Multigrid solvers

M. M. Sussman
sussmanm@math.pitt.edu
Office Hours: 11:10AM-12:10PM, Thack 622

May 12 – June 19, 2014
Multigrid

Geometrical multigrid
  Introduction
  Details of GMG
  Summary

Algebraic multigrid
  Introduction
  Grid coarsening and interpolation
Multigrid

Geometrical multigrid

Introduction
Details of GMG
Summary

Algebraic multigrid

Introduction
Grid coarsening and interpolation
Geometrical multigrid

- Simple iterative methods tend to damp high (spatial) frequency errors fast.
- After a few smoothing steps of a simple method, map the current error out to a coarser grid.
- Errors will have relatively higher spatial frequency there.
- Take a few more steps of a simple method on the coarser grid.
- Continue mapping to coarser grids until grid is coarse enough to solve.
- Interpolate back to the next finer grid and do few smoothing steps
- Continue to the finest grid
- Repeat until converged.
Advantages of GMG

- Number of iterations should not depend on number of mesh points!
- Works very well as preconditioner for Krylov methods
Gauss-Seidel iterations

To solve an $n \times n$ matrix system,

$$Au = f$$

given an initial guess $u^{(0)}$, for $k = 1, 2, \ldots$, set

$$u_i^{(k+1)} = \left( f_i - \sum_{j=1}^{i-1} A_{ij} u_j^{(k+1)} - \sum_{j=i+1}^{n} A_{ij} u_j^{(k)} \right) / A_{ii}$$
Gauss-Seidel starts fast, slows down
Error gets smooth fast
Error gets smooth fast
Multiple 1D grids
Interpolation or prolongation

If a solution is known on a grid, how should it be transferred to the next finer grid?

- For fine grid points that agree with coarse points, copy.
- For fine grid points between two coarse points, average.
Interpolation matrix 5 pts to 9 pts

\[ P_{9 \times 5} = \begin{pmatrix}
1 & .5 & .5 \\
.5 & .5 & 1 \\
.5 & 1 & \frac{5}{2} \\
.5 & \frac{5}{2} & 1 \\
.5 & 1 & 1
\end{pmatrix} \]
Restriction

If a solution is known on a fine grid, how should it be transferred to the next \textit{coarser} grid?

\[ P_{5 \times 9} = (P_{9 \times 5})^T \]

- Maintain symmetry!
- Proofs fail without it!
- It works better this way.
Multigrid

Geometrical multigrid
- Introduction
- Details of GMG
- Summary

Algebraic multigrid
- Introduction
- Grid coarsening and interpolation
The V-cycle

Smooth

Smooth

Smooth

Smooth

Smooth

Solve
The V-cycle: Python code

def vcycle(A,f):
    # perform one v-cycle on the matrix A

    sizeF = np.size(A,axis=0);

    # N1=number of Gauss-Seidel iterations before coarsening
    N1 = 5;
    v = np.zeros(sizeF);
    for numGS in range(N1):
        for k in range(sizeF):
            v[k] = (f[k] - np.dot(A[k,0:k], v[0:k])
                -np.dot(A[k,k+1:], v[k+1:]) ) / A[k,k];

    # construct interpolation operator from next coarser to this mesh
    # next coarser has ((n-1)/2 + 1 ) points
    assert(sizeF%2 ==1)
    sizeC = (sizeF-1)/2 +1
    P = np.zeros((sizeF,sizeC));
    for k in range(sizeC):
        P[2*k,k] = 1; # copy these points
    for k in range(sizeC-1):
        P[2*k+1,k] = .5; # average these points
        P[2*k+1,k+1] = .5;
The V-cycle: Python code

def vcycle(A,f):
    # perform one v-cycle on the matrix A
    sizeF = np.size(A,axis=0);

    # size for direct inversion < 15
    if sizeF < 15:
        v = la.solve(A,f)
        return v

    N1=number of Gauss-Seidel iterations before coarsening

    v = np.zeros(sizeF);
    for numGS in range(N1):
        for k in range(sizeF):
            v[k] = (f[k] - np.dot(A[k,0:k], v[0:k])
                    -np.dot(A[k,k+1:], v[k+1:])) / A[k,k];

    # construct interpolation operator from next coarser to this mesh
    # next coarser has ((n-1)/2 + 1 ) points
    assert(sizeF%2 ==1)
    sizeC = (sizeF-1)/2 +1
    P = np.zeros((sizeF,sizeC));
    for k in range(sizeC):
        P[2*k,k] = 1; # copy these points
    for k in range(sizeC-1):
        P[2*k+1,k] = 0.5; # average these points
        P[2*k+1,k+1] = 0.5;
The V-cycle: Python code

def vcycle(A, f):
    # perform one v-cycle on the matrix A

    sizeF = np.size(A, axis=0);

