Projection-based iterative methods

Shortly on projectors

Projectors and properties

Definitions:
Consider C^n and a mapping $P : C^n \to C^n$.
P is called a *projector* if $P^2 = P$
If P is a projector, then $I - P$ is also such:

$$(I - P)^2 = I - 2P + P^2 = I - P.$$

$\mathcal{N}(P) = \{\mathbf{x} \in C^n : P\mathbf{x} = 0\}$ (null space (kernel) of P)
$\mathcal{R}(P) = \{A\mathbf{x} : \mathbf{x} \in C^n\}$ (range of P).

A subspace S is called *invariant under a square matrix* A whenever $AS \in S$.

Properties:

$\boxed{\textit{P1:}}\, \mathcal{N}(P) \bigcap \mathcal{R}(P) = \{0\}$ Indeed,

if $\mathbf{x} \in \mathcal{R}(P) \Rightarrow \exists \mathbf{y} : \mathbf{y} = P\mathbf{x} \Rightarrow P\mathbf{y} = P^2\mathbf{x} = P\mathbf{x} \Rightarrow \mathbf{y} = \mathbf{x} \Rightarrow \mathbf{x} = P\mathbf{x}$

If $\mathbf{x} \in \mathcal{N}(P) \Rightarrow P\mathbf{x} = 0 \Rightarrow \mathbf{x} = P\mathbf{x} \Rightarrow \mathbf{x} = 0$.

$\boxed{\textit{P2:}}\, \mathcal{N}(P) = \mathcal{R}(I - P)$

$\mathbf{x} \in \mathcal{N}(P) \Rightarrow P\mathbf{x} = 0$. Then $\mathbf{x} = I\mathbf{x} - P\mathbf{x} = (I - P)\mathbf{x}$.

$\mathbf{y} \in \mathcal{R}(I - P) \Rightarrow \mathbf{y} = (I - P)\mathbf{y} \Rightarrow P\mathbf{y} = 0$.

$\boxed{\textit{P3:}}\, C^n = \mathcal{R}(P) \bigoplus \mathcal{N}(P)$.

$\boxed{\textit{P4:}}$ Given two subspaces K and L of same dimension m, the following two conditions are mathematically equivalent:

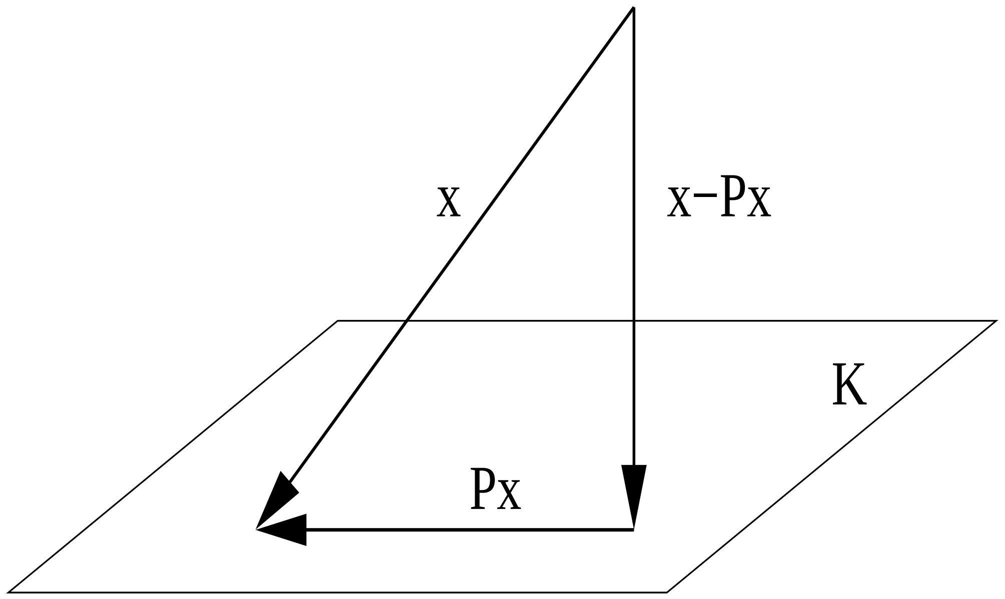
(i) No nonzero vector in K is orthogonal to L

(ii) $\forall \mathbf{x} \in C^n \exists$ unique vector $\mathbf{y} : \mathbf{y} \in K, \mathbf{x} - \mathbf{y} \in L$.

Proof (i)$\Rightarrow$(ii): $K \bigcap L^\perp = \{\emptyset\} \Rightarrow C^n = K \bigoplus L^\perp \Rightarrow \forall \mathbf{x} \in C^n : \mathbf{x} = \mathbf{y} + \mathbf{z}$, where $\mathbf{y} \in K$ and $\mathbf{z} \in L^\perp$. Thus, $\mathbf{z} = \mathbf{x} - \mathbf{y} \Rightarrow$ (ii). ■

$\boxed{P5:}$ *Orthogonal* and *oblique* projectors

P is *orthogonal* if $\mathcal{N}(P) = \mathcal{R}(P)^{\perp}$. Otherwise P is *oblique*.
Thus, if P is orthogonal onto K, then $P\mathbf{x} \in K$ and $(I - P)\mathbf{x} \perp K$. Equivalently, $((I - P)\mathbf{x}, \mathbf{y}) = 0, \forall \mathbf{y} \in K$.



Properties (cont.)

$\boxed{\textit{P6:}}$ If P is orthogonal, then $\|P\| = 1$.

$\boxed{\textit{P7:}}$ Any orthogonal projector has only two eigenvalues 0 and 1. Any vector from $\mathcal{R}(P)$ is an eigenvector to $\lambda = 1$. Any vector from $\mathcal{N}(P)$ is an eigenvector to $\lambda = 0$.

Theorem 1 *Let P be orthogonal onto K. Then for any vector $\mathbf{x} \in C^n$ there holds*

$$\min_{\mathbf{y} \in K} \|\mathbf{x} - \mathbf{y}\|_2 = \|\mathbf{x} - P\mathbf{x}\|_2. \tag{1}$$

Proof For any $\mathbf{y} \in K$, $P\mathbf{x} - \mathbf{y} \in K$, $P\mathbf{x} \in K$, $(I - P)\mathbf{x} \perp K$
$\|\mathbf{x} - \mathbf{y}\|_2^2 = \|(\mathbf{x} - P\mathbf{x}) + (P\mathbf{x} - \mathbf{y})\|_2^2 =$
$\|\mathbf{x} - P\mathbf{x}\|_2^2 + \|P\mathbf{x} - \mathbf{y}\|_2^2 + 2(\mathbf{x} - P\mathbf{x}, P\mathbf{x} - \mathbf{y}) = \|\mathbf{x} - P\mathbf{x}\|_2^2 + \|P\mathbf{x} - \mathbf{y}\|_2^2$.
Therefore, $\|\mathbf{x} - \mathbf{y}\|_2^2 \geq \|\mathbf{x} - P\mathbf{x}\|_2^2 \; \forall \mathbf{y} \in K$ and the minimum is reached for $\mathbf{y} = P\mathbf{x}$. ■

Corollary 1 *Let $K \subset C^n$ and $\mathbf{x} \in C^n$ be given. Then $\min_{\mathbf{y}} \|\mathbf{x} - \mathbf{y}\|_2 = \|\mathbf{x} - \mathbf{y}^*\|_2$ is equivalent to $\mathbf{y}^* \in K$ and $\mathbf{x} - \mathbf{y}^* \perp K$.*

Iterative solution methods

- Steepest descent
- Conjugate gradient method (CG)
- ORTHOMIN
- Generalized conjugate gradient method (GCG)
- Minimal residual method (MINRES)
- Generalized minimal residual method (GMRES)
- Lanczos method
- Arnoldi method
- Orthogonal residual method (ORTHORES)
- Full orthogonalization method (FOM)
- Incomplete orthogonalization method (IOM)

Parallel Algorithms for Scientific Computing Projectio based iterative solution methods

- SYMMLQ
- Biconjugate gradient method (BiCG)
- BiCGStab
- BiCG squared
- Conjugate gradients squared (CGS)
- Minimal residual method (MR)
- Quasiminimal residual method (QMR)
- Generalized conjugate residual method (GCR)
- . . .

Projection-based iterative methods - the basic ideas

General framework – projection methods

Want to solve $\mathbf{b} - A\mathbf{x} = \mathbf{0}, \mathbf{b}, \mathbf{x} \in R^n, A \in R^{n\times n}$

Instead, choose two subspaces $L \subset R^n$ and $K \subset R^n$ and

$$\boxed{* \text{ find } \widetilde{\mathbf{x}} \in \mathbf{x}^{(0)} + K, \text{ such that } \mathbf{b} - A\widetilde{\mathbf{x}} \perp L}$$

K - search space
L - subspace of constraints

$*$ - basic projection step

The framework is known as Petrov-Galerkin conditions.

