Iterative methods and preconditioners

Victor Eijkhout
eijkhout@cs.utk.edu
ICL, University of Tennessee

CS 594, March 2003

Overview

Theory of iterative methods

Computational kernels

Preconditioners

Pro/con iterative or direct methods

Against direct Variants of Gaussian elimination: Time and space demand can be huge.

No use of analytical properties.

Pro iterative Use analytical properties of systems.

Less time and space. Possibly.

Pro direct Predictable in advance. Solution up to round-off.

Against iterative Unpredictable. Solution not guaranteed.

Iterative methods

Stationary methods

Iterative refinement:

- To solve: $Ax = b$.
- Suppose $M \approx A$ and solve $Mx_1 = b$
- Error: $e_1 = x_1 - x$; residual: $r_1 = Ax_1 - b$
- Since $x = A^{-1}b = x_1 - A^{-1}r_1$,
  - define $x_2 = x_1 - M^{-1}r_1$,
  - and $x_3 = x_2 - M^{-1}r_2$,
  - et cetera

Analysis of stationary methods

Basic step:

$$r_2 = Ax_2 - b = A(x_1 - M^{-1}r_1) - b = (I - AM^{-1})r_1$$

Inductively:

$$r_n = (I - AM^{-1})^{n-1}r_1$$

so convergence if all

$$|\lambda(I - AM^{-1})| < 1.$$

.

Examples of stationary iterative methods

Jacobi method:

$$M = D_A = \text{diag}(A)$$

Gauss-Seidel method:

$$M = D_A + L_A$$

SOR method:

$$M = D_A + \omega L_A$$

These methods converge for M matrices:

- A positive definite
- $a_{ij} \leq 0$ for $i \neq j$.

.

Steepest descent

Iterative process for $Ax = b$:

$$x_{i+1} = x_i + M^{-1} r_i$$

Slight generalization:

$$x_{i+1} = x_i + \alpha_i M^{-1} r_i$$

line search;
with optimal α_i: *steepest descent*

.

Derivation of Krylov space methods

Stationary methods:

$$r_n = (I - AM^{-1})r_{n-1}, \qquad r_n = (I - AM^{-1})^{n-1} r_1$$

Try more general (with $M = I$):

$$r_n = P_{n-1}(A) r_1$$

or: define the Krylov sequence:

$$k_1 = r_1, \qquad k_{n+1} = A k_n,$$

and set:

$$r_n \in \mathrm{span}(k_1, \ldots, k_n) = \mathrm{span}(r_1, Ar_1, A^2 r_1 \ldots, A^n r_1)$$

or equivalently

$$r_n \in \mathrm{span}(r_1, \ldots, r_{n-1}, Ar_n).$$

.

Acceleration

In the definition

$$r_n = P_{n-1}(A)r_1$$

choose the polynomial coefficients.

Two choices for polynomial coefficients:

- Get P_n as close to zero as possible.
  - $P_n(0) = 1$ consistency condition
  - knowledge of spectrum required
  - $\Rightarrow$ Chebyshev polynomials
- Make residuals mutually orthogonal under some inner product.

Orthogonality

Making error orthogonal to generated subspace

$\Leftrightarrow$ projecting error on subspace

$\Leftrightarrow$ minimising error on subspace

Prerequisite: inner product

$\Rightarrow$ matrix needs to be definite

$\Rightarrow$ the further from elliptic you get, the worse an iterative method will perform

(or change your inner product)

.

Properties of orthogonalising methods

- Theoretically: convergence in N steps (N matrix dimension).
- In practice: error decreases, though not monotonically.
- Theoretically: iterands are optimal under energy norm.
- In practice: not optimal under L_2 norm.
- Monotonic bound: $\#it \sim \sqrt{\kappa(A)}$.

The matter of Symmetry

Question: does

$$r_n \in \mathrm{span}(r_1, \ldots, r_{n-1}).$$

imply that all r_n have to be retained?

