

Parallel Numerics

Prof Huckle *

April 4, 2003

*Technische Universität München
Institut für Informatik

Contents

1	Introduction	4
1.1	Computer Science Aspects of Parallel Numerics	4
1.1.1	Memory Organization	5
1.1.2	Parallel Processors	6
1.1.3	Topology of processors/memory: Interconnection	6
1.1.4	Performance Analysis	8
1.1.5	Further Keywords:	9
1.2	Numerical Problems	10
1.3	Data Dependency Graphs	11
1.3.1	Directed Graph $G=(E,V)$ with edges E and vertices V	11
1.3.2	Dependency Graphs of Iterative Algorithms	14
1.3.3	Dependency graph for solving a triangular linear system	19
2	Elementary Linear Algebra Problems(dense, parallel-vectorised)	23
2.1	BLAS – Basic Linear Algebra Subroutines Program package .	23
2.2	Analysis of Matrix-Vector product	26
2.2.1	Vectorization	27
2.2.2	Parallelization by building blocks	27
2.2.3	$c=A \cdot b$ for banded matrix	29
2.3	Analysis of the Matrix-Matrix-product	31
3	Linear Equations with dense matrices	33
3.1	Gaussian Elimination: Basic facts	33
3.2	Vectorization of the Gaussian Elimination	36
3.3	Gaussian Elimination in Parallel	39
3.3.1	right looking blockwise GE:	39
3.3.2	left looking blockwise GE:	40
3.3.3	Grout form	40
3.4	QR-Decomposition with Householder matrices	41
3.4.1	QR-decomposition	41
3.4.2	Householder method	41
3.4.3	Householder method in parallel	42
4	Linear Equations with sparse matrices	43
4.1	General properties of sparse matrices	43
4.1.1	Storage in coordinate form	44
4.1.2	Compressed Sparse Row Format: CSR	44
4.1.3	Improving CSR	45
4.1.4	Diagonalwise storage	46

4.1.5	rectangular, rowwise storage scheme	46
4.1.6	Jagged diagonal form	47
4.2	Sparse Matrices and Graphs	48
4.2.1	$A = A^T$ (nxn - matrix) symmetric, define Graph $G(A)$:	48
4.2.2	A non symmetric: diagonal graph	49
4.3	Reordering	53
4.3.1	Smaller Bandwidth by Cuthill Mckee-Algorithm	53
4.3.2	Dissection Reordering	54
4.3.3	Algebraic pivoting: reordering during GE	55
4.4	Gaussian Elimination in Graph	58
4.5	Different Methods for solving sparse linear equations	62
4.5.1	Frontal methods for band matrices	62
4.5.2	Partitioning method for tridiagonal matrices	63
4.5.3	Dissection method	65
5	Iterative methods for sparse matrices	67
5.1	stationary methods	67
5.1.1	Richardson Iteration	67
5.1.2	Better splitting of A	68
5.1.3	Jacobi (Diagonal) - Splitting:	68
5.1.4	Gauss-Siedel method by improving convergence	69
5.2	Nonstationary Methods	70
5.2.1	Let A symmetric positive definite $A = A^T > 1$	70
5.2.2	Improving the gradient method \rightarrow conjugate gradients	72
5.2.3	GMRES for General Matrix A	74
5.3	Preconditioning	77
6	Domain Decomposition Methods for Solving PDE	80

Literature: Numerical Linear Algebra for High Performance Computers
Dongarra,....

1 Introduction

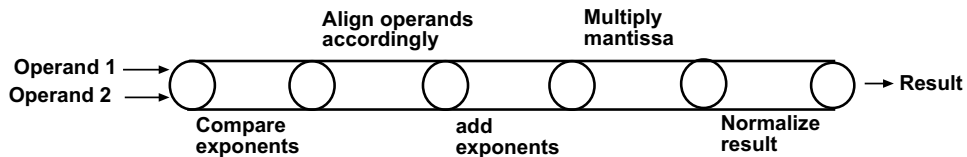
1.1 Computer Science Aspects of Parallel Numerics

Elementary operations in CPU are carried out in pipelines:

Divide a task into a sequence of smaller tasks.

Each small task is executed on a piece of hardware, that operates concurrently with the other stages of the pipeline.

Example: Multiplication



Advantage: If pipeline is filled, per clock one result. All multiplication should be organized such that the pipeline is always filled!

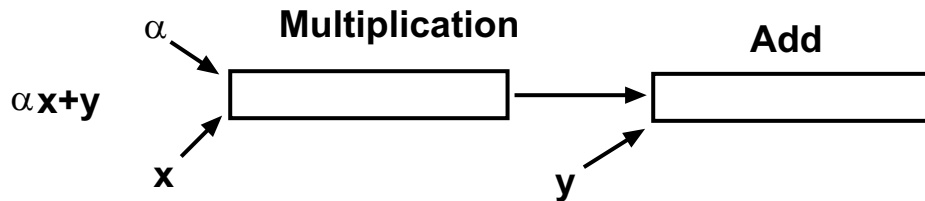
Special Case: Vector instruction: for set of data the same operation has to

be executed: $\alpha \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$

Cost:

Startup time + vector length * clock period

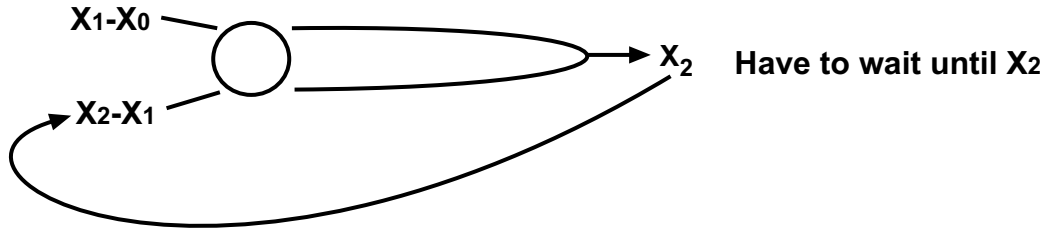
Chaining: Combine pipelines directly:



advantage: total cost: total cost = startup time + vector length

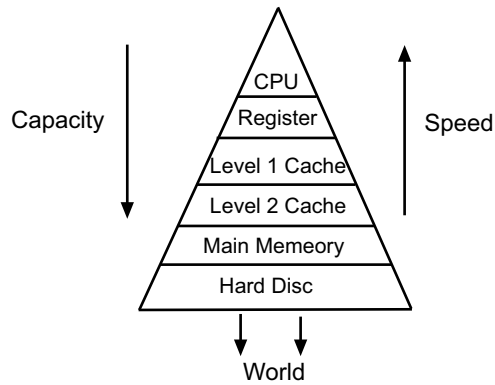
Problem: Data Dependency

Fibonacci: $x_0 = 0, x_1 = 1, x_2 = x_1 + x_0, \dots, x_i = x_{i-1} + x_{i-2}$

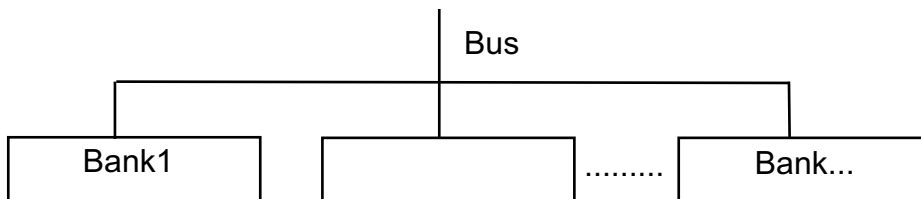


Have to wait until x_2

1.1.1 Memory Organization



Main memory organized in Memory Banks



Cache idea: Buffer between large slow memory and small fast memory.
 By considering the flow of the last used data, we try to predict which data will be required in the next step:

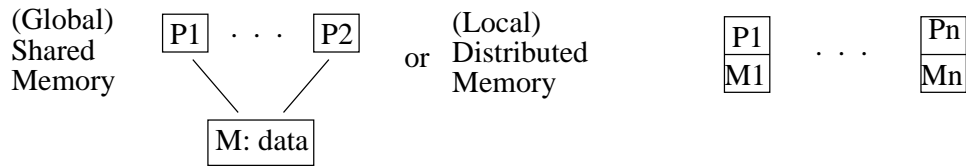
- keep the last used data in cache for fast access
- keep also the neighbourhood of this data in cache

Cache hit: CPU looks for data and data is in cache
 saves a lot of time!

Cache miss: Data is not in cache: look in main memory and additionally
 put data plus the neighbourhood in the cache
 costs more time!

1.1.2 Parallel Processors

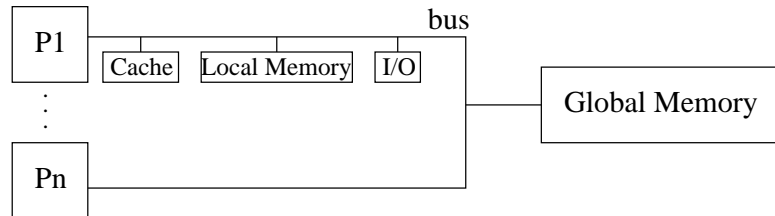
MIMD-Architecture: Multiple Instruction-Multiple Data
 (global)shared memory:



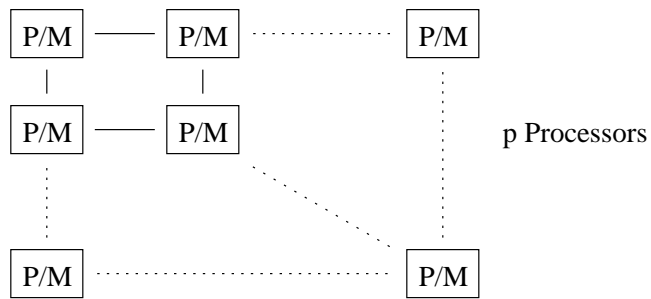
or virtual shared memory: distributed data but organized as shared memory

1.1.3 Topology of processors/memory: Interconnection

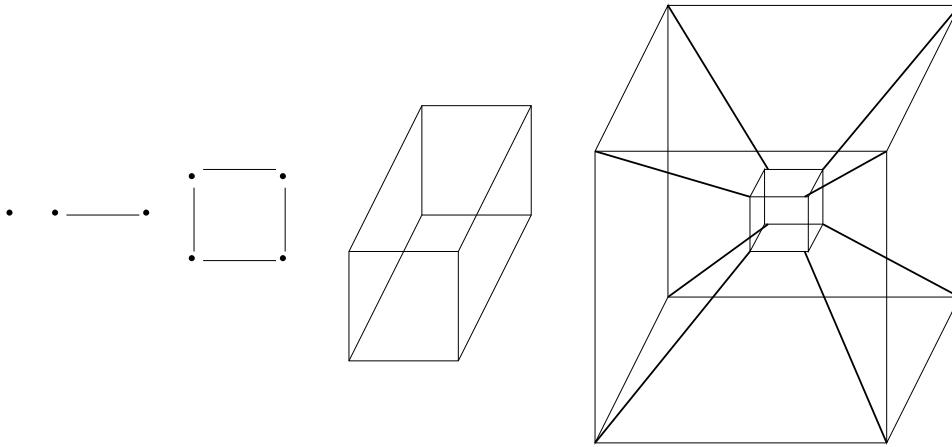
Bus:



Mesh:



Hypercube:



Time to exchange data $\hat{=}$ distance of processors

Shared memory communication for different processors by:

semaphores: variable used to control access to shared data

synchronization: e.g barrier

$$\begin{array}{c}
 \text{halt} \\
 p_1 \left| \right. \\
 \vdots \\
 p_n \left| \right.
 \end{array}
 \longrightarrow \text{continue iff all completed}$$

local memory: Message passing:
send, receive, wait, continue

Data transfer much more costly than flop!

MPI: Message Passing Interface: *Communication library*
for C, C++, FORTRAN

Compiling: mpicc <options> prog.c

Start: mpirun - arch <architecture> - up <up> prog

Commands: MPI_Send
MPI_Bcast
MPI_Recv
MPI_Gather
MPI_Barrier

1.1.4 Performance Analysis

computation speed:

$$r = \frac{N}{t} \text{ Mflops} \quad N \text{ flops in } t \text{ microseconds}$$

or by known speed:

$$t = \frac{N}{r} \text{ flops}$$

Amdahl's law: Algorithm takes N flops

fraction f carried out with speed of V Mflops (good parallel)
fraction $1 - f$ carried out with speed of S Mflops (bad parallel)
fraction f is well-suited for parallel execution ; $1 - f$ is not.

Total CPU-time:

$$t = \frac{f \cdot N}{V} + \frac{(1 - f) \cdot N}{S} = N \left(\frac{f}{V} + \frac{1 - f}{S} \right) \text{ microseconds}$$

Overall Speed (Performance):

$$r = \frac{N}{t} = \frac{1}{\frac{f}{V} + \frac{1-f}{S}} \text{ Mflops} \quad \underline{\text{Amdahl's law}}$$

Interpretation: f must be close to 1 in order to benefit significantly from parallelism

Speedup by using p parallel processors for a job:

$t_j :=$ wall clock time to execute the job on j parallel processor

Speedup: $S_p = t_1/t_p$

Efficiency: $E_p = \frac{S_p}{p}$ $0 \leq E_p \leq 1$

$E_p \approx 1$: very good parallelizable $t_p = t_1/p$ problem scales

$$t_p = \frac{\overbrace{ft_1}^{\text{parallel}}}{p} + (1-f)t_1 = t_1 \frac{f + (1-f)p}{p} \geq (1-f)t_1$$

$$S_p = \frac{1}{\frac{f-(1-f)p}{p}} = \frac{p}{f + (1-f)p} \quad \text{Ware's law}$$

$$E_p = \frac{1}{f + (1-f)p}$$

Assume that the problem can be solved in 1 unit of time on a parallel machine with p processors.

Again fraction f is a good parallelizable, $1-f$ is not:

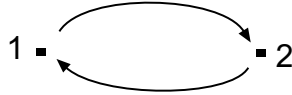
Compared with the parallel implementation, a uniprocessor would perform $(1-f) + f \cdot p$ for the same job

Speedup:

$$S_{pf} = \frac{t_1}{t_p} = \frac{1-f+fp}{1} = p + (1-p)(1-f) \quad \text{Gustafson's law}$$

1.1.5 Further Keywords:

- An algorithm is scaling iff with p processors we can reduce the operation time by a factor of p
(This means: speedup $\approx p$, efficiency ≈ 1)
- load balancing: The job has to be distributed such that all processors are busy: Avoid idle processors
- deadlock: two or more processors are waiting indefinitely for an event that can be caused by one of the waiting processors



Each waiting for the results of the other one

- data dependency: compute

$$C = A + B \quad (1)$$

$$Z = C \cdot X + Y \quad (2)$$

(2) can be computed only after (1)

example loop:

$$\text{for}(i = 1; \quad i \leq n; \quad i++) \quad a[i] = b[i] + a[i - 1] + c[i]$$

1.2 Numerical Problems

vectors $x, y \in \mathbb{R}^n$

dot product (inner product) $x^T y = (x_1, \dots, x_n) \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \sum_{i=1}^n x_i y_i$

sum of vectors: $x + \alpha y = \begin{pmatrix} x_1 + \alpha y_1 \\ \vdots \\ x_n + \alpha y_n \end{pmatrix}$

outer product: $xy^T = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} (y_1, \dots, y_n) = \begin{pmatrix} x_1 y_1 & \dots & x_1 y_n \\ \vdots & & \vdots \\ x_n y_1 & \dots & x_n y_n \end{pmatrix}$

nb xy^T rank-1-matrix

matrix product: $A \in \mathbb{R}^{n,k}, B \in \mathbb{R}^{k,m}, C=A \cdot B \in \mathbb{R}^{n,m}$

$$\begin{pmatrix} a_{11} & \dots & a_{1k} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nk} \end{pmatrix} \begin{pmatrix} b_{11} & \dots & b_{1m} \\ \vdots & & \vdots \\ b_{k1} & & b_{km} \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1m} \\ \vdots & & \vdots \\ c_{n1} & & c_{nm} \end{pmatrix}$$

with $c_{ij} = \sum_{r=1}^k a_{ir} b_{rj}, \quad \begin{matrix} i=1, \dots, n \\ j=1, \dots, m \end{matrix}$

Solving linear equations,

e.g. triangular:
$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ 0 & a_{21} & a_{2n} \\ \vdots & \ddots & \\ 0 & 0 & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

or

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &= b_1 \\ a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{nn}x_n &= b_n \end{aligned}$$

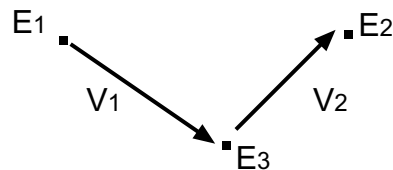
solution : $x_n = \frac{b_n}{a_{nn}}, \quad a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1}$
 $\Rightarrow x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}$

general form $x_j = \frac{b_j - \sum_{k=j+1}^n a_{jk}x_k}{a_{jj}}$

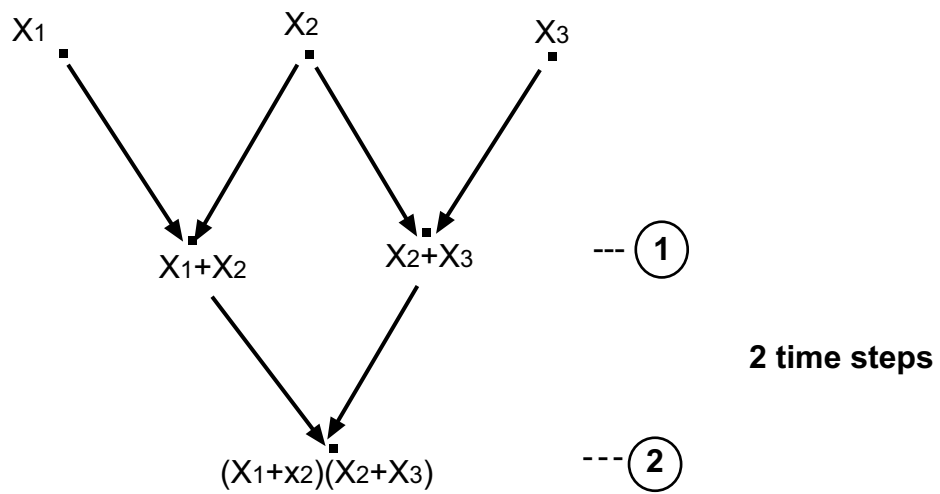
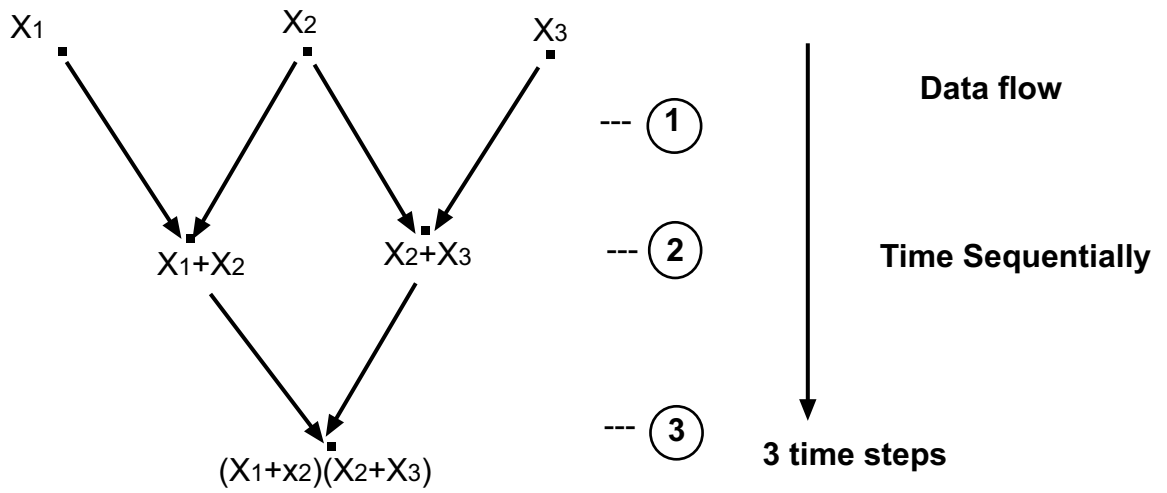
- Gaussian Elimination, LU-Decomposition (Cholesky-Decomposition)
- Least Squares problem (normal equation) $\min_x \|Ax - B\|_2$
- QR -Decomposition
- Differential Equations (PDE, elliptic) $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$
- eigenvalues, singular values, FFT

