Numerical Simulation Methods

Part 2 - Applied Concepts

Lecture 14: Advanced Implicit Schemes

Roadmap

Applied Concepts

- Introduction to meshing
- The finite volume method
- Jameson's scheme
- Solution storage approaches and their implications
- **Today: Advanced implicit methods**
  - **Matrix bandwidth & computational cost**
  - **Alternating direction implicit (ADI)**
  - **Approximate factorisation (AF)**
  - **Factored unfactored (FUN)**
- Next lecture: Introduction to computer hardware and high performance computing
- Parallel decomposition and efficiency

Implicit schemes - Recap (1)

Consider a scalar equation in 2-D for simplicity.

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} = 0 \tag{1}$$

For simplicity, assume we solve the equation on a cartesian mesh of constant spacing (Δx and Δy are constant) using a finite-difference implementation (finite-volume schemes follow similarly).
The backward time, centred space implicit method (see earlier notes) gives

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} + a\frac{u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}}{2\Delta x} + b\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} = 0. \tag{2}$$

Implicit schemes - Recap (2)

Write our BTCS scheme in terms of differences:

$$\frac{\Delta u^n_{i,j}}{\Delta t} + a\frac{u^n_{i+1,j} + \Delta u^n_{i+1,j} - u^n_{i-1,j} - \Delta u^n_{i-1,j}}{2\Delta x} + b\frac{u^n_{i,j+1} + \Delta u^n_{i,j+1} - u^n_{i,j-1} - \Delta u^n_{i,j-1}}{2\Delta y} = 0 \quad (3)$$

where:

$$\Delta u^n_{i,j} = u^{n+1}_{i,j} - u^n_{i,j} \quad (4)$$

Rearrange as normal, to place known values on the right, and unknowns on the left,

$$\begin{aligned} -a\frac{\Delta t}{2\Delta x}\Delta u^n_{i-1,j} - b\frac{\Delta t}{2\Delta y}\Delta u^n_{i,j-1} + \Delta u^n_{i,j} + b\frac{\Delta t}{2\Delta y}\Delta u^n_{i,j+1} + a\frac{\Delta t}{2\Delta x}\Delta u^n_{i+1,j} = \\ -\Delta t\left(a\frac{u^n_{i+1,j} - u^n_{i-1,j}}{2\Delta x} + b\frac{u^n_{i,j+1} - u^n_{i,j-1}}{2\Delta y}\right) \end{aligned} \quad (5)$$

Implicit schemes - Recap (3)

We have our BTCS implicit scheme:

$$-a\frac{\Delta t}{2\Delta x}\Delta u^n_{i-1,j} - b\frac{\Delta t}{2\Delta y}\Delta u^n_{i,j-1} + \Delta u^n_{i,j} + b\frac{\Delta t}{2\Delta y}\Delta u^n_{i,j+1} + a\frac{\Delta t}{2\Delta x}\Delta u^n_{i+1,j} = \\ -\Delta t\left(a\frac{u^n_{i+1,j} - u^n_{i-1,j}}{2\Delta x} + b\frac{u^n_{i,j+1} - u^n_{i,j-1}}{2\Delta y}\right) \tag{6}$$

- Recall, the right-hand-side is the **residual**
- The implicit BTCS scheme applied to the 2-D scalar equation results in a matrix equation where each row has five non-zero elements (pentadiagonal) - very expensive to solve.
- For non-scalar equations (*i.e.* systems of equations) the matrix elements become blocks.
  - For example, a 2-D system of **four equations** would results in a block pentadiagonal matrix, where each of the five blocks is a 4×4 matrix - Very, very expensive !

Implicit schemes - Computational cost (1)

To exactly solve an $n \times n$ matrix with bandwidth b, then the number of operations required (multiplications, additions, subtractions and divisions) is given by:

$$N_{op} = \frac{1}{12}(b^2 - 1)(3n - b) \tag{7}$$

Implicit schemes - Computational cost (1)

To exactly solve an $n \times n$ matrix with bandwidth b, then the number of operations required (multiplications, additions, subtractions and divisions) is given by:

$$N_{op} = \frac{1}{12}(b^2 - 1)(3n - b) \tag{7}$$

- Hence, going from a tri- to pentadiagonal matrix for a realistic mesh dimension means approximately three times the number of operations.
- Similarly, going to three dimensions results in a septadiagonal matrix, and approximately six times the work.
- Direct solution of the matrix equation is not often used even for scalar equations. For systems of equations, almost never.

