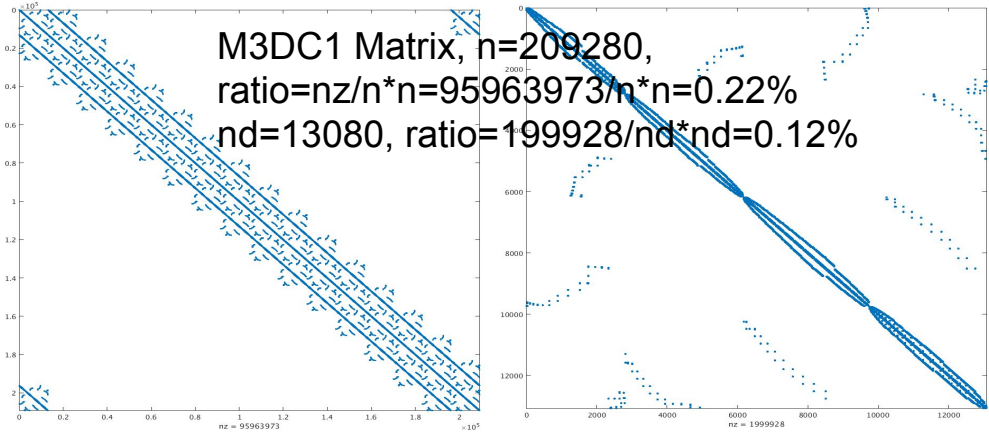
Solvers for Sparse Linear Systems

Jin Chen

CPPG, Dec 10 2018

Sparse Matrices, Source

- ➔ Differential Equation
- ➔ Integral Equation
- ➔ Structural Mechanics
- ➔ …….



Finite difference, Finite elements, Finite volume ⇒ *large* and *sparse* matrix systems

$\mathcal{A}X=b$

Singular / Non-Singular
non-symmetry/Symmetry
Symmetric Positive Definite /

Enough zeros that it pays to take advantage of them to avoids storing the numerically zero entries of $\mathcal{A}$
Memory usage: far less
Memory access: much faster access
Flop operations: significantly reduction

The choice of a solver will often depend on the structure of $\mathcal{A}$;
Basic Matrix Operations for sparse matrices: BLAS 1, BLAS 2, BLAS 3.

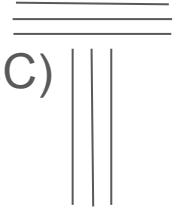
Goal:

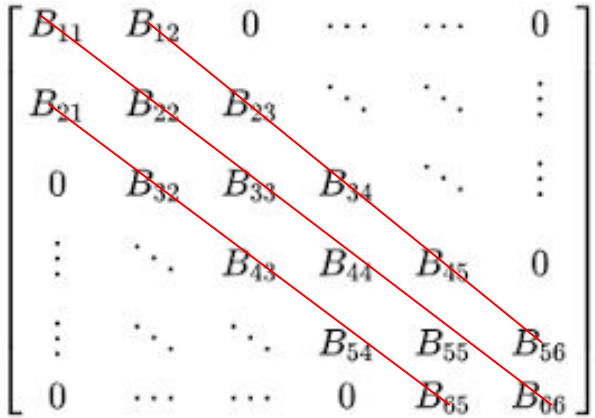
- ➔ Overview of the data structure, algorithms, and solver packages of sparse direct solvers and sparse iterative solvers
- ➔ Working knowledge of how best to use them, mainly in parallel
- ➔ Other Parallel Strategies
- ➔

Sparse Matrix, Storage Arrangement

Sparse compact matrix Data structure (distributed):

- ➔ Compressed Sparse Row (CSR)
- ➔ Compressed Sparse Column (CSC)
- ➔ Diagonal Format
- ➔ Triplet (i,j,v)





memory size and efficient access (*direct/indirect access, gather/scatter*)

floating point operations reduction

Sparse Matrix, Algorithms

Householder Reflection
Givens Rotation
Gram-Schmidt Process

1. Direct solve: $\mathcal{A}=\mathcal{LU}$, $\mathcal{QR}$, $\mathcal{LL}^{\mathcal{T}}$, $\mathcal{LDL}^{\mathcal{T}}$
2. Iterative solve ($\mathcal{M}^{-1}\mathcal{A}x=\mathcal{M}^{-1}b$), Preconditioned Iterative Solve ($\mathcal{P}\mathcal{A}x=\mathcal{P}b$)
3. Hybrid (Iterative + Direct)