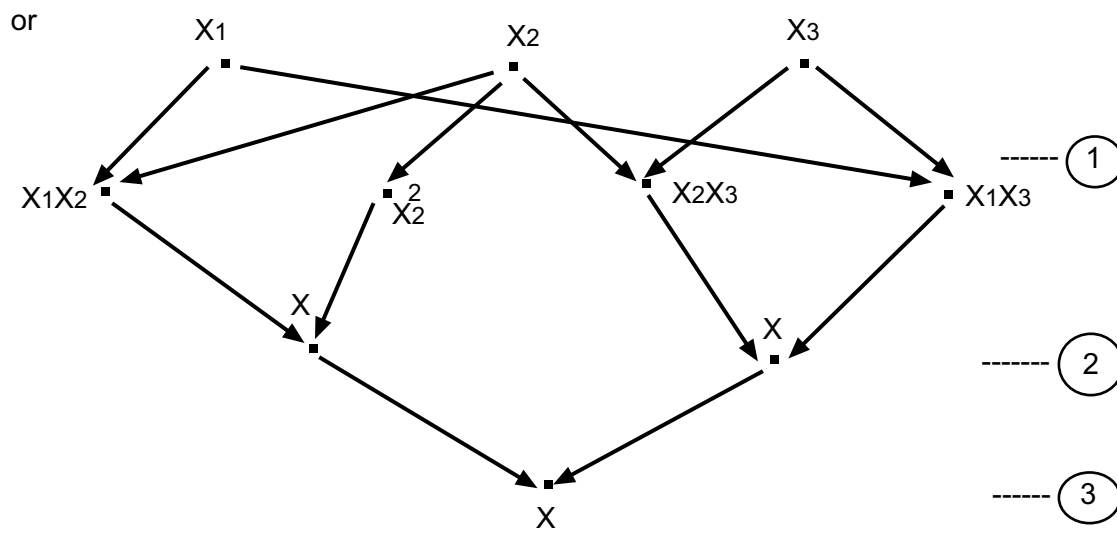
1.3 Data Dependency Graphs

1.3.1 Directed Graph $G=(E,V)$ with edges E and vertices V



Example: Computation of $(x_1 + x_2)(x_2 + x_3) = x_1x_2 + x_2^2 + x_1x_3 + x_2x_3$
 In parallel: $x_1 + x_2$ and $x_2 + x_3$ can be computed independently





1.3.2 Dependency Graphs of Iterative Algorithms

Given: Function f , start x_0 , $x_{k+1} = f(x_k)$

$$x_k \xrightarrow{x \rightarrow \infty} \bar{x} = f(\bar{x}) \quad \text{fix point of } f$$

compare Newton's method $x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$ $g(\bar{x}) = 0$

$$\text{In vector form: } \begin{pmatrix} x_1(k+1) \\ \vdots \\ x_n(k+1) \end{pmatrix} = \vec{x}_{k+1} = \vec{f}\vec{x}_{k+1} = \begin{pmatrix} f_1(x_1(k)_1), \dots, x_n(k) \\ \vdots \\ f_n(x_1(k)_n), \dots, x_n(k) \end{pmatrix}$$

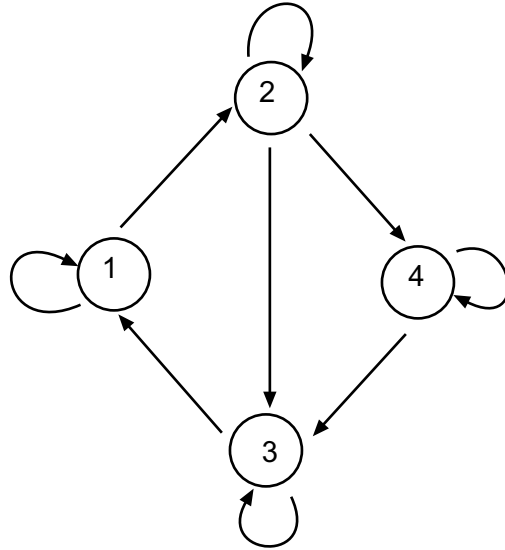
Example:

$$x_1(k+1) = f_1(x_1(k), x_3(k))$$

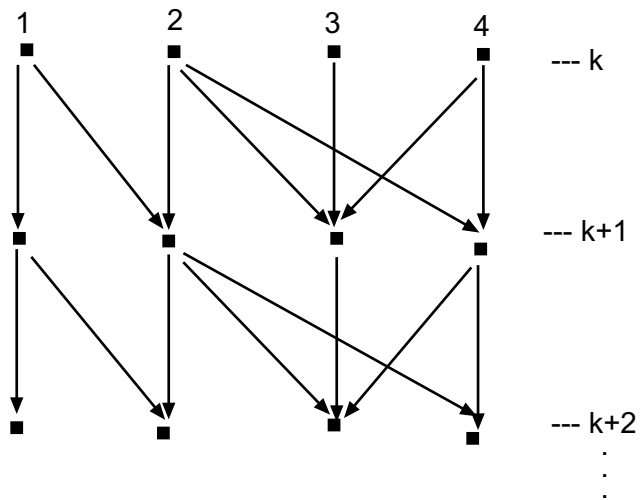
$$x_2(k+1) = f_2(x_1(k), x_2(k))$$

$$x_3(k+1) = f_3(x_2(k), x_3(k), x_4(k))$$

$$x_4(k+1) = f_4(x_2(k), x_4(k))$$



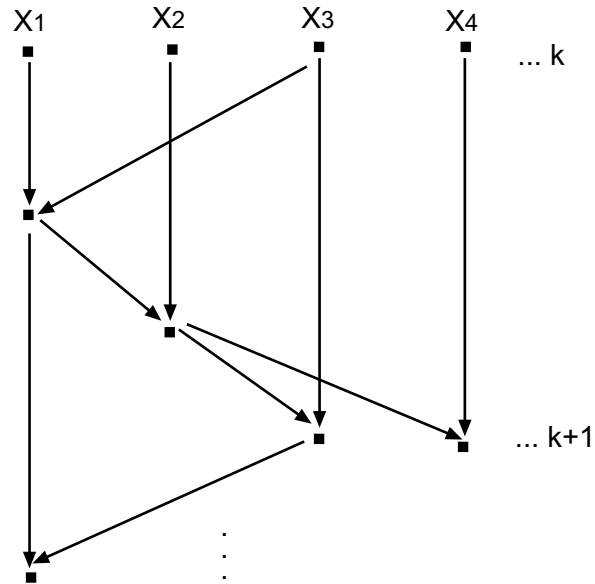
Dependency Graph for Iteration
Single-step or Jacobi-method



Idea for accelerating the Convergence:
Use always the newest available information:

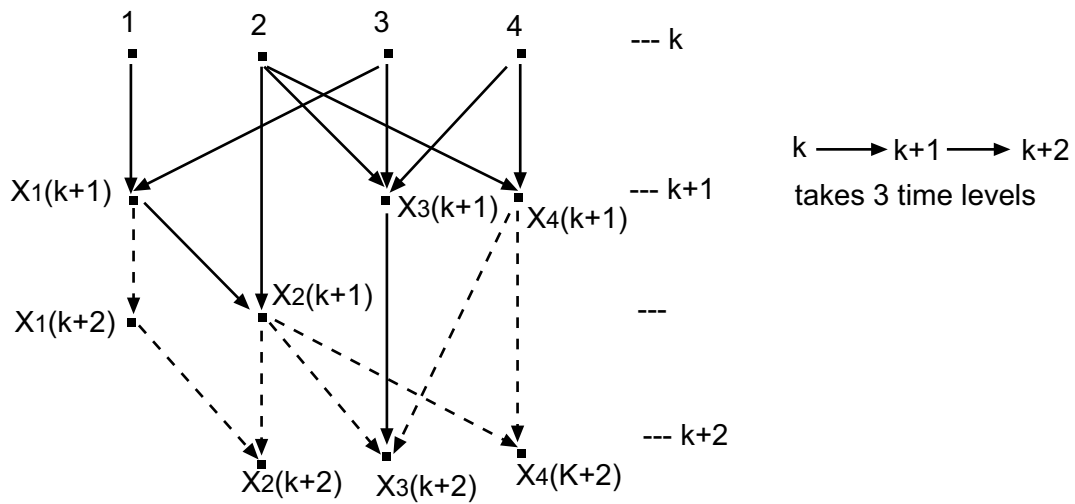
$$\begin{aligned}
 x_1(k+1) &= f_1(x_1(k), x_3(k)) \\
 x_2(k+1) &= f_2(x_1(k+1), x_2(k)) \\
 x_3(k+1) &= f_3(x_2(k+1), x_3(k), x_4(k)) \\
 x_4(k+1) &= f_4(x_2(k+1), x_4(k))
 \end{aligned}$$

Full-step or Gauss-Seidel-method
 Drawback: loss of Parallelism:



In this form the iteration depends on the ordering of the variables x_1, \dots, x_n
 e.g.

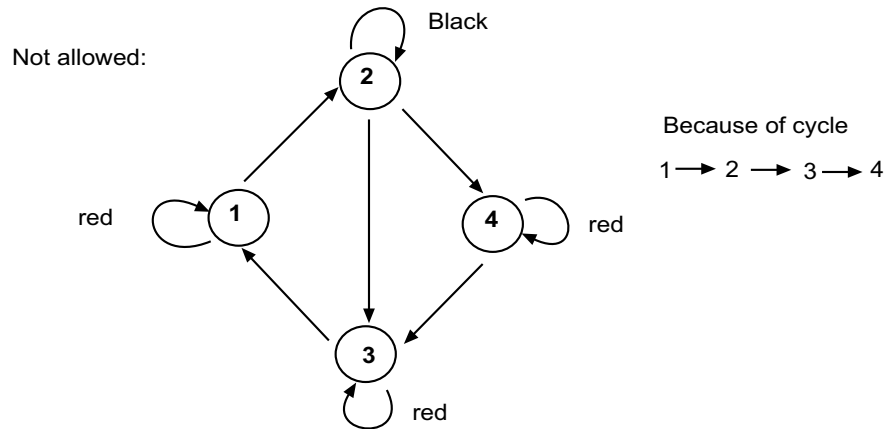
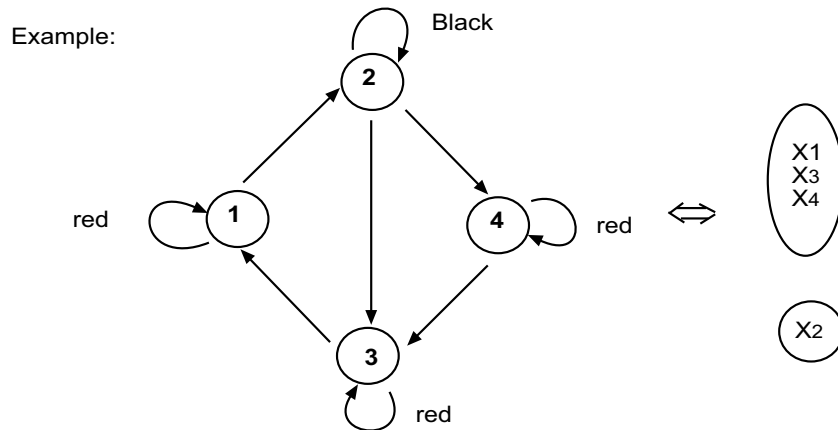
$$\begin{aligned}
 x_1(k+1) &= f_1(x_1(k), x_3(k)) \\
 x_3(k+1) &= f_3(x_2(k), x_3(k), x_4(k)) \\
 x_4(k+1) &= f_4(x_2(k), x_4(k)) \\
 x_2(k+1) &= f_2(x_1(k+1), x_2(k))
 \end{aligned}$$



Better parallelism, but slower convergence
 Colouring algorithms for dependency graphs

- use k colours for the edges
- edges with the same colour should be computed in parallel
- optimal colouring for minimal k , but without cycles for edges with the same colour.

Example:



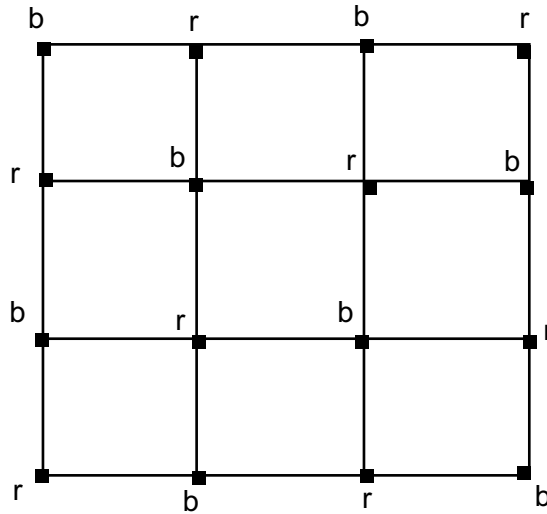
Theorem 1 *Theorem: Two statements are equivalent:*

- a. *There exists an ordering,, such that the one Gauss-Seidel-Iteration-step takes k (time) levels*
- b. *There exists a colouring with k colours, such that there is no cycle of edges of the same colour.*

Proof: Colouring with no cycles \rightarrow ordering of the edges \rightarrow Gauss-Seidel

Colouring \leftrightarrow Partitioning of the edges in groups, such that each group can be computed in one step depending only on other groups.

Graph from discretization of physical problems in \mathbb{R} ; neighbour -connections $k=2$



1.3.3 Dependency graph for solving a triangular linear system

$$\begin{aligned}
 a_{11}x_1 &= b_1 \\
 a_{21}x_1 + a_{22}x_2 &= b_2 \\
 a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \\
 a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4
 \end{aligned}$$

or

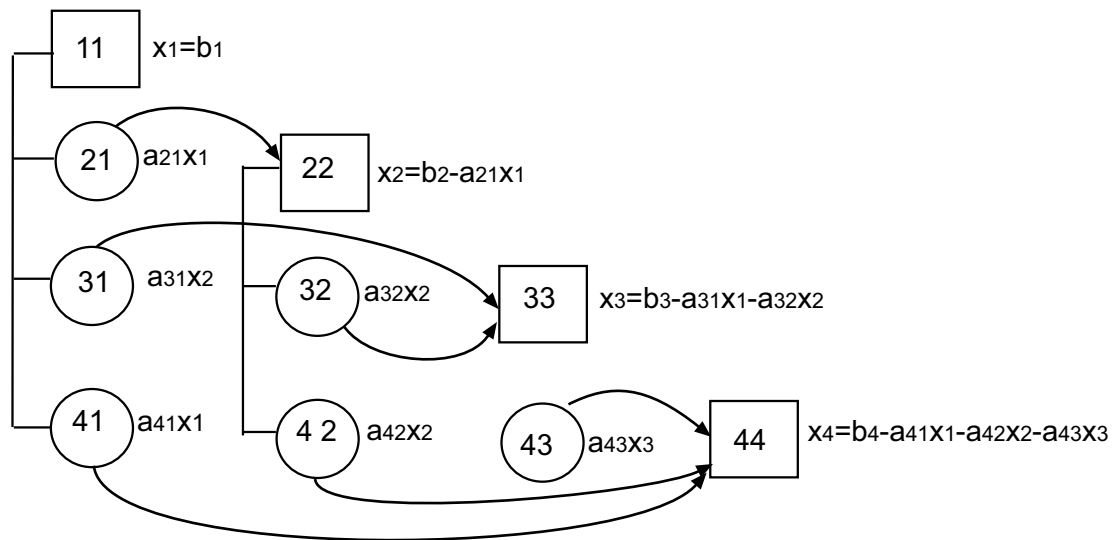
$$\begin{pmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

without loss of generality: $a_{ii} = 1$

solution:

$$\begin{aligned}
 x_1 &= b_1/a_{11} \\
 x_2 &= (b_2 - a_{21}x_1)/a_{22} \\
 x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\
 x_4 &= (b_4 - a_{41}x_1 - a_{42}x_2 - a_{43}x_3)/a_{44}
 \end{aligned}$$

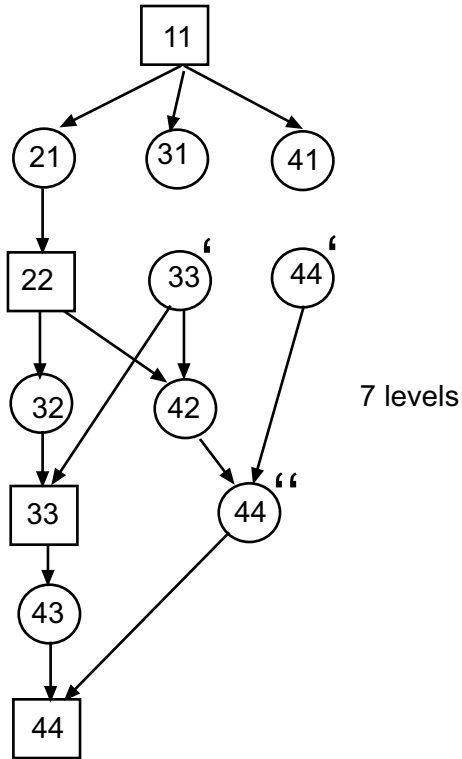
General: $x_k = (b_k - \sum_{j=1}^{k-1} a_{kj}x_j)/a_{kk}$ for $k = 1, \dots, n$



