Projection-based iterative methods
Shortly on projectors
Projectors and properties

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

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

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

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

**P1:** \[ \mathcal{N}(P) \cap \mathcal{R}(P) = \{0\} \]
Indeed,
if \( x \in \mathcal{R}(P) \) \( \Rightarrow \exists y : y = Px \Rightarrow Py = P^2x = Px \Rightarrow y = x \Rightarrow x = Px \)
If \( x \in \mathcal{N}(P) \) \( \Rightarrow Px = 0 \Rightarrow x = Px \Rightarrow x = 0. \)

**P2:** \[ \mathcal{N}(P) = \mathcal{R}(I - P) \]
\( x \in \mathcal{N}(P) \) \( \Rightarrow Px = 0. \) Then \( x = Ix - Px = (I - P)x. \)
\( y \in \mathcal{R}(I - P) \) \( \Rightarrow y = (I - P)y \Rightarrow Py = 0. \)

**P3:** \[ C^n = \mathcal{R}(P) \bigoplus \mathcal{N}(P). \]

**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) \( \forall x \in C^n \exists \) unique vector \( y : y \in K, x - y \in L. \)

**Proof** (i)\( \Rightarrow \) (ii): \( K \cap L^\perp = \{\emptyset\} \Rightarrow C^n = K \bigoplus L^\perp \Rightarrow \forall x \in C^n : x = y + z, \)
where \( y \in K \) and \( z \in L^\perp. \) Thus, \( z = x - y \Rightarrow \) (ii).
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 $Px \in K$ and $(I - P)x \perp K$. Equivalently, $((I - P)x, y) = 0, \forall y \in K$. 
\textbf{Properties (cont.)}

\[ P6: \] If $P$ is orthogonal, then $\|P\| = 1$.

\[ P7: \] 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 $x \in \mathbb{C}^n$ there holds

$$\min_{y \in K} \|x - y\|_2 = \|x - Px\|_2.$$  \hspace{1cm} (1)

**Proof** For any $y \in K$, $Px - y \in K$, $Px \in K$, $(I - P)x \perp K$

$$\|x - y\|_2^2 = \|(x - Px) + (Px - y)\|_2^2 = \|x - Px\|_2^2 + \|Px - y\|_2^2 + 2(x - Px, Px - y) = \|x - Px\|_2^2 + \|Px - y\|_2^2.$$

Therefore, $\|x - y\|_2^2 \geq \|x - Px\|_2^2 \forall y \in K$ and the minimum is reached for $y = Px$. 

**Corollary 1** Let $K \subset \mathbb{C}^n$ and $x \in \mathbb{C}^n$ be given. Then $\min_y \|x - y\|_2 = \|x - y^*\|_2$ is equivalent to $y^* \in K$ and $x - y^* \perp K$. 
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)
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 \( b - A x = 0, \ b, \ x \in R^n, \ A \in R^{n \times n} \)

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

\[
\text{find } \tilde{x} \in x^{(0)} + K, \text{ such that } b - A \tilde{x} \perp L
\]

\( K \) - 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 $b - Ax = 0$
Given $x^0$, $K$ and $L$,

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

Notations:

$\tilde{x} = x^0 + \delta$ ($\delta$ – correction)
$r^0 = b - Ax^0$ ($r^0$ – residual)

* find $\delta \in K$, such that $r^0 - A\delta \perp L$
Matrix formulation

Choose a basis in $K$ and $L$: $V = \{v_1, v_2, \cdots, v_m\}$ and $W = \{w_1, w_2, \cdots, w_m\}$. Then, $\tilde{x} = x^0 + \delta = x^0 + V y$ for some $y \in \mathbb{R}^m$.

The orthogonality condition can be written as

\[(** \hspace{1cm} W^T (r^0 - AV y) \)\]

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

\[
W^T r^0 = W^T A V y
\]
\[
y = (W^T A V)^{-1} W^T r^0
\]
\[
\tilde{x} = x^0 + V (W^T A V)^{-1} W^T r^0
\]

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

!!! $W^T A V$ should be invertible.
A prototype projection-based iterative method:

Given \( x^{(0)} ; x = x^{(0)} \)

Until convergence do:

Choose \( K \) and \( L \)

Choose basis \( V \) in \( K \) and \( W \) in \( L \)

Compute \( r = b - Ax \)

\[
    y = (W^T AV)^{-1} W^T r
\]

\[
    x = x + V y
\]

