P4480 / 7680 A7690 Sign up

Blackboard

S/U only

Prereq: ~2 sem math

Write programs in any computer language. ↓ C++ Python. (will say more later)

Text NR (3rd ed best)

Online access: library — search for numerical recipes, select and "online" or (better) numerical.recipes/corporate then click on "try"

Homework (Do! Audit not much use)

Topics: See General Info.

(2)

Computer Stuff

In this course, as a practical matter, you should program in one of the following languages/environments

One aim: help you with your research outside this class.

C++ skim NR §1.3 – 1.5.
Python
No! [Matlab
Mathematica] blackboxes – downside.
(Fortran)
Not: Java, Ruby, R, ...

If you've never programmed before, Python or ~~MatLab~~ probably easiest to learn. Otherwise: C++

~~For~~ Computational packages / Download the code!

C++: NR www.gnu.org/software/gsl

www.scipy.org/getting-started.html ← Python: scipy, numpy, matplotlib

matplotlib.org/gallery.html

Matlab: already included.

Plotting: Use Python, Matlab, Mathematica, xmgrace, gnuplot
C++: output text file, then plot in / use xmgrace.

Do HW #0

V2. or 3?? → prob. ok, unless legacy dependency on old V2 library

Installing python: download from www.python.org.
Also install scipy + numpy

OS

www.ubuntu.com/desktop

Do not use Windows. dual boot

Use a Unix-like OS: { Linux eg. ubuntu or fedora
Unix: Mac (use terminal window) }

Linux Command line: use a tutorial
eg. codeacademy or just google.

Intro to Course

Course in numerical methods. "How-to" — No proofs.

Read "~~Ch~~ §1.0 (i.e. skim).
§1.1
(For C++ programmers, read rest of Ch1)

Programming Hints

Structured programming:

Natural organization of human thought is hierarchical:
languages
music etc.

Computers are (were) sequential
Until recently computer languages reflected this.
More modern languages build in structures at higher levels.

→ Large gain in verifiability.
You give up freedom to invent your own (lousy) structures (cf. sonnets string quartets)

Ex Bisection on sorted array A(J), J=1,...N to find AA

Fortran 77

```
      KLO = 1
      KHI = N
3     K = (KHI+KLO)/2
      IF(A(K).LE.AA) GO TO 1
      KHI = K
      GO TO 2
1     KLO = K
2     IF(KHI-KLO.GT.1) GO TO 3
```

(i.e. AA is to the right of K)

C/C++

```
klo = 1;
khi = n;
while (khi-klo>1) {
   k = (khi+klo)/2;
   if (a(k)>aa) {
      khi = k;
   }
   else {klo = k;
   }
}
```

"Avoid statement labels + go to's".

Test at end of loop:

```
do {
   ...
   ...
} while ( );
```

Test in middle of loop:

```
for ( ; ; ) {
   ...
   if ( ) break;
   ...
}
```

Also: Document code!

<u>Object-oriented programming (OOP)</u>

The only design for large software projects.
Good even for small ones!

Start by choosing which objects to represent
e.g. ecology simulation, fish, ...

fish — environment — display

salmon, bass — lake, bay

In C++, object represented by a class (or struct

(5)

<u>object</u> <u>class</u>

state — data (variables)
behavior — functions (methods) eg accessors, mutators

Some advantages: 1) code reusability (inheritance)
2) data hiding (public vs private)
each class → well-defined interface for other programmers
"black boxes"

eg. Spec code

Makes it easy eg. to avoid global variables.

~~For C++~~: To use NR: make sure you can use
NR vector → VecDoub
NR matrix → MatDoub
(templated)

See §1.4

3 Key ideas in numerical algorithms:

1) Roundoff error
2) Truncation error
3) Numerical stability

Roundoff error

Due to representing floating-pt. numbers with finite number of significant digits

Worst case: loss of precision during subtraction.

eg. (fl.-pt.)

$a = 1$
$b = 1$
$c = \pi / 10^{14}$

$d = 10^{14} (a + c - b)$

will find $d = 3.1 \ldots\ldots$
↑ junk

Reason: machine precision is $\sim 10^{-16}$ on modern machines for double precision.
(float $\sim 10^{-8}$ when memory was expensive)

better: $d = 10^{14} \left((a-b) + c \right)$
(but some compilers feel free to reorder! Would have to do e.g. temp = a-b)

Often don't know ahead of time that a & b may be comparable in size.

