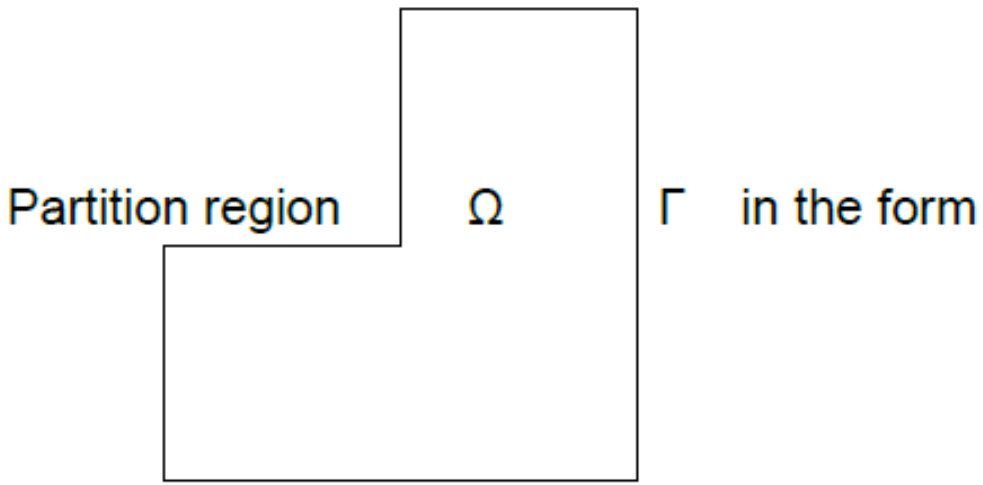
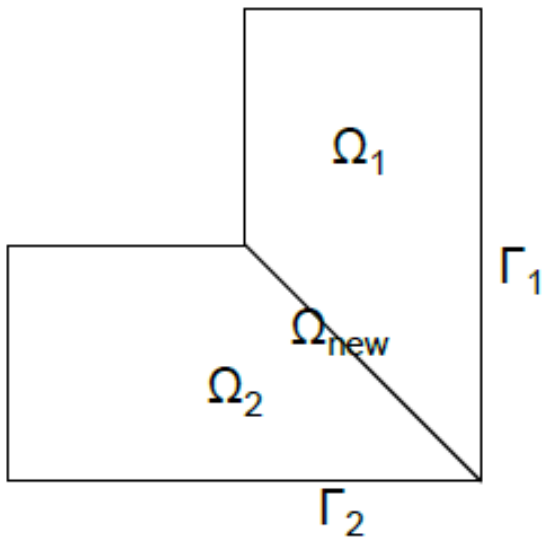
6.2. Non-overlapping Domain Decomposition





Discretization of the original problem with numbering of the unknowns relative to the partitioning given by Ω_1 and Ω_2 leads to a linear system with a matrix in dissection form.

Non-overlapping DD (cont. 2)

Poisson equation on domain Ω

$$\begin{aligned} -\Delta u &= f && \text{in} \quad \Omega, \\ u &= 0 && \text{on} \quad \partial\Omega \end{aligned}$$

is equivalent to

$$\begin{aligned} -\Delta u_1 &= f && \text{in} \quad \Omega_1, \\ u_1 &= 0 && \text{on} \quad \partial\Omega_1 \setminus \Gamma, \end{aligned}$$

$$\begin{aligned} u_1 &= u_2 && \text{on} \quad \Gamma, \\ \frac{\partial u_1}{\partial n_1} &= -\frac{\partial u_2}{\partial n_2} && \text{on} \quad \Gamma, \end{aligned}$$

$$\begin{aligned} -\Delta u_2 &= f && \text{in} \quad \Omega_2, \\ u_2 &= 0 && \text{on} \quad \partial\Omega_2 \setminus \Gamma. \end{aligned}$$

Non-overlapping DD (cont. 3)

In matrix-vector notation $Au = f$ can be written as

$$A = \begin{pmatrix} A_{I,I}^{(1)} & 0 & A_{I,\Gamma}^{(1)} \\ 0 & A_{I,I}^{(2)} & A_{I,\Gamma}^{(2)} \\ A_{\Gamma,I}^{(1)} & A_{\Gamma,I}^{(2)} & A_{\Gamma,\Gamma} \end{pmatrix}, u = \begin{pmatrix} u_I^{(1)} \\ u_I^{(2)} \\ u_\Gamma \end{pmatrix}, f = \begin{pmatrix} f_I^{(1)} \\ f_I^{(2)} \\ f_\Gamma \end{pmatrix}, \qquad (1)$$

where the degrees of freedom are partitioned into those internal to Ω_1, and to Ω_2, and those of the interior of Γ.

On next slide we formulate problem (1) in a more general notation.

Non-overlapping DD (cont. 4)

- A_3 is the so called interface matrix:

$$f = Au = \begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

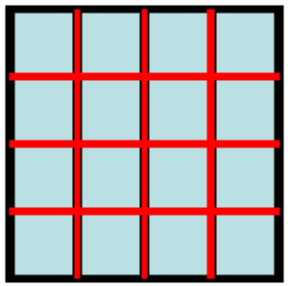
  Better reduce the original problem to two partial subproblems and one interface Schur complement system.

- We can solve $Au = f$ iteratively with PCG and preconditioner:

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

  Here, for M we can use the identity or an approximate inverse for the Schur complement.

Non-overlapping DD (cont. 5)



Leads to 16 block matrices on the diagonal $A_1, \ldots, A_{16}$ and Schur complement S.

$$A = \begin{pmatrix} A_1 & & & F_1 \\ & \ddots & & \vdots \\ & & A_{16} & F_{16} \\ G_1 & \cdots & G_{16} & A_{17} \end{pmatrix}$$

$$\left(S = A_{17} - G_1 A_1^{-1} F_1 - \ldots - G_{16} A_{16}^{-1} F_{16} \right)$$

- Solve small problems e.g. with multigrid in parallel.
- Overlapping is easy to parallelize, but slow convergence
- Non-overlapping is harder to parallelize, more influence on convergence in S.

