Computational Fluid Dynamics I
Numerical Methods for Elliptic Equations-IV

GRÉTAR TRYGGVASON
Spring 2009

http://users.wpi.edu/~gretar/me612.html

Examples of elliptic equations
Direct Methods for 1D problems
Elementary Iterative Methods
Iteration as Time Integration
Example
Boundary Conditions
Convergence of Iterative Methods
1D Example
Formal Discussion
Multigrid methods
Fast Direct Method
Krylov Methods

So far we have covered elementary iterative methods to solve elliptic equations. Of those the SOR method is the most useful, particularly for code development and debugging.

However, considerable effort has been devoted to the solution of elliptic equations and currently there exist a number of much more efficient methods. For codes intended for large problems where runtime is important you should use such a method.

Multigrid methods are among the more popular ones.

---

Basic Idea

Suppose we want to solve a 1-D elliptic equation

$$\frac{\partial^2 f}{\partial x^2} - g = 0$$

An iterative process is analogous to solving an unsteady problem

$$\frac{df}{dt} = \left( \frac{\partial^2 f}{\partial x^2} - g \right)$$

And take it to the limit $t \to \infty$

---

Analytic Solution of

$$\frac{df}{dt} = \theta \left( \frac{\partial^2 f}{\partial x^2} - g \right)$$

The equation can be solved by Fourier series

$$f = \sum a_i \phi_i$$

and assume that $g$ can be expanded as

$$g = \sum b_i k^i e^{ikx}$$

Substituting into the equation,

$$\sum \frac{da_i}{dt} e^{ikx} = -\theta \sum (a_i(t) - b_i) k^i e^{ikx}$$

Solving for each $k$

$$\frac{da_i}{dt} = -\omega^2 (a_i(t) - b_i)$$

$$a_i(t) - b_i = (a_i(0) - b_i) \epsilon e^{-\omega t} \quad (\epsilon = e^{-\omega t})$$

Therefore, $a_i(t \to \infty) = b_i$

Rate of convergence $\sim -\omega^2$

$$\tau = \frac{1}{\omega^2}$$

High wave number modes damps out faster.
For iterative method to solve elliptic equations, there are high wave number and low wave number components of errors that need to be damped out.

Consider a domain \( L = 1 \) discretized by \( N \) points:

\[
\begin{align*}
\Delta x &= \frac{L}{N-1} \\
\Delta y &= \frac{L}{N-1}
\end{align*}
\]

This can also be visualized by following the decay of a initial conditions composed of two waves:

\[
f(x) = \sin(2\pi x) + \sin(10\pi x)
\]

For explicit time step, stability condition yields:

\[
\frac{\alpha \Delta t}{h^2} = \frac{1}{2} \Rightarrow \Delta t = \frac{h^2}{2\alpha}
\]

If the error at various wave number decays at:

\[
\varepsilon = e^{-\alpha t} \quad \text{where} \quad t = n \Delta t = \frac{nh^2}{2\alpha}
\]

\[
\frac{\varepsilon_T}{\varepsilon_L} = \exp\left(-k_L^2 n h^2 / 2\right)
\]

Short-wave errors \( \varepsilon_L = \exp\left(-n\pi^2 h^2 / 2\right) \) for \( k_L = 2\pi \)

decay much faster

Multigrid Method – the Idea

- A low wave number component on a fine grid becomes a high wave number component on a coarse grid:

\[
\varepsilon_L / \varepsilon_H = \exp\left(-n\pi^2 h^2 / 2\right)
\]

- Use a coarse grid system to converge low wave number component of the solution rapidly.

- Map it onto the fine grid system to converge high wave number component.

1D Example
Example

\[ \frac{d^2 f}{dx^2} = -g \]

Approximate by finite differences

\[ f_{i+1} + f_{i-1} - 2f_i = -g_i \quad R_i = h^2 g_i \]

Solve for \( f_i \)

\[ f_i = \frac{1}{2} \left( f_{i+1} + f_{i-1} + h^2 g_i \right) = \frac{1}{2} \left( f_{i+1} + f_{i-1} + R_i \right) \]

Iterate until convergence is "slow" on a fine grid

giving an approximate solution (but not fully converged).