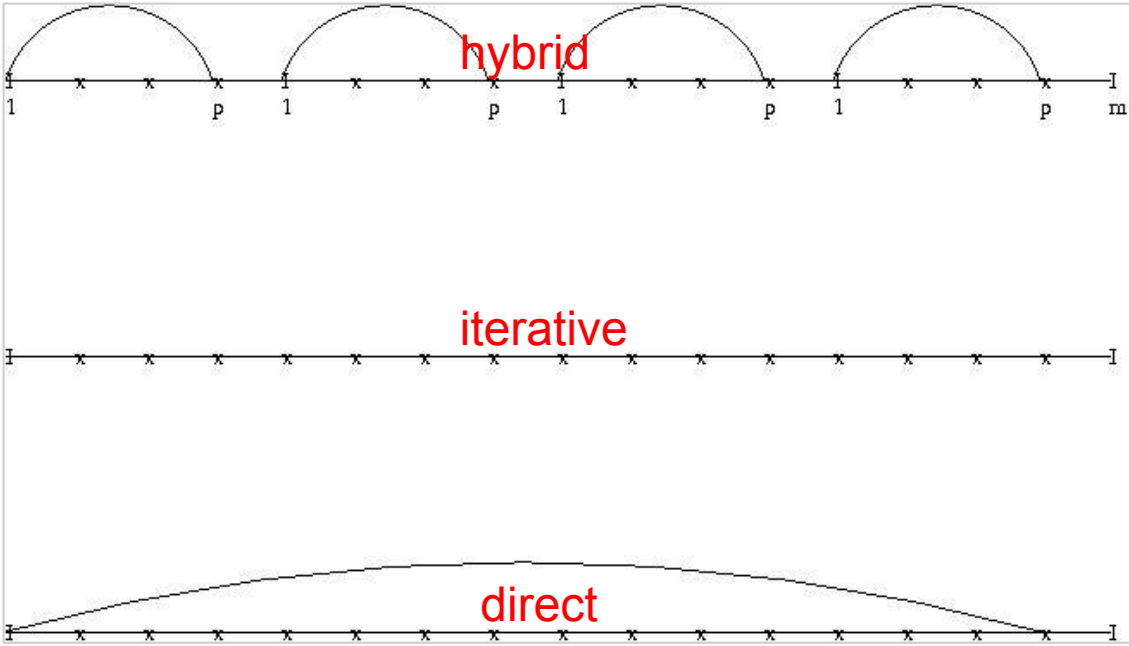
Convergence Rate

$$\frac{\partial u}{\partial t} = \frac{\partial^2 y}{\partial x^2}$$

$$AX = b \qquad \begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & \ddots & \ddots & & \\ & & \ddots & \ddots & 1 & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{pmatrix}$$

$$M\tilde{X} = NX + b$$

$$\tilde{X} = M^{-1}NX + b$$



$$\| M^{-1}N \| : \begin{cases} = \dfrac{2r}{1+2r} & p = 1 \quad \text{iterative} \\ \left(\dfrac{2r}{1+24+\sqrt{1+4r}} , \dfrac{r}{1+r} \right) & 1 < p < m \quad \text{hybrid} \\ = 0 & p = m \quad \text{direct} \end{cases}$$

1. Sparse Direct Solve, Steps of $\mathcal{A}=\mathcal{LU}$, $\mathcal{QR}$, $\mathcal{LL}^{\mathcal{T}}$, $\mathcal{LDL}^{\mathcal{T}}$

1. Preordering/Permutation is applied to minimize memory usage, flop count, increase numerical stability (Fill-reducing ordering)
2. A symbolic factorization of $\mathcal{PAQ}$ is performed.
3. The numerical factorization (actual factors $\mathcal{L}$ and $\mathcal{U}$ are formed) with dynamical pivot (maintain numerical accuracy) is processed.
4. The forward and backward triangular sweeps are executed for each $\mathcal{b}$.

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & & \vdots \\ 0 & 0 & u_{33} & & \vdots \\ \vdots & & & \ddots & \vdots \\ \vdots & & & & \vdots \\ 0 & \cdots & \cdots & 0 & u_{nn} \end{bmatrix}$$

u_ii=1

$$L = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & & \vdots \\ l_{31} & l_{32} & l_{33} & & \vdots \\ \vdots & & & \ddots & \vdots \\ \vdots & & & & 0 \\ l_{n1} & \cdots & \cdots & \cdots & l_{nn} \end{bmatrix}$$

Supernodal methods
Frontal methods
Multifrontal methods

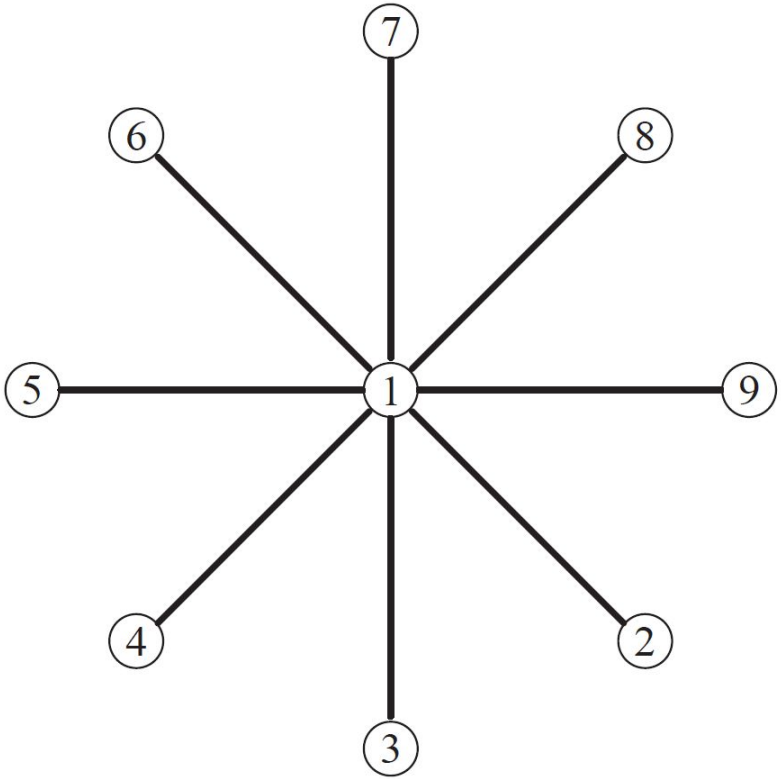
1.1 Sparse Directive Solve, Parameters: Permutations reorder rows / columns)