Dependency Graph: Different variations:

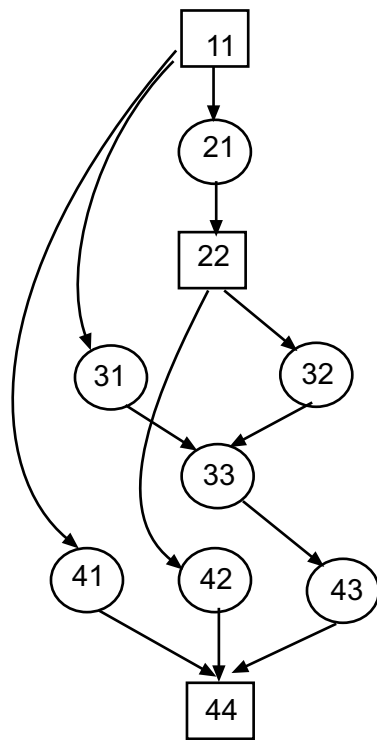
1. columnwise

$$\begin{array}{l}
 x_1 = b_1 \left| \begin{array}{l} a_{21}x_1 \quad b_2 \rightarrow b_2 - a_{21}x_1 \\ a_{31}x_1 \quad b_3 \rightarrow b_3 - a_{31}x_1 \\ a_{41}x_1 \quad b_4 \rightarrow b_4 - a_{41}x_1 \end{array} \right| x_2 = b_2 \left| \begin{array}{l} a_{32}x_2 \quad b_3 \rightarrow b_3 - a_{32}x_2 \\ a_{42}x_2 \quad b_4 \rightarrow b_4 - a_{42}x_2 \end{array} \right| x_3 = \\
 b_3 \left| a_{43}x_3 \quad b_4 \rightarrow b_4 - a_{43}x_3 \right| x_4 = b_4
 \end{array}$$



2. rowwise

$$\begin{aligned}
 x_1 &= b_1 \\
 b_{21}x_1 \\
 x_2 &= b_1 - b_{21}x_1 \\
 b_{31}x_1 \quad b_{32}x_2 \\
 x_3 &= b_3 - b_{31}x_1 - b_{32}x_2 \\
 b_{41}x_1, \quad b_{42}x_2, \quad b_{43}x_3 \\
 x_4 &= b_4 - b_{41}x_1 - b_{42}x_2 - b_{43}x_3
 \end{aligned}$$

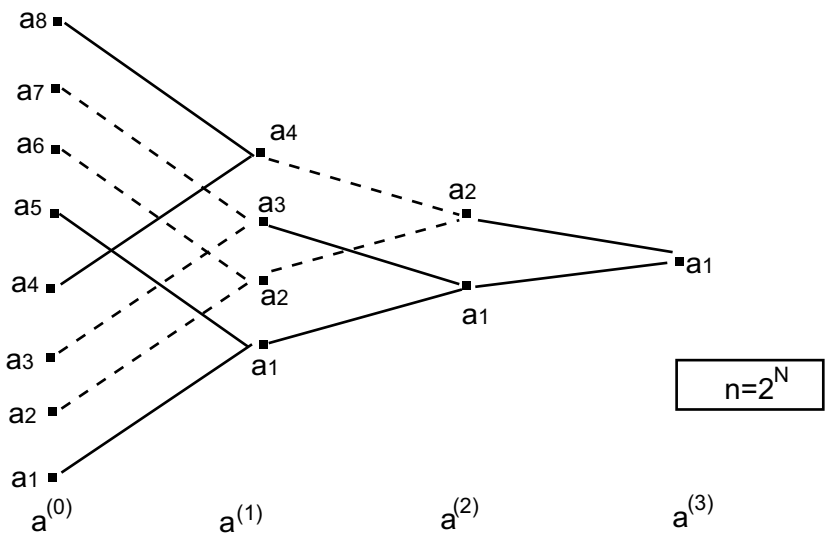


2 Elementary Linear Algebra Problems(dense, parallel-vectorised)

2.1 BLAS – Basic Linear Algebra Subroutines Program package

Sum:

$$s = \sum_{i=1}^n a_i \quad \text{by fan-in process}$$



$$a^{(k)} = \begin{pmatrix} a_1^{(k)} \\ \vdots \\ a_{2^{N-k}}^{(k)} \end{pmatrix} = \begin{pmatrix} a_1^{(k-1)} \\ \vdots \\ a_{2^{N-k}}^{(k-1)} \end{pmatrix} + \begin{pmatrix} a_1^{(k-1)} \\ 2^{N-k} + 1 \\ \vdots \\ a_{2^{N-k}}^{(k-1)} \end{pmatrix}$$

Grouping: $a_1 + \dots + a_8 = [(a_1 + a_5) + (a_3 + a_7)] + [(a_2 + a_6) + (a_4 + a_8)]??$

for $(k = 1; k \leq N; k++)$

for $(j = 1; j \leq 2^{N-k}; j++)$

$$a_j = a_j + a_{j+2^{N-k}};$$

end

end

full binary tree with $n = 2^N$ leaves
 \Rightarrow depth $\hat{=}$ time $\log n = N$ (sequential: $O(n)$)

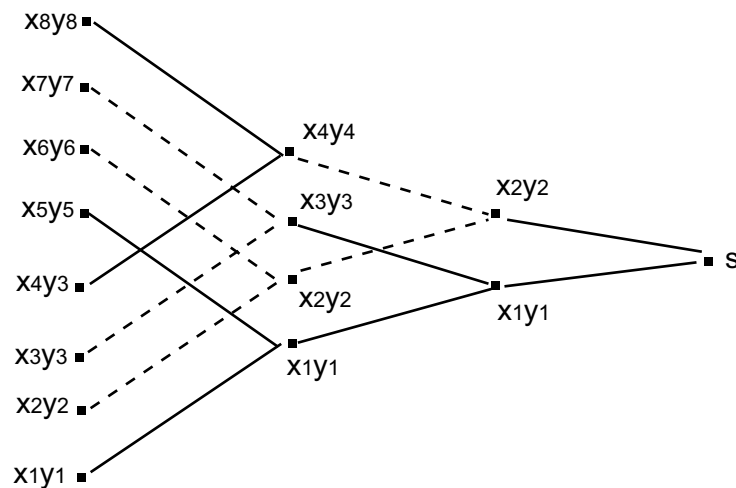
Level-1 BLAS: Basic Linear Algebra Subroutines with $O(n)$ problems
 (Vectors only)

e.g. DOT-product

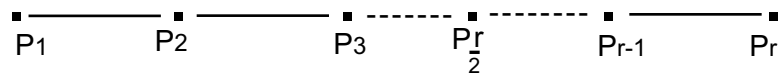
$$s = x^T y = \sum_{j=1}^n x_j y_j$$

by fan-in

Parallelization of dot-product: $\sum_{j=1}^n x_j y_j$



Distribute data on linear 1 dimensional processor array
 with $r = n/k$ processors



Each processor computes $a_{j_1} b_{j_1} + \dots + a_{j_k} b_{j_k}$

time for this parallel computation: $k \cdot (\text{add} + \text{mult})$ i.e. time for one Addition/Multiplication

After computing this part, processor P_1/P_r sends his result to the right/left neighbour, which adds the new data to his own result, and sends the new

data to his right/left neighbour until $P_{r/2}$ holds the final number.
 Total time:

$$f(r) = K(\text{add} + \text{multi}) + \frac{r}{2} \cdot \text{send} + \frac{r}{2} \cdot \text{add} = \frac{n}{r} \cdot (\text{add} + \text{multi}) + r \cdot \frac{a+s}{2}$$

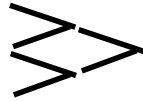
Minimize total time $f(r)$:

$$0 = f'(r) = -\frac{a+m}{r^2}n + \frac{a+s}{2} \Rightarrow r = \sqrt{\frac{a+s}{2(a+m)}} \cdot \sqrt{n} = O(\sqrt{n})$$

optimal: with \sqrt{n} processors, the time is $O(\sqrt{n})$

Replace linear Array by binary tree:

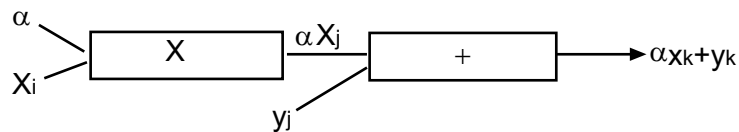
Then with n processors the total time is $O(\log n)$.



Further level-1 BLAS problems

$$\begin{array}{l} S - \text{single} \\ A - \alpha \\ X - \vec{x} \\ P - + \\ Y - \vec{y} \end{array} \quad : \quad \vec{y} = \alpha \vec{x} + \vec{y}$$

by pipelining vectorization:



Parallelization by partitioning: $\langle 1, n \rangle = \{1, 2, 3, \dots, n\} = I_1 \cup I_2 \cup \dots \cup I_R$

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_R \end{pmatrix}, \vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \end{pmatrix}$$

Each processor gets \vec{x}_r and \vec{y}_r and computes $\alpha \vec{x}_r + \vec{y}_r$
 very good vectorizable and parallelizable:

SCOPY:

$$\vec{y} = \vec{x}$$

NORM:

$$\|x\|_2 = \sqrt{\sum_{j=1}^n x_j^2} \quad \text{compare DOT}$$

Level-2 BLAS: Matrix-Vector

$$\left. \begin{array}{l} S \\ E \\ E \\ M \\ V \end{array} \right\} \begin{array}{l} \text{single precision} \\ \\ \text{general matrix : } \vec{y} = \alpha A \vec{x} + \beta \vec{y} \\ \\ \text{vector} \end{array}$$

or solving triangular system $Lx = b$, L is lower triangular matrix

Level-3 BLAS: Matrix-Matrix

$$\left. \begin{array}{l} S \\ G \\ E \\ M \\ M \end{array} \right\} \begin{array}{l} \text{single precision} \\ \\ \text{general matrix : } \vec{y} = \alpha AB + \beta C \\ \\ \end{array}$$

Based on BLAS: LAPACK- subroutines for solving linear equations, least squares, QR-decomposition, eigenvalues, singular values.

2.2 Analysis of Matrix-Vector product

$$A = (a_{i=1..n, j=1..m}) \quad \mathbb{R}^{n,m}, \quad b \in \mathbb{R}^m, \quad c \in \mathbb{R}^n$$

2.2.1 Vectorization

$$\begin{aligned}
 \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} &= \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \\ a_{n1} & \dots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + \dots + a_{1m}b_m \\ \vdots \\ a_{n1}b_1 + \dots + a_{nm}b_m \end{pmatrix} \\
 &= \underbrace{\begin{pmatrix} \sum^m a_{1j}b_j \\ \vdots \\ \sum^m a_{nj}b_j \end{pmatrix}}_{\text{collection of DOT}} = \underbrace{\sum_{j=1}^m b_j}_{\text{collection of SAXPY's}} \begin{pmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{pmatrix} \rightarrow \text{GAXPY}
 \end{aligned}$$

(*ij*)-form: $c = \emptyset$;

```

for i = 1,...,n
  for j = 1,...,m
     $c_i = c_i + a_{ij}b_j$ 
  end
end

```

} *DOT*, where $c_i = (a_i) \cdot b$

(*ji*)-form:

```

for j = 1,...,m
  for i = 1,...,n
     $c_i = c_i + a_{ij}b_j$ 
  end
end

```

} *SAXPY*

} $c = c + b_j(a_j)$

} Add j-th column of A

} *GAXPY*

} Number of SAXPY's

} with the same vector c

Advantage of GAXPY: keep c in fast register memory
 SAXPY/GAXPY good vectorizable

2.2.2 Parallelization by building blocks

Reduce Matrix-vector on smaller Matrix-vector on preprocessors

$\langle 1, n \rangle = \{1, 2, 3, \dots, n\} = I_1 \cup I_2 \cup I_3 \dots I_R$ disjoint: $I_j \cap I_k = \emptyset$

$\langle 1, m \rangle = J_1 \cup J_2 \cup J_3 \dots J_s$ $J_j \cap J_k = \emptyset$ for $j \neq k$

processor P_{rs} gets matrix $A_{rs} := A(I_r, J_s)$, $b_s = b(J_s)$, $c_r = c(I_r)$

$$I_r \left(\begin{matrix} J_s \\ A_{rs} \end{matrix} \right) \cdot \left(\begin{matrix} \bar{b}_s \end{matrix} \right) \} J_s = \left(\begin{matrix} \bar{c}_r \end{matrix} \right) \} I_r$$

$$c_r = \sum_{s=1}^S A_{rs} \cdot b_s = \sum_1^S c_r^{(s)} \text{ for } r = 1, \dots, R$$

for r = 1, ..., R	}	small, independent matrix-vector products no communication
for s = 1, ..., S		
$c_r^{(s)} = A_{rs} \cdot b_s$		
end		
end		

for r = 1, ..., R	}	blockwise collection and addition of vectors rowwise communication
$c_r = 0$		
for s = 1, ..., S		
$c_r = c_r + c_r^{(s)}$		
end		
end		

Special case S=1:

$$c = \begin{pmatrix} A_1. \\ A_2. \\ \vdots \end{pmatrix} \cdot b = \begin{pmatrix} A_1. \cdot b \\ A_2. \cdot b \\ \vdots \end{pmatrix} \text{ no communication between processors } \begin{pmatrix} P_1 \\ P_2 \\ \vdots \end{pmatrix} :$$

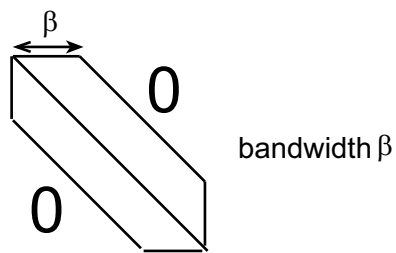
compute $A_i \cdot b$ in vectorizable form by GAXPY's

$$\text{special case } R=1: c = (A_{.1}|A_{.2}|\dots) \cdot \begin{pmatrix} b_1 \\ b_2 \\ \vdots \end{pmatrix} = A_{.1}b_1 + A_{.2}b_2 + \dots$$

$A_{.i} \cdot b_i$ independent, then collection of result $P_1 \dots P_s$

Rule: inner loop of a program should be simple, vectorizable
outer loop should be complex, independent, parallelizable

2.2.3 $c=A \cdot b$ for banded matrix



e.g $\beta = 1$: tridiagonal matrix

$$A = \begin{pmatrix} \ddots & \ddots & 0 & - & 0 \\ \ddots & \ddots & \ddots & & | \\ 0 & \ddots & \ddots & \ddots & 0 \\ | & & \ddots & \ddots & \ddots \\ 0 & - & 0 & \ddots & \ddots \end{pmatrix} \quad \text{notation} \longrightarrow$$

$$\tilde{A} = \begin{pmatrix} \tilde{a}_{10} & \tilde{a}_{11} & \tilde{a}_{1\beta} & 0 & \dots & 0 \\ \tilde{a}_{2,-1} & \tilde{a}_{20} & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & 0 \\ \vdots & \ddots & \ddots & \ddots & & \tilde{a}_{n-\beta,\beta} \\ \tilde{a}_{\beta+1,-\beta} & & \ddots & \ddots & \ddots & \\ 0 & & & \ddots & \ddots & \vdots \\ 0 & - & 0 & \tilde{a}_{n,-\beta} & \dots & \tilde{a}_{n0} \end{pmatrix}$$

$$\longrightarrow \begin{pmatrix} 0 & \dots & 0 & \tilde{a}_{1,0} & \dots & \dots & \tilde{a}_{1,\beta} \\ \vdots & / & / & \vdots & \vdots & \vdots & \vdots \\ 0 & / & & \vdots & . & . & \vdots \\ \tilde{a}_{\beta+1,-\beta} & \vdots & \vdots & \vdots & . & . & \vdots \\ \vdots & . & . & \vdots & . & . & \vdots \\ \vdots & . & . & \vdots & . & . & \tilde{a}_{n-\beta,\beta} \\ \vdots & . & . & \vdots & . & / & 0 \\ \vdots & . & . & . & / & / & \vdots \\ \tilde{a}_{n,-\beta} & \dots & \dots & \tilde{a}_{n,0} & 0 & \dots & 0 \end{pmatrix}$$

$\tilde{a}_{is} = a_{i,i+s}$ for row $i = 1, \dots, n$

$S \in [l_i, r_i] = [\max\{-\beta, 1 - i\}, \min\{\beta, n - i\}]$

Therefore we get the inequality

$$1 - i \leq S \leq n - i, \quad -\beta \leq S \leq \beta, \quad 1 - S \leq i \leq n - S$$

e.g.

$$\begin{array}{ll} \text{row } i = 1 : & S \in [0, \beta] \\ & \vdots \\ \text{row } i = \beta + 1 : & S \in [-\beta, \beta] \\ & \vdots \\ & i = n - \beta : S \in [-\beta, \beta] \\ & \vdots \\ & i = n : S \in [-\beta, 0] \end{array}$$

computation of matrix-vector-product $C = A \cdot b$ on vector processor

$$C_i = A_{ij} \cdot b = \sum_j a_{ij} b_j = \sum_{S=l_i}^{r_i} a_{i,i+S} \cdot \underbrace{b_{i+S}}_j = \sum_{S=l_i}^{r_i} \tilde{a}_{i,S} \cdot b_{i+S}$$

for $i = 1, \dots, n$

Algorithm:

```

for s = -β : 1 : β
  for i = max{1-s, 1} : 1 : min {n-s, n}
     $c_i = c_i + \tilde{a}_{ij} b_{i+s}$ 
  end
end
end

```

} general triade (no SAXPY)

parallel computation:

$$\langle 1, n \rangle = \bigcup_{r=1}^R I_r$$

```

for i ∈ Ir
   $c_i = \sum_{s=l_i}^{r_i} \tilde{a}_{is} \cdot b_{i+s}$ 
end

```

Processor P_r gets rows to index set $I_r := [m_r, M_r]$ to compute its part of C.

