COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Lecture 7 Solving Linear Systems and Scientific Computing Libraries

Prof. Brendan Kochunas

9/25/2019

NERS 590-004

MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Outline

- Short Recap of Last Lecture
- Classical Iterative Methods
- Multigrid Methods
- Krylov Methods
- Scientific Computing Libraries

Learning Objectives

- Understand the differences of various approaches to solving linear systems
- Understand how to determine what method might be best for your application
- Become aware of popular, high quality scientific computing libraries

Basic Linear Algebra Operations

Residual and Norms of Vectors

$\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{b}$	residual
$\lVert\mathbf{r}\rVert_1 = \sum_i \lvert r_i \rvert$	1-norm
$\lVert\mathbf{r}\rVert_2 = \sqrt{\sum_i r_i^2}$	2-norm ("average error")
$\lVert\mathbf{r}\rVert_\infty = \max_i \left(\lvert r_i \rvert\right)$	∞-norm ("max local error")
$\lVert\mathbf{r}\rVert_p = \left(\sum_i \lvert r_i \rvert^p\right)^{1/p}$	*p*-norm

Inner/Dot Product (vector-vector multiply)

$$\mathbf{u}^T \cdot \mathbf{v} = \sum_i u_i v_i$$

Matrix-vector Multiply

$$\mathbf{A}\mathbf{x} = \mathbf{b} \rightarrow b_i = \sum_j a_{i,j} x_j$$

Matrix-Matrix Multiply

$$\mathbf{A}\mathbf{B} = \mathbf{C} \rightarrow c_{i,j} = \sum_k a_{i,k} b_{k,j}$$

LU and QR Decompositions

LU Decomposition

$$\mathbf{A} = \mathbf{LU}$$

Useful for solving linear systems

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{LUx} = \mathbf{b} \qquad \mathbf{L}^{-1}\mathbf{LUx} = \mathbf{L}^{-1}\mathbf{b}$$

$\mathbf{Ly} = \mathbf{b}$ Forward elimination $y_i = \dfrac{b_i - \sum_{j=1,i} \ell_{i,j} y_j}{\ell_{i,i}}$

$\mathbf{Ux} = \mathbf{y}$ Backward Substitution $x_i = \dfrac{y_i - \sum_{j=n,i,-1} u_{i,j} x_j}{y_{i,i}}$

QR Decomposition

$$\mathbf{A} = \mathbf{QR}$$

Useful for obtaining orthonormal basis

Obtaining by Gram-Schmidt

$\mathrm{proj}_{\mathbf{a}_{i-1}} \mathbf{a}_i = \dfrac{\mathbf{a}_{i-1}^T \cdot \mathbf{a}_i}{\mathbf{a}_{i-1}^T \cdot \mathbf{a}_{i-1}} \mathbf{a}_{i-1}$ project

$\mathbf{u}_i = \mathbf{a}_i - \mathrm{proj}_{\mathbf{a}_{i-1}} \mathbf{a}_i$ orthogonalize

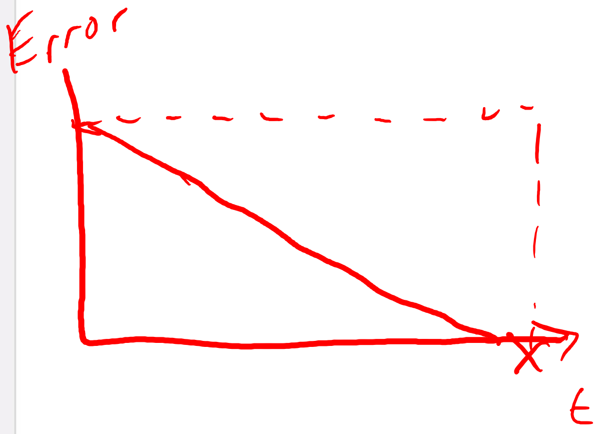
$\mathbf{q}_i = \dfrac{\mathbf{u}_i}{\left\|\mathbf{u}_i\right\|}$ normalize

$$r_{i,j} = \mathbf{q}_i^T \cdot \mathbf{a}_j$$

Solving Linear Systems

$\boxed{Ax = b}$ $\rightarrow$ MATLAB $x = b\backslash A$

"Solve"



Direct - $O(n^3)$

Iterative - $O(n^2) \times m$

Iterative Methods

- "Classical"
  - Gauss-Seidel / Jacobi
  - Successive Over-Relaxation
- Multigrid (1970's - 1980's)
- Krylov Methods
  - GMRES (1990's)
  - Lanczos & Arnoldi's (1950's)
  - Conjugate Gradient

} Symmetric / non-Symmetric

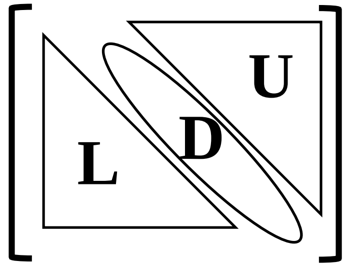
DIRECT METHODS

- Gaussian Elim.
- LU decomp.

Classical Iterative Methods

Classical Iteration Schemes

$$\mathbf{A} = (\mathbf{L} + \mathbf{D} + \mathbf{U}) =$$



Jacobi

$$x_i^{(\ell+1)} = -\frac{1}{a_{ii}}\left(\sum_{j=1}^{i-1} l_{ij} x_j^{(\ell)} + \sum_{j=i+1}^{n} u_{ij} x_j^{(\ell)}\right) x_i^{(\ell)} + b_i$$

$$\mathbf{x}^{(\ell+1)} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(\ell)} + \mathbf{D}^{-1}\mathbf{b}$$

$$\mathbf{F} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$$
$$\mathbf{c} = \mathbf{D}^{-1}\mathbf{b}$$

Gauss-Siedel

$$x_i^{(\ell+1)} = -\frac{1}{a_{ii}}\left(\sum_{j=1}^{i-1} l_{ij} x_j^{\enclose{circle}{(\ell+1)}} + \sum_{j=i+1}^{n} u_{ij} x_j^{(\ell)}\right) x_i^{(\ell)} + b_i$$

$$\mathbf{x}^{(\ell+1)} = -(\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(\ell)} + (\mathbf{D} + \mathbf{L})^{-1}\mathbf{b}$$

$$\mathbf{F} = -(\mathbf{D}^{-1} + \mathbf{L})\mathbf{U}$$
$$\mathbf{c} = (\mathbf{D}^{-1} + \mathbf{L})\mathbf{b}$$

$$\boxed{\mathbf{x}^{(\ell+1)} = \mathbf{F}\mathbf{x}^{(\ell)} + \mathbf{c}}$$

Do they converge?

- Fixed point iteration

$$\mathbf{x}^{(\ell+1)} = \mathbf{F}\mathbf{x}^{(\ell)} + \mathbf{c}$$

- Express iterate as combination of exact solution and error

$$\mathbf{x} + \boldsymbol{\varepsilon}^{(\ell+1)} = \mathbf{F}\left(\mathbf{x} + \boldsymbol{\varepsilon}^{(\ell)}\right) + \mathbf{c}$$

- If the method converges then:

$$\lim_{\ell \to \infty} \boldsymbol{\varepsilon}^{(\ell)} = 0$$

Condition for Convergence

$\varepsilon^{(\ell+1)} + \cancel{x} = \cancel{F x} + F\varepsilon^{(\ell)} + \cancel{c}$

Evolution of error?

$$\varepsilon^{(\ell+1)} = F\varepsilon^{(\ell)}$$

$$\varepsilon^{(1)} = F\varepsilon^{(0)}$$

$$\underline{\varepsilon^{(2)}} = F\varepsilon^{(1)} = \underline{F^{\ell}\varepsilon^{(0)}}$$

$$\boxed{\lim_{\ell\to\infty} \varepsilon^{(\ell)} = 0}$$

Recall Eigen decomposition $A = Q\Lambda Q^{-1}$

$$\underline{\varepsilon^{(\ell)} = Q\Lambda^{\ell}Q^{-1}\varepsilon^{(0)}}$$

$QQ^{-1} = I$ $\quad A^2 = (Q\Lambda \cancel{Q^{-1})(Q}\Lambda Q^{-1}) = Q\Lambda^2 Q^{-1}$

$|\lambda_1| > |\lambda_2| > \dots$

More about the spectral radius

$|\lambda_1| = |\lambda_{max}| =$ spectral radius $\rho(F)$

- The spectral radius determines the rate of convergence (for fixed point iteration schemes)

$\rho(F) < 1$ $\quad \|Ax\| \le \|A\| \, \|x\|$ $\quad \|\varepsilon^{(\ell+1)}\| \le \|F\| \|\varepsilon^{(\ell)}\|$

$$\frac{\|\varepsilon^{(\ell+1)}\|}{\|\varepsilon^{(\ell)}\|} \le \|F\| \qquad \frac{\|\varepsilon^{(\ell+1)}\|}{\|\varepsilon^{(\ell)}\|} \approx \rho(F) = \lim_{\ell \to \infty} \frac{\|\varepsilon^{(\ell+1)}\|}{\|\varepsilon^{(\ell)}\|}$$

$\varepsilon^{(\ell)} = r^{(\ell)} = \|Ax^{(\ell)} - b\|$ $\quad 0.99^{\ell} = 10^{-5} \rightarrow 1{,}100$ $\quad 0.3^{\ell} = 10^{-5} \rightarrow \sim 10$

Summary of Classical Iteration Schemes

- Implementations are very simple
- Error properties are very well understood
- Generally slowly converging in practical problems
- Good for simple problems

Multigrid Methods

COLLEGE OF ENGINEERING
NUCLEAR ENGINEERING & RADIOLOGICAL SCIENCES
UNIVERSITY OF MICHIGAN

Multigrid Methods

- Logical extension to classical methods that arises from error analysis.
  - Consider "shape" of error → frequency transform

