Chap 6 - Iterative Methods

Part 9 - Preconditioning to Accelerate Iterative Methods

Goal: change $Ax=b$ to

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

(1) $M^{-1}A$ much better conditioned than A so convergence faster

(2) still cheap to multiply by M^{-1}

Straightforward to apply to algorithms like GMRES, only multiply by $(M^{-1}A)$, no restrictions on $M^{-1}A$

CG: requires $M^{-1}A$ to be spd, if used straightforwardly:

How to precondition CG?

if M also spd. $M = Q \Lambda Q^T$

and $M^{1/2} = Q \Lambda^{1/2} Q^T$ so $(M^{1/2})^2 = M$

Imagine applying CG to

$$(*)\quad (\underline{M^{-1/2}} A \underbrace{M^{-1/2} M^{1/2}}_{=I} x = \underline{M^{-1/2}} \cdot b$$

also $M^{-1/2} A M^{-1/2} = M^{1/2}(M^{-1}A)M^{-1/2}$

so $M^{-1/2}AM^{-1/2}$ and $M^{-1}A$ similar

$\Rightarrow$ if one well conditioned, so is other

Apply CG implicity to (*) without needing $M^{1/2}$

Preconditioned CG:

$k=0$, $x_0=0$, $r_0=b$, $p_1=M^{-1}b$, $y_0=M^{-1}r_0$

repeat

$k=k+1$

$z = A\cdot p_k$

$v_k = (y_{k-1}^T r_{k-1})/(p_k^T z)$

$x_k = x_{k-1} + v_k \cdot p_k$

$r_k = r_{k-1} - v_k \cdot z$

$y_k = M^{-1} r_k$

$\mu_{k+1} = (y_k^T r_k)/(y_{k-1}^T r_{k-1})$

$p_{k+1} = y_k + \mu_{k+1} p_k$

until $\|r_k\|_2$ small enough

Thm 6.9 shows that above alg implicitly running CG on (*)

How to pick preconditioner M^{-1}

Goals!

(1) cheap to multiply by M^{-1}
(2) $M^{-1}A$ (much) better conditioned than A

Choice 1: $M = I$, (1) ok, (2) not
Choice 2: $M = A$, (1) hard (2) perfect
Big design space of M, depend on structure of A

(1) If A has widely varying A_{ii}, choose $M = \mathrm{diag}(A_{11}, A_{22}, \ldots, A_{nn})$
Called Jacobi preconditioning:
$M^{-1}A = I - R_j$
so if $\rho(R_j)$ small, than $M^{-1}A$ well conditioned

(2) More generally $M = \mathrm{diag}(A_{11}, A_{22}, \ldots, A_{kk})$ where each A_{ii} is a diagonal block of some size. Large blocks $\Rightarrow$ M^{-1} more expensive but faster convergence
Eg A_{ii} may correspond to different

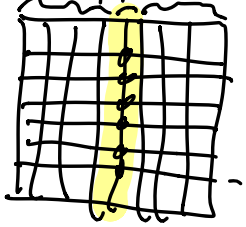
physical domains of a simulation, where each domain may be "easier" by itself (similar idea below for domain decomposition): Also block Jacobi preconditioning. Note by replacing A by PAP^T, P a permutation, each A_{ii} can correspond to any subset of rows and columns.

(3) "Incomplete Cholesky"
compute an approximate factorization of spd $A \sim LL^T = M$ where we limit L in some way, eg. same sparsity pattern as $A \Rightarrow$ multiplying by $M^{-1} = (LL^T)^{-1} = L^{-T}L^{-1}$ cheap, cost 2 triangular solves with matrices as sparse as A.
Similar idea for general A, called "incomplete LU" or ILU

(4) One or more V cycles of multigrid as preconditioner, given a suitable A or A_{ii} in Block Jacobi

(5) Domain Decomposition (see sec 6.10) idea is to use appropriate preconditioners for each block A_{cc} and their "connections" interface

Ex:



$n \times n$ mesh
number mesh points
steel first
concrete second
interface third

$$A = \left[\begin{array}{c|c|c} A_{ss} & 0 & A_{si} \\ \hline 0 & A_{cc} & A_{ci} \\ \hline A_{is} & A_{ic} & A_{ii} \end{array}\right]$$

$\dim(A_{ss}) = \dim(A_{cc}) = n(n-1)/2$ (n odd)

