

SOLVERS AND PRECONDITIONERS.

Paolo Crosetto

Modeling and Scientific Computing, EPF Lausanne

Lausanne, January 7th 2010



FSI FRAMEWORK

FSI problem consists in

- 3D NS equations in ALE form;
- 3D linear elastic wall;
- continuity of velocity and stresses through the interface;
- equation for the fluid domain displacement at each iteration.

Time discretization

- Geometry-Convective Explicit;
- Convective Explicit;
- Full Implicit;

Nonlinearities

- Inexact Newton
- Exact Newton (shape derivatives)

GCE: Linear system

$$A = \begin{pmatrix} F & C_1 \\ C_3 & S \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{d}_s \end{pmatrix} = \begin{pmatrix} \mathbf{f}_F \\ \mathbf{f}_S \end{pmatrix}$$

- P_{GS-AS} : modular, cheap, scalable

FSI FRAMEWORK

FSI problem consists in

- 3D NS equations in ALE form;
- 3D linear elastic wall;
- continuity of velocity and stresses through the interface;
- equation for the fluid domain displacement at each iteration.

Time discretization

- Geometry-Convective Explicit;
- Convective Explicit;
- Full Implicit;

Nonlinearities

- Inexact Newton
- Exact Newton (shape derivatives)

GCE: Linear system

$$A = \begin{pmatrix} F & C_1 \\ C_3 & S \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{d}_s \end{pmatrix} = \begin{pmatrix} \mathbf{f}_F \\ \mathbf{f}_S \end{pmatrix}$$

- P_{GS-AS} : modular, cheap, scalable

FSI FRAMEWORK

FSI problem consists in

- 3D NS equations in ALE form;
- 3D linear elastic wall;
- continuity of velocity and stresses through the interface;
- equation for the fluid domain displacement at each iteration.

Time discretization

- Geometry-Convective Explicit;
- Convective Explicit;
- Full Implicit;

Nonlinearities

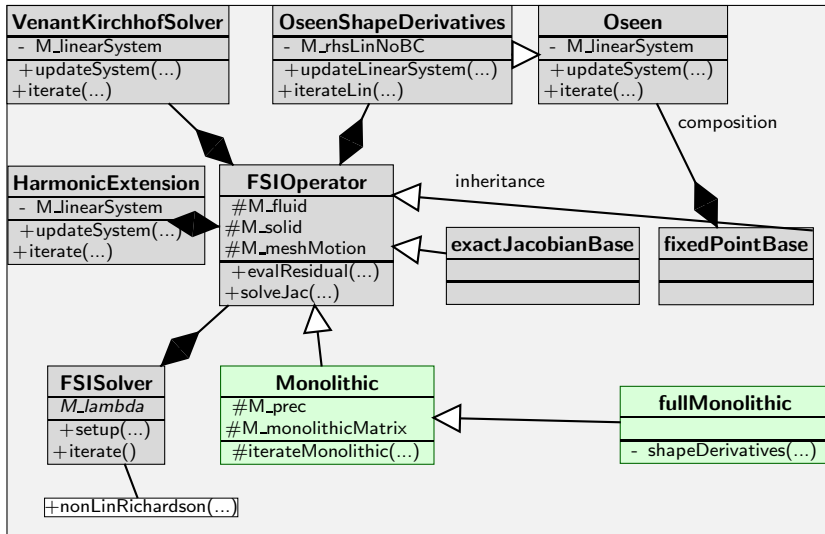
- Inexact Newton
- Exact Newton (shape derivatives)

CE-FI: Linearized system

$$A = \begin{pmatrix} F & C_1 & C_2 \\ C_3 & S & C_4 \\ 0 & C_5 & H \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{d}_s \\ \mathbf{d}_f \end{pmatrix} = \begin{pmatrix} \mathbf{f}_F \\ \mathbf{f}_S \\ \mathbf{f}_H \end{pmatrix}$$

- P_{GS-AS} : modular, cheap, scalable

FSI SOLVER CLASSES



CLASS OF PARALLEL PRECONDITIONERS

In **LifeV**: class for generic parallel preconditioners of **algebraic additive Schwarz** type (based on Trilinos/IFPACK). Among the **requested** features:

- given a matrix A , the class builds an overlapping Schwarz preconditioner $P_{AS}(A)$.
- local direct solver: LU by default (KLU, UMFPACK, ...)
- it is possible to **compose** preconditioning strategies

Example: GCE:

$$A = \begin{pmatrix} F & C_1 \\ C_3 & S \end{pmatrix} = \begin{pmatrix} F & 0 \\ C_3 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & S \end{pmatrix}$$

Idea: develop a class handling **preconditioners composed** by many factors.