6.3. Schur Complements

Write matrix A from (1) in block factorized form $A = LR$

$$L = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A^{(1)}_{\Gamma,I} A^{(1)^{-1}}_{I,I} & A^{(2)}_{\Gamma,I} A^{(2)^{-1}}_{I,I} & I \end{pmatrix}, \qquad R = \begin{pmatrix} A^{(1)}_{I,I} & 0 & A^{(1)}_{I,\Gamma} \\ 0 & A^{(2)}_{I,I} & A^{(2)}_{I,\Gamma} \\ 0 & 0 & S \end{pmatrix}$$

leading to the resulting linear system

$$\begin{pmatrix} A^{(1)}_{I,I} & 0 & A^{(1)}_{I,\Gamma} \\ 0 & A^{(2)}_{I,I} & A^{(2)}_{I,\Gamma} \\ 0 & 0 & S \end{pmatrix} u = \begin{pmatrix} f^{(1)}_I \\ f^{(2)}_I \\ b_\Gamma \end{pmatrix},$$

where

$$S = A_{\Gamma,\Gamma} - A^{(1)}_{\Gamma,I} A^{(1)^{-1}}_{I,I} A^{(1)}_{I,\Gamma} - A^{(2)}_{\Gamma,I} A^{(2)^{-1}}_{I,I} A^{(2)}_{I,\Gamma}$$

being the Schur complement relative to the unknowns Γ.

Schur Complements (cont.)

We can transform $Au = f$ into $LRu = f$, resp. $Ru = L^{-1}f = b$ with

$$L^{-1} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A^{(1)}_{\Gamma,I} A^{(1)}_{I,I}{}^{-1} & A^{(2)}_{\Gamma,I} A^{(2)}_{I,I}{}^{-1} & I \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -A^{(1)}_{\Gamma,I} A^{(1)}_{I,I}{}^{-1} & -A^{(2)}_{\Gamma,I} A^{(2)}_{I,I}{}^{-1} & I \end{pmatrix}$$

$$\begin{pmatrix} A^{(1)}_{I,I} & 0 & A^{(1)}_{I,\Gamma} \\ 0 & A^{(2)}_{I,I} & A^{(2)}_{I,\Gamma} \\ 0 & 0 & S \end{pmatrix} u = \begin{pmatrix} f^{(1)}_I \\ f^{(2)}_I \\ b_\Gamma \end{pmatrix}$$

$$b_\Gamma = f_\Gamma - A^{(1)}_{\Gamma,I} A^{(1)}_{I,I}{}^{-1} f^{(1)}_I - A^{(2)}_{\Gamma,I} A^{(2)}_{I,I}{}^{-1} f^{(2)}_I$$

Once u_Γ is found, the internal components can be found by

$$u^{(i)}_I = A^{(i)}_{I,I}{}^{-1} (f^{(i)}_I - A^{(i)}_{I,\Gamma} u_\Gamma).$$

Direct Derivation of the Schur Complement

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\begin{aligned} \Rightarrow \quad & A_1 x_1 + F_1 x_3 = b_1 \\ & A_2 x_2 + F_2 x_3 = b_2 \\ & G_1 x_1 + G_2 x_2 + A_3 x_3 = b_3 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & x_1 = A_1^{-1} b_1 - A_1^{-1} F_1 x_3 \quad \text{and} \\ & x_2 = A_2^{-1} b_2 - A_2^{-1} F_2 x_3 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & (G_1 A_1^{-1} b_1 - G_1 A_1^{-1} F_1 x_3) + (G_2 A_2^{-1} b_2 - G_2 A_2^{-1} F_2 x_3) + A_3 x_3 = b_3 \\ \Rightarrow \quad & (A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2) x_3 = b_3 - G_1 A_1^{-1} b_1 - G_2 A_2^{-1} b_2 \\ \Rightarrow \quad & S x_3 = \widetilde{b}_3 \end{aligned}$$

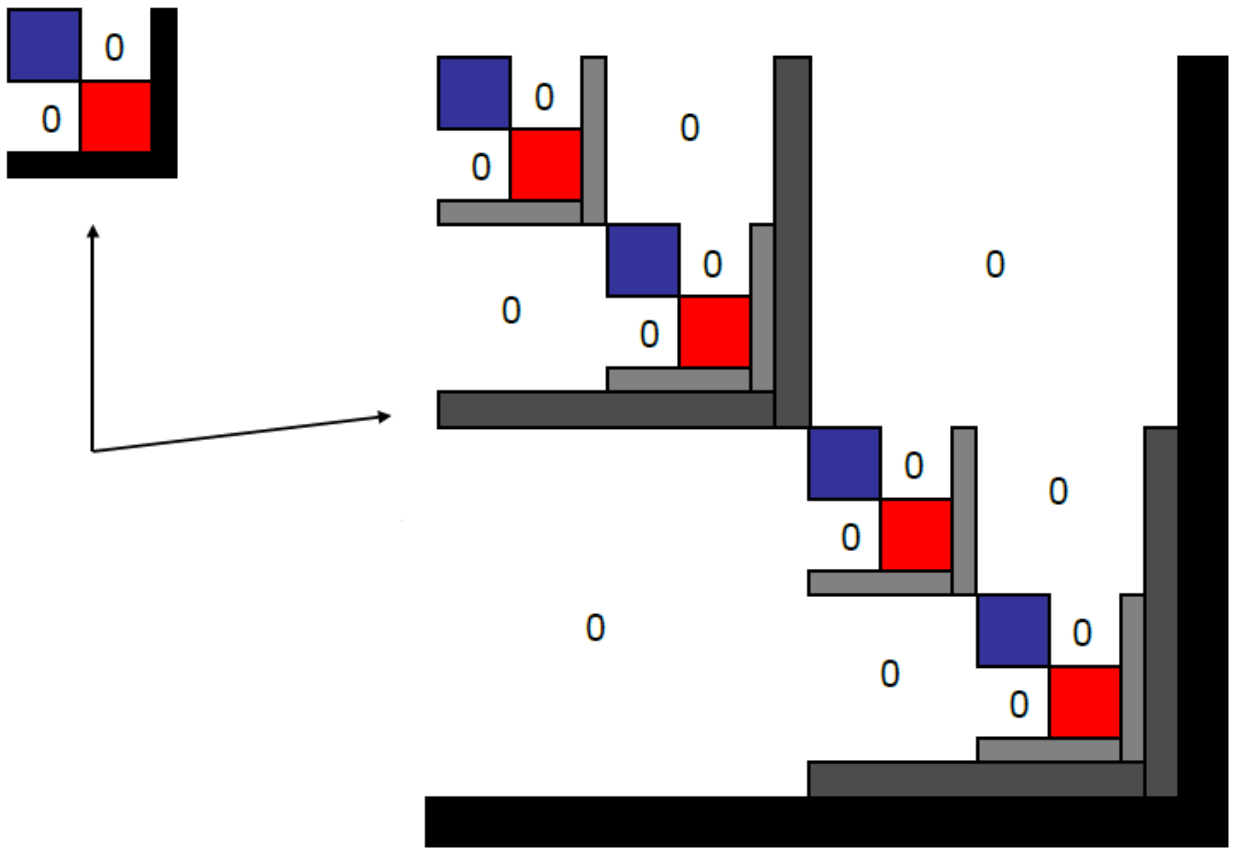
(Parallel) Algorithm to solve $Ax = b$ based on Schur Complement