Implicit schemes - Computational cost (2)

Consider 3-D mesh with 250,000 points. $N_{op} = \frac{1}{12}(b^2 - 1)(3n - b)$

Scalar equation -
$n = 250,000$
$b = 3 \Rightarrow 0.50 \times 10^6$ operations
$b = 5 \Rightarrow 1.50 \times 10^6$ operations
$b = 7 \Rightarrow 3.00 \times 10^6$ operations

System of equations, Euler or N-S five equations
$n = 250,000 \times 5$
$b = 15 \Rightarrow 70 \times 10^6$ operations
$b = 25 \Rightarrow 200 \times 10^6$ operations
$b = 35 \Rightarrow 400 \times 10^6$ operations

Implicit schemes - Computational cost (2)

Consider 3-D mesh with 250,000 points. $N_{op} = \frac{1}{12}(b^2 - 1)(3n - b)$

Scalar equation -
$n = 250,000$
$b = 3 \Rightarrow 0.50 \times 10^6$ operations
$b = 5 \Rightarrow 1.50 \times 10^6$ operations
$b = 7 \Rightarrow 3.00 \times 10^6$ operations

System of equations, Euler or N-S five equations
$n = 250,000 \times 5$
$b = 15 \Rightarrow 70 \times 10^6$ operations
$b = 25 \Rightarrow 200 \times 10^6$ operations
$b = 35 \Rightarrow 400 \times 10^6$ operations

However, this analysis is too optimistic because **bandwidth does not equal the number of non-zero elements in each row**.

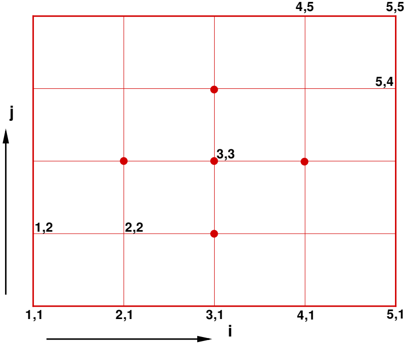
Bandwidth of a matrix

The largest number of columns separating the first and last non-zero elements in any row

Matrix bandwidth example (1)

For example consider a 5×5 mesh, and the points of influence in a 2D finite-difference stencil. Point i, j uses neighbours $i-1, j$, $i+1, j$, $i, j-1$, and $i, j+1$, so consider the row in the matrix corresponding to point 3,3.

$$
\begin{bmatrix}
\text{.....5 elements for }_{1,1}\text{......} \\
\text{.....5 elements for }_{2,1}\text{......} \\
\text{.....5 elements for }_{3,1}\text{......} \\
. \\
\text{.....5 elements for }_{3,3}\text{......} \\
. \\
\text{.....5 elements for }_{3,5}\text{......} \\
\text{.....5 elements for }_{4,5}\text{......} \\
\text{.....5 elements for }_{5,5}\text{......}
\end{bmatrix}
.
\begin{bmatrix}
\Delta u^n_{1,1} \\ \Delta u^n_{2,1} \\ \Delta u^n_{3,1} \\ . \\ \Delta u^n_{3,3} \\ . \\ \Delta u^n_{3,5} \\ \Delta u^n_{4,5} \\ \Delta u^n_{5,5}
\end{bmatrix}
=
\begin{bmatrix}
RHS^n_{1,1} \\ RHS^n_{2,1} \\ RHS^n_{3,1} \\ . \\ RHS^n_{3,3} \\ . \\ RHS^n_{3,5} \\ RHS^n_{4,5} \\ RHS^n_{5,5}
\end{bmatrix}
$$