Answer:

- If A is symmetric: no. (CG; 3 vectors)
- If A is not symmetric and you want orthogonality: yes. (OrthoDir, OrthoRes)

  Practical solutions:

  – Restart

  – Truncate
- If A is not symmetric and you can relax orthogonality: no. (BiCG; 5 vectors)

.

Optimality

Idea: take $\mathrm{span}(r_1, \ldots, r_{n-1})$ and construct a minimal-length affine combination.

- Theoretical optimum obtained exactly. (MinRes from CG, GMRES from OrthoRes)
- Optimum obtained up to small factor. (QMR from BiCG)
- Long sequences iff the original method has them.

Speed of convergence

Number of iterations bounded by $\sqrt{\kappa(A)}$ or $\sqrt{\kappa(M^{-1}A)}$.

Actual number is complicated function of spectrum

Preconditioner: reduce condition number by constant or order

.

Use of transpose

Some methods for non-symmetric matrices need the transpose matrix-vector multiplication:

- OrthoRes / OrthoDir and GMRES: no.
- BiCG and QMR: yes.

Problem for implicitly defined matrix-vector product.

Sketch of CG

FOR $j = 1, 2, \ldots, m$

$$z_j = M^{-1} r_j$$
$$\rho_j = z_j^t r_j$$
$$p_j = r_{j-1} - \frac{\rho_j}{\rho_{j-1}} p_{j-1}$$
$$\pi_j = p_j^t A p_j$$
$$r_{j+1} = r_j + \frac{\rho_j}{\pi_j} A p_j$$
$$x_{j+1} = x_j + \frac{\rho_j}{\pi_j} p_j$$

Observations about CG

- preconditioner / matrix not applied as combo
- inner products introduce synchronization points
- limited storage needs
- no coupling of x_j and r_j sequences

.

Sketch of GMRES

FOR $j = 1, 2, \ldots, m$

$w = M^{-1} A v_j$

FOR $i = 1, \ldots, j$

$h_{ij} = (w, v_i)$

$w = w - h_{ij} v_i$

$h_{j+1,j} = \|w\|_2$; $v_{j+1} = w / \|w\|_2$

Solve $H_m y_m = \|r_0\| e_1$

$x = x_0 + [v_1, \ldots, v_m]\, y_m$

if necessary repeat

.

Computational aspects of GMRES

- Matrix-vector product
- Preconditioner solve
- Inner products: linear in iteration #
- Vector updates

Computational kernels

Sparse matrix-vector product

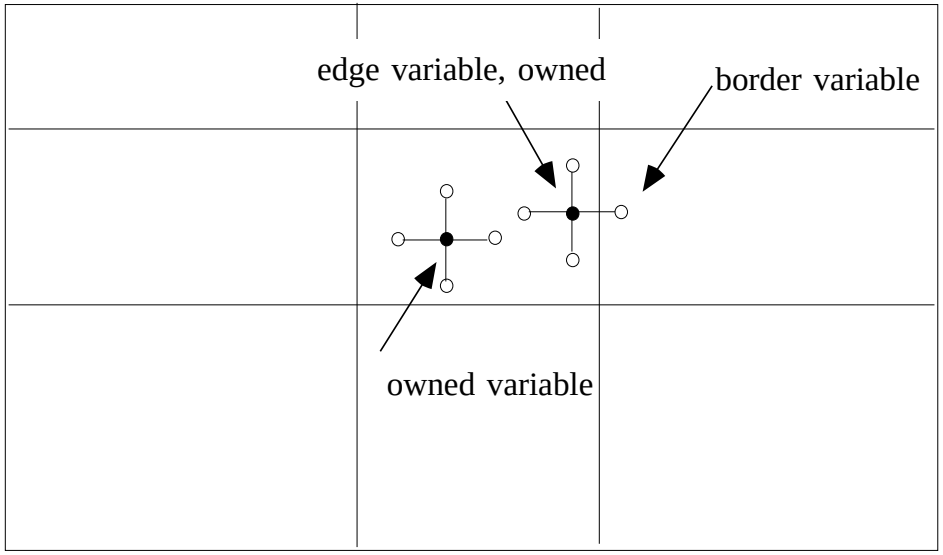
```
do i=1,n
  s = 0
  do j=ptr(i),ptr(i+1)-1
     s = s + a(j) * x(idx(j))
  end do
  x(i) = s
end do
```

3 mem refs / 2 ops $\Rightarrow$ low efficiency, little cache reuse

.