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 $\tilde{x}$ is an oblique projection on $K$ orthogonally to $AK$ with a starting vector $x^0$ if and only if $\tilde{x}$ minimizes the 2-norm of the residual over $x^0 + K$, i.e.,

$$\|r - A\tilde{x}\|_2 = \min_{x \in x^0 + K} \|r - Ax\|_2.$$  \hspace{1cm} (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., $x^0 - A\tilde{x} \perp K$. Then a vector $\tilde{x}$ is an orthogonal projection onto $K$ with a starting vector $x^0$ if and only if it minimizes the $A$-norm of the error $e = x^* - x$ over $x^0 + K$, i.e.,

$$\|x^* - \tilde{x}\|_A = \min_{x \in x^0 + K} \|x^* - x\|_A. \quad (3)$$

The error decreases monotonically in the $A$-norm.

**Error-projection methods**
**Example: m = 1**

Consider two vectors: \( d \) and \( e \). Let \( K = \text{span}\{d\} \) and \( L = \text{span}\{e\} \).

Then \( \tilde{x} = x^0 + \alpha d \ (\delta = \alpha d) \) and the orthogonality condition reads as:

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

If \( d = e \) - Steepest Descent method (minimization on a line.

If we minimize over a plane - ORTHOMIN.
Choice of $K$:

\[ K = \mathcal{K}^m(A, v) = \{v, Av, A^2v, \ldots, A^{m-1}v\} \]

Krylov subspace methods

- $L = K = \mathcal{K}^m(A, r^0)$ and $A$ spd $\Rightarrow$ CG
- $L = AK = A\mathcal{K}^m(A, r^0)$ $\Rightarrow$ 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 $L$: $V = \{v_1, v_2, \cdots, v_m\}$ and 
$W = \{w_1, w_2, \cdots, w_m\}$.
Then, $\tilde{x} = x^0 + \delta = x^0 + Vy$ for some $y \in R^m$.

The orthogonality condition can be written as 

\[
(**) \quad W^T(r^0 - AVy)
\]

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

\[
W^Tr^0 = W^TAVy

y = (W^TAV)^{-1}W^Tr^0

\tilde{x} = x^0 + V(W^TAV)^{-1}W^Tr^0
\]
Arnoldi’s method for general matrices

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

1. Choose a vector $v_1$ such that $\|v_1\| = 1$
2. For $j = 1, 2, \cdots, m$
3. For $i = 1, 2, \cdots, j$
4. $h_{ij} = (Av_j, v_i)$
5. End
6. $w_j = Av_j - \sum_{i=1}^{j} h_{ij}v_i$
7. $h_{j+1,j} = \|w_j\|$
8. If $h_{j+1,j} = 0$, stop
9. $v_{j+1} = w_j / h_{j+1,j}$
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 $j$. 


\[ p. 23/49 \]
The result of Arnoldi’s process

\[ V^m = \{v_1, v_2, \cdots, v_m\} \] is an orthonormal basis in \( \mathcal{K}^m(A, v) \)

\[ AV^m = V^m H^m + w_{m+1} e_m^T \]
Arnoldi’s process - example

\[ H^3 = \begin{bmatrix}
(Av_1, v_1) & (Av_2, v_1) & (Av_3, v_1) \\
\|w_1\| & (Av_2, v_2) & (Av_3, v_2) \\
0 & \|w_2\| & (Av_3, v_3)
\end{bmatrix} \]

Since \( V^{m+1} \perp \{v_1, v_2, \ldots, v_m\} \) then it follows that \( (V^m)^T AV^m = H^m \). 
\( H^m \) is an upper-Hessenberg matrix.
Arnoldi’s method for symmetric matrices

Let now $A$ be a 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

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

Thus, the vectors $v^j$ satisfy a three-term recursion:

$$\beta_{i+1}v^{i+1} = Av^i - \gamma_i v^i - \beta_i v^{i-1}$$
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 \( k = 0, 1, \ldots \), 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
\[ x = x_0 \]
\[ r = A \cdot x - b \]
\[ \text{delta0} = (r, r) \]
\[ g = -r \]

Repeat:
\[ h = A \cdot g \]
\[ \text{tau} = \text{delta0} / (g, h) \]
\[ x = x + \text{tau} \cdot g \]
\[ r = r + \text{tau} \cdot h \]
\[ \text{delta1} = (r, r) \]
if \( \text{delta1} \leq \epsilon \), stop
\[ \text{beta} = \text{delta1} / \text{delta0}; \quad \text{delta0} = \text{delta1} \]
\[ g = -r + \text{beta} \cdot g \]

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 $g^j = 0$ then $\tau_k$ is not defined. The vectors $g^j$ are computed from the formula $g^k = r^{(k)} + \beta_k g^{k-1}$. Then
  
  $0 = (r^{(k)}, g^j) = -(r^{(k)}, r^{(k)}) + \beta_k \underbrace{(r^{(k)}, g^{k-1})}_{0} \Rightarrow r^{(k)} 0$, i.e., the solution is already found.

