20th AIAA Computational Fluid Dynamics Conference
27 - 30 June 2011, Honolulu, Hawaii

AIAA 2011-3234

An Implicit Semi-Meshless Scheme with Stencil Selection for Anisotropic Point Distributions

D. J. Kennett*, S. Timme†, J. Angulo‡ and K. J. Badcock§

University of Liverpool, Liverpool, England L69 3BX, United Kingdom

An implicit meshless scheme is developed for calculation of the Navier-Stokes equations. Spatial derivatives are approximated using a least squares method on clouds of points as opposed to cells used with conventional finite volume techniques. The resulting linear system of equations is solved using an iterative Krylov subspace method. Details such as the linear solver and construction of the Jacobian are discussed, and results which demonstrate the accuracy and efficiency of the scheme are presented for various steady and unsteady test cases. A new method of selecting the stencils from anisotropic point distributions, which are obtained from overlapping structured grids is outlined. The original connectivity and the concept of a resolving direction are used to help construct good quality stencils with limited user input that not only provide well conditioned least squares matrices, but will also capture accurately flow features such as boundary layers.

I. Introduction

To simulate unsteady flows associated with fixed wing aircraft using computational fluid dynamics (CFD) can be extremely difficult using conventional finite volume, multi-block methods. Cases with moving geometries require mesh deformations that are limited to those where the block topology is fixed. This rules out certain physical problems when the motion of the bodies is not known a-priori such as store release from a wing or cavity.

A large amount of research has been made in the last few years to try and remedy this problem. One such avenue of research is with the so called meshless method[1,2] in which clouds of points are used as opposed to conventional grids. These clouds are then used to solve the governing equations by performing a local least squares function approximation in these clouds to find the derivatives of the partial differential equation. Least squares methods already see wide use in traditional CFD methods, often as a means of reconstructing higher order variables in finite volume schemes.[3] The meshless solver is effectively an extension of this in that we use least squares to directly compute the entire flux derivatives. Cells are not used at all and so many of the difficulties associated with grid generation are avoided. The use of a method that does not require successive remeshings can dramatically simplify the modelling of the class of problems outlined above and thus require reduced man-hours.

Point collocation methods are generally preferred in CFD and there are many examples of such meshless solvers in the literature.[4–8] Most of these use explicit methods to solve the resultant system of equations. Meshless implicit schemes are much rarer in the literature, but those presented include Sridar and Balakrishnana[9] with flux approximations made using the Taylor series, and Chen and Shu[10] who use radial basis functions. Both of these schemes use a lower-upper symmetric Gauss-Siedel (LU-SGS) method. This work presents an implicit method with a preconditioned Krylov subspace solver with approximate Jacobians, with polynomial basis functions used for the flux approximations. The formulation of the meshless and implicit schemes are outlined in 2D in sections II and III respectively, generalisation to 3D is straight forward. Results are presented in section IV that show the accuracy and efficiency of the method.

*PhD Candidate, CFD Laboratory, School of Engineering, D.Kennett@liverpool.ac.uk.
†Research Associate, CFD Laboratory, School of Engineering, S.Timme@liverpool.ac.uk.
‡PhD Candidate, CFD Laboratory, School of Engineering, J.Angulo@liverpool.ac.uk.
§Professor, CFD Laboratory, School of Engineering, K.J.Badcock@liverpool.ac.uk, Senior Member AIAA.

One of the major points of discussion for meshless methods is how to obtain the points and construct the stencils for the use of the solver. Point generation schemes have been proposed,[11,12] which will make it possible to completely harness the advantages of using a meshless solver. Another way is to obtain the point distributions from meshes. This may seem contradictary, though the effort required to generate the points is low as it is possible to use relatively simple meshes for several components within the domain and overlap them. The solver does not see the cells, only the points that make them up. This is the method used in this paper. Structured grid points are preferred due to their ability to capture high gradient flow effectively.

