$$E = \frac{f^{(2n)}(\xi)}{(2n)!} \int_a^b \left(\prod_{i=0}^{n-1} (x - x_i) \right)^2 w(x)\, dx,$$

where $\xi \in (a,b)$.

§ 4.5 Romberg Integration

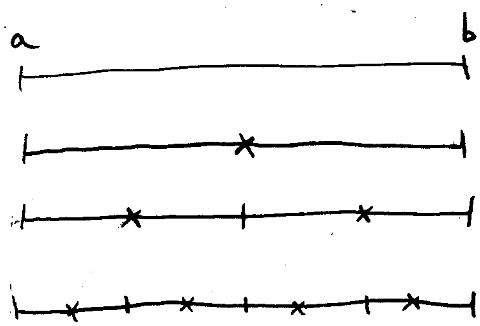
Idea: use Richardson's extrapolation to increase accuracy of composite trapezoidal rule.
→ get high accuracy method from multiple applications of a low accuracy method.

Recall composite trapezoidal rule:

$$\int_a^b f(x)\,dx = \frac{h}{2}\left(f(x_0) + f(x_n) + 2\sum_{j=1}^{n-1} f(x_j) \right) - \frac{(b-a)}{12} h^2 f''(\xi)$$

where $\xi \in (a,b)$, $h = \frac{b-a}{n}$ and $x_j = a + jh$.

Romberg integration needs to evaluate composite trapezoidal rule for $n = 2^k$, $k = 0, 1, \dots, M$ without superfluous function evaluations



$n = 1 = 2^0$
$n = 2 = 2^1$
$n = 4 = 2^2$
$n = 8 = 2^3$
⋮

x = new function evaluation
| = function evaluation available from previous "level".

Thus composite trapz rule becomes with $h_k = 2^{-k}(b-a)$:

$$\int_a^b f(x)\,dx = \frac{h_k}{2}\left[f(a) + f(b) + 2\sum_{i=1}^{2^k-1} f(a+ih_k)\right] - \frac{(b-a)}{12} h_k^2 f''(\xi_k)$$

$$= R_{k,0} + \text{error}$$

where $\xi_k \in (a,b)$.

So:

$$R_{00} = \frac{h_0}{2}(f(a)+f(b)) = \frac{b-a}{2}(f(a)+f(b))$$

$$R_{10} = \frac{h_1}{2}(f(a)+f(b)+2f(a+h_1))$$

$$= \frac{1}{2}R_{00} + h_1 f(a+h_1)$$

$$R_{20} = \frac{1}{2}R_{10} + h_2(f(a+h_2) + f(a+3h_2))$$

$$\vdots$$

$$R_{k0} = \frac{1}{2}R_{k-1,0} + h_k \sum_{j=1}^{2^{k-1}} f(a+(2j-1)h_k)$$

evaluation only at odd numbered nodes.

It can be shown that:

① $$\int_a^b f(x)\,dx = R_{k0} + K_1 h_k^2 + K_2 h_k^4 + K_3 h_k^6 + \cdots$$

② $$\int_a^b f(x)\,dx = R_{k+1,0} + K_1 h_{k+1}^2 + K_2 h_{k+1}^4 + K_3 h_{k+1}^6 + \cdots$$

$$= R_{k+1,0} + K_1 \frac{h_k^2}{4} + K_2 \frac{h_k^4}{16} + K_3 \frac{h_k^6}{64} + \cdots$$

Do Richardson's extrapolation trick to cancel out leading term in error:

$4 \times ② - ①$:

$$3\int_a^b f(x)\,dx = 4R_{k+1,0} - R_{k,0} - \frac{3}{4}h_k^4 - \frac{15}{16}h_k^6 - \cdots$$

$$\int_a^b f(x)\,dx = \underbrace{\frac{4R_{k+1,0} - R_{k,0}}{4-1}}_{\equiv R_{k+1,1} \,=\, \mathcal{O}(h_k^4)\text{ approx}} - \frac{1}{4}h_k^4 - \frac{5}{16}h_k^6 - \cdots$$

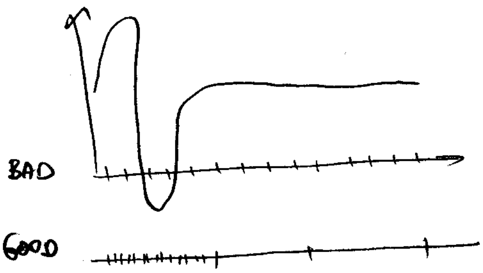
If we keep doing same thing:

$$R_{k+1,j} = \frac{4^j R_{k+1,j-1} - R_{k,j-1}}{4^j - 1} = \mathcal{O}(h_k^{2(j+1)})$$