CLASS OF PARALLEL PRECONDITIONERS

In **LifeV**: class for generic parallel preconditioners of **algebraic additive Schwarz** type (based on Trilinos/IFPACK). Among the **requested** features:

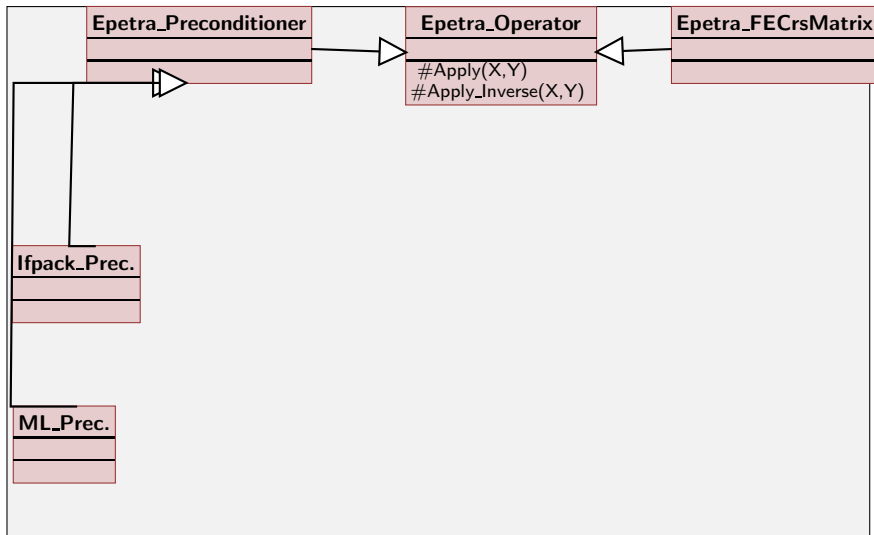
- given a matrix A , the class builds an overlapping Schwarz preconditioner $P_{AS}(A)$.
- local direct solver: LU by default (KLU, UMFPACK, ...)
- it is possible to **compose** preconditioning strategies

Example: GCE:

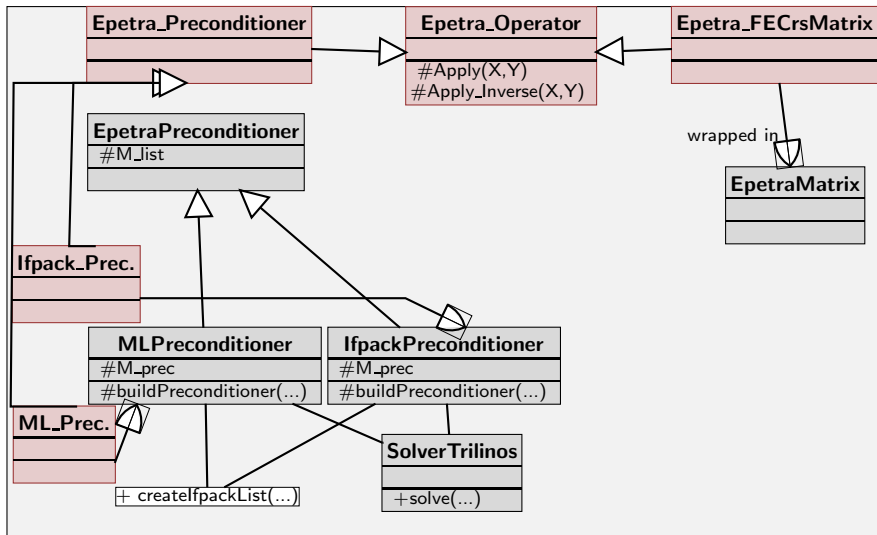
$$P = \begin{pmatrix} F & 0 \\ C_3 & S \end{pmatrix} = \begin{pmatrix} F & 0 \\ C_3 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & S \end{pmatrix}$$

Idea: develop a class handling **preconditioners composed** by many factors.