Despite its importance, the problem of appropriate stencil selection is unfortunately one that has not been addressed a great deal in the literature.[13] The construction of the local clouds is not trivial, and although some papers have been published on this subject, there is yet to be a fully robust, efficient method for choosing the best stencils from any point distributions. This problem is avoided completely for unsteady calculations in,[14] in which an aerofoil is allowed to pitch and plunge within a fixed domain. A fast dynamic cloud method based on a Delaunay graph mapping is proposed, in which the structure of the meshless clouds are not necessarily changed due to the mapping used when the points move. This means that stencil selection is not required at each time step. Hybrid schemes also largely avoid this problem (at least in the majority of the domain) by using a combination of finite volume and meshless solvers together in the same calculation. This means that body fitted grids can be created and linked together using a meshless technique[15,16] or meshless solvers can be used on the boundary while the rest of the domain is discretised with a finite volume technique.[17]

Of the papers that address the issue of selection directly, a quadrant procedure (in 2D) is described in[6] in which the closest points in each quadrant around a given point are chosen for the stencil. This is an improvement on a very simple nearest neighbour approach. There is also a method presented by Löhner,[18] in which we start with a global cloud of points and an initial connectivity on the boundaries, and create stencils using a Delaunay triangulation technique. Kamruzzaman[19] uses a search algorithm and a Gauss-Jordan pivoting method on a set of points to select the best points so that the delta property is fulfilled. This means that the consistency conditions and the partition of unity will be satisfied, though it means that the number of points in the stencil will always be equal to the number of unknown coefficients in the function to be approximated. Seibold[20] presented an approach which produces positive stencils for the laplace operator. This leads to an M-matrix structure of the system matrix. However there are necessary assumptions on the point clouds that are made to ensure that positive stencils always exist.

The selection methods outlined above all start with no original point connectivity except on the boundaries. However, reference[21] uses an intermediate approach to generate full connectivity from some limited connectivity. In this case it is a semi mesh, which is a set of lines eminating from the boundary surfaces. We propose a method that uses the original connectivity of structured grids as an aid to selecting the meshless stencils when these grids are made to overlap. As the point distributions can be anisotropic in some regions of the domain and not others, there are no assumptions made about the distribution of the points. The grid connectivity is used, so the scheme can be defined as semi-meshless. However, the stencil selection and meshless computations are global. The development of this method is outlined in section V and some simple test cases that use this method are presented in section VI. The solver and stencil selection schemes outlined in this paper form part of a combined code under development at the University of Liverpool called Parallel MeshLess (PML).

II. Spatial discretisation

II.A. Overview of the meshless method

An open bounded domain $\Omega \in \mathbb{R}^D$ is discretised by a set of N points, which can be distributed randomly throughout Ω. To each point x_i, $i = 1, .., N$ we assign a subdomain (which is often called a stencil or cloud) Ω_i, which contains x_i often called the "star point" of the subdomain and a set of $(n-1)$ neighbouring points, which we label j. One may use all available data points, but for the purpose of efficiency it is reasonable to select some small subset of data points in the neighbourhood of i so that $\sum_{i=1}^{N} \Omega_i$ represents a covering of Ω. The subdomains must overlap so that the domain is properly linked together, otherwise discontinuities may occur where there is not sufficient overlap. Unfortunately, this idea of sufficient overlap is not easily quantifiable. Assuming that we are given data $\{(x_i, \phi(x_i))\}_{i=1}^{N} \subset \mathbb{R}^D$ at each of these point data sites, where ϕ is some (smooth) function, we seek a local discretised or approximation solution $\hat{\phi}$ of the function ϕ within

