deal.II ifpack Interface for BoomerAMG use as a Solver or Preconditioner

Math 676 Project Report
May 2019

Joshua Hanophy

Contents

1 Introduction

The *hypre* library includes a variety of high performance preconditioners and solvers that are able to function in massively parallel applications [1]. In practice, this means that all algorithms and data structures have been designed to exhibit reasonable parallel scaling in complexity and memory requirements. One important consequence of this from the perspective of a finite element program using *hypre* is that data structures like the problem mesh, solution and forcing vector, and the global matrix that grow as the problem grows must be distributed among the processors being used to solve the problem in parallel. Distributed means that each processor stores only a portion of the total data structure. Obviously, no processor can know about the entire mesh or global matrix in a massively parallel simulation, because each processor has access to only a finite amount of memory to store data and a problem can be made arbitrarily large such that the memory requirements exceed storage capacity.

The deal.II finite element library includes the ability to setup and solve problems using massively parallel systems [2]. For distributed data structures and distributed linear algebra, deal.II relies on separate libraries. For domain decomposition and storage of a fully distributed mesh, deal.II relies on the p4est library [3]. For fully distributed matrices and vectors, deal.II can use petsc or Trilinos. The Trilinos library was used in the program constructed as part of this project [4]. Trilinos can interface with *hypre* in several ways. For this project, the ifpack package was used [5].

Algebraic multigrid (AMG) methods have been widely used to solve systems arising from the discretization of elliptic partial differential equations. In serial, AMG algorithms scale linearly with problem size. In parallel, communication costs scale logarithmically with the number of processors [6]. Recently, a classical AMG method based on approximate ideal restriction (AIR) was developed for nonsymmetric matrices. AIR has already been shown to be effective for solving the linear systems arising from upwind discontinuous Galerkin (DG) finite element discretization of advection-diffusion problems, including the hyperbolic limit of pure advection [7][8]. A new parallel version of AIR, pAIR, has been implemented in the *hypre* library.

Currently, there is no interface in deal.II to *hypre* through Trilinos. The goal of this project was to created an interface to *hypre* using ifpack. The interface will:

- Allow for access to BoomerAMG as either a solver or a preconditioner for one of the other solvers in *hypre*
- Include useful defaults for both Classical AMG and the new AIR AMG
- Inculde the functionality to easily adjust or add AMG parameters

The interface created for this project is fully documented with Doxygen. The Doxygen generated documentation is included as Appendix. The overal design is briefly described in this introduction as well. There are a significant number of parameters that can be used to control the operation of BoomerAMG. These parameters can have several known types. These types include double, int, or int *, as well as several others. Boost::variant was used to store items related to the parameters. This is a container that can hold different types. The types the container can hold are declared statically and so if an incorrect type is placed in the container, this error can be caught at compile time. All parameters are stored in a map with a string key. Helper functions to change parameters values, add new parameters, and delete parameters were all added.

Three different programs were created to demonstrate the interface. These three programs are described in more detail in subsequent sections.

2 SUPG Test Program

The advection diffusion problem shown in Equation 1 is of interest in this section. When this equation is discretized using the Galerkin method with continuous one dimensional elements, the results discretized system can be shown to be analogous to a discretization using central difference [9]. The truncation error associated with central differencing the advection term subtracts from the diffusion coefficient. When the velocity is sufficiently high enough, the actual diffusion coefficient can become negative. This problem with central differencing leads to instability.

Stabilization techniques add diffusion in various ways to cancel out the contribution from the central difference truncation error. In this section a program that implements an SUPG discritization is described. The SUPG discretization implemented in described in [9]. First, the weak formulation of Equation 1 is shown in Equation 2 $w = 0$ on Γ_D where Γ_D is the portion of the boundary with Dirichlet conditions.

$$\vec{a} \cdot \nabla u - \nabla \cdot (\nu \nabla u) = s \tag{1}$$

$$\begin{aligned}\int_\Omega w\left(\vec{a} \cdot \nabla u\right) d\Omega + &\int_\Omega \nabla w \cdot \nu \nabla u d\Omega \\ &= \int_\Omega w s d\Omega \quad \forall w \in V\end{aligned} \tag{2}$$

The SUPG discretization is a consistent stabilization technique. First, the residual is defined as shown in Equation 3. Then the stabilization is added as shown in Equation 4 where $\mathcal{P}(w)$ and τ are defined below. h_η and h_ξ are characteristic lengths whose values are defined in [9].

$$\mathcal{R}(u) = \vec{a} \cdot \nabla u - \nabla \cdot (\nu \nabla u) - s \tag{3}$$

$$a(w,u) + c(\vec{a}; w, v) + \sum_e \int_{\Omega^e} \mathcal{P}(w) \tau \mathcal{R}(u) = (w, s) \tag{4}$$

$$\mathcal{P}(w) = \vec{a} \cdot \nabla w \tag{5}$$

$$\tau = \frac{\bar{\nu}}{||a||^2} \tag{6}$$

$$\bar{\nu} = \frac{1}{2}\left(\bar{\xi} a_\xi h_\xi + \bar{\eta} a_\eta h_\eta\right) \tag{7}$$

Shown below are definitions related to ξ, analogous definitions exist for η. ξ and η are coordinates in the natural coordinate system

$$\begin{aligned}&\bar{\xi} = \coth Pe_\xi - \frac{1}{Pe_\xi} \\ &Pe_\xi = a_\xi h_\xi / (2\nu) \\ &a_\xi = \vec{e}_\xi \cdot \vec{a}\end{aligned} \tag{8}$$

The SUPG test program constructed includes the option so solve a problem with or without stabilization. Additionally, three different solvers can be used. These are the AIR AMG solver, the Classical AMG solver, or the direct Trilinos solver already implemented in deal.II. The program currently only works in two-dimension and with linear elements. The stabilization would have to be adjusted to work in three dimensions or with higher order elements. Also, in the current program, AMR has been disabled and uniform mesh refinement is currently being used instead for simplicity.

