row permutations
  - so PA = LU
- in-situ book-keeping is still feasible
  - but becomes more complicated
  - in principle .... a row-permuted version of A is factorized to LU, imagining that the required row interchanges are known in advance ☺
  - in practice .... you have to keep track of the permutations in P as they are done ☹

LU with pivoting: example

Find an LU factorization [by hand] with pivoting for $A = \begin{bmatrix} 1 & 2 & -3 \\ -3 & -4 & 13 \\ 2 & 1 & -5 \end{bmatrix}$

LU factorization in Matlab

- [Lp,U] = lu(A) returns
  - upper triangular U
  - permuted lower triangular Lp = $P^{-1}L$
- [L,U,P] = lu(A) returns
  - upper triangular U
  - lower triangular L
  - permutation matrix P so that...PA = LU

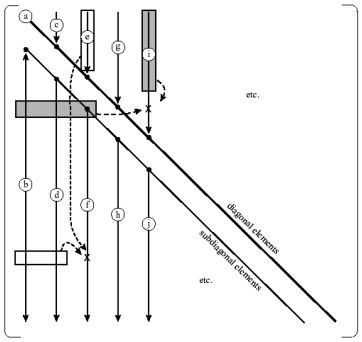
LU factorization in Matlab

- Matlab backslash \ can recognize a permuted triangular matrix and use appropriate(ly inexpensive) methods to solve the system
  - there is no need for explictly having P
  - you can write x = U\(L\b) and avoid the creation of a permanent scratch vector
  - of course x = A\b will (if deemed the most efficient available method) also use LU factorization without explicitly giving the L and U matrices (somewhat blackbox?)

Crout's algorithm

- alternative method to find the L & U matrices
- write out A = LU with unknowns for the non-zero elements of L & U
- equate entries in the n×n matrix equation
  - gives n^2 equations in $n^2 + n$ unknowns
- underdetermined .... so n unknowns are arbitrary
  - choose say the n diagonal entries on L to be 1
  - shows that the LU decomposition is not unique
- Crout's (clever) algorithm
  - re-write the n^2 equations in a carefully chosen order so that....
  - elements of L and U can be found one-by-one very simply
  - no more difficult than the process of back-substitution

Crout's algorithm



- ➲ L and U can be built up in the storage location used for A
- ➲ algorithm is not stable without pivoting, but that can be handled as for Gaussian elimination
- ➲ also called *Doolittle's method*
- ➲ a special case gives a highly valuable method .....

Positive definite matrices

- a matrix A is *positive definite* if $v^TAv > 0$ for all vectors $v \neq 0$
  - $\langle v|w \rangle$ defined by v^TAw is a valid inner product if and only if A is pos. def.
  - the inner product is *induced* by the matrix A
- a matrix is positive definite if and only if all its eigenvalues are positive
- a pos. def. matrix A has
  - all positive entries on the main diagonal [to show: apply $v^TAv > 0$ with the vectors (1,0,...,0), (0,1,0,...,0) etc.]
  - the largest entry (in abs val.) on the main diagonal
  - det (A) > 0 so it is always invertible
  - a unique square root matrix B so that $B^2 = A$

Diagonally dominant matrices

- A is *diagonally dominant* if:

  $|a_{ii}| > \sum|a_{ij}|,\ i \neq j,\ i = 1, \ldots n$
- a diagonally dominant matrix is positive definite if it is:
  - symmetric and ....
  - has all main diagonal entries positive
- ...but the converse is false
  - there are pos. def. matrices that are not diagonally dominant [find one - see slide 55]
  - there are pos. def. matrices that are diagonally dominant and not symmetric [any one with all positive eigenvalues]

Symmetric positive definite matrices

- there are many applications of symm. pos. def. matrices:
  - solution of partial differential equations ... heat conduction, mass diffusion etc (Poisson and Laplace equations)
  - analysis of stress
  - linear regression models
  - optimization problems
- symmetric pos. def. linear systems
  - are not esoteric
  - are not unusual
  - have a particularly efficient method for solution....

