

PyAMG

Algebraic Multigrid Solvers in Python

PyAMG

Algebraic Multigrid Solvers in Python

Nathan Bell, Nvidia

Luke Olson, University of Illinois

Jacob Schroder, Lawrence Livermore National Lab

Boot disc

- ✧ For Mac
 - ✧ Press and hold `c` immediately after reboot
- ✧ For the usual suspects (Dell, HP, Thinkpad, etc...)
 - ✧ Some automatically detect a bootable DVD
 - ✧ Pressing `F12`, `F8`, or `F6` at boot window can list boot options
 - ✧ Last, an advanced approach changes BIOS boot order
 - ✧ Press `F1`, `delete`, or `esc` during initial boot window
 - ✧ From the BIOS, change boot order to start with DVD drive first
- ✧ After booting, open a terminal (Applications—>Accessories—>Terminal)

Task 0.1: Installing PyAMG

Compile PyAMG (if using boot disc)

```
$ tar -xvf pyamg2.0.4.tar.gz  
$ cd pyamg/  
$ sudo python setup.py install  
$ cd Examples/WorkshopCopper12  
$ ipython
```

Test PyAMG during interactive iPython session, enter:

```
import pyamg  
pyamg.test()
```

What is PyAMG

- ❖ Algebraic multigrid (AMG) workbench
 - ❖ Readable and reproducible AMG
 - ❖ Efficient serial solver
- ❖ Python-based
 - ❖ Readability and useability
 - ❖ C++ backend for speed
- ❖ BLAS, LAPACK, optimized routines, etc...

Goal 1: Ease-of-use

- ❖ Accessible interface to non-experts
- ❖ Extensive documentation and references
- ❖ Portability through modular multiplatform Python libraries
- ❖ Rapid prototyping of new techniques
 - ❖ Organize source code into intuitive reusable components
 - ❖ High-level Python allows for rapid swapping of components

Goal 2: Speed

- ❖ Solve millions of unknowns on laptop / desktop
- ❖ Use hybrid coding strategy (Python as glue)
 - ❖ Performance sensitive portions are small fraction of code
 - ❖ 80% Python / 20% natively compiled C/C++/Fortran
- ❖ Example:
 - ❖ High-level multigrid cycling in Python
 - ❖ Calls `gauss_seidel(A,x,b,iterations=1)`
 - ❖ All computation done in C++ routine

PyAMG features

- ❖ Ruge-Stüben AMG
- ❖ Smoothed aggregation (standard and adaptive)
- ❖ Native complex support
- ❖ Nonsymmetric matrices
- ❖ Krylov solvers (CG, BiCGStab, GMRES, fGMRES, CGNR, CR)
- ❖ Compatible relaxation (experimental)
- ❖ Relaxation methods (GS, wJ, SOR, Kacz, Cheby, Schwarz)
- ❖ Visualizations (Paraview, Matplotlib)

Dependencies

PyAMG

Multilevel solvers, relaxation methods, Krylov methods

Scipy

*LAPACK and
sparse matrix
operations*

Numpy

*Array operations
(BLAS)*

C++/Swig

*Easy interface from
Python to C++*

Nose

*Unit tests, i.e.,
does everything
work?*

Matplotlib

Visualizations

iPython

*Python interpreter,
interactive sessions*

What PyAMG does not do...

- ❖ Solve everything in linear time
- ❖ Multicore (coming)
- ❖ GPU acceleration (Cusp)
- ❖ Large-scale parallel simulations
 - ❖ See Hypre (Livermore) and ML (Sandia)

General Structure

- ❖ `multilevel.py` (multilevel solver class)
- ❖ Main structure for hierarchy (SA or RS)
- ❖ Handles cycling and coarse solver
- ❖ Contains list of level object instances
- ❖ Each level object instance contains:
 - ❖ Matrices: A , P , R
 - ❖ Functions: `presmoothen`, `postsmoothen`

