Abstract: We describe the Overture class Oges, an “Overlapping Grid Equation Solver”, that can be used for the solution of sparse matrix equations on overlapping grids such that those created by the grid generator Ogen. Oges acts as a front end to a variety of sparse matrix solvers including direct sparse solvers such as those from Yale or Harwell or iterative solvers (from SLAP and PETSc) that use algorithms such as conjugate gradient or GMRES.

To use Oges one must first generate a a system of equations (usually defining a PDE boundary value problem) using the ‘coefficient matrix’ grid functions and the Overture operator classes. Oges will take a coefficient matrix generated in this way and then call the appropriate sparse matrix solver. Oges can be easily extended to use a new Sparse matrix package.

This document is available from the Overture home page, http://www.llnl.gov/casc/Overture.

Contents

1 Introduction 4

2 Example Codes 5

3 Initial guesses for iterative solvers 5

4 Parallel Results 6

4.1 Circle in a Channel ......................................................... 7
4.2 Cylinder in a Channel ...................................................... 8

5 Oges Parameters 9

5.1 operator= ................................................................. 9
5.2 getSolverName ............................................................. 9
5.3 getSolverTypeName ....................................................... 9
5.4 getSolverMethodName .................................................... 9
5.5 getPreconditionerName .................................................. 9
5.6 getMatrixOrderingName .................................................. 10
5.7 set( OptionEnum , int ) .................................................... 10
10.22 set(OptionEnum, int) ......................................................... 23
10.23 set(OptionEnum, float) .................................................. 24
10.24 set(OptionEnum, double) .............................................. 24
10.25 set(SolverEnum) ............................................................. 24
10.26 set(SolverMethodEnum) ................................................. 24
10.27 set(PreconditionerEnum) .............................................. 24
10.28 set(MatrixOrderingEnum) .............................................. 24
10.29 get(OptionEnum, int&) .................................................. 25
10.30 get(OptionEnum, real&) ................................................. 25
10.31 setOgesParameters ....................................................... 25
10.32 sizeof ................................................................. 25
10.33 printStatistics ............................................................ 25
10.34 updateToMatchGrid ..................................................... 26
10.35 updateToMatchGrid ..................................................... 26
10.36 getMatrix ............................................................... 26
Introduction

We describe the Overture class Oges, an “Overlapping Grid Equation Solver”, that can be used for the solution of sparse matrix equations on overlapping grids such that those created by the grid generator Ogen. Oges acts as a front end to a variety of sparse matrix solvers. Currently we have support for

Yale : direct sparse matrix package (no pivoting).

Harwell : direct sparse matrix package with partial pivoting.

SLAP : The Sparse Linear Algebra Package from Greenbaum and Seager, an iterative solver package, includes conjugate gradient and gmres solvers.

PETSc : The Portable Extensible Toolkit for Scientific computations from iterative solver package, includes conjugate gradient and gmres solvers in addition to many others. Thanks to Petri Fast for writing the interface (PETScEquationSolver) to the serial version of PETSc. There is also a newer interface (PETScSolver) for use with PETSc in parallel.

By changing one or two parameters the user may easily try a different solver. For example, although Yale is in general faster than Harwell, the latter, which does pivoting, may be better for some problems. The
SLAP and PETSc iterative solvers may be especially useful for very large problems when storage is at a premium.

Oges can be easily extended by you to use a new Sparse matrix package.

To use Oges one must first generate a system of equations (usually defining a PDE boundary value problem) using the 'coefficient matrix' grid functions and the Overture operator classes \[1\][3][2]. A ‘coefficient matrix’ is stored in a realCompositeGridFunction. Typically the creation of a PDE boundary value problem will look something like

```
CompositeGrid cg(...);
realCompositeGridFunction coeff(...);
CompositeGridOperators op(cg);
...
coeff=op.laplacianCoefficients();    // form the laplace operator
coeff.applyBoundaryConditionCoefficients(0,0,dirichlet, allBoundaries);
coeff.applyBoundaryConditionCoefficients(0,0,extrapolate,allBoundaries);
coeff.finishBoundaryConditions();
```

Oges will take a coefficient matrix generated in this way and then call the appropriate sparse matrix solver. Usually this will involve converting the ‘coefficient matrix’ representation to some other representation such as a compressed-row storage format (this is done automatically by Oges).

Given a coefficient matrix, Oges can be used as follows

```
Oges solver(cg);     // build a solver
// use the yale solver:
solver.set(OgesParameters::THEsolverType,OgesParameters::yale);
// ...or... use PETSc
solver.set(OgesParameters::THEsolverType,OgesParameters::PETSc);

solver.setCoefficientArray( coeff );    // supply coefficients
realCompositeGridFunction u(cg),f(cg);    // build solution and right-hand-side
...
```

Generally one must also set other parameters such as the convergence tolerance, preconditioner, etc, when using iterative solvers such as SLAP or PETSC.