What part of vector b is necessary to process P_r ?

b_j for
 $j = i + s \geq m_r + l_{m_r} = m_r + \max\{\beta, 1 - m_r\} = \max\{m_r - \beta, 1\}$
 $j = i + s \leq M_r + r_{M_r} = M_r + \min\{\beta, m - M_r\} = \max\{\beta - M_r, m\}$ [1ex]
 Hence processor P_r ($\sim I_r$) needs
 b_j for $j \in [\max\{1, m_r - \beta\}, \min\{m, M_r + \beta\}]$.

2.3 Analysis of the Matrix-Matrix-product

$$A = (a_{ij})_{\substack{i=1..n \\ j=1..m}} \quad B = (b_{ij})_{\substack{i=1..m \\ j=1..q}} \quad C = A \cdot B = (c_{ij})_{\substack{i=1..n \\ j=1..q}}$$

for $i = 1..n$, for $j=1..q$:

$$c_{ij} = \sum_{k=1}^m a_{ik} b_{kj} = \begin{pmatrix} * & & * \\ \mathbf{a}_{i1} & \dots & \mathbf{a}_{im} \\ * & & * \end{pmatrix} \cdot \begin{pmatrix} * & \mathbf{b}_{1j} & * \\ & \vdots & \\ * & \mathbf{b}_{mj} & * \end{pmatrix} = \begin{pmatrix} * & & * \\ & \mathbf{c}_{ij} & \\ * & & * \end{pmatrix}$$

Algorithm 1 (ijk) - form:

```

for i = 1:n
  for k = 1:m
    for k = 1:m
       $c_{ij} = c_{ij} + a_{ik} b_{kj}$ 
    end
  end
end
end

```

} DOT-product
} $c_{ij} = A_{i \cdot} \cdot B_{\cdot j}$

All entries c_{ij} are fully computed, one after another. Access to A rowwise, to B columnwise

Algorithm 2 (jkj) - form

```

for j = 1:q
  j = 1:q
  for i = 1:n
     $c_{ij} = c_{ij} + a_{ik} b_{kj}$ 
  end
end
end

```

} SAXPY
} $c_j = c_j + a_{\cdot k} \cdot b_{kj}$
} vector c_j

} GAXPY
} $c_{\cdot j} = \sum_k b_{kj} a_{\cdot k}$

c computed columnwise; access to A columnwise

Algorithm 3 (kji) - form

```

for k = 1:m
  for j = 1:q
    for i = 1:n
       $c_{ij} = c_{ij} + a_{ik}b_{kj}$ 
    end
  end
end
end

```

} SAXPY } NO GAXPY
because different $c_{.j}$

There are computed intermediate values $c_{ij}^{(k)}$. Access to A columnwise.

	ijk	ikj	kij	jik	jki	kji
Access to A	row	—	—	row	column	column
Access to B	column	row	row	column	—	—
Computation of c_{ij}	row	row	row	column	column	column
vector operation with vector length	direct	delayed	delayed	direct	delayed	delayed
	DOT	GAXPY	SAXPY	DOT	GAXPY	SAXPY
	m	q	q	m	m	m

usually GAXPY better;
longer vector length better;
choose the right access to A,B, depending on the storage.

Matrix-Matrix-product in parallel

$$\langle 1, n \rangle = \bigcup_{r=1}^R I_r$$

$$\langle 1, m \rangle = \bigcup_{s=1}^S K_s$$

$$\langle 1, q \rangle = \bigcup_{t=1}^T J_t$$

Distribute blocks relative to index sets I_r, K_s, J_t to processor P_{rst} :

$$I_r \left(\begin{array}{c|c|c} & K_s & \\ \hline & & \\ \hline & A_{rs} & \\ \hline & & \end{array} \right) \cdot \left(\begin{array}{c|c|c} & J_t & \\ \hline & B_{ts} & \\ \hline & & \end{array} \right) k_s = \left(\begin{array}{c|c|c} & J_t & \\ \hline & & \\ \hline & c_{rt}^{(s)} & \\ \hline & & \end{array} \right) J_r$$

1. process P_{rst} : $c_{rt}^{(s)} = A_{rs} \cdot B_{st}$

small matrix-matrix-product
all processors independently

2. sum:

$$c_{rt} = \sum_{s=1}^S c_{rt}^{(s)} \quad \text{fan-in in } S$$

Special case $S = 1$:

$$I_r \left(\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \end{array} \right) \cdot \left(\begin{array}{c} J_t \\ | \\ | \\ | \end{array} \right) = \left(\begin{array}{c|c|c} & & \\ \text{---} & & \\ & c_{rt} & \\ \text{---} & & \end{array} \right) I_r$$

Each process can compute one block of c independently without communication.

Each process needs full block of rows of $A (\sim I_r)$ and block of columns of $B (\sim J_t)$, to compute the block c_{rt} .

with $m \cdot q$ processor: each processor has to compute one DOT-product

$$c_{rt} = \sum_k a_{rk} b_{kt} \quad \text{in } O(m)$$

If we use more processors to compute all these DOT-products by fan-in, we can reduce the parallel complexity to $O(\log m)$.

3 Linear Equations with dense matrices

3.1 Gaussian Elimination: Basic facts

Linear equations

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + \dots + a_{nn}x_n &= b_n \end{aligned}$$

$$\rightarrow \underbrace{\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}}_{Ax = b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \quad A = A^{(1)}$$

Solving of triangular equations is easy, so we try, to transform the given system in triangular form:

$$\begin{array}{l}
(1) \\
(2) \\
\vdots \\
(n)
\end{array}
\begin{pmatrix}
a_{11} & a_{12} & \dots & a_{1n} \\
a_{21} & a_{22} & \dots & a_{2n} \\
\vdots & & & \\
a_{n1} & & & a_{nn}
\end{pmatrix}
\begin{array}{l}
(2) \rightarrow (2) - \frac{a_{21}}{a_{11}} \cdot (1) \\
(n) \rightarrow (n) - \frac{a_{n1}}{a_{11}} \cdot (1)
\end{array}$$

$$\rightarrow = A^{(2)} = \begin{pmatrix}
a_{11}^{(2)} & a_{12}^{(2)} & \dots & a_{1n}^{(2)} \\
0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\
\vdots & a_{32}^{(2)} & & \\
\vdots & & & \\
0 & & & a_{nn} a_{nn}^{(2)}
\end{pmatrix}
\begin{array}{l}
(3) \rightarrow (3) - \frac{a_{32}}{a_{22}} \cdot (2) \\
(n) \rightarrow (n) - \frac{a_{n2}}{a_{22}} \cdot (1)
\end{array}$$

$$\rightarrow = A^{(3)} = \begin{pmatrix}
a_{11}^{(3)} & a_{12}^{(3)} & \dots & a_{1n}^{(3)} \\
0 & a_{22}^{(3)} & \dots & a_{2n}^{(2)} \\
\vdots & 0 & a_{32}^{(3)} & \\
\vdots & & & \\
0 & 0 & a_{n3}^{(3)} \dots & a_{nn}^{(3)}
\end{pmatrix}
\rightarrow \dots \rightarrow A^{(n)} = \begin{pmatrix}
a_{11} & \dots & & a_{1n} \\
0 & \ddots & & \\
\vdots & & & \\
& \ddots & a_{n-1,n-1} & a_{n-1,n} \\
0 & & 0 & a_{nn}
\end{pmatrix}$$

upper triangular form

Here we do no pivoting (we assume $a_{kk}^{(k)} \neq 0$ for all k)

we ignore the right hand side b .

Algorithm:

```

for k = 1:n-1
  for i = k+1:n
     $l_{ik} = \frac{a_{ik}}{a_{kk}}$ 
  end
  for i = k+1:n
    for j = k+1:n
       $a_{ij} = a_{ij} - l_{ik} \cdot a_{kj}$ 
    end
  end
end
end
end

```

Intermediate System $A^{(u)} = \begin{pmatrix} A_{11}^{(u)} & & & \\ 0 & \ddots & & \\ & & 0 & A_{kk}^{(u)} & A_{kn}^{(u)} \\ & & & \vdots & \\ 0 & 0 & A_{nk}^{(u)} & A_{nn}^{(u)} \end{pmatrix}$

Define matrix with entries l_{ik} from above algorithm.

$$L = \begin{pmatrix} 1 & 0 & & 0 \\ l_{12} & 1 & \ddots & \\ \vdots & \ddots & \ddots & \ddots \\ \vdots & & \ddots & \ddots & 0 \\ l_{11} & & & l_{11} & 1 \end{pmatrix} \text{ and}$$

$$L_k = \begin{pmatrix} 0 & & & & \\ & \ddots & & & \\ 0 & & l_{k+1,k} & \dots & \\ & & l_{n,k} & 0 & \dots & 0 \end{pmatrix}$$

\uparrow
 k-th column

Each Elimination step in Gaussian Elimination can be written in the form

$$A^{(k+1)} = (1 - L_k) \cdot A^{(n)} = A^{(n)} - L_k A^{(n)} \quad (3)$$

with $A^{(n)} = A$ and $A^{(1)} = U =$ upper triangular

$$U = A^{(n)} = (1 - l_{n-1})A^{(n-1)} \dots = (n-1) = \underbrace{(1 - l_{n-1}) \dots (1 - l_1)}_{\tilde{L}} A^{(1)} = \tilde{L} \cdot A$$

with $\tilde{L} := (1 - l_{n-1}) \dots (1 - l_1)$

$1 - l_j$ lower triangular $\Rightarrow \tilde{L}$ lower triangular $\Rightarrow \tilde{L}^{-1}$ lower triangular

$\Rightarrow A = \tilde{L}^{-1}U$ with \tilde{L}^{-1} lower and U upper triangular.

Theorem 2 $\tilde{L}^{-1} = L$

Proof:

$$i \leq j \Rightarrow 0 = l_i \cdot l_j = \begin{pmatrix} 0 & & & & \\ & \ddots & & & \\ & & 0 & & \\ & & * & & \\ & & \vdots & & \\ & & * & & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & & & & \\ & \ddots & & & \\ & & 0 & & \\ & & * & & \\ & & \vdots & & \\ & & * & & 0 \end{pmatrix}$$

Therefore $(1 + l_j)(1 - l_j) = 1 + l_j - l_j - l_j^2 = I$ and $(1 - l_j)^{-1} = 1 + l_j$

$$\begin{aligned} \Rightarrow \tilde{L}^{-1} [(1 - l_{n-1}) \dots (1 - l_1)]^{-1} &= (1 - l_1)^{-1} \dots (1 - l_{n-1})^{-1} \\ &= (1 - l_1)(1 - l_2) \dots (1 - l_{n-1}) = 1 + l_1 + l_2 + \dots + l_{n-1} = L \end{aligned}$$

because e.g. $(1 - l_1) \cdot (1 - l_2) = 1 + l_1 + l_2 + \underbrace{l_1 l_2}_{=0} = 1 + l_1 + l_2$

Total: $A = L \cdot U$ with L and U out???

insert diag!!

of Gaussian Elimination algorithm, triangular.

3.2 Vectorization of the Gaussian Elimination

(*kij*)-form (standard)

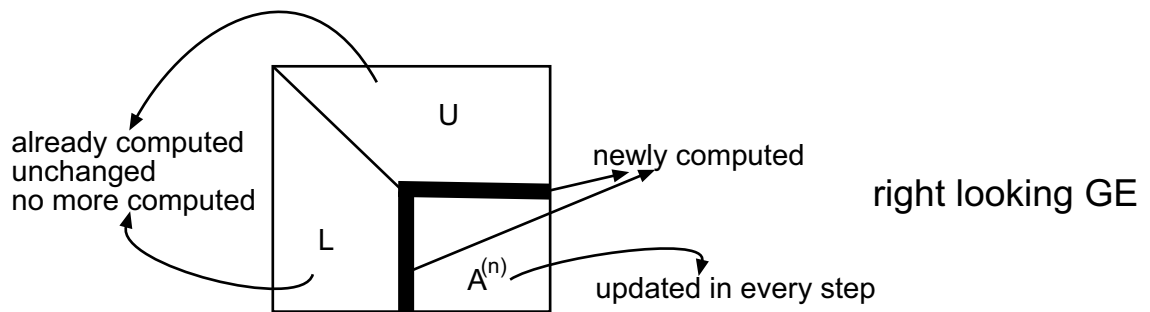
```

for k = 1:n-1
  for i = k+1:n
     $l_{i,k} = \frac{a_{ik}}{a_{kk}}$ 
  end
  for i = k+1:n
    for j = k+1:n
       $a_{ij} = a_{ij} - l_{ik} a_{kj}$ 
    end
  end
end
end

```

$\left. \begin{array}{l} \text{Vector operation} \\ \alpha \cdot \vec{x} \end{array} \right\}$
 $\left. \begin{array}{l} \text{SAXPY in} \\ \text{row } a_i. \text{ and } a_k. \end{array} \right\}$ No GAXPY

U computed rowwise, columnwise



In the following, we want to interchange the kij -loops:

Necessary condition: $1 \leq k < i \leq n$
 $1 \leq k < j \leq n$

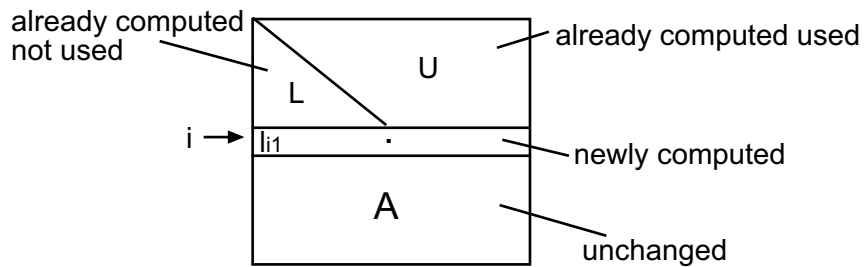
(ikj) -form:

```

for i = 2:n
  for k = 1:i-1
     $l_{ik} = \frac{a_{ik}}{a_{kk}}$ 
    j = k+1:n
     $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
  end
end
end
    
```

} GAXPY in a_{ii}

compute l_{i1} by SAXPY
 combine the 1st row and the i -th row
 then compute l_{i2}, \dots and so on.



L and U are computed rowwise.

(ijk) -form:

```

for i = 2:n
  for j = 2:i
     $l_{i,j-1} = \frac{a_{i,j-1}}{a_{j-1,j-1}}$ 
    for k = 1:j-1
       $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
    end
  } DOT
  end
  for j = i+1:n
    for k = 1:i-1
       $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
    end
  } DOT
  end
end

```

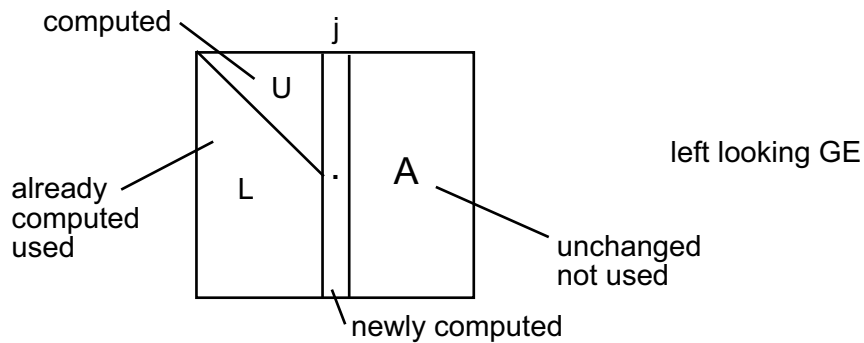
Compute $l_{i,1}$ and update $a_{i,2}$; then compute $l_{i,2}$ and update $a_{i,2}$ and $a_{i,3}$ and so on.

(*jki*)-form

```

for j = 2:n
  for k = j:n
     $l_{k,j-1} = \frac{a_{k,j-1}}{a_{j-1,j-1}}$ 
  }  $\alpha \cdot \vec{x}$ 
  end
  for k = 1:j-1
     $a_{ij} = a_{ij} - l_{ik}a_{kj}$ 
  } GAXPY in  $a_{ij}$ 
  end
end

```



	kij	kji	ikj	ijk	jki	jik
Access to ???	row	column	row	column	column	column
Access to L	—	column	—	row	column	row
Computation of U	row	row	row	row	column	column
Computation of L	column	column	row	row	column	column
Vector Operation	SAXPY	SAXPY	GAXPY	DOT	GAXPY	DOT
Vector Length	$\frac{2}{3}n$	$\frac{2}{3}n$	$\frac{2}{3}n$	$\frac{1}{3}n$	$\frac{2}{3}n$	$\frac{1}{3}n$

Vector length = average of occurring vector lengths

3.3 Gaussian Elimination in Parallel: Blockwise GE

$$\begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{21} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{13} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11}u_{11} & l_{11}u_{12} & l_{11}u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + l_{22}u_{22} & l_{21}u_{13} + l_{22}u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & * \end{pmatrix}$$

Different ways of computing L and U, depending on ordering:

Idea: one part of L/U already computed

second part is to be computed

third part not considered

3.3.1 right looking blockwise GE:

$$\begin{pmatrix} \mathbf{l}_{11} & 0 & 0 \\ \mathbf{l}_{21} & l_{22} & 0 \\ \mathbf{l}_{31} & l_{32} & * \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u}_{11} & \mathbf{u}_{12} & \mathbf{u}_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & * \end{pmatrix}$$

In **Bold**: already computed

In *italics*: has to be computed in this step

$$\left(\begin{array}{c|c} l_{22}u_{22} & l_{22}u_{23} \\ \hline l_{32}u_{22} & * \end{array} \right) = \left(\begin{array}{c|c} A_{22} - l_{21}u_{12} & A_{23} - l_{21}u_{13} \\ \hline A_{32} - l_{31}u_{12} & * \end{array} \right) = \begin{pmatrix} \hat{A}_{22} & \hat{A}_{23} \\ \hat{A}_{32} & * \end{pmatrix}$$

(1) $\begin{pmatrix} l_{22} \\ l_{32} \end{pmatrix} \cdot U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$ by small LU-decomposition gives $l_{22}, l_{32},$ and U_{22}

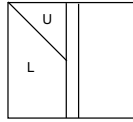
(2) $l_{22}U_{23} = \hat{A}_{23} \Rightarrow U_{23}$ by solving triangular system in l_{22}

in total: $\begin{pmatrix} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \cdot \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{pmatrix} = A$

Put the computed parts in the first row/column blocks of L and U

Split l_{33} and U_{33} in new parts $l_{22}, l_{32}, U_{22}, U_{23}$ and repeat.