Starting from `pyamg/pyamg`

- * `multilevel.py` Multilevel solver class
- * `strength.py` Strength-of-connection routines
- * `classical/` Ruge-Stüben construction routines
- * `aggregation/` SA construction routines
- * `krylov/` Krylov solvers, e.g., CG, GMRES, fGMRES
- * `relaxation/` Relaxation methods, e.g., GS, wJ, Cheby
- * `Graph/` Graph algorithms for coarsening, e.g., MIS
- * `vis/` Visualizations in Matplotlib and Paraview
- * `amg_core/` C++ functions called through Swig
- * `gallery/` Construct model problems
- * `util/` Utility functions, e.g., spectral radius

Task 1.1: Getting used to Python

- ❖ Accessing documentation inside of iPython
 - ❖ Use <tab> on object to see its members
 - ❖ Use ? on object or function for documentation
 - ❖ Use spacebar to page, and q to quit documentation screen
- ❖ Inside of iPython (\$ ipython), enter:

*The most useful things
you'll learn today*

```
from pyamg import gallery, smoothed_aggregation_solver
```

```
A = gallery.poisson( (50,50), format='csr')
```

```
ml = smoothed_aggregation_solver(A)
```

```
ml.<tab>
```

```
ml.__class__
```

```
ml.__doc__
```

```
ml.__init__
```

```
ml.__module__
```

```
ml.__repr__
```

```
ml._multilevel_solver__solve
```

```
ml.aspreconditioner
```

```
ml.coarse_solver
```

```
ml.cycle_complexity
```

```
ml.grid_complexity
```

```
ml.level
```

```
ml.levels
```

```
ml.operator_complexity
```

```
ml.psolve
```

```
ml.solve
```

```
ml.solve?
```

```
gallery.poisson?
```


Task 1.2: Sparse Matrices

- ❖ Construct a sparse matrix
- ❖ Inside of iPython, enter:

*If you ever get lost, exit iPython
(`ctrl-d`, `ctrl-d`), re-enter iPython,
and type `run taskx.x.py`*

```
from scipy.sparse import *  
csr_matrix?  
from numpy import array  
row = array([0,0,1,2,2,2])  
col = array([0,2,2,0,1,2])  
data = array([1,2,3,4,5,6])  
B = csr_matrix( (data,(row,col)), shape=(3,3) )  
B.<tab>  
print(B.todense())  
    [[1 0 2]  
     [0 0 3]  
     [4 5 6]]  
B = B.tocoo()
```

Task 1.3: Using the gallery

- ❖ Generate a sparse matrix by running script
- ❖ Inside of iPython, enter:

```
run task1.3

print(A[5050,:].data)
[-0.22 -0.25  0.22 -0.75  2.   -0.75  0.22 -0.25 -0.22]

print(sten)
[[-0.22 -0.25  0.22]
 [-0.75  2.   -0.75]
 [ 0.22 -0.25 -0.22]]
```

Script: task1.3.py

```
from pyamg.gallery.diffusion import diffusion_stencil_2d
from pyamg.gallery import stencil_grid
from numpy import set_printoptions
set_printoptions(precision=2)
sten = diffusion_stencil_2d(type='FD', \
    epsilon=0.001, theta=3.1416/3.0)
A = stencil_grid(sten, (100,100), format='csr')
```


Task 1.4: Building a MG hierarchy

- ✧ Inside of iPython, enter:

```
from pyamg import *
ml = smoothed_aggregation_solver(A)
print(ml)
    multilevel_solver
    Number of Levels:      3
    Operator Complexity:   1.126
    Grid Complexity:       1.130
    Coarse Solver:         'pinv2'
      level  unknowns      nonzeros
        0      10000      88804 [88.84%]
        1       1156      10000 [10.00%]
        2        144       1156 [ 1.16%]

print(ml.levels[0].A.shape)
(10000, 10000)

print(ml.levels[0].P.shape)
(10000, 1156)

print(ml.levels[0].R.shape)
(1156, 10000)
```