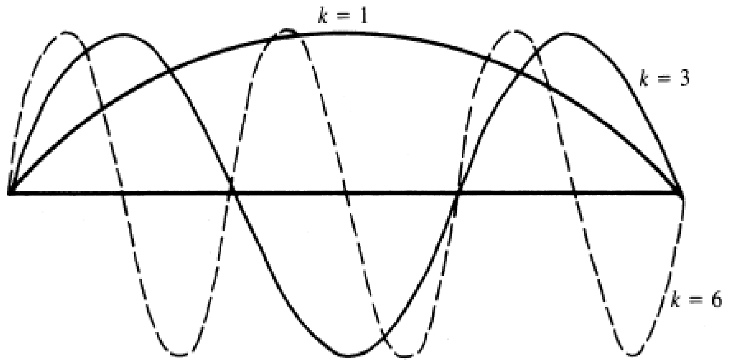


Figure 2.2: *The modes* $v_j = \sin\left(\frac{jk\pi}{n}\right)$, $0 \le j \le n$, *with wavenumbers* $k = 1, 3, 6$. *The kth mode consists of* $\frac{k}{2}$ *full sine waves on the interval.*

Low Frequency

Classical methods very good at "smoothing" high-frequency errors

Real Problem (multiple modes)

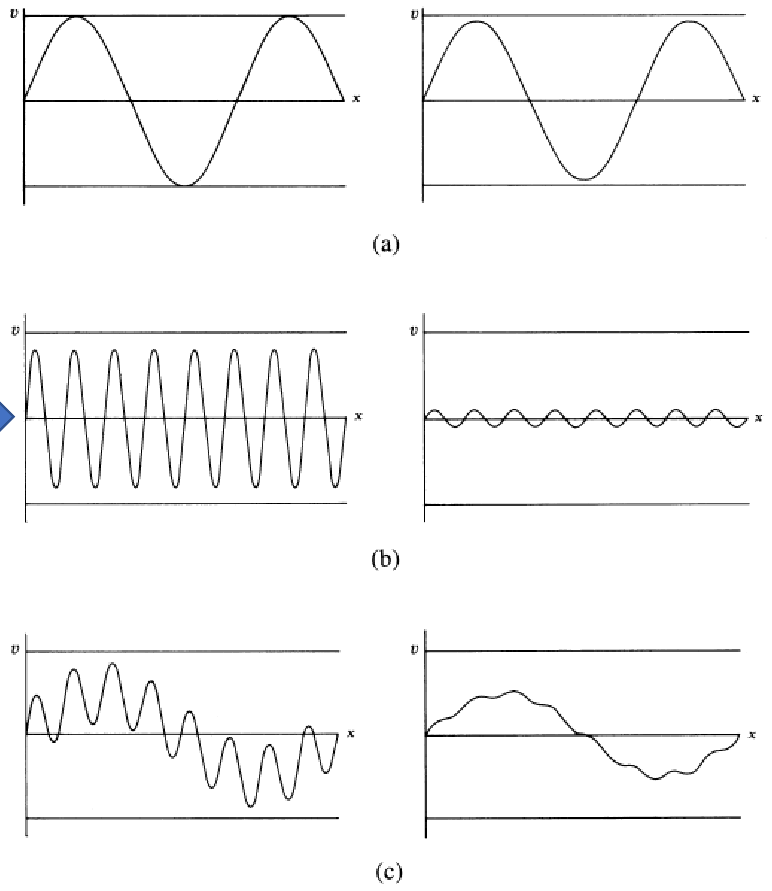
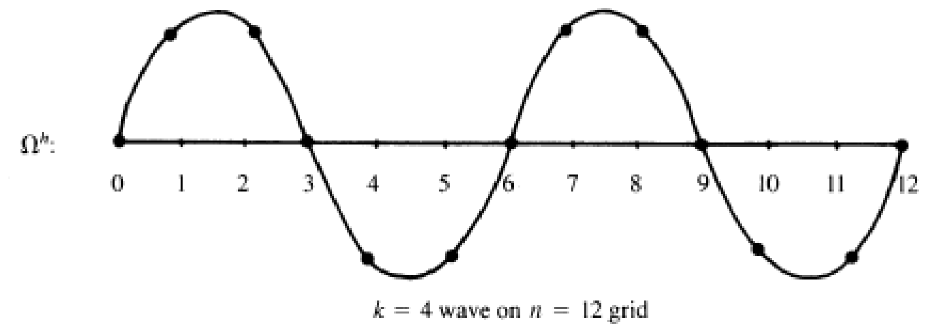


Figure 2.9: *Weighted Jacobi method with* $\omega = \frac{2}{3}$ *applied to the one-dimensional model problem with* $n = 64$ *points and with an initial guess consisting of (a)* $\mathbf{w}_3$, *(b)* $\mathbf{w}_{16}$, *and (c)* $(\mathbf{w}_2 + \mathbf{w}_{16})/2$. *The figures show the approximation after one iteration (left side) and after 10 iterations (right side).*

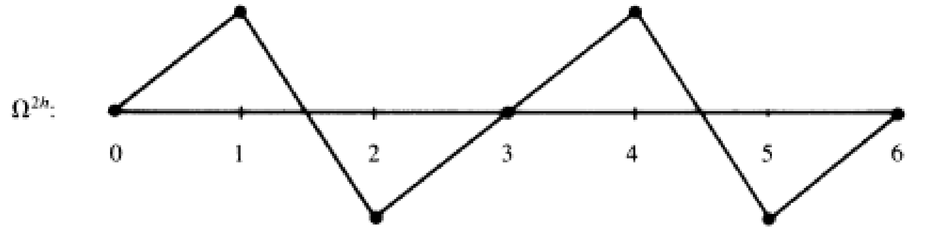
Images from: Briggs, Henson, and McCormick et al., A Multigrid Tutorial, 2nd Ed., SIAM Press (2000).

Multigrid Methods (2)

- Central idea of multigrid is to "map" errors onto coarser grids
  - A low-frequency error on a fine-grid is a high-frequency error on a coarse-grid!
- Recipe for Multigrid includes
  - How to map error from fine-grid to coarse-grid?
    - restriction operator (e.g. bi-linear average)
  - How to smooth error on each grid?
    - classical iteration scheme
  - How to correct error in fine-grid from coarse grid?
    - interpolation operator (e.g. linear interpolate)
  - How to traverse grids?



$k = 4$ wave on $n = 12$ grid



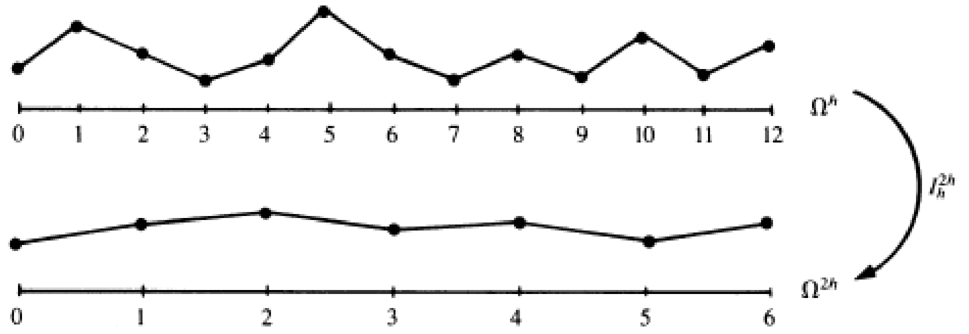
$k = 4$ wave on $n = 6$ grid

Images from: Briggs, Henson, and McCormick et al., A Multigrid Tutorial, 2nd Ed., SIAM Press (2000).

Multigrid: Restriction and Interpolation

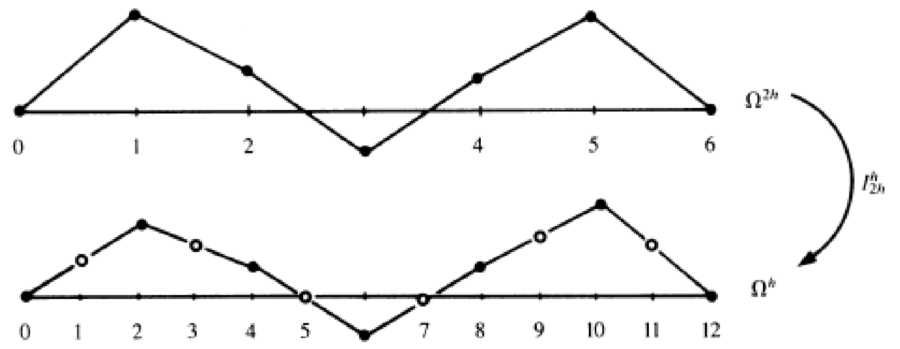
Restriction

$$I_h^{2h}\mathbf{v}^h = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & & 1 & 2 & 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{bmatrix}_h = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_{2h} = \mathbf{v}^{2h}$$



Interpolation

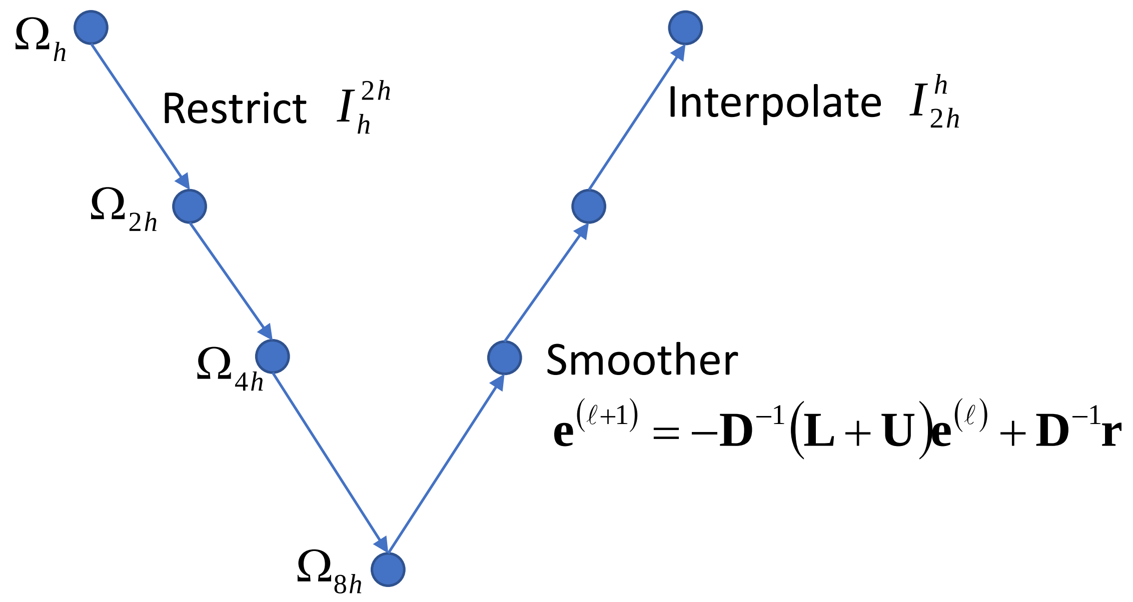
$$I_{2h}^h\mathbf{v}^{2h} = \frac{1}{2}\begin{bmatrix} 1 & & \\ 2 & & \\ 1 & 1 & \\ & 2 & \\ & 1 & 1 \\ & & 2 \\ & & 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}_{2h} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{bmatrix}_h = \mathbf{v}^h$$



Images from: Briggs, Henson, and McCormick et al., A Multigrid Tutorial, 2nd Ed., SIAM Press (2000).

