Fast Prediction of Wing Rock Limit Cycle Oscillations Based on Computational Fluid Dynamics

K.J. Badcock * and M.A. Woodgate †

Computational Fluid Dynamics Laboratory,
Flight Sciences and Technology,
Department of Engineering,
University of Liverpool,
L69 3BX, United Kingdom.

P.S. Beran ‡
Multidisciplinary Technologies Center,
Air Force Research Laboratory,
Wright Patterson Air Force Base,
Ohio, 45433-7531, USA.

The investigation of stability behaviour based on Computational Fluid Dynamics level aerodynamics is traditionally done in the time domain. It is however possible to look to the behaviour of the eigenspectrum of the Jacobian of the semi-discrete system to obtain information at a reduced computational cost. The central computational task in this approach is to solve a sparse linear system. This can be done by using a sparse matrix Krylov solver. The key issue then becomes calculating an effective preconditioner. This is a particularly difficult problem when implementing on a distributed memory parallel computer. The current paper considers this problem, and uses a combination of a polynomial and BILU preconditioner. The method is evaluated for the wing rock stability calculation via the Inverse Power Method.

I. Introduction

Rolling motion about the longitudinal axis of a delta wing has been computed using Computational Fluid Dynamics (CFD) by several researchers. 1-4 Recently a comprehensive numerical study of wing rock was conducted by Saad 7 using a three degree-of-freedom flight mechanics model for a generic fighter aircraft configuration (forebody, 65° leading edge sweep, and vertical fin). Roll, sideslip and vertical degrees of freedom were included. Including the sideslip degree of freedom was found to delay the onset of wing rock and reduce the wing rock amplitude. These problems provide a useful test case for computing flight mechanics instability since they feature constrained motions and nonlinear aerodynamics.

The computational cost of time domain CFD based predictions can be high. A way of reducing the cost of computing parametric searches for aeroelastic stability was proposed by Morton and Beran. 8 Their method uses dynamical systems theory to characterise the nature of the aeroelastic instability, with this additional information concentrating the use of the CFD. In this way the problem of locating a one parameter Hopf bifurcation was
reduced from multiple time marching calculations to a single steady state calculation of a modified (or augmented) system. This augmented system, solved via Newton’s method, calculates the value of the parameter for which an eigenvalue of the system Jacobian matrix crosses the imaginary axis.

This approach was applied to compute the wing rock onset angle. The formulation used a sparse matrix solver to efficiently solve the linear system arising at each Newton step for the augmented system. A matrix-free product was used to evaluate the Jacobian-vector product for the augmented residual, resulting in convergence to the correct wing rock onset angle for a spatial discretisation of second order accuracy. However, the Newton iterations were driven to convergence by the Jacobian matrix of the first order scheme, leading to a loss of quadratic convergence. Note that the Jacobian matrix we refer to is of the coupled CFD-rolling motion system of equations, and so involves the derivatives of the CFD spatial discretisation (here based on Osher’s approximate Riemann solver).

The classical shifted inverse power method (IPM) can be applied to the Jacobian matrix at any angle of attack to compute the interesting part of the eigenspectrum. This is useful to derive a starting solution for the augmented solver, and is a fast method for finding the onset angle in its own right. However, because only the Jacobian of the first order spatial scheme was available in the previous work it was not possible to exploit the full capability of the IPM.

Recently the analytical Jacobian for the second order spatial discretisation was derived and coded. This opened up the possibility of applying the Inverse Power Method to calculate stability. This is exploited in the current paper for predicting the wing rock onset angle.

The computational work involved with the IPM is largely associated with the linear solver. The current paper reports on an efficient Krylov type iterative method formulated in complex variables. The key issue is the preconditioning, and the Incomplete LU decomposition proves to be effective on serial computers. The second key advance on previous work reported in the current paper is the generalisation to parallel calculations.

The paper continues with the formulation of the IPM. The linear solver is then described, followed by the parallel version of the method, and tests. Finally, the wing rock onset angle is calculated on a significantly finer grid than was possible in previous work.

