2.29 Numerical Fluid Mechanics
Spring 2015 – Lecture 21

REVIEW Lecture 20: Time-Marching Methods and ODEs–IVPs

- Time-Marching Methods and ODEs – Initial Value Problems

$$\frac{d\,\bar{\mathbf{\Phi}}}{dt} = \mathbf{B}\,\bar{\mathbf{\Phi}} + (\mathbf{bc}) \quad \text{or} \quad \frac{d\,\bar{\mathbf{\Phi}}}{dt} = \mathbf{B}(\bar{\mathbf{\Phi}}, t) \;\; ; \quad \text{with} \;\; \bar{\mathbf{\Phi}}(t_0) = \bar{\mathbf{\Phi}}_0$$

  - Euler's method
  - Taylor Series Methods
    - Error analysis: for two time-levels, if truncation error is of $O(h^{n})$, the global error is of $O(h^{n-1})$
  - Simple 2nd order methods
    - Heun's Predictor-Corrector and Midpoint Method (belong to Runge-Kutta's methods)
- To achieve higher accuracy in time: utilize information (known values of the derivative in time, i.e. the RHS f) at more points in time, equate to Taylor series
  - Runge-Kutta Methods
    - Additional points are between t_n and t_{n+1}

$$\phi^{n+1} - \phi^{n} = \int_{t_n}^{t_{n+1}} f(t,\phi)\,dt$$

  - Multistep/Multipoint Methods: Adams Methods
    - Additional points are at past time steps
  - Practical CFD Methods
  - Implicit Nonlinear systems
  - Deferred-correction Approach

TODAY (Lecture 21): End of Time-Marching Methods, Grid Generation

- Time-Marching Methods and ODEs – IVPs: End
  - Multistep/Multipoint Methods
  - Implementation of Implicit Time-Marching: Nonlinear systems
  - Deferred-correction Approach
- Complex Geometries
  - Different types of grids
  - Choice of variable arrangements: Cartesian or grid-oriented velocity, staggered or collocated var.
- Grid Generation
  - Basic concepts and structured grids
    - Stretched grids
    - Algebraic methods (for stretched grids)
    - General coordinate transformation
    - Differential equation methods
    - Conformal mapping methods
  - Unstructured grid generation
    - Delaunay Triangulation
    - Advancing Front method

References and Reading Assignments Time-Marching

- Chapters 25 and 26 of "Chapra and Canale, Numerical Methods for Engineers, 2014/2010/2006."
- Chapter 6 on "Methods for Unsteady Problems" of "J. H. Ferziger and M. Peric, Computational Methods for Fluid Dynamics. Springer, NY, 3rd edition, 2002"
- Chapter 6 on "Time-Marching Methods for ODE's" of "H. Lomax, T. H. Pulliam, D.W. Zingg, Fundamentals of Computational Fluid Dynamics (Scientific Computation). Springer, 2003"

Multistep/Multipoint Methods

- Additional points are at time steps at which data has already been computed
- Adams Methods: fitting a (Lagrange) polynomial to the derivatives at a number of points in time
  - Explicit in time (up to t_n): Adams-Bashforth methods

$$\phi^{n+1} - \phi^n = \sum_{k=n-K}^{n} \beta_k \; f(t_k, \phi^k) \, \Delta t$$

  - Implicit in time (up to t_{n+1}): Adams-Moulton methods

$$\phi^{n+1} - \phi^n = \sum_{k=n-K}^{n+1} \beta_k \; f(t_k, \phi^k) \, \Delta t$$

  - Coefficients β_k's can be estimated by Taylor Tables:
    - Fit Taylor series so as to cancel as high-order terms as possible

Example: Taylor Table for the Adams-Moulton 3-steps (4 time-nodes) Method

Denoting $h \equiv \Delta t$, $\phi \equiv u$, $\frac{du}{dt} = u' = f(t,u)$ and $u'_n = f(t_n, u^n)$, one obtains for $K = 2$:

$$u^{n+1} - u^n = \sum_{k=-K}^{1} \beta_k f(t_{n+k}, u^{n+k})\,\Delta t = h\left[\beta_1 f(t_{n+1}, u^{n+1}) + \beta_0 f(t_n, u^n) + \beta_{-1} f(t_{n-1}, u^{n-1}) + \beta_{-2} f(t_{n-2}, u^{n-2})\right]$$

Taylor Table (at t_n):

- The first row (Taylor series) + next 5 rows (Taylor series for each term) must sum to zero
- This can be satisfied up to the 5th column (cancels 4th order term)
- Hence, the AM method with 4-time levels is 4th order accurate

	u_n	$h \cdot u'_n$	$h^2 \cdot u''_n$	$h^3 \cdot u'''_n$	$h^4 \cdot u''''_n$
u_{n+1}	1	1	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{24}$
$-u_n$	-1				
$-h\beta_1 u'_{n+1}$		$-\beta_1$	$-\beta_1$	$-\beta_1 \frac{1}{2}$	$-\beta_1 \frac{1}{6}$
$-h\beta_0 u'_n$		$-\beta_0$			
$-h\beta_{-1} u'_{n-1}$		$-\beta_{-1}$	β_{-1}	$-\beta_{-1} \frac{1}{2}$	$\beta_{-1} \frac{1}{6}$
$-h\beta_{-2} u'_{n-2}$		$-(-2)^0 \beta_{-2}$	$-(-2)^1 \beta_{-2}$	$-(-2)^2 \beta_{-2} \frac{1}{2}$	$-(-2)^3 \beta_{-2} \frac{1}{6}$

solving for the β_k's $\Rightarrow \beta_1 = 9/24, \quad \beta_0 = 19/24, \quad \beta_{-1} = -5/24$ and $\beta_{-2} = 1/24$

Examples of Adams Methods for Time-Integration

Explicit Methods. (Adams-Bashforth, with ABn meaning n^{th} order AB)

$u_{n+1} = u_n + hu'_n$	Euler
$u_{n+1} = u_{n-1} + 2hu'_n$	Leapfrog
$u_{n+1} = u_n + \frac{1}{2}h\left[3u'_n - u'_{n-1}\right]$	AB2
$u_{n+1} = u_n + \frac{h}{12}\left[23u'_n - 16u'_{n-1} + 5u'_{n-2}\right]$	AB3

Implicit Methods. (Adams-Moulton, with AMn meaning n^{th} order AM)

$u_{n+1} = u_n + hu'_{n+1}$	Implicit Euler
$u_{n+1} = u_n + \frac{1}{2}h\left[u'_n + u'_{n+1}\right]$	Trapezoidal (AM2)
$u_{n+1} = \frac{1}{3}\left[4u_n - u_{n-1} + 2hu'_{n+1}\right]$	2nd-order Backward
$u_{n+1} = u_n + \frac{h}{12}\left[5u'_{n+1} + 8u'_n - u'_{n-1}\right]$	AM3