1. Compute S by using inv(A_1) and inv(A_2)
2. Solve $Sx_3 = \widetilde{b}_3$
3. Compute x_1 and x_2 by using inv(A_1) and inv(A_2)

- The explicit computation of S can be avoided by solving the linear system in S iteratively, e.g. Jacobi, PCG, ...
- Then we need only a part of S and in every iteration step we have to compute $S *$ intermediate vector.
- To achieve fast convergence, a preconditioner (approximation) for S has to be used!
- Precondition Schur complement e.g. with MSPAI

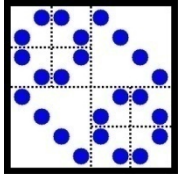
Recursive Form of Non-overlapping DD

$\leftrightarrow$ Nested (recursive) dissection:



Domain Decomposition: Outlook

- Approach can be generalized to
  - non-conforming discretizations (mortar methods/Lagrange multipliers/FETI)
  - time-dependent systems
  - ...
- Literature:
  - A. Toselli, O. Widlund: *Domain Decomposition Methods—Algorithms and Theory*, Springer, 2004
  - A. Quarteroni, A. Valli: *Domain Decomposition Methods for Partial Differential Equations*, Oxford Science Publications, 1999



Multigrid

Starting point: Solve Partial Differential Equation, e.g.

$$-u_{xx} - u_{yy} = f(x, y)$$

with boundary conditions, e.g. Dirichlet BC.

Discretization leads to system of linear equations, resp. matrix A.

A is sparse, (structured,) and ill-conditioned.
Looking for O(n) solver.

Direct solver $O(n^2)$ or $O(n \log(n))$
PCG also > O(n) because matrix is ill-conditioned

Idea

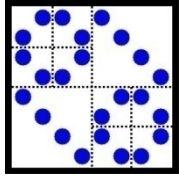
Vector of unknowns:

(1) Project the fine discretization on a coarser (smaller) problem

(2) Solve the coarse matrix (smaller matrix)

(3) Project the coarse solution back on fine grid

Problems: - only possible for smooth vector
without high oscillatory components
- backprojection introduces (high-oscillatory)
errors that have to be removed, too.



Observation:

For typical PDE matrices high-oscillatory vectors are related to the subspace to large eigenvalues and are removed e.g. by the stationary Gauss-Seidel iteration:

$|| I - M^{-1}A ||$ small for eigenvectors to large eigenvalues!

To solve Ax=f:

(1) Apply a few steps Gauss-Seidel smoothing steps → x_a
Residual equation $A(x_a+x)=f$ → $Ax = f - Ax_a = r$

(2) Project (restrict) r, resp. A on coarse grid by mean value
Solve $A_c\, x_c = r_c$

