V. Using Trilinos in application codes - Part II

Scope of this tutorial

- Use linear solvers/preconditioners from TRILINOS to solve systems of linear equations.

Prerequesites:

- Application code with parallel distributed data based on TPETRA
- Linear system $Ax = b$ with matrix A and righ-hand side vector b already assembled

Linear solvers in Trilinos

- Linear solvers available for both EPETRA and TPETRA stack.
- Concrete choice of packages depends on linear algebra stack.

Direct solvers

- Packages: AMESOS, AMESOS2[a]
- Solver implementation / interfaces:
  - KLU (implemented in TRILINOS)
  - UMFPACK
  - SUPERLU-DIST
  - PARDISO
  - MUMPS

(Except for KLU, TRILINOS has to be configured with the respective TPLs)

[a] Bavier, E. *et al.* Amesos2 and Belos: Direct and Iterative Solvers for Large Sparse Linear Systems. *Scientific Programming* **20,** 241–255. http://dx.doi.org/10.3233/SPR-2012-0352 (2012).

Iterative solvers

- Packages: AZTECOO[a], BELOS[b]
- Methods (also some block variants):
  - Conjugate Gradient (CG)
  - BiCGStab
  - GMRES / Flexible GMRES
  - MINRES
  - LSQR / TFQMR
  - ...

[a] Heroux, M. A. *AztecOO User Guide.* Tech. rep. SAND2004-3796 (Sandia National Laboratories, Albuquerque, NM (USA) 87185, 2007).
[b] Bavier, E. *et al.* Amesos2 and Belos: Direct and Iterative Solvers for Large Sparse Linear Systems. *Scientific Programming* **20,** 241–255. http://dx.doi.org/10.3233/SPR-2012-0352 (2012).

Preconditioners in Trilinos

- Preconditioners available for both EPETRA and TPETRA stack.
- Concrete choice of packages depends on linear algebra stack.

One-level methods

- Packages: IFPACK[a], IFPACK2[b]
- Solver implementations:
  - Incomplete LU
  - Relaxation methods (Jacobi, Gauss-Seidel, ...)
  - Polynomial (Chebyshev, ...)
  - ...

[a] Sala, M. G. & Heroux, M. A. *Robust Algebraic Preconditioners using IFPACK 3.0.* Tech. rep. SAND2005-0662 (Sandia National Laboratories, Albuquerque, NM

Multigrid methods

- Packages: ML[a], MUELU[b]
- Methods:
  - PA-AMG
  - SA-AMG
  - Emin
  - Structured AMG
  - ...

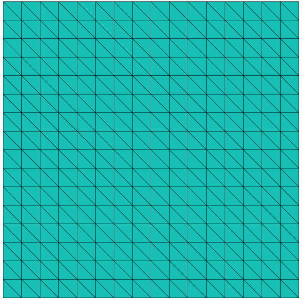
[a] Gee, M. W. *et al. ML 5.0 Smoothed Aggregation User's Guide.* Tech. rep. SAND2006-2649 (Sandia National Laboratories, Albuquerque, NM (USA) 87185, 2006).
[b] Berger-Vergiat, L. *et al. MueLu User's Guide.* Tech. rep. SAND2019-0537 (Sandia National Laboratories, Albuquerque, NM

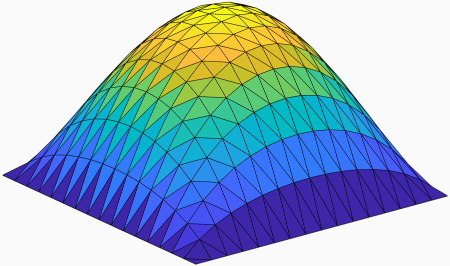
Multilevel domain decomposition methods

- Packages: SHYLU
- Methods:
  - BDDC
  - Overlapping Schwarz, GDSW (FROSCH[a])
  - ...

[a] Heinlein, A. *et al. FROSch: A Fast And Robust Overlapping Schwarz Domain Decomposition Preconditioner Based on Xpetra in Trilinos.* in *Domain Decomposition Methods in Science and Engineering XXV* (eds Haynes, R. *et al.*) (Springer International Publishing, Cham, 2020), 176–184.

11 Laplacian Model Problem



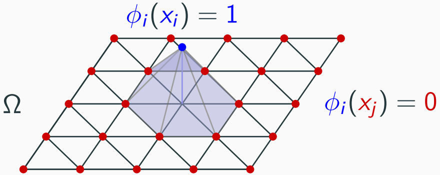


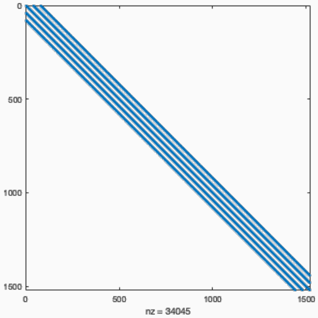
Let us consider the simple **diffusion model problem** ($\alpha(x) = 1$):

$$-\Delta u = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega.$$

Discretization using finite elements yields the linear equation system

$$Ku = f.$$





- Due to the local support of the finite element basis functions, the resulting system is **sparse**.
- However, due to the **superlinear complexity and memory cost**, the use of direct solvers becomes infeasible for fine meshes, that is, for the **resulting large sparse equation systems**.