More subtle example:

NR §5.6

Solving a quadratic: $ax^2 + bx + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Suppose $|b| \gg |a|$ or $|c|$

Then cancellation for 1 of the roots.

How to deal with?

$$X = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}$$

Good alg. - don't have to know in advance if anything will cancel.

So set $q = -\frac{1}{2}\left[b + \mathrm{sgn}(b)\sqrt{b^2 - 4ac}\right]$

Then $x_1 = \frac{q}{a}$, $x_2 = \frac{c}{q}$.

Truncation error

Due to numerical scheme itself - under your control (in principle!)

e.g. $\frac{df}{dx} \simeq \frac{f(x+h) - f(x)}{h}$

Can change size of h or use a ~~diff~~ higher order approximation.

Stability

We always try to use algorithms that are numerically stable. An unstable algorithm amplifies small roundoff or truncation errors so they become so large that error is unacceptable. Will discuss many examples of numerical instability.

Read §2.0 – 2.6 (can skip small print)
Optional: Read rest of Ch 2.

Linear Algebraic Eqns

All arrays zero-based

$$a_{00}x_0 + a_{01}x_1 + \dots + a_{0,N-1}x_{N-1} = b_0$$
$$a_{10}x_0 + \dots + a_{1,N-1}x_{N-1} = b_1$$
$$\vdots$$
$$a_{M-1,0}x_0 + \dots + a_{M-1,N-1}x_{N-1} = b_{N-1}$$

$$\iff \sum_{j=0}^{N-1} a_{ij}x_j = b_i \iff \underset{\approx}{A}\cdot\underset{\sim}{x} = \underset{\sim}{b}$$

(M eqns for N unknowns).

Cases: 1) If $M<N$, or if $M=N$ but $\underset{\approx}{A}$ singular, then $\underset{\sim}{x}$ is not unique. SVD is best algorithm.

2) If $M>N$, no soln. Least squares soln: $\min \| \underset{\approx}{A}\cdot\underset{\sim}{x} - \underset{\sim}{b} \| \to$ linear least squares
norm = sum of squares
SVD can be used for this too

Will consider first case $M=N$, $\underset{\approx}{A}$ nonsingular.

Will Consider several possible tasks:

1) Solve for $\underset{\sim}{x}$
2) solve for $\underset{\sim}{x}$'s for several $\underset{\sim}{b}$'s, same $\underset{\approx}{A}$
3) Find $\underset{\approx}{A}^{-1}$ ($\iff$ (2) with $\underset{\approx}{B} = (b_0\ b_1 \dots b_{N-1})$
   $b_i = \begin{pmatrix}0\\ \vdots \\ 1 \\ \vdots \\ 0\end{pmatrix} \leftarrow$ in row $i-1$)
4) Find det A

Numerical Challenges

1) Eqns close to lin. dep. (i.e. A close to sing.) Roundoff may make them lin. dep – alg. will fail (& you'll know)

2) If N large, roundoff errors can swamp true soln $\to$ wrong $\underset{\sim}{x}$ (& you'll not know even if you subst. in Ax again).

If A is close to sing, (2) tends to happen even if N is small.

In this case, may need a sophisticated method even for $N \approx 10$.

Algorithms

Very bad: Cramer's rule $(N!)$

Bad: $Ax = b \Rightarrow x = A^{-1} b$

Factor of 3 inefficiency in getting A^{-1}

Also unnecessary roundoff in $A^{-1} b$

Fair: Gauss-Jordan elimination:

e.g.
$$\begin{pmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 7 \\ 4 \\ 6 \end{pmatrix}$$

pivot

$\div 10$
$$\begin{pmatrix} 1 & -.7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} .7 \\ 4 \\ 6 \end{pmatrix}$$

row ops:
$$\begin{pmatrix} 1 & -.7 & 0 \\ 0 & -.1 & 6 \\ 0 & 2.5 & 5 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} .7 \\ 6.1 \\ 2.5 \end{pmatrix}$$

pivot

$$\begin{pmatrix} 1 & -.7 & 0 & .7 \\ 0 & 1 & -60 & -61 \\ 0 & 2.5 & 5 & 2.5 \end{pmatrix}$$

row ops:
$$\begin{pmatrix} 1 & 0 & -42 & -42 \\ 0 & 1 & -60 & -61 \\ 0 & 0 & \underset{\nearrow \text{pivot}}{155} & 155 \end{pmatrix}$$

pivot + row ops
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$x_0 = 0$
$x_1 = -1$
$x_2 = 1.$

Obviously, trouble if find a zero pivot.
Also, algorithm numerically bad (due to roundoff) if pivot nearly zero ← Example of numerical instability!