There are two major classes of projection methods:

- orthogonal - if $K \equiv L$,
- oblique - if $K \neq L$.

Want to solve $\mathbf{b} - A\mathbf{x} = \mathbf{0}$
Given $\mathbf{x}^0$, K and L,

$$\boxed{\ast \text{ find } \widetilde{\mathbf{x}} \in \mathbf{x}^{(0)} + K, \text{ such that } \mathbf{b} - A\widetilde{\mathbf{x}} \perp L}$$

Notations:
$\widetilde{\mathbf{x}} = \mathbf{x}^0 + \delta$ (δ – correction)
$\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0$ ($\mathbf{r}^0$ – residual)

$$\boxed{\ast \text{ find } \delta \in K, \text{ such that } \mathbf{r}^0 - A\delta \perp L}$$

Matrix formulation

Choose a basis in K and L: $V = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m\}$ and $W = \{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_m\}$.
Then, $\widetilde{\mathbf{x}} = \mathbf{x}^0 + \delta = \mathbf{x}^0 + V\mathbf{y}$ for some $\mathbf{y} \in R^m$.

The orthogonality condition can be written as

$$(**) \boxed{W^T(\mathbf{r}^0 - AV\mathbf{y})}$$

which is exactly the Petrov-Galerkin condition.
From $(**)$ we get

$$\begin{aligned} W^T\mathbf{r}^0 &= W^TAV\mathbf{y} \\ \mathbf{y} &= (W^TAV)^{-1}W^T\mathbf{r}^0 \\ \widetilde{\mathbf{x}} &= \mathbf{x}^0 + V(W^TAV)^{-1}W^T\mathbf{r}^0 \end{aligned}$$

In practice, $m < n$, even $m \ll n$, for instance, $m = 1$.
The matrix W^TAV will be small and, hopefully, with a nice structure.

!!! W^TAV should be invertible.

A prototype projection-based iterative method:

```
Given   $\mathbf{x}^{(0)}$; $\mathbf{x} = \mathbf{x}^{(0)}$
Until   convergence do:
        Choose $K$ and $L$
        Choose basis $V$ in $K$ and $W$ in $L$
        Compute $\mathbf{r} = \mathbf{b} - A\mathbf{x}$
                $\mathbf{y} = (W^T AV)^{-1} W^T \mathbf{r}$
                $\mathbf{x} = \mathbf{x} + V\mathbf{y}$
```

Degrees of freedom: m, K, L, V, W.
Clearly, if $K \equiv L$, then $V = W$.

To do next:
(1) Consider two important cases: $L = K$ and $L = AK$
(2) Make a special choice of K.

Property 1:

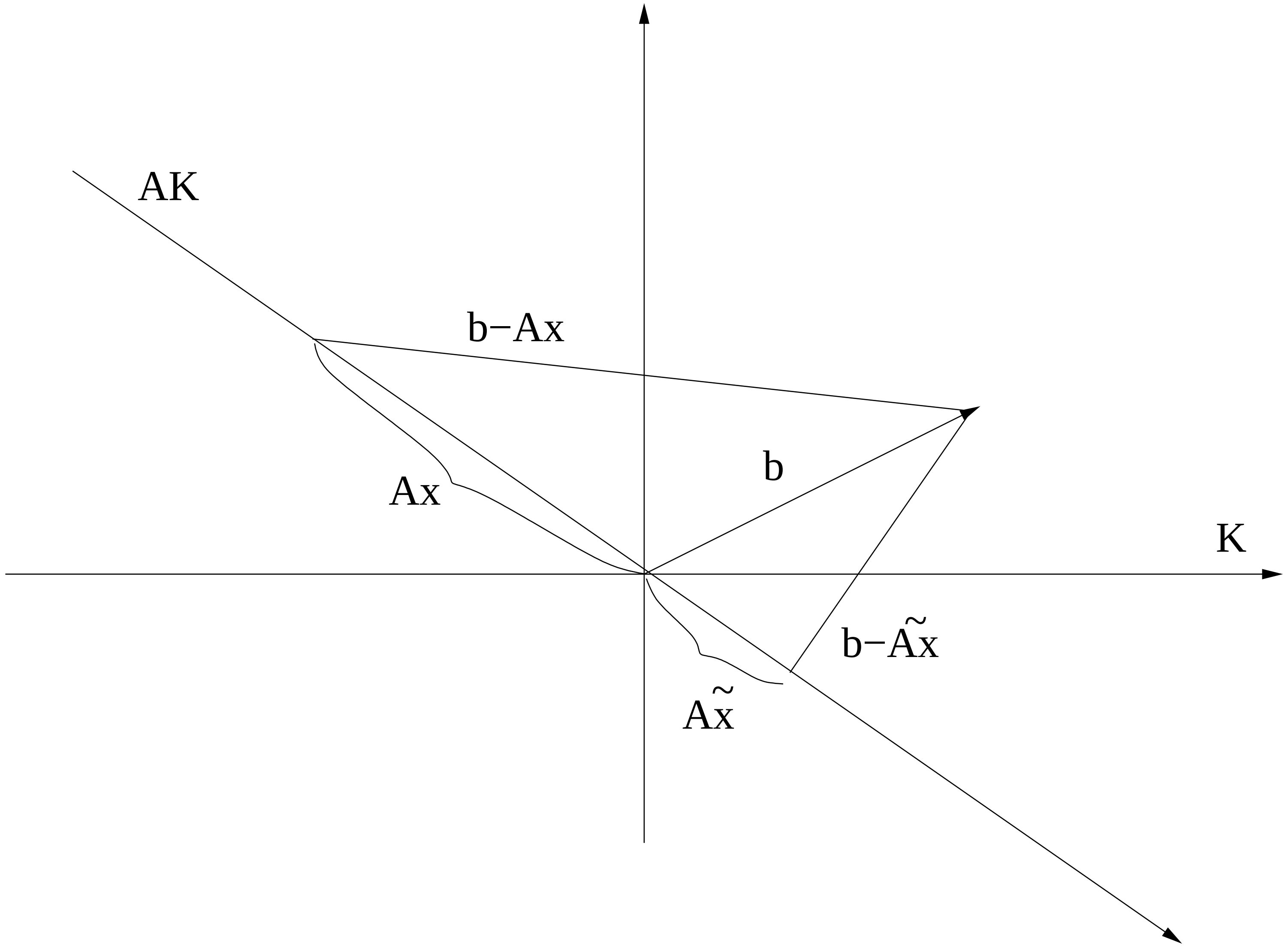
Theorem 2 *Let A be square, $L = AK$. Then a vector $\widetilde{\mathbf{x}}$ is an oblique projection on K orthogonally to AK with a starting vector $\mathbf{x}^0$ if and only if $\widetilde{\mathbf{x}}$ minimizes the 2-norm of the residual over $\mathbf{x}^0 + K$, i.e.,*

$$\|\mathbf{r} - A\widetilde{\mathbf{x}}\|_2 = \min_{\mathbf{x}\in\mathbf{x}^0+K} \|\mathbf{r} - A\mathbf{x}\|_2. \tag{2}$$

Thus, the residual decreases monotonically.

Referred to as *minimal residual methods*
CR, GCG, GMRES, ORTHOMIN

Property 1:



Property 2:

Theorem 3 *Let A be symmetric positive definite, i.e., it defines a scalar product $(A\cdot,\cdot)$ and a norm $\|\cdot\|_A$. Let $L = K$, i.e., $\mathbf{r}^0 - A\widetilde{\mathbf{x}} \perp K$. Then a vector $\widetilde{\mathbf{x}}$ is an orthogonal projection onto K with a starting vector $\mathbf{x}^0$ if and only if it minimizes the A-norm of the error $\mathbf{e} = \mathbf{x}^* - \mathbf{x}$ over $\mathbf{x}^0 + K$, i.e.,*

$$\|\mathbf{x}^* - \widetilde{\mathbf{x}}\|_A = \min_{\mathbf{x}\in\mathbf{x}^0+K} \|\mathbf{x}^* - \mathbf{x}\|_A. \tag{3}$$

The error decreases monotonically in the A-norm.
Error-projection methods

Example: $m = 1$

Consider two vectors: $\mathbf{d}$ and $\mathbf{e}$. Let $K = span\{\mathbf{d}\}$ and $L = span\{\mathbf{e}\}$.
Then $\widetilde{\mathbf{x}} = \mathbf{x}^0 + \alpha\mathbf{d}$ ($\delta = \alpha\mathbf{d}$) and the orthogonality condition reads as:

$$\mathbf{r}^0 - A\delta \perp \mathbf{e} \Rightarrow (\mathbf{r}^0 - A\delta, \mathbf{e}) = 0 \Rightarrow \alpha(A\mathbf{d}, \mathbf{e}) = (\mathbf{r}^0, \mathbf{e}) \Rightarrow \alpha = \frac{(\mathbf{r}^0, \mathbf{e})}{(A\mathbf{d}, \mathbf{e})}.$$

If $\mathbf{d} = \mathbf{e}$ - Steepest Descent method (minimization on a line.
If we minimize over a plane - ORTHOMIN.