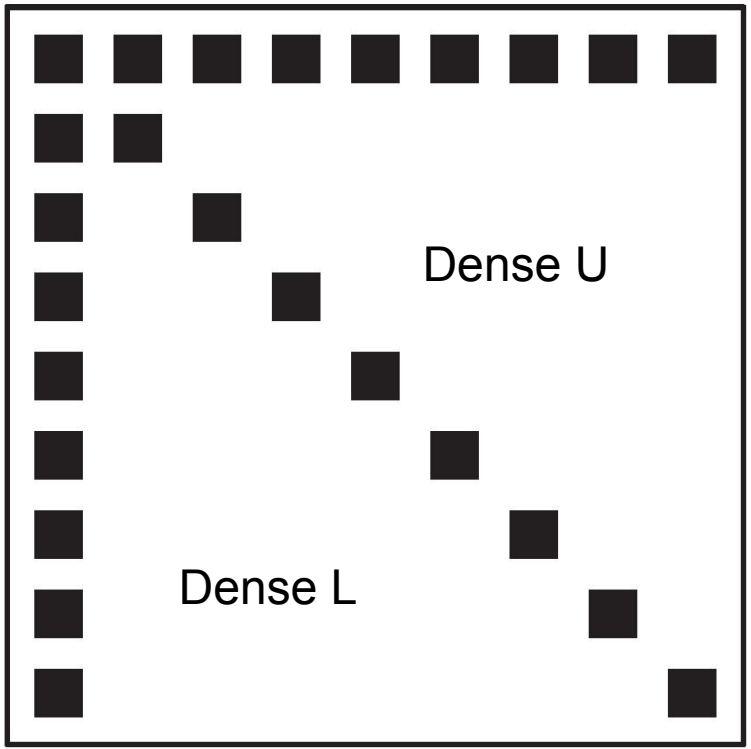
$$\begin{pmatrix} a11 & a15 & a13 & a14 & a12 & a16 \\ a51 & a55 & a53 & a54 & a52 & a56 \\ a31 & a35 & a33 & a34 & a32 & a36 \\ a41 & a45 & a43 & a44 & a42 & a46 \\ a21 & a25 & a23 & a24 & a22 & a26 \\ a61 & a65 & a63 & a64 & a62 & a66 \end{pmatrix}$$

Fill-in is the introduction of new nonzeros in the factors $\mathcal{L}/\mathcal{U}$ that do not appear in the corresponding positions in the matrix $\mathcal{A}$ being factorized.

A sparse matrix must typically be permuted either before or during $\mathcal{L}/\mathcal{U}$ factorization, either for reducing fill-in or for increasing numerical stability, or often for both.

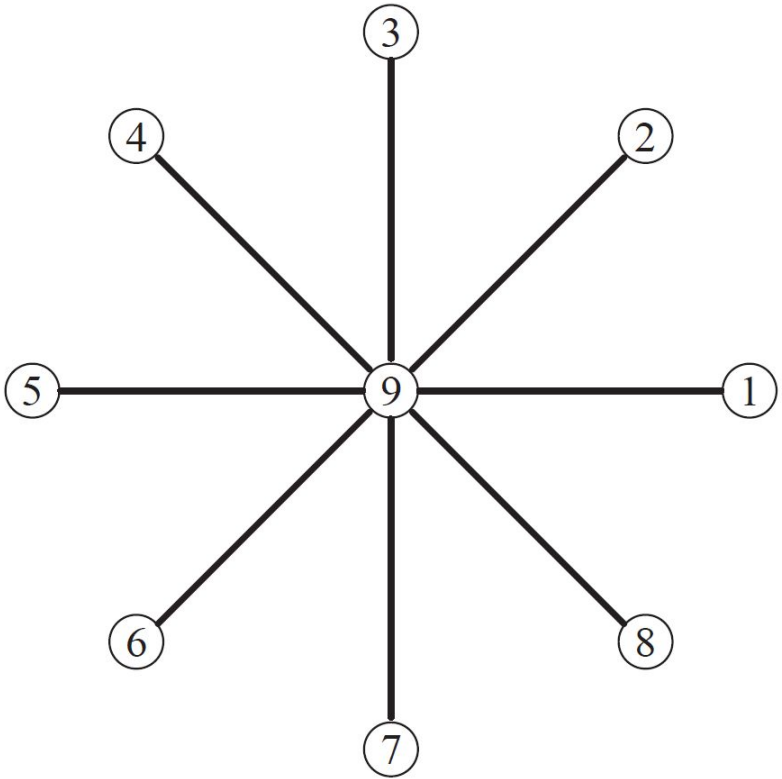
Sparse Directive Solve, Parameters: Permutation and Reordering

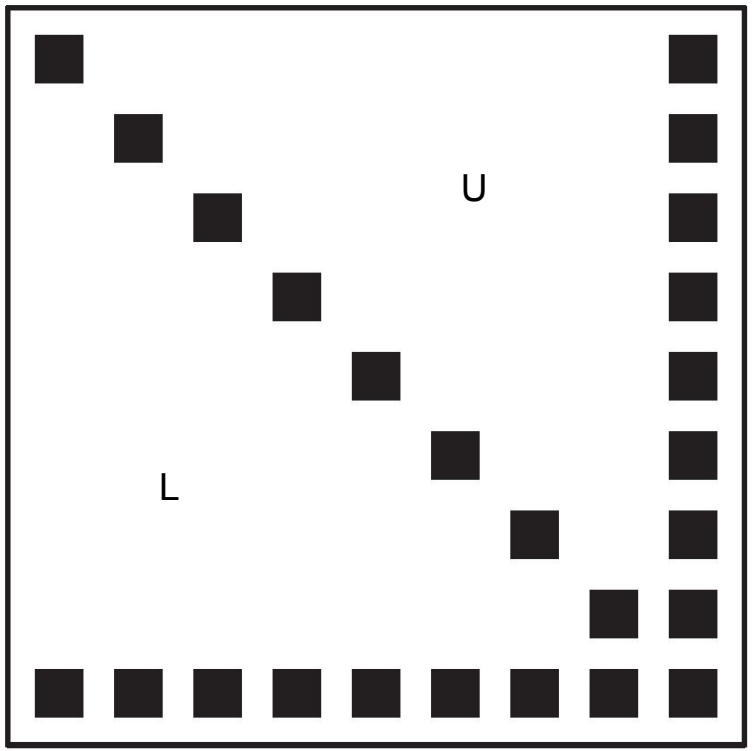




Sparse Directive Solve, Parameters (Reverse Cuthill-McKee Ordering)