Cholesky LU decomposition

- the *Cholesky LU factorization* of a symmetric pos. def. matrix A is:
  - $A = L L^T$ (more common) or equivalently...
  - $A = U^T U$ (as done in Matlab)
- use it to solve a symmetric pos. def system Ax=b
- how to get L (or U)?
  - write out the factorization and solve for the values [special case of Crout's method]
  - only $(n^2+n)/2$ equations and unknowns
  - the positive definiteness of A guarantees the solution can be obtained (no bad square roots)
  - see *cholesky.m* for an implementation

Cholesky factorization: example

Find the Cholesky factorization of $A = \begin{bmatrix} 7 & 3 & 1 \\ 3 & 4 & 0 \\ 1 & 0 & 2 \end{bmatrix}$

Cholesky: numerical comments

- Cholesky is a stable algorithm without pivoting
- factor of two faster than the alternatives
- improved storage requirements
  - U and L use the same values
  - these can be stored in A
- the *chol* function in Matlab checks the form of A first and returns an error if it isn't symmetric pos. def.
  - write U = chol(A)
- backslash operator \ will use Cholesky preferentially if appropriate for the matrix

Cholesky: Matlab example

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 6 & 10 & 15 \\ 1 & 4 & 10 & 20 & 35 \\ 1 & 5 & 15 & 35 & 70 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 3 & 6 \\ 0 & 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- A is symm. pos. def. but NOT diagonally dominant
- A = pascal(5);
- U=chol(A); then take ...
- A(5,5) = 69 to destroy pos. definiteness
- see if it still works

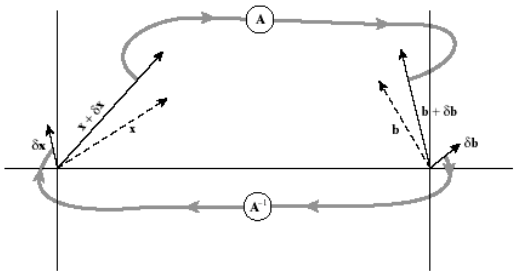
Iterative improvement

- floating point arithmetic limits precision possible
- for large systems or ill-conditioned small systems precision is generally far worse than *eps*
  - direct methods accumulate roundoff errors
  - these are magnified according to the degree of ill-conditioning
  - loss of 2-3 significant digits isn't unusual even for well-behaved systems
- iterative improvement will get your solution back to machine precision efficiently and effectively

Iterative improvement

- suppose
  - x is the (unknown) exact solution of Ax = b
  - x+δx is a calculated (inexact) solution with unknown error δx
- substitute in original equation:
  A(x+δx) = b+δb [...eqn 1]
- and subtract:
  A δx = δb [...eqn 2]
- eqn [1] gives:
  δb = A(x+δx) - b [...eqn 3]
- both terms on the rhs of [3] are known, so we can:
  - use [3] to get δb
  - and use this in [2] to solve for δx

Iterative improvement



Iterative improvement

- LU factorization of the original system Ax = b can be used to solve [2]:
  A δx = LU δx = δb to get δx
- then subtract δx from x+δx to get an improved solution
- repeat this method as necessary till δx ~ *eps*

Iterative methods: Jacobi

- write A = L + D + U:
  - D has the diagonal elements of A and...
  - L and U are zero-diagonal lower and upper triangular
- then Ax=b is (L+D+U)x =b [...eqn 1]
- so Dx = b - (L+U)x [...eqn 2]
- given x^i obtain x^{i+1} by solving [2] with $x = x^i$:

  $x^{i+1} = - D^{-1}(L+U)x^i + D^{-1}b$ [...eqn 3]
- J = $D^{-1}(L+U)$ is called the *iteration matrix*
  - calculating D^{-1} is trivial since D is diagonal

Jacobi: convergence

- re-arrange [1] for (L+U)x = b - Dx
- then $D^{-1}(L+U)x = D^{-1}b - x$ [...eqn 4]
- the (i+1)th error term is

  $x^{i+1} - x = [D^{-1}b - D^{-1}(L+U)x^i] - x$ subst. x^{i+1} from [3]

  $= - D^{-1}(L+U)x^i + [D^{-1}b - x]$ re-arranging

  $= - D^{-1}(L+U)x^i + [D^{-1}(L+U)x]$ subst. from [4]

  $= - D^{-1}(L+U)\,(x^i - x)$

  $= - J\,(x^i - x)$
- the error is expressed in terms of the iteration matrix J
  - the eigenvalues of J are a good measure of convergence rate
  - convergence fails if any eigenvalue of J has magnitude ≥ 1