2.1 Results

Figure 1 and Figure 2 both show results with natural boundary conditions which here mean natural for an advection diffusion problem. The inflow boundary is set with a homogeneous Dirichlet condition and the outflow boundary condition is set with a homogeneous Neumann boundary condition. In this case, there is negligible difference between the stabilized result and the non stabilized result.

Figure 3 and Figure 4 show simulation results with homogeneous Dirichlet boundary conditions. In this case, the unstabilized results oscillates severely towards the outflow face of the problem. The stabilized version is not completely smooth, but is much smoother by comparison.

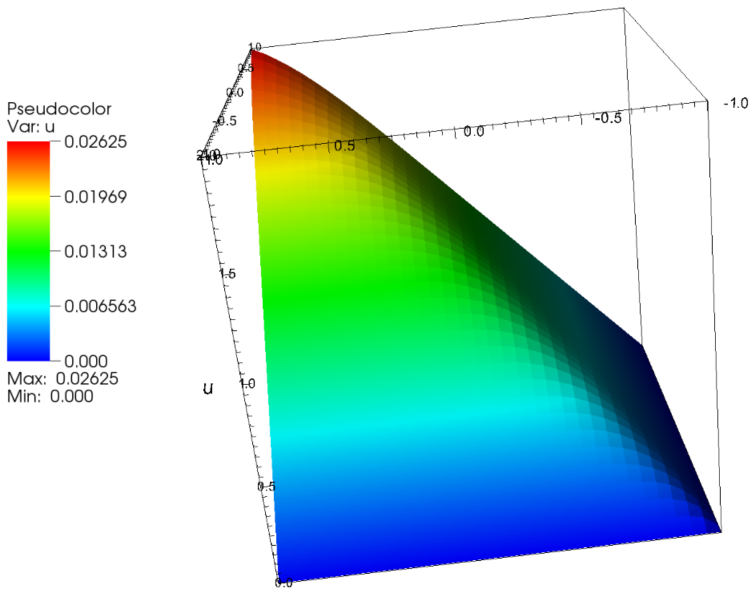


Figure 1: Natural boundary conditions without stabilization.

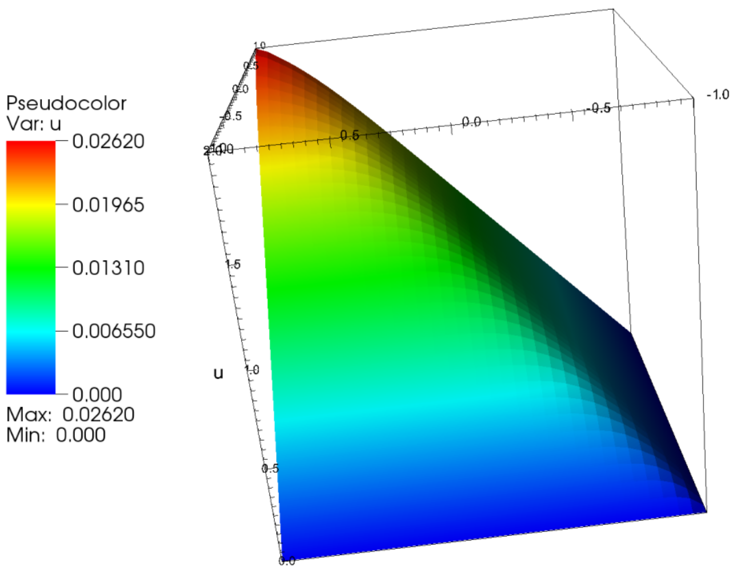


Figure 2: Natural boundary conditions with stabilization.