So: Interchange rows or: interchange rows + cols
↑ partial pivoting ↑ full pivoting (interchanges $x_i \leftrightarrow x_j$)
to put a desirable pivot on the diagonal

However, choice influenced by scaling. (e.g. multiply some row by 10^6 first).

<u>Implicit pivoting</u> — pick element that would have been largest if all eqns scaled, so max element in each is 1.

Not known what the "best" pivoting strategy is, but theorems ⇒ implicit partial pivoting is "good enough"

Continue with algorithms:

Gaussian elimination: Like G-J, but only reduce elements below the diag to zero:

$$\begin{matrix} 10 & -7 & 0 & 7 \\ -3 & 2 & 6 & 4 \\ 5 & -1 & 5 & 6 \end{matrix}$$

$$\begin{matrix} 10 & -7 & 0 & 7 \\ 0 & -.1 & 6 & 6.1 \\ 0 & 2.5 & 5 & 2.5 \end{matrix}$$

partial pivoting

$$\begin{matrix} 10 & -7 & 0 & 7 \\ 0 & 2.5 & 5 & 2.5 \\ 0 & -.1 & 6 & 6.1 \end{matrix}$$

$$\begin{matrix} 10 & -7 & 0 & 7 \\ 0 & 2.5 & 5 & 2.5 \\ 0 & 0 & 6.2 & 6.2 \end{matrix}$$

"Backsubstitution":

$$6.2\, x_2 = 6.2 \Rightarrow x_2 = 1$$

$$2.5 x_1 + 5(1) = 2.5 \Rightarrow x_1 = -1$$

$$10\, x_0 - 7(-1) = 7 \Rightarrow x_0 = 0$$

Faster than G-J by factor ~ 1.5.

Disadvantge: All rhs must be known in advance

Best: LU decomposition.

Suppose can write $A = LU$

Note that $\det A = \det L \det U = 1 \cdot \prod_{j=0}^{N-1} \beta_{jj}$

$$\begin{pmatrix} a_{00} & a_{01} & \cdots \\ a_{10} & a_{11} & \cdots \\ \vdots & & \ddots \end{pmatrix} = \begin{pmatrix} \alpha_{00} & & \\ \alpha_{10} & \alpha_{11} & 0 \\ \alpha_{20} & \alpha_{21} & \alpha_{22} \\ \cdots & & \end{pmatrix} \begin{pmatrix} \beta_{00} & \beta_{01} & \cdots \\ & \beta_{11} & \cdots \\ 0 & & \beta_{22} & \cdots \end{pmatrix}$$

Th: Can always do this and take diagonals of L to be 1

See NR §2.3 for eqns. Requires $\sim \frac{1}{3} N^3$ ops.

$Ax = b$
$\rightarrow LUx = b$
$\rightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$

Implemented in NR as LUdcmp

Code:

```
const Int n = ....
Mat_Doub a(n,n);
VecDoub b(n), x(n);

LUdcmp alu(a);          (alu arb. name)
alu.solve(b, x);

VecDoub bnew(n);        <- O(N^2).

alu.solve(bnew, x);
```

For inverse: ~~alu.inverse~~ `MatDoub ainv;`
`alu.inverse(ainv);` ← or use gaussj.

~~etc~~ Determinant: `alu.det`

§2.5 Iterative Improvement

Roundoff errors accumulate for large N

$$Ax = b \qquad (x \text{ means exact, not known})$$

$$A(x+\delta x) = b + \delta b \qquad x+\delta x \text{ known}, \ \delta x = \text{unknown error.}$$

Subtr: $A\,\delta x = \delta b$

$$= A(x+\delta x) - b \quad (*)$$

RHS can be computed i.t.o known stuff.
So can solve for δx in $O(N^2)$
Then $x_{\text{improved}} = x_{\text{old}} - \delta x$
($x_{\text{old}} = x + \delta x$)

Note: Best if compute RHS in higher precision e.g. long double.
But still get improvement if don't (despite what textbooks say)

Not much improvement beyond 1 iteration.