II. Formulation

The coupled rolling motion/Euler equations can be written to emphasise the dependencies as

\[
\frac{d}{dt} \begin{bmatrix} w_a \\ \phi_t \\ \phi \end{bmatrix} = \begin{bmatrix} R_a(w_a, \phi, \phi_t, \alpha) \\ \mu C_{l_0}(w_a) + D \phi_t \\ \phi_t \end{bmatrix}.
\]

Here the vector \( R_a \) denotes the discretisation of the spatial terms in the Euler equations, the details of which follow the formulation described in reference, which can be summarised as discretisation using Osher’s method with MUSCL interpolation on moving multiblock meshes. The stability of an equilibrium \( w = w_0 \) which satisfies \( R(w_0) = 0 \) is determined by the eigensystem of the Jacobian matrix \( A = \partial R/\partial w \). The matrix can be written out in block format as

\[
A = \begin{bmatrix}
\frac{\partial R}{\partial w_a} & \frac{\partial R}{\partial \phi} & \frac{\partial R}{\partial \phi_t} \\
\mu C_{l_0} & 0 & 0 \\
0 & 1 & 0
\end{bmatrix} = \begin{bmatrix}
A_{aa} & A_{af} \\
A_{fa} & A_{ff}
\end{bmatrix}.
\]

The term \( A_{aa} \) is a large sparse matrix which represents the Jacobian of the discretisation of the fluid equations with respect to the fluid variables.

We consider the stability problem when the angle of attack \( \alpha \) is the parameter. We are interested in finding the onset angle for the wing rock and assume that stability is lost through a Hopf Bifurcation. In this case the Jacobian matrix \( A \) has a pair of purely imaginary eigenvalues at the critical angle.

The Power Method is an algorithm for calculating the dominant eigenvalue/eigenvector pair of any given diagonalizable matrix \( A \). Its extension to the shifted inverse power method is practical for finding any eigenvalue provided that a good initial approximation is known. Assume that the \( n \times n \) matrix \( A \) has distinct eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_n \) and consider the eigenvalue \( \lambda_j \). Then a constant \( \omega \) can be chosen so that \( 1/(\lambda_j - \omega) \) is the dominant eigenvalue of
the matrix \((A - \omega I)^{-1}\).

For initial guess \(x_0\) and constant \(\omega\)

For \(k = 1, 2, \ldots\) Do:

\[
\begin{align*}
  z_k &= (A - \alpha I)^{-1}x_{k-1} \\
  \mu_k &= ||z_k||_\infty \\
  x_k &= z_k / \mu_k
\end{align*}
\]

EndDo

The shifted inverse power method can be used to calculate the location of the critical eigenvalue in the complex plane at a fixed angle \(\alpha\). By computing the location for multiple values of \(\alpha\) the angle at which the eigenvalue crosses the imaginary axis can be computed. Variations on this approach are possible and are formulated in detail in reference. 14

III. Linear Solver - Serial

A key part of the computational work in the IPM is the solution of a large sparse linear system.

Eisenstat, Elman and Schultz 17 developed a generalized Conjugate Gradient method that depends only on \(A\) rather than \(A^T A\) and is called the Generalized Conjugate Residual (GCR) algorithm. Saad and Schultz developed the Generalized Minimal Residual (GM-RES) algorithm which is mathematically equivalent to GCR but is less prone to breakdown for certain problems and requires less storage and arithmetic operations. However GCR remains the easier algorithm to implement, especially in parallel. The GCR algorithm is:

\[
\begin{align*}
  r_0 &= b - Ax_0 \\
  p_0 &= r_0 \\
  \text{For } j = 0, 1, 2, \ldots, \text{ until convergence. Do:} \\
  \alpha_j &= \frac{\langle r_j, Ap_j \rangle}{\langle Ap_j, Ap_j \rangle} \\
  x_{j+1} &= x_j + \alpha_j p_j \\
  r_{j+1} &= r_j - \alpha_j Ap_j \\
  \beta_{j+1} &= -\frac{\langle Ar_{j+1}, Ap_j \rangle}{\langle Ap_j, Ap_j \rangle}, \text{ for } i = 0, 1, 2, \ldots, j \\
  p_{j+1} &= r_{j+1} + \sum_{i=0}^{j} \beta_{j+1} p_i
\end{align*}
\]