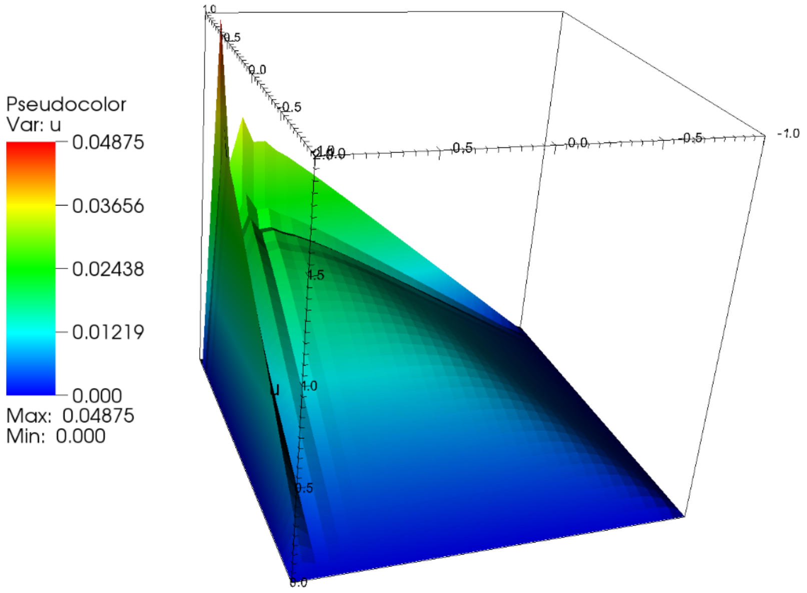


Figure 3: Homogeneous Dirichlet boundary conditions without stabilization

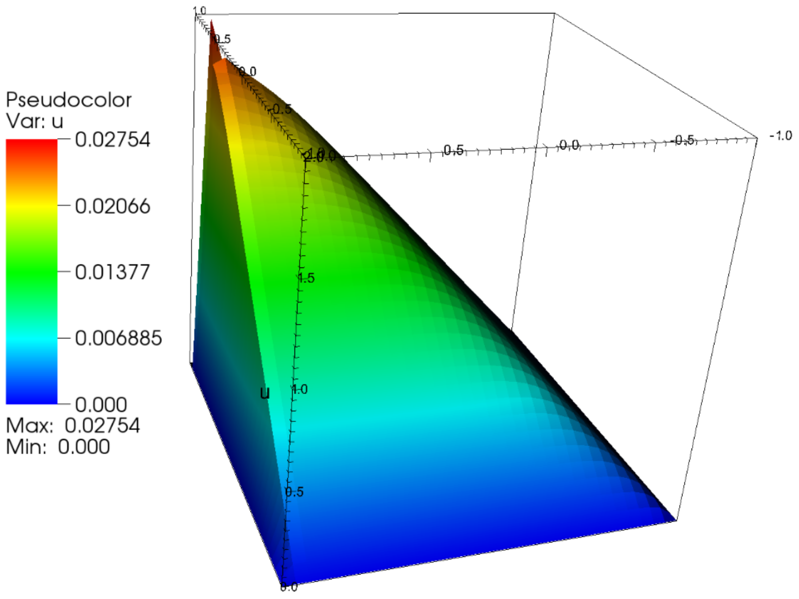


Figure 4: Homogeneous Dirichlet boundary conditions with stabilization

2.2 Solving with AMG

When the advection diffusion equation is discritized, the resulting system is not symmetric. Therefore, such a problem may be difficult for classical AMG to solve while being a good candidate for solution with AIR AMG. Table **??** shows the results for solving the system with AIR AMG and Classical AMG. Surprisingly, classical AMG works well for this problem. More investigation into this is required.

Table 1: Result from stabilized SUPR problem with homogeneous Dirichlet boundary conditions and 265,240 cells. The problem was run in parallel on four processors.

Solver	Average Convergence Factor	Total Time
Classic Interpolation based AMG	0.10	3.44 s
AIR AMG	0.31	5.96 s

3 Diffusion Test Program

The interface created for this project includes the ability to use BoomerAMG as a preconditioner to another solver in *hypre* or as a solver. To demonstrate the interfaces and test functionality, a program was created that solves diffusion. The program was partially based off of the tutorial step 40 program. Several different solvers were tested. These were the Conjugate Gradient (CG), Incomplete Cholesky Decomposition preconditioned CG (IC PCG), BoomerAMG preconditioned CG, ML Preconditioned CG, and BoomerAMG as a solver. This was not a rigorous evaluation, but provides some basic insights into performance. The converged tolerance of 1e-10 was used to generate all results. Additionally, only two dimensional results are presented here.

Table 2 and Table 3 show results for solving diffusion with a constant diffusion coefficient. The AMG preconditioners as well as the AMG solvers all perform well. IC preconditioned CG and CG are performing poorly as for the larger problem. Table 4 shows results for a problem where the diffusion coefficient varies discontinuously. The jump is from 100.0 to 0.001 and such a configuration will generally lead to a poorly conditioned matrix. CG does not converge for this problem within 3,000 iterations. The AMG preconditioned solvers as well as the AMG solver work effectively for this problem.

Table 2: Spatially constant diffusion coefficient for a problem with 1228 cells. The problem was run in parallel on four processors.

	Linear Elements		Quadratic Elements	
Solver	**Iterations**	**Time (s)**	**Iterations**	**Time (s)**
CG	84	0.00366	223	0.0337
IC PCG	69	0.00353	88	0.0229
BoomerAMG PCG	8	0.00987	8	0.0236
BoomerAMG as Solver	14	0.00870	16	0.0280
ML_Epetra PCG	11	0.00798	23	0.0412

Table 3: Spatially constant diffusion coefficient for a problem with 67093 cells. The problem was run in parallel on four processors.

	Linear Elements		Quadratic Elements	
Solver	**Iterations**	**Time (s)**	**Iterations**	**Time (s)**
CG	630	1.57	1735	28.2
IC PCG	411	1.98	492	13.4
BoomerAMG PCG	9	0.346	9	1.83
BoomerAMG as Solver	19	0.477	21	2.71
ML_Epetra PCG	12	0.286	25	2.34

Table 4: Spatially varying discontinuous diffusion coefficient as shown in Figure 5 for a problem with 67093 cells. The problem was run in parallel on four processors.