Choice of K:

$$K = \mathcal{K}^m(A, \mathbf{v}) = \{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \cdots, A^{m-1}\mathbf{v}\}$$

Krylov subspace methods

- $L = K = \mathcal{K}^m(A, \mathbf{r}^0)$ and A spd $\Rightarrow$ CG
- $L = AK = A\mathcal{K}^m(A, \mathbf{r}^0)$ $\Rightarrow$ GMRES

Why are Krylov subspaces of interest?

Dealt with in the ’Numerical Linear Algebra’ course.

How to construct a basis for $\mathcal{K}$?

The Conjugate Gradient method

Let $L = K$ or $L = AK$: The missing part is how to construct a basis in K

Recall:
Choose a basis in K and L: $V = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m\}$ and $W = \{\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_m\}$.
Then, $\widetilde{\mathbf{x}} = \mathbf{x}^0 + \delta = \mathbf{x}^0 + V\mathbf{y}$ for some $\mathbf{y} \in R^m$.

The orthogonality condition can be written as

$$(**) \quad \boxed{W^T(\mathbf{r}^0 - AV\mathbf{y})}$$

which is exactly the Petrov-Galerkin condition.
From $(**)$ we get

$$\begin{aligned} W^T\mathbf{r}^0 &= W^TAV\mathbf{y} \\ \mathbf{y} &= (W^TAV)^{-1}W^T\mathbf{r}^0 \\ \widetilde{\mathbf{x}} &= \mathbf{x}^0 + V(W^TAV)^{-1}W^T\mathbf{r}^0 \end{aligned}$$

Arnoldi's method for general matrices

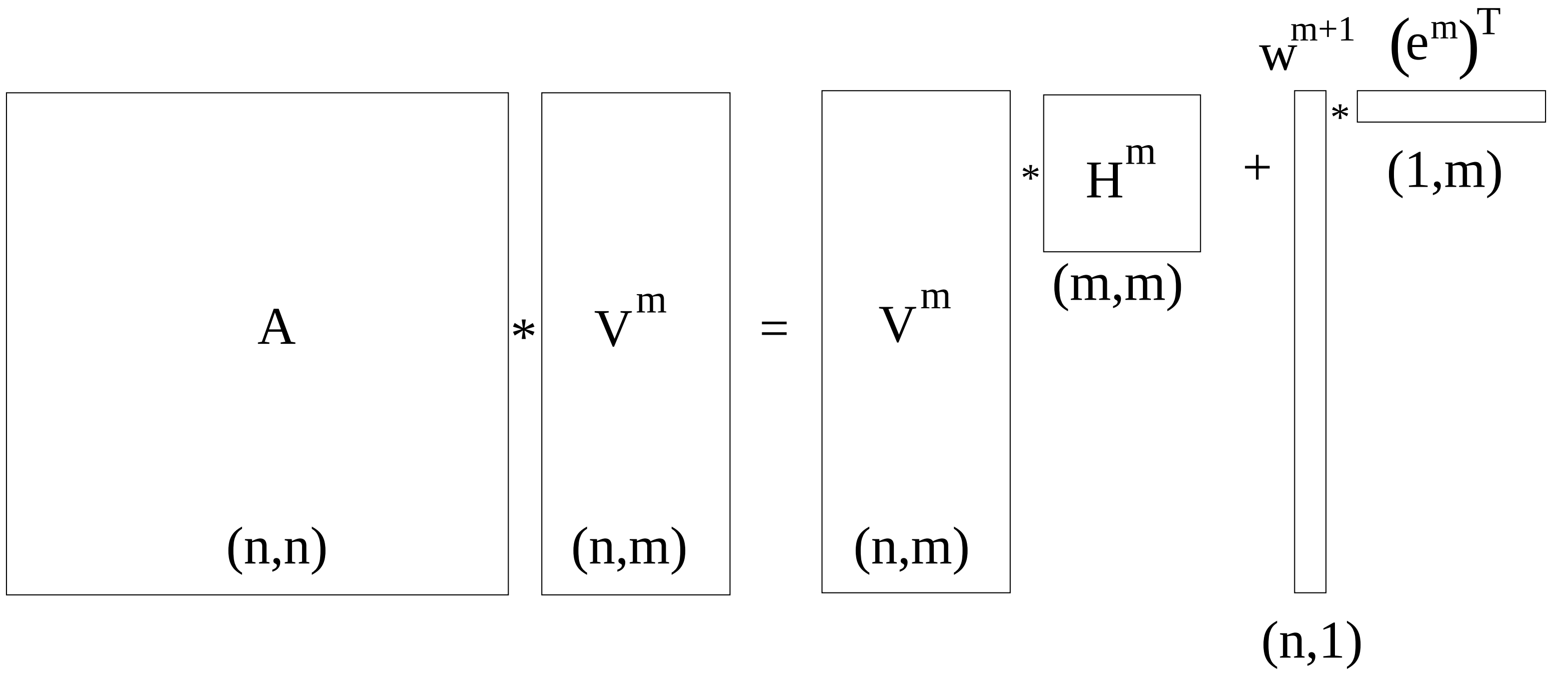
Consider $\mathcal{K}^m(A, \mathbf{v}) = \{\mathbf{v}, A\mathbf{v}, A^2\mathbf{v}, \cdots, A^{m-1}\mathbf{v}\}$, generated by some matrix A and vector $\mathbf{v}$.

```
 1.  Choose a vector $\mathbf{v}_1$ such that $\|\mathbf{v}_1\| = 1$
 2.  For $j = 1, 2, \cdots, m$
 3.      For $i = 1, 2, \cdots, j$
 4.          $h_{ij} = (A\mathbf{v}_j, \mathbf{v}_i)$
 5.      End
 6.      $\mathbf{w}_j = A\mathbf{v}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i$
 7.      $h_{j+1,j} = \|\mathbf{w}_j\|$
 8.      If $h_{j+1,j} = 0$, stop
 9.      $\mathbf{v}_{j+1} = \mathbf{w}_j / h_{j+1,j}$
10.  End
```

The algorithm breaks down in step j, i.e., $h_{j+1,j} = 0$, if and only if the minimal polynomial of A is of degree j.

The result of Arnoldi's process

- $V^m = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m\}$ is an orthonormal basis in $\mathcal{K}^m(A, \mathbf{v})$
- $AV^m = V^m H^m + \mathbf{w}_{m+1}\mathbf{e}_m^T$



Arnoldi's process - example

$$H^3 = \begin{bmatrix} (A\mathbf{v}_1, \mathbf{v}_1) & (A\mathbf{v}_2, \mathbf{v}_1) & (A\mathbf{v}_3, \mathbf{v}_1) \\ \|\mathbf{w}_1\| & (A\mathbf{v}_2, \mathbf{v}_2) & (A\mathbf{v}_3, \mathbf{v}_2) \\ 0 & \|\mathbf{w}_2\| & (A\mathbf{v}_3, \mathbf{v}_3) \end{bmatrix}$$

Since $V^{m+1} \perp \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m\}$ then it follows that $(V^m)^T A V^m = H^m$.
H^m is an upper-Hessenberg matrix.

Arnoldi's method for symmetric matrices

Let now A be real symmetric matrix. Then the Arnoldi method reduces to the Lanczos method.