Jacobi: example

Use Jacobi iteration to solve Ax=b with

$$A = \begin{bmatrix} 4 & 0 & -1 & -1 \\ 0 & 4 & -1 & -1 \\ 1 & 1 & -4 & 0 \\ 1 & 1 & 0 & -4 \end{bmatrix} \quad b = \begin{bmatrix} 4 \\ 4 \\ -4 \\ -4 \end{bmatrix}$$

Gauss-Seidel method

- an alternative iterative method to Jacobi
- as for Jacobi the previous x attempt is used in the original equations (x^i) to produce a better estimate for the solution (x^{i+1})
  - in Jacobi .... the complete vector of solutions is obtained before substituting to get the next iterate
  - in Gauss-Seidel .... each component of x^{i+1} is substituted as soon as it is obtained, before solving for the next component of x^{i+1}
- expressed in matrix form the difference in these two methods becomes transparent ....

Gauss-Seidel method

- Ax=b as (L+D+U)x =b [...eqn 1]
- then (L+D)x = b - Ux [...eqn 2]
- given x^i obtain x^{i+1} by solving [2] with $x = x^i$:

  $x^{i+1} = - (L+D)^{-1}U\,x^i + (L+D)^{-1}b$ [...eqn 3]
- G = $(L+D)^{-1}U$ is the *iteration matrix*
- the (i+1)th error term is $x^{i+1} - x = - G\,(x^i - x)$
- convergence fails if any eigenvalue of G has magnitude ≥ 1

Gauss-Seidel: example

Use G-S iteration to solve the system on slide 63. Calculate the eigenvalues of the iteration matrices and compare.

Jacobi's method: motivation

- equations may need to be re-arranged so they are diagonally dominant:

$6x_1 - 2x_2 + x_3 = 11$ $\qquad$ $6x_1 - 2x_2 + x_3 = 11$
$x_1 + 2x_2 - 5x_3 = -1$ becomes $-2x_1 + 7x_2 + 2x_3 = 5$
$-2x_1 + 7x_2 + 2x_3 = 5$ $\qquad$ $x_1 + 2x_2 - 5x_3 = -1$

- 'solve' for each variable in succession from each equation

$x_1 = 1.8333 + 0.3333x_2 - 0.1667x_3$
$x_2 = 0.7143 + 0.2857x_1 - 0.2857x_3$
$x_3 = 0.2000 + 0.2000x_1 + 0.4000x_2$

- this is the basis of the iteration equation to improve the ith vector

$x_1^{(i+1)} = 1.8333 + 0.3333x_2^{(i)} - 0.1667x_3^{(i)}$
$x_2^{(i+1)} = 0.7143 + 0.2857x_1^{(i)} - 0.2857x_3^{(i)}$
$x_3^{(i+1)} = 0.2000 + 0.2000x_1^{(i)} + 0.4000x_2^{(i)}$

Gauss-Seidel method: motivation

- all x_1, x_2 and x_3 are improved before the new values substituted together in the iteration equation to generate the new iterate
- this approach might be useful for parallel processing, but convergence is improved anyway by using the improved x values as soon as they are available
- the iteration equations look like this now:

$x_1^{(i+1)} = 1.8333 + 0.3333x_2^{(i)} - 0.1667x_3^{(i)}$
$x_2^{(i+1)} = 0.7143 + 0.2857x_1^{(i+1)} - 0.2857x_3^{(i)}$
$x_3^{(i+1)} = 0.2000 + 0.2000x_1^{(i+1)} + 0.4000x_2^{(i+1)}$

- compare these versions to the matrix equations and you'll see the motivation
- with diagonal dominance both methods will converge
- without diagonal dominance one, or both, of them may diverge
- if both methods converge then G-S will converge more quickly than J

Calculating the error...revisited

- the error on the (i+1) iteration is $\varepsilon_{n+1} = -G\,\varepsilon_n$
  - where G is the iteration matrix
- $\varepsilon_{n+1} = -G\,\varepsilon_n = -G(-G\,\varepsilon_{n-1}) = G^2\,\varepsilon_{n-1} = \ldots = (-G)^{n+1}\,\varepsilon_0$
- so if $G^n \to 0$ (zero matrix) then $\varepsilon_n \to 0$
- the key to understanding this condition is the eigenvalue decomposition of G:

$$G = UDU^{-1}$$

  - the columns of U consist of eigenvectors of G and...
  - D is a diagonal matrix of eigenvalues of G
- then $G^n = UD^nU^{-1}$
- if all the eigenvalues of G have magnitude < 1 then $D^n \to 0$ and consequently $G^n \to 0$