	Linear Elements		Quadratic Elements	
Solver	**Iterations**	**Time (s)**	**Iterations**	**Time (s)**
IC PCG	589	2.49	731	18.3
BoomerAMG PCG	8	0.347	9	1.91
BoomerAMG as Solver	17	0.456	20	2.62
ML_Epetra PCG	15	0.308	36	3.02

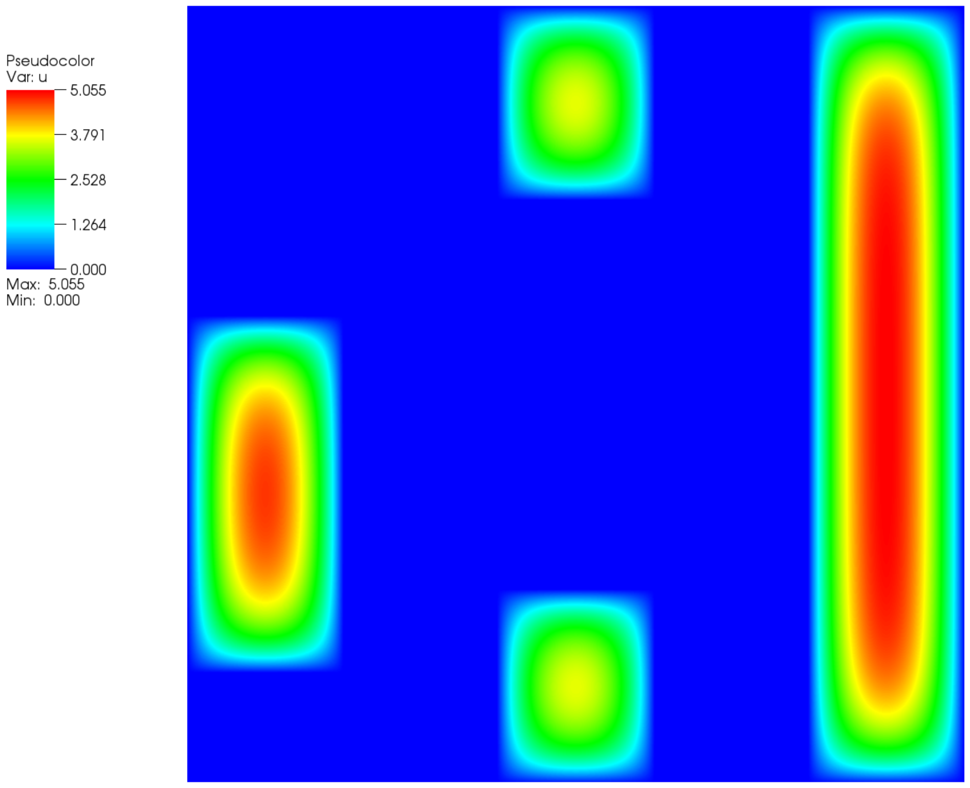


Figure 5: Test diffusion problem created where the diffusion coefficient changes discontinuously in space

4 DG Advection in Parallel

The deal.II tutorial program step 12 implements DG advection. The sample program in written to run on a single processor. The test AIR AMG as a solver, the step 12 tutorial program was changed to work in parallel and the AIR AMG solver was implemented. Figure 6 shows the domain decomposition after step 12 was run in parallel on four processors. The AIR solver is able to solve the problem in parallel. No specific results are presented in this section, however, parallel scaling results for AIR as implemented in *hypre* will be published in the near future.

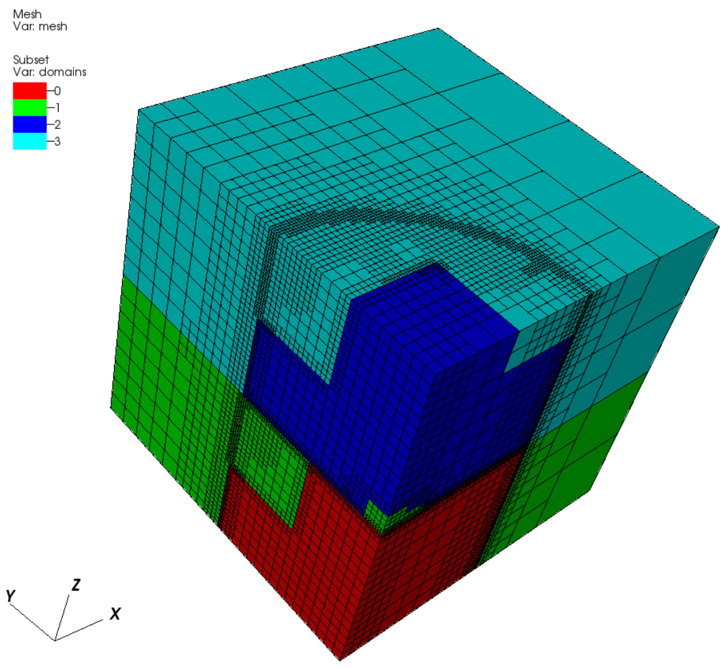


Figure 6: Domain decomposition after the step 12 problem was run in parallel on four processors.

References

[1] R.D. Falgout, J.E. Jones, and U.M. Yang. The Design and Implementation of hypre, a Library of Parallel High Performance Preconditioners, chapter in Numerical Solution of Partial Differential Equations on Parallel Computers, A.M. Bruaset and A. Tveito, eds., Springer-Verlag, 51 (2006), pp. 267-294. UCRL-JRNL-205459.

[2] G. Alzetta, D. Arndt, W. Bangerth, V. Boddu, B. Brands, D. Davydov, R. Gassmoeller, T. Heister, L. Heltai, K. Kormann, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. "The *deal.II* Library, Version 9.0." *Journal of Numerical Mathematics* (2018, accepted).

[3] C. Burstedde, L. C. Wilcox, and O. Ghattas. "*p4est*: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees." *SIAM Journal on Scientific Computing*, **volume 33**(3), pp. 1103-1133 (2011).

[4] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, An overview of the trilinos project, ACM Trans. Math. Softw., vol. 31, no. 3, pp. 397423, 2005.

[5] M. Sala and M. Heroux, Robust algebraic preconditioners with IFPACK 3.0, Tech. Rep. SAND-0662, Sandia National Laboratories, 2005.