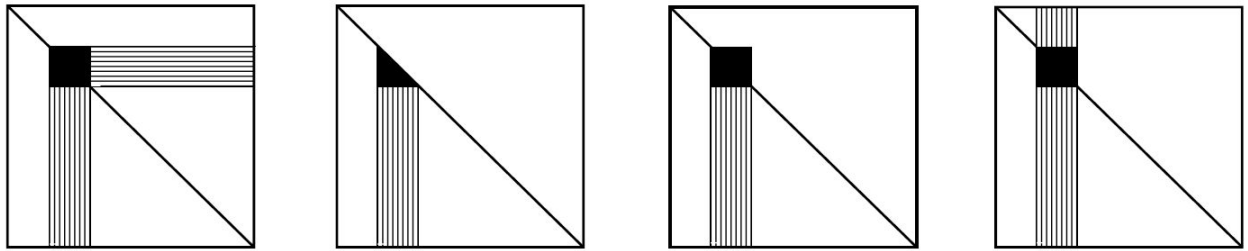
LU:no fill-in introduced





1.2 Sparse Directive Solve : supernodal

Adjacent columns in the factors with identical nonzero pattern (or nearly identical) are glued together to form supernodes. Greatly improves memory traffic and allows for computations to be done in **dense submatrices**, each representing a supernodal column.



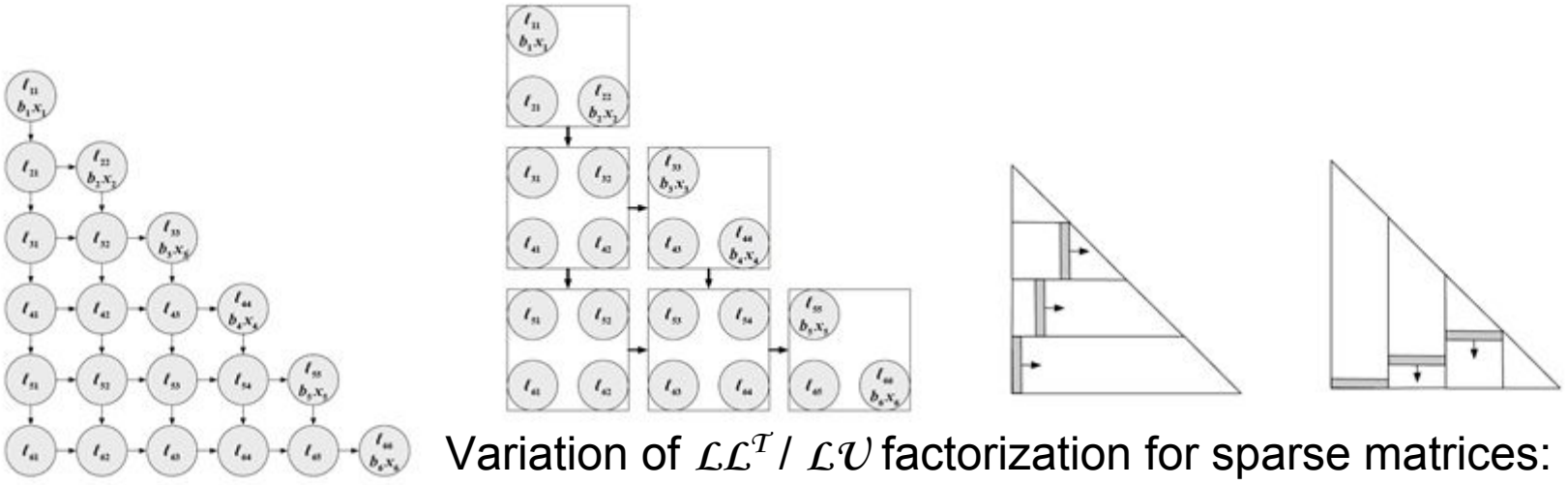
⇒ Replace most of the irregular gather/scatter operations with regular operations.

Variation of $\mathcal{L}\mathcal{L}^{\mathcal{T}}$ / $\mathcal{L}\mathcal{U}$ factorization for sparse matrices: SuperLU

1.3 Sparse Directive Solve : frontal & multifrontal

Frontal: a single **dense submatrix** holds the part of the sparse matrix actively being factorized, and rows and columns come and go during factorization.

Multifrontal: the matrix is represented not by one frontal matrix, but by many of them, all related to one another via the assembly tree.



Variation of $\mathcal{L}\mathcal{L}^{\mathcal{T}}$ / $\mathcal{L}\mathcal{U}$ factorization for sparse matrices: Mumps

PETSc Sparse Director Solvers

Portable, Extensible Toolkit for Scientific Computation

-ksp_type preonly
-pc_type lu
-pc_factor_mat_solver_type mumps

-ksp_type preonly
-pc_type lu
-pc_factor_mat_solver_type superlu_dist

-pc_factor_mat_ordering_type <rcm,...>

https://www.mcs.anl.gov/petsc/index.html

essl
L lusol
matlab
mumps
cholesky
superlu_dist
umfpack
cholmod
clique
klu
elemental
mkl_pardiso
mkl_cpardiso
pastix
bas
cusparse

2. Sparse Iterative Solve

- Traditional iterative method: Jacobi, GS, SOR, SSOR, and their block versions

- Projection iterative method: Krylov subspace as search space, (named after Russian applied mathematician and naval engineer Alexei Krylov, who published a paper about it in 1931.)