Organizing work in a table we get: (for example)

$$\begin{array}{lllll}
R_{00} & & & & \\
R_{10} & \hookrightarrow R_{11} & & & \\
R_{20} & \hookrightarrow R_{21} & \hookrightarrow R_{22} & & \\
R_{30} & \hookrightarrow R_{31} & \hookrightarrow R_{32} & \hookrightarrow R_{3,3} & \\
R_{40} & \hookrightarrow R_{41} & \hookrightarrow R_{42} & \hookrightarrow R_{43} & R_{4,4} \\
\mathcal{O}(h_4^2) & \mathcal{O}(h_4^4) & \mathcal{O}(h_4^6) & \mathcal{O}(h_4^8) & \mathcal{O}(h_4^{10})
\end{array}$$

§4.6 Adaptive quadrature methods



Using a uniform partition of an interval for a quadrature rule for approximating the integral of a function is not the most efficient way of doing things!

Obviously we can do better if we adapt the # of points to put more points where they are needed the most $\Rightarrow$ principle behind adaptive methods.

key ingredient: we need a way that tells us how good our quadrature is working. We could do a number of things for example using more nodes to estimate error term or using a more accurate method. Here we choose to refine # of points to estimate error.

On some interval $[a,b]$:

$$(*) \quad \int_a^b f(x)\,dx = \underbrace{\frac{b-a}{6}\left(f(a) + f\left(\tfrac{a+b}{2}\right) + f(b)\right)}_{\equiv S(a,b)} - \frac{h^5}{90} f^{(4)}(\xi)$$

for some $\xi \in (a,b)$ where $h = \dfrac{b-a}{2}$

Of course we could estimate error if we knew $f^{(4)}$ but this is asking too much from end user!

Instead we apply Simpson's rule again: on two subintervals of $[a,b]$:

$$(**) \quad \int_a^b f(x)\,dx = S\left(a, \tfrac{a+b}{2}\right) + S\left(\tfrac{a+b}{2}, b\right) - \frac{h^5}{2^5 90} f^{(4)}(\xi_1) - \frac{h^5}{2^5 90} f^{(4)}(\xi_2)$$

$$= \quad '' \qquad\qquad '' \quad - \frac{h^5}{90}\left(\frac{1}{16}\right) f^{(4)}(\tilde{\xi})$$

where $\xi_1 \in \left(a, \frac{a+b}{2}\right)$, $\xi_2 \in \left(\frac{a+b}{2}, b\right)$ and $\tilde{\xi} \in (a,b)$.

Now assume $f^{(4)}$ does not vary too much on (a,b) so:

$$f^{(4)}(\xi) \approx f^{(4)}(\tilde{\xi})$$

So to estimate the error we simply do $(**)-(*)$ to get:

$$\frac{h^5}{90} f^{(4)}(\xi) \approx -\frac{16}{15}\left[S(a,\tfrac{a+b}{2}) + S(\tfrac{a+b}{2},b) - S(a,b)\right]$$

Plugging into $(**)$ and bounding integration error:

$$\left| \int_a^b f(x)dx - S(a,\tfrac{a+b}{2}) - S(\tfrac{a+b}{2},b)\right| \leqslant \frac{1}{15}\left| S(a,b) - S(a,\tfrac{a+b}{2}) - S(\tfrac{a+b}{2},b)\right|$$

So if we want $(**)$ to be accurate to within ε we need:

$$\left| S(a,b) - S(a,\tfrac{a+b}{2}) - S(\tfrac{a+b}{2},b)\right| < 15\varepsilon$$

To be conservative (and account for some changes in $f^{(4)}$) we can require 10ε bound.

<u>Algorithm</u>: the book implements a recursive procedure using a for loop. This is like writing a tree traversal algorithm with for loops: very complicated! ⇝ use <u>recursion</u>: i.e. a function that calls itself.

⚠ Using recursion is probably less efficient than running code in the book but it's much easier to understand!

⚠ Also some programming languages are limited in # of recursion levels that are allowed e.g. if you run a C program you may get "stack overflow" or other messages of the kind.

⚠ It may be good to impose a limit on # of recursions in "industrial" code as infinite recursions can be nastier than infinite loops!

function $r = asimpson(a, b, f, \varepsilon)$

$$S = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

$$S1 = \frac{b-a}{12}\left(f(a) + 4f\left(a+\frac{b-a}{4}\right) + f\left(\frac{a+b}{2}\right)\right)$$

$$S2 = \frac{b-a}{12}\left(f\left(\frac{a+b}{2}\right) + 4f\left(a+\frac{3(b-a)}{4}\right) + f(b)\right)$$

If $|S - S_1 - S_2| < \varepsilon$

$r = S1 + S2;$ % we are happy with approx

else

$r = asimpson\left(a, \frac{a+b}{2}, f, \varepsilon/2\right) + asimpson\left(\frac{a+b}{2}, b, f, \varepsilon/2\right)$

- recursive calls of adaptive Simpson code for left and right halfs of intervals. Requiring that error be $\varepsilon/2$ ensures the total error from left + right subint will not exceed ε when summed up.
- instead of accumulating only integrals we can also accumulate e.g. points of discretization.