The global variable Oges::debug is a bit flag that generates various diagnostic output from Oges. Setting Oges::debug=63 (63=1+2+4+8+16+32) will generate lots of debugging output. Setting Oges::debug=3 will generate only some debugging output.

## 2 Example Codes


## 3 Initial guesses for iterative solvers

When using an iterative solver, the grid function holding the solution should contain an initial guess. Choose zero if you don’t have any idea. If you do provide and initial guess then all points used in the discretization should be given initial values including interpolation points and ghost points.
4 Parallel Results

Here are some parallel results using Oges.

First some comments about the form of the parallel algorithms.

Krylov space methods (KSP) are a common approach to solving linear systems. Examples of KSP algorithms are GMRES and BCGS (bi-conjugate-gradient-stabilized).

A KSP method will generally use a preconditioner (PC) which acts as an approximate inverse. Incomplete-LU is an example of a PC.

In parallel it is common to use a block-Jacobi preconditioner (there may be 1 block per processor, for e.g.). This means that each block is independently updated and then boundary information between the blocks is synchronized. Within each block one may again use a KSP method (referenced as the sub_ksp in PETSc) with optional PC (referenced as the sub_pc in PETSc).

A typical case will use ksp=BCGS, pc=block-Jacobi, sub_ksp=preconditioner-only, sub_pc=ILU(2).

The Hypre package (which can be accessed through the PETSc interface) has a number of preconditioners. One of these is a parallel AMG (algebraic multigrid) method call BoomerAMG. Although the AMG method can be used as a stand alone solver (Using ksp=Richardson and pc=boomeramg), it is generally more robust to use AMG as a preconditioner for a KSP method such as ksp=BCGS and pc=boomeramg.

In the first set of results we solve Poisson’s equation on an overlapping grid. We consider different boundary conditions

*dirichlet* : Dirichlet BC’s on all boundaries

*neumann* : Neumann BC’s on all boundaries

*mixed* : Neumann BC’s on all boundaries but one, where a mixed boundary condition $\alpha u + \beta u_n$ is applied (with $\alpha = 1$, $\beta = 1$). This option represents a common set of BC’s when solving the pressure equation for the incompressible Navier-Stokes equations.
4.1 Circle in a Channel

Results from solving Poisson’s equation on the two-dimensional circle-in-a-channel grid, cic5e (2.8e5 grid points), are computed for both dirichlet and mixed boundary conditions.

Table 1 shows the results using BCGS with a block-jacobi preconditioner where each block uses an ILU preconditioner, ILU(2).

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.16</td>
<td>16.83</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>3.90</td>
<td>12.29</td>
<td>141</td>
</tr>
<tr>
<td>4</td>
<td>2.30</td>
<td>5.77</td>
<td>133</td>
</tr>
<tr>
<td>8</td>
<td>1.13</td>
<td>3.13</td>
<td>139</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.61</td>
<td>45.54</td>
<td>275</td>
</tr>
<tr>
<td>2</td>
<td>3.97</td>
<td>32.29</td>
<td>378</td>
</tr>
<tr>
<td>4</td>
<td>2.24</td>
<td>17.72</td>
<td>412</td>
</tr>
<tr>
<td>8</td>
<td>1.13</td>
<td>9.19</td>
<td>411</td>
</tr>
</tbody>
</table>

Table 1: Results for grid cic5e (2.8e5 grid points), bcgss-bjacobi-ilu(2) (mcr).

Table 2 shows the results using BCGS with a block-jacobi preconditioner where each block uses an LU (direct solver). These results show the limitations of the BCGS+block-jacobi combination. When we perform a direct solve on each block (by a back-substitution) the number of BCGS iterations is reduced from the previous case. The overall solve time is longer, however.

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>14.96</td>
<td>19.59</td>
<td>69</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>48.21</td>
<td>49.53</td>
<td>82</td>
</tr>
</tbody>
</table>

Table 2: Results for grid cic5e (2.8e5 grid points), bcgss-bjacobi-lu (mcr).

Table ?? shows the results using the AMG method, BoomerAMG, from Hypre.

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.57</td>
<td>4.40</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6.18</td>
<td>2.39</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3.56</td>
<td>1.25</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>1.77</td>
<td>0.69</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.94</td>
<td>8.99</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>5.48</td>
<td>4.39</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>3.15</td>
<td>2.33</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>1.69</td>
<td>1.16</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 3: Results for grid cic5e (2.8e5 grid points), bcgss-hypre-AMG (mcr).
4.2 Cylinder in a Channel

Results from solving Poisson’s equation on the three-dimensional cylinder-in-a-channel grid cylinderInAChannel2 (1.2e6 grid points) are computed for both dirichlet and mixed boundary conditions.