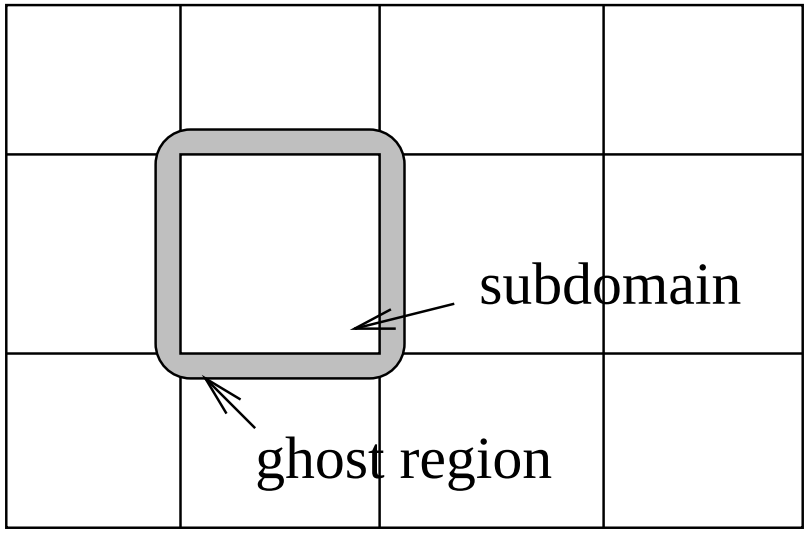
Sparse matrix-vector product, distributed

Vector elements referenced in stencil fashion:



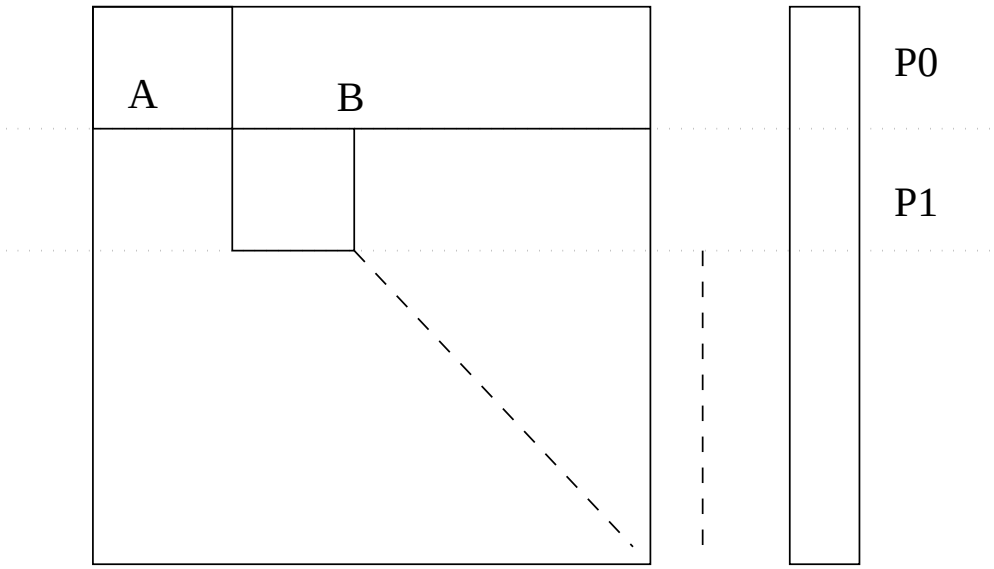
Sparse matvec, distributed, cont'd

Processor subdomain references 'ghost' border region:



Sparse matvec, distributed, ccont'd

Matrix-vector product has local and remote references:



Sparse matrix-vector product, algorithm

Overlapping communication/computation:

```
Initiate send/recv of boundary data
Perform local matvec
Complete recv of boundary data
Perform matvec with boundary data,
and update
```

.