LU and Cholesky factorizations

BLAS 1,2,3
LAPACK

Consider the linear system

$$Ax = b, \quad \text{where } A \in \mathbb{R}^{n \times n}, \ x \in \mathbb{R}^n, \ b \in \mathbb{R}^n.$$

Easiest system to solve is when $A = \mathrm{diag}(a_1, a_2, \dots a_n)$ i.e.

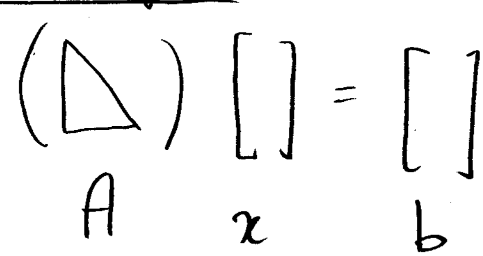
$$A = \begin{bmatrix} a_1 & & & & \\ & a_2 & & & \\ & & a_3 & & \\ & & & \ddots & \\ & & & & a_n \end{bmatrix}$$

then: (if the $a_i \neq 0$):

$$x = A^{-1}b = \begin{bmatrix} b_1/a_1 \\ b_2/a_2 \\ \vdots \\ b_n/a_n \end{bmatrix} = b./a \quad \text{in Matlab notation.}$$

Here is another system that is convenient:

<u>lower triangular</u>:



$$A = \begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & 0 \cdots & 0 \\ \vdots & & & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}$$

How do we solve such a system?

$$x_1 = b_1 / a_{11}$$
$$x_2 = (b_2 - a_{21} x_1) / a_{22}$$
$$x_3 = (b_3 - a_{31} x_1 - a_{32} x_2) / a_3$$
$$\vdots$$
$$x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j\right) / a_{ii} \qquad i \leq n$$

This is called <u>forward substitution</u> as we compute x_i "forward" (direction of increasing i's). Writing this as a for loop:

for $i = 1 .. n$
$$\left| \; x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j\right) / a_{ii}\right.$$

yet another way of writing same algorithm but using Matlab:

for $i = 1..n$
$\quad x_i = (b_i - A(i, 1:i-1)\, x(1:i-1)) / a_{ii}$
end.

In a similar way <u>upper triangular</u> (∇) systems are easy to solve. The algorithm is same but we walk our way backwards from x_n to x_1:

for $i = n : -1 : 1$ (<u>backward substitution</u>)
$$\left| \; x_i = b_i - \sum_{j=i+1}^{n} a_{ij} x_j\right.$$

Obviously if we permute rows of system we can still solve easily. For example: consider permutation

$p_1, p_2, \ldots, p_n$ (which we can put in a single vector p)

if the matrix $\left[a_{p_i, j}\right]_{i=1..n, j=1..n}$ is lower (or upper) triangular

then we can perform forward substitution for permuted systems:

for $i = 1 \ldots n$

$$x_i = \left(b_{p_i} - \sum_{j=1}^{i-1} a_{p_i j} x_j\right) / a_{p_i i}$$

n^2 flops

or in Matlab notation:

for $i = 1:n$,

$x(i) = (b_{p_i} - a(p(i), 1:i-1)\, x) / a_{p(i), i}$.

Similarly backwards substitution becomes:

for $i = n:-1:1$,

$$x_i = b_{p_i} - \sum_{j=i+1}^{n} a_{p_i j} x_j \,/\, a_{p_i i}.$$

LU decomposition:

$$A = L\,U$$

⚠ not all matrices admit LU decomp

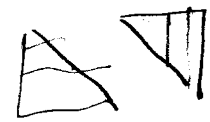
If we have LU decomposition available:

solving $Ax = b$

$\Leftrightarrow$ | Solving $Lz = b$ for z. } relatively easy to do

| Solving $Ux = z$ for x }

<u>Derivation of LU factorization:</u>



$$(LU)_{ij} = \sum_{k=1}^{n} l_{ik} u_{kj}$$

$$= \sum_{k=1}^{\min(i,j)} l_{ik} u_{kj}$$

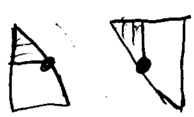
Fix either values l_{kk} or u_{kk}. say $u_{kk} = 1 \Rightarrow U$ unit upper triangular

$\equiv$ <u>Crout's facto.</u>

$l_{kk} = 1 \Rightarrow L$ unit lower triang.

