Presentation of XLiFE++

Iterative solvers

Colin Chambeyron, Manh-Ha Nguyen

Unité de Mathématiques Appliquées,
ENSTA - Paristech

25 Juin 2014

- table of contens -

- table of contens -

- table of contens -

- table of contens -

- table of contens -

- Color code -

class `IterativeSolver` & `Preconditioner`

Xlife++

variables chosen by user

template class

[facultative arguments]

variational formulation:

$\Omega \subset \mathbb{R}^n$.

Find $u \in H^1(\Omega)$ such that $\forall v \in H^1(\Omega)$:

$$\begin{cases} a(u,v) = b(v) & \text{in } \Omega \\ \frac{\partial u}{\partial n} = 0 & \text{on } \partial\Omega \end{cases}$$

where

$$\begin{aligned} a(u,v) &= \int_\Omega (u \cdot v + div(u) \cdot div(v) + \epsilon(u) : \epsilon(v)) d\Omega \\ b(v) &= \int_\Omega f \cdot v d\Omega \end{aligned}$$

code Xlife++:

[$\cdots$ space and unknowns declarations $\cdots$]

- Linear and bilinear form definition:
  ```
  BilinearForm   auv=intg( omg, epsilon(u)%epsilon(v) + div(u) * div(v) + u | v);
  LinearForm     bv = intg(omg, f * v);
  ```
- Matrix and vector definition
  ```
  TermMatrix A(auv,"a(u,v)");
  TermVector b(bv,"b(v)");
  ```
- compute(A, b);

variational formulation:

$\Omega \subset \mathbb{R}^n$.
Find $u \in H^1(\Omega)$ such that $\forall v \in H^1(\Omega)$:

$$\begin{cases} a(u,v) = b(v) & \text{in } \Omega \\ \frac{\partial u}{\partial n} = 0 & \text{on } \partial\Omega \end{cases}$$

where

$$\begin{aligned} a(u,v) &= \int_\Omega (u \cdot v + div(u) \cdot div(v) + \epsilon(u) : \epsilon(v)) d\Omega \\ b(v) &= \int_\Omega f \cdot v d\Omega \end{aligned}$$

code Xlife++:

[$\cdots$ space and unknowns declarations $\cdots$]

- Linear and bilinear form definition:
  ```
  BilinearForm   auv=intg( omg, epsilon(u)%epsilon(v) + div(u) * div(v) + u | v);
  LinearForm     bv = intg(omg, f * v);
  ```
- Matrix and vector definition
  ```
  TermMatrix A(auv,"a(u,v)");
  TermVector b(bv,"b(v)");
  ```
- compute(A, b);

AIM:

Solve:

$$A \cdot \vec{X} = \vec{b} \quad , \quad \begin{array}{l} A \in K^{nn} \\ \vec{X}, \vec{b} \in K^{n} \end{array}$$

AIM:

Solve:

$$A \cdot \vec{X} = \vec{b} \quad , \quad \begin{array}{l} A \in K^{nn} \\ \vec{X}, \vec{b} \in K^{n} \end{array}$$

↓

direct solver
(class `TermMatrix`)

- luSolve
- ldlstarSolve
- ldltSolve

AIM:

Solve:

$$A \cdot \vec{X} = \vec{b} \quad , \quad \begin{array}{l} A \in K^{nn} \\ \vec{X}, \vec{b} \in K^n \end{array}$$

↓

(preconditioned) iterative solver

AIM:

Solve:

$$A \cdot \vec{X} = \vec{b} \quad , \quad \begin{array}{l} A \in K^{nn} \\ \vec{X}, \vec{b} \in K^{n} \end{array}$$

↓

(preconditioned) iterative solver

- class `iterativeSolver` –

- class Preconditioner<*Matrix*> -

templated by a matrix class

- constructors
- function solve(*Vector*)

- class Preconditioner<*Matrix*> -

! templated by a matrix class

Constructors

- *P* Preconditioning matrix.
- *solvertype*: { noPrec, lu, ldlt, ldlstar, sor, ssor, diag, myPrec
- *w*: relaxation factor [1].

function solve(*Vector* Y)

solve $P \cdot X = Y$
using *solvertype* method.

Example

Matrix M;
Preconditioner<*Matrix*> myPrec(*M*, *solvertype*);

Preconditioner myPrec is constructed.

- class Preconditioner<*Matrix*> -

! templated by a matrix class

Constructors

- P Preconditioning matrix.
- *solvertype*: { noPrec, lu, ldlt, ldlstar, sor, ssor, diag, myPrec
- w: relaxation factor [1].

function solve(*Vector* Y)

solve $P \cdot X = Y$
using *solvertype* method.