Matrix bandwidth example (2)

$$
\begin{bmatrix} \vdots \\ 0,0,0,0,0,0,0,*,0,0,0,*,*,*,0,0,0,*,0,0,0,0,0,0,0 \\ \vdots \end{bmatrix}
\cdot
\begin{bmatrix}
\Delta u^n_{1,1} \\ \Delta u^n_{2,1} \\ \Delta u^n_{3,1} \\ \Delta u^n_{4,1} \\ \Delta u^n_{5,1} \\
\Delta u^n_{1,2} \\ \Delta u^n_{2,2} \\ \boldsymbol{\Delta u^n_{3,2}} \\ \Delta u^n_{4,2} \\ \Delta u^n_{5,2} \\
\Delta u^n_{1,3} \\ \boldsymbol{\Delta u^n_{2,3}} \\ \boldsymbol{\Delta u^n_{3,3}} \\ \boldsymbol{\Delta u^n_{4,3}} \\ \Delta u^n_{5,3} \\
\Delta u^n_{1,4} \\ \Delta u^n_{2,4} \\ \boldsymbol{\Delta u^n_{3,4}} \\ \Delta u^n_{4,4} \\ \Delta u^n_{5,4} \\
\Delta u^n_{1,5} \\ \Delta u^n_{2,5} \\ \Delta u^n_{3,5} \\ \Delta u^n_{4,5} \\ \Delta u^n_{5,5}
\end{bmatrix}
=
\begin{bmatrix} \vdots \\ RHS^n_{3,3} \\ \vdots \end{bmatrix}
$$

Now consider in detail the row for point 3,3.

$*$ represents a non-zero element.

Matrix bandwidth example (2)

$$
\begin{bmatrix}
\vdots \\
0,0,0,0,0,0,0,*,0,0,0,*,*,*,0,0,0,*,0,0,0,0,0,0,0 \\
\vdots
\end{bmatrix}
\cdot
\begin{bmatrix}
\Delta u^n_{1,1} \\ \Delta u^n_{2,1} \\ \Delta u^n_{3,1} \\ \Delta u^n_{4,1} \\ \Delta u^n_{5,1} \\
\Delta u^n_{1,2} \\ \Delta u^n_{2,2} \\ \boldsymbol{\Delta u^n_{3,2}} \\ \Delta u^n_{4,2} \\ \Delta u^n_{5,2} \\
\Delta u^n_{1,3} \\ \boldsymbol{\Delta u^n_{2,3}} \\ \boldsymbol{\Delta u^n_{3,3}} \\ \boldsymbol{\Delta u^n_{4,3}} \\ \Delta u^n_{5,3} \\
\Delta u^n_{1,4} \\ \Delta u^n_{2,4} \\ \boldsymbol{\Delta u^n_{3,4}} \\ \Delta u^n_{4,4} \\ \Delta u^n_{5,4} \\
\Delta u^n_{1,5} \\ \Delta u^n_{2,5} \\ \Delta u^n_{3,5} \\ \Delta u^n_{4,5} \\ \Delta u^n_{5,5}
\end{bmatrix}
=
\begin{bmatrix}
\vdots \\
RHS^n_{3,3} \\
\vdots
\end{bmatrix}
$$

Now consider in detail the row for point 3,3.

$*$ represents a non-zero element.

For a general 2-D mesh, $NI \times NJ$, the bandwidth is actually $2NI + 1$.

Implicit schemes - Computational cost (3)

- The dimension of the implicit system n is the number of grid cells times the number of equations: $NI \times NJ \times NK \times N_{\text{eqn}}$
- The cost of exactly solving the matrix system is proportional to the matrix dimension n times the square of matrix bandwidth b
- BUT the matrix bandwidth b is also a function of the grid dimensions, since we must store grid points in a 1-D vector

Implicit schemes - Computational cost (3)

- The dimension of the implicit system n is the number of grid cells times the number of equations: $NI \times NJ \times NK \times N_{\text{eqn}}$
- The cost of exactly solving the matrix system is proportional to the matrix dimension n times the square of matrix bandwidth b
- BUT the matrix bandwidth b is also a function of the grid dimensions, since we must store grid points in a 1-D vector

Hence, exactly solving the implicit system very quickly becomes too computationally expensive to consider

BUT do we need to solve the matrix exactly?

- Can we solve it approximately using cheaper methods?

Alternating Direction Implicit (ADI) (1)

This approach obtains the change in the solution over a time-step in multiple stages. (Also called the method of fractional steps.)

Consider again, the BTCS scheme for the 2-D scalar equation: in full.

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} + a\frac{u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}}{2\Delta x} + b\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} = 0. \tag{8}$$

First, a 'half-step' of $\frac{\Delta t}{2}$ is used to solve

$$\frac{u_{i,j}^{n+\frac{1}{2}} - u_{i,j}^{n}}{\Delta t/2} + a\frac{u_{i+1,j}^{n+\frac{1}{2}} - u_{i-1,j}^{n+\frac{1}{2}}}{2\Delta x} + b\frac{u_{i,j+1}^{n} - u_{i,j-1}^{n}}{2\Delta y} = 0. \tag{9}$$

Notice only the x derivative is implicit (the y derivative is at the known time-level); this means there are only three unknowns per row, and hence a tridiagonal matrix .

