Tridiagonal matrices If no pivoting is needed, then solving systems with a tridiagonal matrix is very efficient ($\mathcal{O}(n)$ flops)

Here is an example of such a system:

$$\begin{bmatrix} d_1 & c_1 & & & \\ a_1 & d_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & & & c_{n-1} \\ & & & a_{n-1} & d_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_n \end{bmatrix}$$

In matlab we can write A using vectors a, d, c :

```
A = diag(a,-1) + diag(d,0) + diag(c,1);
```

In real life we only need to store 3 vectors.

Here is how Gaussian elimination looks like:

$$\begin{bmatrix} \times & \times & & & & \\ \times & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & \times & & & & \\ 0 & \times & \times & & & \\ & \times & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix} \longrightarrow \begin{bmatrix} \times & \times & & & & \\ & \times & \times & & & \\ & 0 & \times & \times & & \\ & & \times & \times & \times & \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

$$\cdots \longrightarrow \begin{bmatrix} \times & \times & & & & \\ & \times & \times & & & \\ & & \times & \times & & \\ & & & \times & \times & \\ & & & & \times & \times \\ & & & & & \times \end{bmatrix}$$

Introducing zeros below first equation:

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

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

(note: here we carry operations for L at the same time on RHS. we are not computing factor L !!)

for $i = 2, .., n$

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

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

(update formulas have only one term

Then to find x we do back substitution, which also simplifies:

$x_n = b_n / d_n$

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

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

Similar algorithms exist for band matrices (i.e. only a few sub, super diagonals are non zero)

Direct Sparse Solvers (brief intro)

A sparse matrix is a matrix with most of its entries being zero.

- saves memory as only non-zero elements are stored.
- saves also computation time as algorithms need only to access non-zero elements.

Here are two common ways of storing a sparse matrix

Coordinate format

iA = row index of nz entries
jA = column " " " "
vA = values of nz entries.

$$a_{iA(k),jA(k)} = vA(k)$$

Compressed sparse column format

PA = pointer to all nz elements in a column.
iA = row indices of nz entries in column
vA = values " " " " "

Example: $A = \begin{bmatrix} 4 & 0 & 3 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & 3 & 0 \\ 5 & 3 & 0 & 6 \end{bmatrix}$

Coordinate

iA	jA	vA
1	1	4
4	1	5
2	2	1
3	2	2
4	2	3
1	3	3
3	3	3
2	4	1
4	4	6

CSC

PA	iA	vA
1	1	4
3	4	5
6	2	1
8	3	2
10	4	3
	1	3
	3	3
	2	1
	4	6

Some popular direct sparse solvers:

SuperLU (Lawrence Berkeley National Lab)

MUMPS (French group: CERFACS, CNRS, INPT, INRIA)

UMFPACK (University of Florida, what is used by Matlab's backslash for solving sparse systems)

...

These are sophisticated libraries that can use a graph representation of a matrix to run the LU (or Cholesky) factorization symbolically and automatically switch between sparse and dense algorithms.

We are not going to see these algorithms in detail. Consider only the following examples to see how sparse structure reduces amount of computations. (these come from http:www.cise.ufl.edu/~davis the creator of UMFPACK homepage)

Matrix vector product: $y = Ax$

Dense version:

$y = 0$
for $j = 1..n$
  for $i = 1..n$
    $y_i = y_i + a_{ij} x_j$

$O(n^2)$

Sparse version:

$y = 0$
for $j = 1..n$
  for each i for which $a_{ij} \neq 0$:
    $y_i = y_i + a_{ij} x_j$

$O(n \times (nz \text{ in a column}))$

Triangular solve: $Lx = b$ wlog: $L_{ii} = 1$ (unit LT.)

$(\Delta)1 = 1$

Dense version:

$x = b$
for $j = 1 \ldots n$
  for $i = j+1 \ldots n$
    $x_i = x_i - l_{ij} x_j$

$O(n^2)$

Sparse version #1

$x = b$
for $j = 1 \ldots n$
  if $x_j \neq 0$
    for each $i > j$ with $l_{ij} \neq 0$
      $x_i = x_i - l_{ij} x_j$

$O(n + |b|)$

Can be made even more efficient if we knew in advance the sparsity pattern of x.

let $\mathcal{X} = \{ i \mid x_i \neq 0 \} \equiv$ sparsity pattern of sol x.

$\mathcal{B} = \{ i \mid b_i \neq 0 \} \equiv$ sparsity pattern of RHS b.

If we knew $\mathcal{X}$ then algo. becomes:

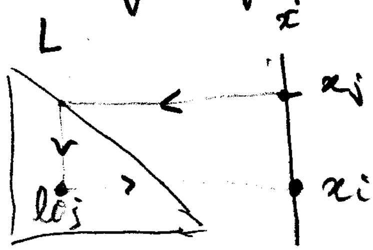
Sparse version #2

$x = b$
for j in $\mathcal{X}$
  for each $i > j$ w/ $l_{ij} \neq 0$
    $x_i = x_i - l_{ij} x_j$

$O(|b|)$ flops

(which can be better than version 1).

The idea to obtain $\mathcal{X}$ is to run triangular solve symbolically in graph representing L.