Enddo

To calculate the \(\beta_j\) the vector \(Ar_j\) and the previous \(Ap_j\)'s are required. The number of matrix vector products per step can be reduced to one if \(Ap_{j+1}\) is calculated by

\[
Ap_{j+1} = Ar_{j+1} + \sum_{i=0}^{j} \beta_{j+1} p_i
\]

This may not be beneficial if \(A\) is sparse and \(j\) is large. A restarted version called GCR(m) is defined so that when the iteration reaches step \(m\) all the \(p_j\)'s and \(Ap_j\)'s are thrown away.

For the Block Incomplete Lower Upper (BILU) factorization the matrix is partitioned into \(5 \times 5\) matrix blocks associated with each cell in the mesh. The use of this blocking reduces the memory required to store the matrix in a sparse matrix format. For clarity matrix elements refer to these blocks from now on.

Consider a general sparse matrix \(A\) whose elements are \(a_{ij}, i,j = 1, \ldots, n\). A general incomplete factorization computes a sparse lower triangular matrix \(L\) and a sparse upper triangular matrix \(U\) so that the residual matrix \(R = LU - A\) satisfies certain constraints, such as having entries in a prescribed pattern. A common constraint consists of taking the zero pattern of the \(LU\) factors to be precisely the zero pattern of \(A\). However the accuracy of the \(ILU(0)\) incomplete factorization may be insufficient to provide an adequate rate of convergence.

More accurate block Incomplete LU factorisations allowing extra terms to be filled into the factorisation are often more efficient as well as more robust. Consider updating the \(a_{ij}\) element in full Gaussian Elimination (GE). The inner loop contains the equation

\[
a_{ij} = a_{ij} - a_{ik}a_{kj}.
\]
If \( \text{lev}_{ij} \) is the current level of element \( a_{ij} \) then the new level is defined to be

\[
\text{lev}_{ij} = \min(\text{lev}_{ij}, \text{lev}_{ik} + \text{lev}_{kj} + 1).
\]  

(6)

The initial level of fill-in for an element \( a_{ij} \) of a sparse matrix \( A \) is 0 if \( a_{ij} \neq 0 \) and \( \infty \) otherwise. Each time the element is modified in the GE process its level of fill-in is updated by equation 6. Observe that the level of fill-in of an element will never increase during the elimination. Thus if \( a_{ij} \neq 0 \) in the original matrix \( A \), then the element will have a level of fill-in equal to zero throughout the elimination process. The above gives a systematic algorithm for discarding elements. Hence ILU(\( k \)) contains all of the fill-in elements whose level of fill-in does not exceed \( k \). The algorithm is given by:

For all non zero elements \( a_{ij} \) define \( \text{lev}_{a_{ij}} = 0 \)

For \( i = 2, \ldots, n \) Do:

For \( j = 1, 2, \ldots, i - 1 \) and for \( \text{lev}_{a_{ij}} < k \)

\[
a_{ij} = a_{ij}/a_{jj}
\]

\[
a_{il} = a_{il} - a_{ij}a_{jl} \quad l = j + 1, \ldots, n
\]

Update the levels of fill in for non zero \( a_{ij} \)

EndDo

If \( \text{lev}_{a_{ij}} > k \) then \( a_{ij} = 0 \)

EndDo

There are two variations used for the preconditioning in the current work. First the linear solver described above is coded for complex variables. A complex variable formulation has 3 advantages over splitting the variable into real and imaginary parts. First it does not expand the bandwidth of the matrix. Secondly the preconditioner is faster to calculate and requires less storage. Thirdly the preconditioner was found to be more robust with respect to dropping off-diagonal terms.

The Block ILU(\( k \)) preconditioner of the Jacobian matrix is also available to precondition the system with the Jacobian matrix of the second order spatial scheme. The factorisation of the second order matrix can suffer from instability and often makes a poorer preconditioner than the factorisation of the matrix from the first order scheme. The first order matrix also has the advantage that it has 7 non-zero blocks per cell in the fluid grid, whereas the second order matrix has 13 non-zero blocks.

