Lecture 8: Fast Linear Solvers (Part 7)

Modified Gram-Schmidt Process with Reorthogonalization

- $v_{k+1} = Av_k$
  for $j = 1, \ldots, k$
  $h_{jk} = v_{k+1}^T v_j$
  $v_{k+1} = v_{k+1} - h_{jk} v_j$
- $h_{k+1,k} = \|v_{k+1}\|_2$
- If loss of orthogonality is detected
  For $j = 1, \ldots, k$
  $h_{tmp} = v_{k+1}^T v_j$
  $h_{jk} = h_{jk} + h_{tmp}$
  $v_{k+1} = v_{k+1} - h_{tmp} v_j$
- $h_{k+1,k} = \|v_{k+1}\|_2$
- $v_{k+1} = v_{k+1} / \|v_{k+1}\|_2$

Test Reorthogonalization

If $\left|\left|Av_k\right|\right|_2 + \delta \left|\left|v_{k+1}\right|\right|_2 = \left|\left|Av_k\right|\right|_2$ to working precision.
$\delta = 10^{-3}$

Householder Arnoldi

- In Arnoldi algorithm, the column vectors of a matrix to be orthonormalized are not available ahead of time.
- In stead, the next vector is Av_j, where v_j is current basis vector.
- In the Householder algorithm, an orthogonal column v_i is obtained as $H_1 \ldots H_i e_i$.

ALGORITHM 6.3: Householder Arnoldi

1. *Select a nonzero vector* v; *Set* $z_1 = v$
2. *For* $j = 1, \ldots, m, m+1$ *Do:*
3. *Compute the Householder unit vector* w_j *such that*
4. $(w_j)_i = 0, i = 1, \ldots, j-1$ *and*
5. $(P_j z_j)_i = 0, i = j+1, \ldots, n$, *where* $P_j = I - 2w_j w_j^T$
6. $h_{j-1} = P_j z_j$
7. $v_j = P_1 P_2 \ldots P_j e_j$
8. *If* $j \leq m$ *compute* $z_{j+1} := P_j P_{j-1} \ldots P_1 A v_j$
9. *EndDo*

H.F. Walker: Implementation of the GMRES method using Householder transformation. *SIAM J. on Sci. Comput.* 9:152-163, 1988

Givens Rotations

$minimize_{y \in R^k} ||\beta \boldsymbol{e}_1 - \bar{H}_m \boldsymbol{y}^k||_2$ involves QR factorization.

Do QR factorizations of H_k by Givens Rotations.

- A 2×2 Givens rotation is a matrix of the form $G = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}$ where $c = \cos(\theta)$, $s = \sin(\theta)$ for $\theta \in [-\pi, \pi]$. The orthogonal matrix G rotates the vector $(c, -s)^T$, which makes an angle of $-\theta$ with the x-axis, through an angle θ so that it overlaps the x-axis.

$$GA \begin{bmatrix} c \\ -s \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

An $N \times N$ Givens rotation $G_j(c, s)$ replaces a 2×2 block on the diagonal of the $N \times N$ identity matrix with a 2×2 Givens rotations. $G_j(c, s)$ is with a 2×2 Givens rotations in rows and columns j and $j + 1$.

- Givens rotations can be used in reducing Hessenberg matrices to triangular form. This can be done in $O(N)$ floating-point operations.
- Let H be an $N \times M (N \geq M)$ upper Hessenberg matrix with rank M. We reduce H to triangular form by first multiplying the matrix by a Givens rotations that zeros h_{21} (values of h_{11} and subsequent columns are changed)

- Step 1: Define $G_1(c_1, s_1)$ by $c_1 = h_{11}/\sqrt{h_{11}^2 + h_{21}^2}$ and $s_1 = -h_{21}/\sqrt{h_{11}^2 + h_{21}^2}$. Replace H by $G_1 H$.
- Step 2: Define $G_2(c_2, s_2)$ by $c_2 = h_{22}/\sqrt{h_{22}^2 + h_{32}^2}$ and $s_2 = -h_{32}/\sqrt{h_{22}^2 + h_{32}^2}$. Replace H by $G_2 H$.
- …
- Step j: Define $G_j(c_j, s_j)$ by $c_j = h_{jj}/\sqrt{h_{jj}^2 + h_{j+1,j}^2}$ and $s_j = -h_{j+1,j}/\sqrt{h_{jj}^2 + h_{j+1,j}^2}$. Replace H by $G_j H$.