Inner products

Inner products are almost unavoidable:
Chebyshev method needs spectrum information

Inner products are bad news in parallel: synchronisation points $\Rightarrow$ try to bundle them.

Combine inner products:

- Three-term methods: combine two inner products per iteration (Chronopoulos/Gear, Saad/Meurant, D'Azevedo/Eijkhout/Romine)
  $\Rightarrow$ can be stable
- Arnoldi methods: use non-modified Gramm-Schmid, GS twice, block orthogonalisation on partial Krylov series.
  $\Rightarrow$ stability a problem.

or hide them (vd Vorst)

.

Advanced iterative methods

Reduction GMRES-iterations

restrict number of GMRES steps
by combination with other method

1. further reduction of r_i, with e.g. GMRES itself:
   GMRESR (VDV and Vuik '94)
2. precond of search direction: FGMRES (Saad '93)

GMRESR: more memory; robust, reductions add

FGMRES: cheaper in memory; not robust

.

GMRESR

$r_0 = b - Ax_0$;
for $i = 0, 1, 2, \ldots$
z_m approx. sol. of $Az = r_i$ (nested GMRES)
$c = Az_m$
for $k = 0, 1, \ldots, i-1$
$z_m = z_m - \alpha u_k$, $c = c - \alpha c_k$ ($\alpha = (c_k, c)$)
$c_i = c/\|c\|_2$; $u_i = z_m/\|c\|_2$
$x_{i+1} = x_i + (c_i, r_i)u_i$, $r_{i+1} = r_i - (c_i, r_i)c_i$

(FGMRES works with u_i)

Example of GMRESR; Navier Stokes

	GMRES	GMRES(50)	GMRESR+GMRES(10) as "solve"
matvec	184	1220	198
daxpy	17000	35000	1386
ddot	17000	35000	1224
memory	184	50	46
cpu-time	7.2	17.0	1.2

.

Choice of m in GMRESR

GMRES: 177 it., GMRES(50): 1323 it

GMRESR(m):

m	it	m*it	cpu-time	memory
4	45	180	1.15	94
8	23	184	0.89	54
12	16	192	0.90	44
16	12	192	1.05	40
20	10	200	1.23	40

Bi-CG and variants

with short recurrencies we can construct
x_i such that $r_i \perp \mathcal{K}^i(A^T; s_0)$

- not optimal in $\mathcal{K}^i(A; r_0)$
- 2 MV's per iteration (one with transpose)
- CG-like computational overhead
- CG-like memory requirements
- more iterations than GMRES:
  $\|Ax_i^{BiCG} - b\|_2 \geq \|Ax_i^{GMRES} - b\|_2$

Variants of BiCG: QMR

QMR (Freund & Nachtigal)

- slightly better than Bi-CG, but not essentially
- more smooth convergence
- more iterations than GMRES
- product with transpose

Squared methods

For BiCG polynomials, it is possible to compute $P_n^2(A)r_1$ (CGS)

- Theoretically double convergence speed, in practice irregular
- No longer need for transpose
- 2 MV per iteration

Motivation to improve CGS

Goal:
smoother / faster convergence

Possibilities:

- different s_0: unsolved problem
- instead of $r_i = P_i^2(A)r_0$:

$$r_i = \widetilde{P}_i(A)P_i(A)r_0$$

with ”nearby” $\widetilde{P}_i$

Variants of BiCG: BiCGstab

BiCGSTAB (Van Der Vorst)