Table 4 shows the results using BCGS with a block-jacobi preconditioner where each block uses an ILU preconditioner, ILU(2).

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25.48</td>
<td>53.25</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>226.41</td>
<td>37.32</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>95.76</td>
<td>20.36</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>33.03</td>
<td>9.43</td>
<td>51</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>96.58</td>
<td>53.58</td>
<td>138</td>
</tr>
<tr>
<td>8</td>
<td>32.54</td>
<td>30.78</td>
<td>162</td>
</tr>
</tbody>
</table>

Table 4: Results for grid cylinderInAChannel2 (1.2e6 grid points), bcgs-bjacobi-ilu(2) (mcr).

Table ?? shows the results using the AMG method, BoomerAMG, from Hypre.

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>40.49</td>
<td>18.48</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>248.22</td>
<td>13.30</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>108.57</td>
<td>6.88</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>40.28</td>
<td>4.04</td>
<td>5</td>
</tr>
<tr>
<td>16</td>
<td>14.59</td>
<td>2.61</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NP</th>
<th>factor(s)</th>
<th>solve(s)</th>
<th>its</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>failed</td>
<td>failed</td>
<td>failed</td>
</tr>
</tbody>
</table>

Table 5: Results for grid cylinderInAChannel2 (1.2e6 grid points), bcgs-hypre-AMG (mcr).
5 Oges Parameters

Solver dependent parameters are found in the OgesParameters class. It is a container class for such parameters are the type of solver, type of preconditioner, convergence tolerance etc. Oges contains an OgesParameters object to hold these parameters. Parameters can be set by directly using the Oges set functions. This will indirectly set the values in an OgesParameters object contained in an Oges object. Alternatively one can first create an OgesParameters object, set parameters in that object and then provide the OgesParameters object to Oges using the setParameters function (which will copy you values into it’s local version). Parameters can also be set interactively by calling the Oges update function or the OgesParameters update function.

5.1 operator=

OgesParameters&
operator=(const OgesParameters& x)

Description: deep copy of data.

5.2 getSolverName

aString
getSolverName() const

Description: Return the name of the solver, a composite of the solver type, method and preconditioner.

5.3 getSolverTypeName

aString
getSolverTypeName(SolverEnum solverType = defaultSolver) const

Description: Return the name of the solverType such as ”yale”, ”harwell”, ”SLAP”, ... By default return the name of the currently chosen solver.

solverType (input) : return the name of this solver type. By default return the name of the currently chosen solver.

5.4 getSolverMethodName

aString
getSolverMethodName(SolverMethodEnum solverMethodType = defaultSolverMethod) const

Description: Return the name of the solver method such as ”gmres”. By default return the name of the currently chosen method.

solverMethodType (input):

5.5 getPreconditionerName

aString
getPreconditionerName(PreconditionerEnum preconditionerType = defaultPreconditioner) const
Description: Return the name of the preconditioner. By default return the name of the currently chosen preconditioner.

preconditionerType (input):

5.6 getMatrixOrderingName

aString
getMatrixOrderingName(MatrixOrderingEnum matrixOrderingType = defaultMatrixOrdering) const

Description: Return the name of the matrix ordering. By default return the name of the currently chosen matrix ordering.

matrixOrderingType (input):

5.7 set( OptionEnum , int )

int
set( OptionEnum option, int value = 0)

Description: Set an int option from the OptionEnum.

enum OptionEnum
{
    THEabsoluteTolerance,
    THEisAxisymmetric,   // for predefined equations
    THEbestIterativeSolver,   // choose the 'best' iterative solver and options.
    THEbestDirectSolver,   // choose the 'best' direct solver and options.
    THEcompatibilityConstraint,
    THEfillinRatio,
    THEfillinRatio2,
    THEfixupRightHandSide,
    THEgmresRestartLength,
    THEharwellPivotingTolerance,
    THEincompleteLUExpectedFill,
    THEiterativeImprovement,
    THEkeepCoefficientGridFunction,   // keep a reference to the user’s coeff grid function
    THEkeepSparseMatrix,   // keep ia, ja, a sparse matrix even it not needed by the
    THEmatrixCutoff,
    THEmatrixOrdering,
    THEmaximumInterpolationWidth,
    THEmaximumNumberOfIterations,
    THEminimumNumberOfIterations,
    THEnullVectorScaling,
    THEnumberOfIncompleteLULevels,
    THEsolveForTranspose,
    THEpreconditioner,
    THEparallelPreconditioner,
    THEexternalSolver,
    THEparallelExternalSolver,
    THEremoveSolutionAndRHSVector,   // de-allocate sol and rhs vector after every solve
}
THEremoveSparseMatrixFactorization, // de-allocate any factorization info after every solve.
THErelativeTolerance,
THErescaleRowNorms,
THEsolverType,
THEsolverMethod,
THEparallelSolverMethod,
THEtolerance,
THEuserSuppliedCompatibilityConstraint,
THEzeroRatio
};

5.8 set( OptionEnum , float )

int
set( OptionEnum option, float value )

Description: Set a real valued option from the OptionEnum.