Task 1.5: Solving a problem

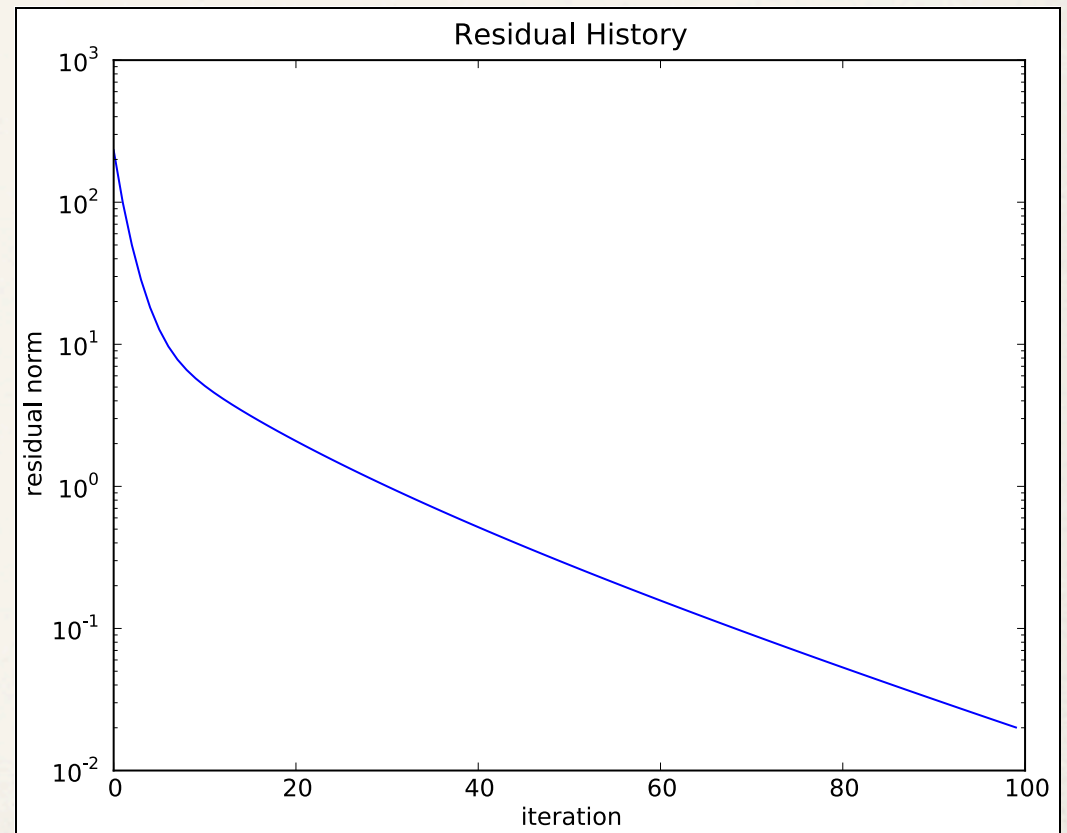
- ❖ Inside of iPython, enter:

```
run task1.5
```