Write the fully converged solution as the approximate solution plus a correction:

\[ f = \tilde{f} + \Delta f \]

Substitute into

\[ \frac{d^2 f}{dx^2} = -g \]

The decomposition

\[ f_j = \tilde{f}_j + \Delta f_j \]

gives

\[ \frac{d^2 \Delta f}{dx^2} + \frac{d^2 \tilde{f}}{dx^2} = -g \]

The correction is unknown, but the second derivative of the approximate solution found on the finer grid is known.

Solve for the correction on a coarser grid

\[ \Delta f_j = \frac{1}{2} \left( \Delta f_{j+1} + \Delta f_{j-1} - 2 \Delta f_j \right) \]

\[ R_j = \frac{4h^2 g_j + f_{j+1} + f_{j-1} - 2f_j}{h^2} \]

Which can be solved iteratively as before

This approach can be generalized so that we solve for the correction to the correction on even coarser grids.

Introduce

\[ x_i = f_i, \quad x_j = \Delta f_j, \quad x_k = \Delta \Delta f_k, \ldots \]

Where the \( x \) refers to \( f \) or the various corrections, depending on which grid we are working on

\[ \Delta \Delta f_k \]

On the j-grid, write:

\[ \Delta f_j = \frac{1}{2} \left[ \Delta f_{j+1} + \Delta f_{j-1} + 4(h^2 g_j + f_{j+1} + f_{j-1} - 2f_j) \right] \]

or

\[ \Delta f_j = \frac{1}{2} \left[ \Delta f_{j+1} + \Delta f_{j-1} + R_j \right] \]

where

\[ R_j = 4(h^2 g_j + f_{j+1} + f_{j-1} - 2f_j) \]
This can be generalized to the coarser grids

\[ \Delta x_j = \frac{1}{2} [\Delta x_{j+1} + \Delta x_{j-1} + 4(2^n h^2 g_j + x_{j+1} + x_{j-1} - 2x_j)] \]

or

\[ \Delta x_j = \frac{1}{2} [\Delta x_{j+1} + \Delta x_{j-1} + R_j] \]

where

\[ R_j = 4(2^n h^2 g_j + x_{j+1} + x_{j-1} - 2x_j) \]

Once the correction has been found, the solution on the finer grid can be corrected:

\[ x_{i+1} = x_i + \Delta x_i \]

when grid points overlap

\[ x_{i+1} = x_i + 0.5(\Delta x_i + \Delta x_{i+1}) \]

when grid points do not overlap

Transforming the source term to the coarser grid

\[ \text{DIFF: } 0.000662840903 \]

\[ \text{DIFF: } 0.00132568181 \]

\[ \text{DIFF: } 0.00530272722 \]

\[ \text{DIFF: } 0.0106054395 \]

\[ \text{DIFF: } 0.0212108791 \]

\[ \text{DIFF: } 0.0656985492 \]

\[ \text{NUMBER OF MESH INTERVALS: } 8 \]

\[ \text{DIFF: } 0.0499347299 \]

\[ \text{DIFF: } 0.0283034556 \]

\[ \text{DIFF: } 0.0147094727 \]

\[ \text{DIFF: } 6.65616244E-05 \]

\[ \text{DIFF: } 0.000117543153 \]

\[ \text{DIFF: } 0.000314788893 \]

\[ \text{DIFF: } 0.000945105217 \]

\[ \text{DIFF: } 0.0106054395 \]

\[ \text{DIFF: } 0.0212108791 \]

\[ \text{DIFF: } 0.0656985492 \]

\[ \text{NUMBER OF MESH INTERVALS: } 16 \]

\[ \text{DIFF: } 5.63040376E-05 \]

\[ \text{DIFF: } 0.000105820596 \]