5.9 set( OptionEnum , double )

int
set( OptionEnum option, double value )

Description: Set a real valued option from the OptionEnum.

5.10 set( SolverEnum )

int
set( SolverEnum option )

Description: Set the solver, a value from the SolverEnum.

enum SolverEnum
{
    defaultSolver,
sor,
yale,
harwell,
SLAP,
PETSc,
multigrid,
PETScNew,
userSolver1, // these are reserved for new user defined solvers.
userSolver2,
userSolver3,
userSolver4,
userSolver5
};
5.11  set( SolverMethodEnum )

int
set( SolverMethodEnum option )

Description: Set the solver method, a value from the SolverMethodEnum.

enum SolverMethodEnum
{
    richardson,
    chebychev,
    conjugateGradient,
    cg=conjugateGradient,  // cg = short PETSc name
    biConjugateGradient,
    bicg=biConjugateGradient,
    conjugateGradientSquared,
    cgs=conjugateGradientSquared,
    biConjugateGradientSquared,
    biConjugateGradientStabilized,
    bcgs=biConjugateGradientStabilized,
    generalizedMinimalResidual,
    gmres=generalizedMinimalResidual,
    transposeFreeQuasiMinimalResidual,
    tfqmr=transposeFreeQuasiMinimalResidual,
    transposeFreeQuasiMinimalResidual2,  // tcqmr Tony Chan’s version
    tcqmr=transposeFreeQuasiMinimalResidual,
    conjugateResidual,
    cr=conjugateResidual,
    leastSquares,
    lsqr=leastSquares,
    preonly,
};

5.12  set( PreconditionerEnum )

int
set( PreconditionerEnum option )

Description: Set the preconditioner, a value from the PreconditionerEnum.

enum PreconditionerEnum
{
    noPreconditioner,
    jacobiPreconditioner,
    sorPreconditioner,
    luPreconditioner,
    shellPreconditioner,
    blockJacobiPreconditioner,
    multigridPreconditioner,
    eisenstatPreconditioner,
    incompleteCholeskyPreconditioner,
5.13 set( MatrixOrderingEnum )

int
set( MatrixOrderingEnum option )

Description: Set the matrix ordering, a value from the MatrixOrderingEnum.

enum MatrixOrderingEnum
{
    naturalOrdering,
    nestedDisectionOrdering,
    oneWayDisectionOrdering,
    reverseCuthillMcKeeOrdering,
    quotientMinimumDegreeOrdering,
    rowlengthOrdering
};

5.14 setPetscOption

int
setPetscOption( const aString & name, const aString & value )

Description: Set a PETSc option: example: name="-ksp_type" value="bcgs"

name (input) : the name of a Petsc option, e.g. name="-ksp_type"
value (input) : the value (as a string) of the petsc option, e.g. value="bcgs" or value="1.0"

5.15 setPetscOption

bool
getPetscOption( const aString & name, aString & value ) const

Description: Get a PETSc option (if it exists): example: name="-ksp_type" value="bcgs"

name (input) : the name of a Petsc option, e.g. name="-ksp_type"
value (output) : the value (as a string) of the petsc option, e.g. value="bcgs" or value="1.0"

// /return value; true if found, false if not found: // /return value; true if found, false if not found
5.16  getSolverType
SolverEnum
getSolverType() const

Description: Return the solverType.

5.17  get( OptionEnum , int & )
int
get( OptionEnum option, int & value ) const

Description: Get the value of an ‘int’ valued option.

5.18  get( OptionEnum , real & )
int
get( OptionEnum option, real & value ) const

Description: Get the value of an ‘real’ valued option.

5.19  getOgmgParameters
OgmgParameters*
getOgmgParameters() const

Description: Return a pointer to the OgmgParameters object. This pointer may be NULL.

5.20  buildOgmgParameters
OgmgParameters&
buildOgmgParameters()

Description: Create the OgmgParameters object if it is not there; return a reference to the object.

5.21  get from a data base
int
get( const GenericDataBase & dir, const aString & name)

Description: Get a copy of the OgesParameters from a database file
dir (input): get from this directory of the database.
name (input): the name of Oges on the database.