- 2 MV's in BiCG can be used to combine
  BiCG with repeated GMRES(1)
- same costs as BiCG
- often much faster/smoother than BiCG,CGS
- slow/breakdown depending on GMRES(1) step

Preconditioning

Preconditioning

Convergence behavior depends on spectrum, e.g.

$$\#it = \sqrt{\kappa(A)}$$

Reintroduce M:

- Replace solving $Ax = b$ by $AM^{-1}(Mx) = b$ or $M^{-1}Ax = M^{-1}b$.
- Possibly use different inner product to symmetrise the system.
- New convergence bound: $\#it \sim \sqrt{\kappa(AM^{-1})}$.
- Fewer iterations, but more expensive per iteration.
- Extra cost: computation of M.

Approaches to preconditioning

- Idea: replace A by some easier to solve M
- Completely algebraic: Point ILU and MILU (block ILU, RILU, ILQ, drop-tol,...)
- Based on analytical properties: fast solvers, Domain decomposition, Multigrid
- Approximate inverse

Preconditioners without setup

Use matrix elements:

- Jacobi: $M = D_A$
- Gauss-Seidel: $M = D_A + L_A$
- SOR: $M = D_A + \omega L_A$
- SSOR: $M = (\omega^{-1} D_A + L_A)\big((2\omega^{-1} - 1)D_A\big)^{-1}(\omega^{-1} D_A + U_A)$

Incomplete factorisations

Basic idea: factorisation, but ignore small fill-in.

Exact factorisation:

$$\forall_{i,j>k}: a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj}.$$

Approximate:

$$\forall_{i,j>k}: \text{if } a_{ij} \neq 0 \quad a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj}.$$

- $ILU(0)$ (no-fill) or generalised SSOR:
  $M = (D + L_A)D^{-1}(D + U_A)$ where $D \neq D_A$
- More general: $M = (D + L)D^{-1}(D + U)$.
- Allow fill-in
  - drop tolerance (dynamic storage problem)
  - positional dropping: $ILU(k)$

.

ILU Preconditioners: advantages

- general; robust for elliptic PDE's
- relatively simple implementation, less space than exact factorisation
- bridge between Gaussian elimination and iteration

ILU Preconditioners: disadvantages

- Does not exploit analytical properties of A: close to exact factorisation in norm, maybe in condition, probably not in eigenvalues.
- not optimal (h-dependent) for PDE:
  $\kappa(M^{-1}A) = O(h^{-2})$, lower constant
- Caveat: breakdown problem with zero/negative pivots.
- takes some effort to parallelize

.

Modified ILU

Compensate for discarded fill-in: add to diagonal (MILU)

$$\begin{aligned} \forall_{i,j>k}: \text{if } a_{ij} \neq 0 \qquad & a_{ij} \leftarrow a_{ij} - a_{ik} a_{kk}^{-1} a_{kj} \\ \text{if } a_{ij} = 0 \qquad & a_{ii} \leftarrow a_{ii} - a_{ik} a_{kk}^{-1} a_{kj} \end{aligned}$$

Practical interpretation: preservation of mass.

Relaxed ILU

Instead of moving to diagonal, premultiply by $.95$ (vdV)

Related to $O(h^2)$ modifications (Gustafsson, DKR, others)

Breakdown problem of ILU

Zero or negative pivots may appear for PD matrices

Arbitrary repair: Kershaw

Factor $A + \alpha I$: Manteuffel

Eliminate positive elements before factorisation: Gustafsson

Factor $A + \alpha I$ again: Saad in context of indefinite systems

Block factorisations

Factorisation by matrix sub-block:

elementary operations (mult/div) now become matrix-matrix operations

- From geometry: lines or planes in physical domain
- Coupled differential equations: each equation as separate block
- Multi-component PDEs: each node a block; BLAS3 performance on elementary operations.