Ω_i. The choice of space of $\hat{\phi}$ is a vector space V with simple basis. There are m basis functions, which we use to approximate the function ϕ using a linear combination of these basis functions represented by

$$\hat{\phi}_i(x_j) = \sum_{k=1}^{m} p(x_j)_k \alpha_k = \mathbf{p}^T \boldsymbol{\alpha} \qquad \forall x_j \in \Omega_i \tag{1}$$

where $\mathbf{p}$ is the vector of base monomials at each point j within the subdomain, so for example in 2D

$$\begin{aligned} \mathbf{p}(x_j) &= \{1, x_j, y_j\}^T && \text{for } m = 3 \text{ (linear approximation)} \\ \mathbf{p}(x_j) &= \{1, x_j, y_j, x_j^2, x_j y_j, y_j^2\}^T && \text{for } m = 6 \text{ (quadratic approximation)} \end{aligned}$$

and $\boldsymbol{\alpha}$ is the local vector of coefficients that must be determined

$$\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_m\}^T$$

Hence for each subdomain, we must solve the following linear system of n equations for $\boldsymbol{\alpha}$

$$\hat{\boldsymbol{\phi}}_{(n\times 1)} = X_{(n\times m)} \boldsymbol{\alpha}_{(m\times 1)} \tag{2}$$

where X is formed from the vectors of base monomials

$$X = \{\mathbf{p}(x_1), \mathbf{p}(x_2), ..., \mathbf{p}(x_n)\}^T$$

For more accurate results, each of the equations within the system can be weighted so that the least squares approximation is enhanced in a region of the vicinity of the star point. This means that points further from the star point have less influence on the approximation than those that are closer. In this work a normalised Gaussian function[22] is used.

As one point within the subdomain can belong to several other overlapping subdomains, then each of these subdomains can yield different coefficients for the same point. To prevent this, the approximate function $\hat{\phi}$ within Ω_i is valid only at the star point. This will also mean that the required consistency conditions[1] will be satisfied. The function that best approximates the exact vales at the data sites within the stencil of the star, will be that for which the residual

$$\mathbf{r} = \boldsymbol{\phi} - \hat{\boldsymbol{\phi}}$$

is a minimum. Provided the system is overdetermined (i.e. $n > m$) then we can define the "best" solution for the star within the cloud as the one for which the residual measured in the L2 norm is a minimum, this leads to solving the least squares problem

$$X^T X \boldsymbol{\alpha} = X^T \boldsymbol{\phi} \tag{3}$$

the solution of which gives us the coefficients $\boldsymbol{\alpha}$ to be determined to approximate $\hat{\phi}_i$ at the star. From Eq. (1), this can be written

$$\begin{aligned} \hat{\phi}_i(x_i) &= \mathbf{p}_i^T(x_i)\boldsymbol{\alpha} \\ &= \mathbf{p}_i^T(x_i)(X^T X)^{-1} X^T \boldsymbol{\phi} \\ &= \mathbf{N}_i(x_i)\boldsymbol{\phi} \end{aligned} \tag{4}$$

where $\mathbf{N}_i$ is a $(1 \times n)$ vector called the shape function of point $x_i \in \Omega_i$, this is unique for each star point within each subdomain $\Omega_i \subset \Omega$ and is dependent on the location of each point within the subdomain. The shape function is of fundamental importance in the meshless method, it is written explicitly as

$$\begin{aligned} \mathbf{N}_i &= \mathbf{p}^T(X^T X)^{-1} X^T \\ &= \mathbf{p}^T(A^{-1}B) \qquad\qquad {\scriptstyle (1\times m)(m\times n)} \end{aligned} \tag{5}$$

It can be shown that $\mathbf{N}_i$ is a partition of unity function, which allows us to patch together the approximate functions $\hat{\phi}$ found within each subdomain locally, to give a global solution.[24] The partition of unity means that any function of order k in the basis complete in the polynomials of order k can be reproduced exactly. In the literature this is referred to as the "consistency" of the approximation.[1] For example, if the basis is

linear then the shape functions will satisfy linear consistency, without this we would not be able to form a global approximation within the entire domain Ω.