$\dim(A_{ii}) = n$

factor A:

$$A = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A_{is}A_{ss}^{-1} & A_{ic}A_{cc}^{-1} & I \end{bmatrix} \cdot \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & S \end{bmatrix} \cdot$$

Schur Complement

$$\begin{bmatrix} A_{ss} & 0 & A_{si} \\ 0 & A_{cc} & A_{ci} \\ 0 & 0 & I \end{bmatrix} \quad \text{where} \quad S = A_{ii} - A_{is}A_{ss}^{-1}A_{si} - A_{ic}A_{cc}^{-1}A_{ci}$$

$$A^{-1} = \begin{bmatrix} A_{ss}^{-1} & 0 & -A_{ss}^{-1}A_{si} \\ 0 & A_{cc}^{-1} & -A_{cc}^{-1}A_{ci} \\ 0 & 0 & I \end{bmatrix} \cdot \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & S^{-1} \end{bmatrix}$$

$$\cdot \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -A_{is}A_{ss}^{-1} & -A_{ic}A_{cc}^{-1} & I \end{bmatrix}$$

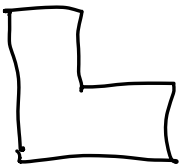
Goal: approximately multiply by A^{-1}

Assume we have good preconditioners for A_{ss} and A_{cc} $\Rightarrow$ easy to approximately multiply by $(A_{is}A_{ss}^{-1})x$

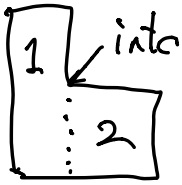
$= A_{is}(\underbrace{A_{ss}^{-1}x}_{\text{use preconditioner for } A_{ss}})$

What about multiplying by S^{-1}?
S much smaller than A_{ss} and A_{cc}
(n vs $n(n-1)/2$) by expensive
to compute explicitly: approximately
but easy to multiply by S^{-1} using
preconditioners for A_{ss} and A_{cc}
Give ability to multiply by S
(approximately) can multiply by S^{-1}
(approximately) using, say, CG,
Note that CG should be fast
because S often much better
conditioned than A_{ss} or A_{cc}
Eg: For Poisson $\text{cond}(A_{ss}) = \text{cond}(A_{cc})$
$= O(n^2)$ vs $O(n)$ for S
All this generalized to more than
2 domains (steel, concrete, wood, ...)

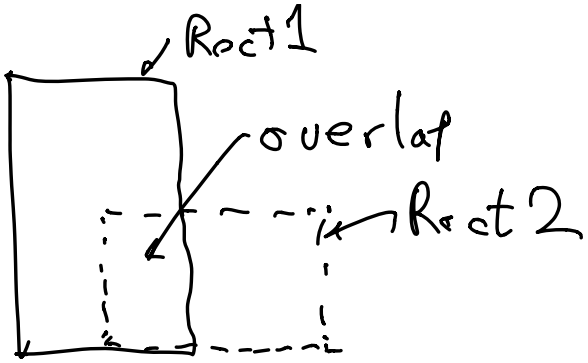
This technique called "nonoverlapping" domain decomposition because regions disjoint! Similar idea for overlapping domains: Eg 2D Poisson on 2 overlapping rectangles:
Eg L shaped domain



Good solvers for rectangles:
What's best way to use them for unions of rectangles?
Could use nonoverlapping DD:

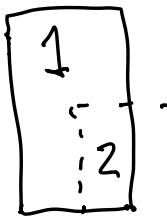


But can converge faster by letting rectangles overlap



Intuition: moves information faster at each step. Same motivation as using multigrid, so info moves more than one edge in graph per step.

Number mesh points as follows



1 = just in Rect 1
2 = overlap
3 = just in Rect 2

$$A = \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & A_{23} \\ 0 & A_{32} & A_{33} \end{bmatrix}$$

no interactions between points just in Rect 1 and just in Rect 2

$$M^{-1} = \begin{bmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}^{-1} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \begin{pmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{pmatrix}^{-1} \end{bmatrix}$$

Name: Additive Schwartz Preconditioner (overlapping Block Jacobi)

Recall Jacobi accelerated by GS (use most recently updated solution values)

Can use same idea here, by using most recent data in overlapping region → called Multiplicative Schwartz

- Also possible to combine this with coarser grid approximation à la multigrid
See sec 6.10 for details,
See website for links to software