$\equiv$ <u>Doolittle facto.</u>

L U



Assume we have already computed first $k \times k$ minors of L, U:

$$a_{kk} = \sum_{s=1}^{k-1} l_{ks} u_{sk} + l_{kk} u_{kk}$$ Let $\boxed{l_{kk} = 1}$ Doolittle

$$\Rightarrow \quad u_{kk} = a_{kk} - \sum_{s=1}^{k-1} l_{ks} u_{sk}$$

compare to A which has n^2.

<u>note</u>: it makes sense to have to specify one of diagonals.

L, U have in total $\frac{n(n+1)}{2} + \frac{n(n+1)}{2}$ elements

$$\begin{cases} a_{kj} = \sum_{s=1}^{k-1} l_{ks} u_{sj} + \underbrace{l_{kk}}_{=1} u_{kj} & k+1 \leq j \leq n \\ a_{ik} = \sum_{s=1}^{k-1} l_{is} u_{sk} + l_{ik} u_{kk} & k+1 \leq i \leq n \end{cases}$$

$$\Rightarrow \quad u_{kj} = \left(a_{kj} - \sum_{s=1}^{k-1} l_{ks} u_{sj}\right) \qquad k \leq j \leq n$$

$$l_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} l_{is} u_{sk}\right) / u_{kk} \qquad k+1 \leq i \leq n$$

LU Algorithm

for $k = 1 : n$,

  $l_{kk} = 1$

  for $j = k : n$

    $u_{kj} = a_{kj} - \sum_{s=1}^{k-1} l_{ks} u_{sj}$ } can be done with BLAS

  end

  for $i = k+1 : n$

    $l_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} l_{is} u_{sk}\right) / u_{kk}$ } "

  end

$\frac{2}{3} n^3$

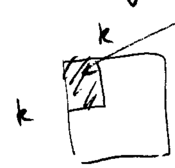
note: algorithm can be done in place overwriting matrix A.

$A = LU$ $\qquad$ $L =$ unit lower triangular

$(\triangle)(\nabla)$ $\qquad$ $U =$ upper triangular

<u>Theorem</u>: If all n leading principal minors of $A \in \mathbb{R}^{n\times n}$ are nonsingular then A has a LU-decomposition.

k-th leading minor $\qquad$ (this is just a sufficient condition)



In many applications A is <u>symmetric positive definite</u>: (s.p.d.)

meaning: $A = A^T$ and

$\forall x \neq 0 \quad x^T A x > 0$.

This means A is non-singular (why?)

$$Ax = 0 \Rightarrow x^T A x = 0 \Rightarrow x = 0.$$

does A have an LU facto? Yes since all principal minors of A are s.p.d. (why?)

$$(x_1, x_2 \ldots x_k, 0 \ldots 0)\, A \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{pmatrix} = (x_1, x_2 \ldots x_k)\, A_k \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}$$

where A_k = k-th principal minor of A
$= A(1:k, 1:k)$

only way this can happen.

$$\left.\begin{array}{l} A = LU \\ \| \\ A^T = U^T L^T \end{array}\right\} \Rightarrow \underset{(\nabla)(\nabla)}{UL^{-T}} = \underset{(\triangle)(\triangle)}{L^{-1}U^T} = D$$

note $\quad (\nabla)^{-1} = \nabla \qquad (\nabla)(\nabla) = (\nabla)$

$\qquad (\triangle)^{-1} = \triangle \qquad (\triangle)(\triangle) = (\triangle)$

$$A = L \underbrace{U L^{-T}}_{D} L^T$$

D must have positive entries since:

$$D = L^{-1} A L^{-T}$$

$$D^{1/2} = \begin{pmatrix} \sqrt{d_{11}} & & & \\ & \sqrt{d_{22}} & & \\ & & \ddots & \\ & & & \sqrt{d_{nn}} \end{pmatrix}$$

$$x^T D x = x^T L^{-1} A L^{-T} x$$
$$= (L^{-T}x)^T A (L^{-T}x) > 0 \quad \text{when } x \neq 0.$$

thus: $A = L D^{1/2} D^{1/2} L^T = \tilde{L} \tilde{L}^T$ where $\tilde{L} = L D^{1/2}$

Theorem: If A is spd then it has a unique Cholesky factorization

$$A = \underset{(\triangle)}{L} \; \underset{(\nabla)}{L^T}$$

Algorithm: Look at matrix matrix product:

$$a_{kk} = \sum_{s=1}^{k-1} l_{ks}^2 + l_{kk}^2$$
$$l_{kk} = \left(a_{kk} - \sum_{s=1}^{k-1} l_{ks}^2 \right)^{1/2}$$

for $k = 1 \ldots n$
  $l_{kk} = \left(a_{kk} - \sum_{s=1}^{k-1} l_{ks}^2\right)^{1/2}$
  for $i = k+1 : n$
    $l_{ik} = \left(a_{ik} - \sum_{s=1}^{k-1} l_{is} l_{ks} \right) / l_{kk}$

