Computational Fluid Dynamics: Lecture 9 (ME EN 6720)

Prof. Rob Stoll
Department of Mechanical Engineering
University of Utah

Spring 2014

Elliptical Equations

<u>Elliptical equations and Solvers (Ferziger chapter 5):</u>

- many problems involve the solution of equations of the form:

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}=f(x,y,z)$$

- this type of equation is an elliptical equation and is usually referred to as the Poisson equation. In vector form:
  - $\nabla^2\phi = f(\vec{x})$ (Poisson equation)
  - $\nabla^2\phi = 0$ (Laplace equation)

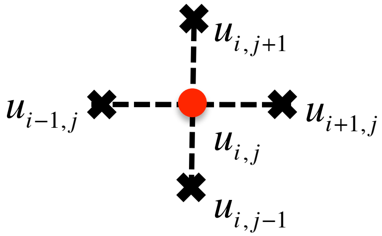
- Examples of problems involving elliptical equations:
  - ○ velocity field for invicid incompressible flow (potential flow)
  - ○ pressure field for incompressible flow
  - ○ steady-state Navier-Stokes equations (in a highly nonlinear complex way)
  - ○ steady-state multi-dimensional heat equation

F.D.E.s for Elliptical Problems

Finite Difference approximation for 2D Elliptical equations:

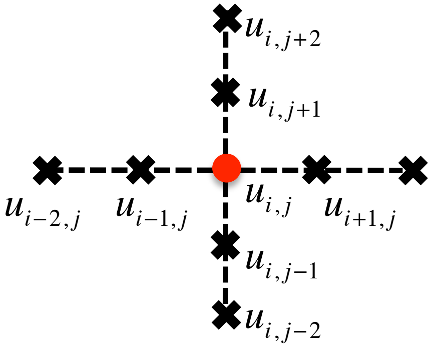
- 5-point: $$\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{(\Delta x)^2}+\frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{(\Delta y)^2}=0$$

  -T.E. ~ [(Δx)^2, (Δy)^2]



- 9-point: $$\frac{-u_{i-2,j}+16u_{i-1,j}-30u_{i,j}+16u_{i+1,j}-u_{i+2,j}}{12(\Delta x)^2}+$$ $$\frac{-u_{i,j-2}+16u_{i,j-1}-30u_{i,j}+16u_{i,j+1}-u_{i,j+2}}{12(\Delta x)^2}=0$$

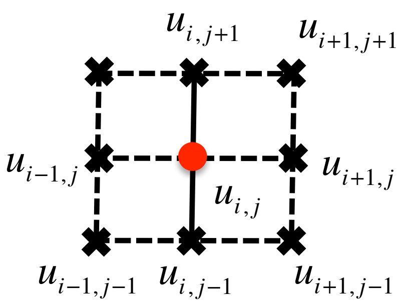
  -T.E. ~ [(Δx)^4, (Δy)^4]



- 9-point:

$$u_{i+1,j+1}+u_{i-1,j+1}+u_{i+1,j-1}+u_{i-1,j-1}-2\frac{(\Delta x)^2-5(\Delta y)^2}{(\Delta x)^2+(\Delta y)^2}\left(u_{i+1,j}+u_{i-1,j}\right)$$

$$-2\frac{5(\Delta x)^2-(\Delta y)^2}{(\Delta x)^2+(\Delta y)^2}\left(u_{i,j+1}+u_{i,j-1}\right)-20u_{i,j}=0$$



-T.E. ~ [(Δx)^2, (Δy)^2] on any grid ~ [(Δx)^6, (Δy)^6] on a square grid (Δx=Δy)

Solving Elliptical Problems

How do we solve these types of equations?

- Lets look closer at the 5-point scheme, rearranging we get:

$$u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + \left(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}\right)\frac{(\Delta x)^2}{(\Delta y)^2} = 0 \quad \text{or}$$

$$u_{i+1,j} + u_{i-1,j} + \beta^2 u_{i,j+1} + \beta^2 u_{i,j-1} - 2\left(1+\beta^2\right)u_{i,j} = 0 \qquad \text{where} \;\; \beta = \frac{\Delta x}{\Delta y}$$

- This is our pentadiagnol matrix system from lecture 8 described by the general equation:

coefficient matrix

$$\mathbf{A}\vec{x} = 0 \qquad \text{(for a Poission equation we would just have a } \vec{b} = f_{i,j} \text{ on the RHS)}$$