Alternating Direction Implicit (ADI) (2)

After the first fractional step (implicit in x), the implicit time step is completed by solving a second system, now only implicit in y (since the x derivative is at the known fractional time-level):

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n+\frac{1}{2}}}{\Delta t/2} + a\frac{u_{i+1,j}^{n+\frac{1}{2}} - u_{i-1,j}^{n+\frac{1}{2}}}{2\Delta x} + b\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} = 0. \tag{10}$$

Alternating Direction Implicit (ADI) (2)

After the first fractional step (implicit in x), the implicit time step is completed by solving a second system, now only implicit in y (since the x derivative is at the known fractional time-level):

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n+\frac{1}{2}}}{\Delta t/2} + a\frac{u_{i+1,j}^{n+\frac{1}{2}} - u_{i-1,j}^{n+\frac{1}{2}}}{2\Delta x} + b\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} = 0. \tag{10}$$

- We have obtained an approximate 2-D solution for the price of two tridiagonal matrix solutions.
- Importantly, this is **much less computational work** than solving the full matrix all in one go.

Alternating Direction Implicit (ADI) (2)

After the first fractional step (implicit in x), the implicit time step is completed by solving a second system, now only implicit in y (since the x derivative is at the known fractional time-level):

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n+\frac{1}{2}}}{\Delta t/2} + a\frac{u_{i+1,j}^{n+\frac{1}{2}} - u_{i-1,j}^{n+\frac{1}{2}}}{2\Delta x} + b\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} = 0. \tag{10}$$

- We have obtained an approximate 2-D solution for the price of two tridiagonal matrix solutions.
- Importantly, this is **much less computational work** than solving the full matrix all in one go.

Alternating Direction Implicit (AD)

Solve the implicit system by using implicit differences **in each direction separately** - we alternate between $x-$ and $y-$ differences.

Alternating Direction Implicit (ADI) (3) - Cost savings

Let's now compare the cost between the full system solution and using ADI in 2-D. For a full solution of an $NI \times NJ$ mesh we would store matrix with the index with smallest number first. Say NI smallest, would have:

$$N_{op} = \frac{1}{12}(b^2 - 1)(3n - b) = \frac{1}{12}((2NI + 1)^2 - 1)(3(NI.NJ) - (2NI + 1))$$

For an ADI, we would have

$$N_{op} = \frac{1}{12}(3^2 - 1)(3(NI.NJ) - 3) \qquad (11)$$

for i direction. Then we would store the matrix with j varying first, and so followed by:

$$N_{op} = \frac{1}{12}(3^2 - 1)(3(NI.NJ) - 3) \qquad (12)$$

Let's now compare the cost between the full system solution and using ADI in 2-D. For a full solution of an $NI \times NJ$ mesh we would store matrix with the index with smallest number first. Say NI smallest, would have:

$$N_{op} = \frac{1}{12}(b^2 - 1)(3n - b) = \frac{1}{12}((2NI + 1)^2 - 1)(3(NI.NJ) - (2NI + 1))$$

For an ADI, we would have

$$N_{op} = \frac{1}{12}(3^2 - 1)(3(NI.NJ) - 3) \qquad (11)$$

for i direction. Then we would store the matrix with j varying first, and so followed by:

$$N_{op} = \frac{1}{12}(3^2 - 1)(3(NI.NJ) - 3) \qquad (12)$$

Say $NI = NJ = 100$.
Full: 100323300.
ADI: 39996 (19998 $\times$ 2)
$\Rightarrow$ 2508$\times$ faster !