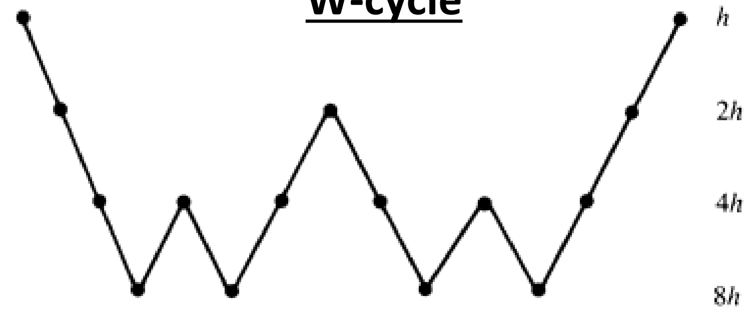
Multigrid: Traversing the Grids

V-cycle

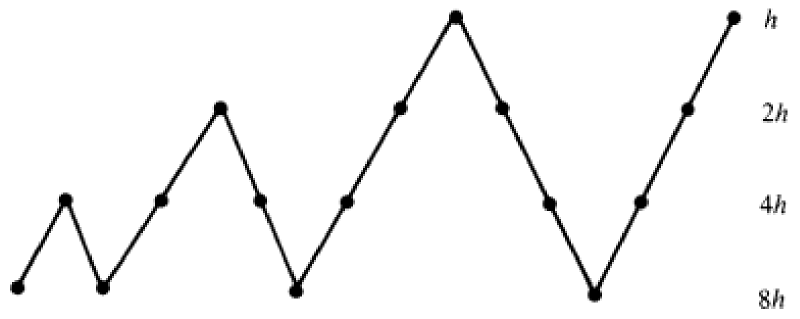


$$\mathbf{e}^{(\ell+1)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{e}^{(\ell)} + \mathbf{D}^{-1}\mathbf{r}$$

W-cycle



Full Multigrid



Images from: Briggs, Henson, and McCormick et al., A Multigrid Tutorial, 2nd Ed., SIAM Press (2000).

Summary of Multigrid

- Very good for elliptic problems
- A type of fixed point iteration
  - May be analyzed via Fourier/Von Neumann Analysis for asymptotic convergence
- Builds on traditional classical fixed point iterative techniques
  - Uses same elements and adds a few more (interpolation/prolongation)
- Lots of parameters in the iteration that can be "tuned"
- Good for structured grids and finite differenced or finite volume disc (e.g. discretized operator is a stencil)
- Can be generalized to algebraic multi-grid (AMG)

Krylov Methods

Reference: Yousef Saad, *Iterative Methods for Sparse Linear Systems*, SIAM

Krylov (subspace) Methods

- Use Krylov subspaces

$$K_p(\mathbf{A},\mathbf{b}) = \operatorname{span}\{\mathbf{b}, \mathbf{Ab}, \mathbf{A}^2\mathbf{b}, \ldots, \mathbf{A}^{\mathbf{p}}\mathbf{b}\}$$

- Krylov subspaces are easy and efficient to construct
  - Use only matrix-vector operations (this is good for large sparse systems!)

- Krylov methods also rely on projection
  - Specifically projection to the Krylov subspace

General Idea: Projection Methods

- Two types: orthogonal and oblique
- For a linear system: $\mathbf{A}\mathbf{x} = \mathbf{b}$
  - $\mathbf{A}$ exists in $\boldsymbol{R}^n$ and $\boldsymbol{K}$ and $\boldsymbol{L}$ are two subspaces within $\boldsymbol{R}^n$
  - An approximation of the solution is: $\tilde{\mathbf{x}} = \mathbf{x}_0 + \delta$
  - which has the residual: $\mathbf{r} = \mathbf{A}\tilde{\mathbf{x}} - \mathbf{b}$
- Solution by orthogonal projection means

$$\text{Find} \quad \tilde{\mathbf{x}} \in \mathbf{x}_0 + K \quad \text{such that} \quad \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}^{(\ell)} \perp L$$

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$$

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}(\mathbf{x}_0 + \delta) \perp L$$

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\delta \perp L$$

$$(\mathbf{r}_0 - \mathbf{A}\delta)^T \cdot \mathbf{r}_{new} = 0$$

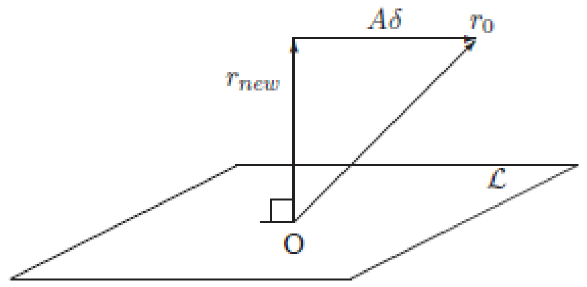
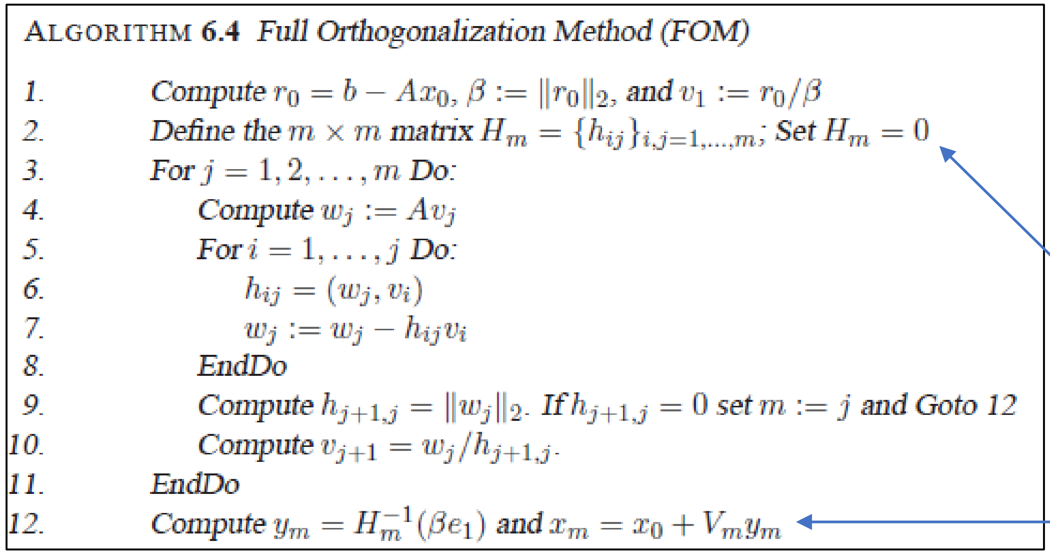


Figure 5.1: Interpretation of the orthogonality condition.

Orthogonalization Schemes: Arnoldi and GMRES

Arnoldi

- Uses Gram-Schmidt procedure to build an orthonormal basis

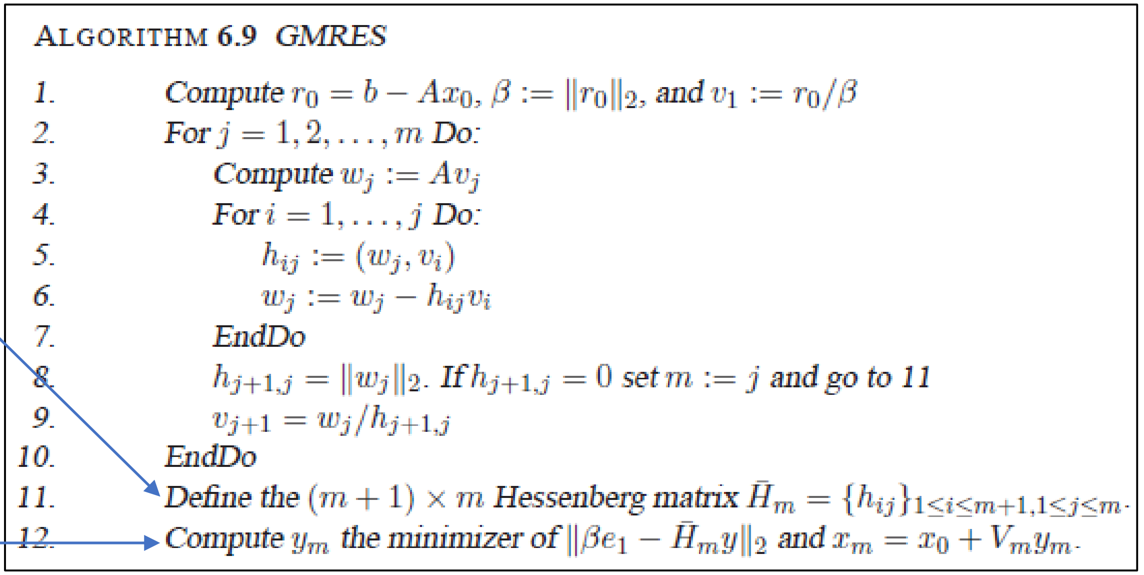


ALGORITHM 6.4 *Full Orthogonalization Method (FOM)*

```
1.  Compute $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, and $v_1 := r_0/\beta$
2.  Define the $m \times m$ matrix $H_m = \{h_{ij}\}_{i,j=1,\ldots,m}$; Set $H_m = 0$
3.  For $j = 1, 2, \ldots, m$ Do:
4.      Compute $w_j := Av_j$
5.      For $i = 1, \ldots, j$ Do:
6.          $h_{ij} = (w_j, v_i)$
7.          $w_j := w_j - h_{ij}v_i$
8.      EndDo
9.      Compute $h_{j+1,j} = \|w_j\|_2$. If $h_{j+1,j} = 0$ set $m := j$ and Goto 12
10.     Compute $v_{j+1} = w_j/h_{j+1,j}$.
11. EndDo
12. Compute $y_m = H_m^{-1}(\beta e_1)$ and $x_m = x_0 + V_m y_m$
```