    # size for direct inversion < 15
    if sizeF < 15:
        v = la.solve(A, f)
        return v

    # N1=number of Gauss-Seidel iterations before coarsening
    N1 = 5;
    v = np.zeros(sizeF);
    for numGS in range(N1):
        for k in range(sizeF):
            v[k] = (f[k] - np.dot(A[k, 0:k], v[0:k]) -
                    np.dot(A[k, k+1:], v[k+1:])) / A[k, k];
def vcycle(A,f):
    # perform one v-cycle on the matrix A

    sizeF = np.size(A,axis=0);

    # size for direct inversion < 15
    if sizeF < 15:
        v = la.solve(A,f)
        return v

    # N1=number of Gauss-Seidel iterations before coarsening
    N1 = 5;
    v = np.zeros(sizeF);
    for numGS in range(N1):
        for k in range(sizeF):
            v[k] = (f[k] - np.dot(A[k,0:k], v[0:k]) -
                    np.dot(A[k,k+1:], v[k+1:])) / A[k,k];

    # construct interpolation operator from next coarser to this mesh
    # next coarser has ((n-1)/2 + 1 ) points
    assert(sizeF%2 ==1)
    sizeC = (sizeF-1)/2 +1
    P = np.zeros((sizeF,sizeC));
    for k in range(sizeC):
        P[2*k,k] = 1;  # copy these points
    for k in range(sizeC-1):
        P[2*k+1,k] = .5;  # average these points
        P[2*k+1,k+1] = .5;
The V-cycle: Python code cont’d

```python
# compute residual
residual = f - np.dot(A,v)
```

# compute residual
residual = f - np.dot(A,v)

# project residual onto coarser mesh
residC = np.dot(P.transpose(),residual)
# compute residual
residual = f - np.dot(A,v)

# project residual onto coarser mesh
residC = np.dot(P.transpose(),residual)

# Find coarser matrix  (sizeC X sizeC)
AC = np.dot(P.transpose(),np.dot(A,P))

vC = vcycle(AC,residC);

# extend to this mesh
v = np.dot(P,vC)

# N2=number of Gauss-Seidel iterations after coarsening
N2 = 5;
for numGS in range(N2):
    for k in range(sizeF):
        v[k] = (f[k] - np.dot(A[k,0:k], v[0:k]) \
                -np.dot(A[k,k+1:], v[k+1:]) ) / A[k,k];
return v
# compute residual
residual = f - np.dot(A,v)

# project residual onto coarser mesh
residC = np.dot(P.transpose(),residual)

# Find coarser matrix (sizeC X sizeC)
AC = np.dot(P.transpose(),np.dot(A,P))

vC = vcycle(AC,residC);
The V-cycle: Python code cont’d

```python
# compute residual
residual = f - np.dot(A,v)

# project residual onto coarser mesh
residC = np.dot(P.transpose(),residual)

# Find coarser matrix (sizeC X sizeC)
AC = np.dot(P.transpose(),np.dot(A,P))

vC = vcycle(AC,residC);

# extend to this mesh
v = np.dot(P,vC)
```
# compute residual
residual = f - np.dot(A,v)

# project residual onto coarser mesh
residC = np.dot(P.transpose(),residual)

# Find coarser matrix  (sizeC X sizeC)
AC = np.dot(P.transpose(),np.dot(A,P))

vC = vcycle(AC,residC);