  As soon as $x^{(k)} \neq x_{exact}$, then $r^{(k)} \neq 0$ and then $g^{k+1} \neq 0$.
  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 $x_{exact} = x^{(m)}$ 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 $\tilde{P}(\lambda) \in \Pi_k^1$ and some constant $M$ there holds $\max_{\lambda \in S} |\tilde{P}(\lambda)| \leq M$. Then,

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

Then,

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

Seek now the smallest $k$, such that

$$\|e^k\|_A \leq \varepsilon \|e^0\|_A$$

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

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

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 \( \overline{H}_m = \{h_{ik}\}, 1 \leq i \leq m + 1, 1 \leq k \leq m \)
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.
Theorem: Let $A$ be diagonalizable, $A = X^{-1} \Lambda X$ where $\Lambda = \text{diag}\{\lambda_1, \cdots, \lambda_n\}$ contains the eigenvalues of $A$. Define

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

Then, the residual norm at the $m$th step of GMRES satisfies

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

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:

\[ x = x_0 \]
\[ r = A \cdot x - b \]
\[ \delta_0 = (r, r) \]
\[ g = -r \]

Repeat: \[ h = A \cdot g \]

\[ \tau = \delta_0 / (g, h) \]
\[ x = x + \tau \cdot g \]
\[ r = r + \tau \cdot h \]
\[ \delta_1 = (r, r) \]

If \[ \delta_1 \leq \varepsilon \], stop

\[ \beta = \delta_1 / \delta_0, \; \delta_0 = \delta_1 \]
\[ g = -r + \beta \cdot g \]

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
Introduction to PETSc

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, ...
Introduction to PETSc

Relationship between libraries

- TAO
  - PETSc
  - ScaLAPACK
  - BLACS
  - MPI
- PARPACK
- ARPACK
  - GSL
  - PBLAS
  - BLAS
- LAPACK
  - MKL
  - ACML
  - ESSL
## Introduction to PETSc

### PETSc numerical component

<table>
<thead>
<tr>
<th><strong>Nonlinear Solvers</strong></th>
<th><strong>Time Steppers</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-based Methods</td>
<td>Euler</td>
</tr>
<tr>
<td>Line Search</td>
<td>Backward Euler</td>
</tr>
<tr>
<td>Trust Region</td>
<td>Pseudo-Time Stepping</td>
</tr>
<tr>
<td>Other</td>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Krylov Subspace Methods</strong></th>
<th><strong>Preconditioners</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>GMRES</td>
<td>Additive Schwarz</td>
</tr>
<tr>
<td>CG</td>
<td>Block Jacobi</td>
</tr>
<tr>
<td>CGS</td>
<td>Jacobi</td>
</tr>
<tr>
<td>Bi-CG-Stab</td>
<td>ILU</td>
</tr>
<tr>
<td>TFQMR</td>
<td>ICC</td>
</tr>
<tr>
<td>Richardson</td>
<td>LU (sequential only)</td>
</tr>
<tr>
<td>Chebychev</td>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Matrices</strong></th>
<th><strong>Index Sets</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed Sparse Row (AIJ)</td>
<td>Indices</td>
</tr>
<tr>
<td>Block Compressed Sparse Row (BAIJ)</td>
<td>Block Indices</td>
</tr>
<tr>
<td>Block Diagonal (BDiag)</td>
<td>Stride</td>
</tr>
<tr>
<td>Dense</td>
<td>Other</td>
</tr>
</tbody>
</table>
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