3.3.2 left looking blockwise GE:



$$\begin{pmatrix} \mathbf{l}_{11} & 0 & 0 \\ \mathbf{l}_{21} & l_{22} & 0 \\ \mathbf{l}_{31} & l_{32} & l_{33} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u}_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ & & u_{23} \end{pmatrix} = A$$

In **Bold**: already computed

In *italics*: has to be computed in this step

equations: $l_{11}U_{12} = A_{12}$ can be solved by triangular gives U_{12}

Compute $\begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix} = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - \begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} U_{12}$ by matrix multiplication

and $\begin{pmatrix} l_{22} \\ l_{32} \end{pmatrix} \cdot U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$

3.3.3 Grout form

$$\begin{pmatrix} \mathbf{l}_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u}_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ & & u_{23} \end{pmatrix} = A$$

In **Bold**: already computed

In *italics*: has to be computed in this step

Equations:

$l_{21}U_{11} = A_{21} \Rightarrow l_{21}$ by solving $U_{11}^T l_{21}^T = A_{21}^T$ triangular solve

$l_{11}U_{12} = A_{12} \Rightarrow U_{12}$ by solving $U_{11}^T l_{21}^T = A_{21}^T$ triangular solve

$l_{22}U_{22} = A_{22} - l_{21}U_{12} = \hat{A}_{22}$ by LU-decomposition of \hat{A}_{22}

In comparison, all variants have nearly the same efficiency in parallel. 3.3.1 and 3.3.3 are slightly better, because more flops in Matrix-Matrix-Multiplication, less triangular solve and less small LU-decomposition.

3.4 QR-Decomposition with Householder matrices

3.4.1 QR-decomposition

Similar to LU-decomposition by Gaussian-Elimination. We are interested in $A = QR$ with Q orthogonal and R upper triangular
 $b = Ax = QRx \Leftrightarrow Rx = Q^T b$ for solving linear system. QR has advantages for ill-conditioned A .

Application for overdetermined systems $\boxed{\mathbf{A}} \cdot \boxed{\mathbf{x}} \stackrel{!}{=} \boxed{\mathbf{b}}$

$Ax = b$ has no solution. best approximate solution by solving $\mathbf{Min}_x \|Ax - b\|_2^2 = \mathbf{min}_x (x^T A^T Ax - 2x^T A^T b + b^T b)$
 gradient equal zero leads to $\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b}$ normal equation.
 $A^T A$ has a larger condition number than A .

Advantages of QR-decomposition:

$$A = QR, \quad R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

$$A^T Ax = A^T b$$

$$\Leftrightarrow (QR)^T QRx = (QR)^T b$$

$$\Leftrightarrow R^T Rx = R^T \underbrace{Qb}_b$$

$$\Leftrightarrow \begin{pmatrix} R_1^T & 0 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x = \begin{pmatrix} R_1^T & 0 \end{pmatrix} \hat{b}$$

$$\Leftrightarrow R_1^T R_1 x = \begin{pmatrix} R_1^T & 0 \end{pmatrix} \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \end{pmatrix}$$

$$\Leftrightarrow R_1^T R_1 x = R_1^T \hat{b}_1$$

$$\Leftrightarrow \mathbf{R}_1 \mathbf{x} = \hat{\mathbf{b}}_1 \quad \text{instead of } A^T A = A^T b$$

3.4.2 Householder method

u vector $\in \mathbf{R}^n$ with length 1, $\|u\|_2 = 1$

$H := I - 2uu^T$ is called Householder matrix

H is orthogonal ($H^T H = I$):

$$H^T H = H^2 = (I - 2uu^T)(I - 2uu^T) = I - 2uu^T - 2uu^T + 4u \underbrace{u^T u}_1 u^T = I$$

First step: use H to transform the first column of A in upper triangular form:

$$H_1 \cdot A = (I - 2u_1 u_1^T)(a_1 | \dots) = (a_1 - 2(u_1^T a_1)u_1 | * \dots)^T = \left(\begin{array}{c|c} \alpha & \\ \hline 0 & ** \\ \vdots & \\ 0 & \end{array} \right) \text{ Hence}$$

we have to find u_1 of length 1 with

$$\mathbf{a}_1 - 2(\mathbf{u}_1^T \mathbf{a}_1) \mathbf{u}_1 \stackrel{!}{=} \alpha \mathbf{e}_1$$

$$H_1 \text{ is orthogonal, therefore } \|a_1\|_2 = \left\| \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right\| = |\alpha|$$

We can set $\alpha = \|a_1\|$

$$\text{Therefore } u_1 = \frac{a_1 - \|a_1\|_2 e_1}{2(u_1^T a_1)} = \frac{a_1 - \|a_1\|_2 e_1}{\|a_1 - \|a_1\|_2 e_1\|_2}$$

For this u_1 we get:

$$\begin{aligned} (1 - 2u_1 u_1^T) a_1 &= a_1 - 2 \frac{(a_1 - \|a_1\|_2 e_1)^T \cdot a_1}{\|a_1 - \|a_1\|_2 e_1\|} \cdot \frac{a_1 - \|a_1\|_2 e_1}{\|a_1 - \|a_1\|_2 e_1\|} \\ &= a_1 \cdot \frac{(\|a_1\|^2 + \|a_1\|^2 - 2\|a_1\| \|a_1\|_2) - 2\|a_1\|^2 + 2\|a_1\| \|a_1\|_2}{\|a_1 - \|a_1\|_2 e_1\|^2} + e_1 \|a_1\| \frac{2\|a_1\|^2 - 2\|a_1\| \|a_1\|_2}{\|a_1 - \|a_1\|_2 e_1\|^2} = \|a_1\| \cdot e_1 \end{aligned}$$

$$H_1 \cdot A_1 = (1 - 2u_1 u_1^T) \cdot A = \left(\begin{array}{c|ccc} \|a_1\| & * & \dots & * \\ \hline 0 & & & \\ \vdots & & & \\ 0 & & & A_2 \end{array} \right) \quad V_1 := u_1$$

Apply the same procedure on A_2 :

$$H_2 A_2 = (1 - 2u_2 u_2^T) \cdot A_2 = \left(\begin{array}{c|ccc} * & & & \\ \hline 0 & & & \\ \vdots & & & \\ 0 & & & A_3 \end{array} \right)$$

$(1 - 2u_2 u_2^T) \rightarrow$ dimension n-1

extend u_2 to vector of length n : $v_2 := \begin{pmatrix} 0 \\ u_2 \end{pmatrix}$,

$$\text{Hence } H_2 H_1 \cdot A = (1 - 2v_2 v_2^T)(1 - 2v_1 v_1^T) A = \left(\begin{array}{ccc|ccc} * & & & & & \\ 0 & * & & & & \\ & & 0 & & & \\ \vdots & & & & & \boxed{A_3} \\ 0 & 0 & & & & \end{array} \right)$$

Total: $\underbrace{H_{n-1} \cdots H_2}_{Q^T} \cdot H_1 \cdot A = R =$ upper triangular

$$A = QR \quad \text{with} \quad Q = (H_{n-1} \cdots H_2 H_1)^T = H_1 \cdot H - 2 \cdots H_{n-1}$$

3.4.3 Householder method in parallel

Idea: Compute $u_1 \dots u_k$, but application of $H_k \dots H_1 \cdot A$ in *blocked* form

Question: What is the structure of $H_k \dots H_i =: V$

Theorem 3 $H_k \dots H_i = (1 - 2v_k v_k^T) \dots (1 - 2v_i v_i^T) = I - (v_k \dots v_i) T_i \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \end{pmatrix}$

with T_i upper triangular matrix

Proof by Induction:

$$\begin{aligned} & [(1 - 2v_k v_k^T) \dots (1 - 2v_i v_i^T)] \cdot (1 - 2v_{i-1} v_{i-1}^T) \\ &= \left[I - (v_k \dots v_i) T_i \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \end{pmatrix} \right] \cdot (1 - 2v_{i-1} v_{i-1}^T) \\ &= I - (1 - 2v_i v_i^T) - (v_k \dots v_i) T_i \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \end{pmatrix} + 2(v_k \dots v_i) T_i \underbrace{\begin{pmatrix} v_k^T v_{i-1} \\ \vdots \\ v_i^T v_{i-1} \end{pmatrix}}_y v_{i-1}^T \\ &= I - (v_k \dots v_i v_{i-1}) \cdot \left(\begin{array}{c|c} T_i & -2y \\ \hline 0 & 2 \end{array} \right) \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \\ v_{i-1}^T \end{pmatrix} \end{aligned}$$

Computation of $H_k \dots H_i$. A clockwise as

$$(I - YTY^T) \cdot A = A - YT(Y^T A) = \left(\begin{array}{c|c} R & * \\ \hline 0 & A \end{array} \right)$$

with $y = (v_k \dots v_i)$

Repeat with \tilde{A} .

4 Linear Equations with sparse matrices

4.1 General properties of sparse matrices

Full nxn matrix: $\left. \begin{array}{l} O(n^2) \text{ storage} \\ O(n^3) \text{ solution} \end{array} \right\} \text{ too costly}$

Formulate the given problem such that the resulting linear system is sparse

$O(n)$ storage

$O(n)$ solution?

example: tridiagonal: most of the entries are zero

Example matrix:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

$n = 5, nnz = 12 = \text{number of nonzero entries}$

4.1.1 Storage in coordinate form

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Superfluous information: storage $3 * nnz(A)$

Computation of $C = A.b$

$C \equiv \emptyset :$

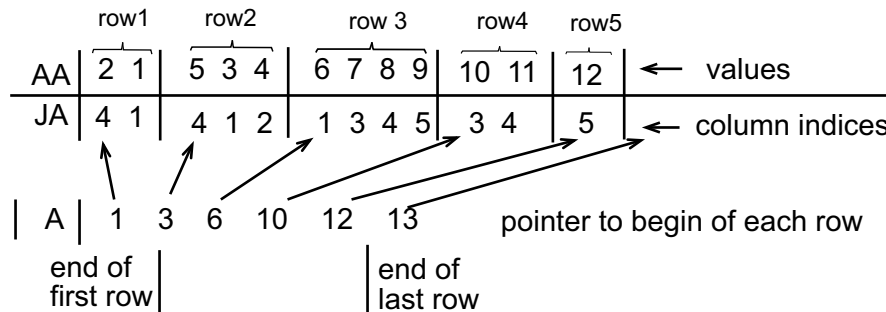
for $j = 1 : nnz(A)$

$$C_{JR(j)} = C_{JR(j)} + \underbrace{AA(j)}_{a_{JR(j),JC(j)}} . b_{JR(j)}$$

end

indirect addressing no c and b \rightarrow jumping in memory

4.1.2 Compressed Sparse Row Format: CSR



vector IA of length $n+1$

total storage: $2nnz(A) + n+1$

$C=A.b$:

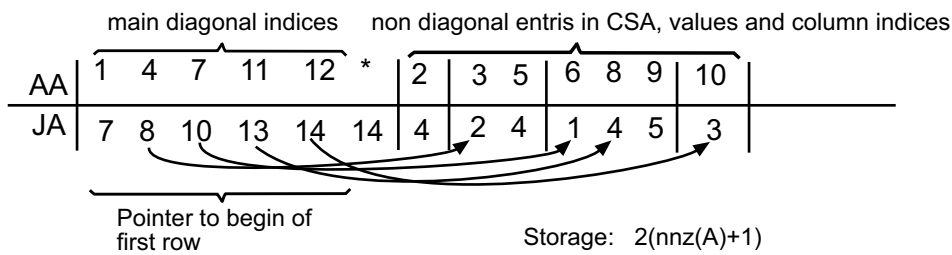
```

for i=1:n
  for IA(i) : IA(i + 1) - 1
    C(i) = C(i) + AA(j).b(JA(j))
  end
end

```

Similarly: CSC = Compressed Sparse Column format

4.1.3 Improving CSR by extracting the main diagonal entries often all $a_{ii} \neq 0$



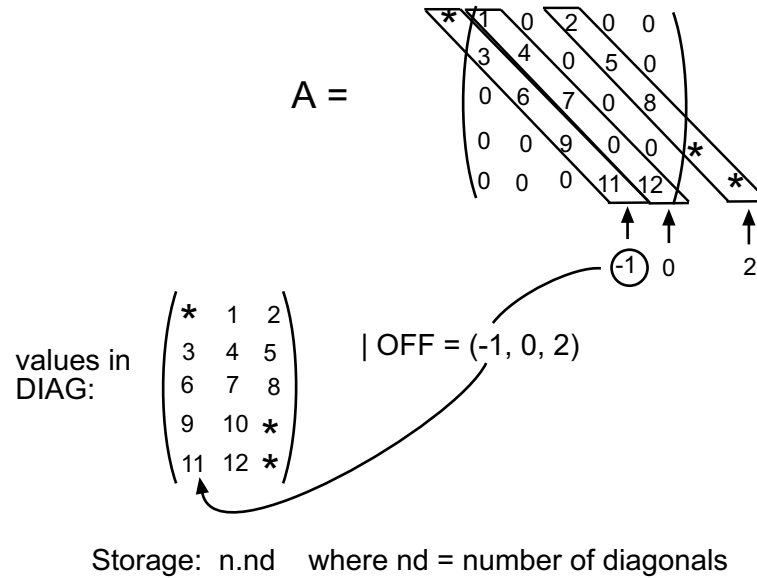
```

for i=1:n
  C(i) = AA(i)
  for JA(i) : JA(i + 1) - 1
    c(i) = c(i) + AA(j).b(JA(j))
  end
end

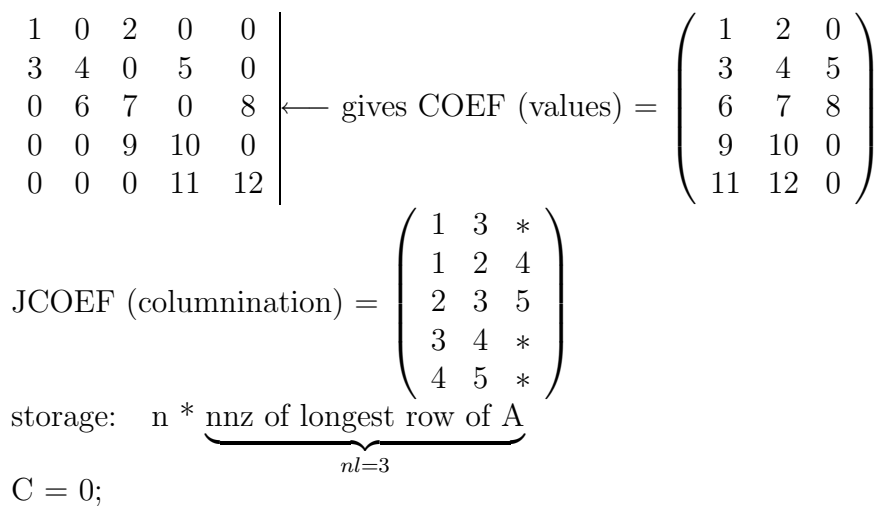
```

4.1.4 Diagonalwise storage, e.g. for band matrices

example:



4.1.5 rectangular, rowwise storage scheme by comparing from the right



```

for i=1:n
  for i=1:nl
    C(i) = C(i) + COEFF(i, j) * b(JCOEFF(i, j))
  end
end

```

Advantage: applicable also for non-diagonal structure

4.1.6 Jagged diagonal form

First step: Sort rows after their length:

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix} \longrightarrow P.A = \begin{pmatrix} 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix} \left. \begin{array}{l} \text{] Length 3} \\ \text{] Length 2} \end{array} \right\}$$

Storage for PA in the form:

$$\text{DJ values: } \underbrace{3 \ 6 \ 1 \ 9 \ 11}_{\text{first jaggeddiagonal}} \quad | \underbrace{4 \ 7 \ 2 \ 10 \ 12}_{\text{second}} | 5 \ 8 |$$

Column indices:

$$\begin{array}{l}
 \text{JDIAG: } 1 \ 2 \ 1 \ 3 \ | \ 4 \ 2 \ 3 \ 3 \ 4 \ 5 \ | \ 4 \ 5 \ | \\
 \text{IDIAG: } 1 \ 2 \ 3 \ 4 \ | \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ | \ 11 \ 12 \ | \ 13
 \end{array}$$

of length NDIAG

NDIAG = number of jagged diagonals + 1 = 4 in example
 storage: 2nnz(A) + (length of longest row) + 1

C = A.b

```

for i=1:NDIAG
  for i=1:INDIAG(j+1) - IDIAG(j)
    C(i) = C(i) + DJ( IDIAG(j) + i - 1).b(JDIAG(IDIAG(j) + i - 1))
    similar to SAXPY
  end
end

```

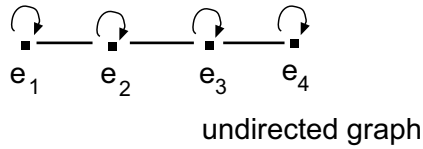
Indirect Addressing only in b

Operations on local block data!

4.2 Sparse Matrices and Graphs

4.2.1 $A = A^T$ (nxn - matrix) symmetric, define Graph $G(A)$:

Edges e_1, \dots, e_n
 vertices $K_{ij} = \begin{pmatrix} * & * & 0 & * \\ * & * & * & 0 \\ 0 & * & * & * \\ * & 0 & * & * \end{pmatrix}$
 $\rightarrow G(A)^2 =$

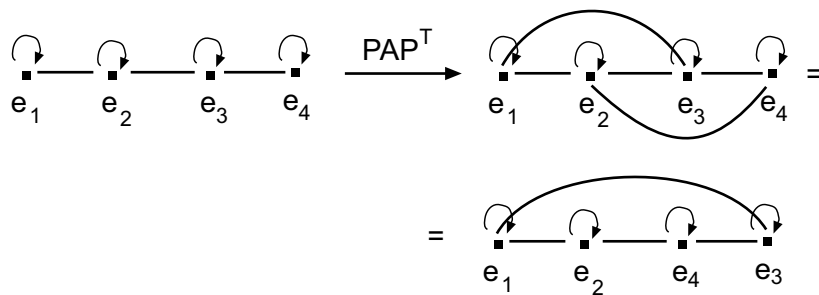


Graph $G(A)$ has adjacency matrix

$$A(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \text{ has exactly the structure of } A.$$

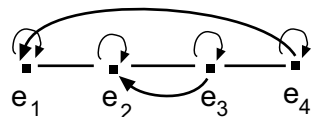
Symmetric permutation $P.A.P^T$, by permuting row and columns of A in the same way \Leftrightarrow changing the numbering in $G(A)$:

Symmetric permutation is only a change in the number of edges of $G(A)$.