Gaussian Elimination

Example:

$$A x = A^{(1)} x = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 1/2 & 3 & 1 & 0 \\ -1 & -1/2 & 2 & 1 \end{bmatrix}$$

$$A^{(2)} x = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ 21 \\ -26 \end{bmatrix}$$

$$A^{(3)} x = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ -9 \\ -21 \end{bmatrix}$$

$$A^{(4)} x = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ -9 \\ -3 \end{bmatrix}$$

sol: $x = \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix}$

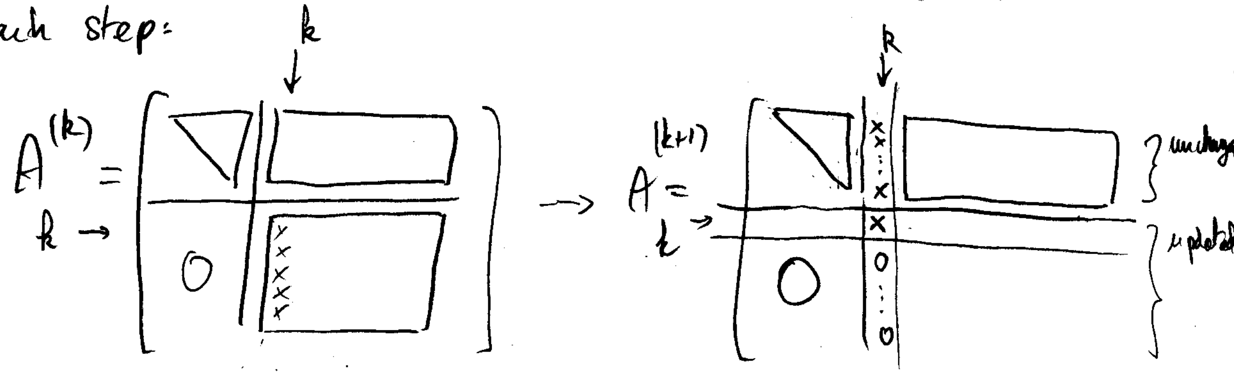
$A^{(3)} = U$

Formally:

$$A = A^{(1)} \to A^{(2)} \to A^{(3)} \to \cdots \to A^{(n)} = U$$

(if all goes well)

at each step:



thus to obtain $A^{(k+1)}$ we need to introduce zeros in column marked by $\times\times\ldots$

and we do not need to change $A^{(k)}(1:k,\ 1:k)$

new entries are:

$$a_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k)} & \text{if } i \leq k \\ a_{ij}^{(k)} - \dfrac{a_{ik}^{(k)} a_{kj}^{(k)}}{a_{kk}^{(k)}} & \text{if } i \geq k \text{ and } j \geq k+1 \\ 0 & \text{if } i \geq k+1 \text{ and } j \leq k \end{cases}$$

and L is defined by:

$$l_{ik} = \begin{cases} a_{ik}^{(k)} / a_{kk}^{(k)} & \text{if } i \geq k+1 \\ 1 & \text{if } i = k \\ 0 & \text{if } i \leq k-1 \end{cases}$$

<u>Theorem</u>: If the <u>pivots</u> $a_{kk}^{(k)} \neq 0$ then $A = LU$

When can this break down?

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

can't add multiple of (1st eq) to introduce new zeros in (2nd eq).

even in situations like:

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$\downarrow$ Gaussian Elim:

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & 1-\varepsilon^{-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2-\varepsilon^{-1} \end{bmatrix}$$

$$x_2 = \frac{(2-\varepsilon^{-1})}{(1-\varepsilon^{-1})} \approx 1$$

if ε is very small.
say $\varepsilon <$ machine epsilon

$$x_1 = (1-x_2)\varepsilon^{-1} \approx 0$$

however true sol is:

$$x_2 = \frac{2\varepsilon - 1}{\varepsilon - 1} \approx 1$$

$$x_1 = \frac{(1-x_2)}{\varepsilon} = \frac{1}{\varepsilon}\left(\frac{\varepsilon-1}{\varepsilon-1} - \frac{(2\varepsilon-1)}{\varepsilon-1}\right)$$

$$= \frac{1}{\varepsilon}\,\frac{-\varepsilon}{\varepsilon-1} = \frac{1}{1-\varepsilon} \approx 1$$

The problem is not only with smallness of a_{11}, but to how small is a_{11} compared to all other elements in its row:

$$\begin{pmatrix} 1 & \varepsilon^{-1} \\ 1 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \varepsilon^{-1} \\ 2 \end{pmatrix}$$ (we just rescaled by ε first row)

GE ↓

$$\begin{pmatrix} 1 & \varepsilon^{-1} \\ 0 & 1-\varepsilon^{-1} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \varepsilon^{-1} \\ 2-\varepsilon^{-1} \end{pmatrix}$$

sol:
$$x_2 = \frac{2-\varepsilon^{-1}}{1-\varepsilon^{-1}} \approx 1$$

$$x_1 = \varepsilon^{-1} - \varepsilon^{-1} x_2 \approx 0$$ which is again wrong!