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

Numerical matrix operations

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatAXPY Mat Y, PetscScalar a, Mat X, MatStructure</td>
<td>( Y = Y + a \times X )</td>
</tr>
<tr>
<td>MatMult Mat A, Vec x, Vec y</td>
<td>( y = A \times x )</td>
</tr>
<tr>
<td>MatMultAdd Mat A, Vec x, Vec y, Vec z</td>
<td>( z = y + A \times x )</td>
</tr>
<tr>
<td>MatMultTranspose Mat A, Vec x, Vec y</td>
<td>( y = A^T \times x )</td>
</tr>
<tr>
<td>MatMultTransposeAdd Mat A, Vec x, Vec y, Vec z</td>
<td>( z = y + A^T \times x )</td>
</tr>
<tr>
<td>MatNorm Mat A, NormType type, double *r</td>
<td>( r =</td>
</tr>
<tr>
<td>MatDiagonalScale Mat A, Vec l, Vec r</td>
<td>( A = \text{diag}(l) \times A \times \text{diag}(r) )</td>
</tr>
<tr>
<td>MatScale Mat A, PetscScalar a</td>
<td>( A = a \times A )</td>
</tr>
<tr>
<td>MatConvert Mat A, MatType type, Mat *B</td>
<td>( B = A )</td>
</tr>
<tr>
<td>MatCopy Mat A, Mat B, MatStructure</td>
<td>( B = A )</td>
</tr>
<tr>
<td>MatGetDiagonal Mat A, Vec x</td>
<td>( x = \text{diag}(A) )</td>
</tr>
<tr>
<td>MatTranspose Mat A, MatReuse, Mat* B</td>
<td>( B = A^T )</td>
</tr>
<tr>
<td>MatZeroEntries Mat A</td>
<td>( A = 0 )</td>
</tr>
<tr>
<td>MatShift Mat Y, PetscScalar a</td>
<td>( Y = Y + a \times I )</td>
</tr>
</tbody>
</table>
Matrix AIJ format

The default matrix representation within PETSc is the general sparse **AIJ format** (Yale sparse matrix or Compressed Sparse 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.

\[
\begin{pmatrix}
  1 & 2 & 0 \\
  0 & 5 & 6 \\
  9 & 0 & 10 \\
\end{pmatrix}
\begin{pmatrix}
  0 & 3 & 0 & 0 & 4 \\
  7 & 0 & 0 & 8 & 0 \\
  11 & 0 & 0 & 12 & 0 \\
\end{pmatrix}
\begin{pmatrix}
  13 & 0 & 14 \\
  0 & 18 & 0 \\
  0 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
  15 & 16 & 17 \\
  19 & 20 & 21 \\
  22 & 23 & 0 \\
\end{pmatrix}
\begin{pmatrix}
  0 & 0 \\
  0 & 0 \\
  24 & 0 \\
\end{pmatrix}
\begin{pmatrix}
  25 & 26 & 27 \\
  30 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
  0 & 0 & 28 \\
  31 & 32 & 33 \\
\end{pmatrix}
\begin{pmatrix}
  29 & 0 \\
  0 & 34 \\
\end{pmatrix}
\]
**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