4.2.2 A non symmetric: diagonal graph

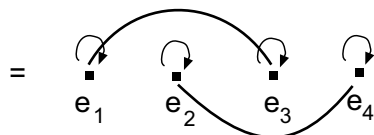
$$\begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix} \rightarrow$$



"good" sparsity pattern:

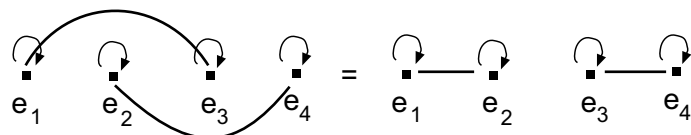
Block Diagonal:

$$\text{Example: } A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \rightarrow$$



graph splits into two subgraphs; use permutation that groups together edges in the same subgraph: $2 \leftrightarrow 3$

new $G(A)$:



$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2^{-1} \end{pmatrix} \text{ block diagonal}$$

$$\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}^{-1} = \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix}$$

Gaussian Elimination only on small blocks!

Band Matrix

$$A = \begin{pmatrix} a_{11} & \dots & a_{1p} & 0 & 0 \\ \vdots & \ddots & & \ddots & \vdots \\ a_{q1} & & \ddots & & \ddots & 0 \\ 0 & \ddots & & \ddots & & \vdots \\ \vdots & & \ddots & & \ddots & \vdots \\ 0 & 0 & & \dots & a_{nn} \end{pmatrix}$$

Gaussian Elimination without pivoting maintains the structure elimination only in $q \times p$ - block

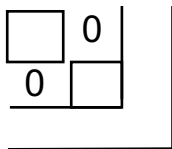
cols: $O(n \cdot pq)$, $A=L.U$ with

$$L = \begin{pmatrix} l_{11} & 0 & & & 0 \\ \vdots & \ddots & & & \\ l_{q1} & & \ddots & & \\ 0 & \ddots & & \ddots & \\ \vdots & & \ddots & & \ddots & 0 \\ 0 & 0 & & \dots & l_{nn} \end{pmatrix} \text{ and } U = \begin{pmatrix} u_{11} & \dots & u_{p1} & 0 & 0 \\ 0 & \ddots & & \ddots & \\ & & \ddots & & \ddots & 0 \\ & & & \ddots & & \vdots \\ 0 & & & 0 & u_{11} \end{pmatrix}$$

Similarly: $A = \begin{pmatrix} \square & & & \\ & \square & & \\ & & \square & \\ & & & \square \end{pmatrix}$ structure is preserved by GE

structure is preserved by GE

Dissection form preserved during GE



Excurs???: Schur Complement for Block Matrices

$$\begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \cdot \begin{pmatrix} B_1^{-1} & D \\ 0 & S^{-1} \end{pmatrix} = \begin{pmatrix} I & B_1 D + B_2 S^{-1} \\ B_3 B_1^{-1} & B_3 D + B_4 S^{-1} \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} I & 0 \\ * & I \end{pmatrix}$$

Therefore $B_1 D + B_2 S^{-1} \stackrel{!}{=} 0 \implies D = -B_1^{-1} B_2 S^{-1}$

and $B_3 D + B_4 S^{-1} \stackrel{!}{=} I \implies I = -B_3 B_1^{-1} B_2 S^{-1} + B_4 S^{-1}$

$$\implies I = (B_4 - B_3 B_1^{-1} B_2) S^{-1}$$

$\implies S = B_4 - B_3 B_1^{-1} B_2$ Schur Complement.

$$\begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B_3 B_1^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} B_1 & -D \\ * & S \end{pmatrix}$$

Instead of solving LE in B, we have to solve small systems in B_1 and S

Application in Dissection form:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

$$\stackrel{=}{=} \left(\begin{array}{cc|c} \square & 0 & \\ \hline 0 & \square & \\ \hline \square & \square & \square \end{array} \right)$$

Schur complement relative to $\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$:

$$\begin{aligned} S &= A_3 - (G_1 \ G_2) \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} \\ &= A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2 \end{aligned}$$

Use A_1^{-1}, A_2^{-1} to compute S

Linear Equation in Dissection form:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \implies \begin{aligned} A_1x_1 + F_1x_3 &= b_1 \\ A_2x_2 + F_2x_3 &= b_2 \\ G_1x_1 + G_2x_2 + A_3x_3 &= b_3 \end{aligned}$$

$$\implies \begin{aligned} x_1 &= A_1^{-1}b_1 - A_1^{-1}F_1x_3 \\ x_2 &= A_2^{-1}b_2 - A_2^{-1}F_2x_3 \end{aligned} \implies (G_1A_1^{-1}b_1 + G_1A_1^{-1}F_1x_3) + (G_2A_2^{-1}b_2 + G_2A_2^{-1}F_2x_3) + A_3x_3 = b_3$$

$$\implies (A_3 - G_1A_1^{-1}F_1 - G_2A_2^{-1}F_2)x_3 = b_3 - G_1A_1^{-1}b_1 - G_2A_2^{-1}F_2$$

$$\boxed{Sx_3 = \hat{b}_3}$$

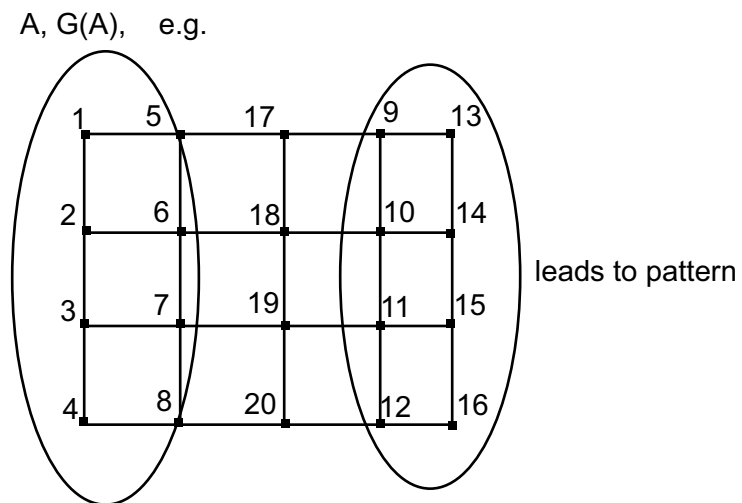
1. Compute S by using A_1^{-1} and A_2^{-1}
2. Solve $Sx_3 = \hat{b}_3$
3. Compute x_1 and x_2 by using A_1^{-1} and A_2^{-1}

Sometimes S is full or too expensive to compute. Then use iterative method for $Sx_3 = \hat{b}_3$, that uses only s*vector, which can be computed easily with F_1 , F_2 , G_1 , G_2 and A_1^{-1} , A_2^{-1} .

often reverse Cuthill Mckee-Algorithm better, numbering exactly in the opposite direction.

4.3.2 Dissection Reordering

A, $G(A)$, e.g.



leads to pattern

1 * * * 2 * * * 3 * * * 4 * * * 5 * * * * 6 * * * 7 * * * 8	0	* * *
0	9 * * * * 10 * * * * 11 * * * * 12 * * * * 13 * * * * 14 * * * * 15 * * * * 16	* * * *
* * * *	* * * *	17 18 19 20

$$= \begin{pmatrix} \boxed{A_1} & 0 & \boxed{F_1} \\ 0 & \boxed{A_2} & \boxed{F_2} \\ \boxed{G_1} & \boxed{G_2} & \boxed{A_3} \end{pmatrix}$$

4.3.3 Algebraic pivoting: reordering during GE

(Numerical pivoting: choose largest $a_{ij} \rightarrow a_{kk}$ as pivot element)

Algebraic pivoting: choose largest a_{ij} from sparse row—column $\rightarrow a_{kk}$ as pivot element

Aim: reduce fill-in during GE

Minimum degree ordering for $A = A^T > 0$, A spd.

first step: define $r_j = \#$ entries in row j

= vertices of edge j

choose i such that $r_i = \min_j r_j$

a_{ii} pivot element $i \leftrightarrow \wedge$ by permutation

Do one step elimination

reduce the matrix to row—column $2 \dots n$

Go back to the first step *
 leads to small fill-in in GE

Generalization to nonsymmetric matrices A : Markowitz-Criterion

deine $r - j = \#$ entries in row j

$c_k = \#$ entries in column k

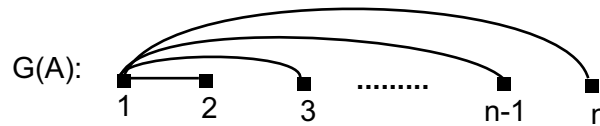
minimizes: $\min_{j,k} (N_j - 1)(c_k - 1)$ give indices i, s

$a_{i,s} \rightarrow a_{11}$ by permutation

Further generalization: include numerical pivoting

pivot $|a_{i,s}|$ should be not too small!

Example:
$$\begin{pmatrix} * & * & \dots & * \\ * & * & & 0 \\ \vdots & & \ddots & \\ 0 & & & * \\ * & & & * \end{pmatrix} \xrightarrow{GE} \begin{pmatrix} * & * & \dots & * \\ 0 & \boxed{\begin{matrix} * & * & \dots & * \\ & * & & * \\ \vdots & & \ddots & * \\ & * & * & * \end{matrix}} & & \\ \vdots & & & \\ \vdots & & & \\ 0 & & & \end{pmatrix} \text{ -- full}$$

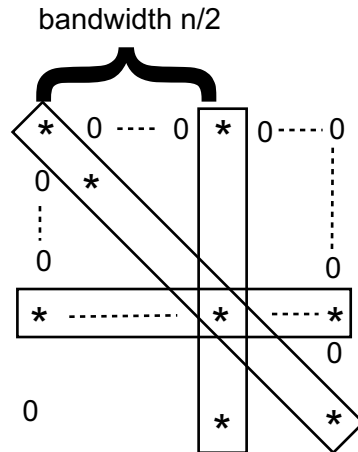


Cuthill-Mckee with starting edge $\{1\}$: $S_1 = \{1\}$, $S_1 = \{1, \dots, n\}$

given no improvement with start $\{2\}$: $S_1 = \{1\}$, $S_2 = \{1\}$, $S_3 = \{2, \dots, n\}$

given no improvement

Permutation such that smallest bandwidth also not very helpful.



Minimum Degree:

Choose edge 1 with degree $n-1$

Therefore replace 1 by 2 with degree 1

$$\rightarrow \begin{pmatrix} * & * & 0 & & 0 \\ * & * & \dots & \dots & * \\ 0 & & * & 0 & 0 \\ & & 0 & \ddots & \\ & & \vdots & & \ddots & 0 \\ 0 & * & 0 & & 0 & * \end{pmatrix} \xrightarrow{GE} \begin{pmatrix} * & * & \dots & & * \\ 0 & \boxed{\begin{matrix} * & * & \dots & * \\ * & * & 0 & 0 \\ & & 0 & \\ \vdots & & & \ddots & 0 \\ * & 0 & & 0 & * \end{matrix}} & & & \end{pmatrix}$$

Next pivot 3, and so on

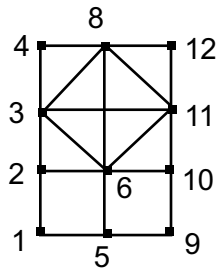
Works in $O(n)$

Global reordering:

$$\begin{pmatrix} * & * & 0 & & 0 \\ * & * & \dots & \dots & * \\ 0 & & * & 0 & 0 \\ & & 0 & \ddots & \\ & & \vdots & & \ddots & 0 \\ 0 & * & 0 & & 0 & * \end{pmatrix} \rightarrow \begin{pmatrix} * & & & & * \\ & * & & & \\ & & \ddots & & \\ & & & * & \\ \boxed{\begin{matrix} ** & & * \end{matrix}} & & & & * \end{pmatrix}$$

be permutation $1 \leftrightarrow n$

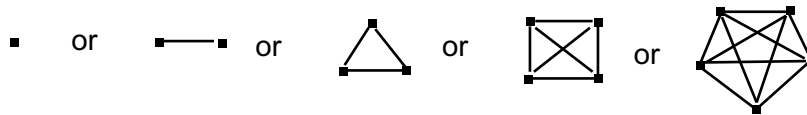
GE in $O(n)$



one step GE in the graph consists in

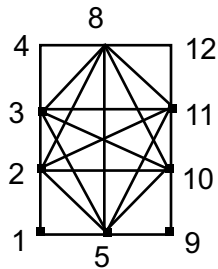
- remove edge 7
- add vertices such that all neighbours of 7 get fully connected

Definition: A fully connected graph is called 'clique'
e.g.

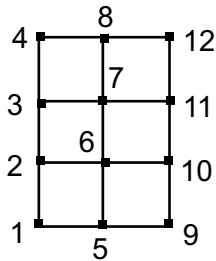


and so on

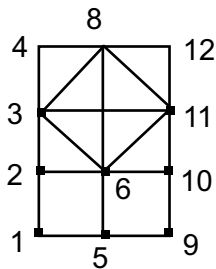
next step in GE: with pivot 6: neighbours: 2, 3, 5, 8, 10, 11



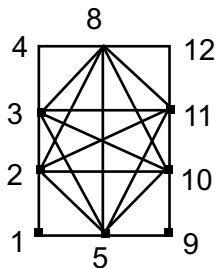
We can describe any graph by listing its clique subgraphs



: has only cliques of two edges.
clique representation = standard representation



remove 7 and add clique $\{3, 6, 8, 11\}$



remove 6 and add clique $\{2, 3, 5, 8, 10, 11\}$

Start: $\{1,2\}, \{1,5\}, \{2,6\}, \{3,4\}, \{3,7\}, \{4,8\}, \{5,6\}, \{5,9\}, \{6,7\}, \{6,10\}, \{7,8\}, \{7,11\}, \{8,12\}, \{9,10\}, \{10,11\}, \{11,12\}$

Elimination of 7: $\{1,2\}, \{1,5\}, \{2,3\}, \{2,6\}, \{3,4\}, \{4,8\}, \{5,6\}, \{5,9\}, \{6,10\}, \{8,12\}, \{9,10\}, \{10,11\}, \{11,12\}, \{3, 6, 8, 11\}$

Elimination of 6: $\{1,2\}, \{1,5\}, \{2,3\}, \{3,4\}, \{4,8\}, \{5,9\}, \{8,12\}, \{9,10\}, \{10,11\}, \{11,12\}, \{2, 3, 5, 8, 10, 11\}$

Elimination of 2: $\{1,5\}, \{3,4\}, \{4,8\}, \{5,9\}, \{8,12\}, \{9,10\}, \{10,11\}, \{11,12\}, \{1,3,5,8, 10, 11\}$

Elimination of 4: **{5,9}**, {8,12}, **{9,10}**, {11,12}, {1,3,5,8,10,11}

Elimination of 9: **{8,11}**, **{11,12}**, {1,3,5,8,10,11}

Elimination of 12: {1,3,5,8,10,11}

Elimination of 1: {3,5,8,10,11}

⋮

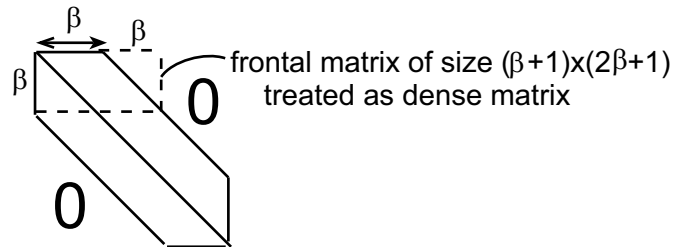
Gaussian elimination can be modelled without numerical computations only by computing the graphs algebraically.

Advantages:

- algebraic prestep is cheap
- gives information on the data structure of the resulting matrices
- shows how costly the numerical computations will be
- formulation in cliques, because in the course of GE, there will appear more and more cliques: cliques give short discretization???? of the graphs

4.5 Different Methods for solving sparse linear equations

4.5.1 Frontal methods for band matrices



frontal matrix of size $(\beta + 1) \times (\beta + 1)$ treated as dense matrix

- Apply first GE step with column pivoting in dense frontal matrix
- compute next row—column and move frontal matrix down right

Multifrontal method for general sparse matrices

example:

$$\begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 0 \\ * & * & 0 & * \end{pmatrix}$$

a_{11} first pivot element is related to first frontal matrix, that contain all numbers related to one step GE with

$a_{11} : \frac{a_{i1}a_{1j}}{a_{11}}$: in dense submatrix:

a_{11}	a_{13}	a_{14}
a_{31}	$\frac{a_{31}a_{13}}{a_{11}}$	$\frac{a_{31}a_{14}}{a_{11}}$
a_{41}	$\frac{a_{41}a_{13}}{a_{11}}$	$\frac{a_{41}a_{14}}{a_{11}}$

Because $a_{12} = 0$, we can in parallel consider a_{22} and the frontal matrix, related to the one step GE with a_{22} :

a_{22}	a_{23}	a_{24}
a_{32}	$\frac{a_{32}a_{23}}{a_{22}}$	$\frac{a_{32}a_{24}}{a_{22}}$
a_{42}	$\frac{a_{42}a_{23}}{a_{22}}$	$\frac{a_{42}a_{24}}{a_{22}}$

The computations $a_{ij} \rightarrow a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}$ and $a_{ij} \rightarrow a_{ij} - \frac{a_{i2}a_{2j}}{a_{22}}$

can be done independently using the frontal matrices related with a_{11} and a_{12}

\tilde{b}_7 to remove \tilde{c}_6

In this first step we have parallelism different blocks

Now we work parallel inside the blocks:

Result:

$$\left(\begin{array}{ccccccc} \boxed{\begin{array}{ccc} \hat{b}_1 & \hat{c}_1 & \hat{c}_1 \\ & \hat{b}_2 & \hat{c}_2 \\ & & \hat{b}_3 \end{array}} & & & & & & \\ & & & & \hat{c}_3 & & \\ & \hat{a}_4 & \boxed{\begin{array}{ccc} \hat{b}_4 & \hat{c}_4 & \hat{c}_4 \\ & \hat{b}_5 & \hat{c}_5 \\ & & \hat{b}_6 \end{array}} & & & & \\ & \hat{a}_5 & & & & & \hat{c}_6 \\ & \hat{a}_6 & & & & & \\ & & & & \hat{a}_7 & \boxed{\begin{array}{ccc} \hat{b}_7 & \hat{c}_7 & \hat{c}_7 \\ & \hat{b}_8 & \hat{c}_8 \\ & & \hat{b}_9 \end{array}} & \\ & & & & \hat{a}_8 & & \\ & & & & \hat{a}_9 & & \end{array} \right)$$

First step in parallel:

use \hat{b}_3 to remove $\hat{a}_4, \hat{a}_5, \hat{a}_6$

After this:

use \hat{b}_6 to remove $\hat{c}_7, \hat{c}_8, \hat{c}_9$

Gives:

$$\left(\begin{array}{ccccccc} \boxed{\begin{array}{ccc} b'_1 & c'_1 & c'_1 \\ & b'_2 & c'_2 \\ & & b'_3 \end{array}} & & & & & & \\ & & & & c'_3 & & \\ & & \boxed{\begin{array}{ccc} b'_4 & c'_4 & c'_4 \\ & b'_5 & c'_5 \\ & & b'_6 \end{array}} & & & & \\ & & & & & & c'_6 \\ & & & & & \boxed{\begin{array}{ccc} b'_7 & c'_7 & c'_7 \\ & b'_8 & c'_8 \\ & & b'_9 \end{array}} & \end{array} \right)$$

First step in parallel:

use b'_9 to remove c'_8, c'_7, c'_6

After this:

use b'_6 to remove c'_5, c'_4, c'_3

After this:

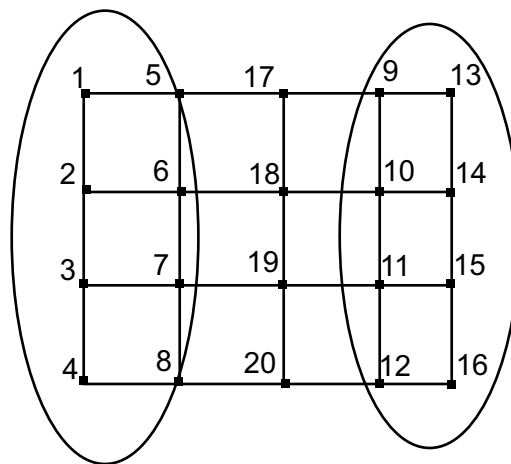
use b'_3 to remove c'_2, c'_1

Result: Diagonal system

block size and number of blocks should be nearly the same, because then parallel in block and between blocks with same efficiency.