Setting $Q = G_N \ldots G_1$. $R = QH$ is upper triangular.

Let $\bar{H}_m = QR$ by Givens rotations matrices.

$$minimize_{y \in R^k} ||\beta \boldsymbol{e}_1 - \bar{H}_m \boldsymbol{y}^k||_2$$

$$= minimize_{y \in R^k} ||Q(\beta \boldsymbol{e}_1 - \bar{H}_m \boldsymbol{y}^k)||_2$$

$$= minimize_{y \in R^k} ||\beta Q \boldsymbol{e}_1 - R \boldsymbol{y}^k||_2$$

ALGORITHM 3.5.1. `gmres`$(x, b, A, \epsilon, kmax, \rho)$

1. $r = b - Ax$, $v_1 = r/\|r\|_2$, $\rho = \|r\|_2$, $\beta = \rho$,
   $k = 0$; $g = \rho(1, 0, \ldots, 0)^T \in R^{kmax+1}$

2. While $\rho > \epsilon\|b\|_2$ and $k < kmax$ do
   (a) $k = k + 1$
   (b) $v_{k+1} = Av_k$
   for $j = 1, \ldots k$
   i. $h_{jk} = v_{k+1}^T v_j$
   ii. $v_{k+1} = v_{k+1} - h_{jk}v_j$
   (c) $h_{k+1,k} = \|v_{k+1}\|_2$
   (d) Test for loss of orthogonality and reorthogonalize if necessary.
   (e) $v_{k+1} = v_{k+1}/\|v_{k+1}\|_2$
   (f) i. If $k > 1$ apply Q_{k-1} to the kth column of H.
   ii. $\nu = \sqrt{h_{k,k}^2 + h_{k+1,k}^2}$.
   iii. $c_k = h_{k,k}/\nu$, $s_k = -h_{k+1,k}/\nu$
   $h_{k,k} = c_k h_{k,k} - s_k h_{k+1,k}$, $h_{k+1,k} = 0$
   iv. $g = G_k(c_k, s_k)g$.
   (g) $\rho = |(g)_{k+1}|$.

3. Set $r_{i,j} = h_{i,j}$ for $1 \le i, j \le k$.
   Set $(w)_i = (g)_i$ for $1 \le i \le k$.
   Solve the upper triangular system $Ry^k = w$.

4. $x_k = x_0 + V_k y^k$.

Preconditioning

Basic idea: using GMRES on a modified system such as $M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}$.

The matrix $M^{-1}A$ need not to be formed explicitly. However, $M\boldsymbol{w} = \boldsymbol{v}$ need to be solved whenever needed.

Left preconditioning

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

Right preconditioning

$$AM^{-1}\boldsymbol{u} = \boldsymbol{b} \ \ with \ \ \boldsymbol{x} = M^{-1}\boldsymbol{u}$$

Split preconditioning: M is factored as $M = M_L M_R$

$$M_L^{-1}AM_R^{-1}\boldsymbol{u} = M_L^{-1}\boldsymbol{b} \ \ with \ \ \boldsymbol{x} = M_R^{-1}\boldsymbol{u}$$

GMRES with Left Preconditioning

ALGORITHM 9.4: GMRES with Left Preconditioning

1. *Compute* $r_0 = M^{-1}(b - Ax_0)$, $\beta = \|r_0\|_2$ *and* $v_1 = r_0/\beta$
2. *For* $j = 1, \ldots, m$ *Do:*
3. *Compute* $w := M^{-1}Av_j$
4. *For* $i = 1, \ldots, j$, *Do:*
5. $h_{i,j} := (w, v_i)$
6. $w := w - h_{i,j}v_i$
7. *EndDo*
8. *Compute* $h_{j+1,j} = \|w\|_2$ *and* $v_{j+1} = w/h_{j+1,j}$
9. *EndDo*
10. *Define* $V_m := [v_1, \ldots, v_m]$, $\bar{H}_m = \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le m}$
11. *Compute* $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$, *and* $x_m = x_0 + V_m y_m$
12. *If satisfied Stop, else set* $x_0 := x_m$ *and GoTo 1*

The Arnoldi process constructs an orthogonal basis for $\text{Span}\{\boldsymbol{r}_0, M^{-1}A\boldsymbol{r}_0, (M^{-1}A)^2\boldsymbol{r}_0, \ldots (M^{-1}A)^{k-1}\boldsymbol{r}_0\}$.