5.22  put to a data base
int
put( GenericDataBase & dir, const aString & name) const

Description: Output an image of OgesParameters to a data base.
dir (input): put onto this directory of the database.
name (input): the name of Oges on the database.
5.23 display

int
display(FILE *file = stdout)

Description: Print out current values of parameters

file (input): print to this file (standard output by default).

5.24 update

int
update( GenericGraphicsInterface & gi, CompositeGrid & cGrid )

Description: Update parameters interactively.

gi: use this graphics interface.

cg: parameters will apply to this grid.

5.25 isAvailable(SolverEnum)

int
isAvailable( SolverEnum solverType )

Description: Return TRUE if a given solver (esp. PETSc) is available.

5.26 isSolverIterative

bool
isSolverIterative() const

Description: Return TRUE if the solver chosen is an iterative method

5.27 buildEquationSolvers

int
buildEquationSolvers( SolverEnum solver)

Description: This function will build an equation solver of a particular type. This function is found in the Oges/buildEquationSolvers.C file. It is this file that you may have to copy and edit in order to turn on the availability solvers that are not distributed with Overture (such as PETSc).

6 Convergence criteria

There are many ways to define convergence criteria for iterative methods. The trick for Oges is to have a reasonable uniform way of defining a convergence tolerance for the different methods.

The standard PETSc convergence test is

$$\|r_k\|_2 < \max(\text{rtol} \times \|r_0\|_2, \text{atol})$$  \quad \text{PETSc}

where

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

The SLAP convergence test is somewhat different:
SGMRES solves a linear system $A*X = B$ rewritten in the form:

$$(SB*A*(M^{-1})*(SX^{-1}))*(SX*M*X) = SB*B,$$

with right preconditioning, or

$$(SB*(M^{-1})*A*(SX^{-1}))*(SX*X) = SB*(M^{-1})*B,$$

with left preconditioning, where $A$ is an $N$-by-$N$ real matrix, $X$ and $B$ are $N$-vectors, $SB$ and $SX$ are diagonal scaling matrices, and $M$ is a preconditioning matrix. It uses preconditioned Krylov subspace methods based on the generalized minimum residual method (GMRES). This routine optionally performs either the full orthogonalization version of the GMRES algorithm or an incomplete variant of it. Both versions use restarting of the linear iteration by default, although the user can disable this feature.

The GMRES algorithm generates a sequence of approximations $X(L)$ to the true solution of the above linear system. The convergence criteria for stopping the iteration is based on the size of the scaled norm of the residual $R(L) = B - A*X(L)$. The actual stopping test is either:

$$\text{norm}(SB*(B-A*X(L))) \leq TOL*\text{norm}(SB*B),$$

for right preconditioning, or

$$\text{norm}(SB*(M^{-1})*(B-A*X(L))) \leq TOL*\text{norm}(SB*(M^{-1})*B),$$

for left preconditioning, where $\text{norm}()$ denotes the euclidean norm, and $TOL$ is a positive scalar less than one input by the user. If $TOL$ equals zero when SGMRES is called, then a default value of $500*(\text{the smallest positive magnitude, machine epsilon})$ is used. If the scaling arrays $SB$ and $SX$ are used, then ideally they should be chosen so that the vectors $SX*X$ (or $SX*M*X$) and $SB*B$ have all their components approximately equal to one in magnitude. If one wants to use the same scaling in $X$ and $B$, then $SB$ and $SX$ can be the same array in the calling program.
7  Linking to PETSc

An example of linking to the PETSc libraries can be found in the Overture/tests directory. Type 'make tcm3p' to build the tcm3.c test code with PETSc. Type 'tcm3p cic.hdf -solver=petsc' to run the example on the grid cic.hdf with PETSc. This example assumes that the PETSC_DIR PETSC_ARCH and PETSC_LIB environmental variables have been defined per instructions with the PETSc installation.

Here is an explanation of the steps required to build an Overture application with PETSc (as implemented in the above example). By default, the Overture library is unaware whether PETSc solvers are available. To use PETSc you should

1. Build or locate a version of PETSc. I have only built and linked Overture to the non-parallel version of PETSc. Link to the PETSc libraries (and lapack). I link to

   
   petscLib = -L$(PETSC_LIB) -lpetscsles -lpetscdm -lpetscvec -lpetsc -L/usr/local/lib -llapack -L$(PETSC_LIB) -lmpiuni
   
   where $(PETSC_LIB) is the location of the PETSc libraries.

2. Copy the files Oges/buildEquationSolvers.C and Oges/PETScEquationSolver.C to your application directory and compile this file with the flags -DOVERTURE_USE_PETSC (or edit the file and define this variable inside with #define OVERTURE_USE_PETSC).

3. Link these new files, buildEquationSolvers.o and PETScEquationSolver.o with your application (ahead of the Overture library so that you get the new version) along with the PETSc libraries.