GMRES

- Generalized Minimum Residual



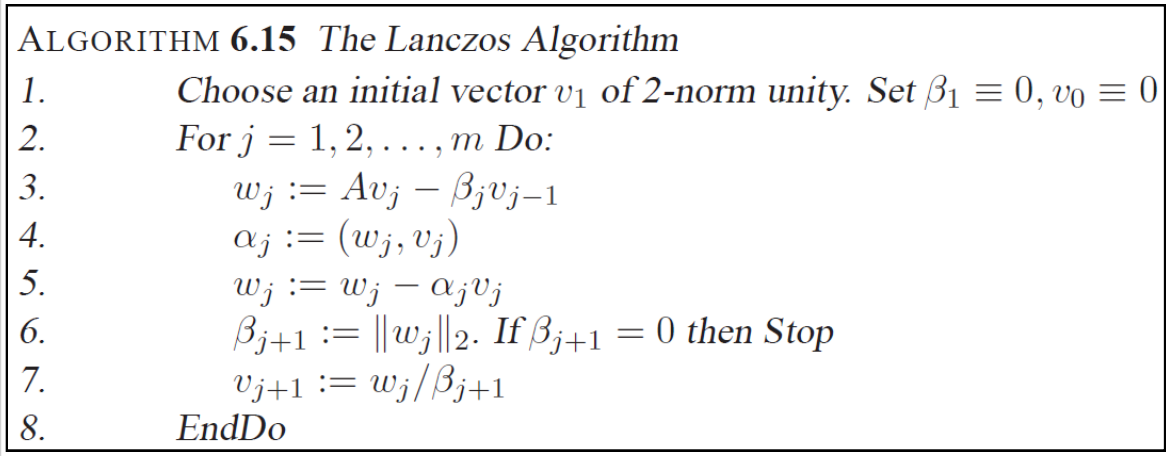
ALGORITHM 6.9 *GMRES*

```
1.  Compute $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, and $v_1 := r_0/\beta$
2.  For $j = 1, 2, \ldots, m$ Do:
3.      Compute $w_j := Av_j$
4.      For $i = 1, \ldots, j$ Do:
5.          $h_{ij} := (w_j, v_i)$
6.          $w_j := w_j - h_{ij}v_i$
7.      EndDo
8.      $h_{j+1,j} = \|w_j\|_2$. If $h_{j+1,j} = 0$ set $m := j$ and go to 11
9.      $v_{j+1} = w_j/h_{j+1,j}$
10. EndDo
11. Define the $(m+1) \times m$ Hessenberg matrix $\bar{H}_m = \{h_{ij}\}_{1 \le i \le m+1, 1 \le j \le m}$.
12. Compute $y_m$ the minimizer of $\|\beta e_1 - \bar{H}_m y\|_2$ and $x_m = x_0 + V_m y_m$.
```

Special cases: Symmetric Matrices

Lanczos algorithm for symmetric matrices

- This is a simplification of Arnoldi’s method, where the Hessenberg matrix is tridiagonal

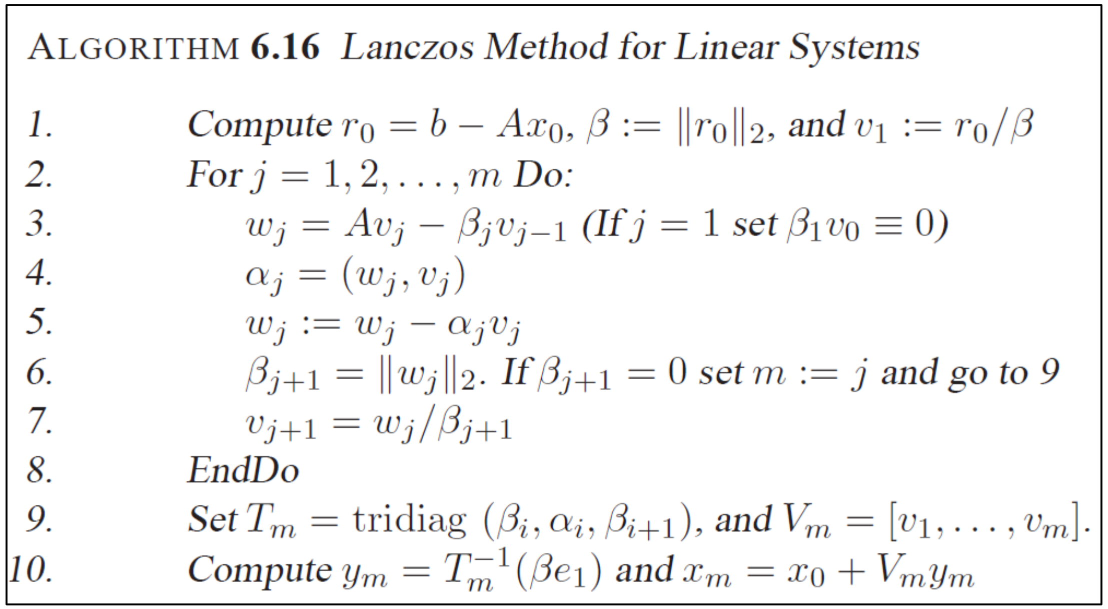


ALGORITHM **6.15** *The Lanczos Algorithm*

```
1.  Choose an initial vector $v_1$ of 2-norm unity. Set $\beta_1 \equiv 0, v_0 \equiv 0$
2.  For $j = 1, 2, \ldots, m$ Do:
3.      $w_j := Av_j - \beta_j v_{j-1}$
4.      $\alpha_j := (w_j, v_j)$
5.      $w_j := w_j - \alpha_j v_j$
6.      $\beta_{j+1} := \|w_j\|_2$. If $\beta_{j+1} = 0$ then Stop
7.      $v_{j+1} := w_j / \beta_{j+1}$
8.  EndDo
```

Conjugate Gradient (CG)

- Only for symmetric systems



ALGORITHM **6.16** *Lanczos Method for Linear Systems*

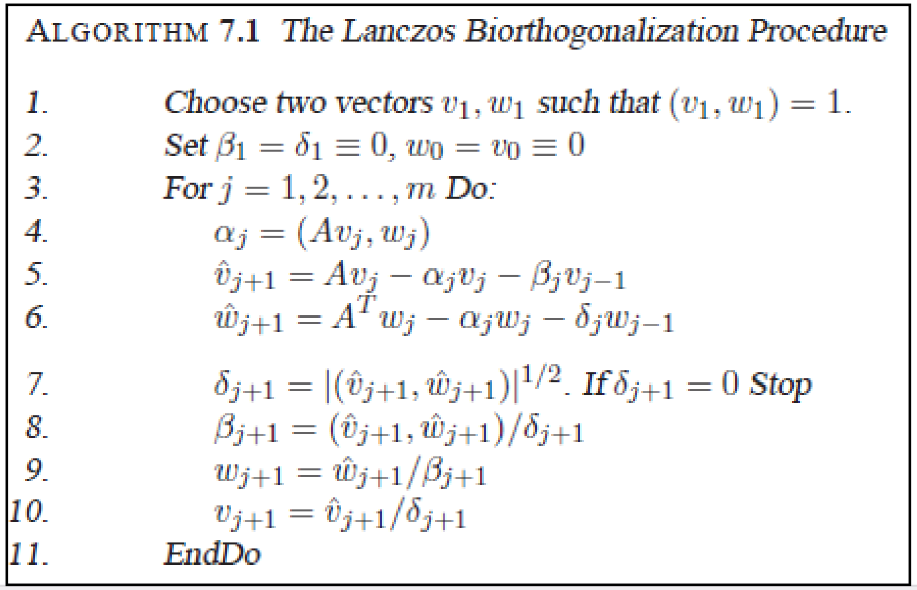
```
1.  Compute $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, and $v_1 := r_0/\beta$
2.  For $j = 1, 2, \ldots, m$ Do:
3.      $w_j = Av_j - \beta_j v_{j-1}$ (If $j = 1$ set $\beta_1 v_0 \equiv 0$)
4.      $\alpha_j = (w_j, v_j)$
5.      $w_j := w_j - \alpha_j v_j$
6.      $\beta_{j+1} = \|w_j\|_2$. If $\beta_{j+1} = 0$ set $m := j$ and go to 9
7.      $v_{j+1} = w_j / \beta_{j+1}$
8.  EndDo
9.  Set $T_m = \text{tridiag}\,(\beta_i, \alpha_i, \beta_{i+1})$, and $V_m = [v_1, \ldots, v_m]$.
10. Compute $y_m = T_m^{-1}(\beta e_1)$ and $x_m = x_0 + V_m y_m$
```

Bi-Orthognalization Schemes: Lanczos and BiCGSTAB

Lanczos (Generalize for non-symmetric)