  Residual projection: General matrix, ARNOLDI, GMRES

  Error projection: SPD matrix, LANZCOS, CG

$$\mathcal{K}_m(\mathcal{A}, r_0) = span\{r_0, \mathcal{A}r_0, \mathcal{A}^2 r_0, \ \ldots, \mathcal{A}^{m-1} r_0\}$$

  Bi-orthogonalization projection: Nonsymmetric matrix, LANCZOS, BCG, BCGStab

$$\mathcal{K}_m(\mathcal{A}, r_0) = span\{r_0, \mathcal{A}r_0, \mathcal{A}^2 r_0, \ \ldots, \mathcal{A}^{m-1} r_0\}$$
$$\mathcal{K}_m(\mathcal{A}^T, r_0) = span\{r_0, \mathcal{A}^T r_0, \mathcal{A}^{T,2} r_0, \ \ldots, \mathcal{A}^{T,m-1} r_0\}$$

Sparse Iterative Solve, Parallel

1. Extracts parallelism whenever possible from standard implementation.
2. Develop alternative implementation which have enhanced parallelism.

Direct solvers are being used in conjunction with iterative solvers to develop robust preconditioners:

Efficient + Robust (conflict)

Condition Number:

$$\frac{\| \tilde{x} - x \|}{\| x \|} \leq \varepsilon \| A \| \| A^{-1} \| C + o(\varepsilon)$$

$$\kappa(A) = \| A \| \| A^{-1} \|$$

2.1 Sparse Iterative Solve, Relaxation

Annihilate one or a few components of the residual vector $r=b-Ax$:

Jacobi, Gauss-Seidel, SOR, SSOR, and their block types
(non-overlapping & overlapping)

These techniques are rarely used separately.

However, when combined with the more efficient methods, such as direct solve, they can be quite successful.

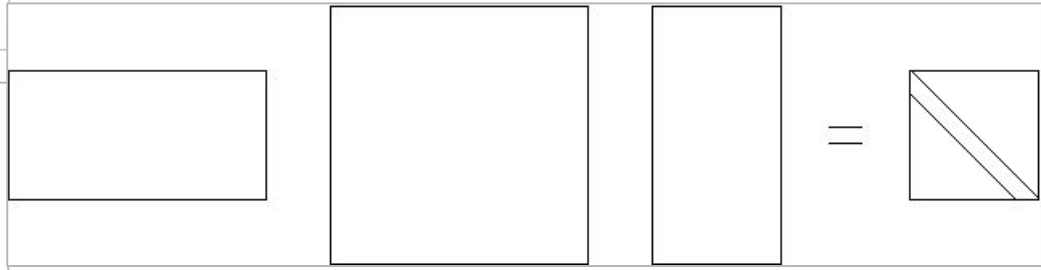
For any splitting $A=M-N$, write $Mx=Nx+b$
Define the iterative sequence $Mx^{m+1}=Nx^{m}+b$

Use $A=D-E-F$

Jacobi	$M = D$	$R := J = I - D^{-1}A$
Relaxed Jacobi	$M = \frac{1}{\omega}D$	$R = I - \omega D^{-1}A$
Gauss-Seidel	$M = D - E$	$R := \mathcal{L}_1 = I - D^{-1}A$
SOR	$M = \frac{1}{\omega}D - E,$	$R := \mathcal{L}_\omega = (D - \omega E)^{-1}((1-\omega)D + \omega F)$
Richardson	$M = \frac{1}{\rho}I$	$R = I - \rho A$

2.2 Projection Methods: Extract an approx sol from $\mathcal{K}_m$ of R^n

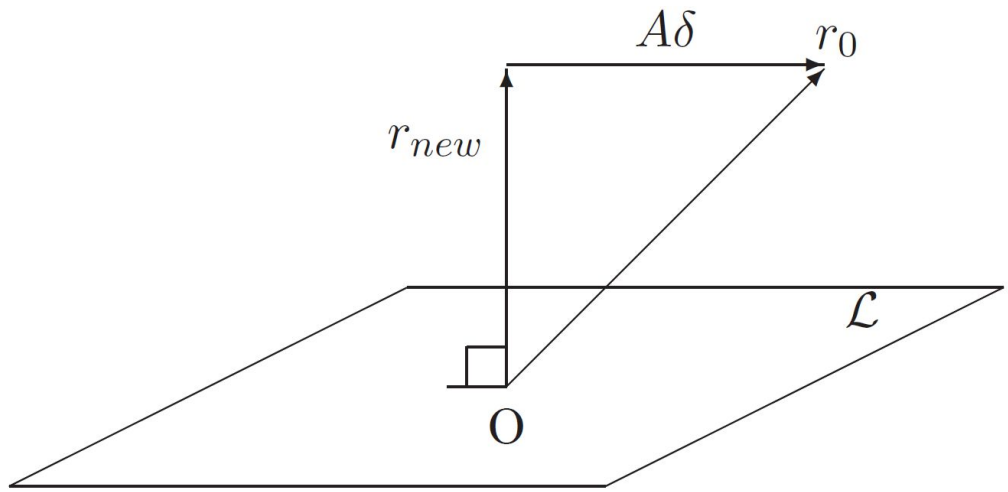
$$Find\ \tilde{x} \in x_0 + \mathcal{K}(m), such\ that\ r = b - A\tilde{x}\ \perp \mathcal{L}(m)$$



Three types:

$\mathcal{L}=\mathcal{K}$, error projection
$\mathcal{L}=\mathcal{AK}$, residual projection
$\mathcal{L}=\mathcal{A}^T\mathcal{K}$, bi-orthogonal residual projection

$\mathcal{V}=[v_1, \dots, v_m]_{nxm}$,
$\mathcal{W}=[w_1, \dots, w_m]_{nxm}$,
$x=x_0+\mathcal{V}y$,
$\mathcal{W}^T\mathcal{A}\mathcal{V}y=\mathcal{W}^T r_0$, $(nxn \Rightarrow mxm, m<<n)$