Package Routines

LAPACK is the champ. (But Fortran with C interface)

Condition Number

Need concept of matrix norm: Every vector norm induces a corresponding matrix norm: (l.u.b.)

Not important.

$$\|\underset{\approx}{A}\| = \sup_{\|\underset{\sim}{x}\|=1} \|\underset{\approx}{A}\cdot\underset{\sim}{x}\| = \sup_{\|\underset{\sim}{x}\|\neq 0} \frac{\|\underset{\approx}{A}\cdot\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|}$$

e.g. for max norm, $\|\underset{\sim}{x}\|_\infty = \max |x_i|$

$$\|\underset{\approx}{A}\|_\infty = \max_{(\text{rows})} \sum |\text{elements in row}|$$

Theorems indep of which norm is used.

Th: $\|A \cdot x\| \le \|A\| \, \|x\|$ (like C-S ineq)

let x be a computed soln, $x_* =$ true soln

Define **residual** $r = b - A \cdot x$ (how well x approximates b in $A \cdot x = b$)

error $e = x_* - x$

$= A^{-1} \cdot r$ since $A \cdot x_* = b$

$$\|e\| \le \|A^{-1}\| \, \|r\| \quad + \quad \|b\| \le \|A\| \, \|x_*\|$$

$$\therefore \frac{\|e\|}{\|A\| \, \|x_*\|} \le \frac{\|A^{-1}\| \, \|r\|}{\|b\|}$$

$\downarrow A \cdot x$
~~computed~~ b ~~b~~ computed

$$\Rightarrow \frac{\|e\|}{\|x_*\|} \le \|A\| \, \|A^{-1}\| \, \frac{\|r\|}{\|b\|}$$

Define $\mathrm{cond}(A) = \|A\| \, \|A^{-1}\|$ condition #

Lecture 5

Interpret $\le$ as $\sim$

Rel error in soln $\sim$ (cond. #) (rel. error in residual)

r is like computed rhs b

"amplification factor"

best one can do is roundoff i.e. $\sim 10^{-16}$

A sing $\Rightarrow \mathrm{cond}(A) \to \infty$.

So if cond. # is large, soln can be inaccurate.

might get lucky, but ~~usually not~~ hardly ever in practice!

LAPACK etc have cheap methods to estimate cond # $O(N^2)$.

In scipy there is cond(A)

In NR: use SVD: (will see later)
(expensive)

```
SVD svd(a);
cond_num = svd.w[0] / svd.w[n-1]
or cond_num = svd.inv_condition()
```

inverse in case = 0

Special Forms of A

1) Tridiagonal. $O(N)$ §2.4 pivoting usually not nec e.g. if diagonally domin

2) Band diagonal §2.4

3) Sparse 2.7
Solving Very large sparse systems: Iterative methods §2.7.6 – huge industry.

4) Symmetric +ve definite
$$A = L L^T \quad - \quad \tfrac{1}{6} N^3 \ (\text{factor of } 2)$$
very stable.

5): $A = Q R$
Q: orthog, $Q^T Q = 1$; R: upper Δ^r

Uses: constructing orth. basis (usually better than Gram Schmidt)
Least squares (although I prefer SVD – handles pathological cases)

SVD (Singular Value Decomposition).

One of the best things you will learn in this course!

Singular Value Decomposition (SVD)

Very powerful technique for dealing with singular or nearly singular matrices

∃ other statements of Theorem eg. $U = m\times m$ orthog, $W = [w_j\ 0]$ $m\times n$ or $\begin{bmatrix} w_j \\ 0 \end{bmatrix}$ $n\times n$ see handbook of linear algebra ref.