$u_{i,j}$ vector

- We can't split this problem up (like we did with the parabolic 2D Heat equation)

- In general we see that many problems can be written in matrix form as:

$$\mathbf{A}\vec{x} = \vec{b}$$

- We have 2 general methodologies for solving these type of equations:

  -Direct Methods

  -Iterative Methods

Direct Solvers

We have already discussed some direct solvers

- **<u>Gauss Elimination:</u>** (Ferziger pg. 92-93)

  -Basic technique learned in linear algebra classes

  -Very computationally expensive (for full matrices actually isn't so bad)

  -Typical systems resulting from FDE equations are sparse so this method is wasteful

- **<u>LU Decomposition:</u>** (Ferziger pg. 94)

  -Matrix $\mathbf{A}$ can be factored into the product of two matrices

  $\mathbf{A} = \mathbf{LU}$ where $\mathbf{L}$ is lower triangular
  $\mathbf{U}$ is upper triangular

  -We can then solve our problem in 2 steps

  - defining: $\mathbf{U}\vec{x} = \vec{Y}$

    $\mathbf{L}\vec{Y} = \vec{b}$
  - we can then find $\vec{x}$ by first finding $\vec{Y}$ and then $\vec{x}$

  -Advantage of this method:

  - we can do the factorization (into $\mathbf{L}$ and $\mathbf{U}$) without knowing $\vec{b}$
  - for systems requiring many repeats of the same $\mathbf{A}$ this can result in considerable speed up

  -Our tridiagonal solver (Thomas-Algorithm) was a special case of this for out 1D PDE

Iterative Solvers

What about elliptical equations?

In general elliptical equations are (at least) pentadiagonal and (in general) we can't use the splitting trick we used with the heat equation

Iterative Methods:

Idea: guess then improve through the solution of the equation. If *k* is our iteration number we can write our general system of equations as:

$$\mathbf{A}\vec{x}^{k} = \vec{b} - \vec{\rho}^{k}$$ where $\vec{\rho}^{k}$ is our non-zero residual

Our iteration error is $\vec{\varepsilon}^{k} = \vec{x} - \vec{x}^{k}$

The residual is defined in terms of this error: $\mathbf{A}\vec{\varepsilon}^{k} = \vec{\rho}^{k}$

and our goal is to make $\vec{\rho}^{k} \rightarrow 0$

In general we can write any iterative method as: $\mathbf{M}\vec{x}^{k+1} = \mathbf{N}\vec{x}^{k+1} + \vec{B}$

Since $\vec{x}^{k+1} = \vec{x}^{k} = \vec{x}$ at convergence the above equation must have

$$\mathbf{A} = \mathbf{M} - \mathbf{N} \;\; \text{and} \;\; \vec{\mathrm{B}}=\vec{\mathrm{b}}$$

With different methods corresponding to different $\mathbf{M}$ and $\mathbf{N}$ matrices.

Iterative Solvers

Iterative Methods:

What are $\mathbf{M}$ and $\mathbf{N}$ and how should they be specified?

-$\mathbf{N}$ = what we transform (or multiply) our previous iteration by

-$\mathbf{M}$ = what we need to invert to get $\vec{x}^{k+1}$

We can rewrite this as (by subtracting $\mathbf{M}\vec{x}^{k}$ from each side):

$$\mathbf{M}\underbrace{\left(\vec{x}^{k+1}-\vec{x}^{k}\right)}_{\substack{\text{update}\\\text{(iteration error)}}}=\vec{B}-\left(\mathbf{M}-\mathbf{N}\right)\vec{x}^{k}$$

Different methods (and spatial discretizations) have different $\mathbf{A}$’s and → $\mathbf{M}$’s and $\mathbf{N}$’s

For a method to be “good”:

- $\mathbf{N}\vec{x}^{k}$ has to be simple (sparse nature of $\mathbf{A}$ takes care of this)
- $\mathbf{M}$ must be easily invertible
- $\mathbf{M}$ should be a good approximation of $\mathbf{A}$

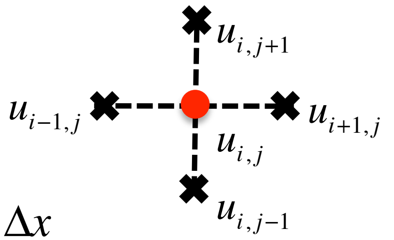
Iterative Solver Example

A simple example of an iterative solver for the Laplace Equation:

- Recall that for the 2D Laplace equation given by:

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=0$$

5-point scheme stencil



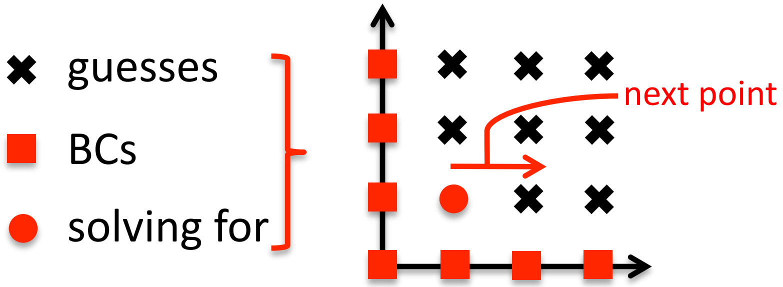
- we can write a 5-point scheme F.D.E as:

$$u_{i+1,j}+u_{i-1,j}+\beta^2 u_{i,j+1}+\beta^2 u_{i,j-1}-2\left(1+\beta^2\right)u_{i,j}=0 \qquad \text{where} \quad \beta=\frac{\Delta x}{\Delta y}$$

- **Jacobi Iterations:** simplest solution method

  -move $u_{i,j}$ over to the over side of the equation:

$$u_{i,j}^{k+1}=\frac{1}{2\left(1+\beta^2\right)}\left[u_{i+1,j}^{k}+u_{i-1,j}^{k}+\beta^2\left(u_{i,j+1}^{k}+u_{i,j-1}^{k}\right)\right]=0$$



- we guess all our values (to start) then solve a new field (at iteration k+1) based on those guesses (at iteration k) moving across the grid
- continue to next iteration and repeat until convergence is achieved

Iterative Solver Example

- We can compare the Jacobi iteration scheme to our explicit FTCS for the 2D heat equation (with Δx = Δy): $u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{\alpha\Delta t}{(\Delta x)^2}\left[u_{i+1,j}^{n} + u_{i-1,j}^{n} - 4u_{i,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}\right]$

  where our stability criteria is:

  $$\frac{\alpha\Delta t}{(\Delta x)^2} \le 1/4 \text{ for a value equal to } 1/4 \Rightarrow u_{i,j}^{n+1} = \frac{1}{4}\left[u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}\right]$$

- How is the explicit FTCS scheme different than the Jacobi scheme?

  -FTCS is an "exact" solution at every time step. We give initial conditions and move our solution forward in time

  -Jacobi iterations starts with a guess and moves towards convergence. **Different iteration steps have no physical meaning**.

How does the Jacobi iteration scheme fit into $\mathbf{M}\vec{x}^{k+1} = \mathbf{N}\vec{x}^{k} + \vec{B}$?

$\vec{B} = 0$ (Laplace equation)

$\mathbf{M} = [1]$ (constant identiy matrix)

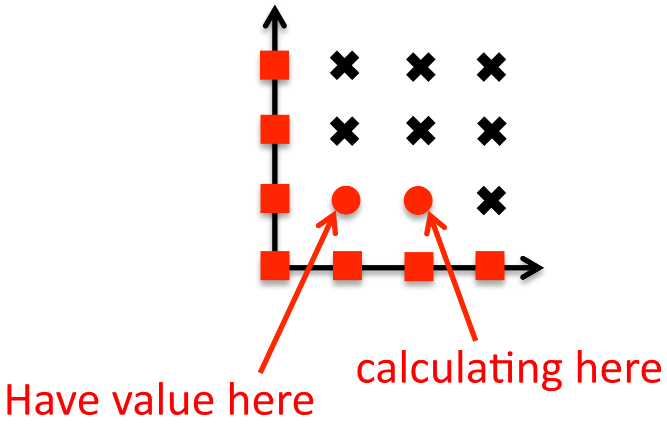
$\mathbf{N} =$ a 4 diagonal matrix

How does this meet our rules for "good" solvers?

- $\mathbf{M}$ is simple to invert
- $\mathbf{N}$ isn't too bad to calculate
- $\mathbf{M}$ isn't even close to $\mathbf{A}$

Iterative Solver Example

- Jacobi iterations is very slow. How can we improve it?