GMRES with Right Preconditioning

Right preconditioned GMRES is based on solving $AM^{-1}\boldsymbol{u} = \boldsymbol{b} \;\; with \;\; \boldsymbol{x} = M^{-1}\boldsymbol{u}.$

- The initial residual is: $\boldsymbol{b} - AM^{-1}\boldsymbol{u}_0 = \boldsymbol{b} - A\boldsymbol{x}_0.$
  - This means all subsequent vectors of the Krylov subspace can be obtained without any references to the $\boldsymbol{u}$.
- At the end of right preconditioned GMRES:

$$\boldsymbol{u}_m = \boldsymbol{u}_0 + \sum_{i=1}^{m} \boldsymbol{v}_i \eta_i \quad with \quad \boldsymbol{u}_0 = M\boldsymbol{x}_0$$

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + M^{-1} \sum_{i=1}^{m} \boldsymbol{v}_i \eta_i$$

GMRES with Right Preconditioning

ALGORITHM 9.5: GMRES with Right Preconditioning

1. *Compute* $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, *and* $v_1 = r_0/\beta$
2. *For* $j = 1, \ldots, m$ *Do:*
3. *Compute* $w := AM^{-1}v_j$
4. *For* $i = 1, \ldots, j$, *Do:*
5. $h_{i,j} := (w, v_i)$
6. $w := w - h_{i,j}v_i$
7. *EndDo*
8. *Compute* $h_{j+1,j} = \|w\|_2$ *and* $v_{j+1} = w/h_{j+1,j}$
9. *Define* $V_m := [v_1, \ldots, v_m]$, $\bar{H}_m = \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le m}$
10. *EndDo*
11. *Compute* $y_m = \text{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$, *and* $x_m = x_0 + M^{-1}V_m y_m$.
12. *If satisfied Stop, else set* $x_0 := x_m$ *and GoTo 1.*

The Arnoldi process constructs an orthogonal basis for $\text{Span}\{\boldsymbol{r}_0, AM^{-1}\boldsymbol{r}_0, (AM^{-1})^2\boldsymbol{r}_0, \ldots (AM^{-1})^{k-1}\boldsymbol{r}_0\}$.

Sadd. Iterative Methods for Sparse Linear Systems.

Split Preconditioning

- M can be a factorization of the form $M = LU$.

- Then $L^{-1}AU^{-1}\boldsymbol{u} = L^{-1}\boldsymbol{b}, \quad with \quad \boldsymbol{x} = U^{-1}\boldsymbol{u}$.
  - Need to operate on the initial residual by $L^{-1}(\boldsymbol{b} - A\boldsymbol{x_0})$
  - Need to operate on the linear combination $U^{-1}(V_m\boldsymbol{y}_m)$ in forming the approximate solution

Comparison of Left and Right Preconditioning

- Spectra of $M^{-1}A$, AM^{-1} and $L^{-1}AU^{-1}$ are identical.
- In principle, one should expect convergence to be similar.
- When M is ill-conditioned, the difference could be substantial.

Jacobi Preconditioner

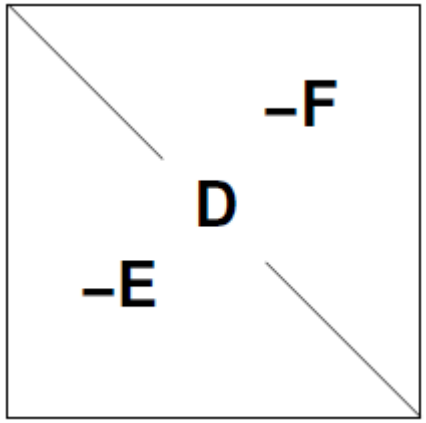
Iterative method for solving $Ax = b$ takes the form: $\boldsymbol{x}_{k+1} = M^{-1}N\boldsymbol{x}_k + M^{-1}\boldsymbol{b}$ where $M\ and\ N$ split A into $A = M - N$.