8  Adding a new sparse matrix solver to Oges

If you want to add a new sparse matrix solver to Oges you should look at one of the existing solvers, YaleEquationSolver, HarwellEquationSolver, SlapEquationSolver or PETScEquationSolver. These classes all derive from the base class EquationSolver. Oges contains a list of pointers to these EquationSolver's. You will be able to add a new solver to this list. It will be known as OgesParameters::userSolver1, or OgesParameters::userSolver2 etc., depending on how many new solvers have been added.

You should

1. Derive a new class from EquationSolver, copying one of the existing solvers (which ever is closest) to your new solver. Hopefully you can reuse parameters that already exist in OgesParameters.

2. Change the Oges/buildEquationSolvers.C file to ‘new’ the solver to have defined and add it to the list of EquationSolver's. Change the other functions in Oges/buildEquationSolvers.C as appropriate.

3. Compile your files and the new version of buildEquationSolvers.C and link to these ahead of the Overture library when you build an executable.

9  Some More Details about Oges

In general, Oges expects that the user wants to solve one or more equations at each valid grid point on an overlapping grid. The number of equations that are given at each grid point is called the numberOfComponents. In the simple case only one equation, such as a discrete Laplace operator, is specified at each grid point, numberOfComponents=1. In a more complicated case there will be a system of equations at each grid point. For example, one may want to solve the biharmonic equation as a system of two Possion equations in which case numberOfComponents=2.
Oges will create a large sparse matrix where each unknown for the sparse matrix will correspond to a particular component \( n \), at a particular grid point \((i_1,i_2,i_3)\) on a particular component grid, \( \text{grid} \). Thus there is a mapping from \((n,i_1,i_2,i_3,\text{grid})\) to a unique equation number. The member functions

\[
\text{int equationNo(int n, int i1, int i2, int i3, int grid)}
\]

\[
\text{intArray equationNo(int n, Index & I1, Index & I2, Index & I3, int grid)}
\]

give the equation number(s) for each grid point. For now the function \(\text{equationNo}\) is defined to use all the grid points in a given order. In the future, a user should be able to define this function in a different way. There is also a member function

\[
\text{void equationToIndex(int eqnNo, int n, int i1, int i2, int i3, int grid)}
\]

that maps an equation number, \(\text{eqnNo}\), back to a grid point and component, \((n,i_1,i_2,i_3,\text{grid})\) (i.e. it is the inverse of \(\text{equationNo}\)).

Sometimes extra unknowns and extra equations are required in order to specify a problem. For example, an eigenvalue problem has an extra unknown, the eigenvalue. An extra unknown may be added to the singular Neumann problem in order to create a nonsingular system. Extra unknowns are associated with grid points that are not used. The number of extra equations is specified with \(\text{setNumberOfExtraEquations}\). Oges will find unused points that can be used for extra equations; the equation numbers for these points will be saved in \(\text{extraEquationNumber(i)}\).

10 Oges Function Descriptions

10.1 default constructor

\(\text{Oges()}\)

\text{Description: } Default constructor.

10.2 setGridsToUse

\(\text{int setGridsToUse( const IntegerArray & gridsToUse )}\)

\text{Description: } Only solve the equations on some grids, these are called the active grids. If an active grid interpolates from an inactive grid, the corresponding interpolation equation will be replaced by a Dirichlet condition (i.e. the identity equation) and the solution at that point will left unchanged. (Note that RHS will be altered at this interpolation point and set equal to the solution value at that point.)

\text{gridsToUse (input) : a list of grids to use when solving. If this array is empty (i.e. a NULL array) then ALL grids will be used.}

10.3 activeGrid

\(\text{bool activeGrid( int grid ) const}\)

\text{Description: } Return true if this grid is used.

\text{grid (input) : grid to check}

\text{Return value (output): } true if this grid is active (used)
10.4  getUseThisGrid

const IntegerArray &
getUseThisGrid() const

Description: Return the array that indicates which grids are active, useThisGrid(grid)=true if the grid is active

Return value (output): a reference to useThisGrid.

10.5  getMaximumResidual

real
getMaximumResidual() const

Description: Return the maximum residual from the last solve.

10.6  get

int
get( const GenericDataBase & dir, const aString & name)

Description: Get a copy of Oges from a database file

dir (input): get from this directory of the database.

name (input): the name of Oges on the database.

10.7  put

int
put( GenericDataBase & dir, const aString & name) const

Description: Output an image of Oges to a database.

dir (input): put onto this directory of the database.

name (input): the name of Oges on the database.

10.8  setExtraEquationValues

int
setExtraEquationValues( realCompositeGridFunction & f, real *value )

Description: Assign values to the right-hand-side for the extra equations

f (input/output): fill in rhs values here

value[i] (input): values for each extra equation, i=0,1,2,....