Have value here

calculating here

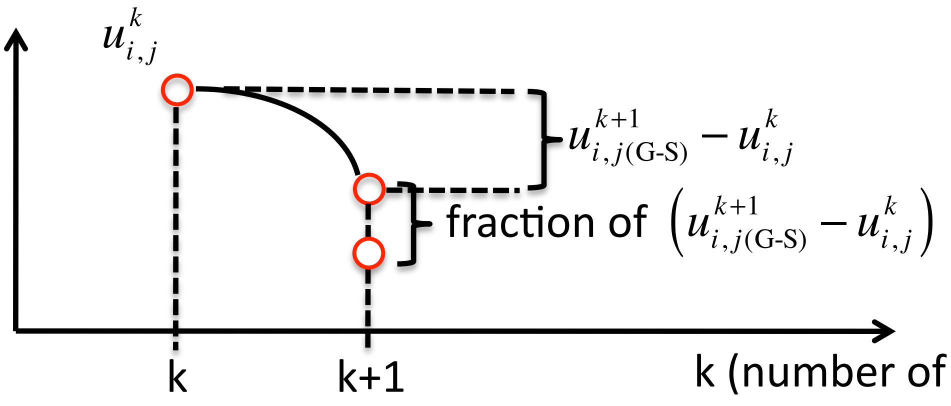
- During the calculation of $u_{i,j}$ at iteration k+1 we already have values at $u_{i-1,j}^{k+1}$ and $u_{i,j-1}^{k+1}$.
- We can use these values to speed up our convergence.

$$u_{i,j}^{k+1}=\frac{1}{2\left(1+\beta^2\right)}\left[u_{i+1,j}^{k}+u_{i-1,j}^{k+1}+\beta^2\left(u_{i,j+1}^{k}+u_{i,j-1}^{k+1}\right)\right]$$

- This method is called the **Gauss-Seidel method**
- It is 100% faster than Jacobi iterations (but still slower than needed)

- How can we make further improvements?
  - use the trend in the iterations to accelerate convergence (in general we can do this with any iteration scheme)

Relaxation Methods (SOR)


$u_{i,j}^{k}$
$u_{i,j(\text{G-S})}^{k+1} - u_{i,j}^{k}$
fraction of $\left(u_{i,j(\text{G-S})}^{k+1} - u_{i,j}^{k}\right)$
k
k+1
k (number of iterations)

- we project that our next iteration will be in the same direction as the last

- Extracting from the trend we have:

$$u^{k+1} = u^{k} + \omega\left[u_{(\text{G-S})}^{k+1} - u^{k}\right] = (1-\omega)u^{k} + \omega u_{(\text{G-S})}^{k+1}$$

- This acceleration is usually referred to as Successive Over Relaxation (SOR) when applied to Gauss-Seidel: $u_{i,j}^{k+1} = (1-\omega)u_{i,j}^{k} + \dfrac{\omega}{2\left(1+\beta^{2}\right)}\left[u_{i+1,j}^{k} + u_{i-1,j}^{k+1} + \beta^{2}\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+1}\right)\right]$

- What value should we use for ω (relaxation parameter)?
  - For convergence: 0 < ω < 2

    -If ω = 1 we recover our scheme (G-S)

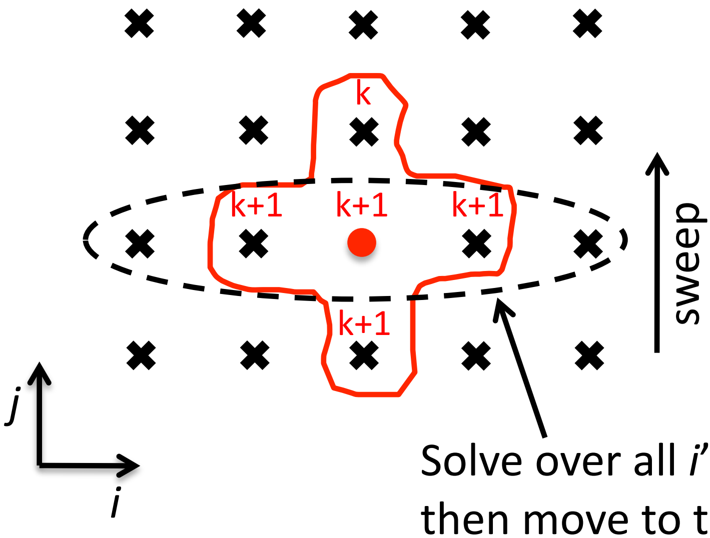
    -0 < ω < 1 we have under relaxation (damps oscillatory behavior)

    -1 < ω < 2 over relaxation

Iterative Solvers

- So far, all the methods we have discussed (Gauss-Seidel, etc.) are **point iterative methods**
- One problem with these type of methods is the nature of B.V.P.s For a converged solution our boundary conditions have to propagate throughout the solution domain.
- With point iterative schemes this can be a slow process (especially for fine grids).
- One solution is to this is **block iterative schemes**.