[6] R. Falgout. An introduction to algebraic multigrid. *Computing in Science & Engineering*, **volume 8**(6), pp. 2433 (2009).

[7] T. A. Manteuffel, S. F. McCormick, S. Munzenmaier, J. W. Ruge, and B. S. Southworth. Reduction-based Algebraic Multgrid for Upwind Discretizations. *SIAM Journal on Scientific Computing* (submitted).

[8] T. A. Manteuffel, J. W. Ruge, and B. S. Southworth. Nonsymmetric Reduction-based Algebraic Multigrid Based on Local Approximate Ideal Restriction (LAIR). *SIAM Journal on Scientific Computing* (submitted).

[9] J. Donea and A. Huerta. "Finite Element Methods for Flow Problems." John Wiley & Sons Ltd, West Sussex, England, 2003.

Appendices

Interface Doxygen Documentation

deal.II ifpack Interface for BoomerAMG use as a Solver or Preconditioner

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 TrilinosWrappers::BoomerAMG_PreconditionedSolver Class Reference

This class serves as an interface to ifpack for using a hypre solver with a BoomerAMG preconditioner.

Public Member Functions

- BoomerAMG_PreconditionedSolver (BoomerAMGParameters &BoomerAMG_precond_parameters, ifpack↩ SolverParameters &solver_parameters)

    Constructor.
- void solve (LA::SparseMatrix &system_matrix, LA::Vector &right_hand_side, LA::Vector &solution)

    Solver function.

3.1.1 Detailed Description

This class serves as an interface to ifpack for using a hypre solver with a BoomerAMG preconditioner.

Author

Joshua Hanophy, 2019

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BoomerAMG_PreconditionedSolver()

```
TrilinosWrappers::BoomerAMG_PreconditionedSolver::BoomerAMG_PreconditionedSolver (
            BoomerAMGParameters & BoomerAMG_precond_parameters,
            ifpackSolverParameters & solver_parameters )  [inline]
```

Constructor.

Parameters

BoomerAMG_precond_parameters	is the instance of BoomerAMGParameters hanlding the BoomerAMG parameters
solver_parameters	is the instance of ifpackSolverParameters hanlding the solver parameters

3.1.3 Member Function Documentation

3.1.3.1 solve()

```
void TrilinosWrappers::BoomerAMG_PreconditionedSolver::solve (
            LA::SparseMatrix & system_matrix,
            LA::Vector & right_hand_side,
            LA::Vector & solution )
```

Solver function.

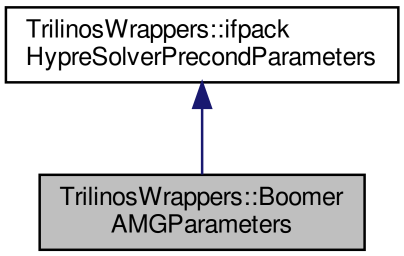
Parameters

system_matrix	is the system matrix
right_hand_side	it the right hand side for the system
solution	is the solution vector into which the solution will be written

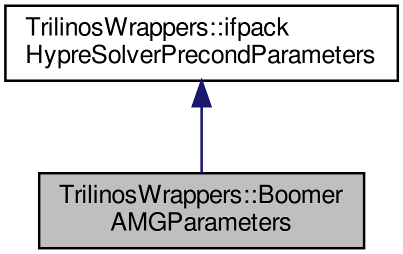
3.2 TrilinosWrappers::BoomerAMGParameters Class Reference

Class meant to handle BoomerAMG solver or preconditioner parameters.

Inheritance diagram for TrilinosWrappers::BoomerAMGParameters:


TrilinosWrappers::ifpack HypreSolverPrecondParameters
TrilinosWrappers::Boomer AMGParameters

Collaboration diagram for TrilinosWrappers::BoomerAMGParameters:



Public Types

- enum AMG_type { CLASSICAL_AMG, AIR_AMG, NONE }

  Enum storing possible default parameter configurations.

Public Member Functions

- BoomerAMGParameters (AMG_type config_selection, Hypre_Chooser solver_preconditioner_selection)

  Constructor.

Additional Inherited Members

3.2.1 Detailed Description

Class meant to handle BoomerAMG solver or preconditioner parameters.

This class adds little functionality to its base class, but includes a comprehensive list of default parameters that may be of interest for BoomerAMG when used as either a solver or a preconditioner. The constructor for this class is of primary interest and sets a number of parameters.

The default parameters are:

String Name	Description

interp_type	The interp_type integer variable sets the interpolation type. Interpolation types, taken from the hypre documentation, are: • 0: classical modified interpolation • 1: LS interpolation (for use with GSMG) • 2: classical modified interpolation for hyperbolic PDEs • 3: direct interpolation (with separation of weights) • 4: multipass interpolation • 5: multipass interpolation (with separation of weights) • 7: extended+i interpolation • 8: standard interpolation • 9: standard interpolation (with separation of weights) • 10: classical block interpolation (for use with nodal systems version only) • 11: classical block interpolation (for use with nodal systems version only) with diagonalized diagonal blocks • 12: FF interpolation • 13: FF1 interpolation • 14: extended interpolation • 100: Pointwise interpolation (intended for use with AIR)
pre_post_relax	The prerelax string specifies the points, order, and relaxation steps for prerelaxation. The options are "A", "F", or "C" where A is relaxation over all points, F is relaxation over the F-points, and C is relaxation over the C-points. Multiple characters specify multiple relaxation steps and the order matters. For example, "AA" specifies two relaxation steps of all points. The postrelax string specifies the points, order, and relaxation steps for postrelaxation. The options are "A", "F", or "C" where A is relaxation over all points, F is relaxation over the F-points, and C is relaxation over the C-points. Multiple characters specify multiple relaxation steps and the order matters. For example, "FFFC" specifies three post relaxations over F-points followed by a relexation over C-points.