Return values: 0=success

Author: wdh
10.9 setExtraEquationValues
int getExtraEquationValues( const realCompositeGridFunction & u, real *value )

Description: Return solution values from the extra equations

u(input) : grid function holding the solution.
value[i] (output) : values for each extra equation, i=0,1,2,...,

Return values: 0=success

Author: wdh

10.10 evaluateExtraEquation
int evaluateExtraEquation( const realCompositeGridFunction & u, real & value, int extraEquation =0)

Description: Evaluate the dot product of the coefficients of an extra equation times u

u (input) : grid function to dot with the extra equation
value (output) : the dot product
extraEquation (input) : the number of the extra equation (0,1,...,numberOfExtraEquations-1)

Return values: 0=success

Author: wdh

10.11 evaluateExtraEquation
int evaluateExtraEquation( const realCompositeGridFunction & u, real & value, real & sumOfExtraEquationCoefficients,
int extraEquation =0)

Description: Evaluate the dot product of the coefficients of an extra equation times u Also return the sum of the coefficients of the extra equation (i.e. the dot product with the ”1” vector)

u (input) : grid function to dot with the extra equation
value (output) : the dot product
sumOfExtraEquationCoefficients (output) : sum of the coefficients of the extra equation
extraEquation (input) : the number of the extra equation (0,1,...,numberOfExtraEquations-1)

Return values: 0=success

Author: wdh
10.12 writeMatrixToFile

int
writeMatrixToFile( aString filename )

Description: // Write the current solver matrix (using indicies with base 1) to the file ¡fileName¿. //
The file consists of triplets $i, j, A(i, j)$ (without commas) // for each non-zero element of the matrix.
// (Here $i$=row, $j$=column, and // $A(i, j) = A_{ij}$ element of the matrix.)

Author: pf, wdh

10.13 outputSparseMatrix

int
outputSparseMatrix( const aString & fileName )

Description: Output the matrix in compressed row format OR uncompressed format (with indices starting at 0). See the format below

fileName (input) : save the results to this file

10.14 writeMatrixGridInformationToFile

int
writeMatrixGridInformationToFile( aString filename )

Description: Write the grid information about the current solver matrix to the file ¡fileName¿.

For each equation in the matrix, a line
is saved in the file with the following format:

ieq  grid  simpleClassify  fullClassify

where:

ieq= equation number in the linear system
grid= grid number for this point

(In the classify flags, any non-negative value indicates a used point. Negative values are equations with zero for the rhs )

simpleClassify=
-1=connecting grids (=interpolation, extrapolation, or periodic bdry)
0=hole point (unused)
1=interior (=discretization) point
2=boundary point
   (=boundary, ghostline, periodic)

fullClassify=
interior= 1,
boundary= 2,
ghost1= 3,
Author: pf

10.15 writePetscMatrixToFile

```cpp
int writePetscMatrixToFile( aString filename,
    realCompositeGridFunction & u,
    realCompositeGridFunction & f)
```

**Description:** Only available when linked with PETSc (-DOVERTURE_USE_PETSC)

Write the current solver matrix to the file `filename`. Uses the PETSc binary format. Supply u,f as to 'solver', the RHS corresponding to f is also saved in the matrix file.

Author: pf

10.16 canSolveInPlace

```cpp
bool canSolveInPlace() const
```

**Description:** Return true if the rhs and sol vectors can be the same.

10.17 setCoefficientArray

```cpp
int setCoefficientArray( realCompositeGridFunction & coeff0,
    const IntegerArray & boundaryConditions =Overture::nullIntArray(),
    const RealArray & bcData =Overture::nullRealArray())
```

**Purpose:** Supply a coefficient grid function to be used to discretize the equations.

**coeff0 (input):** Here are the coefficients. Oges will keep a reference to this grid function.

**boundaryConditions(0: 1,0:2,numberOfComponentGrids) (input) :** optionally supply boundary conditions. These are needed by the multigrid solver.

**bcData** : data for the boundary conditions.

10.18 setCoefficientArray

```cpp
int setCoefficientArray( realMappedGridFunction & coeff0,
    const IntegerArray & boundaryConditions =Overture::nullIntArray(),
    const RealArray & bcData =Overture::nullRealArray())
```
Purpose: Supply a coefficient grid function (single grid only) to be used to discretize the equations.

coeff0 (input): Here are the coefficients. Oges will keep a reference to this grid function.

boundaryConditions(side, axis, grid) : optionally supply boundary conditions. These are needed by the multigrid solver.

bcData : data for the boundary conditions.

10.19 setEvaluateJacobian

void
defineEvaluateJacobian( const int evaluateJacobian0 )

Purpose: ?