Recall: $H^m = (V^m)^T A V^m$
If A is symmetric, then H^m must be symmetric too, i.e., H^m is three-diagonal

$$H^m = \begin{bmatrix} \gamma_1 & \beta_2 & & \\ \beta_2 & \gamma_2 & \beta_3 & \\ & & \ddots & \\ & & \beta_m & \gamma_m \end{bmatrix}$$

Thus, the vectors $\mathbf{v}^j$ satisfy a three-term recursion:

$$\beta_{i+1}\mathbf{v}^{i+1} = A\mathbf{v}^i - \gamma_i\mathbf{v}^i - \beta_i\mathbf{v}^{i-1}$$

The CG method:

The CG algorithm using the above relations:

```
Initialize:   $\mathbf{r}^{(0)} = A\mathbf{x}^{(0)} - \mathbf{b}$, $\mathbf{g}^{(0)} = \mathbf{r}^{(0)}$
For           $k = 0, 1, \cdots$ , until convergence
              $\tau_k = \frac{(\mathbf{r}^{(k)},\mathbf{r}^{(k)})}{(A\mathbf{g}^k,\mathbf{g}^{(k)})}$
              $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \tau_k \mathbf{g}^k$
              $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} + \tau_k A\mathbf{g}^k$
              $\beta_k = \frac{(\mathbf{r}^{(k+1)},\mathbf{r}^{(k+1)})}{(\mathbf{r}^{(k)},\mathbf{r}^{(k)})}$
              $\mathbf{g}^{k+1} = -\mathbf{r}^{(k+1)} + \beta_k \mathbf{g}^k$
end
```

$\mathbf{r}^{(k)}$ – iteratively computed residuals

$\mathbf{g}^k$ – search directions

CG: computer implementation

```
x = x0
r = A*x-b
delta0 = (r,r)
g = -r
Repeat: h   = A*g
        tau = delta0/(g,h)
        x   = x + tau*g
        r   = r + tau*h
        delta1 = (r,r)
        if delta1 <= eps, stop
        beta = delta1/delta0; delta0 = delta1
        g    = -r + beta*g
```

Computational complexity (arithmetic cost per iteration): $O(n)$
- one matrix-vector multiply
- three vector updates
- two scalar products

Properties of the CG method, convergence

- Finite termination property: there are no breakdowns of the CG algorithm.
  Reasoning: if $\mathbf{g}^j = \mathbf{0}$ then τ_k is not defined. the vectors $\mathbf{g}^j$ are computed from the formula $\mathbf{g}^k = \mathbf{r}^{(k)} + \beta_k \mathbf{g}^{k-1}$. Then
  $0 = (\mathbf{r}^{(k)}, \mathbf{g}^j) = -(\mathbf{r}^{(k)}, \mathbf{r}^{(k)}) + \beta_k \underbrace{(\mathbf{r}^{(k)}, \mathbf{g}^{k-1})}_{0}$, $\Rightarrow \ \mathbf{r}^{(k)}\mathbf{0}$, i.e., the solution is already found.
  As soon as $\mathbf{x}^{(k)} \neq \mathbf{x}_{exact}$, then $\mathbf{r}^{(k)} \neq \mathbf{0}$ and then $\mathbf{g}^{k+1}) \neq \mathbf{0}$.
  However, we can generate at most n mutually orthogonal vectors in R^n, thus, CG has a finite termination property.

Convergence of the CG method

Theorem: In exact arithmetic, CG has the property that $\mathbf{x}_{exact} = \mathbf{x}^{(m)}$ for some $m \leq n$, where n is the order of A.

Rate of convergence of the CG method

Theorem: Let A is symmetric and positive definite.
Suppose that for some set S, containing all eigenvalues of A, for some polynomial $\widetilde{P}(\lambda) \in \Pi_k^1$ and some constant M there holds $\max_{\lambda \in S} \left|\widetilde{P}(\lambda)\right| \leq M$.
Then,

$$\|\mathbf{x}_{exact} - \mathbf{x}^{(k)}\|_A \leq M \|\mathbf{x}_{exact} - \mathbf{x}^{(0)}\|_A .$$

$$\|\mathbf{e^k}\|_A \leq 2 \left[\frac{\varkappa(A)+1}{\varkappa(A)-1}\right]^k \|\mathbf{e^0}\|_A$$

Seek now the smallest k, such that

$$\|\mathbf{e}^k\|_A \leq \varepsilon \|\mathbf{e}^0\|_A$$

$$k > \frac{1}{2}\sqrt{\varkappa}\, ln\left(\frac{2}{\varepsilon}\right)$$

The GMRES method

Basic GMRES

Choose $\mathbf{v}^{(1)}$ to be the normalized residual $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$.
Any vector $\mathbf{x} \in \mathbf{x}^{(0)} + K$ is of the form $\mathbf{x} = \mathbf{x}^{(0)} + V_m\mathbf{y}$. Then

$$\begin{aligned}\mathbf{b} - A\mathbf{x} &= \mathbf{b} - A(\mathbf{x}^{(0)} + V_m\mathbf{y}) \\ &= \mathbf{r}^{(0)} - AV_m\mathbf{y} \\ &= \beta\mathbf{v}^{(1)} - V_{m+1}\overline{H}_m\mathbf{y} \\ &= V_{m+1}(\beta\mathbf{e}_1 - \overline{H}_m\mathbf{y}).\end{aligned}$$

Since the columns of V_{m+1} are orthonormal, then

$$\|\mathbf{b} - A\mathbf{x}\|_2 = \|\beta\mathbf{e}_1 - \overline{H}_m\mathbf{y}\|_2.$$

Basic GMRES

```
1.   Compute $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$, $\beta = \|\mathbf{r}^{(0)}\|_2$ and $\mathbf{v}^{(1)} = \mathbf{r}^{(0)}/\beta$
2.   For $k = 1, 2, \cdots, m$
3.       Compute $\mathbf{w}^{(k)} = A\mathbf{v}^{(k)}$
4.       For $i = 1, 2, \cdots, k$
5.           $h_{ik} = (\mathbf{w}^{(k)}, \mathbf{v}^{(i)})$
6.           $\mathbf{w}^{(k)} = \mathbf{w}^{(k)} - h_{ik}\mathbf{v}^{(i)}$
7.       End
8.       $h_{k+1,k} = \|\mathbf{w}^{(k)}\|_2$; if $h_{k+1,k} = 0$, set $m = k$, goto 11
9.       $\mathbf{v}^{(k+1)} = \mathbf{w}^{(k)}/h_{k+1,k}$
10.  End
11.  Define the $(m+1) \times m$ Hessenberg matrix $\overline{H}_m = \{h_{ik}\}, 1 \le i \le m+1, 1 \le k \le m$
12.  Compute $\mathbf{y}^{(m)}$ as the minimizer of $\|\beta\mathbf{e}_1 - \overline{H}_m\mathbf{y}\|_2$ and $\mathbf{x}^{(m)} = \mathbf{x}^{(0)} + V_m\mathbf{y}^{(m)}$
```

GMRES:

- No breakdown of GMRES
- As m increases, storage and work per iteration increase fast. Remedies:
  - Restart (keep m constant)
  - Truncate the orthogonalization process
- The norm of the residual in the GMRES method is monotonically decreasing. However, the convergence may stagnate. The rate of convergence of GMRES cannot be determined so easy as that of CG.
- The convergence history depends on the initial guess.

GMRES: convergence

Theorem: Let A be diagonalizable, $A = X^{-1}\Lambda X$ where $\Lambda = diag\{\lambda_1, \cdots, \lambda_n\}$ contains the eigenvalues of A. Define

$$\epsilon^m = \min_{p \in \Pi_m^1} \max_{i=1,\cdots n} |p(\lambda_i)|.$$

Then, the residual norm at the mth step of GMRES satisfies

$$\|\mathbf{r}^{(m)}\| \leq \kappa(X)\epsilon^m \|\mathbf{r}^{(0)}\|,$$

where $\kappa(X) = \|X\| \, \|X^{-1}\|$

Parallelize CG: scenarios

- On what hardware platform?
- Which model - shared memory, distributed memory, both?
- What can we parallelize?
- What can't we parallelize? Bottlenecks?
- Can we modify the method to avoid bottlenecks?
- What data distribution shall we utilize? How much knowledge about the problem shall we utilize in the parallel implementation?
- If we have managed to parallelise CG, did we do the job?

How to parallelize the CG method?

Recall:

```
x = x0
r = A*x-b
delta0 = (r,r)
g = -r
Repeat: h = A*g                              matrix-vector
        tau = delta0/(g,h)                   scalar product
        x = x + tau*g                        vector update
        r = r + tau*h                        vector update
        delta1 = (r,r)                       scalar product
        if delta1 <= eps, stop
        beta = delta1/delta0, delta0=delta1
        g = -r + beta*g                      vector update
```

What approaches have been taken in some of the software libraries?