Example

Matrix M;
Preconditioner<*Matrix*> myPrec(*M*, *solvertype*);

Preconditioner myPrec is constructed.

- class Preconditioner<*Matrix*> -

! templated by a matrix class

Constructors

- *P* Preconditioning matrix.
- *solvertype*: { noPrec, lu, ldlt, ldlstar, sor, ssor, diag, myPrec
- *w*: relaxation factor [1].

function solve(*Vector* Y)

solve $P \cdot X = Y$
using *solvertype* method.

Example

Matrix M;
Preconditioner<*Matrix*> myPrec(*M*, *solvertype*);

Preconditioner myPrec is constructed.

- class Preconditioner<*Matrix*> -

templated by a matrix class

Constructors

- *P* Preconditioning matrix.
- *solvertype*: { noPrec, lu, ldlt, ldlstar, sor, ssor, diag, myPrec
- *w*: relaxation factor [1].

function solve(*Vector* Y)

solve $P \cdot X = Y$
using *solvertype* method.

Example

Matrix M;
Preconditioner<*Matrix*> myPrec(*M*, *solvertype*);

Preconditioner myPrec is constructed.

- class Preconditioner<*Matrix*> -

! templated by a matrix class

Constructors

- *P* Preconditioning matrix.
- *solvertype*: { noPrec, lu, ldlt, ldlstar, sor, ssor, diag, myPrec
- *w*: relaxation factor [1].

function solve(*Vector* Y)

solve $P \cdot X = Y$
using *solvertype* method.

Example

Matrix M;
Preconditioner<*Matrix*> myPrec(*M*, *solvertype*);

Preconditioner myPrec is constructed.

- classe iterativeSolver -

Iterative solver ≡ OBJECT

- **templated** by class *Matrix*, *Vector*
  - self-employed class.
  - independant matrices and vectors choice.
- 3 types of solvers:
  - relaxation methods: SOR, SSOR
  - gradient's methods: CG , CGS, BiCG, BiCGstab
  - Krylov 's methods: GMres, QMR.
- Solvers are preconditioned.
  - class Preconditioner<*Matrix*>

- classe iterativeSolver -

Iterative solver ≡ OBJECT

- **templated** by class *Matrix*, *Vector*

  - self-employed class.
  - independant matrices and vectors choice.

- 3 types of solvers:
  - relaxation methods: SOR, SSOR
  - gradient's methods: CG , CGS, BiCG, BiCGstab
  - Krylov 's methods: GMres, QMR.
- Solvers are preconditioned.

  class Preconditioner<*Matrix*>

- classe iterativeSolver -

Iterative solver ≡ OBJECT

- **templated** by class *Matrix*, *Vector*
  - self-employed class.
  - independant matrices and vectors choice.
- 3 types of solvers:
  - relaxation methods: SOR, SSOR
  - gradient's methods: CG , CGS, BiCG, BiCGstab
  - Krylov 's methods: GMres, QMR.
- Solvers are preconditioned.
  - class Preconditioner<*Matrix*>

- classe iterativeSolver -

Iterative solver $\equiv$ OBJECT

- self-employed class.
- independant matrices and vectors choice.

- **templated** by class *Matrix*, *Vector*

- relaxation methods: SOR, SSOR
- gradient's methods: CG , CGS, BiCG, BiCGstab
- Krylov 's methods: GMres, QMR.

- 3 types of solvers:

class Preconditioner<*Matrix*>

- Solvers are preconditioned.

- classe iterativeSolver -

iterativeSolver

↪

- cgSolver
- cgsSolver
- bicgSolver
- bicgstabSolver
- qmrSolver
- gmresSolver
- sorSolver
- ssorSolver

- class IterativeSolver -

! templated by a matrix class

- constructors
- overloaded operator (.) to solve linear system

- class *XXX*Solver -

Constructors

- X_0 initial data $[\vec{0}]$
- ϵ: tolerance $[10^{-4}]$
- N: Maximum number of iterations [1000]
- $\mathcal{N}$: dimension of Krylov's space.
- w: relaxation factor [1]

Examples

- XXXSolver mysolver(X_0, ϵ, N);
- XXXSolver mysolver(X_0, ϵ, N, $\mathcal{N}$, w); for GMres and QMR solvers.
- XXXSolver mysolver;

Solver object XXXSolver is constructed.

- class *XXX*Solver -

Constructors

- X_0 initial data $[\vec{0}]$
- ϵ: tolerance $[10^{-4}]$
- N: Maximum number of iterations [1000]
- $\mathcal{N}$: dimension of Krylov's space.
- w: relaxation factor [1]

Examples

- XXXSolver mysolver(X_0, ϵ, N);
- XXXSolver mysolver(X_0, ϵ, N, $\mathcal{N}$, w); for GMres and QMR solvers.
- XXXSolver mysolver;

Solver object XXXSolver is constructed.