We get big savings by reducing the matrix bandwidth

Approximate Factorisation (AF) - Formulation

Again consider the left-hand side of the equation,

$$-a\frac{\Delta t}{2\Delta x}\Delta u^n_{i-1,j} - b\frac{\Delta t}{2\Delta y}\Delta u^n_{i,j-1} + \Delta u^n_{i,j} + b\frac{\Delta t}{2\Delta y}\Delta u^n_{i,j+1} + a\frac{\Delta t}{2\Delta x}\Delta u^n_{i+1,j}$$

This can be written in terms of difference operators,

$$\left(1 + a\frac{\Delta t}{2\Delta x}\delta_x + b\frac{\Delta t}{2\Delta y}\delta_y\right)\Delta u^n_{i,j} \quad \text{where: } \delta_x = ()_{i+1,j} - ()_{i-1,j} \quad \text{and: } \delta_y = ()_{i,j+1} - ()_{i,j-1}$$

This is approximated by a factored form,

$$\left(1 + a\frac{\Delta t}{2\Delta x}\delta_x\right)\left(1 + b\frac{\Delta t}{2\Delta y}\delta_y\right)\Delta u^n_{i,j}$$

Approximate Factorisation (AF) - Error

The difference between the factored and unfactored form of the equations is a term that is no larger than the truncation error in deriving the unfactored form. Multiply out, and the difference is

$$\left(a\frac{\Delta t}{2\Delta x}\delta_x b\frac{\Delta t}{2\Delta y}\delta_y\right)\Delta u_{i,j}^n = ab\frac{\Delta t^2}{4\Delta x\Delta y}\delta_x\delta_y\Delta u_{i,j}^n$$

where

$$\delta_x\delta_y\Delta u_{i,j}^n = \delta_x(\Delta u_{i,j+1}^n - \Delta u_{i,j-1}^n)$$

$$= \Delta u_{i+1,j+1}^n - \Delta u_{i+1,j-1}^n - \Delta u_{i-1,j+1}^n + \Delta u_{i-1,j-1}^n$$

The factored form of the method has the same formal accuracy as the original unfactored form. If the set of equations are assembled for each point the left-hand side now consists of two tridiagonal matrices, one for the x- derivatives and one for the y- derivatives. It's much quicker to solve two tridiagonal matrices than one containing all the non-zeros.

Approximate Factorisation (AF) - Solution process

Set

$$\mathbf{A} = \left(1 + a\frac{\Delta t}{2\Delta x}\delta_x\right), \qquad \mathbf{B} = \left(1 + b\frac{\Delta t}{2\Delta y}\delta_y\right)$$

then the factorised implicit system is written as,

$$\mathbf{AB}\Delta\underline{\mathbf{u}} = \underline{\mathbf{R}}$$

Approximate Factorisation (AF) - Solution process

Set

$$\mathbf{A} = \left(1 + a\frac{\Delta t}{2\Delta x}\delta_x\right), \qquad \mathbf{B} = \left(1 + b\frac{\Delta t}{2\Delta y}\delta_y\right)$$

then the factorised implicit system is written as,

$$\mathbf{AB}\Delta\underline{\mathbf{u}} = \underline{\mathbf{R}}$$

This is solved in two stages,

$$\mathbf{A}\Delta\underline{\mathbf{u}}' = \underline{\mathbf{R}}$$

followed by

$$\mathbf{B}\Delta\underline{\mathbf{u}} = \Delta\underline{\mathbf{u}}'$$

Approximate Factorisation (AF) - Solution process

Set

$$\mathbf{A} = \left(1 + a\frac{\Delta t}{2\Delta x}\delta_x\right), \qquad \mathbf{B} = \left(1 + b\frac{\Delta t}{2\Delta y}\delta_y\right)$$

then the factorised implicit system is written as,

$$\mathbf{AB}\Delta\underline{\mathbf{u}} = \underline{\mathbf{R}}$$

This is solved in two stages,

$$\mathbf{A}\Delta\underline{\mathbf{u}}' = \underline{\mathbf{R}}$$

followed by

$$\mathbf{B}\Delta\underline{\mathbf{u}} = \Delta\underline{\mathbf{u}}'$$

Approximate Factorisation (AF)

The technique of splitting (factorising) the operators associated with each coordinate direction.

Approximate Factorisation (AF) - Solution process

Set

$$\mathbf{A} = \left(1 + a\frac{\Delta t}{2\Delta x}\delta_x\right), \qquad \mathbf{B} = \left(1 + b\frac{\Delta t}{2\Delta y}\delta_y\right)$$

then the factorised implicit system is written as,

$$\mathbf{AB}\Delta\underline{\mathbf{u}} = \underline{\mathbf{R}}$$

This is solved in two stages,

$$\mathbf{A}\Delta\underline{\mathbf{u}}' = \underline{\mathbf{R}}$$

followed by

$$\mathbf{B}\Delta\underline{\mathbf{u}} = \Delta\underline{\mathbf{u}}'$$

Approximate Factorisation (AF)

The technique of splitting (factorising) the operators associated with each coordinate direction.