- Define $G = M^{-1}N = M^{-1}(M - A) = I - M^{-1}A$ and $\boldsymbol{f} = M^{-1}\boldsymbol{b}$.
- Iterative method is to solve $(I - G)\boldsymbol{x} = \boldsymbol{f}$, which can be written as $M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}$.

Jacobi iterative method: $\boldsymbol{x}_{k+1} = G_{JA}\boldsymbol{x}_k + \boldsymbol{f}$ where $G_{JA} = (I - D^{-1}A)$ and $\boldsymbol{f} = D^{-1}\boldsymbol{b}$

- $M = D$ for Jacobi method.

SOR/SSOR Preconditioner



- Define: $A = D - E - F$
- Gauss-Seidel: $G_{GS} = I - (D - E)^{-1}A$
- $M_{SOR} = \frac{1}{w}(D - wE)$

A symmetric SOR (SSOR) consists of:

$$(D - wE)\boldsymbol{x}_{k+\frac{1}{2}} = [wF + (1 - w)D]\boldsymbol{x}_k + w\boldsymbol{b}$$

$$(D - wF)\boldsymbol{x}_{k+1} = [wE + (1 - w)D]\boldsymbol{x}_{k+\frac{1}{2}} + w\boldsymbol{b}$$

This gives

$$\boldsymbol{x}_{k+1} = G_{SSOR}\boldsymbol{x}_k + \boldsymbol{f}$$

Where

$G_{SSOR} = (D - wF)^{-1}(wE + (1 - w)D)(D - wE)^{-1}(wF + (1 - w)D)$

- $M_{SSOR} = (D - wE)D^{-1}(D - wF)$; $M_{SGS} = (D - E)D^{-1}(D - F)$;
- Note: SSOR usually is used when A is symmetric

Take symmetric GS for example:

$M_{SGS} = (D - E)D^{-1}(D - F)$

- Define: $L = (D - E)D^{-1} = I - ED^{-1}$ and $U = D - F$.
- L is a lower triangular matrix and U is a upper triangular matrix.
- To solve $M_{SGS}\boldsymbol{w} = \boldsymbol{x}$ for $\boldsymbol{w}$, a forward solve and a backward solve are used:
  - Solve $(I - ED^{-1})\boldsymbol{z} = \boldsymbol{x}$ for $\boldsymbol{z}$
  - Solve $(D - F)\boldsymbol{w} = \boldsymbol{z}$ for $\boldsymbol{w}$

Incomplete LU(0) Factorization

Define: $NZ(X) = \{(i,j) | X_{i,j} \neq 0\}$

Incomplete LU (ILU(0)):

- $A = LU + R$ with $NZ(L) \cup NZ(U) = NZ(A)$

$$r_{ij} = 0 \quad for\ (i,j) \in NZ(A)$$

I.e. L and U have no fill-ins at the entries $a_{ij} = 0$.

for $i = 1$ **to** n
  for $k = 1$ **to** $i - 1$ and if $(i,k) \in NZ(A)$
    $a_{ik} = a_{ik}/a_{kj}$
    for j $= k + 1$ **to** n and if $(i,k) \in NZ(A)$
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$
    end;
  end;
end;

ILU(0)