Severe problem:

$$a_{kk}^{-1}$$

approximation needed (generation/application)

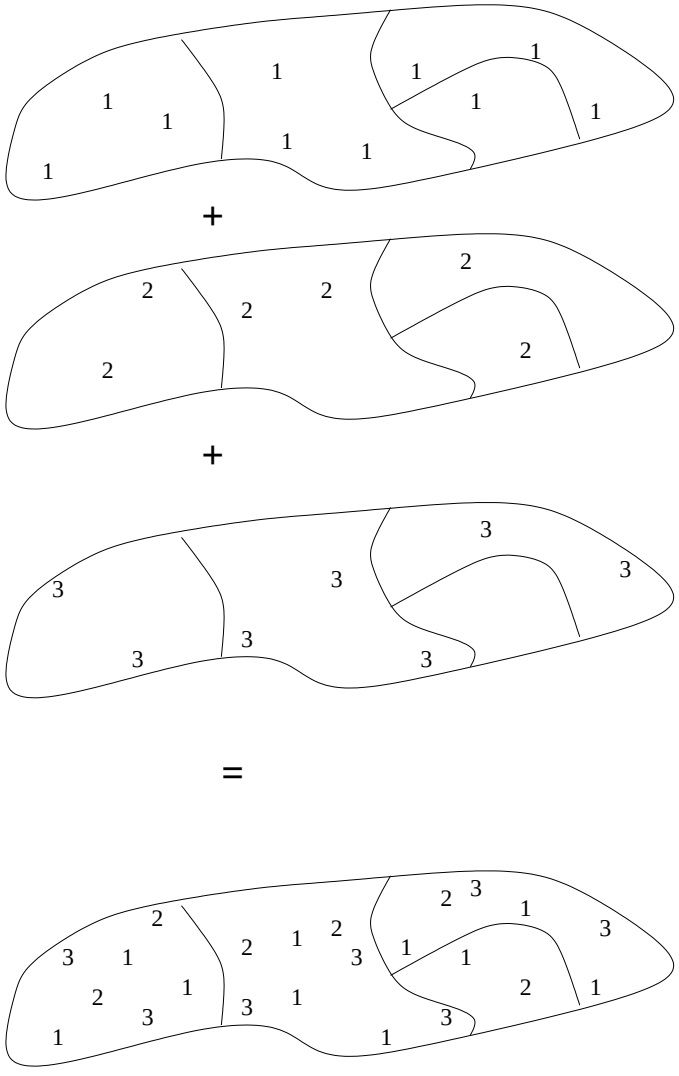
Parallel ILU

Every colour processed in parallel

Sequential time reduced to number of colours

Optimal colour number NP hard

Optimal number not relevant slightly larger number better for convergence



Jones / Plassman algorithm for multicolours

- Give every variable a random value
- Find points with a larger value than all of their neighbours
- $\Rightarrow$ independent set
- Remove this set, repeat.

Parallel generation of multicolouring;
small number of colours, larger than ‘optimal’

BlockSolve95 code (accessible from Petsc)

Analytic methods

Idea:

- Partitioning into physical subdomains
- Elliptic problem on whole domain $\Rightarrow$ elliptic problem on subdomain
- Automatic parallelism of subdomains
- Use existing methods on subdomains
- Interesting theoretical properties

Fast solvers

Use of Fast Fourier or ADI solver on regular grid domains.

Example: Fast Fourier or Alternating Direction Implicit.