2.2.1 Projection methods

$m=1$, one dimensional case
$\mathcal{K}=span\{v\},\ \mathcal{L}=span\{w\}$

Steepest Descent: $v=r,\ w=r$, minimizes $f(x)=(\mathcal{A}(x-x_*),(x-x_*))$ in the direction $-\nabla f$

Minimal Residual: $v=r,\ w=\mathcal{A}r$, minimizes $f(x)=(b-\mathcal{A}x,\ b-\mathcal{A}x)$ in the direction r

Residual Norm Steepest Descent: $v=\mathcal{A}^T r,\ w=\mathcal{A}\mathcal{A}^T r$, minimizes $f(x)=(b-\mathcal{A}x,\ b-\mathcal{A}x)$ in the direction $-\nabla f$

2.2.2

$\mathcal{K}_m(\mathcal{A}, r_0, m)=span\{r_0, \mathcal{A}r_0, \mathcal{A}^2r_0, \ldots, \mathcal{A}^{m-1}r_0\}$

m increases by one at each step of the approximation process

Arnoldi Gram-Schmidt (standard/modified)

Arnoldi Householder, more robust, but more expensive

$\mathcal{L}=\mathcal{K}$, impose $r=b-\mathcal{A}x_m \perp \mathcal{K}$

FOM and restarted FOM: full orthogonalization method

IOM: incomplete (i.e., local) orthogonalization method

DIOM: direct incomplete (i.e., local) orthogonalization method

$\mathcal{L}=\mathcal{A}\mathcal{K}$, minimize $r=b-\mathcal{A}x_m$ *on* $\mathcal{K}$

GMRES and restarted GMRES: generalized minimum residual method

QGMRES: incomplete (i.e., local) orthogonalization GMRES method

DQGMRES: direct incomplete (i.e., local) orthogonalization GMRES method

2.2.3

SPD matrix:

Arnoldi ⇒ Lanczos orthogonalization

FOM ⇒ CG

Hermitian matrix:

GMRES ⇒ CR (conjugate residual)

GMRES: uses an orthogonal basis

CG: uses an $\mathcal{A}$ conjugate basis

CR: uses an $\mathcal{A}^T\mathcal{A}$ conjugate basis

2.3 Projection method: bi-orthogonalization

LANCZOS For nonsymmetric matrices

BCG (BiConjugate Gradient)

CGS (CG Squared, Transpose Free BCG)

BICGSTAB (BiConjugate Gradient Stabilization)

QMR(Quasi Minimal Residual), TFQMG (Transpose Free QMR)

$$\mathcal{K}_m = span\{r_0, \mathcal{A}r_0, \mathcal{A}^2 r_0, \ldots\ldots, \mathcal{A}^{m-1} r_0\}$$

$$\mathcal{L}_m = span\{r_0, \mathcal{A}^{\mathcal{T}} r_0, \mathcal{A}^{\mathcal{T},2} r_0, \ldots\ldots, \mathcal{A}^{\mathcal{T},m-1} r_0\}$$

2.4 Projection Methods, Normal Equation

Normal Equation

$A^TAx=A^Tb$
Minimizes $||b-Ax||_2$
in subspace $K_m(A^TA, A^Tr_0)=span\{A^Tr_0, (A^TA)A^Tr, (A^TA)^2A^Tr_0, ..., (A^TA)^{m-1}A^Tr_0\}$
for over-determined system, CGNR

$AA^Tu=b$, $x=A^Tu$
Minimizes $||x_* - x||_2$,
in subspace $K_m(A^TA, A^Tr_0)=span\{A^Tr_0, (A^TA)A^Tr, (A^TA)^2A^Tr_0, ..., (A^TA)^{m-1}A^Tr_0\}$
for under-determined system, CGNE

Iterative solvers available in PETSc

https://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPType.html

```
#define KSPPREONLY      "preonly"

#define KSPRICHARDSON "richardson"
#define KSPCHEBYSHEV  "chebyshev"

#define KSPGMRES        "gmres"
#define KSPFGMRES       "fgmres"
#define KSPLGMRES       "lgmres"
#define KSPDGMRES       "dgmres"
#define KSPPGMRES       "pgmres"

#define KSPTCQMR        "tcqmr"
#define KSPTFQMR        "tfqmr"

#define KSPCR       "cr"
#define KSPGCR    "gcr"

#define KSPMINRES       "minres"
```

```
#define KSPCG       "cg"
#define KSPGROPPCG    "groppcg"
#define KSPCGNE         "cgne"
#define KSPCGSTCG       "stcg"
#define KSPFCG    "fcg"
#define KSPBCGS         "bcgs"
#define KSPIBCGS        "ibcgs"
#define KSPFBCGS        "fbcgs"
#define KSPFBCGSR       "fbcgsr"
#define KSPBCGSL        "bcgsl"
#define KSPCGS    "cgs"
#define KSPQCG    "qcg"
#define KSPBICG   "bicg"
#define KSPCGLS         "cgls"
```

petsc-3.10.2/src/ksp/ksp/examples/tutorials

3. Preconditioner $\mathcal{M}$

1. similar to $\mathcal{A}$ as much as possible,
2. Inexpensive to apply $\mathcal{M}^{-1}$ to an arbitrary vector.