’Matrix-given strategy’

PETSc – Portable, Extensible Toolkit for Scientific Computation
Last version: 3.3 (CUDA)

Slides borrowed from:
Ambra Giovannini
SuperComputing Applications and Innovation Department
`www.corsi.cineca.it/courses/scuolaEstiva2/petscModule/petsc_2011.pdf`

PETSc main features

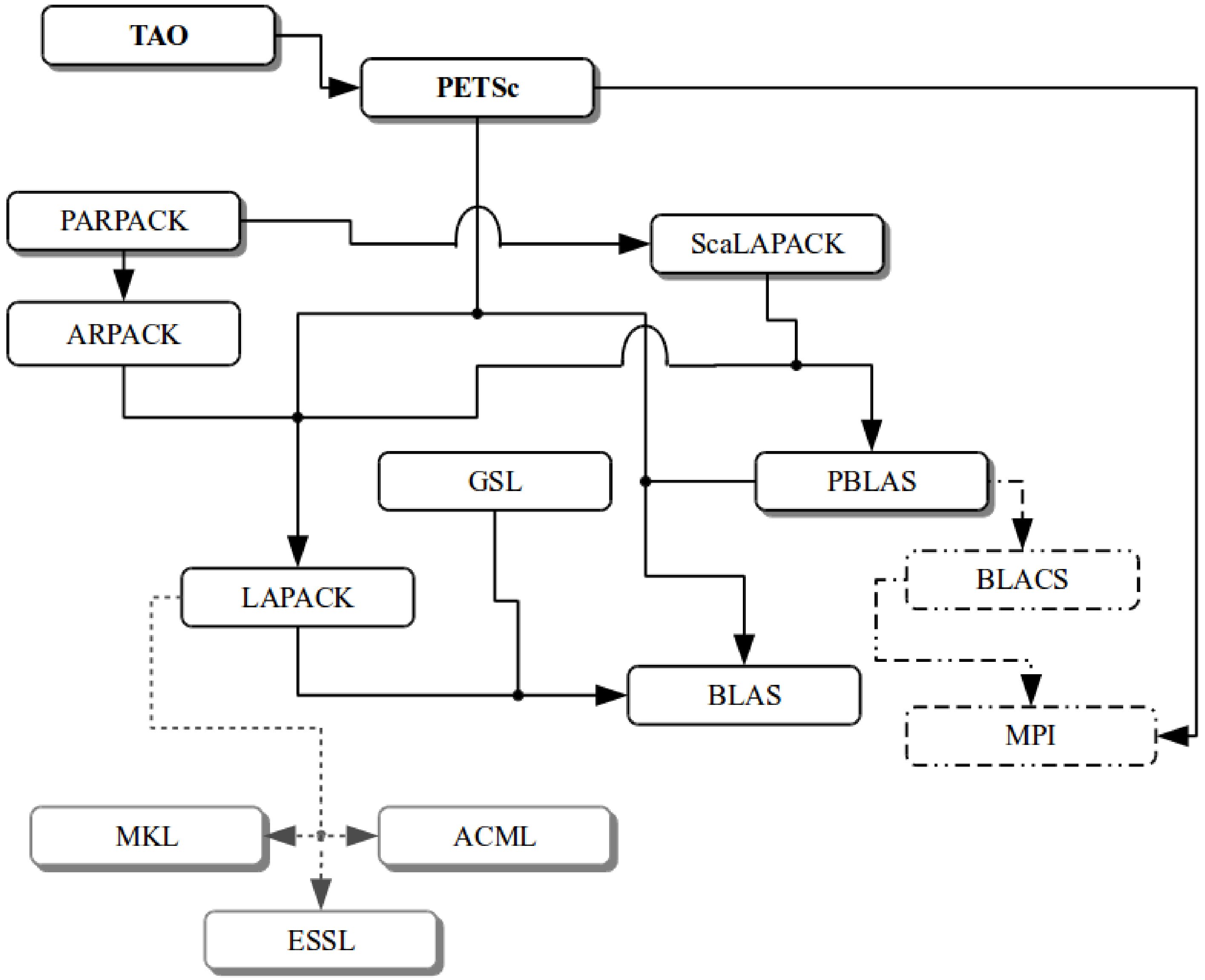
PETSc – Portable, Extensible Toolkit for Scientific Computation

Is a suite of data structures and routines for the scalable (parallel) solution of scientific applications mainly modelled by partial differential equations.

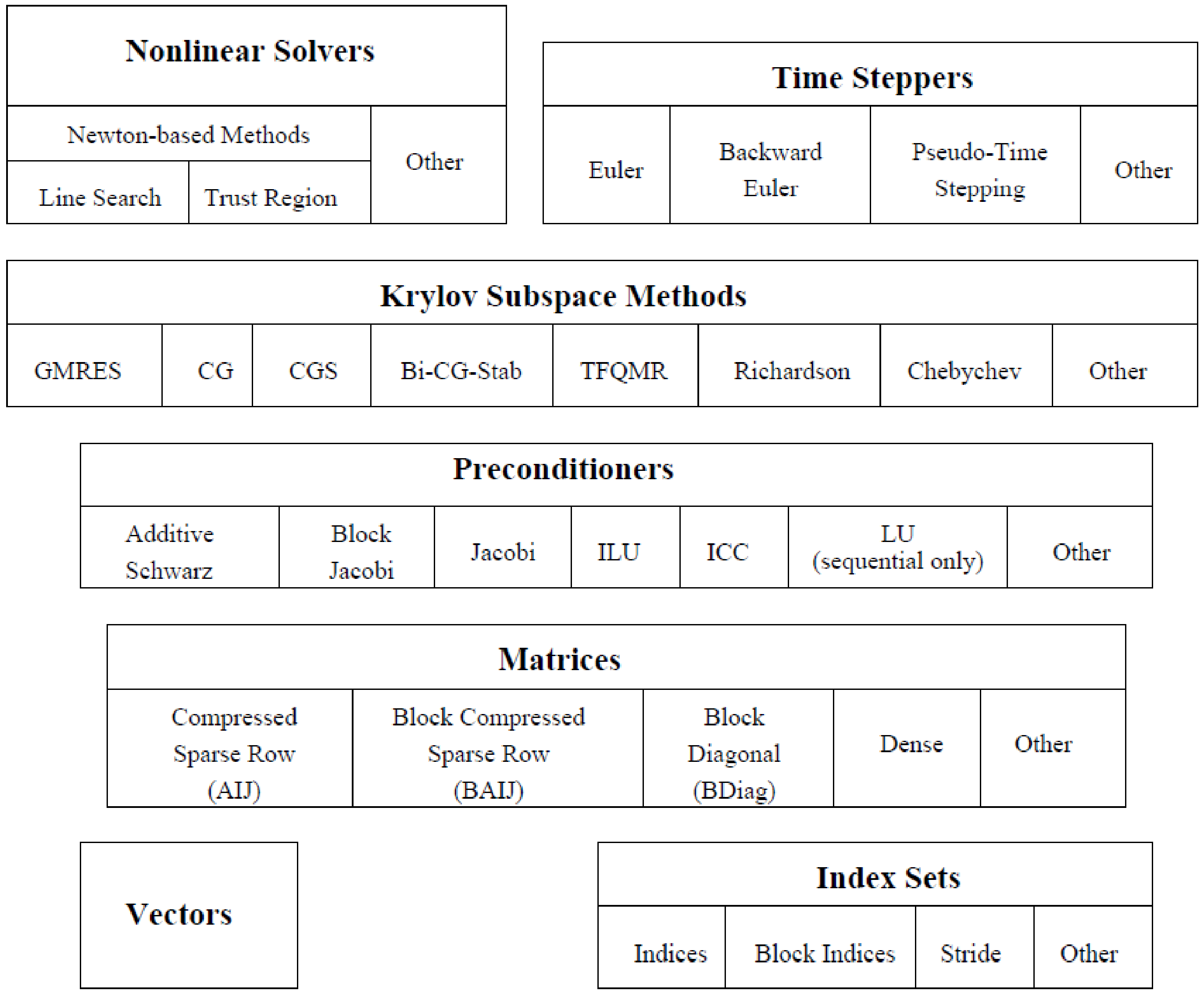
- **ANL** – Argonne National Laboratory
- Begun September **1991**
- Uses the **MPI** standard for all message-passing communication
- **C**, **Fortran**, and **C++**
- Consists of a variety of libraries; each library manipulates a particular family of **objects** and the operations one would like to perform on the objects
- PETSc has been used for modelling in all of these **areas**:

  Acoustics, Aerodynamics, Air Pollution, Arterial Flow, Brain Surgery, Cancer Surgery and Treatment, Cardiology, Combustion, Corrosion, Earth Quakes, Economics, Fission, Fusion, Magnetic Films, Material Science, Medical Imaging, Ocean Dynamics, PageRank, Polymer Injection Molding, Seismology, Semiconductors, ...