- Build 2 subspaces

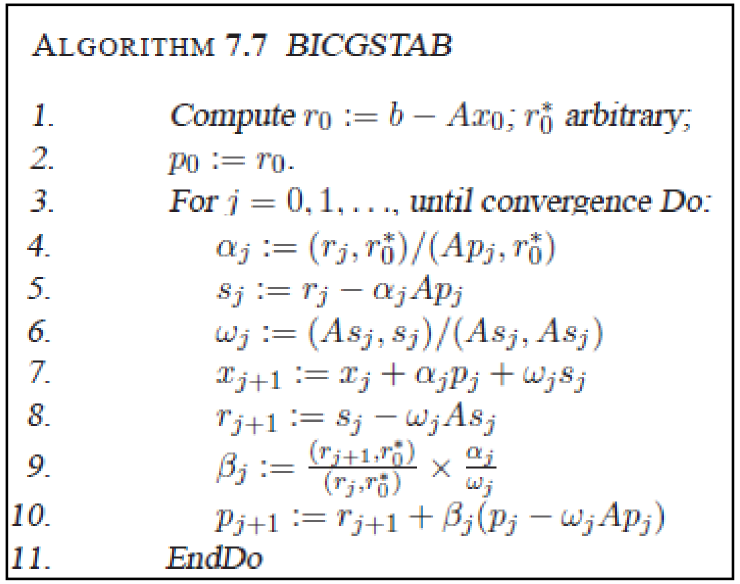
$$\mathcal{K}_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}$$
$$\mathcal{K}_m(A^T, w_1) = \text{span}\{w_1, A^Tw_1, \ldots, (A^T)^{m-1}w_1\}$$



```
ALGORITHM 7.1  The Lanczos Biorthogonalization Procedure

1.      Choose two vectors $v_1, w_1$ such that $(v_1, w_1) = 1$.
2.      Set $\beta_1 = \delta_1 \equiv 0$, $w_0 = v_0 \equiv 0$
3.      For $j = 1, 2, \ldots, m$ Do:
4.          $\alpha_j = (Av_j, w_j)$
5.          $\hat{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6.          $\hat{w}_{j+1} = A^T w_j - \alpha_j w_j - \delta_j w_{j-1}$
7.          $\delta_{j+1} = |(\hat{v}_{j+1}, \hat{w}_{j+1})|^{1/2}$. If $\delta_{j+1} = 0$ Stop
8.          $\beta_{j+1} = (\hat{v}_{j+1}, \hat{w}_{j+1})/\delta_{j+1}$
9.          $w_{j+1} = \hat{w}_{j+1}/\beta_{j+1}$
10.         $v_{j+1} = \hat{v}_{j+1}/\delta_{j+1}$
11.     EndDo
```

BiCGSTAB

- Variation of Conjugate Gradient Squared (Transpose free Bi-Conjugate Gradient)



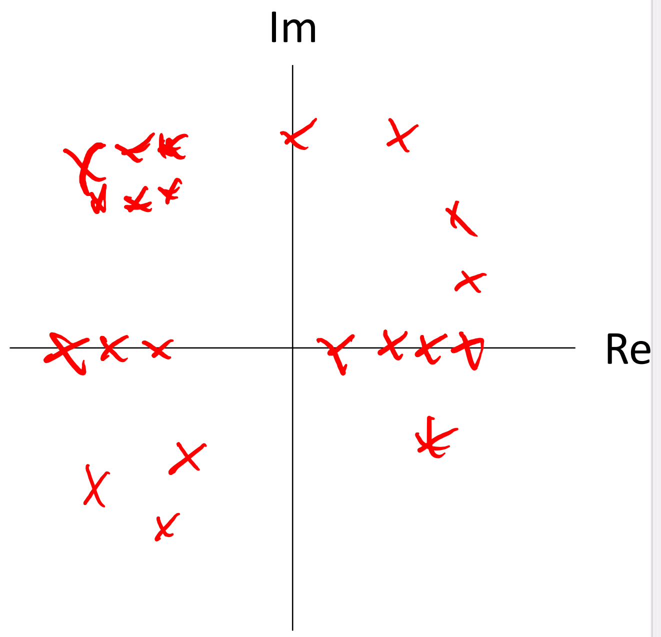
```
ALGORITHM 7.7  BICGSTAB

1.      Compute $r_0 := b - Ax_0$; $r_0^*$ arbitrary;
2.      $p_0 := r_0$.
3.      For $j = 0, 1, \ldots,$ until convergence Do:
4.          $\alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
5.          $s_j := r_j - \alpha_j Ap_j$
6.          $\omega_j := (As_j, s_j)/(As_j, As_j)$
7.          $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
8.          $r_{j+1} := s_j - \omega_j As_j$
9.          $\beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \times \frac{\alpha_j}{\omega_j}$
10.         $p_{j+1} := r_{j+1} + \beta_j (p_j - \omega_j Ap_j)$
11.     EndDo
```

Convergence of Krylov Methods

- Convergence is related to properties of the eigenspectrum of the coefficient matrix, $\mathbf{A}$.
  - But more specifically the condition number:

$$\left\|\mathbf{r}^{(\ell)}\right\| \propto \kappa(\mathbf{A})$$

- Convergence is often not monotonic
- GMRES does well if eigenvalues of $\mathbf{A}$ are clustered.
- BiCGSTAB does well if eigenvalues are spread out
  - It isolates extremal eigenvalues



More about condition numbers

- Generally it tells us how much an output value can change relate to a small change in the input
  - Bounds accuracy of approximate solution to a linear system
- Generally given by:

$$\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})}$$

- Recall that under certain conditions the SVD may be equivalent to the eigendecomposition so sometimes:

$$\kappa(\mathbf{A}) = \frac{|\lambda_{\max}(\mathbf{A})|}{|\lambda_{\min}(\mathbf{A})|}$$

- Note that this differs from the classical iterative techniques where generally convergence is given by:

$$\rho(\mathbf{A}) = \max_i |\lambda_i(\mathbf{A})| = |\lambda_{\max}(\mathbf{A})|$$

- The discretization of most PDE's give condition numbers that are ***unbounded***, and *increase as the problem size increases*.

Preconditioning

- Objective: Lower the condition number of $\mathbf{A}$ to achieve faster convergence
  - Left Preconditioner: $\mathbf{M}^{-1}\mathbf{A}\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$
  - Right Preconditioner: $\mathbf{A}\mathbf{M}^{-1}\mathbf{M}\mathbf{x} = \mathbf{b}$ solve $\mathbf{A}\mathbf{M}^{-1}\mathbf{u} = \mathbf{b}$, then $\mathbf{u} = \mathbf{M}\mathbf{x}$
- Properties of a good preconditioner
  - Large decrease in condition number for a wide range of A
  - Efficient to construct and apply
- Krylov methods generally only as good as their preconditioner.
  - This is conservation of misery.
  - Research into preconditioners still an active topic
  - Consequently, for some problems classical iteration schemes are still the best choice

Lots of Types of Preconditioners

- Classical methods; Jacobi, SOR, SSOR
- Decomposition:
  - Incomplete LU (ILU): do an LU factorization but don’t change sparsity
    - Very difficult to parallelize
  - Cholesky
- Polynomials: Chebyshev, Least-Squares
- Multigrid
- “Physics-based”

Summary of Krylov Methods

Orthogonalization

- Usually requires more storage
  - e.g. Hessenberg matrix and GMRES vectors
  - Restarted forms attempt to mitigate this
- Guaranteed to converge
  - May require full orthogonalization

Bi-Orthogonalization

- Avoid storage by doing twice the work (bi-orthogonalization)
  - Uses 3-term recurrence for orthogonalizing
- Not guaranteed to converge, but it usually does

Probably need a preconditioner

Scientific Computing Libraries

BLAS, LAPACK, PETSc, Trilinos, and Others

BLAS (Basic Linear Algebra Subprograms)

- BLAS is a (Fortran) programming interface to low-level linear algebra routines.
- Examples of basic linear algebra operations:
  - Dot products, addition of vectors, scalar multiplication of vectors.
  - Matrix-vector multiplications.
  - Matrix-matrix multiplications.
- Why BLAS?
  - Handwritten simple linear algebra routines can run at widely varying speeds.
  - Loop unrolling and finding correct compiler flags is key and sometimes becomes difficult
  - when using a new compiler or computer.
  - Basic linear algebra routines form the backbone of many sophisticated solvers and BLAS package tries to provide most optimized version of basic linear algebra routines.
  - Linking to BLAS, we get code that runs much faster than hand-written version of code.

BLAS (Basic Linear Algebra Subprograms)

- BLAS-1 operations:
  - Routines which involve only vector operations: dot-products, vector norms.
  - S – Single Precision, D - Double Precision, C – Complex, Z – Double precision complex

_SWAP	x <---> y	S, D
_SCAL	x <-- alpha * x	S, D, C, Z, CS, ZD
_COPY	x <-- y	S, D, C, Z
_AXPY	y <-- alpha * x + y	S, D, C, Z
_DOT	dot <--	S, D, DS
_DOTU	dot <-- x^T*y	C, Z
_DOTC	dot <-- x^H*y	C, Z
_NRM2	nrm2 <-- || x ||_2	S, D, SC, DZ
_ASUM	nrm1 <-- || x ||_1	S, D, SC, DZ

BLAS (Basic Linear Algebra Subprograms)

- BLAS-2 operations:
  - This level contains matrix-vector operations including, among other things, a generalized matrix-vector multiplication (gemv):