IV. Linear Solver - Parallel

A key develop for making the IPM a practical proposition for full scale problems is the development of a parallel implementation of the method. Calculating the steady state and forming the Jacobian matrix are standard operations in parallel and will not be discussed. The difficult part is the linear solution and this is described in the current section.

A. Matrix Generation and Krylov Method

The data decomposition is done by storing whole blocks of the multiblock grid on a single processor. This means that the partitioning of the grid can be consired as a problem of partitioning the blocks. Each processor stores a certain number of blocks in the grid and their associated fluid variables. The structural variables are treated in a different way. Each processor stores all the structural information. This is currently not too expensive as the number of structural equations is small compared to the number of fluid equations, but might have to be revisited for different problems.

The residual and Jacobian are decomposed in the method shown in figure 1. Here 4 processors are assumed and the schematic in the figure indicates the system components

\[
\begin{bmatrix}
A_{ff} & A_{fs} \\
A_{sf} & A_{ss}
\end{bmatrix}
\begin{bmatrix}
R_f \\
R_s
\end{bmatrix}
\]

Processors with no solid boundary conditions will have zeros in the \( A_{sf} \) part of the Jacobian.

Since the Jacobian matrix is calculated in 4 distinct parts \( A_{ff}, A_{fs}, A_{sf} \) and \( A_{ss} \), the matrix vector product is done in 4 parts which are summed together. The products with
Figure 1. Effective data decomposition of the Jacobian and right hand side vector

$A_{fs}$ and $A_{ss}$ are trivial as the structural unknowns are on all processors. The product with $A_{sf}$ requires a “reduce all” global sum at the end of the multiply to get the information to all the processors.

The ordering of the fluid unknowns within each block is important. The cells are partitioned as shown in figure 2, with 3 different sets of cells. The matrix-vector product of the internal set can be updated without any communication and hence it is possible to complete this part of the product while waiting on the messages to arrive from other processors. The border set are the cells owned by this processor but which require some communication with other processors to be fully calculated. The external set is not updated at all on this processor but is used to update the border points.

To maximise the parallel performance of the code the blocks must be carefully assigned to the processors. If there is a large number of blocks it is not difficult to get approximately the same workload onto each processor for the CFD only case. However in the IPM the workload for non solid wall fluid points is greater than that for any other point - due to the contributions from $A_{sf}$. Hence the number of fluid points on each processor and the number of solid wall points on each processor should be balanced. Lastly the number of processor block boundaries needs to be kept as small as possible. Not only does this keep the size of the messages small but much more importantly BILU(k) can be calculated with no extra approximations on block boundaries if both blocks are on the same processor.

B. Preconditioning

Now, the crucial issue is the parallel implementation of the linear solver. The calculation of the LU factors and the forward and backward substitutions on these factors are both sequential operations. It is possible to ignore terms in the Jacobian matrix $A$ which couple the system between processors. However, this can have a very bad influence on the convergence of the Krylov iterations for the systems being considered in the IPM. The decoupling of blocks through dropping terms localises the approximation to the solution of the linear system.

Calculating LU factors only has a limited amount of parallelism. In general a partitioning algorithm minimizes the number of interface elements by reducing the area of the boundaries. Then each processor can independently factor its set of nodes. This is a completely local operation. The interior nodes are then eliminated for the interface rows forming a reduced matrix $A^I$ corresponding to interface nodes only. Finally all the processors factor $A^I$. This final operation is where all the difficulty in parallel sparse factorisations

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{Effective data decomposition of the Jacobian and right hand side vector}
\end{figure}
resides. The factorisation of $A^f$ is done in stages in a nested fashion until $A^f$ is factored. The full factorisation has not been implemented in the current work.