relax_type	The relax_type integer variable sets the relaxation type. Relaxation types, taken from the hypre documentation, are: • 0: Jacobi • 1: Gauss-Seidel, sequential (very slow!) • 2: Gauss-Seidel, interior points in parallel, boundary sequential (slow!) • 3: hybrid Gauss-Seidel or SOR, forward solve • 4: hybrid Gauss-Seidel or SOR, backward solve • 5: hybrid chaotic Gauss-Seidel (works only with OpenMP) • 6: hybrid symmetric Gauss-Seidel or SSOR • 8: ℓ_1 Gauss-Seidel, forward solve • 9: Gaussian elimination (only on coarsest level) • 13: ℓ_1 Gauss-Seidel, forward solve • 14: ℓ_1 Gauss-Seidel, backward solve • 15: CG (warning - not a fixed smoother - may require FGMRES) • 16: Chebyshev • 17: FCF-Jacobi • 18: ℓ_1-scaled jacobi
coarsen_type	The coarsen_type integer variable sets the coarsening algorithm. Coarsening algorithm options, taken from the hypre documentation, are: • 0: CLJP-coarsening (a parallel coarsening algorithm using independent sets. • 3: classical Ruge-Stueben coarsening on each processor, followed by a third pass, which adds coarse points on the boundaries • 6: Falgout coarsening (uses 1 first, followed by CLJP using the interior coarse points generated by 1 as its first independent set) • 8: PMIS-coarsening (a parallel coarsening algorithm using independent sets, generating lower complexities than CLJP, might also lead to slower convergence) • 10: HMIS-coarsening (uses one pass Ruge-Stueben on each processor independently, followed by PMIS using the interior C-points generated as its first independent set) • 21: CGC coarsening by M. Griebel, B. Metsch and A. Schweitzer • 22: CGC-E coarsening by M. Griebel, B. Metsch and A.Schweitzer
max_levels	The max_levels integer specifies the maximum number of AMG that hypre will be allowed to use

cycle_type	The cycle_type integer variable sets the cycle type. Cycle types available, taken from the hypre documentation, are: • 0: F-cycle type • 1: V-cycle type • 2: W-cycle type
sabs_flag	sabs_flag sets whether the classical strength of connection test based on testing the negative of matrix coefficient or if the absolute value is tested. If set to 0, the negative coefficient values are tested, if set to 1, the absolute values of matrix coefficients are tested.
distance_R	The distance_R double variable sets whether Approximate Ideal Restriction (AIR) multigrid or classical multigrid is used. • 0.0: Use classical AMG, not AIR • 1.0: Use AIR, Distance-1 LAIR is used to compute R • 2.0: Use AIR, Distance-2 LAIR is used to compute R • 3.0: Use AIR, degree 0 Neumann expansion is used to compute R • 4.0: Use AIR, degree 1 Neumann expansion is used to compute R • 5.0: Use AIR, degree 2 Neumann expansion is used to compute R

Author

Joshua Hanophy, Ben Southworth 2019

3.2.2 Member Enumeration Documentation

3.2.2.1 AMG_type

```
enum TrilinosWrappers::BoomerAMGParameters::AMG_type
```

Enum storing possible default parameter configurations.

Generally, different sets of parameters may be of interest for different AMG solvers. The AMG_type enum determines which parameters are set when an instance of BoomerAMGParameters is constructed.

Enumerator

CLASSICAL_AMG	Load default parameters consistent with Classical AMG. CLASSICAL_AMG
AIR_AMG	Load default parameters consistent with AIR. AIR_AMG
NONE	Create an empty parameter map. NONE

3.2.3 Constructor & Destructor Documentation

3.2.3.1 BoomerAMGParameters()

```
TrilinosWrappers::BoomerAMGParameters::BoomerAMGParameters (
            AMG_type config_selection,
            Hypre_Chooser solver_preconditioner_selection )
```

Constructor.

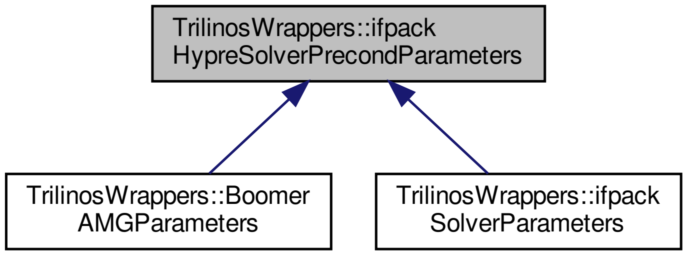
Parameters

config_selection	is the AMG_type specifying what default parameters should be used
solver_preconditioner_selection	is either Hypre_Chooser:Solver or Hypre_Chooser:Preconditioner and specifies whether the parameter object will be used of a solver or a preconditioner.

3.3 TrilinosWrappers::ifpackHypreSolverPrecondParameters Class Reference

This class is meatn to handle parameters that are used by hypre solvers and preconditioners.

Inheritance diagram for TrilinosWrappers::ifpackHypreSolverPrecondParameters:


TrilinosWrappers::ifpack HypreSolverPrecondParameters
TrilinosWrappers::Boomer AMGParameters
TrilinosWrappers::ifpack SolverParameters

Classes

- struct parameter_data

  This struct holds the data required for a single parameter.