- class *XXX*Solver -

Constructors

- X_0 initial data $[\vec{0}]$
- ϵ: tolerance $[10^{-4}]$
- N: Maximum number of iterations [1000]
- $\mathcal{N}$: dimension of Krylov's space.
- w: relaxation factor [1]

Examples

- XXXSolver mysolver(X_0, ϵ, N);
- XXXSolver mysolver(X_0, ϵ, N, $\mathcal{N}$, w); for GMres and QMR solvers.
- XXXSolver mysolver;

Solver object XXXSolver is constructed.

- class *XXX*Solver -

Constructors

- X_0 initial data $[\vec{0}]$
- ϵ: tolerance $[10^{-4}]$
- N: Maximum number of iterations [1000]
- $\mathcal{N}$: dimension of Krylov's space.
- w: relaxation factor [1]

Examples

- XXXSolver mysolver(X_0, ϵ, N);
- XXXSolver mysolver(X_0, ϵ, N, $\mathcal{N}$, w); for GMres and QMR solvers.
- XXXSolver mysolver;

Solver object XXXSolver is constructed.

- classe *XXX*Solver -

overloaded operator () : return $\vec{X}$

- A matrix of the problem.
- $\vec{b}$: Right hand side.
- P: preconditioner.
- X_0: initial value.

Example

$$X = mysolver(A, b, [P, X_0])$$

CV $\Longrightarrow$ return X / No CV $\Longrightarrow$ STOP

- classe *XXX*Solver -

overloaded operator () : return $\vec{X}$

- A matrix of the problem.
- $\vec{b}$: Right hand side.
- P: preconditioner.
- X_0: initial value.

Example

$$X = mysolver(A, b, [P, X_0])$$

CV $\Longrightarrow$ return X / No CV $\Longrightarrow$ STOP

- classe *XXX*Solver -

overloaded operator () : return $\vec{X}$

- A matrix of the problem.
- $\vec{b}$: Right hand side.
- P: preconditioner.
- X_0: initial value.

Example

$$X = mysolver(A, b, [P, X_0])$$

CV $\Longrightarrow$ return X / No CV $\Longrightarrow$ STOP

- classe *XXX*Solver -

overloaded operator () : return $\vec{X}$

- A matrix of the problem.
- $\vec{b}$: Right hand side.
- P: preconditioner.
- X_0: initial value.

Example

$$X = mysolver(A, b, [P, X_0])$$

CV $\Longrightarrow$ return X / No CV $\Longrightarrow$ STOP

- Example of use -

Intialisation

•**Initialisation-**

Matrix A; *Vector* b ,X0;

Real epsilon=10^{-6}, Number nbIteration=100, krylovDim=10;

- Example of use -

Intialisation & construction of the solver

•**Initialisation-**

Matrix A; *Vector* b ,X0;
Real epsilon=10^{-6}, Number nbIteration=100, krylovDim=10;

•**Construction of the solver -**

GmresSolver MyGmres (krylovDim, epsilon, nbIteration, 0);

- Example of use -

Intialisation & construction of the solver

•Initialisation-
Matrix A; *Vector* b

•Construction of the solver -
GmresSolver MyGmres ;

Resolution

Vector X = Mygmres(A, b);

- Example of use -

Intialisation & construction of the solver & the preconditioner

•**Initialisation-**
Matrix A; *Vector* b

•**Construction of the preconditioner -**
Preconditioner<*Matrix*> P(A, _diagPrec);

•**Construction of the solver -**
GmresSolver MyGmres ;

Resolution

Vector X = Mygmres(A, b, P) ;

- Interface `Solver` / xLife++ -

- *Matrix* ⟶ `class TermMatrix`
- *Vector* ⟶ `class TermVector`
- *Preconditionner<Matrix>* ⟶ `class PreconditionnerTerm`

- Interface Solver / xLife++ -

In class TermMatrix: Creation of functions:

cgSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])
cgsSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])
bicgSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])
bicgstabSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])
qmrSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])
sorSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])
ssorSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])
gmresSolver	(TermMatrix A, TermVector B, [PreconditionnerTerm P, TermVector $X_0, \cdots$])

return TermVector X (if CV)

AIM:

with iterative methods, solve:

$$A \cdot \vec{X} = \vec{b} \quad , \quad \begin{array}{l} A \in K^{nn} \\ \vec{X}, \vec{b} \in K^{n} \end{array}$$