Script: task1.5.py

```
from numpy import ones
b = ones((A.shape[0],1))
res = []
x = ml.solve(b, tol=1e-8, \
             residuals=res)

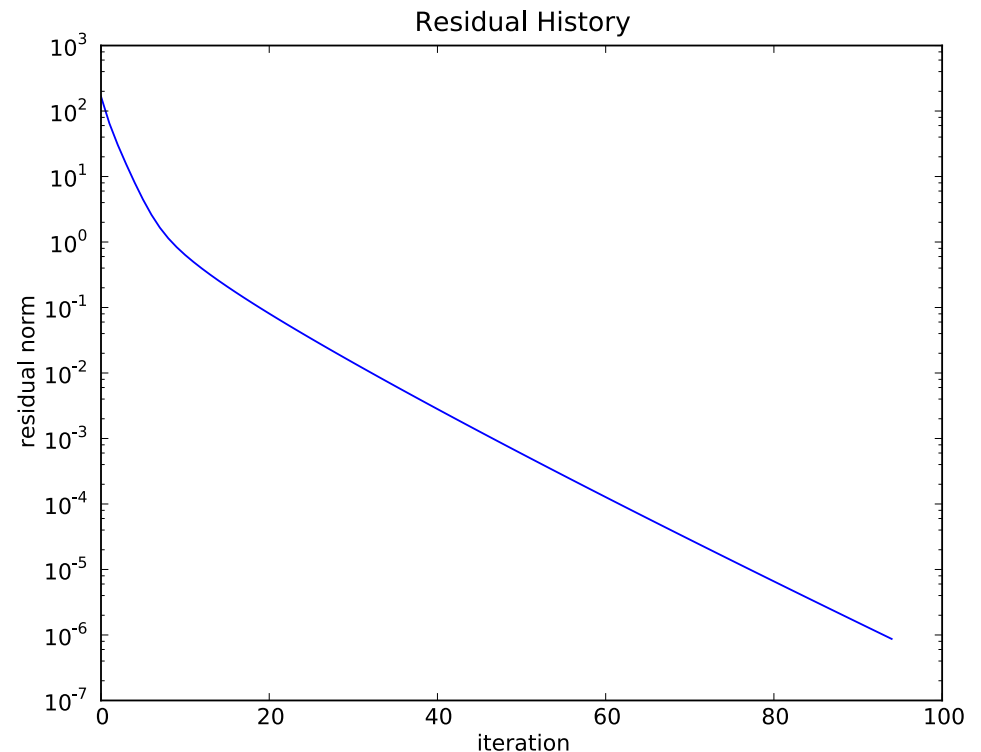
from pylab import *
semilogy(res[1:])
xlabel('iteration')
ylabel('residual norm')
title('Residual History')
show()
```



Task 1.6: Changing MG options

- ❖ Use advanced coarsening and prolongation smoothing options
- ❖ Inside of iPython, enter:

```
from pyamg import *
from numpy import ones
ml = smoothed_aggregation_solver(A, \
    strength='evolution', \
    smooth=('energy', {'degree':4}) )
b = ones((A.shape[0],1))
res = []
x = ml.solve(b, tol=1e-8, residuals=res)
from pylab import *
semilogy(res[1:])
xlabel('iteration')
ylabel('residual norm')
title('Residual History')
show()
```



Intermediate Tasks

- ❖ Modify existing multilevel hierarchy
- ❖ Add new prolongation smoothing function to PyAMG source
- ❖ Visualizations with Paraview
- ❖ Loading matrix from file
- ❖ Blackbox solve

Task 2.1: Modifying the hierarchy

- ❖ Modify existing multilevel solver object
- ❖ Replace existing pre/post-smoothers with new user-provided routine
- ❖ Execute commands in shell:

```
ctrl-d, ctrl-d (exit iPython)
```

```
$ python task2.1.py
```

Set new pre/post-smoother {

```
def new_relax(A,x,b):  
    x[:] += 0.125*(b - A*x)  
  
A = gallery.poisson( (100,100), format='csr')  
b = ones( (A.shape[0],1))  
res = []  
ml = smoothed_aggregation_solver(A)  
ml.levels[0].presmoothing = new_relax  
ml.levels[0].postsmoothing = new_relax  
x = ml.solve(b, tol=1e-8, residuals=res)  
  
semilogy(res[1:])  
show()
```


Task 2.2: Adding a smoother

\$ gedit ~/pyamg/pyamg/aggregation/aggregation.py
At line 409, insert two new lines:

New
Lines {

```
if fn == 'jacobi':  
    P = jacobi_prolongation_smoother(A, T, C, B, **kwargs)  
elif fn == 'simple':  
    P = T - 0.2*A*T  
elif fn == 'richardson':  
    ...
```

```
$ cd ~/pyamg/  
$ sudo python setup.py install  
$ cd Examples/WorkshopCopper12  
$ python task2.2.py
```

```
A = gallery.poisson( (100,100), format='csr')  
ml = smoothed_aggregation_solver(A, smooth='simple')  
...
```