Relationship between libraries



PETSc numerical component



Vectors

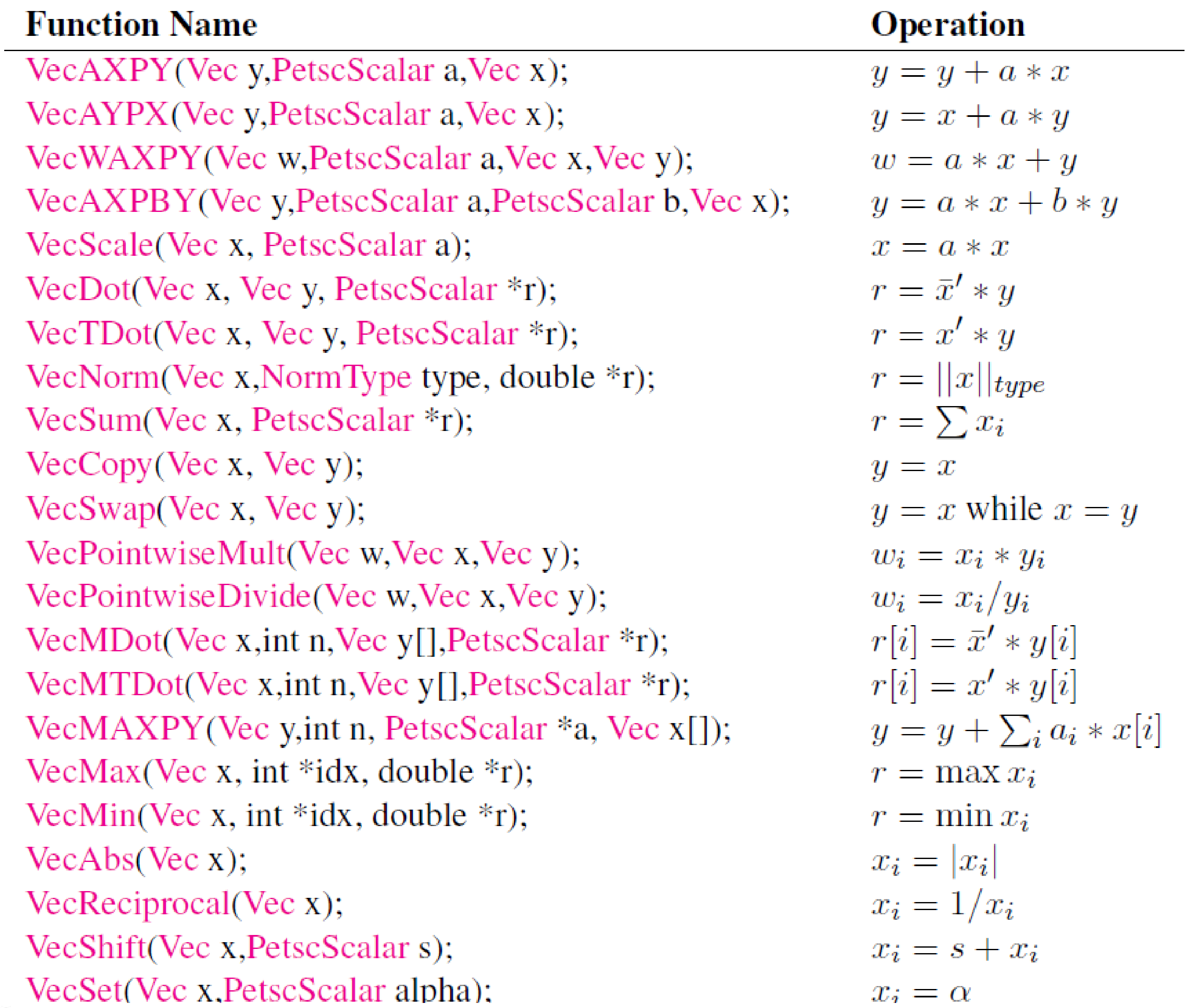
What are PETSc vectors?

- Fundamental objects for storing field solutions, right-hand sides, etc.
- Each process locally owns a subvector of contiguously numbered global indices

Features

- Has a direct interface to the values
- Supports all vector space operations
  - `VecDot()`, `VecNorm()`, `VecScale()`, …
- Also unusual ops, e.g. `VecSqrt()`, `VecInverse()`
- Automatic communication during assembly
- Customizable communication (scatters)

Numerical vector operations



Function Name	Operation
VecAXPY(Vec y,PetscScalar a,Vec x);	$y = y + a * x$
VecAYPX(Vec y,PetscScalar a,Vec x);	$y = x + a * y$
VecWAXPY(Vec w,PetscScalar a,Vec x,Vec y);	$w = a * x + y$
VecAXPBY(Vec y,PetscScalar a,PetscScalar b,Vec x);	$y = a * x + b * y$
VecScale(Vec x, PetscScalar a);	$x = a * x$
VecDot(Vec x, Vec y, PetscScalar *r);	$r = \bar{x}' * y$
VecTDot(Vec x, Vec y, PetscScalar *r);	$r = x' * y$
VecNorm(Vec x,NormType type, double *r);	$r = \|\|x\|\|_{type}$
VecSum(Vec x, PetscScalar *r);	$r = \sum x_i$
VecCopy(Vec x, Vec y);	$y = x$
VecSwap(Vec x, Vec y);	$y = x$ while $x = y$
VecPointwiseMult(Vec w,Vec x,Vec y);	$w_i = x_i * y_i$
VecPointwiseDivide(Vec w,Vec x,Vec y);	$w_i = x_i / y_i$
VecMDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = \bar{x}' * y[i]$
VecMTDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = x' * y[i]$
VecMAXPY(Vec y,int n, PetscScalar *a, Vec x[]);	$y = y + \sum_i a_i * x[i]$
VecMax(Vec x, int *idx, double *r);	$r = \max x_i$
VecMin(Vec x, int *idx, double *r);	$r = \min x_i$
VecAbs(Vec x);	$x_i = \|x_i\|$
VecReciprocal(Vec x);	$x_i = 1/x_i$
VecShift(Vec x,PetscScalar s);	$x_i = s + x_i$
VecSet(Vec x,PetscScalar alpha);	$x_i = \alpha$

Matrices

What are PETSc matrices?

- Fundamental objects for storing linear operators
- Each process locally owns a submatrix of contiguous rows

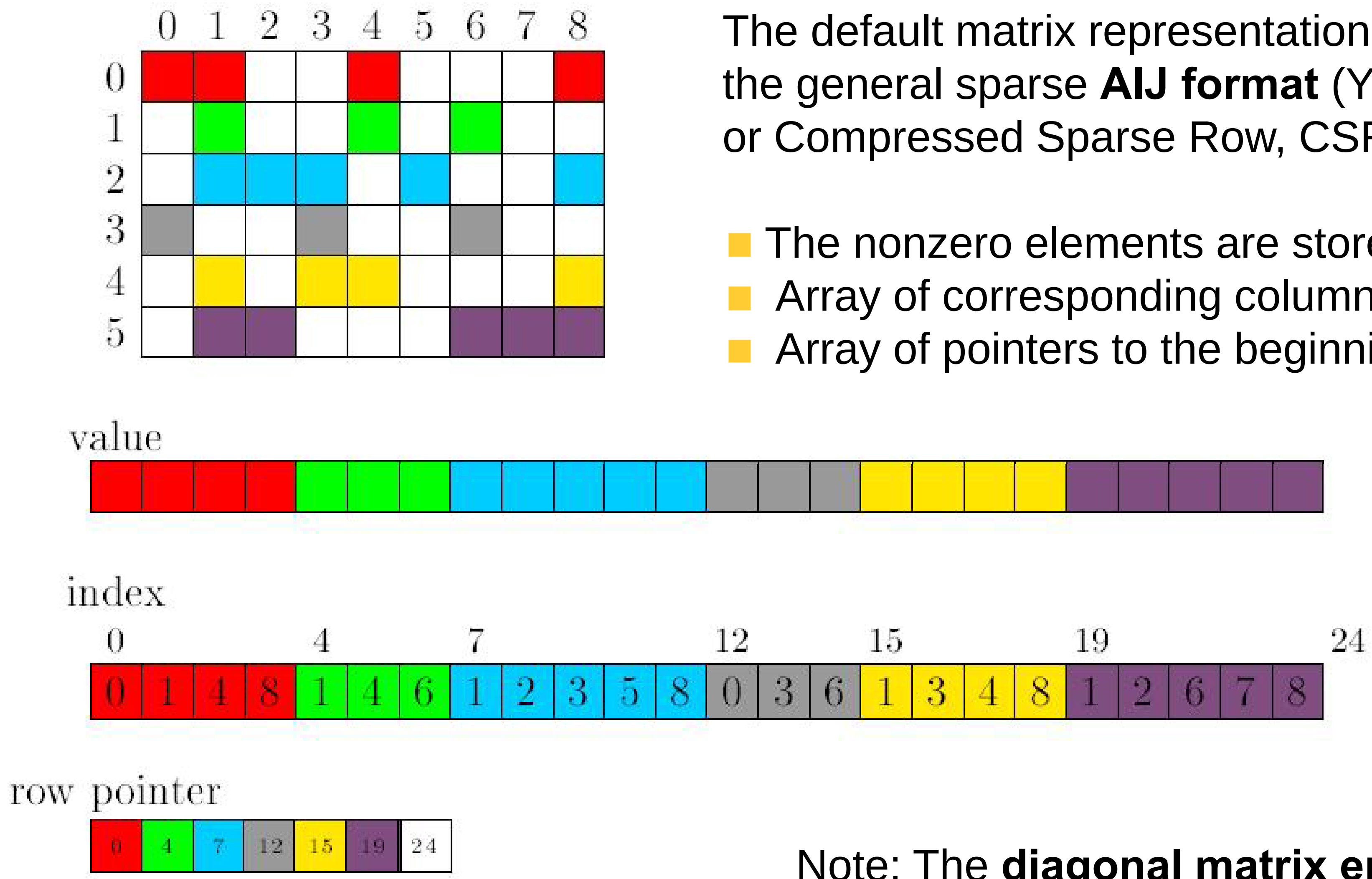
Features

- Supports many data types
  - AIJ, Block AIJ, Symmetric AIJ, Block Diagonal, etc.
- Supports structures for many packages
  - Spooles, MUMPS, SuperLU, UMFPack, DSCPack
- A matrix is defined by its interface, the operations that you can perform with it, not by its data structure