Practical
Multistep Time-Integration Methods for CFD

- High-resolution CFD requires large discrete state vector sizes to store the spatial information
- As a result, up to two times (one on each side of the current time step) have often been utilized (3 time-nodes): $u^{n+1} - u^n = h\left[\beta_1 f(t_{n+1}, u^{n+1}) + \beta_0 f(t_n, u^n) + \beta_{-1} f(t_{n-1}, u^{n-1})\right]$
- Rewriting this equation in a way such that differences w.r.t. Euler's method are easily seen, one obtains ($\theta = 0$ for explicit schemes):

$$(1+\xi)\, u^{n+1} = \left[(1+2\xi)\, u^n - \xi\, u^{n-1}\right] + h\left[\theta\, f(t_{n+1}, u^{n+1}) + (1-\theta+\varphi)\, f(t_n, u^n) - \varphi f(t_{n-1}, u^{n-1})\right]$$

θ	ξ	φ	Method	Order
0	0	0	Euler	1
1	0	0	Implicit Euler	1
1/2	0	0	Trapezoidal or AM2	2
1	1/2	0	2nd-order Backward	2
3/4	0	−1/4	Adams type	2
1/3	−1/2	−1/3	Lees	2
1/2	−1/2	−1/2	Two-step trapezoidal	2
5/9	−1/6	−2/9	A-contractive	2
0	−1/2	0	Leapfrog	2
0	0	1/2	AB2	2
0	−5/6	−1/3	Most accurate explicit	3
1/3	−1/6	0	Third-order implicit	3
5/12	0	1/12	AM3	3
1/6	−1/2	−1/6	Milne	4

- Note that higher order R-K methods in time are now also used, especially low storage R-K.

Numerical Fluid Mechanics

Implementation of Implicit Time-Marching Methods: Nonlinear Systems and Larger dimensions

- Consider the nonlinear system (discrete in space):

$$\frac{d\,\mathbf{\Phi}}{dt} = \mathbf{B}(\mathbf{\Phi}, t)\ ;\ \text{with}\ \ \mathbf{\Phi}(t_0) = \mathbf{\Phi}_0$$

- For an explicit method in time, solution is straightforward
  - For explicit Euler: $\mathbf{\Phi}^{n+1} = \mathbf{\Phi}^{n} + \mathbf{B}(\mathbf{\Phi}^{n}, t_n)\ \Delta\text{t}$
  - More general, e.g. AB: $\mathbf{\Phi}^{n+1} = \mathbf{F}(\mathbf{\Phi}^{n}, \mathbf{\Phi}^{n-1}, ..., \mathbf{\Phi}^{n-K}, t_n)\ \Delta\text{t}$

- For an implicit method
  - For Implicit Euler: $\mathbf{\Phi}^{n+1} = \mathbf{\Phi}^{n} + \mathbf{B}(\mathbf{\Phi}^{\boxed{n+1}}, t_{n+1})\ \Delta\text{t}$
  - More general: $\mathbf{\Phi}^{n+1} = \mathbf{F}(\mathbf{\Phi}^{\boxed{n+1}}, \mathbf{\Phi}^{n}, \mathbf{\Phi}^{n-1}, ..., \mathbf{\Phi}^{n-K}, t_{n+1})\ \Delta\text{t}$ or

    $\tilde{\mathbf{F}}(\mathbf{\Phi}^{n+1}, \mathbf{\Phi}^{n}, \mathbf{\Phi}^{n-1}, ..., \mathbf{\Phi}^{n-K}, t_{n+1}) = 0\ ;$ with $\tilde{\mathbf{F}} = \mathbf{F}\Delta\text{t} - \mathbf{\Phi}^{n+1}$

=> a nontrivial scheme is needed to obtain $\mathbf{\Phi}^{n+1}$

Implementation of Implicit Time-Marching Methods: Larger dimensions and Nonlinear systems

- Two main options for an implicit method, either:

1. Linearize the RHS at t_n:
   - Taylor Series: $\mathbf{B}(\mathbf{\Phi},t) = \mathbf{B}(\mathbf{\Phi}^n,t_n) + \mathbf{J}^n\,(\mathbf{\Phi}-\mathbf{\Phi}^n) + \left.\frac{\partial \mathbf{B}}{\partial t}\right|^n (t-t_n) + O(\Delta t^2)$ for $t_n \le t \le t_{n+1}$

     where $\mathbf{J}^n = \left.\frac{\partial \mathbf{B}}{\partial \mathbf{\Phi}}\right|^n$; i.e. $[\mathbf{J}^n]_{ij} = \left.\frac{\partial \mathbf{B}_i}{\partial \mathbf{\Phi}_j}\right|^n$ (Jacobian Matrix)
   - Hence, the linearized system (for the frequent case of system not explicitly function of t):

$$\frac{d\,\mathbf{\Phi}}{dt} = \mathbf{B}(\mathbf{\Phi}) \quad \Rightarrow \quad \boxed{\frac{d\,\mathbf{\Phi}}{dt} = \mathbf{J}^n\,\mathbf{\Phi} + \left[\mathbf{B}(\mathbf{\Phi}^n) - \mathbf{J}^n\,\mathbf{\Phi}^n\right]}$$

2. Use an iteration scheme at each time step, e.g. fixed point iteration (direct), Newton-Raphson or secant method
   - Newton-Raphson: $x_{r+1} = x_r - \frac{1}{f'(x_r)} f(x_r) \quad \Rightarrow \quad \boxed{\mathbf{\Phi}_{r+1}^{n+1} = \mathbf{\Phi}_r^{n+1} - \left(\left.\frac{\partial \tilde{\mathbf{F}}}{\partial \mathbf{\Phi}^{n+1}}\right|_r\right)^{-1} \tilde{\mathbf{F}}(\mathbf{\Phi}_r^{n+1}, t_{n+1})}$
   - Iteration often rapidly convergent since initial guess to start iteration at t_n close to unknown solution at t_{n+1}

Deferred-Correction Approaches

- Size of computational molecule affects both storage requirements and effort needed to solve the algebraic system at each time-step
  - Usually, we wish to keep only the nearest neighbors of the center node P in the LHS of equations (leads to tri-diagonal matrix or something close to it) $\Rightarrow$ easier to solve linear/nonlinear system
  - But, approximations that produce such molecules are often not accurate enough
- Way around this issue?
  - Leave only the terms containing the nearest neighbors in the LHS and bring all other more-remote terms to the RHS
    - This requires that these terms be evaluated with previous or old values, which may lead to divergence of the iterative scheme
- Better approach?