Task 2.3: Plotting aggregates

❖ Visualization with Paraview

```
$ python task2.3.py
```

```
$ paraview&
```

load data

```
{ data = load_example('unit_square')  
  A = data['A'].tocsr()  
  V = data['vertices']  
  E2V = data['elements']
```

create hierarchy

```
{ ml = smoothed_aggregation_solver(A, keep=True, max_coarse=10)  
  b = sin(pi*V[:,0])*sin(pi*V[:,1])  
  x = ml.solve(b)
```

save aggregates

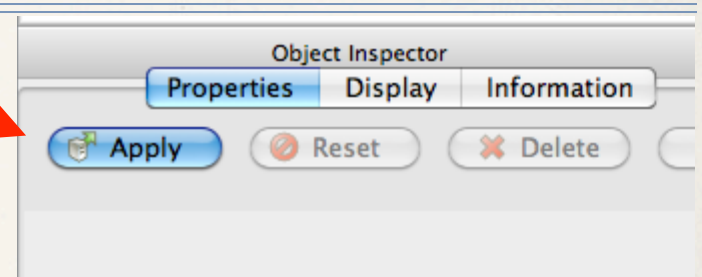
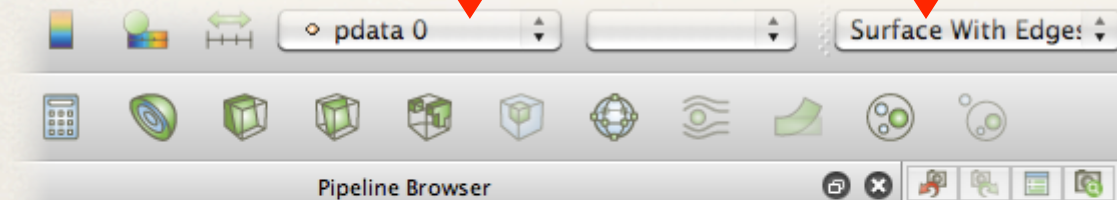
```
{ vis_coarse.vis_aggregate_groups(Verts=V, E2V=E2V,  
  Agg=ml.levels[0].AggOp, mesh_type='tri',  
  output='vtk', fname='output_aggs.vtu')
```

save mesh

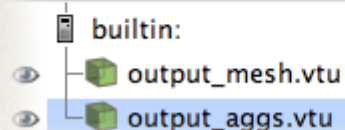
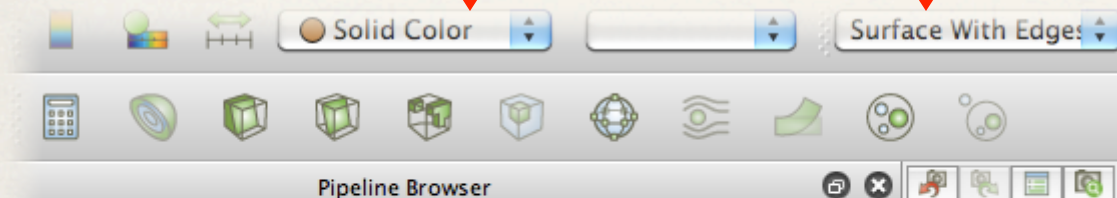
```
{ vtk_writer.write_basic_mesh(Verts=V, E2V=E2V,  
  pdata = x,  
  mesh_type='tri',  
  fname='output_mesh.vtu')
```