Numerical matrix operations

Function Name	Operation
MatAXPY(Mat Y, PetscScalar a,Mat X,MatStructure);	$Y = Y + a * X$
MatMult(Mat A,Vec x, Vec y);	$y = A * x$
MatMultAdd(Mat A,Vec x, Vec y,Vec z);	$z = y + A * x$
MatMultTranspose(Mat A,Vec x, Vec y);	$y = A^T * x$
MatMultTransposeAdd(Mat A,Vec x, Vec y,Vec z);	$z = y + A^T * x$
MatNorm(Mat A,NormType type, double *r);	$r = \|\|A\|\|_{type}$
MatDiagonalScale(Mat A,Vec l,Vec r);	$A = \text{diag}(l) * A * \text{diag}(r)$
MatScale(Mat A,PetscScalar a);	$A = a * A$
MatConvert(Mat A,MatType type,Mat *B);	$B = A$
MatCopy(Mat A,Mat B,MatStructure);	$B = A$
MatGetDiagonal(Mat A,Vec x);	$x = \text{diag}(A)$
MatTranspose(Mat A,MatReuse,Mat* B);	$B = A^T$
MatZeroEntries(Mat A);	$A = 0$
MatShift(Mat Y,PetscScalar a);	$Y = Y + a * I$

Matrix AIJ format



The default matrix representation within PETSc is the general sparse **AIJ format** (Yale sparse matrix or Compressed Sparse Row, CSR)

- The nonzero elements are stored by rows
- Array of corresponding column numbers
- Array of pointers to the beginning of each row

Note: The **diagonal matrix entries** are stored with the rest of the nonzeros

Parallel sparse matrices

Each process locally owns a submatrix of contiguously numbered global rows.

Each submatrix consists of **diagonal** and **off-diagonal** parts.

$$
\begin{array}{c}
\\ \text{P0} \\ \\ \\ \\ \text{P1} \\ \\ \\ \text{P2} \\
\end{array}
\left(\begin{array}{ccc|ccc|cc}
1 & 2 & 0 & 0 & 3 & 0 & 0 & 4 \\
0 & 5 & 6 & 7 & 0 & 0 & 8 & 0 \\
9 & 0 & 10 & 11 & 0 & 0 & 12 & 0 \\
\hline
13 & 0 & 14 & 15 & 16 & 17 & 0 & 0 \\
0 & 18 & 0 & 19 & 20 & 21 & 0 & 0 \\
0 & 0 & 0 & 22 & 23 & 0 & 24 & 0 \\
\hline
25 & 26 & 27 & 0 & 0 & 28 & 29 & 0 \\
30 & 0 & 0 & 31 & 32 & 33 & 0 & 34
\end{array}\right)
$$

KSP: linear equations solvers

The **object KSP** provides uniform and efficient access to all of the package's **linear system solvers**

KSP is intended for solving nonsingular systems of the form

$$Ax = b.$$

```
KSPCreate(MPI Comm comm, KSP *ksp)
KSPSetOperators(KSP ksp, Mat Amat, Mat Pmat,
                MatStructure flag)
KSPSolve(KSP ksp, Vec b, Vec x)
KSPGetIterationNumber(KSP ksp, int *its)
KSPDestroy(KSP ksp)
```

PETSc KSP methods

Method	KSPType	Options Database Name	Default Convergence Monitor†
Richardson	KSPRICHARDSON	richardson	true
Chebychev	KSPCHEBYCHEV	chebychev	true
Conjugate Gradient [11]	KSPCG	cg	true
BiConjugate Gradient	KSPBICG	bicg	true
Generalized Minimal Residual [15]	KSPGMRES	gmres	precond
BiCGSTAB [18]	KSPBCGS	bcgs	precond
Conjugate Gradient Squared [17]	KSPCGS	cgs	precond
Transpose-Free Quasi-Minimal Residual (1) [7]	KSPTFQMR	tfqmr	precond
Transpose-Free Quasi-Minimal Residual (2)	KSPTCQMR	tcqmr	precond
Conjugate Residual	KSPCR	cr	precond
Least Squares Method	KSPLSQR	lsqr	precond
Shell for no KSP method	KSPPREONLY	preonly	precond

†true - denotes true residual norm, precond - denotes preconditioned residual norm

PETSc cont.

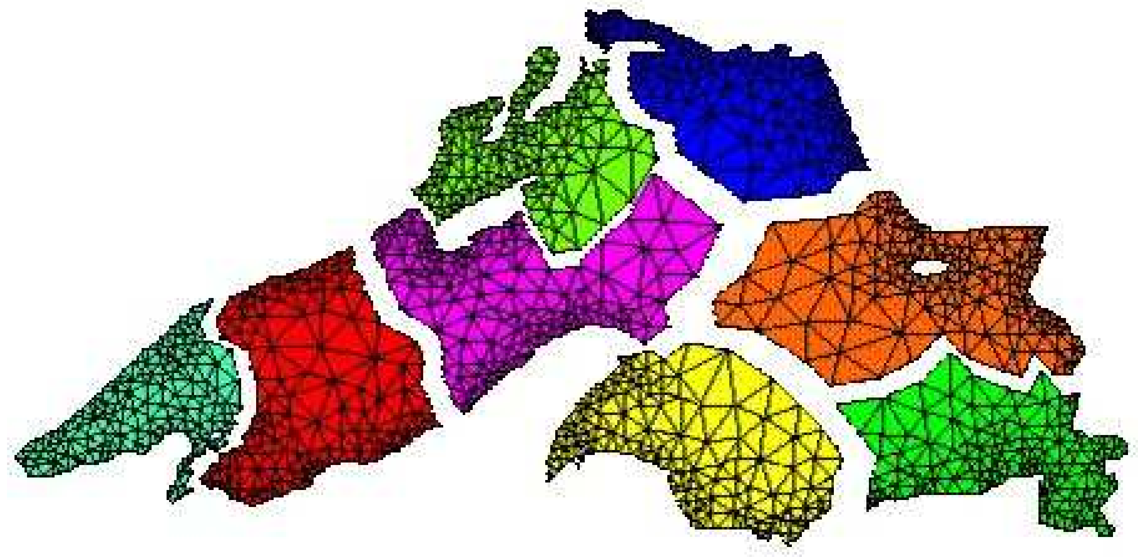
External packages:

- Hypre - a library for solving large, sparse linear systems of equations on massively parallel computers, LLNL
- SuperLU - a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines, Berkeley, Jim Demmel and coworkers
- SuperLU_DIST - SuperLU for distributed memory
- Sundials - SUite of Nonlinear and DIfferential/ALgebraic equation Solvers,
- FFTW - a C -library for computing the discrete Fourier transform (DFT) in various dimensions, of arbitrary input size, real and complex data
- Metis, ParMetis -

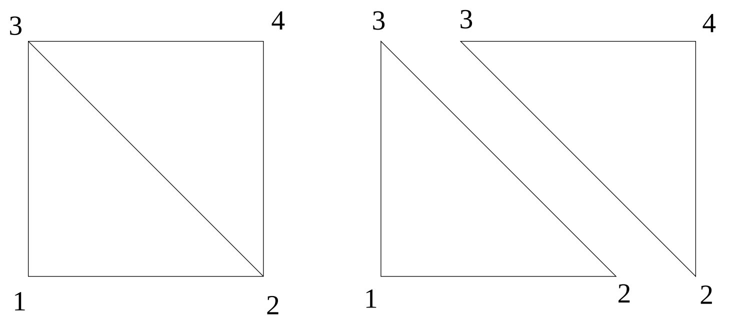
Added:
– Hybrid Chebychev
– pipelined GMRES which performs one non-blocking reduction per iteration instead of two blocking reductions
– flexible BiCGStab, tolerates a nonlinear preconditioner
– improved flexible BiCGStab, tolerates a nonlinear preconditioner and performs one reduction every other iteration

’Problem-given strategy’

Example: Metis - provides mesh/graph partitionings (nonoverlapping) domain decomposition



’Problem-given strategy’



$$A = \begin{bmatrix} 1 & -0.5 & -0.5 & 0 \\ -0.5 & 1 & 0 & -0.5 \\ -0.5 & 0 & 1 & -0.5 \\ 0 & -0.5 & -0.5 & 1 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 1 & -0.5 & -0.5 & 0 \\ -0.5 & 0.5 & 0 & 0 \\ -0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & -0.5 \\ 0 & 0 & 0.5 & -0.5 \\ 0 & -0.5 & -0.5 & 1 \end{bmatrix}$$