<u>Theorem</u> If A is $m\times n$ $(m\geq n)$, ∃ ~~orthogonal~~ U, V + W s.t.

$$A = U W V^T$$

where

$$\underbrace{\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix}}_{m \times n} = \underbrace{\begin{bmatrix} & & \\ & U & \\ & & \end{bmatrix}}_{\substack{\text{columns orthog.} \\ U^TU = 1}} \underbrace{\begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix}}_{\substack{\text{diagonal,} \\ w\text{'s} \geq 0}} \underbrace{\begin{bmatrix} & & \\ & V^T & \\ & & \end{bmatrix}}_{\substack{\text{orthogonal} \\ V^TV = VV^T = 1}}$$

Note: 1) If $m < n$, ~~can apply theorem to~~ A^T

2) SVD is unique a) up to permutation of cols of U, corr. elements of W, + corr. rows of V^T (i.e. cols of V),

or b) forming lin. combos of any cols of U + V whose corr. w's happen to be equal.

<u>Relation to eigenvalues</u> $A^TA = VWU^TUWV^T = V\,\mathrm{diag}(w_j^2)\,V^T$ i.e. eigenvalues of A^TA are squares of sing. values of A

(idea is to use find without forming A^TA explicitly).

see discussion in refs or in Golub + Van Loan "Matrix Computations"

Routine ~~SVDCMP~~ : Given A, returns U (in A), W (vector) + V

decomp in SVD.

<u>Square Matrices</u>

A, U, V, W all square.

$$A^{-1} = (UWV^T)^{-1}$$

$$= (V^T)^{-1}(W)^{-1}U^{-1}$$

$$= V\ \mathrm{diag}\left(\frac{1}{w_j}\right) U^T$$

If a w_j is zero, matrix is singular.

If " " close to zero, " " almost singular + with roundoff it may be singular for all practical purposes.

Condition number of $A = \frac{\max(w_j)}{\min(w_j)}$

"Practical" singularity: ↑ $\sim 10^{6}$ (single prec)

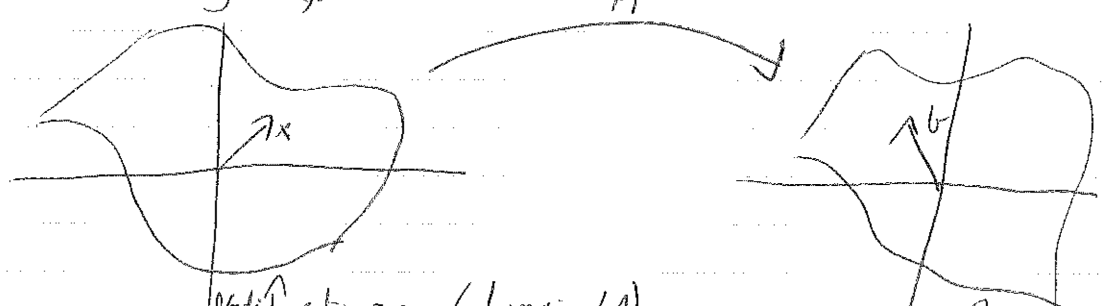
$\sim 10^{12}$ (double prec).

<u>Null space + Range</u>

Consider $Ax = b$

If A is nonsingular:

A



N-dim vector space (domain of A)

N-dim vector space (range of A) of same dimension

If A is singular: $\exists$ subspace of ( ) which is mapped to zero: $Ax = 0$

(i.e. homogeneous eqns have nontrivial soln, $\det A = 0$) called the <u>nullspace</u> of A

dimension of nullspace = <u>nullity</u> of A

$\exists$ subspace of ( ) which can be reached by A

i.e. $Ax = b$ for some x — range of A

dimension of range = <u>rank</u> of A

rank + nullity = N

~~SVD construction~~ If $A = U W V^T$ $\qquad AV = UW$

i.e. $Av_j = w_j u_j$

columns of U that corr. to <u>nonzero</u> w_j = orthonormal basis for range (~~since v_j's span R~~)

columns of V corr. to <u>zero</u> w_j = orthonormal basis for nullspace, since $Av_j = 0$

Soln of $Ax = b$ when A is singular

Case 1 If $Ax = b$ with $b \in \text{range}(A)$, then $\exists$ soln.
Soln not unique : can add any soln of homogeneous eqns $Ax = 0$ i.e. any member of nullspace
i.e. any lin. comb. of cols of V corr. to zero w_j.