\[ \text{DIFF: } 0.000314570963 \]

\[ \text{DIFF: } 0.00427171588 \]

\[ \text{DIFF: } 0.00144574605 \]

\[ \text{DIFF: } 0.00453080423 \]

\[ \text{DIFF: } 0.0141811594 \]

\[ \text{NUMBER OF MESH INTERVALS: } 32 \]

\[ \text{DIFF: } 9.19029117E-05 \]

\[ \text{DIFF: } 0.000103279948 \]

\[ \text{DIFF: } 0.000185795128 \]

\[ \text{DIFF: } 0.00144574605 \]

\[ \text{DIFF: } 0.00453080423 \]

2D Example

\[ L f_{i,j} = \frac{f_{i,j} - 2f_{i,j-1} + f_{i,j-2}}{\Delta x^2} + \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{\Delta y^2} = R_{i,j} \]

and iterate until residual becomes smaller than tolerance

\[ R_{i,j} \to 0 \]

Define

\[ f_{i,j} = f_{i,j}^0 + \Delta f_{i,j} \]

\[ f_{i,j}^0 \quad \text{Converged solution} \]

\[ f_{i,j}^n \quad \text{Solution after n-th iteration} \]

\[ \Delta f_{i,j} \quad \text{Correction} \]
Example: 4-Level V-Cycle

**Fine ➔ Coarse**

1. Iterate on Grid 1 for n times \( Lf_{ij} = 0 \) ⇒ \( R_{ij}^l \)
2. Restriction by injection to Grid 2 \( I^2 R_{ij}^l \)
3. Iterate on Grid 2 \( Lf_{ij} = -I^2 R_{ij}^l \) \( R_{ij}^l = I^2 R_{ij}^l + Lf_{ij} \) \( \text{(Save } \Delta f_{ij}, I^2 R_{ij}^l) \)
4. Restriction by injection to Grid 3 \( I^3 R_{ij}^l \)
5. Iterate on Grid 3 \( Lf_{ij} = -I^3 R_{ij}^l \) \( R_{ij}^l = I^3 R_{ij}^l + Lf_{ij} \) \( \text{(Save } \Delta f_{ij}, I^3 R_{ij}^l) \)
6. Restriction by injection to Grid 4 \( I^4 R_{ij}^l \)
7. Iterate on Grid 4 \( Lf_{ij} = -I^4 R_{ij}^l \)

**Coarse ➔ Fine**

1. Prolongate from Grid 4 to Grid 3 \( I^4 \Delta f_{ij} \)
2. Iterate on Grid 3 \( Lf_{ij} = -I^4 R_{ij}^l \) \( \text{(saved)} \)
3. Prolongate from Grid 3 to Grid 2 \( I^3 \Delta f_{ij} \)
4. Iterate on Grid 2 \( Lf_{ij} = -I^3 R_{ij}^l \) \( \text{(saved)} \)
5. Prolongate from Grid 2 to Grid 1 \( I^2 \Delta f_{ij} \)
6. Iterate on Grid 1 \( Lf_{ij} = -I^2 R_{ij}^l \) \( \text{(saved)} \)
7. Iterate on Grid 1 \( Lf_{ij} = 0 \)
8. If \( R_{ij} < \varepsilon \) then stop. Else, repeat the entire cycle.

Details on Prolongation

- **Coarse Grid**
  - Fine grid solution
  - Interpolation onto coarse grid (restriction)
  - Coarse grid correction
  - Interpolated onto fine grid (prolongation)
  - Converge on fine grid

- **Fine Grid**
  - \( \Delta f_{ij} = \frac{1}{4}(\Delta f_{i+1,j} + \Delta f_{i,j+1} + \Delta f_{i+1,j+1} + \Delta f_{i,j+1}) \)
  - \( \Delta f_{i,j} = -R_{ij} \) Coarse grid correction
Test Problem (Tannehill, p. 170)
- Laplace equation in a square domain
- Dirichlet conditions on four boundaries
- 5 Levels of resolution
  9x9, 17x17, 33x33, 65x65, 129x129
- 4 Methods
  1. Conventional Gauss-Seidel (GS)
  2. Gauss-Seidel-SOR with optimal $\omega$ (GS$^{\text{opt}}$)
  3. Multigrid with 2-level grids (MG2)
  4. Multigrid with maximum levels of grid (MGMAX)

A number of packages exist already and usually it is not necessary to write your own multigrid solver, particularly for simple geometries.