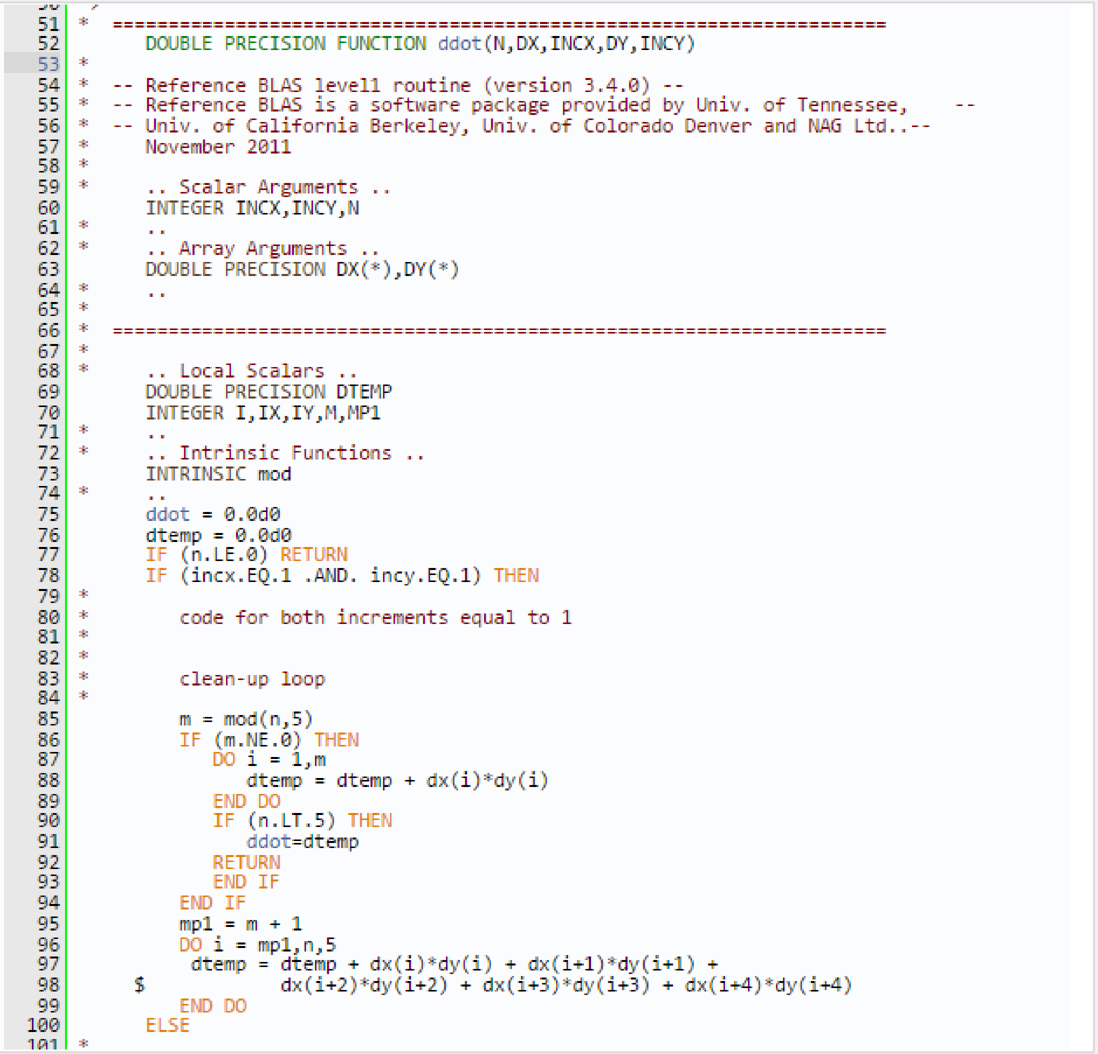
```
_GEMV      y <-- alpha*A*x + beta*y, y <-- alpha*A^T*x + beta*y  (General Real Matrix)
_GBMV      y <-- alpha*A*x + beta*y, y <-- alpha*A^T*x + beta*y  (General Banded)
_HEMV      y <-- alpha*A*x + beta*y                              (Hermitian (complex))
_HBMV      y <-- alpha*A*x + beta*y                              (Hermitian (banded))
_SYMV      y <-- alpha*A*x + beta*y                              (Symmetric)
_SBMV      y <-- alpha*A*x + beta*y                              (Symmetric banded)
_TRMV      y <-- A*x , x <-- A^T*x                               (Triangular)
_TBMV      y <-- A*x , x <-- A^T*x                               (Triangular banded)
_TRSV      y <-- inv(A)*x , x <-- inv(A^T) * x
```

BLAS (Basic Linear Algebra Subprograms)

- BLAS-3 Matrix-matrix operations:
  - This level operations are matrix-matrix multiplications.

```
_GEMM  C <-- alpha op(A) op(B) + beta C
_SYMM  C <-- alpha AB + beta C
_HEMM  C <-- alpha AB + beta C
_SYRK  C <-- alpha A A^T + beta C
_HERK  C <-- alpha A A^H + beta C
_SYRK2 C <-- alpha A B^T + alpha B A^T + beta C
_TRMM  B <-- alpha op(A) B
_TRSM  B <-- alpha op(inv(A)) B
```

BLAS-Example Code



```
*  =====================================================================
      DOUBLE PRECISION FUNCTION ddot(N,DX,INCX,DY,INCY)
*
*  -- Reference BLAS level1 routine (version 3.4.0) --
*  -- Reference BLAS is a software package provided by Univ. of Tennessee,    --
*  -- Univ. of California Berkeley, Univ. of Colorado Denver and NAG Ltd..--
*     November 2011
*
*     .. Scalar Arguments ..
      INTEGER INCX,INCY,N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION DX(*),DY(*)
*     ..
*
*  =====================================================================
*
*     .. Local Scalars ..
      DOUBLE PRECISION DTEMP
      INTEGER I,IX,IY,M,MP1
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC mod
*     ..
      ddot = 0.0d0
      dtemp = 0.0d0
      IF (n.LE.0) RETURN
      IF (incx.EQ.1 .AND. incy.EQ.1) THEN
*
*        code for both increments equal to 1
*
*
*        clean-up loop
*
         m = mod(n,5)
         IF (m.NE.0) THEN
            DO i = 1,m
               dtemp = dtemp + dx(i)*dy(i)
            END DO
            IF (n.LT.5) THEN
               ddot=dtemp
            RETURN
            END IF
         END IF
         mp1 = m + 1
         DO i = mp1,n,5
          dtemp = dtemp + dx(i)*dy(i) + dx(i+1)*dy(i+1) +
     $            dx(i+2)*dy(i+2) + dx(i+3)*dy(i+3) + dx(i+4)*dy(i+4)
         END DO
      ELSE
*
```

http://www.netlib.org/lapack/explore-html/

LAPACK

- LA Pack is a collection of Fortran functions that can help you solve Linear Algebra related problems based on BLAS routines.
- LAPACK provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.
- The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided,
- Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.
- Online Reference: http://www.netlib.org/lapack

LAPACK Routines

- Simple driver routines: Simple driver routines solve a linear algebra problem
- Eg: Finding eigenvalues of a matrix, solving a set of linear equations etc.,
- Expert driver routines: Expert driver routines do the same things as simple driver routines, but will provide more options or information to the user.
- Eg: SGESV is used to solve linear systems whereas the expert driver SGESVX not only solves the linear system but will also provide the estimate of the condition number of input matrix.
- Computational routines: Routines are mainly for internal use by LA Pack itself and called by driver routines.
- Eg: LU, QR and other factorizations or reduction of symmetric matrix to tridiagonal form.

LAPACK Naming conventions

- Lapack functions are usually named in the form XYYZZZ

X = type of problem that the routine solves
S Single precision real
D Double precision real
C Single complex
Z Double precision complex

YY = matrix types
GE General
BD Bidiagonal
HE Hermitian
HB Hermitian Band
SB Symmetric Band

ZZZ = indicate the computation performed
Eg: SV = Solve, SVX = Solve Expert

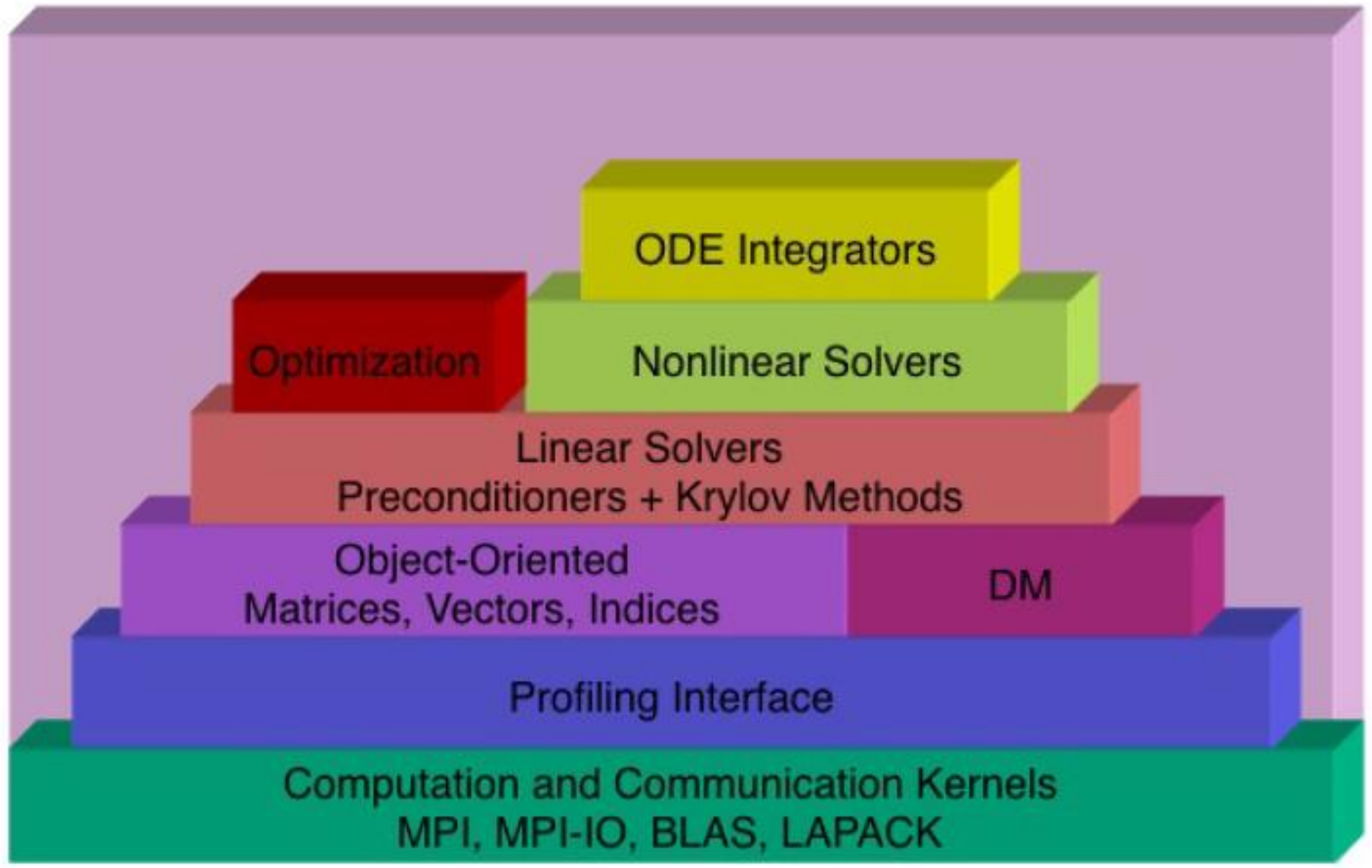
PETSc

- **P**ortable **E**xtensible **T**oolkit for **S**cientific computing:

  Developing parallel, nontrivial PDE solvers that deliver high performance is still difficult and requires months (or even years) of concentrated effort. PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box PDE solver, nor a silver bullet. — Barry Smith

- Philosophy: Everything has a plugin architecture
  - Vectors, Matrices, Partitioning algorithms
  - Preconditioners, Krylov accelerators
  - Nonlinear solvers, Time integrators
  - Spatial discretizations
  - Application user loads plugin at run, no source code in sight. PETSc tries to keep solvers independent of physics and discretization.