To use the meshless method to solve a partial differential equation in the form of a conservation law, it is clear that we are actually more interested in the derivatives of the functions. For each star point $x_i \in \Omega$ we can write the following approximation for ϕ using the Einstein summation notation (so sum over j, here i remains a label) and Eq. (4)

$$\hat{\phi}_i = \hat{\phi}(x_i) = N_{(i)j}\phi_j$$

So in terms of the shape function, the partial derivatives of $\hat{\phi}$ with respect to x can be obtained using the product rule directly from the approximation

$$\begin{aligned}
\frac{\partial \hat{\phi}_i}{\partial x} &= \frac{\partial}{\partial x}(N_{(i)j}\phi_j) \\
&= \frac{\partial}{\partial x}\left\{N_{(i)j}\right\}\phi_j \\
&= \frac{\partial}{\partial x}\left\{p_k(A^{-1}B)_{kj}\right\}\phi_j \qquad\qquad k = 1, ..., m \\
&= \frac{\partial}{\partial x}\left\{p_k\right\}\left\{(A^{-1}B)_{kj}\right\}\phi_j + \left\{p_k\right\}\frac{\partial}{\partial x}\left\{(A^{-1}B)_{kj}\right\}\phi_j \\
&\approx \frac{\partial}{\partial x}\left\{p_k\right\}\left\{(A^{-1}B)_{kj}\right\}\phi_j \\
&= (A^{-1}B)_{2j}\phi_j \\
&= b_{(i)j}\phi_j
\end{aligned} \tag{6}$$

where b_i represents the $(1 \times n)$ shape function derivatives of x_i, with the sum taken over the components j. The derivative with respect to y follows in the same way. Note that we have simplified the approximation and computational cost considerably by assuming that the matrices A and B are constant in Ω_i. This distinguishes the method from the moving least squares (MLS) approach and is often called the fixed least squares (FLS) approach.[5]

Equation (6) can be found relatively cheaply using Cholesky decomposition, which exploits the symmetric, positive-definite property of the least squares matrix $X^T X$. The problem is that the columns forming X may not be linearly independent and so the matrix may be singular.[23] This can be the case if the points are taken from an area of the grid designed to capture flow in the boundary layer. The weights reduce the influence of points within the stencil that are further away from the star point, so for a highly directional stencil, this can make the influence of many of the points negligible. This can often lead to an approximation equivalent to $n < m$, with the remaining points lying in a straight line, resulting in a singular least squares matrix. We can help resolve this by mapping the stencil distribution to a computational domain, finding the shape function derivatives and then mapping back to the physical domain. The computational domain is obtained by normalising the cloud so that the star point is located at the origin. We then measure the distance between the star points and each of its neighbours (located at $\mathbf{x}'$) along the x and y axes and store the largest values. These are denoted R_x and R_y respectively. The coordinates within the stencil are then normalised such that

$$\xi = \frac{x'}{R_x} \qquad\qquad \eta = \frac{y'}{R_y}$$

This improves the accuracy of the procedure near the star point. Once Eq. (4) has been computed, we map the shape function derivatives back to the physical domain. As well as the mapping, it can be further desirable (but more expensive) to compute the shape function using singular value decomposition (SVD) to solve the least squares problem. This method allows us to automatically find and correct the ill-conditioned components of the approximation.

As the shape function derivatives are dependent only on the relative position of the points, then if the geometry of the points remains the same (i.e. for stationary problems), then as the weights and shape functions depend purely on the position of the points, it is sufficient to calculate Eq. (6) once and store the values in data structures to reduce the computational costs.