Deferred-Correction Approaches, Cont'd

- Better Approach
  - Compute the terms that are approximated with a high-order approximation explicitly and put them in the RHS
  - Take a simpler approximation to these terms (that give a small computational molecule). Insert it twice in the equation, with a + and - sign
  - One of these two simpler approximations, keep it in the LHS of the equations (with unknown variables values, i.e. implicit/new). Move the other to the RHS (i.e. computing it explicitly using existing/old values)
  - The RHS now contains the difference between two explicit approximations of the same term, and is likely to be small $\Rightarrow$
    - Likely no convergence problems to an iteration scheme (Jacobi, GS, SOR, etc) or gradient descent (CG, etc)
  - Once the iteration converges, the low order approximation terms (one explicit, the other implicit) drop out and the solution corresponds to the higher-order approximation
- $\Rightarrow$ Using H & L for high & low orders: $\mathbf{A}^H \mathbf{x} = \mathbf{b} \quad \rightarrow \mathbf{A}^L \mathbf{x} = \mathbf{b} - \left[\mathbf{A}^H \mathbf{x} - \mathbf{A}^L \mathbf{x} \right]^{\text{old}}$

Deferred-Correction Approaches, Cont'd

- This approach can be very powerful and general
  - Used when treating higher-order approximations, non-orthogonal grids, corrections needed to avoid oscillation effects, etc
  - Since RHS can be viewed as a correction ⇒ called deferred-correction
  - Note: both L&H terms could be implicit in time: use L&H explicit starter to get first values and then most recent old values in bracket during iterations (similar to Jacobi vs. Gauss Seidel)
    - Explicit for H (high-order) term, implicit for L (low-order) term

$$\mathbf{A}^H \mathbf{x} = \mathbf{b} \quad \rightarrow \mathbf{A}^L \mathbf{x}_{\text{implicit}} = \mathbf{b} - \left[\mathbf{A}^H \mathbf{x}_{\text{explicit}} - \mathbf{A}^L \mathbf{x}_{\text{implicit}} \right]^{\text{old}}$$

    - Implicit for both L and H terms (similar to Gauss-Seidel)

$$\mathbf{A}^H \mathbf{x} = \mathbf{b} \quad \rightarrow \mathbf{A}^L \mathbf{x}_{\text{implicit}} = \mathbf{b} - \left[\mathbf{A}^H \mathbf{x}_{\text{implicit}} - \mathbf{A}^L \mathbf{x}_{\text{implicit}} \right]^{\text{old}}$$

Deferred-Correction Approaches, Cont'd

- Example 1: FD methods with High-order Pade' schemes
  - One can use the PDE itself to express implicit Pade' time derivative $\left(\frac{\partial \phi}{\partial t}\right)_{n+1}$ as a function of ϕ^{n+1} (see homework)
  - Or, use deferred-correction (within an iteration scheme of index r):
    - In time:

$$\left(\frac{\partial \phi}{\partial t}\right)_n^{r+1} = \left(\frac{\phi_{n+1} - \phi_{n-1}}{2\Delta t}\right)^{r+1} + \left[\left(\frac{\partial \phi}{\partial t}\right)_n^{\text{Pade'}} - \frac{\phi_{n+1} - \phi_{n-1}}{2\Delta t}\right]^r$$

    - In space:

$$\left(\frac{\partial \phi}{\partial x}\right)_i^{r+1} = \left(\frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x}\right)^{r+1} + \left[\left(\frac{\partial \phi}{\partial x}\right)_i^{\text{Pade'}} - \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x}\right]^r$$

    - The complete 2nd order CDS would be used on the LHS. The RHS would be the bracket term: the difference between the Pade' scheme and the "old" CDS. When the CDS becomes as accurate as Pade', this term in the bracket is zero
    - Note: Forward/Backward DS could have been used instead of CDS, e.g. in time,

$$\left(\frac{\partial \phi}{\partial t}\right)_{n+1}^{r+1} = \left(\frac{\phi_{n+1} - \phi_n}{\Delta t}\right)^{r+1} + \left[\left(\frac{\partial \phi}{\partial t}\right)_{n+1}^{\text{Pade'}} - \frac{\phi_{n+1} - \phi_n}{\Delta t}\right]^r$$

Deferred-Correction Approaches, Cont'd

- Example 2 with FV methods: Higher-order Flux approximations
  - *Higher-order* flux approximations are computed with "old values" and a lower order approximation is used with "new values" (implicitly) in the linear system solver:

$$F_e = F_e^L + \left[F_e^H - F_e^L \right]^{\text{old}}$$

    where F_e is the flux. For ex., the low order approximation is a UDS or CDS

    - Convergence and stability properties are close to those of the low order implicit term since the bracket is often small compared to this implicit term
    - In addition, since bracket term is small, the iteration in the algebraic equation solver can converge to the accuracy of higher-order scheme
    - Additional numerical effort is explicit with "old values" and thus much smaller than the full implicit treatment of the higher-order terms

  - A factor can be used to produce a mixture of pure low and pure high order. This can be used to remove undesired properties, e.g. oscillations of high-order schemes

$$F_e = \omega F_e^L + (1-\omega) \left[F_e^H - F_e^L \right]^{\text{old}}$$

References and Reading Assignments Complex Geometries and Grid Generation

- Chapter 8 on "Complex Geometries" of "J. H. Ferziger and M. Peric, Computational Methods for Fluid Dynamics. Springer, NY, 3rd edition, 2002"
- Chapter 9 on "Grid Generation" of T. Cebeci, J. P. Shao, F. Kafyeke and E. Laurendeau, *Computational Fluid Dynamics for Engineers*. Springer, 2005.
- Chapter 13 on "Grid Generation" of Fletcher, Computational Techniques for Fluid Dynamics. Springer, 2003.
- Ref on Grid Generation only:
  - Thompson, J.F., Warsi Z.U.A. and C.W. Mastin, "Numerical Grid Generation, Foundations and Applications", North Holland, 1985

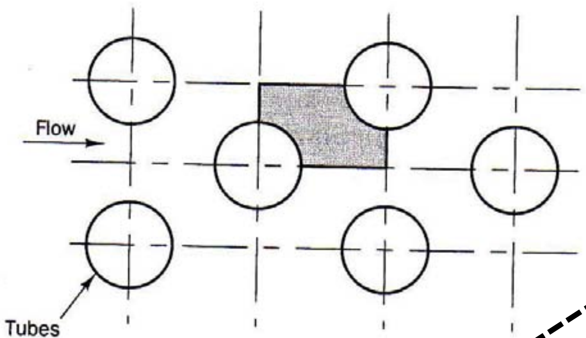
Grid Generation and Complex Geometries: Introduction

- Many flows in engineering and science involve complex geometries
- This requires some modifications of the algorithms:
  - Ultimately, properties of the numerical solver also depend on the:
    - Choice of the grid
    - Vector/tensor components (e.g. Cartesian or not)
    - Arrangement of the variables on the grid
- Different types of grids:
  - Structured grids: families of grid lines such that members of the same family do not cross each other and cross each member of other families only once
  - Advantages: simpler to program, neighbor connectivity, resultant algebraic system has a regular structure => efficient solvers
  - Disadvantages: can be used only for simple geometries, difficult to control the distribution of grid points on the domain (e.g. concentrate in specific areas)
  - Three types (names derived from the shape of the grid):
    - H-grid: a grid which can map into a rectangle
    - O-grid: one of the coordinate lines wraps around or is “endless”. One introduces an artificial cut at which the grid numbering jumps
    - C-grid: points on portions of one grid line coincide (used for body with sharp edges)

Grid Generation and Complex Geometries: Structured Grids

H-Type grids

- Example: create a grid for the flow over a heat exchanger tube bank (only part of it is shown)



- Stepwise 2D Cartesian grid
  - Number of points non constant or use masks
  - Steps at boundary introduce errors
- vs. non-orthogonal, structured grid