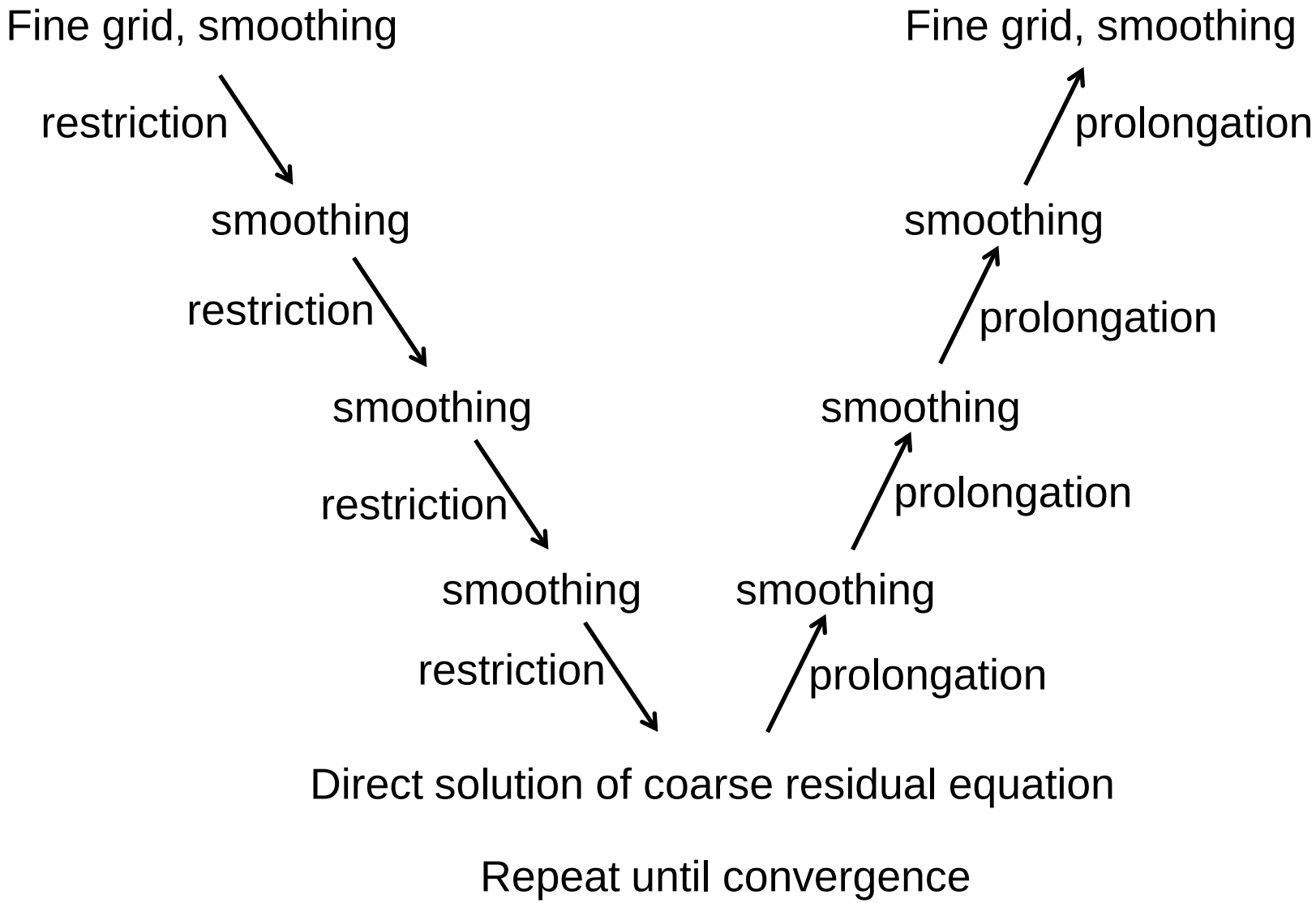
(3) Project (prolongate) x_c back on fine grid x by interpolation
x_c → x → new approximate solution $x_a + x$