Multigrid methods are also used to solve steady-state problems such as flow over airplanes.

See, for example:
MUDPACK: Multigrid Software for Elliptic Partial Differential Equations
Fortran Code with OpenMP Directives for Shared Memory Parallelism
by John C. Adams
http://www.cisl.ucar.edu/css/software/mudpack/
Although iterative methods are the dominant technique for solutions of elliptic equations in CFD, Fast Direct Methods exists for special cases. The methods require simple domains (rectangles), simple equations (separable), and simple boundary conditions (periodic, or the derivative or the function equal to zero at each boundary).

\[ \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = S \]

separable

\[ \frac{\partial a(x,y)}{\partial x} \frac{\partial f}{\partial x} + \frac{\partial b(x,y)}{\partial y} \frac{\partial f}{\partial y} = S \]

non-separable

The fast Fourier transform

\[ f_j = \sum_{l=-N}^{N} \hat{f}_l e^{i \frac{2\pi}{N} lj} \]

inverse

\[ \hat{f}_l = \sum_{j=-N}^{N} f_j e^{-i \frac{2\pi}{N} lj} \]

Can be evaluated in \( 2N \log_2 N \) operations

The Cooley-Tukey algorithm

Write

\[ \Delta f_{i,j} = (f_{i+1,j} + f_{i-1,j} + f_{i+1,j} + f_{i-1,j} - 4f_{i,j}) = b_{i,j} \]

Take the double Fourier Transform

\[ f_{i,j} = \sum_{l=-N}^{N} \sum_{m=-N}^{N} \hat{f}_{l,m} e^{i \frac{2\pi}{N} lj} e^{i \frac{2\pi}{N} mj} \]

\[ \Delta f_{i,j} = \sum_{l=1}^{N} \sum_{m=1}^{N} \hat{f}_{l,m} e^{i \frac{2\pi}{N} lj} e^{i \frac{2\pi}{N} mj} (e^{i \frac{2\pi}{N} l} + e^{-i \frac{2\pi}{N} l} + e^{i \frac{2\pi}{N} m} + e^{-i \frac{2\pi}{N} m} - 4) \]

\[ \left(2 \cos \frac{2\pi}{N} l + 2 \cos \frac{2\pi}{N} m - 4\right) \]

Solve:

\[ \hat{f}_{l,m} = \frac{-b_{l,m}}{2 \left(2 - \cos \frac{2\pi}{N} l + \cos \frac{2\pi}{N} m\right)} \]

The algorithm is

1. Find \( \hat{b}_{l,m} \) by FFT
2. Find \( \hat{f}_{l,m} \) as shown before
3. find \( f_{i,j} \) by FFT

As outlined the method is applicable to periodic boundaries only. Other simple boundary conditions can be handled by simple changes (using cosine or sine series).

Other fast direct methods, such as Cyclic Reduction, are based on similar ideas

See FISHPACK
Subroutine : hwscrt (a,b,m,mbdcnd,bda,bdb,c,d,n,nbdcnd,bdc,bdd,
                elmbda,f,idimf,pertrb,ierror,w)

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                        Fish Pak                          *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*     a package of fortran subprograms for the solution of    *
*  separable elliptic partial differential equations          *
*     (version 3.1 , october 1980)                            *
*     by                                                    *
*     john adams, paul swarztrauber and rolland sweet        *
*     the national center for atmospheric research            *
*     boulder, colorado (DIGSFT) u.a.                        *
*     which is sponsored by                                 *
*     the national science foundation                       *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * *  purpose    * * * * * * * * * * * * * * * * * *
*          subroutine hwscrt solves the standard five-point   *
*     difference approximation to the helmholtz equation in  *
*     cartesian coordinates.                                 *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Iterative methods:
Stationary Iterative Methods
  Jacobi, Gauss-Seidel, SOR
Non Stationary Iterative Methods
  Conjugate Gradient (CG)
  Minimal Residual (MINRES)
  Generalized Minimal Residual (GMRES)
  And Many Others!