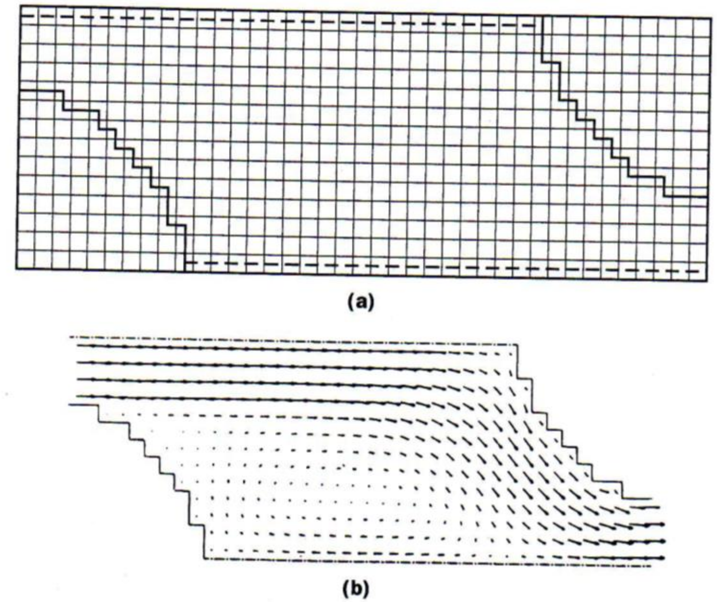


Figure 11.5 (a) Cartesian grid using an approximated profile to represent cylindrical surfaces; (b) predicted flow pattern using a 40 × 15 Cartesian grid

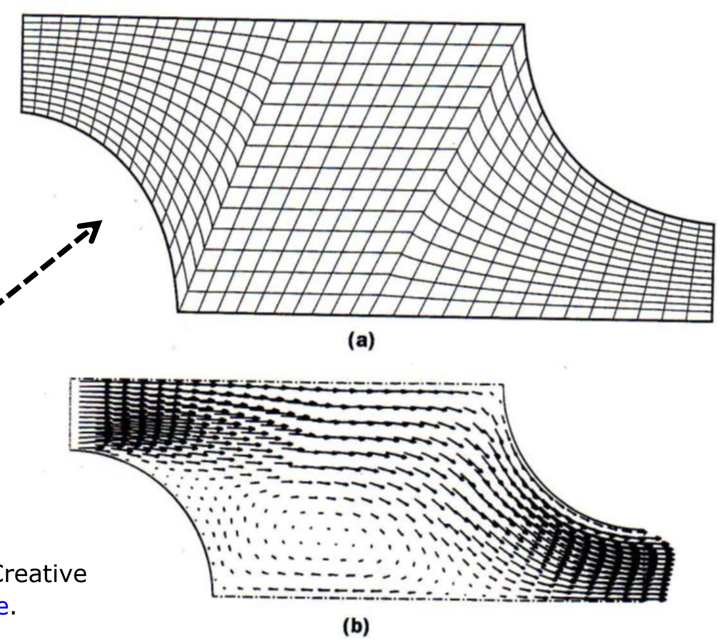


Figure 11.6 (a) Non-orthogonal body-fitted grid for the same problem; (b) predicted flow pattern using a 40 × 15 structured body-fitted grid

Grid Generation and Complex Geometries: Block-Structured Grids

- Grids for which there is one or more level subdivisions of the solution domain
  - Can match at interfaces or not
  - Can overlap or not
- Block structured grids with overlapping blocks are sometimes called "*composite*" or "*Chimera*" grids
  - Interpolation used from one grid to the other
  - Useful for moving bodies (one block attached to it and the other is a stagnant grid)
- Special case: Embedded or Nested grids, which can still use different dynamics at different scales

Grid with 3 Blocks, with an O-Type grid (for coordinates around the cylinder)

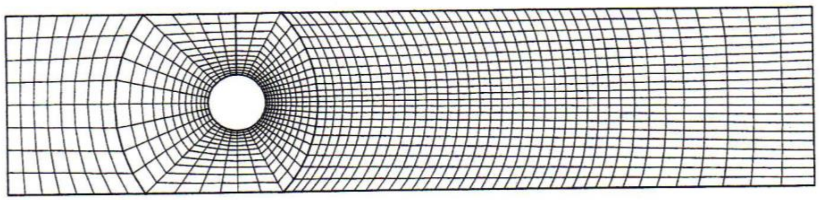


Fig. 2.2. Example of a 2D block-structured grid which matches at interfaces, used to calculate flow around a cylinder in a channel

Grid with 5 blocks, including H-Type and C-Type, and non-matching interface:

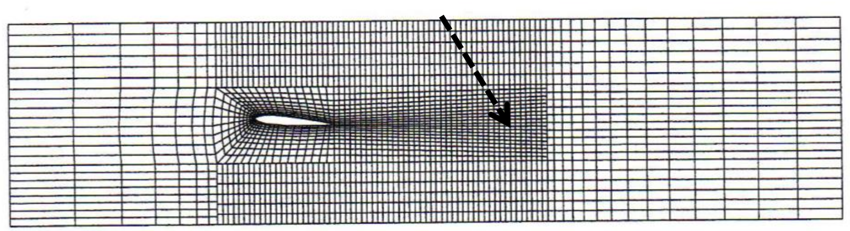


Fig. 2.3. Example of a 2D block-structured grid which does not match at interfaces, designed for calculation of flow around a hydrofoil under a water surface

"composite" or "Chimera" Grid

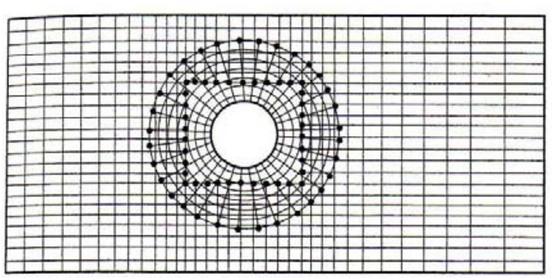


Fig. 2.4. A composite 2D grid, used to calculate flow around a cylinder in a channel

Grid Generation and Complex Geometries:

Other examples of Block-structured Grids

Figure 11.9 Block-structured mesh for a transonic aerofoil. Inset shows cut cells near aerofoil surface. Also note additional grid refinement in the flow region to capture a shock above the aerofoil
Source: Haselbacher (1999)