- AF does not alter the accuracy of the scheme but it does **alter the truncation error (which affects stability)**
- In 3-D the AF method is unconditionally unstable for the linear wave equation.
- Therefore some extra artificial dissipation is added to stabilise the method.

Modern Approaches

Recent increases in CPU speed have meant that 3-D flows no longer need to be solved by a fully AF scheme.

Factored-Unfactored (FUN)
More popular nowadays is to choose the direction in which the flow varies the least, and factorise in that direction only. For example a 3-D wing computation would be performed using a fully factored (exact) scheme in each spanwise plane, i.e. each plane around the section, with factorisation in the spanwise sense only.

Preconditioned Krylov Solvers - NOT EXAMINABLE
Completely unfactored systems are now routinely solved using preconditioned Krylov methods such as GMRES; these are iterative solvers that produce approximate solutions after a small number of iterations. This requires partitioning between multiple CPUs, due to the memory requirement; see next lecture.

Factored-Unfactored - Example (1)

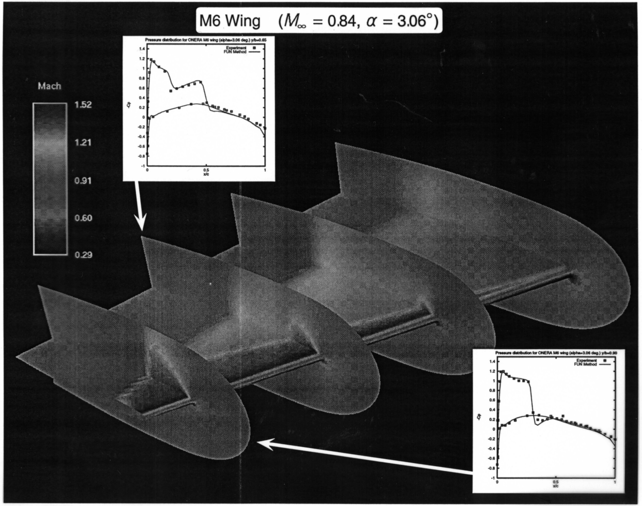
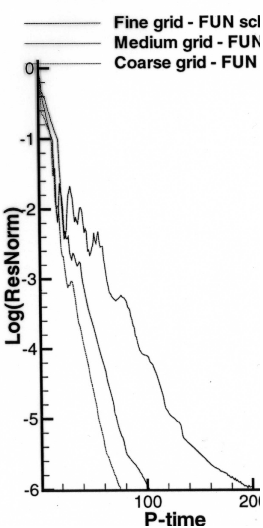
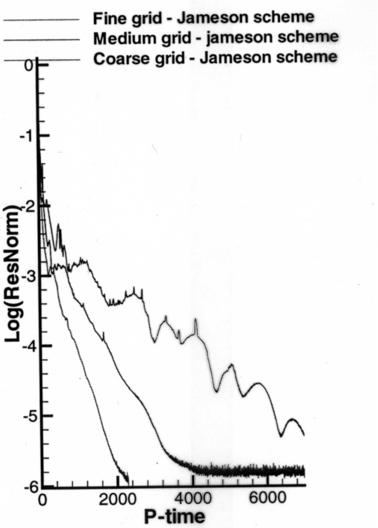


Figure 5: Steady solution on M6 wing using the FUN method.

Factored-Unfactored - Example (2)





Summary

- Implicit schemes are much more efficient than explicit in terms of convergence, but applying to a large 3D mesh is **extremely computationally expensive**.
  - Each row only has a small number of non-zero elements, but the number of non-zero elements is not related to the matrix bandwidth.
- Instead, we try to reduce the 3-D implicit problem to a series of 1D or 2D solutions. Looked at traditional problem reduction methods.
- For a typical wing case, a factor of 40 reduction in cost is achieved in convergence compared to an explicit scheme, using a Factored-UNfactored scheme. (Reductions $>$ 1000 are possible with the full system, although this is operations, NOT memory !)

Next Lecture: Consider the impact of computer hardware developments on the structure/application of CFD codes.