Public Types

- typedef boost::variant< int(∗)(HYPRE_Solver, int), int(∗)(HYPRE_Solver, double), int(∗)(HYPRE_Solver, double, int), int(∗)(HYPRE_Solver, int, int), int(∗)(HYPRE_Solver, int ∗), int(∗)(HYPRE_Solver, double ∗), int(∗)(HYPRE_Solver, int ∗∗), std::nullptr_t > hypre_function_variant

  This type is meant to be used for a pointer to a hypre set function.
- typedef boost::variant< int, double, int ∗, int ∗∗, std::pair< double, int >, std::pair< int, int >, std::pair< std::string, std::string > > param_value_variant

  This type is meant to be used for a parameter value.

Public Member Functions

- ifpackHypreSolverPrecondParameters (Hypre_Chooser solver_preconditioner_selection)

  Constructor.
- void set_parameter_value (std::string name, param_value_variant value)

  This function can be used to change the value of a parameter in the parameters map that already exists.
- void add_parameter (std::string name, parameter_data param_data)

  This function can be used to add a new parameter to the.
- void remove_parameter (std::string name)

  Function to remove a parameter from the parameters parameter map.
- template<typename return_type >
  void return_parameter_value (std::string name)

  Function to return the value of a parameter.
- void set_parameters (Ifpack_Hypre &Ifpack_obj)

  This function is to be used by the solver or preconditioner class to set the parameter values.

Protected Attributes

- std::map< std::string, parameter_data > parameters

  The parameters map stores parameters as a string name key and then a parameter_data instance value.

3.3.1 Detailed Description

This class is meatn to handle parameters that are used by hypre solvers and preconditioners.

It stores parameter values and also interfaces with an ifpack_Hypre object to actually set the parameter values.

Author

Joshua Hanophy 2019

3.3.2 Member Typedef Documentation

3.3.2.1 hypre_function_variant

```
typedef boost::variant<int (∗)(HYPRE_Solver, int),int (∗)(HYPRE_Solver, double),int (∗)(HYPR←
E_Solver,double, int), int (∗)(HYPRE_Solver, int, int),int (∗)(HYPRE_Solver, int∗),int (∗)(H←
YPRE_Solver, double∗), int (∗)(HYPRE_Solver, int∗∗),std::nullptr_t> TrilinosWrappers::ifpack←
HypreSolverPrecondParameters::hypre_function_variant
```

This type is meant to be used for a pointer to a hypre set function.

This is simply an alias for a boost variant type. Note the types in the boos variant contianer are those function pointer types supported by this interface.

3.3.2.2 param_value_variant

```
typedef boost::variant<int,double,int*,int**,std::pair<double,int>, std::pair<int, int>,
std::pair<std::string,std::string> > TrilinosWrappers::ifpackHypreSolverPrecondParameters↩
::param_value_variant
```

This type is meant to be used for a parameter value.

This is simply an alias for a boost variant type.

3.3.3 Constructor & Destructor Documentation

3.3.3.1 ifpackHypreSolverPrecondParameters()

```
TrilinosWrappers::ifpackHypreSolverPrecondParameters::ifpackHypreSolverPrecondParameters (
             Hypre_Chooser solver_preconditioner_selection )  [inline]
```

Constructor.

Parameters

solver_preconditioner_selection	is either a Hypre_Chooser enum::Solver or Hypre_Chooser enum::Solver preconditioner and specifies whether the instance will handle paramets for a solver or a preconditioner.

3.3.4 Member Function Documentation

3.3.4.1 add_parameter()

```
void TrilinosWrappers::ifpackHypreSolverPrecondParameters::add_parameter (
             std::string name,
             parameter_data param_data )
```

This function can be used to add a new parameter to the.

Parameters

name	is the string parameter name. Note that the parameter name should not already exist. To update the value or a parameter that already exists, use the set_parameter_value function
param_data	is an instance of the parameter_data struct which contains the parameter value and a pointer to the proper set function

3.3.4.2 remove_parameter()

```
void TrilinosWrappers::ifpackHypreSolverPrecondParameters::remove_parameter (
            std::string name )
```

Function to remove a parameter from the parameters parameter map.

Parameters

name	is the string parameter name to remove.

3.3.4.3 return_parameter_value()

```
template<typename return_type >
void TrilinosWrappers::ifpackHypreSolverPrecondParameters::return_parameter_value (
            std::string name )
```

Function to return the value of a parameter.

Parameters

name	is the string parameter name of the parameter whose value is to be returned

3.3.4.4 set_parameter_value()

```
void TrilinosWrappers::ifpackHypreSolverPrecondParameters::set_parameter_value (
            std::string name,
            param_value_variant value )
```

This function can be used to change the value of a parameter in the parameters map that already exists.

Use the add_parameter function if the parameter does not already exist

Parameters

name	is the string parameter name. Note that the parameter name should already exist in the parameters parameter map. Use the add_parameter function to add a new parameters.
value	This is the value to assign the parameter found at parameters[name]

3.3.4.5 set_parameters()

```
void TrilinosWrappers::ifpackHypreSolverPrecondParameters::set_parameters (
            Ifpack_Hypre & Ifpack_obj )
```

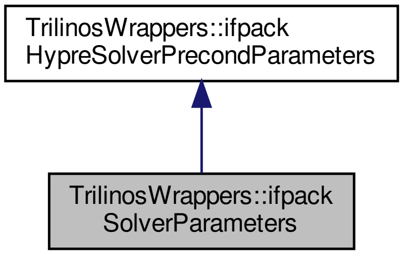
This function is to be used by the solver or preconditioner class to set the parameter values.