Consider soln. with smallest $|x|^2$: Can find from SVD by replacing $\frac{1}{w_j} \to 0$ if $w_j = 0$.

Pseudo-inverse A^+ i.e.

$$x = \text{"}A^{-1}\text{"}\, b$$

$$= V \,\text{diag}\left(\frac{1}{w_j}\right) U^T b$$

omit Proof

General soln is $x + x'$
↑ any vector in null space

modified s.t. $\frac{1}{w_j} \to 0$

$$|x + x'| = |V W^{-1} U^T b + x'|$$

$V V^T$

$$= |V(W^{-1} U^T b + V^T x')|$$

$$= |V| \, |W^{-1} U^T b + V^T x'|$$

$|V| = 1$

$W^{-1} U^T b$: nonzero only where $w_j \neq 0$

$V^T x'$: nonzero only where $w_j = 0$ (since x' in null space i.e. lin comb of cols of V corr to zero w_j).

so varying x' doesn't affect contrib. of 1st term

$\therefore \min |x + x'|$ when $x' = 0$.

i.e. $x \to$ min soln.

Case 2 $Ax = b$, $b \notin \text{range}(A)$
No soln.

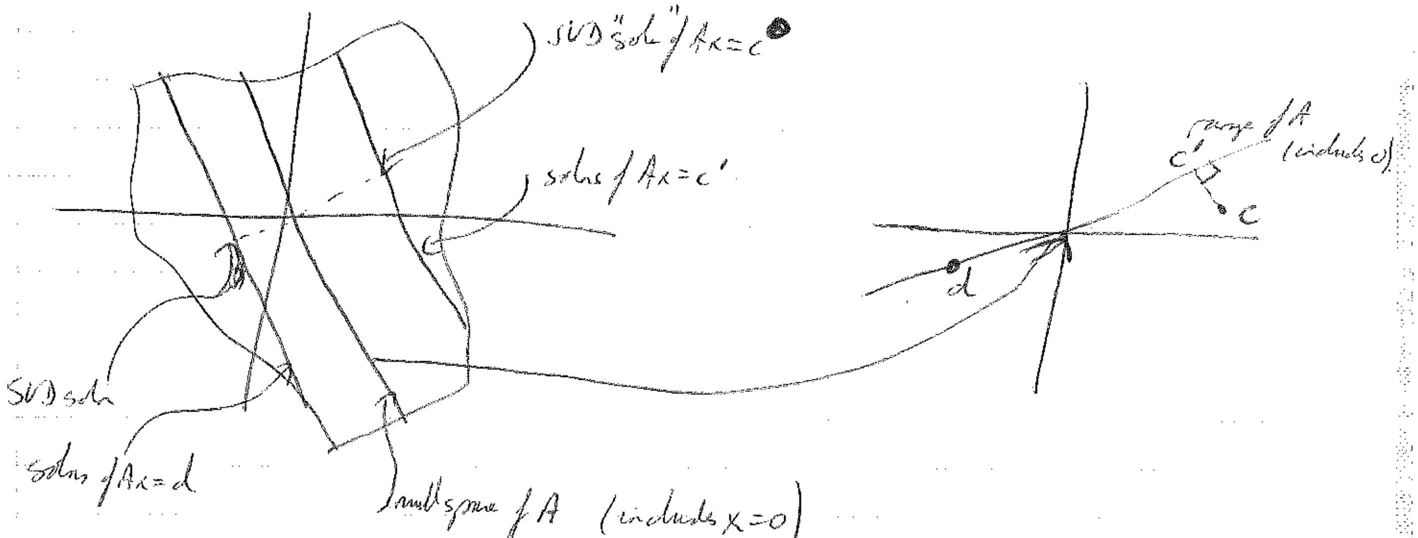
But above x is the least-squares soln i.e. defines

x s.t. $|Ax - b|$ is a minimum. $b' = Ax'$

Proof: Same ~~as also~~ idea as above : $x \to x + x'$ ab. Modifies $Ax - b$?

consider $|Ax - b + b'|$ & show $b' = 0$ for min ... $\Rightarrow$ $Ax - b + b'$ ...

represent lower dim. of range by: plane $\xrightarrow{A}$ line



Consider $Ax = d$, $d \in$ range(A)

solns = particular soln + any vector in nullspace

→ "line" ∥ nullspace

SVD soln is closest to origin ($|x|^2$ = min).