Most of these methods are formulized as minimization techniques, where the following function is minimized

\[ F = x^T Ax - x^T b \]

The search direction and the step are selected in a "best" way

Many methods work only for symmetric systems. While the pressure equation is usually symmetric, implicit methods usually do not lead to symmetric matrices.

Usually the system is PRECONDITIONED to make it better behaved.

Preconditioning
If the inverse of the matrix \( A \) was known, then the solution could be written down in a straightforward way

\[ Ax = b \quad \Rightarrow \quad A^{-1}A = I \]
\[ A^{-1}Ax = A^{-1}b \]
\[ Ix = x = A^{-1}b \]

Iterative methods generally converge faster if we have a matrix \( A \) that is "close" to the identity matrix \( I \)
If we have a matrix $M$ that is in some way close to the inverse of $A$, then the system

$$MAx = Mb$$

has the same solution as the original system

$$Ax = b$$

And should be easier to solve.

Finding the “best” preconditioner is an active research topic, but the simplest choice is:

$$m_{i,j} = \begin{cases} a_{i,j} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Jacobi preconditioner

Other choices include the so-called Incomplete Cholesky factorization or even multigrid methods.

One of the oldest non-stationary iterative method is the conjugate gradient method. For $M=I$ we have the unpreconditioned version.

The conjugate gradient method

Compute $r(0) = b - Ax(0)$ for some initial guess $x(0)$ for $i=1,2,\ldots$

solve $M z(i-1) = r(i-1)$

if $i=1$

$p(i) = z(0)$

else

$\beta_{i-1} = \frac{r(i-1)^T z(i-1)}{r(i-2)^T z(i-2)}$

$p(i) = z(i-1) - \beta_{i-1} p(i-1)$

end if

$q = Ap(i)$

$\alpha_i = \frac{r(i)^T q}{p(i)^T q}$

$x(i) = x(i-1) + \alpha_i p(i)$

$r(i) = r(i-1) - \alpha_i q$

Check convergence; continue if needed

Generally iterative methods generate a sequence of approximations that are used to construct a new approximation. Ideally, we only need to keep a few approximations, but one of the more popular technique, GMRES, requires all the previous iterates. This leads to the restarted GMRES.

At the present time there does not seem to be a “best” Krylov method. In addition to the relatively simple early methods like the conjugate gradient method, GMRES is fairly popular (particularly the restarted version) and BiCGSTAB has been used by a number of people.
A large number of pre-written software packages for the solution of elliptic equations is available.

In MATLAB:

Linear Equations (iterative methods).
- \text{pcg} - Preconditioned Conjugate Gradients Method.
- \text{bicg} - BiConjugate Gradients Method.
- \text{bicgstab} - BiConjugate Gradients Stabilized Method.
- \text{cgs} - Conjugate Gradients Squared Method.
- \text{gmres} - Generalized Minimum Residual Method.
- \text{qmr} - Quasi-Minimal Residual Method.

Resources

MUDPACK
http://www.scd.ucar.edu/css/software/mudpack/

FISHPACK
http://www.scd.ucar.edu/css/software/fishpack/

Multigrid website
http://www.mgnet.org/

Netlib is a collection of mathematical software, papers and databases.
http://www.netlib.org/

Book:
Barrett, Berry, Chan, Dellel, Donato, Dongarra, Eijkhout, Pozo, Romine, and Van der Vorst
Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods
http://www.netlib.org/templates/Templates.html