Digression: eigenvalues and eigenvectors

- suppose T: V $\to$V is a linear operator
- a vector v$\in$V for which T(v) = λv is called an *eigenvector* of T with *eigenvalue* [scalar] λ
- if T is defined by multiplication with a square matrix A we have Av = λv
- an n×n matrix has at most n distinct eigenvalues
- **eigenvectors corresponding to distinct eigenvalues are linearly independent**
- if λ is an eigenvalue of an invertible matrix A then and 1/ λ is an eigenvalue of A^{-1}

Digression: eigenvalues and eigenvectors

- v = 0 is obviously a possible solution of $[A - \lambda I]v = 0$ but not very interesting
  - the zero vector is technically an eigenvector of any matrix since A0 = λ0 for any λ
- what about non-zero solutions?
- a non-zero solution of $[A - \lambda I]v = 0$ exists if and only if the matrix A - λI is not invertible
  - otherwise we could invert A - λI and get the unique solution $v = [A - \lambda I]^{-1}0 = 0$, i.e. only the zero solution
- equivalently we have non-zero eigenvectors if and only if the rank of A - λI < n .....or
- equivalently we want: **det(A - λI) = 0**

Digression: eigenvalues and eigenvectors

- det(A - λI) = 0 is the *characteristic equation* of A
  - it's a polynomial of degree n if A is n×n
  - its solutions give all the eigenvalues λ
- the *algebraic multiplicity* of λ_i is the number of times the eigenvalue λ_i is repeated as a root of the characteristic equation
  - so $(\lambda - \lambda_j)^k$ is a repeated factor k times
- once we know all the λ_1, λ_2, λ_3,..... [solve for them] we take each one in turn and find the corresponding eigenvector(s) v by solving the linear system

$$[A - \lambda_i I]v = 0$$

Digression: eigenspaces

- if v and w are eigenvectors then so is any linear combination kv+w with the same eigenvalue:
  $A(kv) = k(Av) = k(\lambda v) = \lambda(kv)$
  $A(v+w) = Av+Aw = \lambda v+\lambda w = \lambda(v+w)$
- so for each eigenvalue λ the corresponding eigenvectors span a subspace E_λ, called the *eigenspace* of the eigenvalue λ
- **a complete solution consists of finding a basis of eigenvectors for each eigenspace (e.val.)**
- the *geometric multiplicity* of the eigenvalue λ is the dimension of its eigenspace
- the geometric multiplicity of an eigenvalue never exceeds its algebraic multiplicity

Digression: diagonalization

- not all linear operators can be represented by diagonal matrices with respect to some basis
- a square matrix A for which there is some [invertible] P so that $P^{-1}AP = D$ is a diagonal matrix is called *diagonalizable*
- if P is also orthogonal ($PP^T = I$) then A is *orthogonally diagonalizable*
- you should know:
  - which matrices can be diagonalized...
  - how to find the appropriate P and diagonal D
  - how to find an orthogonal P if it's possible to do so

Digression: diagonalization

- P consists of linearly independent eigenvectors of A arranged as columns
- diagonal entries of D are the corresponding e.vals.
- a square matrix A is orthogonally diagonalizable if and only if it is symmetric
  - to find an orthogonal P that diagonalizes a symmetric A ....
  - find a set of orthonormal (orthogonal unit vectors) e. vecs for each e. val.
  - orthogonality is automatic for e.vecs. corresponding to distinct e.vals. (not repeated)
  - otherwise you have to construct an orthogonal set of e.vecs. for each repeated e.val.

Eigenvalue decomposition re-visited

- the eigenvalue decomposition (EVD) for a square matrix A gives AU = UD
- Matlab example
  - A = [0 -6 -1; 6 2 -16; -5 20 -10]
- some matrices are not diagonalizable
  - A = [6 12 19; -9 -20 -33; 4 9 15]
  - this has a repeated degenerate eigenvalue 1 which has only one linearly independent eigenvector
- what about rectangular matrices??

Singular value decomposition

- A is rectangular (m×n, m>n)
- a *singular value* σ and corresponding pair of *singular vectors* u (m×1) and v (n×1) are related by:

  $\mathbf{Av = \sigma u}$ and $\mathbf{A^Tu = \sigma v}$
- arrange:
  - the singular values on the diagonal of a matrix S and
  - the corresponding singular vectors as the columns of two orthogonal matrices U and V
- then we have AV = US and $A^TU = SV$ ....