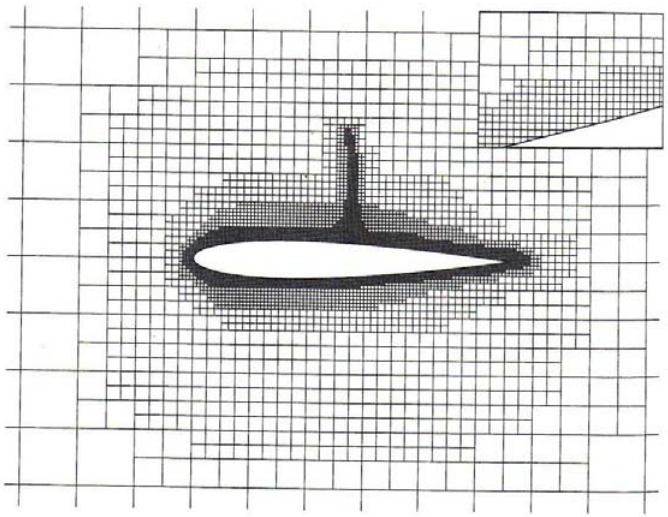
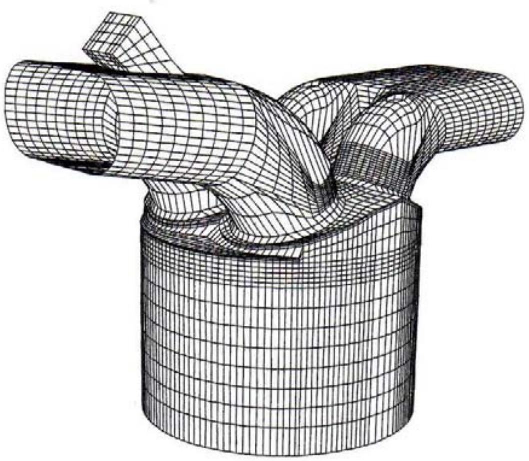


Figure 1.7 in Haselbacher, Andreas C. "A grid-transparent numerical method for compressible viscous flows on mixed unstructured grids." PhD diss., Loughborough University, 1999.

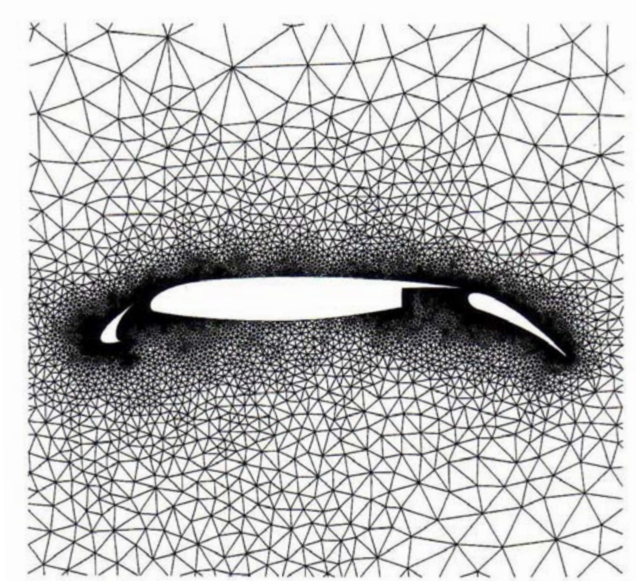
Figure 11.10 Block-structured mesh arrangement for an engine geometry, including inlet and exhaust ports, used in engine simulations with KIVA-3V



Grid Generation and Complex Geometries: Unstructured Grids

- For very complex geometries, most flexible grid is one that can fit any physical domain: i.e. unstructured
- Can be used with any discretization scheme, but best adapted to FV and FE methods
- Grid most often made of:
  - Triangles or quadrilaterals in 2D
  - Tetrahedra or hexahedra in 3D
- Advantages
  - Unstructured grid can be made orthogonal if needed
  - Aspect ratio easily controlled
  - Grid may be easily refined
- Disadvantages:
  - Irregularity of the data structure: nodes locations and neighbor connections need to be specified explicitly
  - The matrix to be solved is not regular anymore and the size of the band needs to be controlled by node ordering

Figure 11.11 A triangular grid for a three-element aerofoil
Source: Haselbacher (1999)



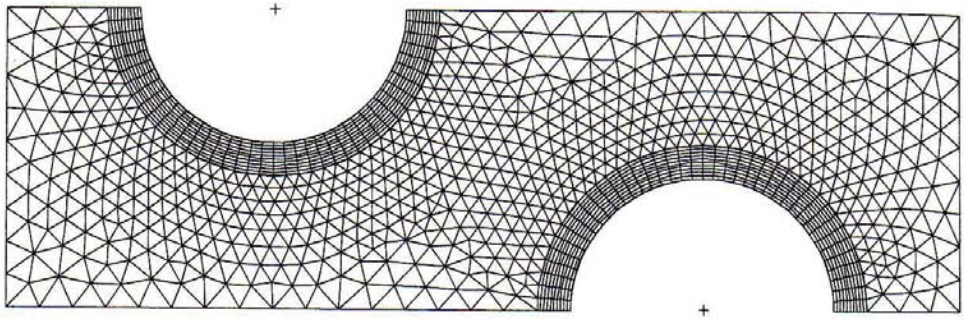


Figure 11.12 An example of an unstructured mesh with mixed elements

Unstructured Grids Examples: Multi-element grids

- For FV methods, what matters is the angle between the vector normal to the cell surface and the line connecting the CV centers ⇒
  - 2D equilateral triangles are equivalent to a 2D orthogonal grid
- Cell topology is important:
  - If cell faces parallel, remember that certain terms in Taylor expansion can cancel ⇒ higher accuracy
  - They nearly cancel if topology close to parallel
- Ratio of cells' sizes should be smooth
- Generation of triangles or tetrahedra is easier and can be automated, but lower accuracy
- Hence, more regular grid (prisms, quadrilaterals or hexahedra) often used near boundary where solution often vary rapidly

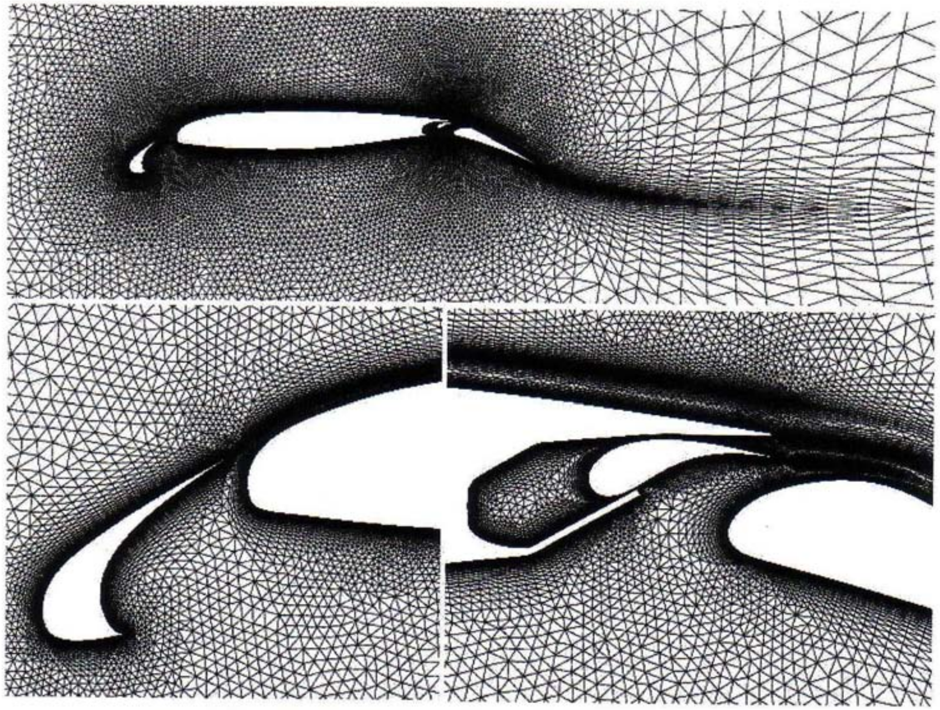


Fig. 9.16. 2D Unstructured grid for Navier–Stokes computations of a multi-element airfoil generated with the hybrid advancing front Delaunay method of Mavriplis [6].