(4) Improve approximate solution by Gauss-Seidel steps

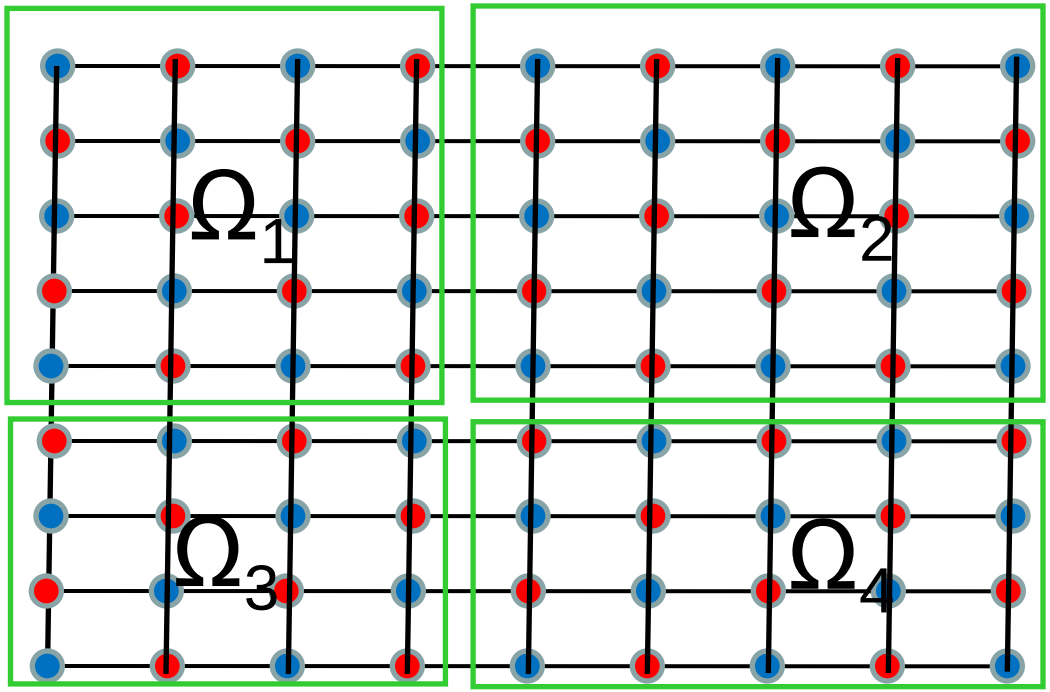
Repeat until convergence

Apply also recursively for solving coarse equations

V-Cycle

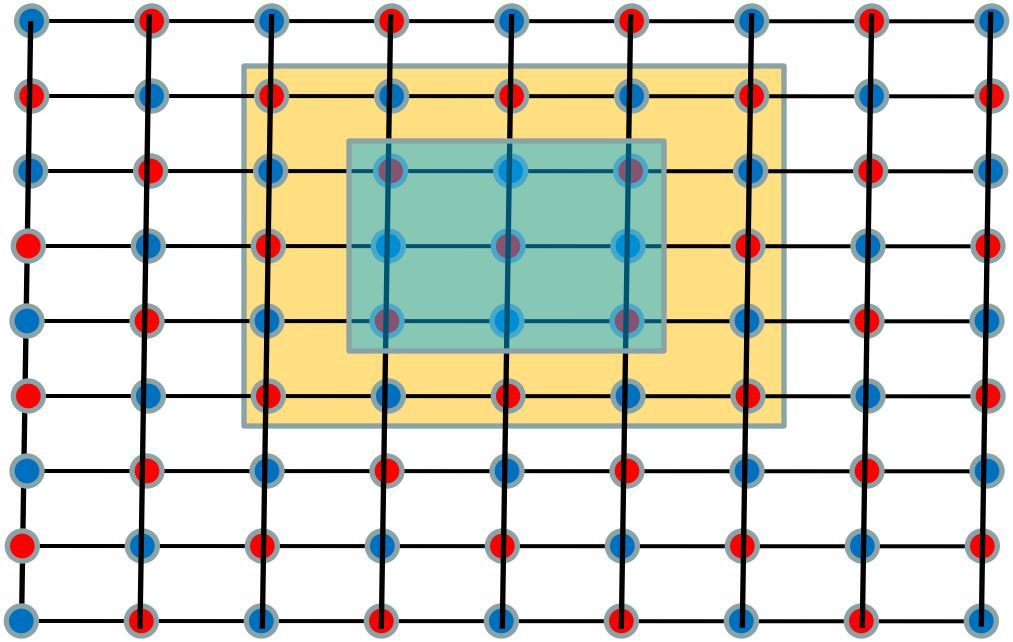

Fine grid, smoothing
restriction
smoothing
restriction
smoothing
restriction
smoothing
restriction
Direct solution of coarse residual equation
prolongation
smoothing
prolongation
smoothing
prolongation
smoothing
prolongation
Fine grid, smoothing
Repeat until convergence

Parallel Aspects



Each processor p_r has data for domain Ω_r and does projections and smoothing for its components.
Needs data from neighboring processor.
Load balancing!

Ghost layers



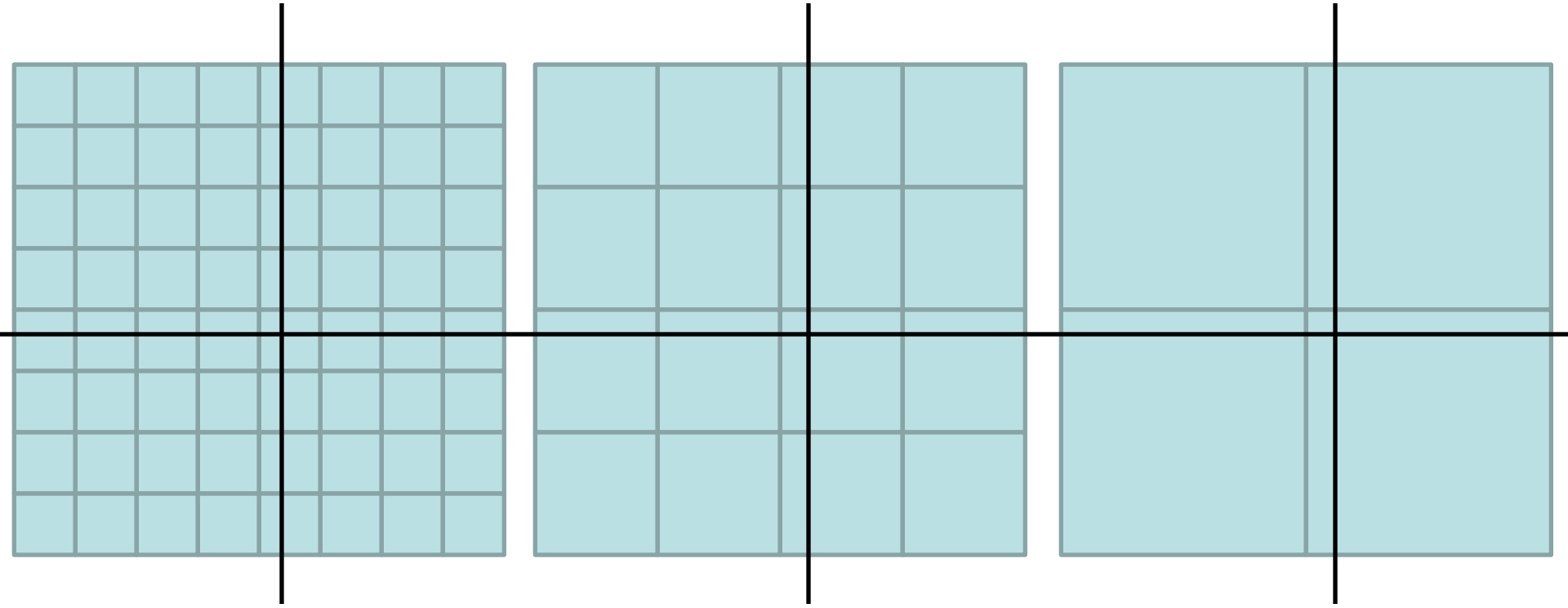
Jacobi smoothing:

(1) Each process performs Jacobi iteration (independently)

(2) Send messages to update ghost layers

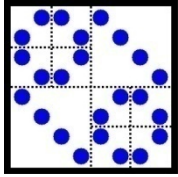
(3) Use communication/computation overlap for interior computations and exchange of boundary data

Restriction/Prolongation



Problems with work load: On coarse grids less computations, more communication!

Efficiency goes like 1/log(p) with the number of processors.



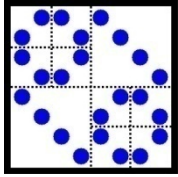
Modifications

Agglomeration: Coarse grid partitions are no longer aligned with the finer grid partitions (in order to avoid inefficiency on coarsest grids)

→ More communication in grid transfer, less in coarse grid solve.

Larger ghost layers can reduce the communication

Reduce number of V-cycle steps by using more powerful projections and smoothers → less data transfer



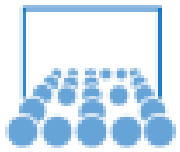
Additive Multigrid

Consider the different levels at the same time,
Compute the related corrections in parallel and sum up the corrections.