Singular value decomposition

- the orthogonality of U & V implies .....

  $\mathbf{A = USV^T}$
- this is the *singular value decomposition* (SVD) of A
  - U is m×m
  - S is m×n
  - V is n×n
  - the bottom m-n rows of S are all zero
- the *economy SVD* eliminates the zero rows of S
  - U is m×n
  - S is n×n
  - V is n×n

Singular value decomposition

- the EVD .....
  - for matrices representing mappings within a given v.s. (no dimension change)
- the SVD .....
  - analyses mappings between different v.s. with possibly different dimensions
- the existence of the SVD is a high point in linear algebra with 100 years history but....
  - it is relatively unknown in standard math teaching and
  - only recently begun to be used in numerical applications

SVD: matlab

- Matlab functions for the SVD:
  - svd(a) returns [U,S,V] as outputs
  - svd(a,0) is the economy SVD
- Matlab illustration with A = [9 4; 6 8; 2 7]

Calculating the SVD: get V

- combine the two conditions that define the u & v vectors:
  - $A^T(Av) = A^T(\sigma u) = \sigma(A^T u) = \sigma(\sigma v) = \sigma^2 v$
- so $A^TAv = \sigma^2 v$
- the singular values are the square roots of the eigenvalues of A^TA
- the columns of V (i.e. rows of V^T in the SVD) are the eigenvectors of A^TA
- we can always choose orthonormal e.vecs. as long as no e.val. is repeated
- .... what about U?

Calculating the SVD: get U

- define the jth column of U by $u_j = \sigma_j^{-1}Av_j$, where v_j is the jth column of V
  - A m×n, V n×n, so there are n of these m×1 columns making U an m×m matrix
- we have $AA^Tu_j = AA^T(1/\sigma_j)Av_j$
  $= (1/\sigma_j)A((A^TA)v_j)$
  $= (1/\sigma_j)A(\sigma_j^2 v_j)$ [v_j is an e.vec of A^TA]
  $= \sigma_j Av_j$
  $= \sigma_j^2 u_j$
- so the same singular values are also the square roots of the eigenvalues of AA^T and....
- the eigenvectors of AA^T are the columns of U

Calculating the SVD: preliminary result

- this gives us a preliminary SVD
  - the singular values (always real) are conventionally arranged in descending order on the main (upper) diagonal of S
  - U and V are real if A is real
  - U and V can easily be chosen to be orthogonal as long as AA^T (or equivalently A^TA) has no repeated e.val.
- example A = [2 4; 1 3; 0 0; 0 0]

Why are U & V orthogonal?

- A^TA is symmetric and positive definite
  - so is AA^T
  - it's actually non-negative definite since it can have zero eigenvalues as well as positive ones
- so ... the eigenvalues of A^TA are non-negative
  - can write them as $\sigma_1^2, \sigma_2^2, ..., \sigma_n^2$ where $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$

  AND....
- A^TA can be orthogonally diagonalized: $A^TA = VDV^T$
  - V is an n×n orthogonal matrix
  - the columns of V are an orthonormal basis of eigenvectors of A^TA
  - $D = \text{diag}(\sigma_1^2, \sigma_2^2, ..., \sigma_n^2)$

Calculating the SVD: U is not unique

- we know the columns of V are an orthonormal set of e.vecs. for A^TA (as we defined them to be)....
- and....the other eigenvector condition related to the columns of U being an orthonormal set of e.vecs. for AA^T is necessary
- however....this e.vec. condition is not sufficient to define U uniquely to give the SVD (hmm...)
- because... even for non-repeated e.vals. a unit e.vec. is determined uniquely except for the choice of direction

Calculating the SVD: U&V are not unique

- if u_j is an e.vec. then so is $-u_j$
- so there is room for manouver with respect to the signs chosen for V and U
- once you've decided on the vectors that form the columns of V your choice for the e.vecs. that form the columns of U is restricted
- you have to pick the correct direction for the u_j so that $A = USV^T$ is guaranteed
- as we found above this requires $\mathbf{u_j = \sigma_j^{-1} A v_j}$

Calculating the SVD: example

$$A = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Step 1. Calculate A^TA and find the e.vals:

$\sigma_1^2 = 29.8661$ & $\sigma_2^2 = 0.1339$

Calculating the SVD: example $A = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

Step 2. Find the corresponding e.vecs. of A^TA:

$v_1 = [0.4046, 0.9145]^T$ & $v_2 = [-0.9145, 0.4046]^T$

- e.vecs. from distinct e.vals will automatically be orthogonal
- you have to choose *unit* e.vecs. (two possibilities for each)
- v_1 & v_2 are unique up to a free choice of +/- direction
- the decision re:signs will be reflected in the signs of the U vecs. found next

Calculating the SVD: example $A = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$

Step 3. Calculate the columns of U:

$u_1 = \sigma_1^{-1} A v_1 = (1/5.4650)\ A\ [0.4046, 0.9145]^T$

$= [0.8174, 0.5760]^T$

$u_2 = \sigma_2^{-1} A v_2 = (1/0.3660)\ A\ [-0.9145, 0.4046]^T$

$= [-0.5760, 0.8174]^T$

Calculating the SVD: example

$$A = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = USV^T = \begin{bmatrix} 0.8174 & -0.5760 & 0 & \\ 0.5760 & 0.8174 & 0 & \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5.4650 & 0 \\ 0 & 0.3660 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.4046 & 0.9145 \\ -0.9145 & 0.4046 \end{bmatrix}$$

- compare this solution to the one obtained from the Matlab svd(a) function
- note that here a change in sign of v_1 is reflected in the sign change of u_1
- the SVD is therefore not unique with respect to sign changes of this kind
- note that u_1 and u_2 are e.vecs. of AA^T (as expected) but the direction is chosen to agree with the initial decision for directions of v_1 and v_2

Calculating the SVD: repeated eigenvalues

- suppose σ_j^2 is an e.val. of A^TA with multiplicity k > 1
- the corresponding eigenspace also has dimension k > 1
  - this is guaranteed because A^TA is pos. def.
- first select an orthonormal basis $\{v_1, ...v_k\}$ of e.vecs. of A^TA for the columns of V corresponding to this e.val
  - now we have an infinite number of possible choices for these
  - let's pick just one basis and stick with it to define a unique orthogonal matrix V
- now what about the U matrix?

Calculating the SVD: repeated eigenvalues

- we need $\{u_1, ...u_k\}$ to be an orthonormal basis for the eigenspace of AA^T corresponding to the e.val. σ_j^2
  - but there are an infinite number of possible choices for the columns of U that satisfy this condition
  - so this necessary condition isn't very helpful in defining U even when multiplicity k=2
- we must use the further restriction that relates U and V so the SVD works: $u_j = \sigma_j^{-1}Av_j$
  - this is applied to each of the $\{v_1, ...v_k\}$ to get the corresponding columns $\{u_1, ...u_k\}$
  - the resulting U matrix will be unique

Calculating the SVD: example

$$A = \begin{bmatrix} 5 & 2 \\ -2 & 5 \end{bmatrix}$$

Step 1. Calculate $A^TA = 29\ I$ and find the double e.val $\sigma_1^2 = 29$.

Calculating the SVD: example $A = \begin{bmatrix} 5 & 2 \\ -2 & 5 \end{bmatrix}$

Step 2. Find an orthonormal basis of e.vecs for σ_1^2.

We might as well take it easy and pick say:

$v_1 = [1\ 0]^T$ & $v_2 = [0,1]^T$

- the e.space is 2-dimensional, i.e. the whole of R^2 so....
- ANY two orthogonal unit vectors in R^2 will work for v_1 and v_2
- the required choices for the U vecs will reflect whatever is decided to use for the columns of V

Calculating the SVD: example $A = \begin{bmatrix} 5 & 2 \\ -2 & 5 \end{bmatrix}$

Step 3. Calculate the columns of U:

$u_1 = \sigma_1^{-1}Av_1 = (1/5.3852)\ A\ [1,0]^T$

$= [0.9285,-0.3714]^T$

$u_2 = \sigma_1^{-1}Av_2 = (1/5.3852)\ A\ [0,1]^T$

$= [0.3714,0.9285]^T$

SVD with repeated eigenvalues: example

$$A = \begin{bmatrix} 5 & 2 \\ -2 & 5 \end{bmatrix} = USV^T = \begin{bmatrix} 0.9285 & 0.3714 \\ -0.3714 & 0.9285 \end{bmatrix} \begin{bmatrix} 5.3852 & 0 \\ 0 & 5.3852 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

- compare this solution to the one obtained from the matlab svd(a) function
- there are sign differences again that reflect the initial choice of V columns
- *any* other choice of two orthogonal unit vectors for the columns of V would have worked, with suitable changes to U
- with e.vals. of multiplicity >1 the SVD is not unique up to an infinite number of variations
- u_1 and u_2 will always be e.vecs. of AA^T but.....that condition is useless in problems with repeated e.vals.