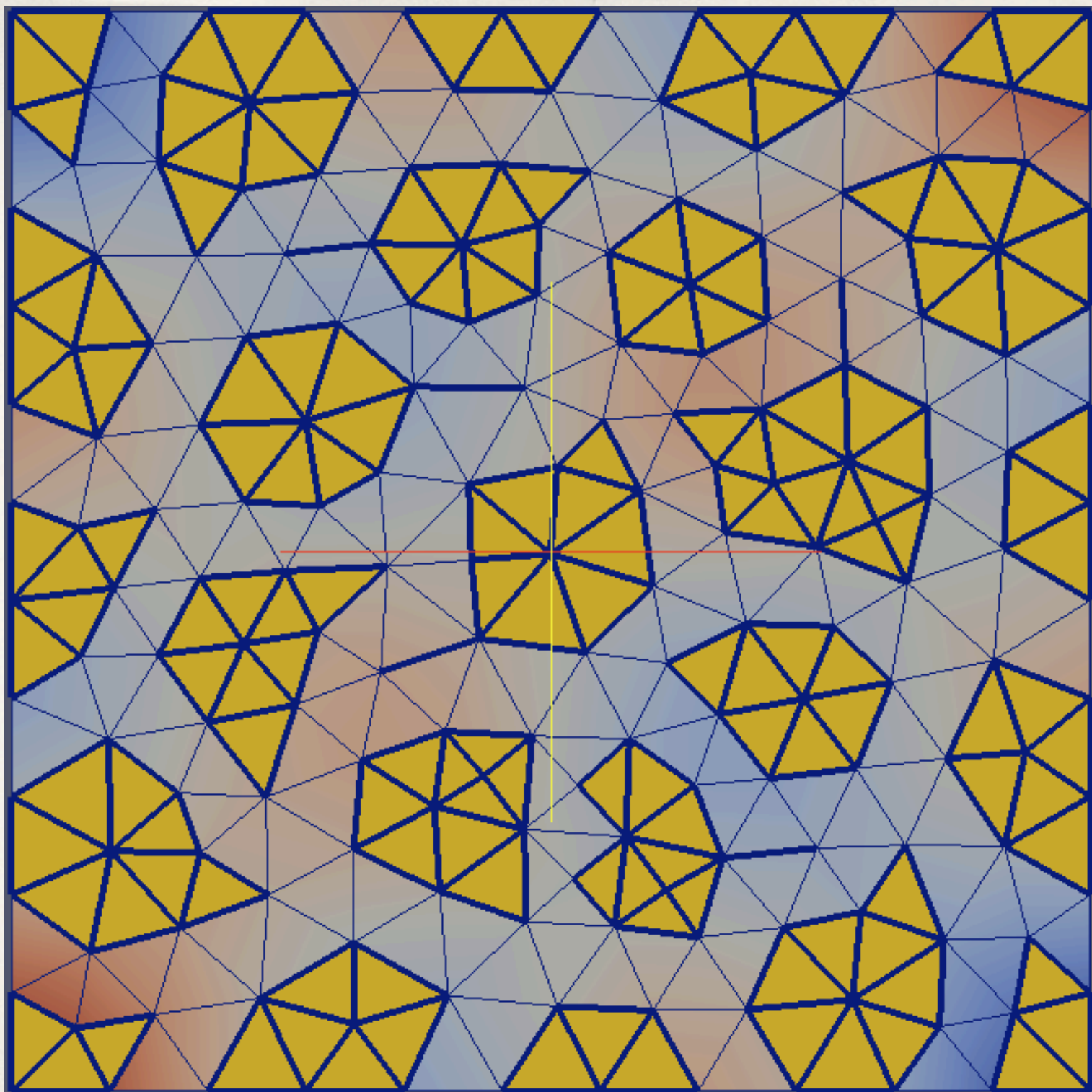
Task 2.3: Plot mesh and aggregates

- ❖ Open `output_mesh.vtu`, then click apply
- ❖ Select options



- ❖ Open `output_aggs.vtu`, then click apply
- ❖ Select options





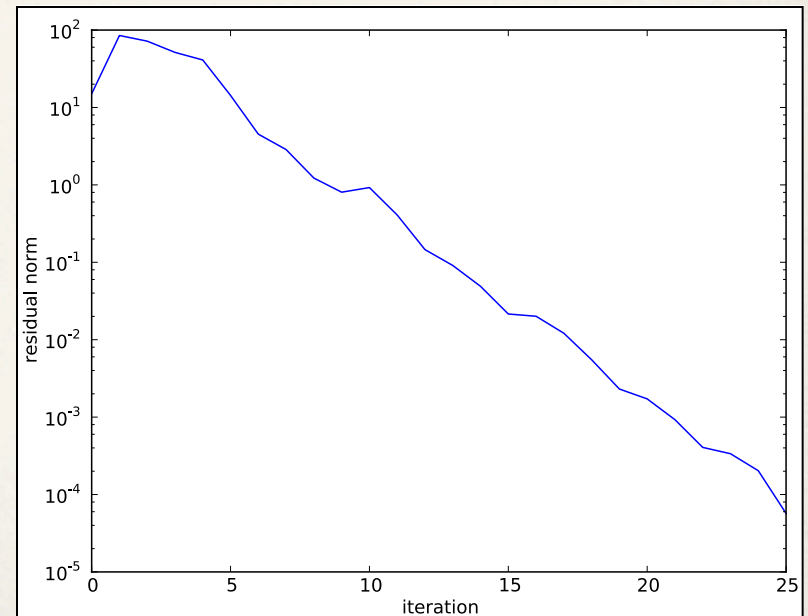
Task 2.4: Loading a matrix

nonsymmetric

```
$ python task2.4.py
```

```
data = loadmat('../..//pyamg/gallery/example_data/recirc_flow.mat')
A = data['A']

from pyamg import *
ml = smoothed_aggregation_solver(A, symmetry='nonsymmetric', max_coarse=5)
```



Try it with...

```
ml = smoothed_aggregation_solver(A, symmetry='symmetric', max_coarse=5)
```

Task 2.5: Running blackbox solve

Inside of iPython, enter:

- ❖ “blackbox” solve attempts to pick the most robust options



- ❖ solve once



- ❖ solve again (same hierarchy)



```
from scipy import rand
from numpy import arange, array
from pyamg import solve
from pyamg.gallery import poisson
from pyamg.util.linalg import norm

# Run solve(...) with the verbose option
n = 100
A = poisson((n,n),format='csr')
b = array(arange(A.shape[0]))
x = solve(A,b,verb=True)

# Return the solver for re-use
(x,ml) = solve(A, b, verb=True,
               return_solver=True, tol=1e-8)

# Run for a new right-hand-side
b2 = rand(b.shape[0],)
x2 = solve(A, b2, verb=True,
           existing_solver=ml, tol=1e-8)
```


Advanced Tasks

- ❖ SWIG

- ❖ SWIG interfaces between C++ and Python
- ❖ Replace slow Python segments with C++
- ❖ This task compares pure Python and hybrid Python/C++ versions of forward and backward substitution