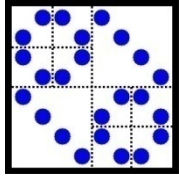
$$x = x_0 + x_1 + \ldots + x_L$$

$$x := x \; + \alpha \sum_{l=0}^{L} A_l^{-1} P_l (f - Ax \;)$$

Hybrid conjugate gradient iteration with MG as preconditioner
→ Less V-cycles



8. Computing Eigenvalues in Parallel



8.1 Introduction

x eigenvector with eigenvalue λ, iff: $Ax = \lambda x, x \neq 0$

A spd, there exists an orthogonal basis of eigenvectors $Au_i = \lambda_i u_i$, i=1,2,…,n

$$A = U\Lambda U^T \qquad or \qquad AU = U\Lambda$$

$$U = (u_1,...,u_n), \qquad \Lambda = diag(\lambda_i,...,\lambda_n)$$

The eigenvalues of A are the zeros of the characteristic polynomial:
$P_n(A) = 0 \leftrightarrow P_n(\lambda_i)=0$ for all eigenvalues λ_i, i=1,2,…,n

$\lambda_{min}(A) <= r(A) = x^HAx / x^Hx =< \lambda_{max}$: Rayleigh quotient

In general U may be complex unitary and Λ an upper triangular complex matrix (Schur decomposition)

Allowed operations that do not change the eigenpair: $Q \cdot A \cdot Q^H$ with unitary Q

8.2 Parallel Methods for computing all eigenvalues: Jacobi Method

$$off(A)^2 := \|A\|_F^2 - \sum_{i=1}^{n} a_{ii}^2 = \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} a_{ij}^2$$

$A = A^T$

Describes the magnitude of the nondiagonal part of A (should → 0)

We use Givens rotations to eliminate a_{pq} for an index pair p,q

$$J(p,q,\theta) := \begin{pmatrix} \ddots & & p & & q & & 0 \\ & 1 & & & & & \\ & & c & & s & & p \\ & & & 1 & & & \\ & & & & \ddots & & \\ & & & & & 1 & \\ & & -s & & c & & q \\ & & & & & 1 & \\ 0 & & & & & & \ddots \end{pmatrix}$$

c = cos(θ),

s = sin(θ)

Jacobi Method

Effect of application of J on A on the nondiagonal entries off(A).

Consider p,q – part of $J^T A J = B$:

$$\begin{pmatrix} b_{pp} & 0 \\ 0 & b_{qq} \end{pmatrix} \overset{!}{=} \begin{pmatrix} b_{pp} & b_{pq} \\ b_{qp} & b_{qq} \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

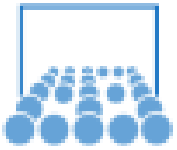
$$b_{pq} = b_{qp} = a_{pq}(c^2 - s^2) + (a_{pp} - a_{qq})cs \overset{!}{=} 0 \Rightarrow \theta, \cos(\theta), \sin(\theta)$$

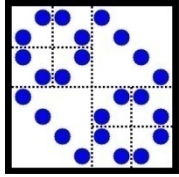
J orthogonal → Frobeniusnorm of A and B are the same:

$$a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2 = b_{pp}^2 + b_{qq}^2$$

$$off^2(B) = \|B\|_F^2 - \sum b_{ii}^2 = \|A\|_F^2 - \sum_{p \neq i \neq q} b_{ii}^2 - (a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2) =$$

$$= \|A\|_F^2 - \sum_{p \neq i \neq q} a_{ii}^2 - (a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2) == \|A\|_F^2 - \sum a_{ii}^2 - 2a_{pq}^2 = off^2(A) - 2a_{pq}^2 < off^2(A)$$





Elimination Sequence

Choose p and q such that a_{pq} is very large (maximum).

Then by $J^T A J$ the size of the off-diagonal entries is reduced by $2 (a_{pq})^2$.

Repeat this transformation for next choice of p and q: A → diagonal.

Different strategies for choosing a sequence of p,q:

Maximum a_{pq} optimal, but sequential and expensive!

Cyclic by row:

First use a_{11} to eleminate first row: (p,q)=(1,2),(1,3),…,(1,n)
Then a_{22} for second row: (p,q)=(2,3),….,(2,n)
a_{33} , …, $a_{n-1,n-1}$
Repeat

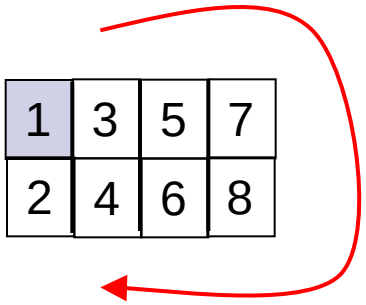
Again sequential!

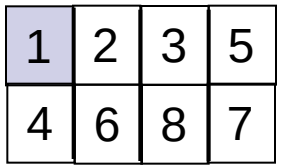
Jacobi Method in Parallel

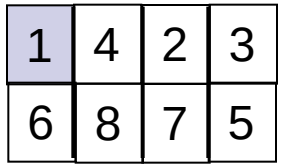
Choose sequence (p,q) such that it allows strong parallelism:

First sweep: (p,q) = (1,2), (3,4), (5,6), (7,8) (in parallel)

Second sweep: (p,q) = (1,4), (2,6), (3,8), (5,7)
Third (1,6), (4,8), (2,7), (3,5)
Fourth (1,8), (6,7), (4,5), (2,3)
.. ..
.. ..







.. ..

n-1 different positions define (n-1)n/2 deleted entries = subdiagonal

Find sequence of partitionings of (1,…n) in pairs, such that all indices appear with the same frequency.