- Interface `Solver` / xLife++ -

Without a preconditioner

with class solver

```
GmresSolver MyGmres;
TermVector X = Mygmres(A, b);
```

with class TermMatrix

```
TermVector X = gmresSolver(A, b);
```

- Interface `Solver` / xLife++ -

Without a preconditioner

with class solver

```
GmresSolver MyGmres;
TermVector X = Mygmres(A, b);
```

with class TermMatrix

```
TermVector X = gmresSolver(A, b);
```

- Interface Solver / xLife++ -

With a preconditioner

with class solver

```
Preconditioner<TermMatrix> preMat(A, _diagPrec);
GmresSolver MyGmres;
TermVector X = Mygmres(A, b, preMat);
```

with class TermMatrix

```
PreconditionerTerm preMat(A, _diagPrec);
TermVector X = gmresSolver(A, b, preMat);
```

- Interface Solver / xLife++ -

With a preconditioner

with class solver

```
Preconditioner<TermMatrix> preMat(A, _diagPrec);
GmresSolver MyGmres;
TermVector X = Mygmres(A, b, preMat);
```

with class TermMatrix

```
PreconditionerTerm preMat(A, _diagPrec);
TermVector X = gmresSolver(A, b, preMat);
```

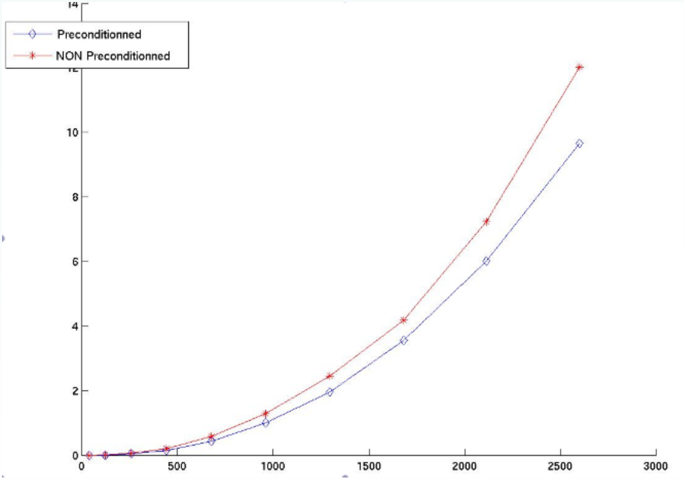
ASSESSMENT

- class `IterativeSolver`
  $\longrightarrow$ access to members attributes

- class `TermMarix`
  $\longrightarrow$ easier & more concise

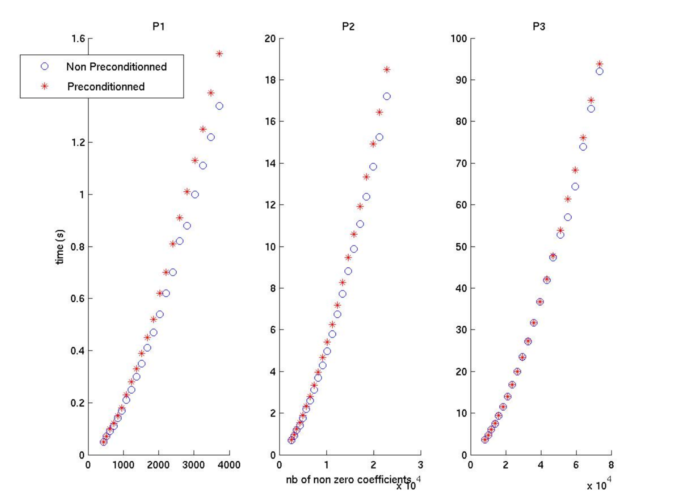
- Results of convergence -

- GMRES: P2 approximation -

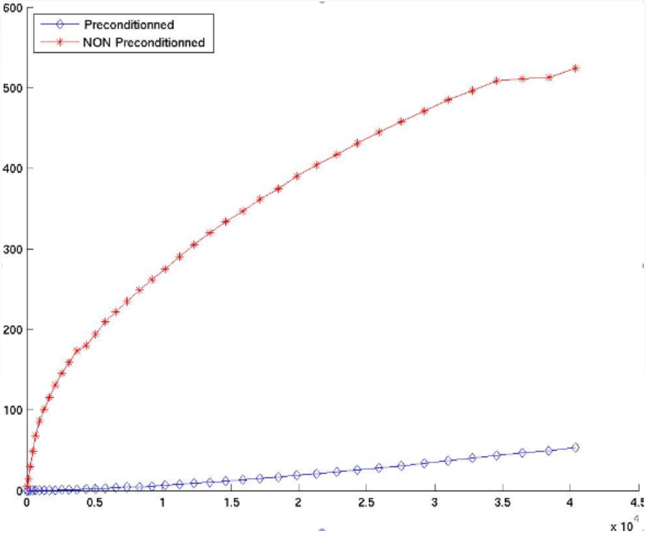


nb of non-zero coefficients

- CG -


P1
P2
P3
Non Preconditionned
Preconditionned
time (s)
nb of non zero coefficients
x 10^4
x 10^4