Note that all parameters contained in the parameters map will be set.

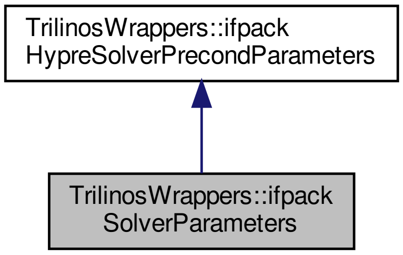
3.4 TrilinosWrappers::ifpackSolverParameters Class Reference

This class adds minimal functionality to its base class.

Inheritance diagram for TrilinosWrappers::ifpackSolverParameters:



Collaboration diagram for TrilinosWrappers::ifpackSolverParameters:



Public Member Functions

- ifpackSolverParameters (Hypre_Solver solver_selection=Hypre_Solver::PCG)

  Constructor.

Public Attributes

- Hypre_Solver **solver_selection**

Additional Inherited Members

3.4.1 Detailed Description

This class adds minimal functionality to its base class.

It is meant to be used to handle parameters for a hypre solver other than BoomerAMG. In particular, it is used by BoomerAMG_PreconditionedSolver to handle the solver parameters.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ifpackSolverParameters()

```
TrilinosWrappers::ifpackSolverParameters::ifpackSolverParameters (
            Hypre_Solver solver_selection = Hypre_Solver::PCG )  [inline]
```

Constructor.

Parameters

solver_selection	specifies the solver type to be used.

3.5 TrilinosWrappers::ifpackHypreSolverPrecondParameters::parameter_data Struct Reference

This struct holds the data required for a single parameter.

Public Member Functions

- parameter_data (param_value_variant value, hypre_function_variant hypre_function)

  Constructor.
- parameter_data (param_value_variant value, std::function< void(const Hypre_Chooser, const parameter_↩ data &, Ifpack_Hypre &)> set_function)

  Constructor.

Public Attributes

- param_value_variant value

  value stores the value of the parameter
- hypre_function_variant hypre_function =nullptr

  hypre_function stores a pointer to the hypre set function to be used to set the parameter value.
- std::function< void(const Hypre_Chooser, const parameter_data &, Ifpack_Hypre &)> set_function =nullptr

  set_function stores a pointer to a custom set function.

3.5.1 Detailed Description

This struct holds the data required for a single parameter.

The required data is a parameter value and a set function. There are two possibilities for a set function. A pointer to a hypre set function defined in the hypre library can used and will be stored as hypre_function. Or if a custom set function is desired in place of directly using the predfined hypre set functions, a pointer to a custom set function is stored as set_function. Note that either set_function or hypre_function shoudl be a nullptr.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 parameter_data() [1/2]

```
TrilinosWrappers::ifpackHypreSolverPrecondParameters::parameter_data::parameter_data (
            param_value_variant value,
            hypre_function_variant hypre_function )  [inline]
```

Constructor.

Parameters

value	is the value of the parameter
hypre_function	is a pointer to the hypre set function defined in the hypre library

3.5.2.2 parameter_data() [2/2]

```
TrilinosWrappers::ifpackHypreSolverPrecondParameters::parameter_data::parameter_data (
            param_value_variant value,
            std::function< void(const Hypre_Chooser, const parameter_data &, Ifpack_Hypre &)>
set_function )  [inline]
```

Constructor.

Parameters

value	is the value of the parameter
set_function	is a pointer to a custom set function

3.5.3 Member Data Documentation

3.5.3.1 hypre_function

```
hypre_function_variant TrilinosWrappers::ifpackHypreSolverPrecondParameters::parameter_data↩
::hypre_function =nullptr
```

hypre_function stores a pointer to the hypre set function to be used to set the parameter value.

Note is this is used then hypre_function should be equal to a nullptr

3.5.3.2 set_function

```
std::function<void(const Hypre_Chooser, const parameter_data &, Ifpack_Hypre &)> Trilinos↩
Wrappers::ifpackHypreSolverPrecondParameters::parameter_data::set_function =nullptr
```

set_function stores a pointer to a custom set function.

This can be used if simply setting a parameter value with the set functions predifined in the hypre library is not sufficient. If this is used, hypre_function should be equal to nullptr

3.6 TrilinosWrappers::SolverBoomerAMG Class Reference

This class serves as an interface to ifpack for using a BoomerAMG as a solver.

Public Member Functions

- SolverBoomerAMG (BoomerAMGParameters &SolverParameters)

  Constructor.
- void solve (LA::SparseMatrix &system_matrix, LA::Vector &right_hand_side, LA::Vector &solution)

  Solver function.

3.6.1 Detailed Description

This class serves as an interface to ifpack for using a BoomerAMG as a solver.

Author

Joshua Hanophy, 2019

3.6.2 Constructor & Destructor Documentation

3.6.2.1 SolverBoomerAMG()

```
TrilinosWrappers::SolverBoomerAMG::SolverBoomerAMG (
            BoomerAMGParameters & SolverParameters )  [inline]
```

Constructor.

Parameters

SolverParameters	is the instance of BoomerAMGParameters container the parameter that the solver will use.

3.6.3 Member Function Documentation

3.6.3.1 solve()

```
void TrilinosWrappers::SolverBoomerAMG::solve (
             LA::SparseMatrix & system_matrix,
             LA::Vector & right_hand_side,
             LA::Vector & solution )
```

Solver function.

Parameters

system_matrix	is the system matrix
right_hand_side	it the right hand side for the system
solution	is the solution vector into which the solution will be written