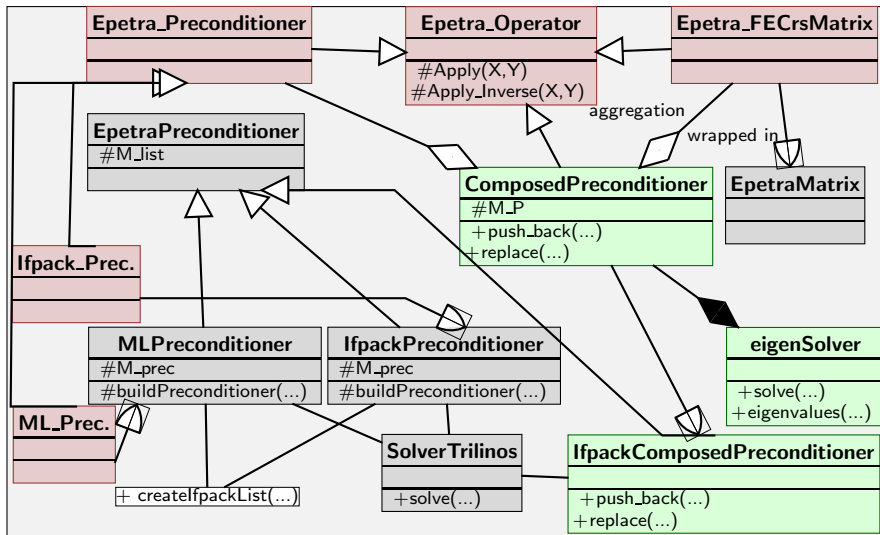
PRECONDITIONER CLASSES



PRECONDITIONER CLASSES



PRECONDITIONER CLASSES



EXAMPLE OF USAGE

find $\sigma_{\max}(P^{-1}A) = \sqrt{\rho(P^{-1}AA^T P^{-T})}$.

- Construction of $P^{-1}AA^T P^{-T}$. Given the shared pointers A and P:

- create a shared pointer PAAP to

```
ComposedPreconditioner<Epetra_Operator>;
```

- push back each factor

```
 $P^{-1}$  :PAAP->push_back(P, true)
```

```
A :PAAP->push_back(A)
```

```
 $A^T$  :PAAP->push_back(A, false, true)
```

```
 $P^{-T}$  :PAAP->push_back(P, true, true)
```

- Solving the eigenproblem:

```
1 EigenSolver eig(M_PAAP, M_PAAP->OperatorDomainMap(), 1);
```

```
2 eig.setDataFromGetPot(dataFile, "eigensolver/");
```

```
3 eig.solve();std::vector<LifeV::Real> real, imaginary;
```

```
4 eig.eigenvalues(real, imaginary);
```

EXAMPLE OF USAGE

$$\text{find } \sigma_{\max}(P^{-1}A) = \sqrt{\rho(P^{-1}AA^T P^{-T})}.$$

- Construction of $P^{-1}AA^T P^{-T}$. Given the shared pointers A and P:

- ① create a shared pointer PAAP to

```
ComposedPreconditioner<Epetra_Operator>;
```

- ② push back each factor

```
 $P^{-1}$  :PAAP->push_back(P, true)
```

```
 $A$  :PAAP->push_back(A)
```

```
 $A^T$  :PAAP->push_back(A, false, true)
```

```
 $P^{-T}$  :PAAP->push_back(P, true, true)
```

- Solving the eigenproblem:

```
① EigenSolver eig(M_PAAP, M_PAAP->OperatorDomainMap(), 1);
```

```
② eig.setDataFromGetPot(dataFile, "eigensolver/");
```

```
③ eig.solve();std::vector<LifeV::Real> real, imaginary;
```

```
④ eig.eigenvalues(real, imaginary);
```

EXAMPLE OF USAGE

$$\text{find } \sigma_{\max}(P^{-1}A) = \sqrt{\rho(P^{-1}AA^T P^{-T})}.$$

- Construction of $P^{-1}AA^T P^{-T}$. Given the shared pointers A and P:
 - 1 create a shared pointer PAAP to
`ComposedPreconditioner<Epetra_Operator>;`
 - 2 push back each factor
 - P^{-1} :PAAP->push_back(P, true)
 - A :PAAP->push_back(A)
 - A^T :PAAP->push_back(A, false, true)
 - P^{-T} :PAAP->push_back(P, true, true)
- Solving the eigenproblem:
 - 1 `EigenSolver eig(M_PAAP, M_PAAP->OperatorDomainMap(), 1);`
 - 2 `eig.setDataFromGetPot(dataFile, "eigensolver/");`
 - 3 `eig.solve();std::vector<LifeV::Real> real, imaginary;`
 - 4 `eig.eigenvalues(real, imaginary);`