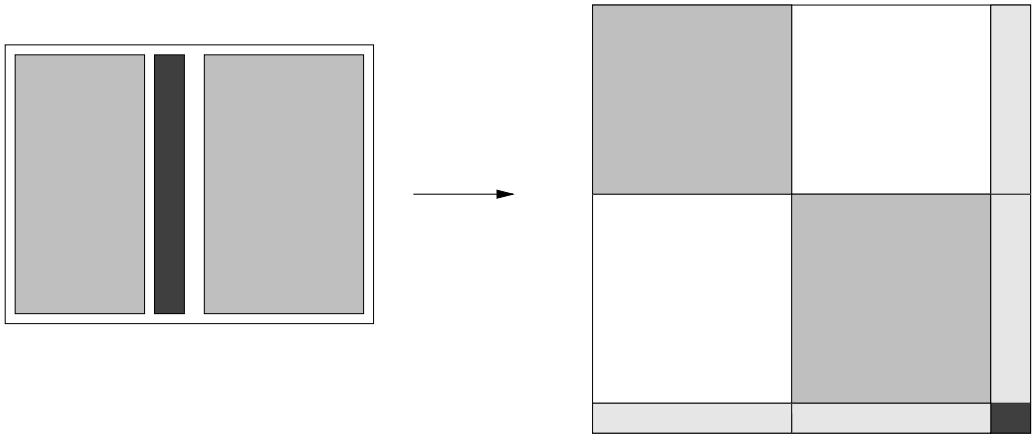
In essence: find separable problem approximating the original one.

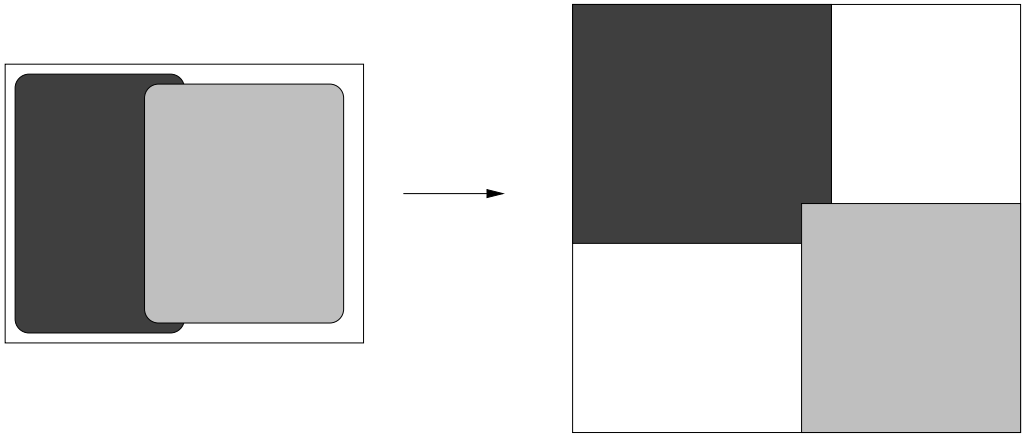
Theoretically: $\#it = O(1)$.

Connection between subdomains

- Explicit interface system (Domain decomposition, or Schur complement method)
- Overlap of subdomains (Schwarz method)
- Zero overlap: block Jacobi
  no asymptotic improvement

Domain / matrix





Explicit approximation of the inverse

- Preconditioners considered so far: construct $M \approx A$ s.t. $Mx = y$ easily solved.
- Approximations of the inverse: construct $M \approx A^{-1}$.

Big advantage: application is very parallel.

Several approaches to construction:

- Minimise $\|I - AM\|$ with a given sparsity pattern for M. Theoretically doubtful, may work in practice.
- Let M be of the form $M = LDL^t$. Very promising.

.

Miscellaneous topics

Order improvement

- Unpreconditioned:
  $\#it \sim \sqrt{\kappa(A)}$, and $\kappa(A) \sim h^{-2}$.
- Incomplete factorisation:
  same order but smaller constant.
- Modified incomplete factorisation:
  $\kappa(AM^{-1}) \sim h^{-1}$.
- Schur complement system:
  $\kappa(S) \sim h^{-1}$.
- Multigrid and Schwarz methods:
  $\kappa(AM^{-1}) \sim \log h^{-1}$ or polylog, possibly even $O(1)$.

.

Symmetry in preconditioners

For symmetric system, preconditioner needs to be symmetric.

Unsymmetric system:

- Incomplete factorisations can break down or become indefinite.
- Use preconditioner based on symmetric part of the matrix:
  $(A + A^t)/2$.

  Properties of symmetric method are often preserved.

Are peaks bad?