Consider $Ax = c$, $c \notin$ range(A)

SVD finds soln of $Ax = c'$

SVD soln of almost singular (badly conditioned) eqns

1) ~~call SVDCMP~~ MatDoub a; SVD ~~…~~ svd(a);
2) find max(w_j) e.g. $n\varepsilon$ or $\frac{1}{2}\sqrt{m+n+1}\,\varepsilon$ ←
3) set any w_j < ~~10^{-6}~~ max(w_j) to zero. ← if omit, uses
4) call ~~SVBKSB~~ svd.solve(b, x, thresh) (give U, W, V, B, returns X).

Will return X with smallest residual $|Ax-b|$

Much Better than straight LU or ~~SVDCMP / SVBKSB~~ svd without editing w_j's.

Note: SVD is about 5-10 times as "expensive" as LU.

SVD soln of homogeneous eqns

$$Av_j = w_j u_j \quad \text{(columns)}$$

∴ for each $w_j = 0$, v_j is a lin. indep. soln

or $w_j < \varepsilon\, \max(w_j)$.

SVD for fewer eqns than unknowns

M eqns, N unknowns, expect N−M dim. family/solns

1) A is M×N
~~add rows of zero until → N×N~~ ~~Also r.h.s. add zero~~ → N eqns, N unknowns

2) SVD will find N−M w_j's that are ≈ 0 ~~for each row added~~.

Omit thresh → default.

3) zero them explicitly then call SVBKSB solve → particular soln.

4) cols of V corr. to $w_j = 0$ → solns of homogeneous eqn that can be added. svd's nullspace(thresh)

SVD for more eqns than unknowns

e.g. linear least squares problems (data fitting)

as above for square case,

$$x = V \,\mathrm{diag}\left(\frac{1}{w_j}\right) U^T b$$ is least squares soln.

See §15.4

e.g. minimize
$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{M} a_k X_k(x_i)}{\sigma_i} \right]^2$$

(x_i, y_i) = N data pts, σ_i = weights

$y = \sum_{k=1}^{M} a_k X_k(x)$ is linear model with M unknown params (e.g. $X_k = x^k$ for polynomial fit)

let A be $N \times M$ matrix: (design matrix)
$$A_{ik} = \frac{X_k(x_i)}{\sigma_i} \qquad (N > M).$$

b is N-vector: $b_i = \dfrac{y_i}{\sigma_i}$

let a denote M-vector $a_1 \dots a_M$

Usually done by normal eqns (§14.3)
SVD: find a to minimize $\chi^2 = |Aa - b|^2$

Soln:
$$a = V \,\mathrm{diag}\left(\frac{1}{w_j}\right) U^T b = \sum_{i=1}^{M} \left(\frac{U_i \cdot b}{w_i}\right) V_i$$

U_i = M cols of length N
V_i = M cols ... M
can show = princ. axes of error ellipsoid of a_i's

~~Standard errors in fit: $a = ? \pm \frac{1}{w_1} V_1 \pm \frac{1}{w_2} V_2 + \dots$~~

Can show $\sigma^2(a_j) = \sum_{i=1}^{M} \frac{1}{w_i^2} (V_{ji})^2$

$$\text{cov}(a_j, a_k) = \sum_{i=1}^{M} \frac{1}{w_i^2} (V_{ji} V_{ki})$$

Key point: If any $w_i = 0$ (or $\simeq 0$), set $\frac{1}{w_i} \to 0$.

implies some lin. comb. of basis fns is degen. for this data.

See §15.6 Also, can sometimes set additional some $\frac{1}{w_i} \to 0$, if don't increase χ^2 too much

i.e. these params don't contribute much to improving fit (~~see §14.5~~)

Disadvantages of SVD: 1) Need storage for $N \times M$ matrix (normal eqns form $A^T A = M \times M$).

2) Slower than normal eqns.

Advantages: 1) "cannot" fail

2) less roundoff

3) easy to interpret lin indep sets of params.

Can also use SVD to construct an orthonormal basis (cf Gram-Schmidt §2.6.5) or to approximate matrices (see NR §2.6.6)

(LIGO templates)