If we simply exchange rows we get no problem!

$$\begin{pmatrix} 1 & 1 \\ \varepsilon & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

↓ GE

$$\begin{pmatrix} 1 & 1 \\ 0 & 1-\varepsilon \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1-2\varepsilon \end{pmatrix} \Rightarrow \begin{aligned} x_2 &= \frac{1-2\varepsilon}{1-\varepsilon} \approx 1 \\ x_1 &= 2 - x_1 \approx 1 \end{aligned}$$

We usually need to reorder rows to get LU facto: Assume we have a permutation P of $\{1, 2, \dots, n\}$ so that if we carry operations on rows in this order we do not encounter these problems then:

algorithm becomes:

for $k = 1 : n-1$
  for $i = k+1 : n$
    $z = a_{p_i k} / a_{p_k k}$
    $a_{p_i k} = 0$
    for $j = k+1 .. n$
      $a_{p_i j} = a_{p_i j} - z a_{p_k j}$

Gaussian elimination w/scaled row pivoting:

end result is facto:

$$PA = LU$$

where P = permutation matrix: PA = A with rows permuted

To solve $Ax = b$: $\Leftrightarrow PAx = Pb$

$\Leftrightarrow L\underbrace{Ux}_{z} = Pb$

1) solve $Lz = Pb$
2) solve $Ux = z$

In the factorization phase we simply choose for pivot the one that is relatively higher w.r.t rest of its row and we permute it to pivot position. i.e. find p_1 s.t.

$$\frac{|a_{p_1 1}|}{\max\limits_{1 \le i \le n} |a_{p_1 i}|} \geq \frac{|a_{j 1}|}{\max\limits_{1 \le i \le n} |a_{j i}|}$$

Another class of matrices where we can dispense from doing pivoting is:

diagonally dominant matrices:

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|$$: diagonal element is larger than all other element in a row.

Thm: Every diag dominant matrix is non-singular and has an LU facto.

Tridiagonal matrix: Assuming no pivoting is needed, it is very efficient to do Gaussian elimination on such matrices.

$$\begin{bmatrix} d_1 & c_1 & & \\ a_1 & \ddots & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

↑ in matlab: A=diag(a,-1)+diag(d,0)+diag(c,1);

storage for A = 3 vectors.

$$\begin{bmatrix} x & x & & & \\ x & x & x & & \\ & x & x & x & \\ & & x & x & x \\ & & & x & x \end{bmatrix} \longrightarrow \begin{bmatrix} x & x & & & \\ 0 & x & x & & \\ & x & x & x & \\ & & x & x & x \\ & & & x & x \end{bmatrix} \to \begin{bmatrix} x & x & & & \\ & x & x & & \\ & 0 & x & x & \\ & & x & x & x \\ & & & x & x \end{bmatrix} \to \cdots \begin{bmatrix} x & x & & & \\ & x & x & & \\ & & x & x & \\ & & & x & x \\ & & & & x \end{bmatrix}$$

first eq:

$$d_2 = d_2 - \frac{a_1 c_1}{d_1}$$

$$b_2 = b_2 - \frac{a_1 b_1}{d_1}$$

then:

for $i=2..n$

$$d_i = d_i - \frac{a_{i-1} c_{i-1}}{d_{i-1}}$$

$$b_i = b_i - \frac{a_{i-1} b_{i-1}}{d_{i-1}}$$

GE.

$x_n = b_n / d_n$

for $i = n-1:-1:1$

$x_i = (b_i - c_i x_{i+1}) / d_i$

Backward substitution

Norms and the analysis of errors.

Doing LU facto + solution of triangular systems is $O(n^3)$ operations.
How do we know whether error accumulate so much that we cannot trust solution?
We need a tool for measuring error in $\mathbb{R}^n$: norm.

Def (norm): $\|x\|$ is a norm if:

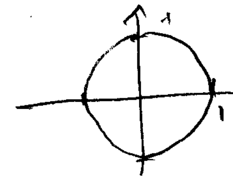
$$\|x\| \geq 0 \quad \forall x \in \mathbb{R}^n$$

$$\|x\| = 0 \iff x = 0$$

$$\|\lambda x\| = |\lambda| \, \|x\| \quad \forall \lambda \in \mathbb{R}, \; x \in \mathbb{R}^n$$

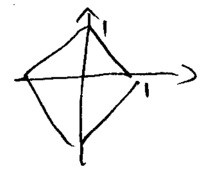
$$\|x+y\| \leq \|x\| + \|y\|$$ triangle inequality.