II.B. Evaluating the inviscid flux

The Euler equations are a simplification of the full Navier-Stokes equations, formed by removing the viscous terms. They are a set of hyperbolic conservation laws, which can be written in two dimensions in conservative form and Cartesian coordinates as

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial \mathbf{f}(\mathbf{w})}{\partial x} + \frac{\partial \mathbf{g}(\mathbf{w})}{\partial y} = 0 \tag{7}$$

where $\mathbf{w}$ denotes the vector of conserved variables, which is to be determined, and $\mathbf{f}$ and $\mathbf{g}$ are the inviscid flux vectors. To solve this system using the meshless method, we write Eq. (7) for each local subdomain i in semi-discrete form using Eq. (6) to approximate the spatial flux derivatives

$$\frac{\partial \mathbf{w}_i}{\partial t} = -\sum_{j\in\Omega_i} b_{(i)j}\mathbf{f}_j - \sum_{j\in\Omega_i} c_{(i)j}\mathbf{g}_j$$

This form of the discretisation is centred and as such is unstable for use with hyperbolic partial differential equations for which information propagates along characteristics in certain definite directions. Stability is obtained if we use an upwind scheme for the flux functions defined half way between the star point i and the neighbouring point j (analogous to the dual cell in a finite volume scheme) at $j-\frac{1}{2}$, this leads to an equivalent semi-discrete expression

$$\frac{\partial \mathbf{w}_i}{\partial t} = -\sum_{j\in\Omega_i} b_{(i)j}\mathbf{f}_{j-\frac{1}{2}} - \sum_{j\in\Omega_i} c_{(i)j}\mathbf{g}_{j-\frac{1}{2}} \tag{8}$$

As opposed to computing the flux at i and j directly, the computation at the location half way across the edge can be seen as an interface between the (approximated as constant) flux values at i and those at j. To calculate the flux is thus a Riemann problem[25] for which the left hand state is always the star point i and the right hand state is always the neighbouring point j. In this work the Riemann problem is solved approximately using Roe's method[26] with an appropriate entropy correction, for which the mid-point flux is written.

$$\mathbf{f}_{j-\frac{1}{2}} = \frac{1}{2}(\mathbf{f}(\mathbf{p}_L) + \mathbf{f}(\mathbf{p}_R)) - \frac{1}{2}\left|\tilde{A}(\mathbf{p}_L, \mathbf{p}_R)\right|(\mathbf{p}_R - \mathbf{p}_L) \tag{9}$$

where $\tilde{A}$ is the Roe averaged Jacobian, and $\mathbf{p}_L$ and $\mathbf{p}_R$ are the vectors of primitive variables at the left and right hand sides of the interface at the half way point. For a first order scheme, these states are simply the values of the flow variables located at the star and neighbouring points

$$p_L = p_i$$

$$p_R = p_j$$

Thus the number of points that the flux depends on (denoted $\mathcal{M}(i)$) is simply the number of points that make up the stencil, so $\mathcal{M}(i) = n_i$. Practical engineering calculations however generally require at least second order spatial accuracy, which is achieved in a method used with unstructured finite volume codes.[28] Here the left and right states of the Riemann problem are obtained by extrapolating the values at i and j based on a reconstructed gradient

$$\begin{aligned} p_L &= p_i + \psi \mathbf{l}_{ji}.\bigtriangledown p_i \\ p_R &= p_j - \psi \mathbf{l}_{ji}.\bigtriangledown p_j \end{aligned} \tag{10}$$

where $\mathbf{l}_{ji} = \frac{1}{2}(\mathbf{x}_j - \mathbf{x}_i)$ is the vector formed half way between the star and neighbouring point. As a result, the flux now depends on the stencils of the neighbouring points as well (as they form $\bigtriangledown p_j$), in other words $\mathcal{M}(i) > n_i$. With second order schemes like this we have the problem that near discontinuities, spurious non-monotinicity (oscillations) appear. To counter this, we use flux limiters ψ_i and ψ_j, which bound the reconstructed function of each flow variable between the extreme values within the stencil, by reducing the scheme to first order whenever these oscillation may occur. Many limiters are available in the literature, but the one used in this work is the Barth and Jespersen limiter,[27] which is commonly used for unstructured grids.