Left, right, split preconditioning techniques

Left Preconditioner

$$\mathcal{M}^{-1}\mathcal{A}x=\mathcal{M}^{-1}b$$

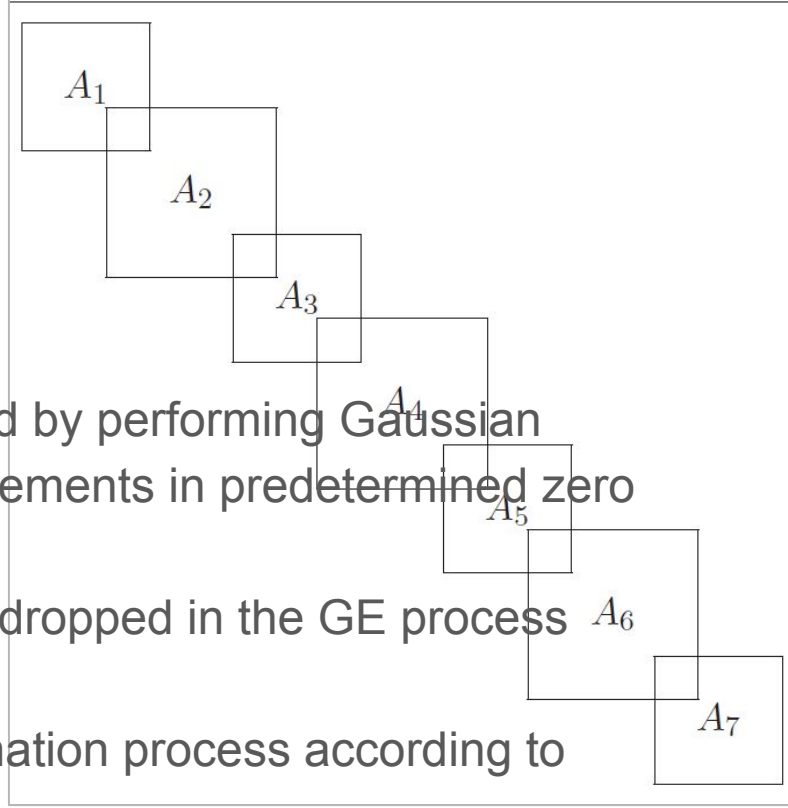
Right Preconditioner

$$\mathcal{A}\mathcal{M}^{-1}u=b,\ u=\mathcal{M}x$$

Split Preconditioner

$$\mathcal{M}=\mathcal{L}\mathcal{U},\ \mathcal{L}^{-1}\mathcal{A}\mathcal{U}^{-1}u=\mathcal{L}^{-1}b,\ x=\mathcal{U}^{-1}u$$

Implement a Preconditioner $\mathcal{M}$



- Scaling all rows of a linear system
  to make the diagonal elements equal to one
- ILU, Building Incomplete LU factorization derived by performing Gaussian Elimination (or its variant) and dropping some elements in predetermined zero pattern or threshold level of fill
- Modified ILU, lumping together all the elements dropped in the GE process and adding them to the diagonal of U
- ILUT(p,tau), Dropping elements in the GE elimination process according to their magnitude rather than their location.
- ILUTP(p,tau,permtol), pivoting A and L/U
- ILUS, taking advantage of symmetry

ICC

- Physics based Preconditioner
- ASM, AMG

Preconditioners available in PETSc

```
#define PCNONE          "none"

#define PCLU            "lu"

#define PCSOR           "sor"
#define PCJACOBI        "jacobi"

#define PCBJACOBI               "bjacobi"
#define PCILU           "ilu"
#define PCICC           "icc"

#define PCASM           "asm"
#define PCGASM          "gasm"

#define PCHYPRE                 "hypre"
```

-factor_levels <k>

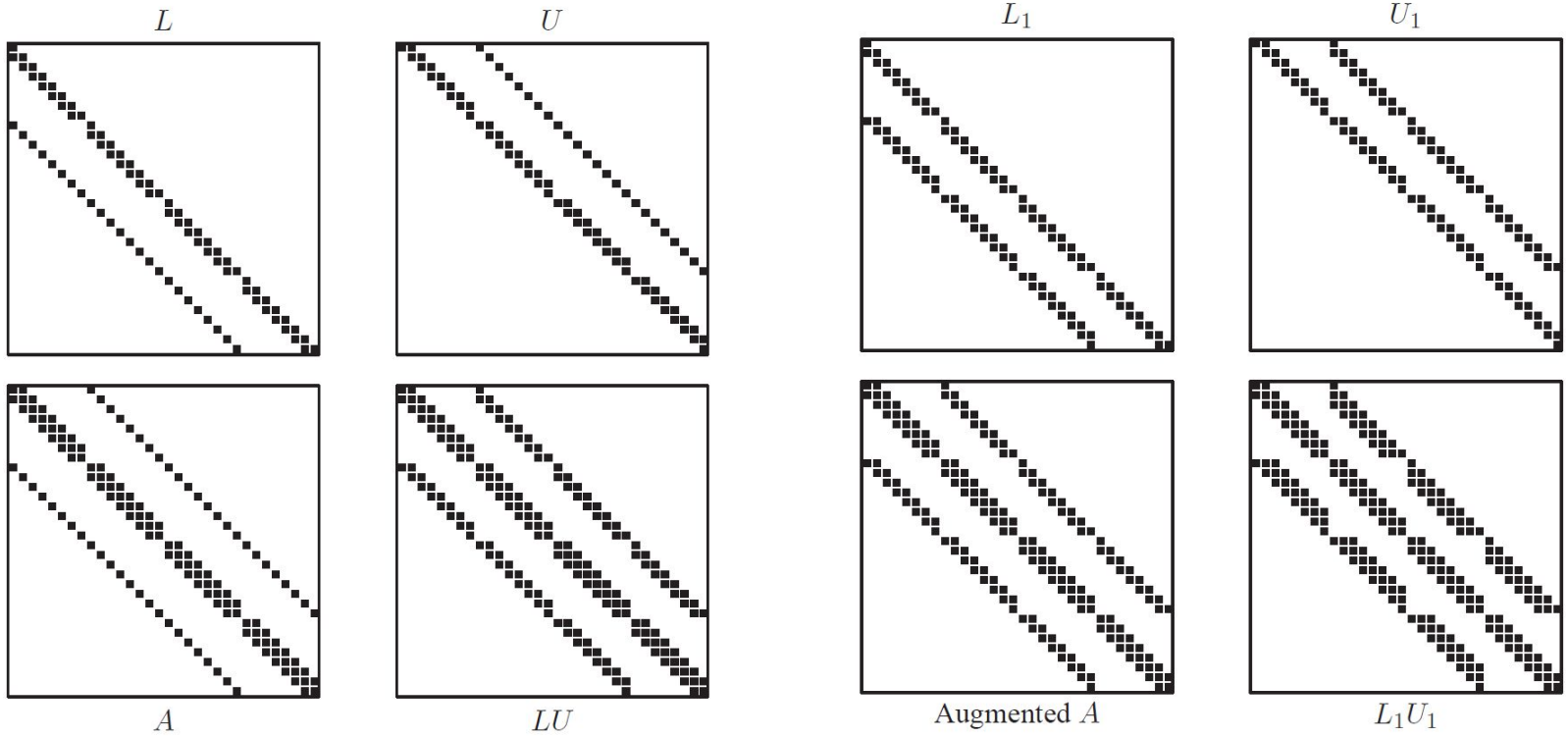
-pc_factor_in_place
-pc_factor_diagonal_fill
-pc_factor_reuse_ordering
-pc_factor_fill <nfill>