KEY FEATURES

- 1 When calling `IfpackComposedPreconditioner::push_back(P)` by default an additive Schwarz preconditioner for the factor P is computed.
- 2 Note that P in the previous slide may be of `IfpackComposedPreconditioner` type as well.
- 3 The eigenSolver implemented in LifeV **does not work** directly with `EpetraMatrix` and `EpetraPreconditioner`: use `ComposedPreconditioner` class instead.
- 4 A matrix can be pushed into a composed preconditioner, but an attempt to use its inverse would cause a Trilinos exception.
- 5 $P(A)$ preconditioner computed with the matrix A :
 - `Epetra_Preconditioner::Apply(...)` applies the matrix A used to compute $P(A)$;
 - `Epetra_Preconditioner::Apply_Inverse(...)` applies $P(A)^{-1}$

KEY FEATURES

- 1 When calling `IfpackComposedPreconditioner::push_back(P)` by default an additive Schwarz preconditioner for the factor P is computed.
- 2 Note that P in the previous slide may be of `IfpackComposedPreconditioner` type as well.
- 3 The eigenSolver implemented in LifeV **does not work** directly with `EpetraMatrix` and `EpetraPreconditioner`: use `ComposedPreconditioner` class instead.
- 4 A matrix can be pushed into a composed preconditioner, but an attempt to use its inverse would cause a Trilinos exception.
- 5 $P(A)$ preconditioner computed with the matrix A :
 - `Epetra_Preconditioner::Apply(...)` applies the matrix A used to compute $P(A)$;
 - `Epetra_Preconditioner::Apply_Inverse(...)` applies $P(A)^{-1}$

KEY FEATURES

- 1 When calling `IfpackComposedPreconditioner::push_back(P)` by default an additive Schwarz preconditioner for the factor P is computed.
- 2 Note that P in the previous slide may be of `IfpackComposedPreconditioner` type as well.
- 3 The eigenSolver implemented in LifeV **does not work** directly with `EpetraMatrix` and `EpetraPreconditioner`: use `ComposedPreconditioner` class instead.
- 4 A matrix can be pushed into a composed preconditioner, but an attempt to use its inverse would cause a Trilinos exception.
- 5 $P(A)$ preconditioner computed with the matrix A :
 - `Epetra_Preconditioner::Apply(...)` applies the matrix A used to compute $P(A)$;
 - `Epetra_Preconditioner::Apply_Inverse(...)` applies $P(A)^{-1}$

KEY FEATURES

- 1 When calling `IfpackComposedPreconditioner::push_back(P)` by default an additive Schwarz preconditioner for the factor P is computed.
- 2 Note that P in the previous slide may be of `IfpackComposedPreconditioner` type as well.
- 3 The eigenSolver implemented in LifeV **does not work** directly with `EpetraMatrix` and `EpetraPreconditioner`: use `ComposedPreconditioner` class instead.
- 4 A matrix can be pushed into a composed preconditioner, but an attempt to use its inverse would cause a Trilinos exception.
- 5 $P(A)$ preconditioner computed with the matrix A :
 - `Epetra_Preconditioner::Apply(...)` applies the matrix A used to compute $P(A)$;
 - `Epetra_Preconditioner::Apply_Inverse(...)` applies $P(A)^{-1}$

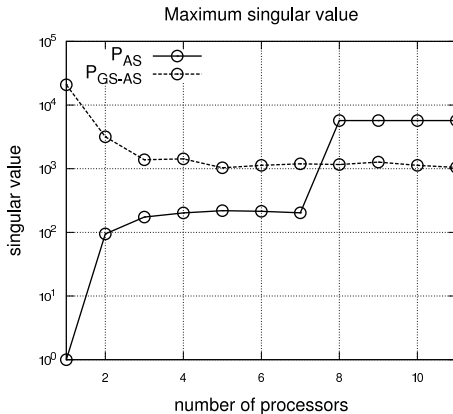
KEY FEATURES

- 1 When calling `IfpackComposedPreconditioner::push_back(P)` by default an additive Schwarz preconditioner for the factor P is computed.
- 2 Note that P in the previous slide may be of `IfpackComposedPreconditioner` type as well.
- 3 The eigenSolver implemented in LifeV **does not work** directly with `EpetraMatrix` and `EpetraPreconditioner`: use `ComposedPreconditioner` class instead.
- 4 A matrix can be pushed into a composed preconditioner, but an attempt to use its inverse would cause a Trilinos exception.
- 5 $P(A)$ preconditioner computed with the matrix A :
 - `Epetra_Preconditioner::Apply(...)` applies the matrix A used to compute $P(A)$;
 - `Epetra_Preconditioner::Apply_Inverse(...)` applies $P(A)^{-1}$

APPLICATION TO GCE

Comparison between

- Additive Schwarz preconditioner $P_{AS}(A)$
- Additive Schwarz preconditioner composed with a Gauss-Seidel preconditioner $P_{AS-GS}(A)$



THANK YOU!