To improve the coupling between processors at the preconditioning stage a polynomial preconditioner was considered. In polynomial preconditioning the matrix $M$ is defined by

$$M^{-1} = s(A)$$

where $s$ is a low degree polynomial. The polynomial $s$ is derived from the Neumann series expansion

$$I + N + N^2 + N^3 + \ldots + N^s \text{ where } N = I - A.$$  

Then

$$M^{-1} A = (1 - N^{s+1}).$$

The option finally selected for the preconditioner used a combination of the localised BILU preconditioner with the first order polynomial preconditioner. The idea is that the polynomial preconditioner brings in some of the global information through matrix-vector products that is lost by the localisation of the BILU factorisation.

V. Test Case and Time Marching Results

There is a considerable amount of published experimental data for 80° sweep delta wings. Hence this sweep angle was selected for the current study. The geometry of the wing was identical to that used by Arena and Nelson. The wing has a flat upper and lower surface with a 45° windward bevel and a root chord of 0.4222m. The moment of inertia for this wing was given as $I_{xx} = 0.00125 \text{ Kg-m}^2$. The experiment was performed at a Reynolds number of $1.5 \times 10^5$. Since the flow solver here is compressible, a Mach number of 0.2 was used to avoid convergence difficulties at the very low Mach numbers used in the experiments. The freestream air density is used to non-dimensionalise the moment of inertia of the wing for the comparisons. The freestream air density in the experiment was unavailable and so the value was assumed to be $1.23 \text{Kg/m}^3$ (sea-level ISA conditions).

A fine grid for the Euler simulations was created with approximately 1.6 million points. From this grid two levels were extracted by removing every second grid point in each direction. A RANS grid was also generated. The comparison of the pressure distributions along a spanwise cut with measurements from Arena and Nelson is shown in figure 3. The RANS predictions agree well with the measurements, with only a slight under-prediction of the secondary separation. The medium grid Euler results predict the suction level close...
to the measured value. The Euler results do not predict the secondary separation, as this is associated with boundary layer separation. Simulations of free-to-roll motion at 15° and 20° angle of attack were conducted on the medium grid using the Euler equations. The roll angle histories for both simulations are shown in figure 4. At 15° angle of attack the solution is dynamically stable (i.e. the amplitude of the oscillations decreases due to aerodynamic damping). However at 20° angle of attack, the initial roll angle of 10° initiates a wing rock response with the amplitude of the motion increasing with time.

VI. Results

A. Coarse Grid Tests

As a first test of the second order Jacobians and the linear solver, the IPM was used to calculate the wing rock onset angle on the coarse grid. 3 Levels of fill-in were used for the preconditioner which allowed the residual of the linear system to be driven down 11 orders in less than 60 iterations.

At 22 degrees, after 7 inverse power iterations the critical eigenvalue had converged to 6 significant figures. The value is $-1.381245 \times 10^{-4} \pm 0.223497$. At 23 degrees the converged value is $1.1805895 \times 10^{-4} \pm 0.232095$. These angles bracket the onset angle, which can be estimated by linear interpolation as 22.4 degrees. This is in agreement with the results of other methods for the coarse grid.

Next, the use of the first order Jacobian to calculate the preconditioner for the linear system which uses the second order Jacobian is examined. The system used to test the linear solver was generated at 21 degrees incidence. There are two preconditioners used: the BILU factorisation with 1, 2 and 3 levels of fill-in, and the Polynomial preconditioner combined with BILU(2). The calculations shown in figure 5 were done on 1 processor and so the BILU factorisation involves no extra approximation to allow efficient parallel calculation. We first evaluate the results in terms of iterations to convergence since we are ultimately concerned with how the performance of the Krylov method scales in parallel. The number of iterations to convergence reduces as expected as the level of fill-in is increased. The number of iterations in each case is significantly larger than when the preconditioner is calculated for the second order Jacobian matrix, when 3 levels of fill-in
are required. The increase in the number of iterations is from 60 to 168 although each iteration is around twice as expensive for the second order matrix preconditioner. Using the polynomial preconditioner on top of BILU(2) in this case does nothing to change the number of iterations to convergence.

The cost of computing the eigenvalues in terms of multiples of steady state calculation cost is 2.80 for BILU(0), 2.34 for BILU(1), 2.44 for BILU(2) and 3.70 for the Polynomial preconditioner in addition.