-pc_factor_nonzeros_along_diagonal

-pc_factor_mat_ordering_type
<natural,nd,1wd,rcm,qmd>

-pc_factor_pivot_in_blocks

Understand fill-ins



4. Hybrid Solves

Direct solvers are being used
in conjunction with iterative solvers
to develop robust preconditioners.

Key point: robust + efficient

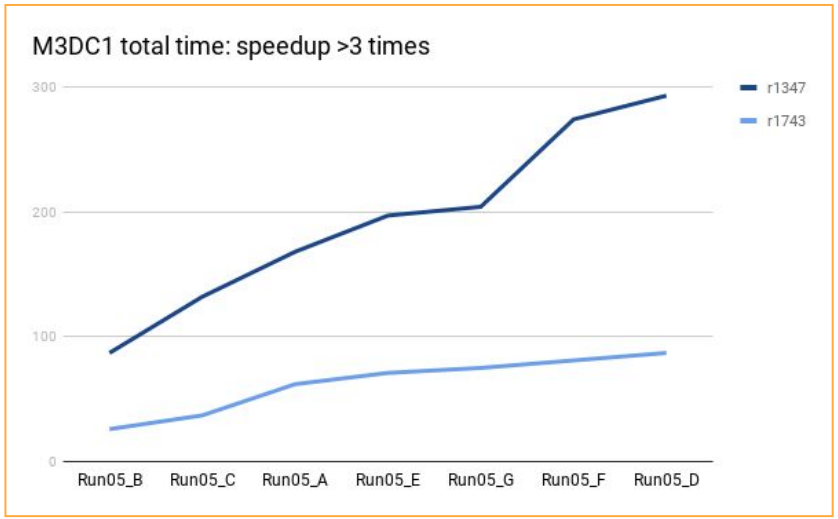
-pc_type bjacobi
-pc_bjacobi_blocks <n>
-ksp_type lgmres/fgmres/gmres
-ksp_gmres_restart <$m+k$>
-ksp_lgmres_augment <k>
-ksp_gmres_classicalgramschmidt
-ksp_gmres_modifiedgramschmidt

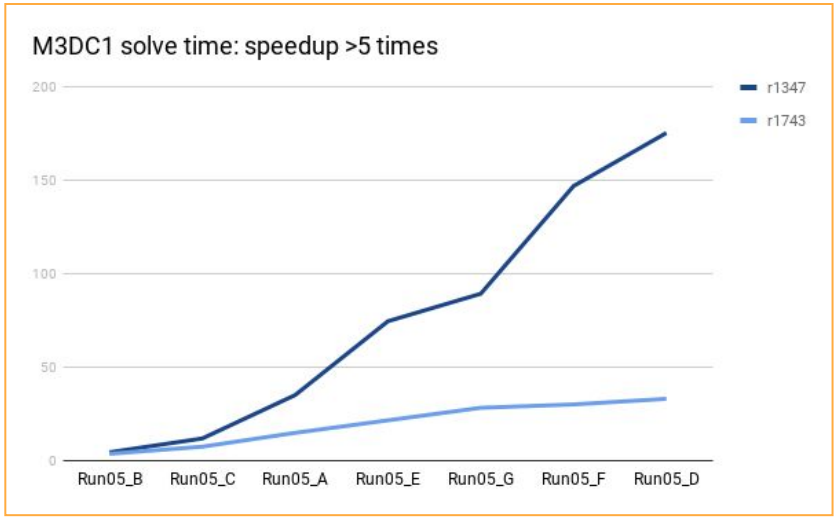
-ksp_rtol <$1.e^{-9}$>
-ksp_atol <$1.e^{-20}$>

-sub_pc_type lu
-sub_pc_factor_mat_solver_type superlu_dist
-sub_ksp_type preonly
-sub_mat_superlu_dist_r <$i=4$>
-sub_mat_superlu_dist_c <$j=8$>
-sub_mat_superlu_dist_rowperm NATURAL

Acceleration Tricks

- ➔ Preconditioner
- ➔ Restart / Double orthogonalization / truncated orthogonalization
- ➔ Preconditioner + Iterative + Direct solve for dense sub-matrices





Summary

1. Direct solver: superlu, mumps
2. Iterative solver: gmres, cg
3. Preconditioner: ilu, icc,asm,amg
4. Hybrid Solver