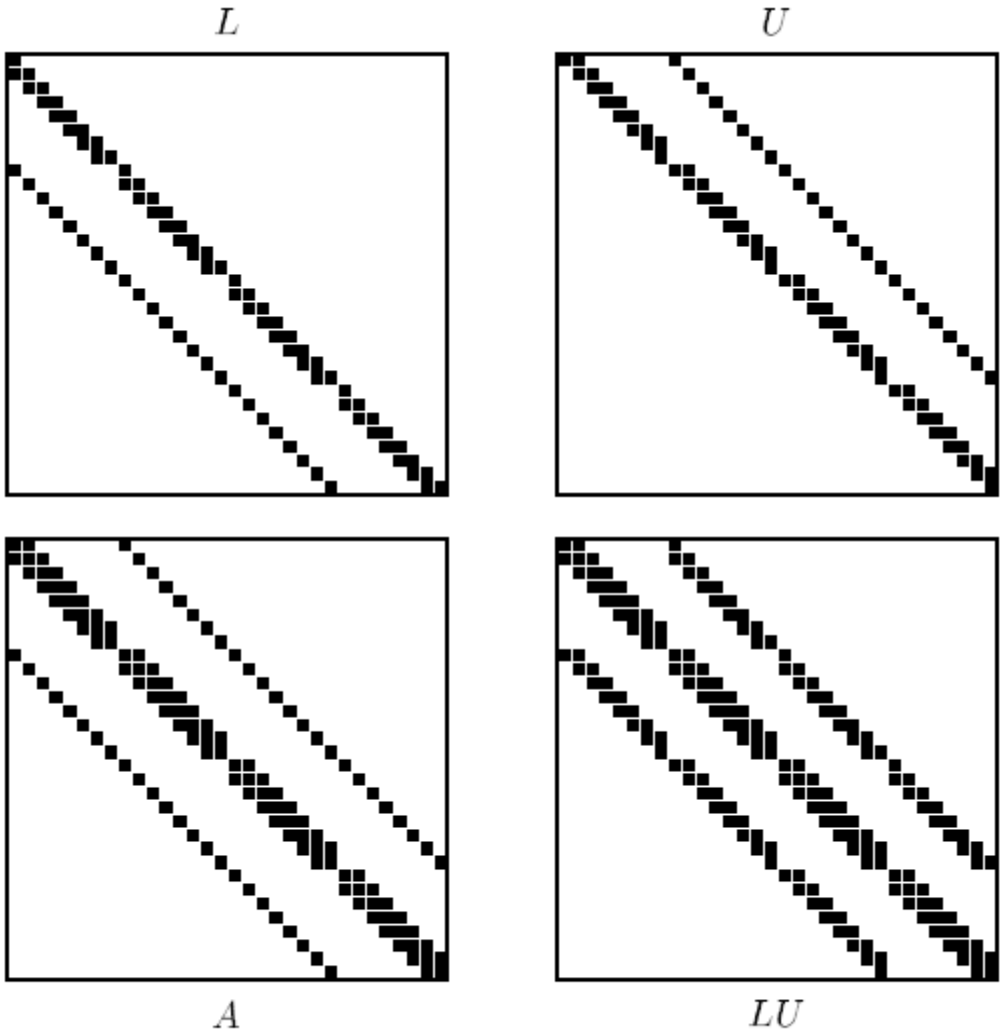
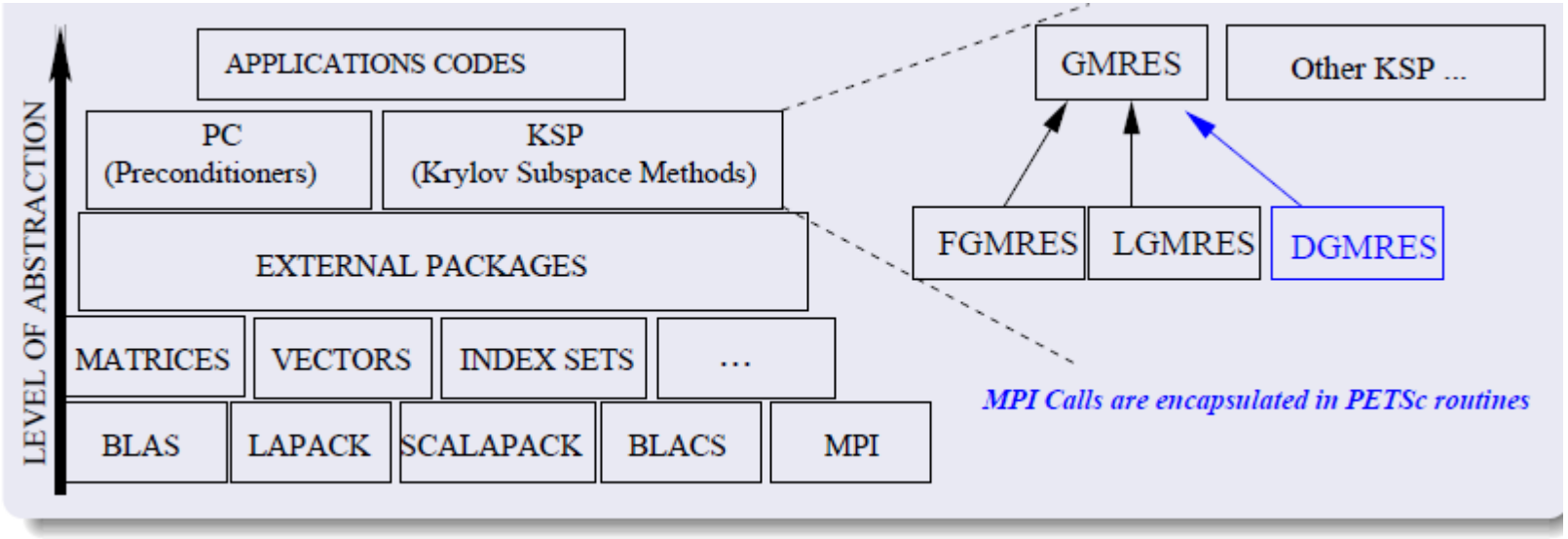