PETSc Structure


ODE Integrators
Optimization
Nonlinear Solvers
Linear Solvers
Preconditioners + Krylov Methods
Object-Oriented
Matrices, Vectors, Indices
DM
Profiling Interface
Computation and Communication Kernels
MPI, MPI-IO, BLAS, LAPACK

Basic PETSc object usage

Every object in PETSc supports a basic interface

Function	Operation
`Create()`	create the object
`Get/SetName()`	name the object
`Get/SetType()`	set the implementation type
`Get/SetOptionsPrefix()`	set the prefix for all options
`SetFromOptions()`	customize object from the command line
`SetUp()`	preform other initialization
`View()`	view the object
`Destroy()`	cleanup object allocation

Also, all objects support the `-help` option.

PETSc Vectors

- PETSc vectors are fundamental datatypes of PETSc which is used represent field solutions, right-hand sides etc. Each process locally owns a subvector of contiguous global data.
- Creating PETSc vectors:
  - `VecCreate(MPI_Comm, Vec *)`
  - `VecSetSizes(Vec, int n, int N)`
  - `VecSetTypes(Vec, VecType typename)`
  - `VecSetFromOptions(Vec)`
- Supports all vector space operations `VecDot()`, `VecNorm()`, `VecScale()`

PETSc Vectors

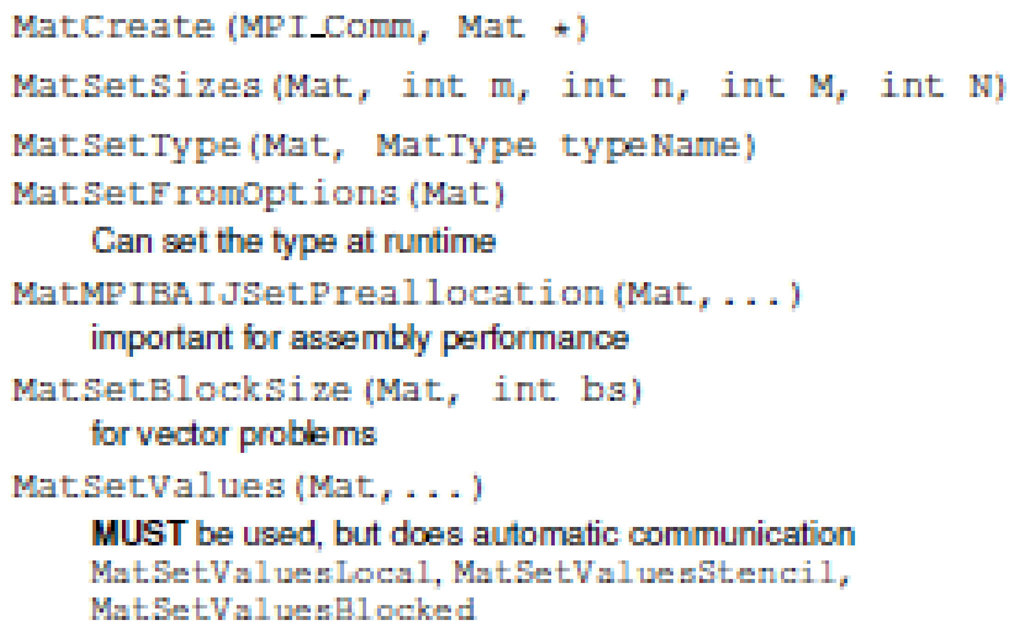
- Inserting entries into PETSc vectors:
  - Each process sets or add values and begins communications to send values to correct process and complete the communication.

  - `VecSetValues(Vec v, int n, int rows[], PetscScalar values[], mode)`
  - `VecAssemblyBegin(Vec v)`
  - `VecAssemblyEnd(Vec v)`

- PETSc allows you to access the local storage with `VecGetArray()` functions.

PETSc Matrices

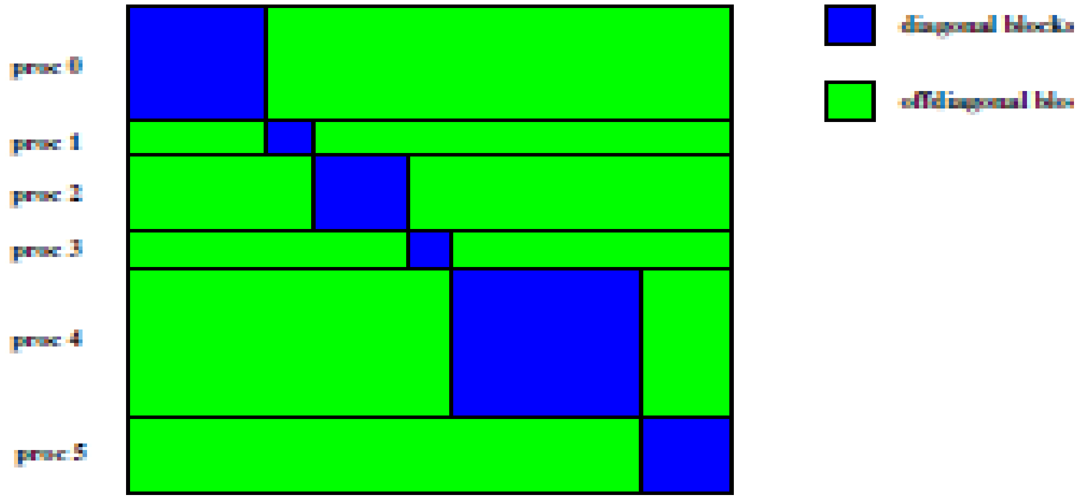
- PETSc handles both sparse and dense matrix formats in parallel.



```
MatCreate(MPI_Comm, Mat *)
MatSetSizes(Mat, int m, int n, int M, int N)
MatSetType(Mat, MatType typeName)
MatSetFromOptions(Mat)
    Can set the type at runtime
MatMPIBAIJSetPreallocation(Mat,...)
    important for assembly performance
MatSetBlockSize(Mat, int bs)
    for vector problems
MatSetValues(Mat,...)
    MUST be used, but does automatic communication
    MatSetValuesLocal, MatSetValuesStencil,
    MatSetValuesBlocked
```

Parallel sparse matrix in PETSc

Each process locally owns a submatrix of contiguous global rows
Each submatrix consists of diagonal and off-diagonal parts



```
MatGetOwnershipRange(Mat A, int *start, int *end)
```
`start`: first locally owned row of global matrix
`end-1`: last locally owned row of global matrix

Other useful features of PETSc

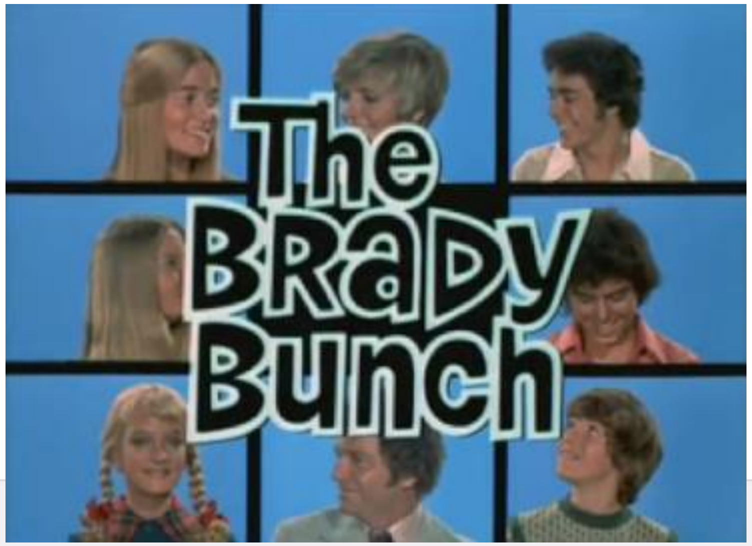
- Iterative solvers:
  - Linear solvers in PETSc KSP: Conjugate Gradient, Bi Conjugate Gradient, GMRES, etc.
  - Lots of sophisticated Preconditioners like block Jacobi, SOR, Multigrid, field-split, etc.
  - Nonlinear solvers (SNES):
    - Newton type with line search and trust-region
    - Quasi Newton methods
    - Nonlinear conjugate gradients
    - User-defined methods.
- Time Integration strategies