\[ Ax = b. \]

```c
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

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

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

\[ A = \begin{pmatrix}
1 & -0.5 & -0.5 & 0 \\
-0.5 & 1 & 0 & -0.5 \\
-0.5 & 0 & 1 & -0.5 \\
0 & -0.5 & -0.5 & 1
\end{pmatrix} \]

\[ A_1 = \begin{pmatrix}
1 & -0.5 & -0.5 & 0 \\
-0.5 & 0.5 & 0 & 0 \\
-0.5 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix} \]

\[ A_2 = \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0.5 & 0 & -0.5 \\
0 & 0 & 0.5 & -0.5 \\
0 & -0.5 & -0.5 & 1
\end{pmatrix} \]
**FEM-DD-unpreconditioned CG**

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

\[
\begin{align*}
    g^{(0)} &= b - Ax^{(0)} \\
    r &= \text{replicate}(g^{(0)}) \\
    d^{(0)} &= -r \\
    \delta_0 &= (g^{(0)}, r^{(0)}) \\

    \text{For } k = 0, 1, \cdots \text{ until convergence} \\
    (1) \quad h &= Ad^{(k)} \\
    (2) \quad \tau &= \delta_0 / (h, d^{(k)}) \\
    (3) \quad x^{(k+1)} &= x^{(k)} + \tau d^{(k)} \\
    (4) \quad g^{(k+1)} &= g^{(k)} + \tau h \\
    (5) \quad r &= \text{replicate}(g^{(k+1)}) \\
    (6) \quad \delta_1 &= (g^{(k+1)}, r) \\
    (7) \quad \beta &= \delta_1 / \delta_0, \delta_0 = \delta_1 \\
    (8) \quad d^{(k+1)} &= r + \beta d^{(k)}
\end{align*}
\]
Why local communications are not enough?

Consider the solution of $Ax = b$ by the standard conjugate gradient, where

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

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

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

for $1 \leq k \leq n - 1$ and $x^n = \hat{x}$. Hence, the information travels one step at a time from left to right and it takes $n$ steps before the last component has changed at all.
Include preconditioning:  \( C^{-1}Ax = C^{-1}b \)

Unpreconditioned CG
\[
\begin{align*}
x &= x_0 \\
r &= A^*x - b \\
delta_0 &= (r,r) \\
g &= -r \\
\text{Repeat: } h &= A^*g \\
\quad \text{tau} &= \delta_0/(g,h) \\
\quad x &= x + \text{tau} \cdot g \\
\quad r &= r + \text{tau} \cdot h \\
delta_1 &= (r,r) \\
\quad \text{if } \delta_1 \leq \text{eps}, \text{ stop} \\
\quad \beta &= \delta_1/\delta_0 \\
\quad g &= -r + \beta \cdot g
\end{align*}
\]

Preconditioned CG
\[
\begin{align*}
x &= x_0 \\
r &= A^*x - b; \quad \mathbf{C}^*h = r \\
delta_0 &= (r,h) \\
g &= -h \\
\text{Repeat: } h &= \mathbf{A}^*g \\
\quad \text{tau} &= \delta_0/(g,h) \\
\quad x &= x + \text{tau} \cdot g \\
\quad r &= r + \text{tau} \cdot h \\
delta_1 &= (r,h) \\
\quad \text{if } \delta_1 \leq \text{eps}, \text{ stop} \\
\quad \beta &= \delta_1/\delta_0 \\
\quad g &= -h + \beta \cdot g
\end{align*}
\]
### FEM-DD- preconditioned CG

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$h = Ad^{(k)}$</td>
</tr>
<tr>
<td>2</td>
<td>$\tau = \delta_0 / (h, d^{(k)})$</td>
</tr>
<tr>
<td>3</td>
<td>$x^{(k+1)} = x^{(k)} + \tau d^{(k)}$</td>
</tr>
<tr>
<td>4</td>
<td>$g^{(k+1)} = g^{(k)} + \tau h$, $g^{(k+1)} = \text{replicate}(g^{(k+1)})$</td>
</tr>
<tr>
<td>5</td>
<td>$h = Bg^{(k+1)}$</td>
</tr>
<tr>
<td>6</td>
<td>$\delta_1 = (g^{(k+1)}, h)$, $h = \text{replicate}(h)$</td>
</tr>
<tr>
<td>7</td>
<td>$\beta = \delta_1 / \delta_0$, $\delta_0 = \delta_1$</td>
</tr>
<tr>
<td>8</td>
<td>$d^{(k+1)} = -h + \beta d^{(k)}$</td>
</tr>
</tbody>
</table>
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 communications?**

Preconditioned Chronopoulos/Gear CG

\[ r_0 = b - A \ast x_0 \]
\[ C \ast u_0 = r_0; \ w_0 = A \ast u_0 \]
\[ \alpha = (r_0, u_0)/(w_0, u_0), \ \beta_0 = 0, \ \gamma_0 = (r_0, u_0) \]

Repeat:

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

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 communications?

Pipelined Chronopoulos/Gear CG

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

\[ r_0 = b - A \times x_0; \quad w_0 = A \times u_0 \]

Repeat:

\[ \gamma_i = (r_i, r_i) \quad \text{scalar product} \]
\[ \delta = (w_i, r_i) \quad \text{scalar product} \]
\[ q_{i+1} = Aw_{i+1} \quad \text{matrix-vector} \]

if \( i > 0 \) then \( \beta_i = \gamma_i / \gamma_{i-1}, \alpha_i = \gamma_i / (\delta - \beta_i \gamma_i / \alpha_{i-1}) \)
else \( \beta_i = 0, \alpha_i = \gamma_i / \delta \)

\[ z_i = q_i + \beta_i z_{i-1} \quad \text{vector update} \]
\[ s_i = w_i + \beta_i s_{i-1} \quad \text{vector update} \]
\[ p_i = r_i + \beta_i p_{i-1} \quad \text{vector update} \]
\[ x_{i+1} = x_i + \alpha_i p_i \quad \text{vector update} \]
\[ r_{i+1} = r_i - \alpha_i s_i \quad \text{vector update} \]
\[ w_{i+1} = w_i - \alpha_i z_i \quad \text{vector update} \]
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.