Figure 10.2 *The ILU(0) factorization for a five-point matrix.*

Sadd. Iterative Methods for Sparse Linear Systems.

Parallel GMRES

- J. Erhel. A parallel GMRES version for general sparse matrices. Electronic Transactions on Numerical Analyis. 3:160-176, 1995.
- Implementation in PETSc (Portable, Extensible Toolkit for Scientific Computation)
  - http://www.mcs.anl.gov/petsc/



Parallel Libraries

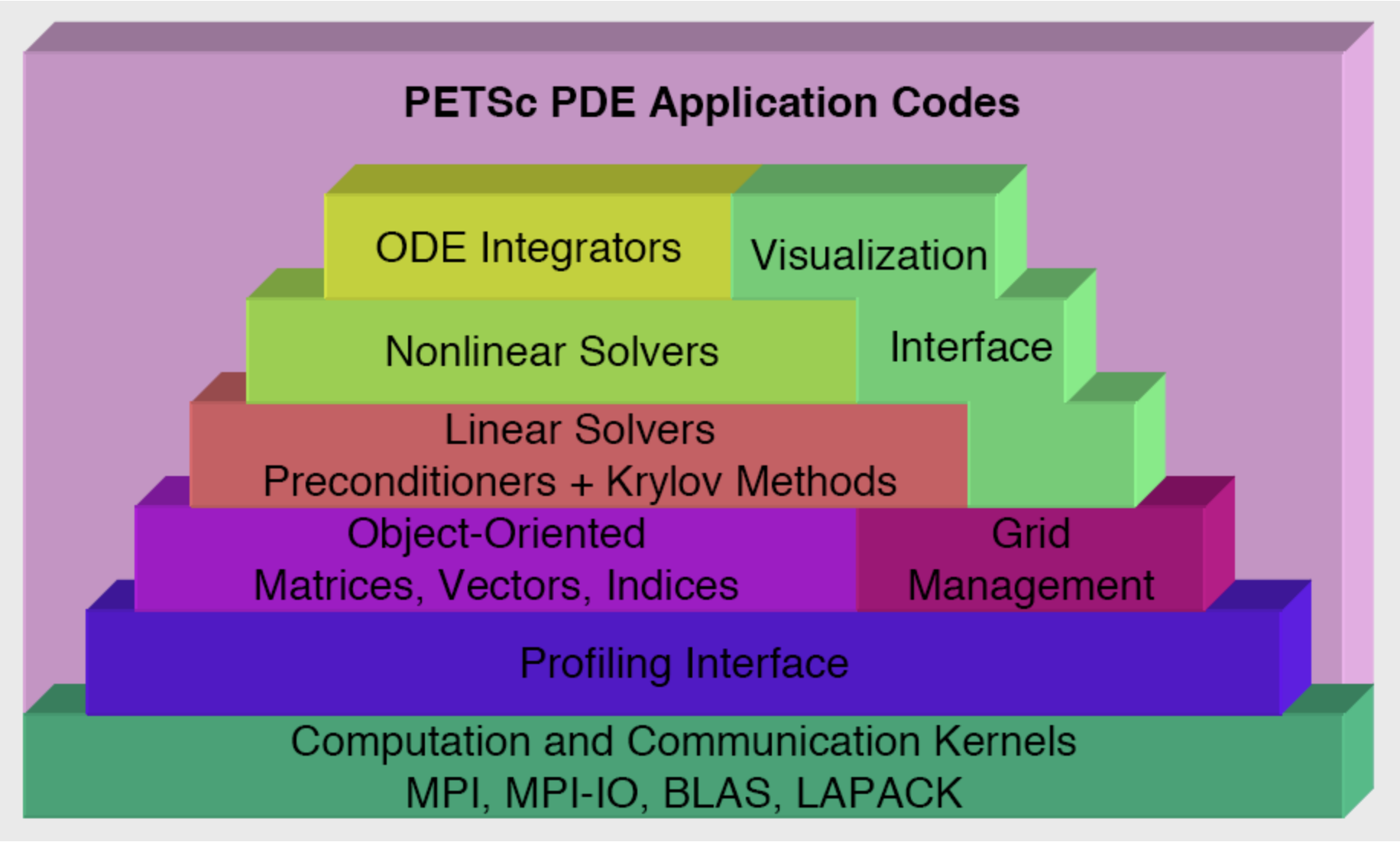
ScaLAPACK