Bi-CG type processes (Bi-CG, CGS, ...):

$$x_i = x_{i-1} + \alpha_i p_i \qquad r_i = r_{i-1} - \alpha_i A p_i$$

errors in x_i noeffect on r_i

In finite precision:

$$r_i = r_{i-1} - \alpha_i A p_i - \alpha_i \Delta_A p_i, \qquad |\Delta_A| \leq n_A \xi |A|$$

$$r_i - (b - A x_i) = -\sum_{j=1}^{i} \alpha_j \Delta_A p_j$$

$$|\|r_i\|_2 - \|b - A x_i\|_2| \leq 2\, i\, n_A \xi \| |A| \| \|A^{-1}\| \max_j \|r_j\|$$

.

Relation computed/real residual

In finite precision:

$$r_i = r_{i-1} - \alpha_i A p_i - \alpha_i \Delta_A p_i$$

$$|\Delta_A| \leq n_A \xi |A|$$

$$r_i - (b - A x_i) = - \sum_{j=1}^{i} \alpha_j \Delta_A p_j$$

$$|\|r_i\|_2 - \|b - A x_i\|_2| \leq 2\, i\, n_A \xi \| |A| \| \|A^{-1}\| \max_j \|r_j\|$$

.

Reliable updating

from suggestion by Neumaier '94, made for CGS

for $i = 0, 1, 2, \ldots$

$\ldots x_u = x_u + \alpha_i p_i, \qquad r = r - \alpha_i A p_i \ldots$

if $(\|r\| < \|\bar{r}\| \wedge i - i_{prev} < m_i)$

$\bar{x} = \bar{x} + x_u, \qquad \bar{r} = r = b - A\bar{x} \qquad x_u = 0$

endif

if $\|r\| \approx \xi \|r_0\|$: $r_i \approx b - A x_i$

.

List of iterative methods

- Jacobi, SOR, SSOR: limited applicibility, mostly to M matrices, extremely simple to implement.
- Conjugate Gradients: for symmetric systems, limited storage requirement, easy to implement.
- MinRes: variant of CG with optimal convergence, for symmetric systems, limited storage, implementation more complicated.
- Orthodir, Orthores, Orthomin: for nonsymmetric systems, does not need transpose, unbounded storage for theoretical convergence, in practice truncation or restart which influences convergence behaviour.
- GMRES: optimal variant of Ortho methods; inherits other properties.
- BiCG: for nonsymmetric systems, needs transpose, limited storage, easy to implement, can break down; robust implementation is hard.
- QMR: variant of BiCG, for nonsymmetric systems, needs transpose,

quasi-optimal convergence, limited storage, can break down, easy to implement, but robust implementation is hard.

- CGS: for nonsymmetric systems, does not need transpose, theoretically high convergence speed, but in practice erratic, implementation simple.
- BiCGstab: for nonsymmetric systems, does not need transpose, more regular convergence than CGS, implementation can be sophisticated.
- TFQMR: optimal variant of CGS.

List of Preconditioners

- Jacobi, SOR, SSOR: trivial to compute, easy to apply, limited improvement, only Jacobi is parallel.
- Incomplete factorisations: take some effort to compute, breakdown problem. Large improvement possible.
- Fast solvers: only on regular domains, hard in parallel.
- Block Jacobi, Additive Schwarz: fully parallel, construction and application reduced to single processor problem.
- Block SSOR, Multiplicative Schwarz: constant parallelism, other properties as previous item.
- Schur system domain decomposition: mostly fully parallel, single processor properties for subsystem; interface system tricky to handle.
- Multilevel/grid methods: hard to construct, potentially optimal, very parallel.

Recommended reading

- The ‘Templates’ book.

  A short, cheap, practical introduction to iterative methods, preconditioners, data structures.
  `http://www.netlib/org/templates`

- Saad’s book.

  Covers almost all iterative methods and preconditioners. Practical sections on derivation of the linear systems, data storage, and parallelism. `http://cs.umn.edu/~saad/`