# extend to this mesh
v = np.dot(P,vC)

# N2=number of Gauss-Seidel iterations after coarsening
N2 = 5;
for numGS in range(N2):
    for k in range(sizeF):
        v[k] = (f[k] - np.dot(A[k,0:k], v[0:k]) \n             -np.dot(A[k,k+1:], v[k+1:])) / A[k,k];
The V-cycle: Python code cont’d

```python
# compute residual
residual = f - np.dot(A,v)

# project residual onto coarser mesh
residC = np.dot(P.transpose(),residual)

# Find coarser matrix (sizeC X sizeC)
AC = np.dot(P.transpose(),np.dot(A,P))

vC = vcycle(AC,residC);

# extend to this mesh
v = np.dot(P,vC)

# N2=number of Gauss-Seidel iterations after coarsening
N2 = 5;
for numGS in range(N2):
    for k in range(sizeF):
        v[k] = (f[k] - np.dot(A[k,0:k], v[0:k]) - np.dot(A[k,k+1:], v[k+1:])) / A[k,k];
return v
```
Solving with V-cycles `gmgsolve.py`

```
N = 2**9+1
x = np.linspace(0,1,N);
h = x[1]-x[0]
```

```python
# tridiagonal matrix
A = np.diag(2.*np.ones(N)) - np.diag(np.ones(N-1), 1) - np.diag(np.ones(N-1), -1)
A = A/h**2
f = np.ones(N, dtype=float) # rhs
udirect = la.solve(A, f) # correct solution
u = np.zeros(N) # initial guess
for iters in range(100):
    r = f - np.dot(A,u)
    if la.norm(r)/la.norm(f) < 1.e-10:
        break
    du = vcycle(A, r)
    u += du
print "step %d, rel error=%e"% (iters+1, la.norm(u-udirect)/la.norm(udirect) )
```
Solving with V-cycles `gmgsolve.py`

```python
N = 2**9+1
x = np.linspace(0,1,N);
h = x[1]-x[0]

# tridiagonal matrix
A = np.diag(2.*np.ones(N)) - np.diag(np.ones(N-1), 1) - np.diag(np.ones(N-1), -1)
A = A/h**2

f = np.ones(N, dtype=float)  # rhs
udirect = la.solve(A, f)  # correct solution
u = np.zeros(N)  # initial guess
for iters in range(100):
    r = f - np.dot(A,u)
    if la.norm(r)/la.norm(f) < 1.e-10:
        break
    du = vcycle(A, r)
    u += du
    print "step %d, rel error=%e"% (iters+1, la.norm(u-udirect)/la.norm(udirect) )
```

```
Solving with V-cycles `gmg.solve.py`

```python
N = 2**9+1
x = np.linspace(0,1,N);
h = x[1]-x[0]