- http://www.netlib.org/scalapack/
- Based on LAPACK (Linear Algebra PACKage) and BLAS (Basic Linear Algebra Subroutines)
- Parallelized by “divide and conquer” or block distribution
- Written in Fortran 90
- Successor of LINPACK, which was originally written for vector supercomputers in the 1970s
- Implemented on top of MPI using MIMD, SPMD, and used explicit message passing

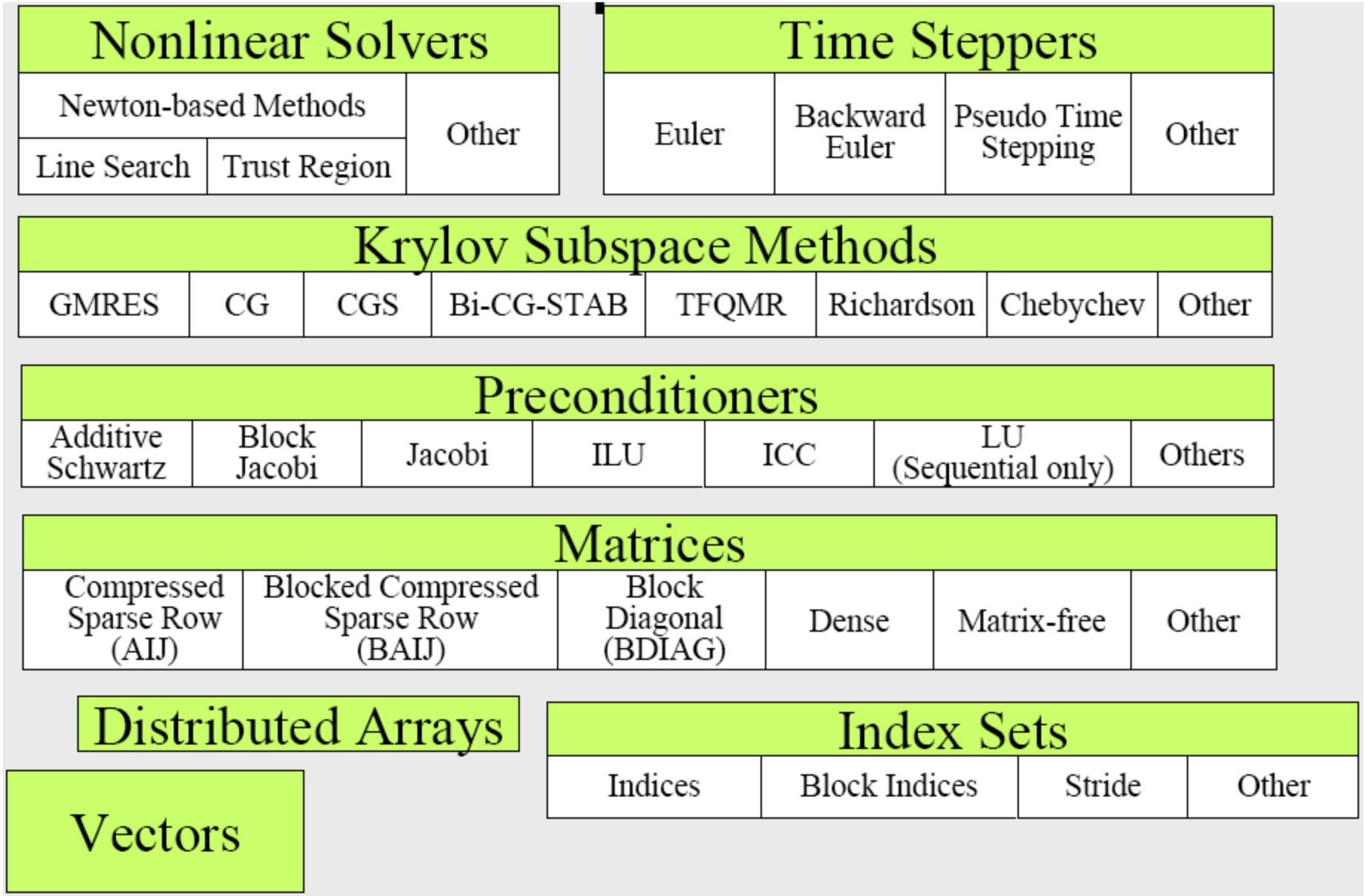
PETSc (Portable, Extensible Toolkit for Scientific Computation)