→ We will employ iterative solvers:
For our elliptic model problem, the system matrix is symmetric positive definite. Hence, we can use the **conjugate gradient (CG) method**.

Theorem 1

Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite. Then the **CG method** *converges and the following error estimate holds:*

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

where $\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$.

Do we need a preconditioner?

Theorem 2 (Condition number of the stiffness matrix)

There exists a constant $c > 0$, independent of h, such that

$$\kappa(K) \leq c \frac{h^d}{\left(\min_{T \in \tau_h} h_T\right)^{d+2}}.$$

$\Rightarrow$ **Convergence of the PCG method will deteriorate** when refining the mesh.

The **preconditioned conjugate gradient (PCG)** methods solves instead the preconditioned system

$$M^{-1}Ax = M^{-1}b \qquad \text{or more precisely} \qquad M^{-1/2}AM^{-1/2}x = M^{-1/2}b,$$

with the preconditioner $M^{-1} \approx A^{-1}$. This system is equivalent to the original system

$$Ax = b.$$

but **easier to solve**.

Theorem 3

Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite. Then the **PCG method** *converges and the following error estimate holds:*

$$\left\| e^{(k)} \right\|_A \leq 2 \left(\frac{\sqrt{\kappa\left(M^{-1}A\right)} - 1}{\sqrt{\kappa\left(M^{-1}A\right)} + 1} \right)^k \left\| e^{(0)} \right\|_A,$$

where $\kappa\left(M^{-1}A\right) = \frac{\lambda_{\max}\left(M^{-1/2}AM^{-1/2}\right)}{\lambda_{\min}\left(M^{-1/2}AM^{-1/2}\right)}$.

Preconditioned Conjugate Gradient (PCG) Method

Algorithm 1: Preconditioned conjugate gradient method

Result: Approximate solution of the linear equation system $Ax = b$

Given: Initial guess $x^{(0)} \in \mathbb{R}^n$ and tolerance $\varepsilon > 0$

$r^{(0)} := b - Ax^{(0)}$

$p^{(0)} := y^{(0)} := M^{-1} r^{(0)}$

while $\left\| r^{(k)} \right\| \geq \varepsilon \left\| r^{(0)} \right\|$ **do**

$\alpha_k := \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(Ap^{(k)}, p^{(k)}\right)}$

$x^{(k+1)} := x^{(k)} + \alpha_k y^{(k)}$

$r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$

$y^{(k+1)} := M^{-1} r^{(k+1)}$

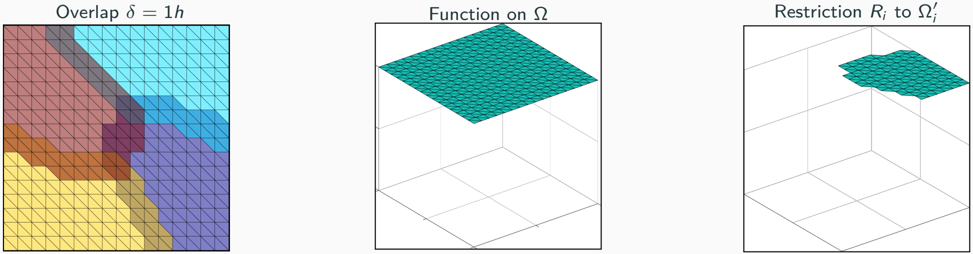
$\beta_k := \frac{\left(y^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

$p^{(k+1)} := r^{(k+1)} - \beta_k p^{(k)}$

end

Let us use a **one-level Schwarz preconditioner**, which can be **constructed algebraically from the system matrix** $A \rightarrow$ Ifpack (for Epetra), Ifpack2 (for Tpetra).

One-Level Schwarz Preconditioner



Based on an **overlapping domain decomposition**, we define an additive **one-level Schwarz preconditioner**

$$M_{\text{OS-1}}^{-1} = \sum\nolimits_{i=1}^{N} R_i^T K_i^{-1} R_i,$$

where R_i and R_i^T are restriction and prolongation operators corresponding to Ω_i', and $K_i := R_i K R_i^T$. The K_i correspond to **local Dirichlet problems** on the overlapping subdomains.