# tridiagonal matrix
A = np.diag(2.*np.ones(N)) - np.diag(np.ones(N-1), 1) - np.diag(np.ones(N-1), -1)
A = A/h**2

f = np.ones(N, dtype=float) # rhs
udirect = la.solve(A, f) # correct solution
```
N = 2**9+1
x = np.linspace(0,1,N);
h = x[1]-x[0]

# tridiagonal matrix
A = np.diag(2.*np.ones(N)) - np.diag(np.ones(N-1), 1) - np.diag(np.ones(N-1), -1)
A = A/h**2

f = np.ones(N, dtype=float) #rhs
udirect = la.solve(A, f) # correct solution

u = np.zeros(N) # initial guess
for iters in range(100):
r = f - np.dot(A,u)
if la.norm(r)/la.norm(f) < 1.e-10:
    break
du = vcycle(A, r)
u += du

print "step %d, rel error=%e"% 
    (iters+1, la.norm(u-udirect)/la.norm(udirect) )
Iterations and problem size

Number of iterations is independent of problem size!

<table>
<thead>
<tr>
<th>Grid size</th>
<th>Number of iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>20</td>
</tr>
<tr>
<td>65</td>
<td>21</td>
</tr>
<tr>
<td>129</td>
<td>22</td>
</tr>
<tr>
<td>257</td>
<td>22</td>
</tr>
<tr>
<td>513</td>
<td>22</td>
</tr>
<tr>
<td>1025</td>
<td>22</td>
</tr>
<tr>
<td>2049</td>
<td>22</td>
</tr>
</tbody>
</table>
Multigrid

Geometrical multigrid
  Introduction
  Details of GMG
  Summary

Algebraic multigrid
  Introduction
  Grid coarsening and interpolation
What is needed for MG?

1. Sequence of grids
2. Intergrid transfer operators
3. Smoothing operator
4. Solver for coarsest grid
Multigrid

Geometrical multigrid
  Introduction
  Details of GMG
  Summary

Algebraic multigrid
  Introduction
  Grid coarsening and interpolation
Multigrid

Geometrical multigrid
  Introduction
  Details of GMG
  Summary

Algebraic multigrid
  Introduction
  Grid coarsening and interpolation
References


What is a “grid”?

- Every matrix has an associated graph

\[
\begin{pmatrix}
1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
\end{pmatrix}
\]

- Given a matrix, the finest grid is its associated graph.
Multigrid

Geometrical multigrid
- Introduction
- Details of GMG
- Summary

Algebraic multigrid
- Introduction
  - Grid coarsening and interpolation
What does “smooth” mean?

- In GMG, we have a notion of “smooth” error and notice that Gauss-Seidel iteration makes rough errors smoother.
- Gauss-Seidel makes very rough errors smooth rapidly, then stalls.
What does “smooth” mean?

- In GMG, we have a notion of “smooth” error and notice that Gauss-Seidel iteration makes rough errors smoother.
- Gauss-Seidel makes very rough errors smooth rapidly, then stalls.
- In AMG, we *define* a “rough” error as one that Gauss-Seidel is effective in reducing and a “smooth” error as one on which Gauss-Seidel stalls.
- Loosely speaking, an error is “smooth” when $A e \approx 0$.
- $a_{ii} e_i \approx - \sum_{i \neq j} a_{ij} e_j$
Simplifying assumption

From now on, assume that the matrix $A$ is a symmetric M-matrix.

1. Diagonal elements are positive, off-diagonal are 0 or negative
2. Diagonal $> -(\text{sum of off-diagonals})$

- Original work on AMG was done for M-matrices.
- Some proofs are possible.
How to construct a coarse grid from a fine one.

- Define the notion of “strong dependence” (“influence”, “coupling”).
- Break the mesh up into regions in which each point is strongly dependent on a few distinguished points.
- The distinguished points will be the coarse mesh points.
- The coarse-to-fine mesh interpolation will be based on strong dependence.
Strong dependence

**Def. 1** Given a threshold \(0 < \theta \leq 1\), the variable \(u_i\) “strongly depends” on the variable \(u_j\) if

\[-a_{ij} \geq \theta \max_{k \neq i} \{-a_{ik}\}\]

**Def. 2** If the variable \(u_i\) strongly depends on the variable \(u_j\), then the variable \(u_j\) “strongly influences” \(u_i\).
Important feature of strong dependence

- Smooth error varies slowly in the direction of strong connection
- (See the discussion in Briggs, Henson, McCormick)
Coarsening

- Suppose you have a given fine grid
- Divide into C-points and F-points
- C-points will be next coarser grid
Coarsening

Requirements for C-points include

- Smooth error can be approximated accurately
- Smooth functions can be interpolated accurately
- Substantially fewer points
Definitions

- Neighborhood $N_i$ is the set of all points $j$ with $a_{ij} \neq 0$
- $S_i$ is the set of all points that strongly influence $i$
- $C_i$ is the set of C-points that strongly influence $i$
Coarsening heuristics

**H-1** For each F-point $i$, every point $j \in S_i$ that strongly influences $i$ either should be in $C_i$ or should strongly depend on at least one point in $C_i$.

**H-1a** (Aggressive coarsening) For each F-point $i$, every point $j \in S_i$ that strongly influences $i$ either should be in $C$ or should strongly depend on at least one point in $C$.

**H-2** The set of all coarse points $C$ should be a maximal subset of all points with the property that no C-point strongly depends on another C-point.
Example

Mesh with strong couplings
Example

Values
Example

C and F points
Example

Increment remaining values
Pick another C point
Example

Increment value
Example

Two more C points
Hypothesis 1 failures in red
Aggressive coarsining finishes here.
Example: final (standard) coarsening
Aggressive Coarsening

- Results in a coarser mesh but slower convergence.
- Can be used on only some levels.
- Requires a different interpolation formula, with longer-range couplings.
Interpolation from F to C

- Want
  \[(P_{C \times F} e)_i = \begin{cases} e_i & i \in C \\ \sum_{j \in C} w_{ij} e_j & i \in F \end{cases} \]

- Error is smooth on F \(\implies\) residual is small
  \[a_{ii} \approx - \sum_{j \in N_i} a_{ij} e_j\]

- \(N_i^S\) is strongly-coupled F points, \(N_i^W\) is weakly
  \[a_{ii} \approx - \sum_{j \in S_i} a_{ij} e_j - \sum_{j \in N_i^S} a_{ij} e_j - \sum_{j \in N_i^W} a_{ij} e_j\]

- Put weakly-coupled F points into diagonal
  \[(a_{ii} + \sum_{j \in N_i^W} a_{ij}) e_i \approx - \sum_{j \in S_i} a_{ij} e_j - \sum_{j \in N_i^S} a_{ij} e_j\]
Strongly-coupled F points get distributed

- Distribute $N_i^S$ points to all of $S_i$. For $j \in N_i^S$,

\[
e_j \approx \frac{\sum_{k \in C_i} a_{jk} e_k}{\sum_{k \in C_i} a_{jk}}
\]

- Hence

\[
w_{ij} = \frac{-a_{ij} + \sum_{m \in N_i^S} \left( \frac{a_{im} a_{mj}}{\sum_{k \in C_i} a_{mk}} \right)}{a_{ii} + \sum_{n \in N_i^W} a_{in}}
\]