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

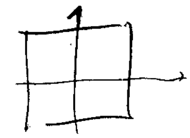


unit ball

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$ l_1 norm



$$\|x\|_\infty = \max_{i=1..n} |x_i|$$ l_∞ norm



<u>Induced matrix norms</u>: Let $\|\cdot\|$ be a norm on $\mathbb{R}^n$ then:

$$\|A\| = \sup_{\|u\|=1} \|Au\| = \sup_{u \neq 0} \frac{\|Au\|}{\|u\|}$$

$\|\cdot\|$ satisfies all axioms of a norm and so is a norm.

Moreover induced matrix norms satisfy

$$\|I\| = 1$$

$$\|AB\| \leq \|A\|\|B\|$$

An important case is $$\|A\|_2^2 = \sup_{x \neq 0} \frac{\|Ax\|^2}{\|x\|^2} = \sup_{x \neq 0} \frac{x^T A^T A x}{x^T x}$$

$$= \lambda_1 (A^T A)$$

$$\Rightarrow \|A\|_2 = \sqrt{\lambda_1 (A^T A)}$$

largest eigenvalue of $A^T A$.

Condition number

Suppose $\tilde{b}$ is a perturbation of b, If $x, \tilde{x}$ satisfy

$$Ax = b$$
$$A\tilde{x} = \tilde{b}$$

then when are $x, \tilde{x}$ close?

$$\|x - \tilde{x}\| = \|A^{-1}b - A^{-1}\tilde{b}\| = \|A^{-1}(b-\tilde{b})\|$$
$$\leq \|A^{-1}\| \, \|b-\tilde{b}\|$$

What about relative error?

$$\|x-\tilde{x}\| \leq \|A^{-1}\| \, \|b-\tilde{b}\| = \|A^{-1}\| \overbrace{\frac{\|Ax\|}{\|b\|}}^{1} \|b-\tilde{b}\|$$
$$\leq \|A^{-1}\| \, \|A\| \, \|x\| \frac{\|b-\tilde{b}\|}{\|b\|}$$

$$(\Rightarrow) \quad \frac{\|x-\tilde{x}\|}{\|x\|} \leq \underbrace{\|A^{-1}\| \, \|A\|}_{\mathcal{K}(A)} \; \frac{\|b-\tilde{b}\|}{\|b\|}$$

$\mathcal{K}(A)$ = condition number of $A = \|A^{-1}\| \, \|A\|$

= measures how sensitive is result of linear system to perturbation.

= depends on vector norm

≥ 1

There are ways of estimating $\mathcal{K}(A)$ efficiently to have an idea on how much we can trust solution.

Here is an example of induced matrix norm.

$$\|A\|_\infty = \sup_{\|u\|_\infty=1} \|Au\|_\infty$$

$$= \sup_{\|u\|_\infty=1} \max_{1\le i\le n} |(Au)_i|$$

$$= \sup_{\|u\|_\infty=1} \max_{1\le i\le n} \left|\sum_{j=1}^{n} a_{ij} u_j\right|$$

$$= \max_{1\le i\le n} \sup_{\|u\|_\infty=1}$$

sup attained when

$$u_j = \operatorname{sgn} a_{ij}$$

$$= \max_{1\le i\le n} \underbrace{\sum_{j=1}^{n} |a_{ij}|}_{\ell_1 \text{ norm of rows}}$$

$$A = \begin{bmatrix} 1 & 1+\varepsilon \\ 1-\varepsilon & 1 \end{bmatrix} \qquad A^{-1} = \varepsilon^{-2}\begin{bmatrix} 1 & -1-\varepsilon \\ -1+\varepsilon & 1 \end{bmatrix}$$

$$\|A\|_\infty = 2+\varepsilon \qquad \|A^{-1}\|_\infty = \varepsilon^{-2}(2+\varepsilon)$$

$$\mathcal{K}(A) = \left(\frac{2+\varepsilon}{\varepsilon}\right)^2 \quad \text{if } \varepsilon \le 0.01 \text{ then } \mathcal{K}(A) \ge 40\,000$$

Neumann Series and Iterative refinement

notion of convergence in $\mathbb{R}^n$:

$$\lim_{k\to\infty} \| v^{(k)} - v \| = 0$$

which norm? In $\mathbb{R}^n$ it doesn't matter! This is because all norms are equivalent in a finite dimensional space:

$$\alpha |||x||| \leq \|x\| \leq \beta |||x||| \text{ ; where } \alpha, \beta > 0 \text{ are constant}$$

and $\|x\|$ and $|||x|||$ are any two norms in $\mathbb{R}^n$.