4.5.3 Dissection method (compare Schur complement)

Example:



Cut the graph in two pieces:

Use memory of indices such that
first subgraph gets indices $1, \dots, k$
second subgraph gets $k + 1, \dots, m$
border gets $m + 1, \dots, n$

Pattern of matrix:

<pre> 1 * * * 2 * * * 3 * * * 4 * * 5 * * * * 6 * * * 7 * * * 8 </pre>	<p style="text-align: center;">0</p>	<pre> * * * * </pre>
<p style="text-align: center;">0</p>	<pre> 9 * * * 10 * * * 11 * * * 12 * * 13 * * * 14 * * * 15 * * * 16 </pre>	<pre> * * * * </pre>
<pre> * * * * </pre>	<pre> * * * * </pre>	<pre> 17 18 19 20 </pre>

$$= \begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & G_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

5 Iterative methods for sparse matrices

X_0 initial guess (e.g $X_0 = 0$)

Iteration function $\mathcal{O} : x_{k+1} = \mathcal{O}(x_k)$ gives sequence $|x_0, x_1, x_2, x_3, x_4, \dots|$

should converge $x_k \xrightarrow{k \rightarrow \infty} \bar{x} = A^{-1}b$

Computation of $\mathcal{O}(x)$ should use only $A \times$ vector (good parallizable)

Questions: Find $\mathcal{O}(x)$ such that we have **fast convergence** to $A^{-1}b$

5.1 stationary methods

5.1.1 Richardson Iteration for Solving $Ax = b : \bar{x} := A^{-1}b$

$$b = (A - I + I)x = (A - I)x + x \implies x = b + (I - A)x = b + Nx$$

Fix point iteration $x = \mathcal{O}(x)$ with $\mathcal{O}(x) = b + Nx$

$$x_0 \text{ start } x_{k+1} = \mathcal{O}(x_k) = b + Nx_k$$

if x_k convergent, $x_k \rightarrow \tilde{x}$, then $\tilde{x} = b + N\tilde{x} \Rightarrow A\tilde{x} = b \Rightarrow \hat{x} = \tilde{x}$

other formulation:

$$\mathcal{O}(x) = b + x - Ax = x + (b - Ax) = x + r(x) \quad r - \text{residual}$$

$$x_{k+1} = b + (I - A)x_k = x_k + r_k$$

Convergence analysis via "Neumann Series"

$$\begin{aligned} x_k &= b + Nx_{k-1} = b + N(b + Nx_{k-2}) = b + Nb + Nx_{k-2} \\ &= b + Nb + N^2b + Nx_{k-3} = \dots = b + Nb + \dots + N^{k-1}b + N^kx_0 \\ &= \left(\sum_{i=0}^{k-1} N^i\right)b + N^kx_0 \end{aligned}$$

$$\text{Special case: } x_0 = 0 : x_k = \left(\sum_{j=0}^{k-1} N^j\right)b$$

$$\Rightarrow x_k \in \text{span}(b, Nb, N^2b, \dots, N^{k-1}b) = \text{span}(b, Ab, A^2b, \dots, A^{k-1}b) = K_k(A, b)$$

= Krylov-row of dimension k to matrix A and vector b

Assume $\|N\| < 1$, then

$$\sum_0^{k-1} N^j \rightarrow \sum_0^{\infty} N^j = (1 - N)^{-1} = A^{-1}$$

is convergent

$$\Rightarrow x_k \rightarrow \left(\sum_0^{\infty} N^j\right)b = \bar{x}$$

Error: $e_k := x_k - \hat{x}$

$$e_{k+1} = x_{k+1} - \hat{x} = (b + Nx_k) - (b + N\hat{x}) = N(x_k - \hat{x}) = Ne_k$$

$$\|e_k\| \leq \|N\| \|e_{k-1}\| \leq \|N\|^2 \|e_{k-2}\| \leq \dots \leq \|N\|^k \|e_0\|$$

$\|N\| < 1 \Rightarrow \|N\|^k \rightarrow_{k \rightarrow \infty} 0 \Rightarrow \|e_k\| \rightarrow_{k \rightarrow \infty} 0$ and $x_k \rightarrow_{k \rightarrow \infty} \bar{x}$
 it is not very likely, that for a given A it holds $\rho(N) = \rho(I - A) < 1$

There fore we need better methods.

5.1.2 Better splitting of A

$$A := M - N$$

$$b = Ax = (M - N)x = Mx - Nx \Leftrightarrow x = M^{-1}b + M^{-1}Nx$$

$$\text{new } \mathcal{O}(x) = M^{-1}b + M^{-1}Nx = M^{-1}b + M^{-1}(M - A)x = M^{-1}(b - Ax) + x = x + M^{-1}r(x)$$

$$x_{k+1} = M^{-1}b + M^{-1}Nx_k = x_k + M^{-1}(b - Ax_k) = x_k + M^{-1}r_k$$

is equivalent to Richardson applied on $M^{-1}Ax = M^{-1}b$

Therefore convergent for $\rho(M^{-1}N) = \rho(I - M^{-1}A) < 1$

M should be easy to invert, and is also called a precondition, because $M^{-1}A$ should be better conditioned than A itself.

5.1.3 Jacobi (Diagonal) - Splitting:

$$A = M - N = D - (L + U)$$

with -

L lower triangular part of A

U upper triangular part of A

D diagonal part of A

$$= \begin{pmatrix} & & -U \\ & D & \\ -L & & \end{pmatrix}$$

$$x_{k+1} = D^{-1}b + D^{-1}(L + U)x_k = D^{-1}b + D^{-1}(D - A)x_k = x_k + D^{-1}r_k$$

convergent if $\rho(M^{-1}N) = \rho(I - D^{-1}A) < 1$

elementwise:

$$x_j^{k+1} = \frac{1}{a_{jj}} \left(b_j - \sum_{\substack{m=1 \\ m \neq j}}^n a_{jm} x_m^{(k)} \right)$$

or

$$a_{jj} x_j^{k+1} = b_j - \sum_{m=1}^n a_{jm} x_m^{(k)} - \sum_{m=j+1}^n a_{jm} x_m^{(k)}$$

Damping: $x_{k+1} = x_k + D^{-1} r_k$ $D^{-1} r_k$ - correction step or search direction

$x_{k+1} = x_k + \omega D^{-1} r_k$ with step length ω

$$\begin{aligned} x_{k+1} &= x_k + \omega D^{-1} r_k = x_k + \omega D^{-1} (b - Ax) = x_k + \omega D^{-1} b - \omega D^{-1} Ax_k = \\ &= (I - \omega D^{-1} A) x_k + \omega D^{-1} b \\ &= (I - \omega D^{-1} (D - L - U)) x_k + \omega D^{-1} b \\ &= \omega D^{-1} b + [(1 - \omega A) I + \omega D^{-1} (L + U)] x_k \end{aligned}$$

convergent if $\rho(\underbrace{[(1 - \omega) I + \omega D^{-1} (L + U)]}_{I \text{ for } \omega \rightarrow 0}) < 1$

Jacobi method is easy to parallelize: $x_{k+1} = x_k + \omega D^{-1} (b - Ax_k)$

variation: blockwise Jacobi by splitting blockwise

$$A = \begin{pmatrix} \boxed{} & & & & \\ & \boxed{} & & & \\ & & \boxed{} & & \\ & & & \boxed{} & \\ & & & & \boxed{} \end{pmatrix} \begin{matrix} \\ \\ -U \\ \\ \\ -L \\ \\ \\ D \end{matrix}$$

5.1.4 Gauss-Siedel method by improving convergence

$$a_{kk} x_j^{k+1} = b_j - \sum_{m=1}^{j-1} a_{jm} x_m^{(k+1)} - \sum_{m=j+1}^n a_{jm} x_m^{(k)}$$

If we want to compute $x_j^{(k+1)}$ in Jacobi method, we assume that $x_1^{(k+1)}, \dots, x_{j-1}^{(k+1)}$ have been computed already.

Then we can use the newest information by defining:

$$a_{kk}x_j^{k+1} = b_j - \sum_{m=1}^{j-1} a_{jm}x_m^{(k+1)} - \sum_{m=j+1}^n a_{jm}x_m^{(k)}$$

or $Dx_{k+1} = b + Lx_{k+1} + Ux_k$

or $(D - L)x_{k+1} = b + Ux_k$

is related to splitting $A = \underbrace{(D - L)}_M - \underbrace{U}_N$

$$\begin{aligned} x_{k+1} &= (D - L)^{-1}b + (D - L)^{-1}Ux_k \\ &= (D - L)^{-1}b + (D - L)^{-1}(D - L - A)x_k \\ &= x_k + (D - L)^{-1}r_k \end{aligned}$$

convergence depending on $\rho(1 - (D - L)^{-1}(D - L - A)) < 1$

$D - L$ lower triangular, easy to solve, but strongly sequential.

Hence better convergence, but not good parallelizable.

Damping: $x_{k+1} = x_k + (D - L)^{-1}r_k$

Stationary methods can be written in the form

$$\begin{aligned} x_{k+1} &= C + Bx_k \text{ with constant vector } C \text{ and iteration matrix } B \\ &= x_k + F.r_k \end{aligned}$$

5.2 Nonstationary Methods

5.2.1 Let A symmetric positive definite $A = A^T > 1$

Consider function $\mathcal{O}(x) = \{x^T Ax - b^T x\}$

Derivative: $\nabla \mathcal{O}(x) = Ax - b$ gradient \mathcal{O} Paraboloid

\bar{x} Minimum of \mathcal{O} is unique with $\nabla \mathcal{O}(\bar{x}) = A\bar{x} - b = 0$

Compute \bar{x} , solution of $Ax = b$ by approximating the minimum of \mathcal{O} iteratively

x_k last iterate: find???? $x_{k+1} = x_k + \lambda v$ with search direction v and stepsize λ , such that $\mathcal{O}(x_{k+1}) < \mathcal{O}(x_k)$ and hence x_{k+1} is nearer to minimum.

Search direction v :

$$\frac{d}{d\lambda} \mathcal{O}(x_k + \lambda v)|_{\lambda=0} = \nabla \mathcal{O}(x_k)$$

gradient goes in the direction of the largest increase of \mathcal{O}

Search direction $v = -\nabla \mathcal{O}(x_k) = b - Ax_k = r_k$

stepsize λ : finding $\min_{\lambda} \mathcal{O}(x_k + \lambda r_k)$ is a simple 1D-problem

$$\begin{aligned} \frac{d}{d\lambda} \mathcal{O}(x_k + \lambda r_k) &= \frac{d}{d\lambda} \left(\frac{1}{2} (x_k^T + \lambda r_k^T) A (x_k + \lambda r_k) - b^T (x_k + \lambda r_k) \right) \\ &= \frac{d}{d\lambda} \left(\frac{1}{2} x_k^T A x_k + \lambda r_k^T A x_k + \frac{\lambda^2}{2} r_k^T A r_k - b^T x_k - \lambda b^T r_k \right) \end{aligned}$$

$$\begin{aligned}
&= r_k^T A x_k - b^T r_k + \lambda r_k^T A r_k \\
&= -r_k^T r_k + \lambda = 0 \\
\Leftrightarrow \lambda &= \frac{r_k^T A r_k}{r_k^T r_k}
\end{aligned}$$

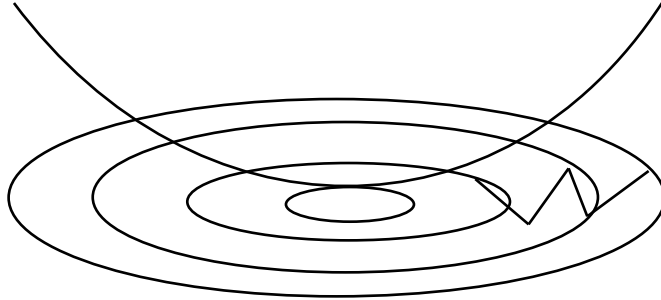
Algorithm: $x_{k+1} = x_k + \frac{r_k^T r_k}{r_k^T A r_k} r_k$ with $r_k = b - A x_k$

Gradient Method, steepest decent.

Disadvantage:

if condition(A) = $\|A\|_2 \|A^{-1}\|_2 = \frac{\lambda_{max}}{\lambda_{min}} \gg 1$

then the paraboloid is very distorted and the ????? are very small and ellipsoids.



The the convergence will be very slow: tick-tack

To analyse this slow convergence also theoretically, we introduce the following norm (so-called A-norm)

$$\|x\|_A := \sqrt{x^T A x}$$

Then it holds for the error $x - \bar{x}$ with $\bar{x} = A^{-1}b$

$$\|x - \bar{x}\|_A^2 = \|x - A^{-1}b\|_A^2 = (x^T - b^T A^{-1}) A (x - A^{-1}b)$$

$$x^T A x - 2b^T x + b^T A^{-1}b = 2\mathcal{O}(x) + b^T A^{-1}b$$

Hence minimizing \mathcal{O} is equivalent to minimizing the error in the A-norm.

with $x_{j+1} := x_j + \lambda_j r_j$, $r_j = b - A x_j$ and $\lambda_j = \frac{r_j^T r_j}{r_j^T A r_j}$

we get the following inequality between $\mathcal{O}(x_{j+1})$ and $\mathcal{O}(x_j)$:

$$\begin{aligned}
\mathcal{O}(x_{j+1}) &= \frac{1}{2}(x_j^T + \lambda_j r_j^T) A (x_j + \lambda_j r_j) - (x_j^T + \lambda_j r_j^T) b \\
&= \mathcal{O}(x_j) - \lambda_j (r_j^T (A r_j - b)) + \frac{\lambda_j^2}{2} r_j^T A r_j \\
&= \frac{1}{2} x_j^T A x_j + \lambda_j x_j^T A r_j + \frac{\lambda_j^2}{2} r_j^T A r_j - x_j^T b - \lambda_j r_j^T b
\end{aligned}$$

$$\begin{aligned}
&= \mathcal{O}(x_j) - \frac{r_j^T r_j}{r_j^T A r_j} \cdot r_j^T r_j + \frac{1}{2} \frac{(r_j^T r_j)^2}{(r_j^T A r_j)^2} \cdot r_j^T A r_j \\
&= \mathcal{O}(x_j) - \frac{1}{2} \frac{(r_j^T r_j)^2}{(r_j^T A r_j)^2} \\
&= \mathcal{O}(x_j) - \frac{1}{2} \frac{(r_j^T r_j)^2}{\underbrace{r_j^T A r_j \cdot r_j^T A^{-1} r_j}_{\rho_j}} \cdot r_j^T A^{-1} r_j \\
&= \mathcal{O}(x_j) - \frac{1}{2} \rho_j \cdot (b^T - a x_j)^T A^{-1} (b^T - a x_j) \\
&= \mathcal{O}(x_j) - \frac{\rho_j}{2} (b^T A^{-1} b + x_j^{-1} T A x_j - 2 b^T x_j) \\
&= \mathcal{O}(x_j) - \frac{1}{2} \rho_j \cdot (\mathcal{O}(x_j) + \frac{1}{2} b^T A^{-1} b) \\
\Rightarrow \mathcal{O}(x_{j+1}) + \frac{1}{2} b^T A^{-1} b &= \mathcal{O}(x_j) \frac{1}{2} b^T A^{-1} b - \rho_j (\mathcal{O}(x_j) + \frac{1}{2} b^T A^{-1} b) \\
\Rightarrow \|x_{j+1} - \bar{x}\|_A^2 &= \|x_j - A^{-1} b\|_A^2 (1 - \rho_j) \\
\rho_j = \frac{(r_j^T r_j)^2}{r_j^T A r_j r_j^T A^{-1} r_j} &\geq \frac{1}{\lambda_{max} \cdot \frac{1}{\lambda_{min}}} = \frac{1}{cond(A)} \\
\Rightarrow \|x_{j+1} - \bar{x}\|_A^2 &= (1 - \frac{1}{cond(A)}) \|x_{j+1} - \bar{x}\|_A^2
\end{aligned}$$

Is therefore $cond(A) \gg 1$, then the improvement is every iteration step ≈ 1

Therefore we have very slow convergence!

5.2.2 Improving the gradient method \rightarrow conjugate gradients

Ansatz: $x_{k+1} = x_k + \alpha_k \cdot p_k$

α_k stepsize

p_k search direction

As search direction we do not use the gradient, but a projection of the gradient.

The gradient direction is only locally a good search direction, but not globally!

Choose new search direction such that p_k is perpendicular to p_j for $j = k$

or p_k is perpendicular to $A p_j$ for $j < k \Leftrightarrow p_k^T A p_j = 0$ for $j < k$

the p_j are 'A-conjugate'!

We choose the new search direction as that part of residual that is conjugate to all previous search directions!