Why the SVD?

- the number of non-zero singular values = rank(A)
- A is singular if and only if it has at least one zero singular value
- in floating-point arithmetic if the size of the smallest singular value $\sigma_n \ll \sigma_1$ then the matrix is close to singular
  - the L_2-norm condition number for a square matrix is the ratio σ_1 / σ_n of the max and min singular values
  - this measure can be extended to rectangular matrices as a measure of conditioning

The SVD ....

- reveals a lot about the properties of A, especially its numerical qualities
- can provide a solution where other methods fail due to singularity or conditioning problems
- is valuable, powerful, and efficient for solving both under-determined and over-determined systems
- applies universally to all matrices regardless of size and regardless of rank
  - the EVD only applies to square matrices with full-rank eigenspaces

SVD: application to data compression

- an alternative way of writing the SVD:

$$\begin{aligned} A &= USV^T \\ &= \left[\ u_1 \ \middle|\ \cdots \ \middle|\ u_n \ \right] \mathrm{diag}(\sigma_1, \ldots, \sigma_n) \left[\ v_1 \ \middle|\ \cdots \ \middle|\ v_n \ \right]^T \\ &= \left[\ u_1 \ \middle|\ \cdots \ \middle|\ u_n \ \right] \left[\ \sigma_1 v_1^T, \ldots, \sigma_n v_n^T \ \right]^T \\ &= \sigma_1 u_1 v_1^T + \cdots + \sigma_n u_n v_n^T \\ &= \sigma_1 u_1 v_1^T + \cdots + \sigma_r u_r v_r^T \end{aligned}$$

- we can drop the terms above r =rank(A) ≤ n
- data storage requirements for A can be significantly reduced in this way

SVD application: data compression

- further reduction is possible by discarding the small terms corresponding to the small singular values
  - these often represent noise
- this gives important applications in image processing, digital data compression etc.
- for example a 500×337 image (168K pixels)

  → 337×337 SVD (114K pixels)

  → .....

  → 50×50 compressed SVD (2.5K pixels)

SVD application: data compression

original image
337×500 pixels
337 terms

70 term SVD

50 term SVD

SVD application: data compression

30 term SVD

10 term SVD

1 term SVD

How to use the SVD: zeroing

- an ill-conditioned system Ax = b may have a direct solution by LU or Gauss, but this may be only a poor approximation of the exact solution x
- *zero* the 'small' singular values in the SVD and proceed
  - i.e. give them exactly zero values
- the residual |Ax - b| may be better than that for both
  - the direct solution method and
  - the SVD without zeroing

How to use the SVD: zeroing

- zeroing is equivalent to discarding one linear combination of equations
- for small singular values the equation is so corrupted by roundoff error that it is
  - useless
  - tends to pull the solution to infinity in a direction almost parallel to a nullspace vector (i.e. one for which Ax = 0)
- what's the threshold value for determining when to zero a singular values in the SVD?
- this depends on:
  - the problem (conditioning)
  - the hardware
  - the desired residual
  - etc etc (the art of numerical methods.....)

SVD: solution of under-determined systems

- an under-determined system (m<n) has an (n-m)-dimensional solution space (in general)
- the SVD will have n-m singular values with zero or negligible size that can be zeroed
  - there may also be others if there are degeneracies in the n-m equations
- after zeroing we can apply an algorithm to find the particular solution
- the columns of V are a basis for the null-space, so linear combinations of these added to the particular solution provides the solution space of the original problem

SVD: solution of over-determined systems

- an over-determined system requires a least-squares fit to find the best fit solution
- the SVD is a valuable method to solve the least-squares problem
  - there may still be some degeneracies (close to zero s.v.'s)
  - the associated columns of V correspond to x values that are insensitive to the data
  - we can zero the s.v.'s to reduce the number of free parameters in the fit
- this topic is explored in Unit III....