Next we test the scaling of the methods in parallel. Here the BILU factorisations are blocked on each processor and it is the influence of this that is the key concern. The convergence using BILU(2) with and without polynomial preconditioning on one and four processors is shown in figure 6. The number of iterations using the BILU preconditioner roughly doubles between 1 and 4 processors. Using the polynomial preconditioner actually reduces the number of iterations to convergence when moving from 1 to 4 processors. These results are encouraging in the sense that the polynomial preconditioner is bringing in enough global information about the solution of the system to allow scaling to larger numbers of processors.

B. Medium Grid Tests

The test of the parallel solver is whether the medium grid can be solved, which it cannot in serial because of the memory requirements of the method (estimated 5.6Gb to store the Jacobian and preconditioner). To evaluate this the medium grid was calculated on 16 processors. The convergence of the linear solver is shown in figure 7. The system used here was generated at 15 degrees incidence.

First the BILU(2) results are shown for reference and the performance of the linear solver becomes very poor due to the localisation of the preconditioner on each processor. The polynomial preconditioner significantly cuts the number of iterations to convergence. Results are shown using different levels of fill-in, and the number of iterations to convergence cluster between 200 and 260 iterations. The number of iterations will inevitably increase as the size of the grid increases, even sequentially.

The performance of the IPM is summarised in table 1. First, the improvement in the performance of the linear solver from using the polynomial preconditioner is clear. The number of linear solver steps to convergence is is reduced by roughly a factor of 3.
Figure 5. Linear solver convergence for BILU(2) and Polynomial/BILU(2) on the coarse grid and 1 processors.

Figure 6. Linear solver convergence for BILU(2) and Polynomial/BILU(1) on the medium grid and 1 and 4 processors.
resulting improvement in the time to convergence of the IPM is roughly 25-40%. Finally, the table shows the bifurcation happens between 16 and 17 degrees. This is consistent with the behaviour of the time domain solver shown in figure 4.

<table>
<thead>
<tr>
<th>( \alpha )</th>
<th>CPU ( \alpha )</th>
<th>CPU Linear Polynomial BILU(2)</th>
<th>Linear Polynomial BILU(2)</th>
<th>Eigenvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>15°</td>
<td>2.91 3.80</td>
<td>229</td>
<td>596</td>
<td>0.172 (-i7.53 \times 10^{-4})</td>
</tr>
<tr>
<td>16°</td>
<td>2.53 4.08</td>
<td>214</td>
<td>641</td>
<td>0.183 (-i2.25 \times 10^{-4})</td>
</tr>
<tr>
<td>17°</td>
<td>2.31 4.00</td>
<td>202</td>
<td>704</td>
<td>0.192 (+i2.01 \times 10^{-4})</td>
</tr>
<tr>
<td>18°</td>
<td>2.69 4.38</td>
<td>265</td>
<td>774</td>
<td>0.202 (+i6.32 \times 10^{-4})</td>
</tr>
</tbody>
</table>

Table 1. Summary of Timings on Medium Grid on 16 processors

Figure 7. Linear solver convergence for BILU(2) and Polynomial/BILU(k) on the medium grid and 16 processors.

VII. Conclusions

The parallel implementation of the IPM was considered in this paper. The problem is in the preconditioning of the Krylov linear solver. The baseline method is BILU factorisation with varying levels of fill-in. When implementing this in parallel a reduction in performance arises from making the BILU factorisation local. It was proposed to use polynomial preconditioning on top of blocked BILU factorisation to bring in more global information into the preconditioning. Results showed that this was effective in maintaining the number of Krylov iterations to convergence, even if the operations associated with the preconditioning are more expensive.

To reduce memory requirements the BILU factorisation was based on the first order Jacobian matrix rather than the second order one. This was found to increase the number of steps to convergence, but was none-the-less an effective approach.

Calculations were successfully carried out on the medium grid on 16 processors. The performance of these calculations showed that the eigenvalue at each angle of incidence could be computed in a computational cost between 2 and 3 times the cost of a steady
state CFD calculation.

VIII. Acknowledgements

This work has been funded by the European Office of Aerospace Research and Development under contract FA8655-03-1-3044.

References