- http://www.mcs.anl.gov/petsc/
- Suite of data structures (core: distributed vectors and matrices) and routines for linea and non-linear solvers
- User (almost) never has to call MPI himself when using PETSc
- Uses two MPI communicators: PETSC_COMM_SELF for the library-internal communication and PETSC_COMM_WORLD for user processes
- Written in C, callable from Fortran
- Has been used to solve systems with over 500 millions unknowns
- Has been shown to scale up to over 6000 processors

PETSc Structure



PETSc Numerical Solvers



Parallel Random Number Generator

SPRNG (The Scalable parallel random number generators library)

- http://sprng.cs.fsu.edu/
- Random number sequence does not depend on the number of processors used, but only on the seed
  a reproducible Monte Carlo simulations in parallel
- SPRNG implements parallel-safe, high-quality random number generators
- C++/Fortran (used to be C/Fortran in previous versions)

Parallel PDE Solver

POOMA (**P**arallel **O**bject-**O**riented **M**ethods and **A**pplications)

- http://acts.nersc.gov/formertools/pooma/index.html
- Collection of templated C++ classes for writing parallel PDE solvers
- Provides high-level data types (abstractions) for fields and particles using data-parallel arrays
- Supports finite-difference simulations on structured, unstructured, and adaptive grids. Also supports particle simulations, hybrid particle-mesh simulations, and Monte Carlo
- Uses mixed message-passing/thread parallelism

Many more…

- Aztec (iterative solvers for sparse linear systems)
- SuperLU (LU decomposition)
- Umfpack (unsymmetric multifrontal LU)
- EISPACK (eigen-solvers)
- Fishpack (cyclic reduction for 2nd & 4th order FD)
- PARTI (Parallel run-time system)
- Bisect (recursive orthogonal bisection)
- ROMIO (parallel distributed file I/O)
- KINSol (solves the nonlinear algebraic systems) https://computation.llnl.gov/casc/sundials/main.html
- SciPy (Scientific Tools for Phython) http://www.scipy.org/
- …

References:

- C.T. Kelley. Iterative Methods for Linear and Nonlinear Equations.
- Yousef Sadd. Iterative methods for Sparse Linear Systems
- G. Karypis and V. Kumar. Parallel Threshold-based ILU Factorization. *Technical Report #96-061. U. of Minnesota*, Dept. of Computer Science, 1998.
- P.-O. Persson and J. Peraire. Newton-GMRES Preconditioning for Discontinuous Galerkin Discretizations of the Navier-Stokes Equations. *SIAM J. on Sci. Comput.* 30(6), 2008.