Complex Geometries: The choice of velocity (vector) components

- Cartesian (used in this course)
  - With FD, one only needs to employ modified equations to take into account of non-orthogonal coordinates (change of derivatives due to change of spatial coordinates from Cartesian to non-orthogonal)
  - In FV methods, normally, no need for coordinate transformations in the PDEs: a local coordinate transformation can be used for the gradients normal to the cell faces
- Grid-oriented:
  - Non-conservative source terms appear in the equations (they account for the re-distribution of momentum between the components)
  - For example, in polar-cylindrical coordinates, in the momentum equations:
    - Apparent centrifugal force and apparent Coriolis force

Complex Geometries: The choice of variable arrangement

- Staggered arrangements
  - Improves coupling $u \leftrightarrow p$
  - For Cartesian components when grid lines change by 90 degrees, the velocity component stored at the cell face makes no contribution to the mass flux through that face
  - Difficult to use Cartesian components in these cases
  - Hence, for non-orthogonal grids, grid-oriented velocity components often used
- Collocated arrangements (mostly used here)
  - The simplest one: all variables share the same CV
  - Requires more interpolation

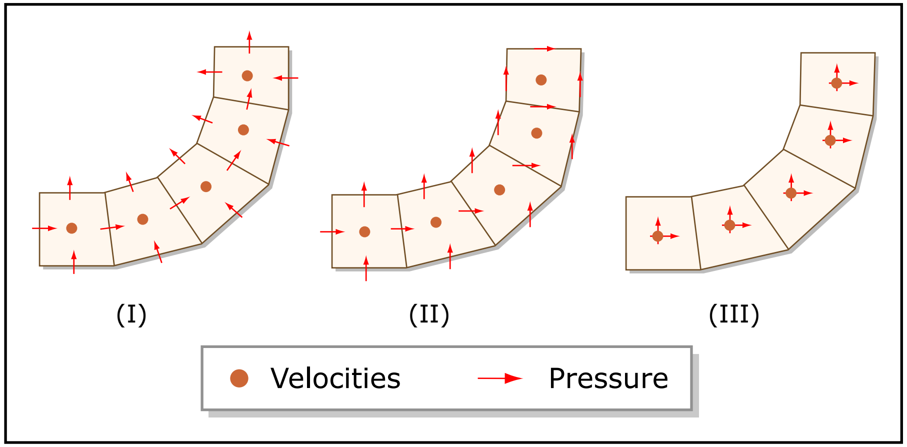


Image by MIT OpenCourseWare.

Variable arrangements on a non-orthogonal grid. Illustrated are a staggered arrangement with (i) contravarient velocity components and (ii) Cartesian velocity components, and (iii) a colocated arrangement with Cartesian velocity components.

Classes of Grid Generation

- An arrangement of discrete set of grid points or cells needs to be generated for the numerical solution of PDEs (fluid conservation equations)
  - Finite volume methods:
    - Can be applied to uniform and non-uniform grids
  - Finite difference methods:
    - Require a coordinate transformation to map the irregular grid in the physical spatial domain to a regular one in the computational domain
    - Difficult to do this in complex 3D spatial geometries
    - So far, only used with structured grid (could be used with unstructured grids with polynomials ϕ defining the shape of ϕ around a grid point)
- Three major classes of (structured) grid generation: i) algebraic methods, ii) differential equation methods and iii) conformal mapping methods
- Grid generation and solving PDE can be independent
  - A numerical (flow) solver can in principle be developed independently of the grid
  - A grid generator then gives the metrics (weights) and the one-to-one correspondence between the spatial-grid and computational-grid

Grid Generation: Basic Concepts for Structured Grids

- Structured Grids (includes curvilinear or non-orthogonal grids)
  - Often utilized with FD schemes
  - Methods based on coordinate transformations
- Consider irregular shaped physical domain (x, y) in Cartesian coordinates and determine its mapping to the computational domain in the (ξ, η) Cartesian coordinates
  - Increase ξ *or* η monotonically in physical domain along "curved lines"
  - Coordinate lines of the same family do not cross
  - Lines of different family don't cross more than once
  - Physical grid refined where large errors are expected

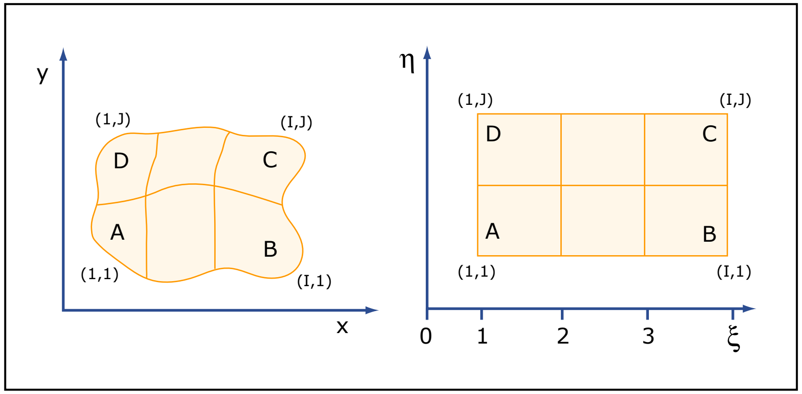


Image by MIT OpenCourseWare.

A simply-connected irregular shape in the physical plane is mapped as a rectangle in the computational plane.

  - Mapped (computational) region has a rectangular shape:
    - Coordinates (ξ, η) can vary from 1 to (I, J), with mesh sizes taken equal to 1
  - Boundaries are mapped to boundaries

Grid Generation:
Basic Concepts for Structured Grids, Cont'd

- The example just shown was the mapping of an irregular, simply connected, region into a rectangle.
- Other configurations are of course possible

– For example, a L-shape domain can be mapped into:

  – a regular L-shape

  – or into a rectangular shape

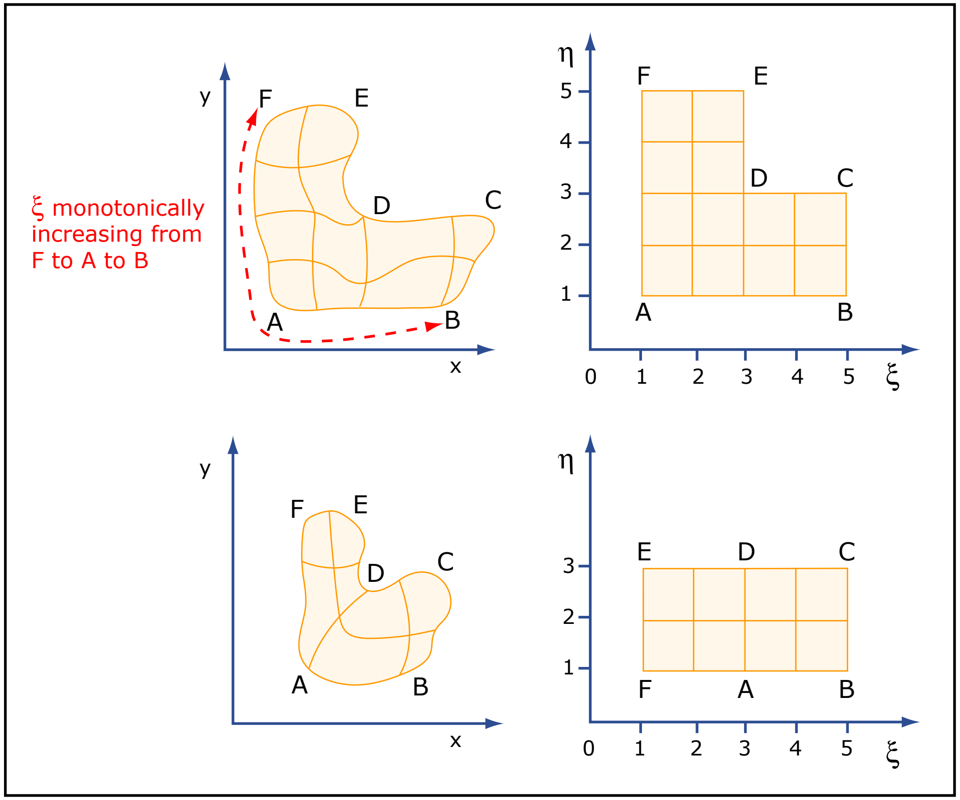


Image by MIT OpenCourseWare.

Grid Generation for Structured Grids: **Stretched Grids**

- Consider a viscous flow solution on a given body, where the velocity varies rapidly near the surface of the body (Boundary Layer)
- For efficient computation, a finer grid near the body and coarser grid away from the body is effective (aims to maintain constant accuracy)
- Possible coordinate transformation: a scaling "$\eta = \log(y)$" ↔ "$y = \exp(\eta)$"

$$\xi = x \qquad \eta = 1 - \frac{\ln[A(y)]}{\ln B}$$

where $A(y) = \dfrac{\beta + (1 - y/h)}{\beta - (1 - y/h)}$ and $B = \dfrac{\beta + 1}{\beta - 1}$

The parameter β $(1 < \beta < \infty)$ is the stretching parameter. As β gets close to 1, more grid points are clustered to the wall in the physical domain.

- Inverse transformation is needed to map solutions back from ξ, η domain:

$$x = \xi \qquad \frac{y}{h} = \frac{(\beta + 1) - (\beta - 1)B^{1-\eta}}{1 + B^{1-\eta}}$$

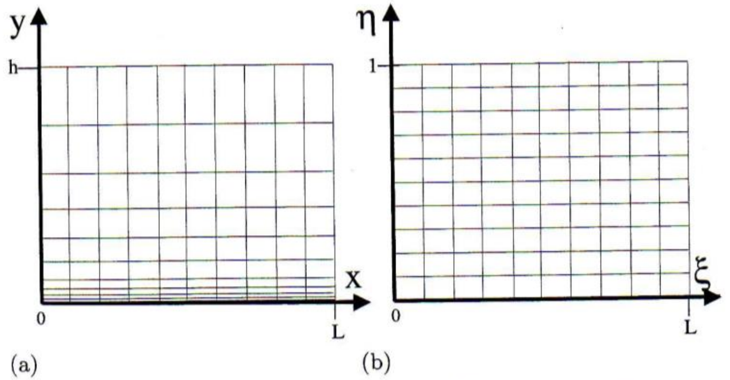


Fig. 9.4. One-dimensional stretching transformation. (**a**) Physical plane, (**b**) computational plane.

Grid Generation for Structured Grids: **Stretched Grids, Cont'd**

- How do the conservation equations change?
- Consider the continuity equation for steady state flow in physical (x, y) space:

$$\nabla.(\rho\vec{v}) = 0 \quad \Rightarrow \quad \frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} = 0$$

- In the computational plane, this equation becomes (chain rule)

$$\left.\begin{aligned} \frac{\partial \rho u}{\partial x} &= \frac{\partial \rho u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial \rho u}{\partial \eta}\frac{\partial \eta}{\partial x} \\ \frac{\partial \rho u}{\partial y} &= \frac{\partial \rho v}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial \rho v}{\partial \eta}\frac{\partial \eta}{\partial y} \end{aligned}\right\} \Rightarrow \frac{\partial \rho u}{\partial \xi}\xi_x + \frac{\partial \rho u}{\partial \eta}\eta_x + \frac{\partial \rho v}{\partial \xi}\xi_y + \frac{\partial \rho v}{\partial \eta}\eta_y = 0$$

- For our stretching transformation, one obtains:

$$\xi_x = 1, \quad \eta_x = 0, \quad \xi_y = 0, \quad \eta_y = \frac{2\beta}{h\ln(B)}\frac{1}{\beta^2 - (1 - y/h)^2}$$

- Therefore, the continuity equation becomes:

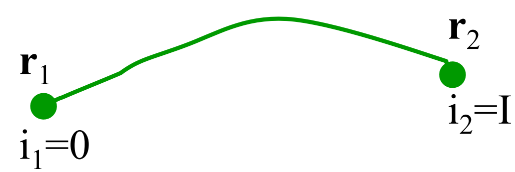
$$\frac{\partial \rho u}{\partial \xi} + \frac{\partial \rho v}{\partial \eta}\eta_y = 0$$

  – This equation can be solved on a uniform grid (slightly more complicated eqn. system), and the solution mapped back to the physical domain using the inverse transform

Algebraic Methods: Transfinite Interpolation

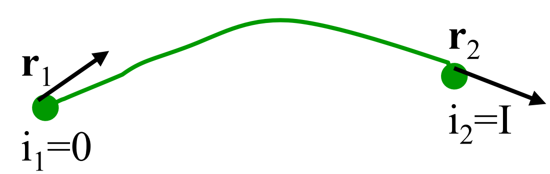
- Multi-directional interpolation (Transfinite Interpolation)
  - To generate algebraic grids within more complex domains or around more complex configurations, multi-directional interpolations can be used
    - They consist of a suite of unidirectional interpolations
- Unidirectional Interpolations (1D curve)
  - The Cartesian coordinate vector of any point on a curve **r**(x,y) is obtained as an interpolation between given points that lie on the boundary curves
  - How to interpolate? the regulars:
    - Lagrange Polynomials: match function values

$$\vec{r}(i) = \sum_{k=0}^{n} L_k(i)\, \vec{r}_k \quad \text{with} \quad L_k(i) = \prod_{j=0, j\neq k}^{n} \frac{i - i_j}{i_k - i_j},$$