① $b_j \neq 0 \Rightarrow x_j \neq 0$

② $x_j \neq 0$ and $l_{ij} \neq 0 \Rightarrow x_i \neq 0$

→ Replace L by an oriented graph with an edge $j \to i$ if $l_{ij} \neq 0$.

→ $\mathcal{X} \equiv$ closure (i.e. fixed point) of neighbor ($\mathcal{B}$)

The enemy of sparse direct solvers is <u>fill-in</u>.

<u>for example:</u>
A sparse matrix may not necessarily have a sparse Cholesky factor L.

→ See sparse matrices Matlab demo.

→ See Also: "Direct Methods for Sparse Linear Systems"
T. Davis

→ has a sample code which is simple enough to read.

Norms and the analysis of errors (§7.1 in textbook)

How are we sure that after doing $O(n^3)$ operations to solve a linear system using LU facto + backward substitution that errors we make don't pile up?

$\rightsquigarrow$ we need notion of error in $\mathbb{R}^n$.

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

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

ii) $\|x\| = 0 \Rightarrow x = 0$

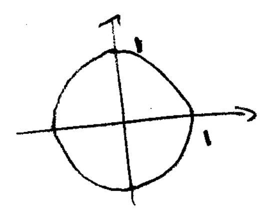
iii) $\|\lambda x\| = |\lambda| \|x\|, \quad \forall x \in \mathbb{R}^n, \lambda \in \mathbb{R}$

iv) $\|x + y\| \leq \|x\| + \|y\|$ (triangle ineq)

Examples:

Euclidean norm

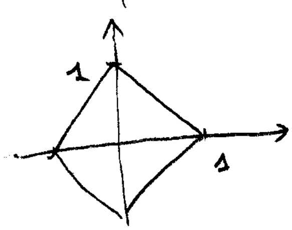
$$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}}$$



unit ball.

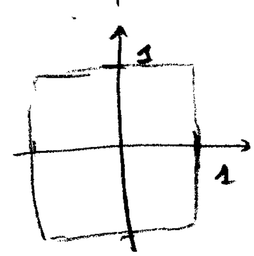
ℓ_1 norm

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



ℓ_∞ norm

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



Induced matrix norm

Let $\|\cdot\|$ be a norm in $\mathbb{R}^n$ then we defined the induced matrix norm

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

$\|\cdot\|$ satisfies all axioms of a norm and:

$$\|I\| = 1$$

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

Examples:

- $$\|A\|_2^2 = \sup_{x \neq 0} \frac{\|Ax\|^2}{\|x\|^2} = \sup_{x \neq 0} \frac{x^T A^T A x}{x^T x}$$

  $$= \lambda_1(A^TA) = \text{largest eigenvalue of } A^TA.$$

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

(recall λ is an eigenvalue of matrix B if there exists $u \neq 0$ s.t.
$$Bu = \lambda u.$$)

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

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

  $$= \sup_{\|u\|_\infty = 1} \max_{1 \leq i \leq n} \Big|\sum_j a_{ij} u_j\Big|$$

  $$= \max_{1 \leq i \leq n} \sup_{\|u\|_\infty = 1} \quad '' \qquad \leftarrow \text{sup attained when } u_j = \operatorname{sgn} a_{ij}$$

  $$= \max_{1 \leq i \leq n} \sum_j |a_{ij}| = \max_{1 \leq i \leq n} \ell_1 \text{ norm } (i\text{-th row of } A)$$

Condition number

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

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

then how close are x and $\tilde{x}$?

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

We know giving a relative error is more informative so here it is.

$$\|x - \tilde{x}\| \leq \|A^{-1}\| \|b - \tilde{b}\| = \|A^{-1}\| \underbrace{\boxed{\frac{\|Ax\|}{\|b\|}}}_{\text{''}1} \|b - \tilde{b}\|$$

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

$$\boxed{\frac{\|x - \tilde{x}\|}{\|x\|} \leq \mathcal{K}(A) \frac{\|b - \tilde{b}\|}{\|b\|}}$$

Here $\mathcal{K}(A)$ = condition number of A
$= \|A\| \, \|A^{-1}\|$ (depends on choice of norm usually $\|\cdot\|_2$ is used)
≥ 1

⇝ <u>quantifies</u> how much we can trust solution x if there are errors in RHS.

⇝ There are ways of estimating $\mathcal{K}(A)$ without computing A^{-1} (in a fast way)

Example:

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

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

$$\Rightarrow \mathcal{K}(A) = \frac{(2+\varepsilon)^2}{\varepsilon^2}.$$

If $\varepsilon \leq 0.01$ then $\mathcal{K}(A) > 40000$.

$\rightsquigarrow$ we loose 4 digits of precision!

As a rule of thumb:

$\log_{10}(\mathcal{K}(A))$ = # of digits of precision lost in solution.

Example:

$$\frac{\|b-\tilde{b}\|}{\|b\|} \approx 10^{-16} \quad \text{(Machine precision)}$$

$$\mathcal{K}(A) \approx 10^{10}$$

$$\rightsquigarrow \frac{\|x-\tilde{x}\|}{\|x\|} \lesssim 10^{-6}.$$

lost 10 digits of precision!