α_k by 1-dimensional minimization as before

Algorithm: $x_0 = 0, \quad r_0 = b - Ax_0$
for $U = 1, 2, \dots$: $\beta_{k-1} = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2} \quad \beta_0 = 0$
 $p_k = r_{k-1} + \beta_{k-1} p_{k-1} \quad (p_k \text{ A-cinjugate part of } r_{k-1})$
 $\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k \quad (1\text{-dimension min.})$
 $x_k = x_{k-1} + \alpha_k p_k$
 $r_k = r_{k-1} - \alpha_k A p_k$
 $r_k = r_{k-1} - \alpha_k A p_k$
if $\|r_k\| < \epsilon$: stop

Main properties of the computed vectors:

$$p_j^T A p_k = 0 = r_j^T r_k \quad \text{for } j \neq k$$

$$\text{span}(p_1, \dots, p_j) = \text{span}(r_0, \dots, r_{j-1}) = \text{span}(r_0, A r_0, \dots, A^{(j-1)} r_0) = K_j(A, r_0)$$

$$\text{especially for } x_0 = 0: \quad \text{span}(b, A b, \dots, A^{(j-1)} b)$$

x_k is the best approximate solution to $Ax = b$ in subspace $K_j(A, N_0)$:

$$\text{for } x_0 = 0: \quad x_k \in \text{span}(b, A b, \dots, A^{(j-1)} b) \quad \text{and} \quad \|x_k - \bar{x}\|_A = \min_{x \in K(A, b)} \|x - \bar{x}\|_A$$

Hence cheap 1-dimensional minimization gives at the same time optimal solution in k-dimensional space!

Consequence: after n steps $K_n(A, b) = \mathbf{R}_k$

$$\implies x_n = \bar{x} \quad \text{in exact arithmetic}$$

Unfortunately, this is only true in exact arithmetic.

Furthermore, mostly n steps is too costly!

error estimation for $x_0 = 0$:

$$\begin{aligned} \|e_k\|_A &= \|x_k - \bar{x}\|_k = \min_{x \in K_k(a, b)} \|x - \bar{x}\|_A = \min_{x_j} \left\| \sum_{j=0}^{k-1} (A^j b) - \bar{x} \right\|_A \\ &= \min_{p_{k-1}(x)} \|p_{k-1}(A) \cdot b - \bar{x}\|_A = \min_{p_{k-1}(x)} \|p_{k-1}(A) \cdot A \bar{x} - \bar{x}\|_A \\ &= \min_{p_{k-1}(x)} \|(p_{k-1}(A) \cdot A - 1)(\bar{x} - x_0)\|_A \\ &= \min_{\substack{q_k(x) \\ q_k(0)=1}} \|q_k(A) e_0\|_A \end{aligned}$$

for polynomial $q_k(x)$ of degree k with $q_k(0) = 1$

To A there exist n eigenvalues $u_j, j = 1, \dots, n$

with $A u_j = \lambda_j u_j$ with eigenvalues λ_j

u_j are an orthonormal basis

we can write:

$$\begin{aligned}
 l_0 &= \sum_{j=1}^n p_j u_j \\
 \Rightarrow \|e_k\|_A &= \min_{x \in q_k(0)=1} \|q_k(A) - \sum_1^n p_j u_j\|_A \\
 &= \min_{q_k(0)=1} \left\| \sum_1^n p_j q_k(A) u_j \right\|_A \\
 &= \min_{q_k(0)=1} \left\| \sum_1^n p_j q_k(\lambda_j) u_j \right\|_A \\
 &\leq \min_{q_k(0)=1} \max_1^n |q_k(\lambda_j)| \cdot \left\| \sum_1^n p_j u_j \right\|_A \\
 &= \min_{q_k(0)=1} \max_{j=1}^n |q_k(\lambda_j)| \cdot \|p_0\|_A
 \end{aligned}$$

by choosing polynomial with $q_k(0) = 1$ we can derive estimates for the error

e.g.: $q_k(x) := \left(1 - \frac{2}{\lambda_{max} + \lambda_{min}} x\right)^k$

leads to

$$\begin{aligned}
 \|e_k\|_A &\leq \max_{j=1}^n |q_k(\lambda_j)| \cdot \|l_0\|_A = \left(1 - \frac{2\lambda_{max}}{\lambda_{max} + \lambda_{min}}\right)^k \|e_0\|_A \\
 &= \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^k \|e_0\|_A = \left(\frac{cond(A)-1}{cond(A)+1}\right)^k \|e_0\|_A
 \end{aligned}$$

Better estimates by choosing normalized Chebychev polynomials:

$T_n(x) = cor(narc\ cor(x))$:?????

$$\|e_k\|_A \leq \frac{1}{T_k\left(\frac{cond(A)+1}{cond(A)-1}\right)} \leq 2 \left(\frac{\sqrt{cond(a)-1}}{\sqrt{cond(a)+1}}\right)^k$$

If the eigenvalues of A are clustered we can choose special polynomials, that give better estimates:

e.g. assume that A has only two eigenvalues λ_1 and λ_2

set $q_2(x) := \frac{(\lambda_1-x)(\lambda_2-x)}{\lambda_1\lambda_2} \Rightarrow q_2(0) = 1$

$\Rightarrow \|e_2\|_A \leq \max_{j=1,2} |q_2(\lambda_j)| \|e_0\|_A = 0$

\Rightarrow convergence of cg-method after 2 steps!

5.2.3 GMRES for General Matrix A

Consider small subspace U_M and determine optimal approximate solution for $Ax = b$ in U_m .

so we restrict x to the form $x = U_m \cdot y$

$$\min_{x \in U_m} \|Ax - b\|_2 = \min_y \|A(U_m y) - b\|_2$$

could be solved via normal equations $U_m^T A^T A U_m = U_m^T A^T b$

would be costly and we are interested in iterative enlargement

$U_m, \longrightarrow U_{m+1} \longrightarrow U_{m+2} \dots$

What subspace should we choose? (relative to $Ax=b$):

$$U_m := U_m(A, b) = \text{span}(b, Ab, \dots, A^{m-1}b) \quad \text{Krylor space}$$

Problem: b, Ab, A^2b is no good basis for U_m

First step: provide Orthonormal basis for U_m :

$$u_1 := b/\|b\|_2$$

$$\begin{aligned} & \text{for } j = 2 : m \\ \tilde{u}_j &:= Au_{j-1} - \sum_{u=1}^{j-1} \underbrace{(u_u^T Au_{j-1})}_{h_{k,j-1}} \cdot u_u ; \\ u_j &:= \tilde{u}_j / \underbrace{\|\tilde{u}_j\|_2}_{h_{j,j-1}} \\ & \text{end} \end{aligned}$$

standard orthogonal code

$$u_m = \text{span}(b, \dots, A^{m-1}b) = \text{span}(u_1, u_2, \dots, u_m) \quad \text{ONB}$$

We can compute this orthogonalization method in compact matrix form:

$$Au_{j-1} = \sum_{u=1}^{j-1} u_{u,j-1} u_u + \tilde{u}_j = \sum_{j=1}^k u_{k,j-1} \cdot u_k$$

$$AU_m = A(u_1, \dots, u_m) = (u_1, \dots, u_m) \tilde{H}_{m+1,m} = \tilde{U}_m \cdot H_{m+1,m}$$

$$\text{with } \tilde{H}_{m+1,m} = \begin{pmatrix} h_{11} & \dots & \dots & h_{1m} \\ h_{21} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \\ \vdots & \ddots & \ddots & h_{m,m} \\ 0 & \dots & 0 & h_{m+1,m} \end{pmatrix} \quad \text{Upper } m+1 \times m \text{ Hessenberg form}$$

Now we can solve the minimization problem:

$$\begin{aligned} \min_{x \in U_m} \|Ax - b\|_2 &= \min_y \|A(U_m y) - b\|_2 \\ &= \min_y \|U_m \tilde{H}_{(m+1,m)} y - \|b\| \cdot u_1\|_2 \\ &= \min_y \|U_m (\tilde{H}_{(m+1,m)} y - \|b\| \cdot e_1)\|_2 \\ &= \min_y \|\tilde{H}_{(m+1,m)} y - \|b\| \cdot e_1\|_2 \end{aligned}$$

because U_m is part of an orthogonal matrix.

We can use Givens rotation to compute a QR-decomposition of the upper

Hessenberg matrix $\tilde{H}_{(m+1,m)}$

$$G_1 \begin{pmatrix} * & & & * \\ * & \ddots & & \\ & \ddots & & \\ & & * & * \end{pmatrix}, \quad G_2 \begin{pmatrix} * & * & & * \\ 0 & * & & \\ & * & \ddots & \\ & & & * & * \\ & & & & * \end{pmatrix}, \dots, G_m \begin{pmatrix} * & & & \\ 0 & * & & \\ & & \ddots & \\ & & & * \\ & & & & 0 & * \end{pmatrix}$$

$$\text{gives } Q \cdot \tilde{H}_{(m+1,m)} = G_m \dots G_2 G_1 \cdot \tilde{H}_{(m+1,m)} = R = \begin{pmatrix} \dots & \\ \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix} = \begin{pmatrix} R_m \\ 0 \end{pmatrix}$$

$$\begin{aligned} \min_{x \in U_m} \|Ax - b\|_2 &= \min_y \|\tilde{H}_{(m+1,m)}y - \|b\| \cdot e_1\|_2 \\ &= \min_y \|(Q^T R)y - \|b\| \cdot e_1\|_2 \\ &= \min_y \|Ry - \|b\| Q^T e_1\|_2 = \min_y \left\| \begin{pmatrix} R_m y \\ 0 \end{pmatrix} - \|\hat{b}\| \right\|_2 \end{aligned}$$

Solution: $R_m y = \tilde{b}_m$ and $x = U_m y$

GMRES is a clever implementation of this minimization

- Compute $\tilde{H}_{(m+1,m)}$ by orthogonalization
- compute QR-decomposition of Upper Hessenberg by Givens rotations
- solve triangular systems for $y \rightarrow x$

Iterative: enlarge $U_m \rightarrow$ new $A^m b \rightarrow$ new column in $\tilde{H}_{(m+1,m)}$
 \rightarrow new Givens to update QR \rightarrow new column in $R_m \rightarrow$ new y
 \rightarrow new x

then continue the iteration

error estimation:

$$\begin{aligned} \|r_m\|_2 &:= \|Ax_m - b\|_2 = \min_{x \in U_m} \|Ax - b\|_2 \\ &= \min_{x_j} \|A(\sum_{j=0}^{m-1} \alpha_j A^j b) - b\|_2 = \min_{p_{m-1}} \|Ap_{m-1}(A)b - b\|_2 \\ &= \min_{q_m(0)} \|q_m(A) \cdot b\|_2 \\ &= \min_{q_m(0)} \|q_m(A) \sum_{k=1}^n p_k u_k\|_2 \\ &= \min_{q_m(0)} \left\| \sum_{k=1}^n q_m(\alpha_k) u_k \right\|_2 \\ &\leq \min_{q_m(0)} \max_{k=1}^n |q_m(\alpha_k)| \cdot \left\| \sum_{k=1}^n p_k u_k \right\|_2 \end{aligned}$$

$$= \min_{q_m(0)} \max_{k=1}^n |q_m(\alpha_k)| \cdot \|r_0\|_2 \quad \text{compare cg}$$

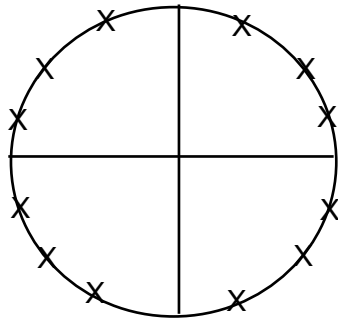
If A has eigenvectors u_1, u_2, \dots, u_n

5.3 Preconditioning

Result from section 1:

Iterative methods are easy to parallelize (only A^* vector), but to get fast convergence, the eigenvalues should be clustered.

bad example: $A = \begin{pmatrix} 0 & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & & 1 \end{pmatrix}$ wit complex eigenvalues



GMRES takes n steps!

$$M^{-1}Ax = M^{-1}b$$

Idea: replace $Ax = b$ by

$$\text{or} \\ PAx = Pb$$

$$M \approx A (\rightsquigarrow M^{-1} \approx A^{-1}) \text{ still } M^{-1}$$

with precondition

$$\text{or} \\ P \approx A^{-1} \quad (PA \approx 1)$$

stationary methods: $A = M - N$

$$b = Ax = (M - N)x = Mx - Nx \longrightarrow x_{k+1} = M^{-1}b + M^{-1}Nx_k \\ x_{k+1} = M^{-1}b + M^{-1}(M - N)x_k = x_k + M^{-1}Ax_k = M^{-1}b + (1 - M^{-1}A)x_k$$

convergence depends on $\|1 - M^{-1}A\|$!

If stationary method convergent $\longrightarrow \|1 - M^{-1}A\| < 1 \longrightarrow M^{-1} \approx 1 \longrightarrow M$ is a good precondition

Good splitting leads to convergent method \longrightarrow good precondition

- (i) Jacobi-splitting with $D = \text{diag}(A) \longrightarrow$ Jacobi preconditioner $M := D$
- (ii) Gauss-Siedel splitting with $L + D =$ lower part of $A \longrightarrow$ Jacobi preconditioner $M := L + D$

Another approach from Gaussian elimination:

- (iii) ILU = incomplete LU decomposition:

idea:

Fix sparsity pattern of L and U to coincide with the pattern of A

run Gaussian elimination for computing LU on this reduces sparsity pattern

restrict computations to entries in the pattern of A

$\longrightarrow LU : A = LU + R$

e.g. $A = \begin{pmatrix} * & & * \\ & * & \\ * & & * \end{pmatrix}$

restrict L and U to $L = \begin{pmatrix} * & & 0 \\ & * & \\ * & & * \end{pmatrix}, U = \begin{pmatrix} * & & * \\ & * & \\ 0 & & * \end{pmatrix}$

Comparison:

- (i) easy to parallelize, but often slow convergence
- (ii) and (iii) good preconditioning, but sequential!

(iv) instead of approximation A by some M like in (i)–(iii) we try to approximate A directly by some P :

$$\min_{P \in \mathcal{P}} \|AP - I\| \text{ over some set } \mathcal{P} \text{ and in some Norm } \|\cdot\|$$

- choice of the norm: Frobenius norm

$$\|B\|_F^2 = \sum_{j=1}^n \|B_{\cdot j}\|_2^2 = \sum_{j=1}^n \|B_{\cdot j}\|_2^2$$

$$= \text{sum}(\text{diag}(B^T B)) = \text{trace}(B^T B)$$

is the euclidean norm of B seen as a vector unitary invariant analytic

- choice of \mathcal{P} (similar to ILU)

Choose \mathcal{P} as the set of matrices with exactly the same pattern as A

e.g. $A = \begin{pmatrix} * & & * \\ & * & \\ * & & * \end{pmatrix} \longrightarrow P = \left| \begin{pmatrix} * & & * \\ & * & \\ * & & * \end{pmatrix} \right|$

$$\min_{P \in \mathcal{P}} \|AP - I\|_F^2 = \min_{P \in \mathcal{P}} \left\| \sum_{j=1}^n \|(AP - I)e_j\|_2^2 \right\|$$

$$= \sum_{j=1}^n \min_{p \in P} \|(AP - I)e_j\|_2^2 \text{ with } P_j \text{ the } j\text{-th column of } P$$

Hence the minimization can be solved in parallel for every column P_j of P !

$$\min_{p \in P} \|Ap_j - e_j\|_2^2 = \min_r \left\| A \cdot \begin{pmatrix} 0 \\ r_1 \\ 0 \\ r_2 \\ \vdots \\ 0 \\ r_k \\ 0 \end{pmatrix} - e_j \right\|_2^2$$

0		r ₁		0		r ₁		0	...		r ₁		0
A													

$$\begin{pmatrix} 0 \\ r_1 \\ 0 \\ r_2 \\ \vdots \\ 0 \\ r_k \\ 0 \end{pmatrix} == A(:, I_j) \cdot \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} = A(:, I_j) \cdot r$$

where I_j are the allowed entries in P_j

$$\min_r \|A(:, I_j) \cdot r - e_j\|_2^2$$

$$A(:, I_j) = \begin{pmatrix} \square \\ \square \\ \square \\ \square \\ \square \end{pmatrix} \text{ has many zero rows that can be eliminated.}$$

$J_j :=$ indices of non zero rows of $A(:, I_j) \longrightarrow$

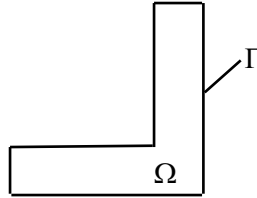
$$\min_r \|A(:, I_j) \cdot r - e_j\|_2^2 = \min_r \|A(J_j, I_j) \cdot r - e_j(J_j)\|_2^2 \text{ with } A(J_j, I_j) \text{ small matrix}$$

Least squares problem $\longrightarrow r \longrightarrow P_j$

Computation of P easy in parallel

application of precondition P vector easy to parallelize

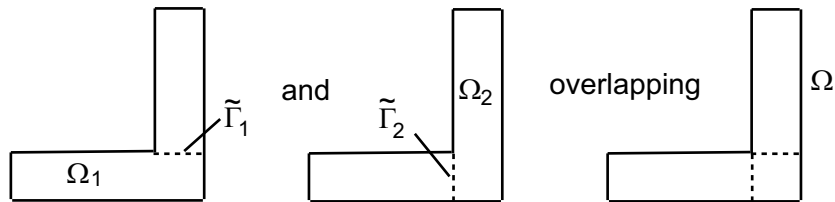
6 Domain Decomposition Methods for Solving PDE



region Ω with boundary Γ

Given PDE, e.g. $\Delta u = u_{xx} + u_{yy} = \frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} = f(x, y)$
 in Ω and $u|_{\Gamma} = q$ Dirichlet problem

How to parallelize?



Γ_1 boundary of Ω_1 with $\tilde{\Gamma}_1$ unknown values

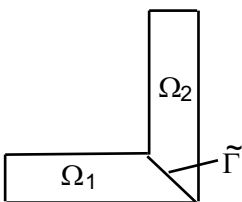
Γ_2 boundary of Ω_2 with $\tilde{\Gamma}_2$ unknown values

Idea:

Solve PDE Ω_1 with some estimate on $\tilde{\Gamma}_1$	Solve PDE Ω_2 with some estimate on $\tilde{\Gamma}_2$
Exchange values	
Solve PDE Ω_1 with some estimate on $\tilde{\Gamma}_1$ taken from Ω_2	Solve PDE Ω_2 with some estimate on $\tilde{\Gamma}_2$ taken from Ω_1
then go back to exchange values	

Convergence against solution of PDE on Ω

Non overlapping method



Discretization in \rightarrow linear system $Au = f$

numbering of unknowns such that first variables in Ω_1 then Ω_2 , then $\tilde{\Gamma} \rightarrow$

$$f = Au = \begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & G_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

like dissection

Solve $Au = f$ by Schur decomposition or iteratively with the precondition

$$\begin{pmatrix} A_1^{-1} & & \\ & A_1^{-1} & \\ & & 1 \end{pmatrix} \sim \text{to solving PDE in } \Omega_1 \text{ and } \Omega_2$$