10.20 setGrid

void
defineGrid( CompositeGrid & cg0, bool outOfDate = true)

Purpose: Supply a CompositeGrid to Oges. Use this routine, for example, if an Oges object was created with the default constructor. Call this routine before calling initialize.

cg0 (input): Oges will keep a reference to this grid.

outOfDate : set to true if the grid is out of date. This is normally true except in the case of using the multigrid solver in which case the multigrid hierarchy only needs to be built once so multiple instances of Oges need only mark the grid as out of date once. The multigrid hierarchy may also be marked out of date if you mark the MultigridCompositeGrid that was optionally supplied to Oges. You do this then you can call setGrid with outOfDate=false in all cases.

10.21 setGrid

void
defineGrid( MappedGrid & mg, bool outOfDate = true)

Purpose: Supply a MappedGrid to Oges. Use this routine, for example, if an Oges object was created with the default constructor. Call this routine before calling initialize.

mg (input): Oges will keep a reference to this grid.

10.22 set(OptionEnum, int)

int
define( OptionEnum option, int value = 0)

Description: Set an option from the OgesParameters::OptionEnum enumerator. See section [5] for a full description of the options available.

option (input) : choose an option

value (input) : value to assign (for options requiring a value).
10.23  set(OptionEnum,float)

int
set(  OptionEnum option, float value )

Description: Set an option from the OgesParameters::OptionEnum enumerator. See section [5] for a full description of the options available.

option (input) : choose an option
value (input) : value to assign (for options requiring a value).

10.24  set(OptionEnum,double)

int
set(  OptionEnum option, double value )

Description: Set an option from the OgesParameters::OptionEnum enumerator. See section [5] for a full description of the options available.

option (input) : choose an option
value (input) : value to assign (for options requiring a value).

10.25  set(SolverEnum)

int
set(  SolverEnum option )

Description: Select a solver from the OgesParameters::SolverEnum enumerator. See section [5] for a full description of the options available.

option (input) : option selected.

10.26  set(SolverMethodEnum)

int
set(  SolverMethodEnum option )

Description: Select a solver method from the OgesParameters::SolverMethodEnum enumerator. See section [5] for a full description of the options available.

option (input) : option selected.

10.27  set(PreconditionerEnum)

int
set(  PreconditionerEnum option )

Description: Select a preconditioner from the OgesParameters::PreconditionerEnum enumerator. See section [5] for a full description of the options available.

option (input) : option selected.
10.28  set(MatrixOrderingEnum)

int
set( MatrixOrderingEnum option )

**Description:** Select a matrix ordering from the OgesParameters::MatrixOrderingEnum enumerator. See section (5) for a full description of the options available.

`option` (input): option selected.

10.29  get(OptionEnum,int&)  

int
get( OptionEnum option, int & value ) const

**Description:** Return the current value of an option (this version appropriate for options that have a value of type ‘int’. See section (5) for a full description of the options available.

10.30  get(OptionEnum,real&)

int
g et ( OptionEnum option, real & value ) const

**Description:** Return the current value of an option (this version appropriate for options that have a value of type ‘real’. See section (5) for a full description of the options available.

10.31  setOgesParameters

int
setOgesParameters( const OgesParameters & par )

**Description:** Assign the values from an OgesParameters object to an Oges object.

10.32  sizeOf

real
sizeOf( FILE *file =NULL) const

**Description:** Return number of bytes allocated by Oges; optionally print detailed info to a file

`file` (input): optionally supply a file to write detailed info to. Choose file=stdout to write to standard output.

Return value: the number of bytes.

10.33  printStatistics

int
printStatistics(FILE *file = stdout) const

**Description:** Output any relevant statistics
10.34 updateToMatchGrid

```c
int updateToMatchGrid( CompositeGrid & cg0 )
```

**Purpose:** Give Oges a new matrix to use. Use this routine, for example, when a grid has moved. This routine will cause the matrix to be refactored the next time solve is called.

**cg0 (input):** use this CompositeGrid

10.35 updateToMatchGrid

```c
int updateToMatchGrid( MappedGrid & mg )
```

**Purpose:** Use this version when you are solving a problem on a MappedGrid.

**mg (input):** use this MappedGrid

10.36 getMatrix

```c
int getMatrix( IntegerArray & ia_, IntegerArray & ja_, RealArray & a_,
              SparseStorageFormatEnum format = compressedRow)
```

**Description:** Return the matrix in a given format.

**ia_, ja_, a_ (output):** reference to the matrix in sparse form.

**format (input):** sparse format

**References**


Index

convergence criteria, 15

grid function
  coefficient matrix, 5

Harwell, 4

OgesParameters, 9

PETSc, 4
  linking to, 17

results
  parallel, 6

SLAP, 4
sparse matrix solver, 17

Yale, 4