Also $\mathbb{R}^n$ is complete: any sequence satisfying Cauchy Criterion converges:

$$\boxed{\forall \varepsilon > 0 \quad \exists N \text{ s.t. } \forall i,j \geq N \quad \| v^{(i)} - v^{(j)} \| < \varepsilon}$$

Theorem (Neuman Series)

Let $A \in \mathbb{R}^{n\times n}$ be such that $\|A\| < 1$, then $I - A$ is invertible and:

$$(I-A)^{-1} = \sum_{k=0}^{\infty} A^k$$

Proof:

Assume $I-A$ is singular (for contradiction):

$\exists x \in \mathbb{R}^n$ s.t. $(I-A)x = 0$, take x s.t. $\|x\| = 1$:

$$1 = \|x\| = \|Ax\| \leq \|A\| \|x\| = \|A\|$$

contradiction!

We need to show

$$\sum_{k=0}^{m} A^k \to (I-A)^{-1}$$

$$(I-A)\sum_{k=0}^{m} A^k = \sum_{k=0}^{m} A^k - A^{k+1} = \underbrace{A^0}_{I} - A^{m+1}$$

Now: $\|A^{m+1}\| \le \|A\|^{m+1} \to 0$ [By property of induced matrix norm: $\|AB\| \le \|A\|\|B\|$]

thus

Ask $\longrightarrow$ $$\left\|(I-A)\sum_{k=0}^{m} A^k - I\right\| \le \|A\|^{m+1} \to 0.$$ QED

<u>Note</u>: This theorem is very intuitive if you know geometric series:

$$\frac{1}{1-x} = 1 + x + x^2 + \dots \quad \text{when } |x|<1.$$

For example:

$$\|(I-A)^{-1}\| \le \sum_{k=0}^{\infty} \|A^k\| \le \sum_{k=0}^{\infty} \|A\|^k = \frac{1}{1-\|A\|}$$

<u>Note</u>: This theorem is very useful for the theory, however there are more powerful methods for finding inverse of a matrix

<u>Theorem (Generalization of Neumann series)</u>

If A and B are $n\times n$ matrices s.t. $\|I - \overset{\text{close to id.}}{\widetilde{AB}}\| < 1$ then A, B are invertible and:

$$A^{-1} = B\sum_{k=0}^{\infty}(I-AB)^k$$

$$B^{-1} = \left[\sum_{k=0}^{\infty}(I-AB)^k\right]A$$

proof: $I - (I - AB) = AB$ is invertible and:

$$(AB)^{-1} = \sum_{k=0}^{\infty} (I - AB)^k$$

$$\Rightarrow \quad A^{-1} = B B^{-1} A^{-1} = B \sum_{k=0}^{\infty} (I - AB)^k$$

$$B^{-1} = B^{-1} A^{-1} A = \left(\sum_{k=0}^{\infty} (I - AB)^k \right) A.$$

Iterative refinement

Let $x^{(0)}$ be an approx. sol to:

$$Ax = b$$

then:

$$x = x^{(0)} + \underbrace{A^{-1}(b - Ax^{(0)})}_{\substack{\text{error vector} \\ = e^{(0)}}} - x^{(0)}$$

let $r^{(0)} = b - Ax^{(0)} =$ residual vector:

$$Ae^{(0)} = r^{(0)}$$

Algorithm.

$x^{(0)}$ given

for $k = 0, 1, \ldots$

$$r^{(k)} = b - Ax^{(k)}$$

$$Ae^{(k)} = r^{(k)}$$

$$x^{(k+1)} = x^{(k)} + e^{(k)}$$

Can improve precision of solution, even if we use a "cheap" version of LU factorization that does not compute factors completely.

How to look at this method?

$$x^{(0)} = Bb \qquad \text{where } B = \text{"approx inverse of } A\text{"}$$

$$x^{(k+1)} = x^{(k)} + B(b - Ax^{(k)}), \quad k \geq 0.$$

$B =$ "approx inverse of A" means $\quad \|I - AB\| < 1$

$$A^{-1} = B \sum_{k=0}^{\infty} (I - AB)^k$$

thus
$$x = B \sum_{k=0}^{\infty} (I - AB)^k b \qquad (*)$$
solves system $\quad Ax = b$

<u>Theorem</u> Iterative refinement iterates are:

$$x^{(m)} = B \sum_{k=0}^{m} (I - AB)^k b \quad (m \geq 0)$$

and because of $(*)$:
$$x^{(m)} \to x \quad \text{as} \quad m \to \infty.$$

<u>Proof</u>: By induction on m

<u>$m=0$</u> $\qquad x^{(0)} = Bb$

Assume mth case is true then

$$x^{(m+1)} = x^{(m)} + B(b - Ax^{(m)})$$
$$= B \sum_{k=0}^{m} (I - AB)^k b + Bb - AB \sum_{k=0}^{m} (I - AB)^k b$$
$$= B\left[b + \underbrace{(I - AB) \sum_{k=0}^{m} (I - AB)^k b}_{\sum_{k=0}^{m} (I - AB)^{k+1} b} \right] = B \sum_{k=0}^{m+1} (I - AB)^k b$$