Condition number bound:

$$\kappa \left(M_{\text{OS-1}}^{-1} K \right) \leq C \left(1 + \frac{1}{H\delta} \right)$$

where the constant C is **independent of the subdomain size** H **and the width of the overlap** δ.

Iterative solvers from the Belos package

- Use Belos::SolverFactory<SC, MV, OP> to create any BELOS solver
  - SC = Scalar type
  - MV = MultiVector type
  - OP = Operator type
- Initiate solver creation via the create() method
  - Select solver via its name passes as std::string
  - Pass solver parameters / configuration via a Teuchos::ParameterList

Example:

```
RCP<Teuchos::ParameterList> params = rcp (new ParameterList());
params->set("Maximum Iterations", 150);
params->set("Convergence Tolerance", 1.0e-6);

Belos::SolverFactory<SC, MV, OP> belosFactory;
RCP<Belos::SolverManager<SC, MV, OP>> solver = belosFactory.create ("GMRES", params);
```

- Pack matrix, left- and right-hand side into a Belos::LinearProblem<SC, MV, OP>
- If desired and available, include the ready-to-use preconditioner
- Pass the linear problem to the solver

```
RCP<Belos::LinearProblem<SC, MV, OP>> problem
    = rcp(new Belos::LinearProblem<SC, MV, OP> (A, x, b));
problem->setProblem();

if (usePreconditioner)
  problem->setRightPrec(preconditioner);

solver->setProblem(problem);
```

- Solve the linear system
- Return value indicates the convergence status

Example:

```
Belos::ReturnType solveResult = solver->solve();
```

Preconditioners from the Ifpack2 package

- Use Ifpack2 :: Factory :: create<Tpetra::RowMatrix<SC,LO,GO,NO>> to create any IFPACK2 method
  - SC = Scalar type
  - LO = LocalOrdinal type
  - GO = GlobalOrdinal type
  - NO = KOKKOS node type
  - Select method via its name passes as std :: string
  - Pass the matrix A

Example:

```
RCP<Ifpack2::Preconditioner<SC,LO,GO,NO>> prec
    = Ifpack2::Factory::create<Tpetra::RowMatrix<SC,LO,GO,NO>> ("RELAXATION", A);
```

- Configure via a Teuchos::ParameterList
- Initialize and compute the preconditioner

Example:

```
Teuchos::ParameterList precParams;
precParams.set("relaxation: type", relaxationType);
precParams.set("relaxation: sweeps", numSweeps);
precParams.set("relaxation: damping factor", damping);
prec->setParameters(precParams);

prec->initialize();
prec->compute();
```

Disclaimer

Today's remarks on Stratimikos are intended as an outlook for interested users. This package will not be covered in today's tutorial.

What is Stratimikos?

- unified set of Thyra-based wrappers to linear solver and preconditioner capabilities in Trilinos
- enables solver customization through an xml-input deck

Exemplary input deck for Stratimikos:

```
<ParameterList>
  <Parameter name="Linear Solver Type" type="string" value="Belos"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="Belos">
      <Parameter name="Solver Type" type="string" value="Block GMRES"/>
      <ParameterList name="Solver Types">
        <ParameterList name="Block GMRES">
          <Parameter name="Block Size" type="int" value="1"/>
          <Parameter name="Convergence Tolerance" type="double" value="1e-13"/>
          <Parameter name="Num Blocks" type="int" value="300"/>
          <Parameter name="Output Frequency" type="int" value="1"/>
          <Parameter name="Maximum Iterations" type="int" value="400"/>
        </ParameterList>
      </ParameterList>
    </ParameterList>
  </ParameterList>
  <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
  <ParameterList name="Preconditioner Types">
    <ParameterList name="Ifpack">
      <Parameter name="Prec Type" type="string" value="ILU"/>
      <Parameter name="Overlap" type="int" value="1"/>
      <ParameterList name="Ifpack Settings">
        <Parameter name="fact: level-of-fill" type="int" value="2"/>
      </ParameterList>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Hands-on exercises

Solve linear systems with a (preconditioned) Krylov solver:

- Complete the app ex_03_solve to solve various linear systems with
    - plain GMRES (without preconditioning)
    - preconditioned GMRES
- Material: exercises/ex_03_solve

- Trilinos **GitHub repository:** https://github.com/Trilinos
- Trilinos **website:** https://trilinos.github.io/index.html
  - **Documentation:** https://trilinos.github.io/documentation.html
  - Each package has its own **Doxygen documentation:** For instance, Tpetra: https://docs.trilinos.org/dev/packages/tpetra/doc/html/index.html
  - **Getting started:** https://trilinos.github.io/getting_started.html
- Trilinos **hands-on tutorials:** https://github.com/Trilinos_tutorial/wiki/TrilinosHandsOnTutorial
- Kokkos ressources on GitHub: https://github.com/kokkos

Thank you for your attention!

Questions?