- ❖ Important files for example

- ❖ `numpy.i` interface between NumPy and C++ (esp. arrays)
- ❖ `complex_ops.h` interface for complex data types
- ❖ `splinalg.i` IN, INPLACE and OUT types, templating
- ❖ `splinalg.h` headers, function definitions (plan vanilla C++)

```
$ cd ~/pyamg/Examples/SWIG
```

Task 3.1: Calling C++

splinalg.h defines the C++:

```
template<class I, class T>
void forwardsolve(const I Ap[], const I Aj[], const T Ax[],
                  T x[], const T b[], const I n)
{...
}
```

splinalg.i defines the C++ interface for SWIG:

```
/* INPLACE types */
#define T_INPLACE_ARRAY1( ctype )
%apply ctype * INPLACE_ARRAY {
    ctype    x [ ]
};
```

SWIG compiles and creates the python interface:

```
$ swig -c++ -python splinalg.i
```

```
splinalg.forwardsolve(L.indptr,L.indices,L.data,x,b,n)
```

Task 3.1: Calling C++

To compile example:

```
ctrl-d, ctrl-d (exit iPython)
$ cd ~/pyamg/Examples/SWIG
$ sudo python setup.py install
```

Run example calling C++ routines with SWIG:

```
$ python testbasic.py
```


Task 3.1: Calling C++

- ❖ C++ call:

time for one LU solve = 0.1998 ms

- ❖ \$ gedit precondition.py

- ❖ change line 74 to

{

```
def preconditioner_matvec(L,U):  
    def matvec(x):  
        return lusolve_reference(L,U,x)
```

- ❖ \$ python testbasic.py

time for one LU solve = 34.15 ms

1 or 2 magnitude difference

PYAMG

Algebraic Multigrid Solvers in Python

<http://www.pyamg.org>

Nathan Bell, Nvidia
Luke Olson, University of Illinois
Jacob Schroder, Lawrence Livermore National Lab

Thanks To NSF