FEM-DD-unpreconditioned CG

Assume A and $\mathbf{b}$ are distributed and an initial guess $\mathbf{x}^{(0)}$ is given, which is replicated.

```
$\mathbf{g}^{(0)} = \mathbf{b} - A\mathbf{x}^{(0)}$
$\mathbf{r} = replicate(\mathbf{g}^{(0)})$
$\mathbf{d}^{(0)} = -\mathbf{r}$
$\delta_0 = (\mathbf{g}^{(0)}, \mathbf{r}^{(0)})$
For $k = 0, 1, \cdots$ until convergence
(1)  $\mathbf{h} = A\mathbf{d}^{(k)}$
(2)  $\tau = \delta_0/(\mathbf{h}, \mathbf{d}^{(k)})$
(3)  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \tau\mathbf{d}^{(k)}$
(4)  $\mathbf{g}^{(k+1)} = \mathbf{g}^{(k)} + \tau\mathbf{h}$
(5)  $\mathbf{r} = replicate(\mathbf{g}^{(k+1)})$
(6)  $\delta_1 = (\mathbf{g}^{(k+1)}, \mathbf{r})$
(7)  $\beta = \delta_1/\delta_0, \delta_0 = \delta_1$
(8)  $\mathbf{d}^{(k+1)} = \mathbf{r} + \beta\mathbf{d}^{(k)}$
```

Why local comunications are not enough?

Consider the solution of $A\mathbf{x} = \mathbf{b}$ by the standard conjugate gradient, where

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

The exact solution is $\hat{\mathbf{x}} = [1, 1, \cdots, 1]^T$.
Starting with $\mathbf{x}^0 = [0, 0, \cdots, 0]^T$ one finds that after k iterations

$$\mathbf{x}^k = \left[\frac{k}{k+1}, \frac{k-1}{k+1}, \cdots, \frac{1}{k+1}, 0, \cdots, 0\right]^T$$

for $1 \le k \le n-1$ and $\mathbf{x}^n = \hat{\mathbf{x}}$.
Hence, the information travels one step at a time from left to right and it takes $\boxed{n}$ steps before the last component has changed at all.

Include preconditioning: $C^{-1}A\mathbf{x} = C^{-1}\mathbf{b}$

Unpreconditioned CG

```
x = x0
r = A*x-b
delta0 = (r,r)
g = -r
Repeat: h = A*g
            tau = delta0/(g,h)
            x = x + tau*g
            r = r + tau*h
            delta1 = (r,r)
            if delta1 <= eps, stop
            beta = delta1/delta0
            g = -r + beta*g
```

Preconditioned CG

```
x = x0
r = A*x-b; C*h = r
delta0 = (r,h)
g = -h
Repeat: h = A*g
            tau = delta0/(g,h)
            x = x + tau*g
            r = r + tau*h; C*h = r
            delta1 = (r,h)
            if delta1 <= eps, stop
            beta = delta1/delta0
            g = -h + beta*g
```

FEM-DD- preconditioned CG

Assume A, B and $\mathbf{b}$ are distributed and the initial guess $\mathbf{x}^{(0)}$ is replicated.

$$
\begin{array}{lrcll}
 & \mathbf{g}^{(0)} & = & A\mathbf{x}^{(0)} - \mathbf{b}, & \mathbf{g}^{(0)} = \mathit{replicate}(\mathbf{g}^{(0)}) \\
 & \mathbf{h} & = & B\mathbf{g}^{(0)} & \\
 & \delta_0 & = & (\mathbf{g}^{(0)}, \mathbf{h}) & \mathbf{h} = \mathit{replicate}(\mathbf{h}) \\
 & \mathbf{d}^{(0)} & = & -\mathbf{h} & \\
\multicolumn{5}{l}{\textit{For } k = 0, 1, \cdots \textit{ until convergence}} \\
(1) & \mathbf{h} & = & A\mathbf{d}^{(k)} & \\
(2) & \tau & = & \delta_0 / (\mathbf{h}, \mathbf{d}^{(k)}) & \\
(3) & \mathbf{x}^{(k+1)} & = & \mathbf{x}^{(k)} + \tau\mathbf{d}^{(k)} & \\
(4) & \mathbf{g}^{(k+1)} & = & \mathbf{g}^{(k)} + \tau\mathbf{h}, & \mathbf{g}^{(k+1)} = \mathit{replicate}(\mathbf{g}^{(k+1)}) \\
(5) & \mathbf{h} & = & B\mathbf{g}^{(k+1)}, & \\
(6) & \delta_1 & = & (\mathbf{g}^{(k+1)}, \mathbf{h}) & \mathbf{h} = \mathit{replicate}(\mathbf{h}) \\
(7) & \beta & = & \delta_1 / \delta_0, & \delta_0 = \delta_1 \\
(8) & \mathbf{d}^{(k+1)} & = & -\mathbf{h} + \beta\mathbf{d}^{(k)} &
\end{array}
$$

deal.ii

I: Shared memory paradigm

- Task-based
- Thread-based

II: Distributed memory paradigm

- Each machine keeps the entire mesh and DoF handler locally, but only a share of the global matrix, sparsity pattern, and solution vector is stored on each machine.
- The mesh and DoFhandler are also distributed, i.e. each processor stores only a share of the cells and degrees of freedom. No processor has knowledge of the entire mesh, matrix, or solution, and in fact problems solved in this mode are usually so large (say, 100s of millions to billions of degrees of freedom) that no processor can or should store even a single solution vector.

CG - more efficient global ommunications?

Preconditioned Chronopoulos/Gear CG

$r_0 = b - A * x_0$
$C * u_0 = r_0$; $w_0 = A * u_0$ $\alpha = (r_0, u_0)/(w_0, u_0)$, $\beta_0 = 0$, $\gamma_0 = (r_0, u_0)$
Repeat:

$p_i = u_i + \beta_i p_{i-1}$	vector update
$s_i = w_i + \beta_i s_{i-1}$	vector update
$x_{i+1} = x_i + \alpha_i p_i$	vector update
$r_{i+1} = r_i - \alpha_i s_i$	vector update
$Cu_{i+1} = r_{i+1}$	system solve
$w_{i+1} = Au_{i+1}$	matrix-vector
$\gamma_{i+1} = (r_{i+1}, u_{i+1})$	scalar product
$\delta = (w_{i+1}, u_{i+1})$	scalar product
$\beta_{i+1} = \gamma_{i+1}/\gamma_i$	
$\alpha_{i+1} = \gamma_{i+1}/(\delta - \beta_{i+1}\gamma_{i+1}/\alpha_i)$	

The communication phase for both dot-products from the algorithm can be combined in a single global reduction. The update for x_i can be postponed and used to overlap the global reduction. However, even for small parallel machines the runtime of a single vector update will not be enough to fully cover the latency of the global communication.

CG - more efficient global ommunications?

Pipelined Chronopoulos/Gear CG

Hiding global synchronization latency in the PCG algorithm
P. Ghysels, W. Vanroose, December 2012

```
$r_0 = b - A * x_0$; $w_0 = A * u_0$
Repeat:
    $\gamma_i = (r_i, r_i)$                  scalar product
    $\delta = (w_i, r_i)$                    scalar product
    $q_{i+1} = Aw_{i+1}$                     matrix-vector
    if $i > 0$ then $\beta_i = \gamma_i/\gamma_{i-1}$, $\alpha_i = \gamma_i/(\delta - \beta_i\gamma_i/\alpha_{i-1})$
    else $\beta_i = 0$, $\alpha_i = \gamma_i/\delta$
    $z_i = q_i + \beta_i z_{i-1}$            vector update
    $s_i = w_i + \beta_i s_{i-1}$            vector update
    $p_i = r_i + \beta_i p_{i-1}$            vector update
    $x_{i+1} = x_i + \alpha_i p_i$           vector update
    $r_{i+1} = r_i - \alpha_i s_i$           vector update
    $w_{i+1} = w_i - \alpha_i z_i$           vector update
```

CG - more efficient global ommunications?

Danger: what is going on with the NUMERICAL STABILITY of the method?

- *However, with increasing s, the stability of the s-step Krylov basis deteriorates.*
- *We report on extensive numerical tests that show the stability of the pipelined CG and CR methods.*
- *Stability is also negatively impacted in the pipelined methods by the extra multiplication with the matrix A (r_i, u_i and w_i are replaced by $r_i = b - Ax_i$, $u_i = M^{-1}r_i$ and $w_i = Au_i$ every 50-th iteration.)*
- *A possible improvement might be to add a shift in the matrix-vector product, similar to what is also done in s-step Krylov methods and for pipelined GMRES. The CG iteration can provide information on the spectrum of A, which can be used to determine good shifts.*

To summarize: need a preconditioner; not only local communications for good numerical performance.