Parallel Transformation

$J^T A$:

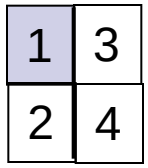
$$\begin{pmatrix} * & * & * & * & * & * & * & * \\ * & * & * & * & * & * & * & * \\ \# & \# & \# & \# & \# & \# & \# & \# \\ \# & \# & \# & \# & \# & \# & \# & \# \\ + & + & + & + & + & + & + & + \\ + & + & + & + & + & + & + & + \\ \sim & \sim & \sim & \sim & \sim & \sim & \sim & \sim \\ \sim & \sim & \sim & \sim & \sim & \sim & \sim & \sim \end{pmatrix}$$

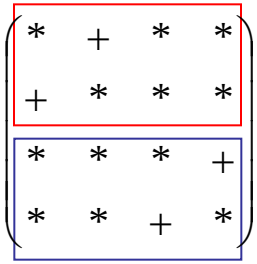
$(J^T A)\, J$:

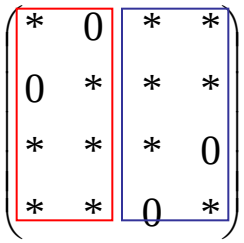
$$\begin{pmatrix} * & * & \# & \# & + & + & \sim & \sim \\ * & * & \# & \# & + & + & \sim & \sim \\ * & * & \# & \# & + & + & \sim & \sim \\ * & * & \# & \# & + & + & \sim & \sim \\ * & * & \# & \# & + & + & \sim & \sim \\ * & * & \# & \# & + & + & \sim & \sim \\ * & * & \# & \# & + & + & \sim & \sim \\ * & * & \# & \# & + & + & \sim & \sim \end{pmatrix}$$

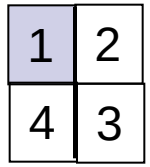
Multiplications with J^T , resp. J

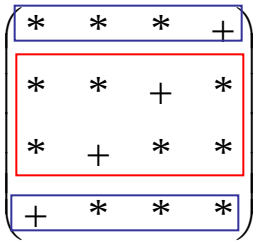
can be done in parallel.

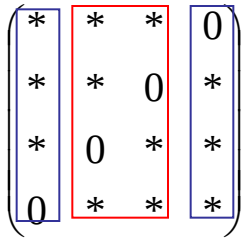


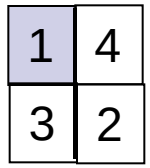


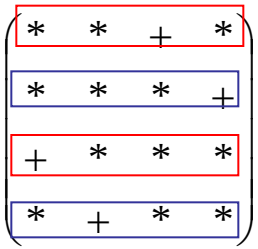


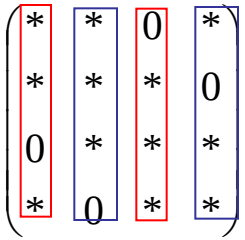












Twelve zeros after three sweeps ←→ twelve nondiagonal entries

Repeat until convergence to diagonal matrix.

8.3 Divide & Conquer for tridiagonal A

A divide and conquer approach for computing eigenvalues of a symmetric tridiagonal matrix T.

$$T = \begin{pmatrix} a_1 & b_1 & & \\ b_1 & a_2 & & \\ & & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{pmatrix}$$

Idea: Split T in two tridiagonal matrices T_1 and T_2.
Compute eigenvalues of T_1 and T_2.
Recover the original eigenvalues of T as perturbations.

Repeat recursively.

Splitting of T

Set $v := (0 \quad \cdots\cdots \quad 0 \quad 1 \mid \theta \quad 0 \quad \cdots\cdots \quad 0)^T$

$$\tilde{T} := T - \rho v v^T$$ Rank-1 perturbation of T

Aim: Generate zeros at the sub/superdiagonal entries in the middle of T

$$\tilde{T}(m:m+1, m:m+1) = \begin{pmatrix} a_m & b_m \\ b_m & a_{m+1} \end{pmatrix} - \rho \begin{pmatrix} 1 & \theta \\ \theta & \theta^2 \end{pmatrix} =$$

$$= \begin{pmatrix} a_m - \rho & b_m - \rho\theta \\ b_m - \rho\theta & a_{m+1} - \rho\theta^2 \end{pmatrix} \overset{!}{=} \begin{pmatrix} * & 0 \\ 0 & * \end{pmatrix}$$

$$\rho\theta = b_m \longrightarrow T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + \rho v v^T$$

Relation between T and T_1, T_2

Assume, that we know the eigenvalues and eigenvectors of T_1 and T_2 .

How can we get the eigenpairs of T?

$$T_1 = U_1 \Lambda_1 U_1^T, \qquad T_2 = U_2 \Lambda_2 U_2^T, \quad \overset{?}{\Rightarrow} \quad T = U \Lambda U^T$$

Note, that T is a rank-1 perturbation of diag(T_1 , T_2).

Recover the original eigenvalues as perturbations of eigenvalues of T_1 and T_2 .

$$T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + \rho v v^T = \begin{pmatrix} U_1 \Lambda_1 U_1^T & 0 \\ 0 & U_2 \Lambda_2 U_2^T \end{pmatrix} + \rho v v^T = \begin{pmatrix} U_1 & 0 \\ 0 & U_2 \end{pmatrix} \begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix} \begin{pmatrix} U_1^T & 0 \\ 0 & U_2^T \end{pmatrix} + \rho v v^T$$

$$\Longrightarrow \quad \begin{pmatrix} U_1^T & 0 \\ 0 & U_2^T \end{pmatrix} T \begin{pmatrix} U_1 & 0 \\ 0 & U_2 \end{pmatrix} = \begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix} + \rho \tilde{v} \tilde{v}^T$$

Computing the eigenvector

Hence, we need to compute the eigenvalues of a matrix

of the form "diagonal + rank-1": $D + \rho \tilde{v} \tilde{v}^T$

Let λ_i and u_i be an eigenpair of $D + \rho v v^T$. Then it holds

$$\left(D + \rho \tilde{v} \tilde{v}^T\right) u_i = \lambda_i u_i \Leftrightarrow \left(D - \lambda_i I\right) u_i + \rho \left(\tilde{v}^T u_i\right) \tilde{v} = 0$$

$$u_i = const \cdot \left(D - \lambda_i I\right)^{-1} \tilde{v}$$

Hence, if we know λ_i, then we directly get the eigenvector u_i .