- CG: P2 -


Preconditionned
NON Preconditionned
600
500
400
300
200
100
0
0
0.5
1
1.5
2
2.5
3
3.5
4
x 10^4

- Perspectives -

- Improve preconditioned solver algorithms
- correct last bugs
- ?????????

Thanks to:

- Relaxation method -

SOR principe

Regular decomposition de A: $A = L_A + D_A + U_A$

+

Introduction of the relaxation factor $w \in][0;2[$ such as:

$$X_{n+1} = M^{-1} \cdot N \cdot X_n + M^{-1} \cdot b$$

where: $\begin{cases} M = & \frac{1}{w} \cdot D_A + L_A \\ N = & (\frac{1}{w} - 1) \cdot D_A - U_A \end{cases}$

- Relaxation method -

Principe SSOR

Regular decomposition de A: $A = L_A + D_A + U_A$

+

Introduction of the relaxation factor $w \in][0; 2[$ such as:

$$X_{n+1} = G \cdot X_n + H \cdot b$$

where: $$\begin{cases} G = & (\frac{1}{w} \cdot D_A + U_A)^{-1} \cdot (\frac{w-1}{w} \cdot D_A + L_A) \cdot (\frac{w-1}{w} \cdot D_A + U_A) \\ H = & \frac{2-w}{w} \cdot D_A \cdot (\frac{1}{w} \cdot D_A + L_A)^{-1} \end{cases}$$

- Relaxation method -

- Relaxation methods needs a lot of matrix-vector products.
- Relaxation methods needs a lot of matrix inversions.
- It doesn't exist an optimal value foe w.

- Gradient method -

gradient method principe

$$\text{Minimize } \phi : \begin{array}{ccc} \mathbb{R}^n & \longrightarrow & \mathbb{R} \\ X & \longrightarrow & \frac{1}{2}\cdot(A\cdot X; X) - (B; X) \end{array} \text{, car } \nabla\phi = A\cdot X - B$$

Similar methods exist using the acceleration of the diminution of the residue r_i at the iteration i:

BiCG	:	$r_i \approx P_i(A)\cdot r_0$
CGS	:	$r_i \approx P_i^2(A)\cdot r_0$
BiCGstab	:	$r_i \approx G_i \cdot P_i \cdot r_0$

where G_i et P_i are polynoms of degree i.

- Gradient methods don't need lots of operations
- Convergence of CG and CGS are guaranteed if A is SPD.

- Gradient method -

gradient method principe

Minimize $\phi : \mathbb{R}^n \longrightarrow \mathbb{R}$, car $\nabla\phi = A \cdot X - B$

$X \longrightarrow \frac{1}{2} \cdot (A \cdot X; X) - (B; X)$

Similar methods exist using the acceleration of the diminution of the residue r_i at the iteration i:

BiCG : $r_i \approx P_i(A) \cdot r_0$

CGS : $r_i \approx P_i^2(A) \cdot r_0$ where G_i et P_i are polynoms of degree i.

BiCGstab : $r_i \approx G_i \cdot P_i \cdot r_0$

!

- Gradient methods don't need lots of operations
- Convergence of CG and CGS are guaranteed if A is SPD.

- GMRES/QMR methods -

Use Krylov space $\kappa_m(A, r_0)$.

GMRES principe

Boucle on iterations k {

- Boucle sur m
  - Construction of V_m: orthogonal basis of $\kappa_m(A, r_0)$ (Processus d'Arnoldi ans Given's rotation)
  - calcul of $x_m = x_0 + V_m \cdot y$
- restart with $x_0 = x_m$ if no CV

}

where: V_m : orthonormal basis of κ_m

y : minimise $||b - A \cdot x_m||$ in κ_m

- GMRES/QMR methods -

Use Krylov space $\kappa_m(A, r_0)$.

QMR is the same idea with V_m a biorthonormal basis.
Risks of "breakdown".

- GMRES/QMR methods -

Use Krylov space $\kappa_m(A, r_0)$.

- It doesn't exist an optimal value for m.
- GMRES needs a lot of memory.
- **Convergence of GMRES is guaranteed.**