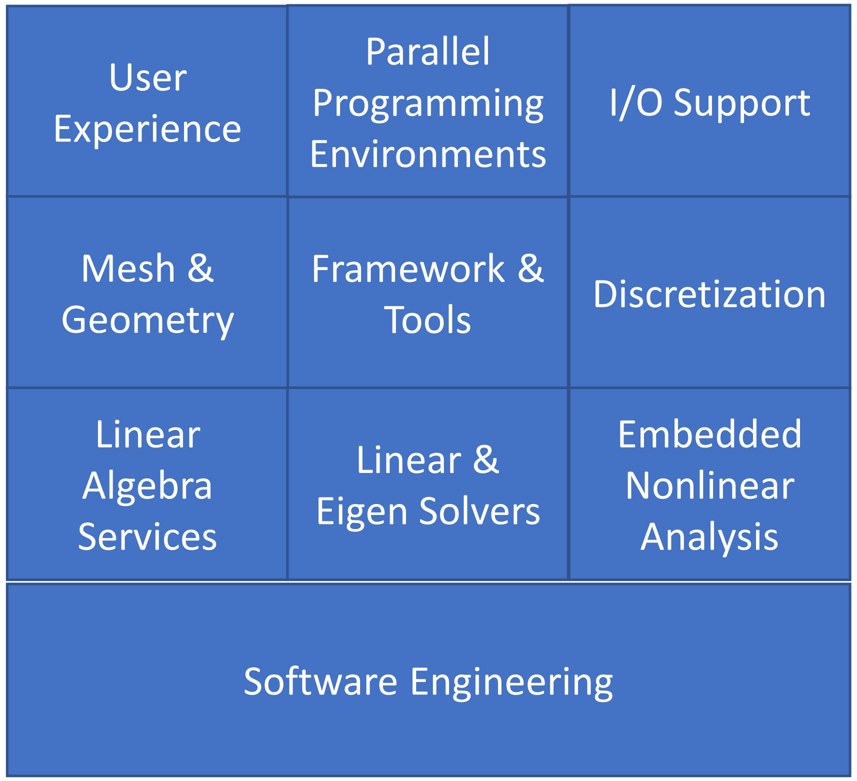
TRILINOS

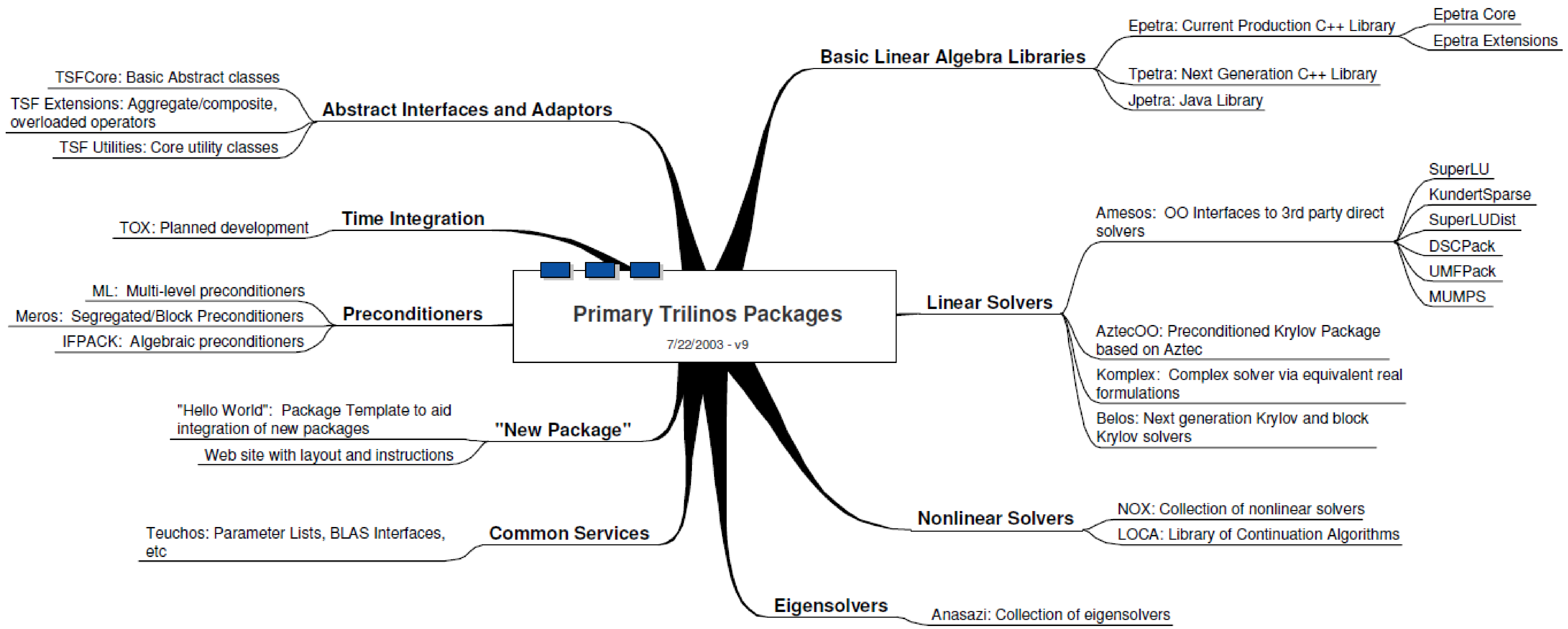
- The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages.
- More extensive than PETSc and more complex to use.
- Trilinos tries to provide an environment for solving FEM problems and PETSc provides an environment for solving sparse linear algebra problems.

Trilinos Capabilities

- Trilinos – Greek for "String of Pearls"
  - Most package names are Greek
  - Duplication of capability
  - Some are deprecated






Primary Trilinos Packages
7/22/2003 - v9
Basic Linear Algebra Libraries
Epetra: Current Production C++ Library
Epetra Core
Epetra Extensions
Tpetra: Next Generation C++ Library
Jpetra: Java Library
Abstract Interfaces and Adaptors
TSFCore: Basic Abstract classes
TSF Extensions: Aggregate/composite, overloaded operators
TSF Utilities: Core utility classes
Time Integration
TOX: Planned development
Preconditioners
ML: Multi-level preconditioners
Meros: Segregated/Block Preconditioners
IFPACK: Algebraic preconditioners
Linear Solvers
Amesos: OO Interfaces to 3rd party direct solvers
SuperLU
KundertSparse
SuperLUDist
DSCPack
UMFPack
MUMPS
AztecOO: Preconditioned Krylov Package based on Aztec
Komplex: Complex solver via equivalent real formulations
Belos: Next generation Krylov and block Krylov solvers
"New Package"
"Hello World": Package Template to aid integration of new packages
Web site with layout and instructions
Common Services
Teuchos: Parameter Lists, BLAS Interfaces, etc
Nonlinear Solvers
NOX: Collection of nonlinear solvers
LOCA: Library of Continuation Algorithms
Eigensolvers
Anasazi: Collection of eigensolvers

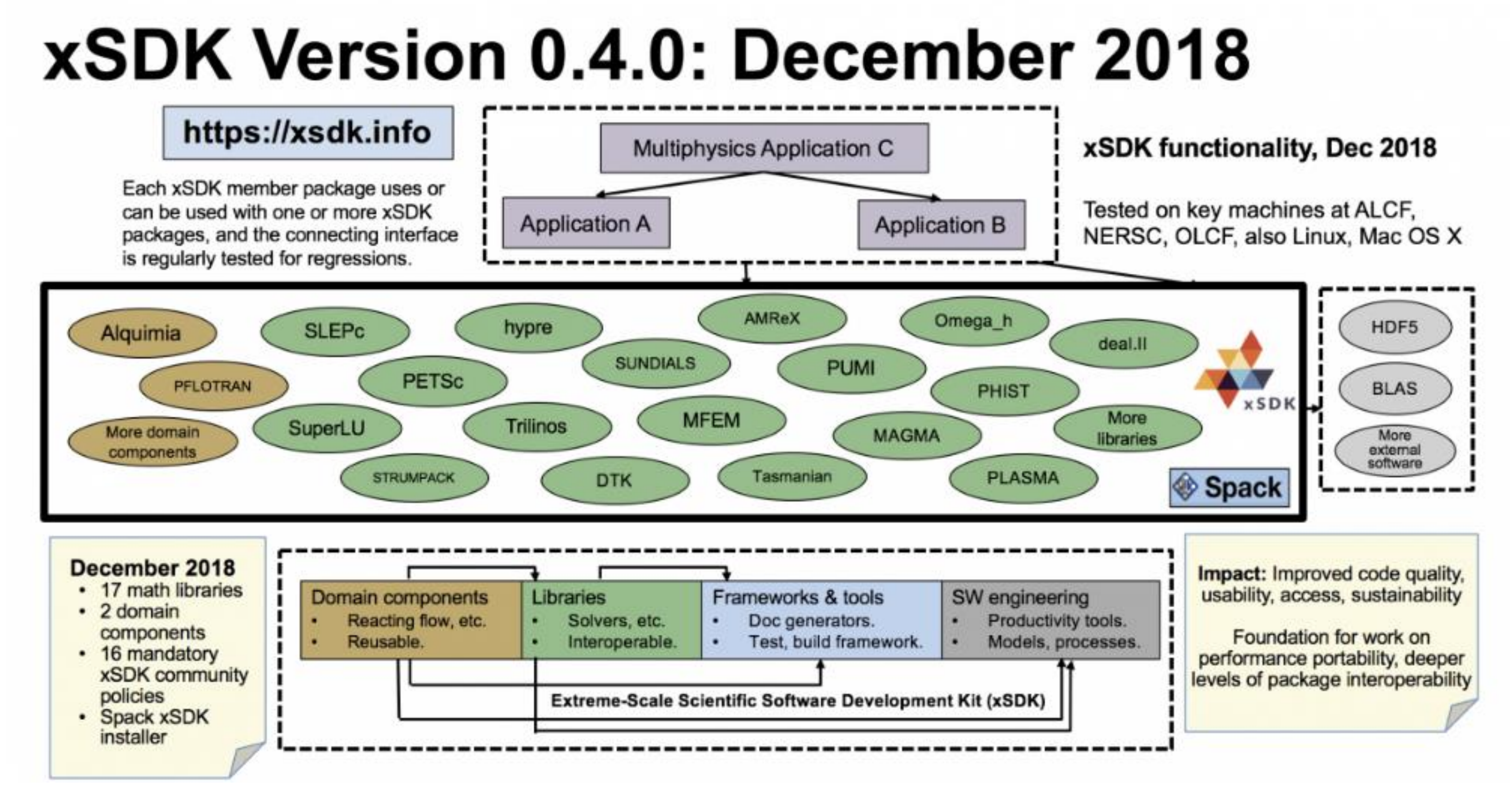
https://trilinos.org/about/capabilities/

FASTMATH

- Contains Tools for:
  - Problem Discretization
  - Solution of Algebraic Systems
  - Uncertainty Quantification
  - Numerical Optimization

https://fastmath-scidac.llnl.gov/software-catalog.html

Integrating Libraries with your Code



xSDK Version 0.4.0: December 2018

https://xsdk.info

Each xSDK member package uses or can be used with one or more xSDK packages, and the connecting interface is regularly tested for regressions.

xSDK functionality, Dec 2018

Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X

December 2018
- 17 math libraries
- 2 domain components
- 16 mandatory xSDK community policies
- Spack xSDK installer

Impact: Improved code quality, usability, access, sustainability

Foundation for work on performance portability, deeper levels of package interoperability