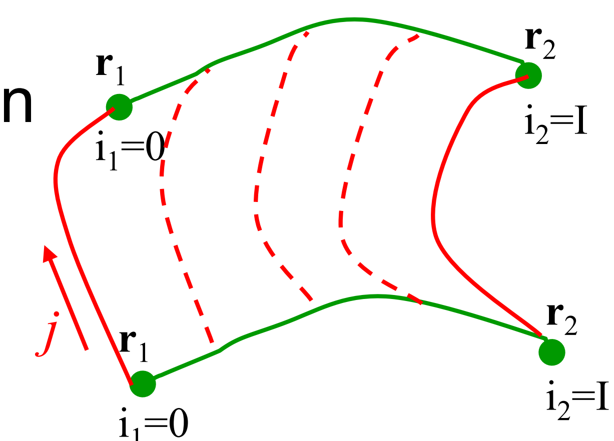
    - Hermite Polynomials: match both function and 1st derivative values

$$\vec{r}(i) = \sum_{k=1}^{n} a_k(i)\, \vec{r}_k + \sum_{k=1}^{m} b_k(i)\, \vec{r}\,'_k$$



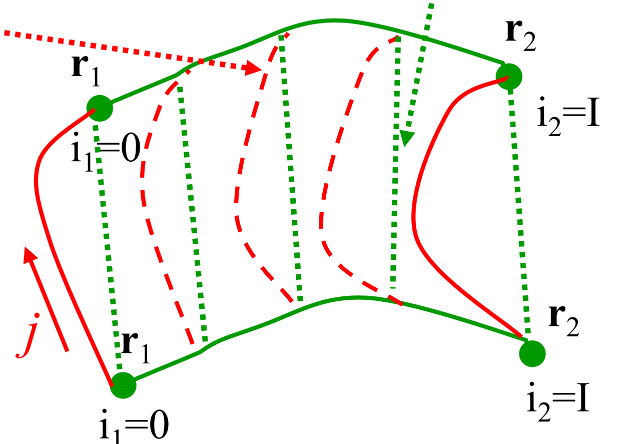
Algebraic Methods: Transfinite Interpolation, Cont'd

- Unidirectional Interpolations (1D curve), Cont'd
  - Lagrange and Hermite Polynomials fit a single polynomial from one boundary to the next => for long boundaries, oscillations may occur
  - Alternative 1: use set of lower order polynomials to form a piece-wise continuous interpolation:
    - Spline interpolation (match as many derivatives as possible at interior point junctions), Tension-spline (more localized curvature) and B-splines (allows local modification of the interpolation)
  - Alternative 2: use interpolation functions that are not polynomials, usually "stretching functions": exp, tanh, sinh, etc
- Multi-directional or Transfinite Interpolation
  - Extends 1D results to 2D or 3D by successive applications of 1D interpolations
  - For example, *i* then *j*.



Grid Generation for Structured Grids: Algebraic Methods: Transfinite Interpolation, Cont'd

- Multi-directional or Transfinite Interpolation, Cont'd
  - In 2D, the transfinite interpolation can be implemented as follows
    - Interpolate position vectors $\mathbf{r}$ in i-direction => leads to points $\mathbf{f}_1 = \mathcal{I}_i(\mathbf{r})$ and i-lines
    - Evaluate the difference between this result and $\mathbf{r}$ on the j-lines that will be used in the j-interpolation (e.g. 2 differences: one with curve $i=0$ & one with $i=\mathrm{I}$): $\mathbf{r} - \mathbf{f}_1$
    - Interpolation of the discrepancy in the j-direction: $\mathbf{f}_2 = \mathcal{I}_j(\mathbf{r} - \mathbf{f}_1)$
    - Addition of the results of this j-interpolation to the results of the i-interpolation: $\mathbf{r}\,(i, j) = \mathbf{f}_1 + \mathbf{f}_2$
- Of course, Lagrange, Hermite Polynomials, Spline and non-polynomial (stretching) functions can be used for transfinite interpolations
- In 2D, inputs to program are 4 boundaries
- Issues: Propagates discontinuities in the interior and grid lines can overlap in some situations
- => needs to be refined by grid generator solving a PDE



Grid Generation for Structured Grids: Algebraic Methods: Transfinite Interpolation, Cont'd

- Examples:

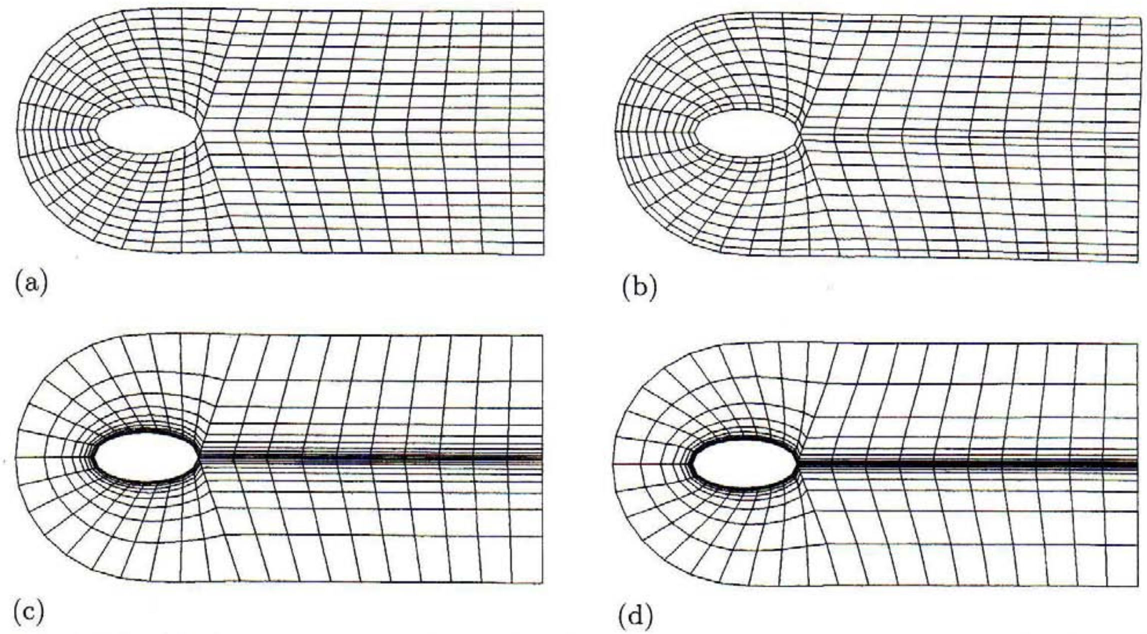


Fig. 9.12. (a) C-grid around ellipse: Unidirectional Lagrange Interpolation, (b) C-grid around ellipse: Unidirectional Hermite Interpolation, (c) C-grid around ellipse: Unidirectional Lagrange Interpolation with Hyperbolic Tangent Spacing, (d) C-grid around ellipse: Unidirectional Hermite Interpolation with Hyperbolic Tangent Spacing.

2.29 Numerical Fluid Mechanics
Spring 2015