-Example of a simple block iterative scheme: **SOR by lines (SLOR)**



Solve over all *i*'s at each *j* then move to the next row

- starting at the bottom and working towards the top we iterate over I values solving the resulting tridiagonal system.

$$u_{i,j}^{k+1} = \frac{u_{i+1,j}^{k+1} + u_{i-1,j}^{k+1} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+1}\right)}{2\left(1+\beta^2\right)}$$

where $\beta = \dfrac{\Delta x}{\Delta y}$

Iterative Solvers

- We can use relaxation (SOR) with this in two different ways

    1) after we calculate each row we can apply SOR before moving to the next row (ie SOR between rows) using:

$$u_{i,j}^{k+1} = u_{i,j}^{k} + \omega\left(u_{i,j}^{k+1} - u_{i,j}^{k}\right)$$

    2) Include SOR in our approach directly

$$u_{i,j}^{k+1} = \left(1-\omega\right)u_{i,j}^{k} + \frac{\omega}{2\left(1+\beta^2\right)}\left[u_{i+1,j}^{k+1} + u_{i-1,j}^{k+1} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+1}\right)\right]$$

Note: Recall Thomas-algorithm is guaranteed to have a solution if the matrix (tri-diagonal system) is diagonally dominant. For this to always be true: $\omega \leq 1+\beta^2$

- An iteration cycle is complete when we finish all the rows.

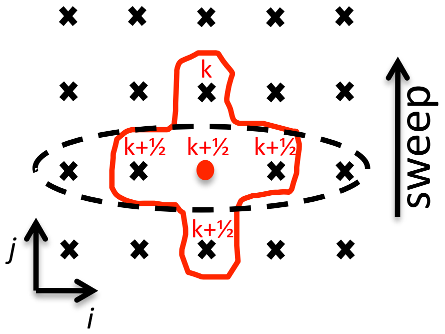
    -we then repeat until we meet our convergence criteria

- SLOR for the Laplace equation with Dirichlet BCs can be shown to take 1/√2 as many iterations as standard G-S with SOR.
- Faster convergence (lower number of iterations) is attributed to the BCs influencing the entire equations at once
- The trade off for this is that each iteration is more costly…

ADI as an Iterative Solver

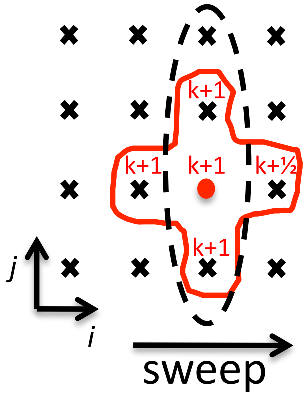
- We can extend SLOR by sweeping over columns after the rows

x-sweep: (for each *j*)



$$u_{i,j}^{k+\frac{1}{2}} = (1-\omega)u_{i,j}^{k} + \frac{\omega}{2\left(1+\beta^2\right)}\left[u_{i+1,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+\frac{1}{2}} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k+\frac{1}{2}}\right)\right]$$

y-sweep: (for each *i*)



$$u_{i,j}^{k+1} = (1-\omega)u_{i,j}^{k+\frac{1}{2}} + \frac{\omega}{2\left(1+\beta^2\right)}\left[u_{i+1,j}^{k+\frac{1}{2}} + u_{i-1,j}^{k+1} + \beta^2\left(u_{i,j+1}^{k+1} + u_{i,j-1}^{k+1}\right)\right]$$

Note: to preserve diagonal dominance we need: $\begin{cases} \omega \le 1+\beta^2 & \text{for the row sweep (over } j\text{'s)} \\ \omega \le \left(1+\beta^2\right)/\beta^2 & \text{for the column sweep (over } i\text{'s)} \end{cases}$