Eigenvalues as Zeros

Furthermore, we get the equation

$$\tilde{v}^T \left(D - \lambda_i I\right)^{-1} \cdot \left[\left(D - \lambda_i I\right) u_i + \rho\left(\tilde{v}^T u_i\right) \tilde{v}\right] = 0$$

$$\tilde{v}^T u_i + \rho\left[\tilde{v}^T \left(D - \lambda_i I\right)^{-1} \tilde{v}\right] \cdot \left[\tilde{v}^T u_i\right] = 0$$

$$f(\lambda) = 1 + \rho\left[\tilde{v}^T \left(D - \lambda I\right)^{-1} \tilde{v}\right] = 1 + \rho\left(\frac{\tilde{v}_1^2}{d_1 - \lambda} + \cdots + \frac{\tilde{v}_n^2}{d_n - \lambda}\right) = 0$$

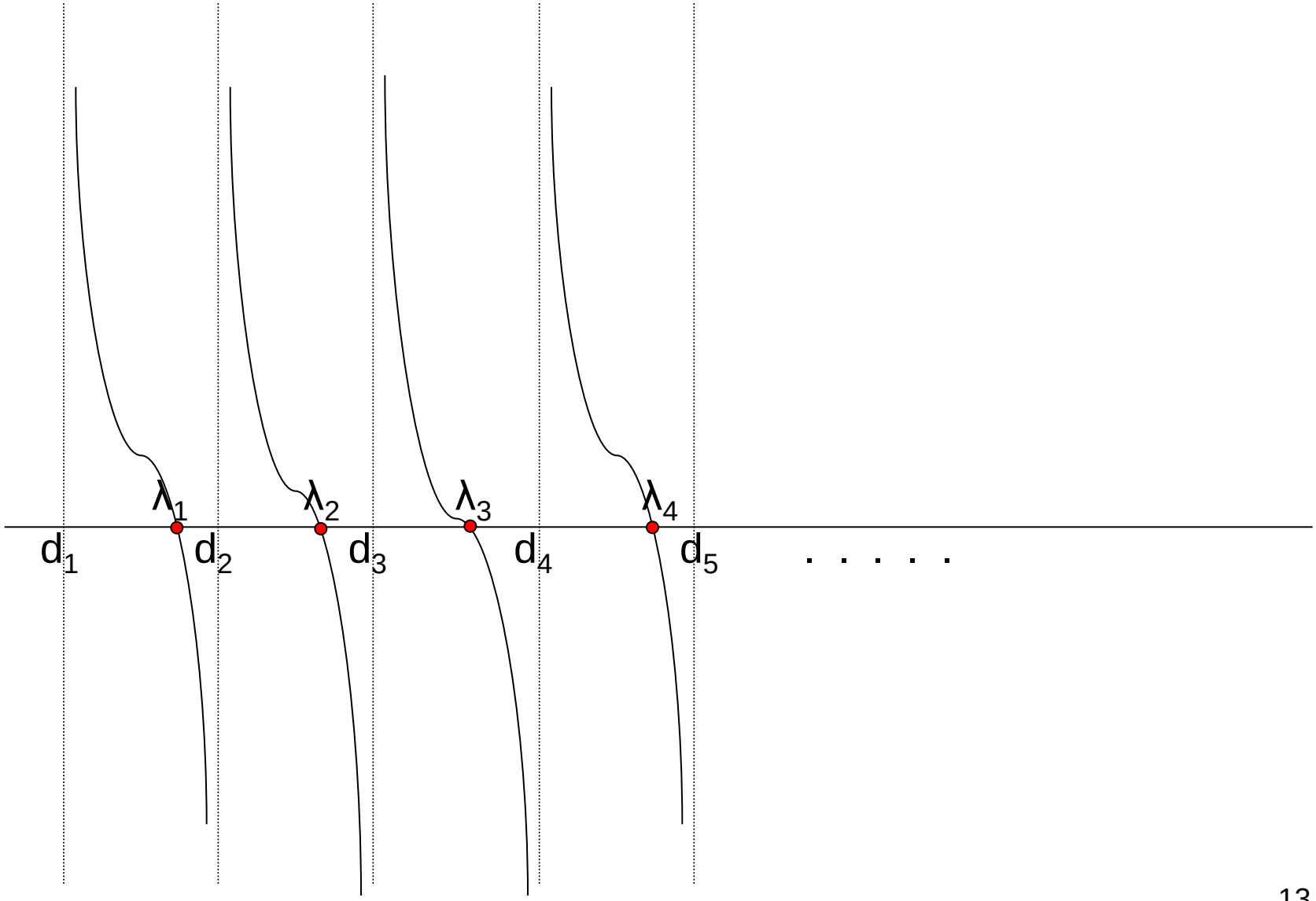
Use Newton's method, to determine the zeroes of function f(λ)

These zeroes are the eigenvalues of $D + \rho \tilde{v} \tilde{v}^T$

and therefore also of T.

Repeat recursively for T_1 and T_2 .

Zeros and poles of f(λ):



8.4 Algorithms for computing a few eigenpairs:

Vector iteration:

$$x^{(k)} = A^k x^{(0)} \Big/ \left\| A^k x^{(0)} \right\| \to v$$

eigenvector to eigenvalue with maximum absolute value

Easy to parallelize (only Ax), but slow convergence! Only λ_{max}!

Subspace Iteration: Apply the same idea to set of vectors $U^{(0)}=(x^{(0)},\ldots,x^{(m)})$
Consider eigenvalues of $U^{(k)H}AU^{(k)}$ and then replace $U^{(k)}$ by $AU^{(k)}$

Inverse iteration:

Apply vector iteration on shifted problem $(A - \sigma I)^{-1}$
for computing the eigenvector nearest to σ.

Expensive! Ill-conditioned linear system!