II.C. Evaluating the viscous flux

The full Navier-Stokes equations require the addition of the viscous fluxes to the Euler equations. Thus Eq. (7) is modified to

$$\frac{\partial \mathbf{w}}{\partial t} + \frac{\partial}{\partial x}(\mathbf{f} - \mathbf{f}^v) + \frac{\partial}{\partial y}(\mathbf{g} - \mathbf{g}^v) = 0 \tag{11}$$

where $\mathbf{f}^v$ and $\mathbf{g}^v$ are the non-dimensionalised viscous fluxes in the x and y directions. In some respects the discretisation of the viscous terms is not as problematic as the inviscid terms as they are parabolic in nature and so do not require any upwinding. Instead, a variation on the central difference scheme is sufficient for these terms. However the viscous fluxes contain not just the flow quanties but also the derivatives of the flow quantities.

The method advocated by Batina[4] is used to deal with this, in which the gradients at the stencil points are used directly in the least squares flux derivatives. Thus the central differencing of the gradients at i and j gives the midpoint gradient

$$(\overline{\nabla\phi})_{ij} = \frac{1}{2}(\nabla\phi_i + \nabla\phi_j)$$

The advantage of this method is that the same shape function derivatives as in Eq. (8) are used. The disadvantage is that it does not suppress odd-even node decoupling. To overcome this the derivative quantities that appear in the viscous flux can be constructed using a modified gradient[28] defined as

$$(\overline{\nabla\phi})_{ij} = \frac{1}{2}(\nabla\phi_i + \nabla\phi_j) - \frac{1}{2}\left[\nabla\phi_i + \nabla\phi_j\right].\frac{\mathbf{l}_{ij}}{|\mathbf{l}_{ij}|} + \frac{\phi_j - \phi_i}{|\mathbf{l}_{ij}|}$$

where $\mathbf{l}_{ij}$ is the vector joining points i and j. The flux derivatives of these midpoint terms are then found by using the same shape functions evaluated half way between i and j for the inviscid flux. In other words, the full meshless discretisation of Eq. (11) is given by

$$\frac{\partial \mathbf{w}_i}{\partial t} = -\sum_{j\in\Omega_i} b_{(i)j}(\mathbf{f}_{j-\frac{1}{2}} - \mathbf{f}^v_{ij}) - \sum_{j\in\Omega_i} c_{(i)j}(\mathbf{g}_{j-\frac{1}{2}} - \mathbf{g}^v_{ij}) \tag{12}$$

which is the system of linear equations, that is solved in section III.

II.D. Boundary conditions

The points that form the boundary $\partial\Omega$ make up a set of elements that provide a covering such that the domain Ω is bounded. In 2D the elements are line segments made up of two points, while in 3D they can be either triangles or quadrilaterals, made up of three and four points respectively. The points that make up each element i form the set $\mathcal{B}(i)$. In addition, the interior points that are common to all of the stencils of $\mathcal{B}(i)$ make up the set $\mathcal{C}(i)$.

Halo points located outside the domain are used to impose boundary conditions on each of these elements and thus provide the required behaviour on the boundary itself. There is one halo per element, the location of which is such that the halo point lies along the inward normal of the element from its centre. The distance from the centre is determined by the normal distances between the element centre and the points of $\mathcal{C}(i)$. Far field conditions are enforced by setting the halo point h to freestream values. This treatment assumes that the boundary is located far enough to allow the flow to return to far field conditions.

For the Euler equations, we enforce the slip condition on solid wall boundaries. For unsteady simulations with moving surfaces, this means that the values at the halo point are set such that the velocity of the wall is equal to the flow velocity normal to the boundary. That is

$$u_i n_x + v_i n_y = \dot{x} n_x + \dot{y} n_y$$

where n_x and n_y are the outer normals of the boundary element and the cartesian wall velocity components are represented by $\dot{x}$ and $\dot{y}$. The flow variables u_i used are obtained by taking the average of the values at the points within the set $\mathcal{C}(i)$.

This boundary condition is imposed by using a negative normal velocity component of u_i at the halo point, which will cancel out those of $\mathcal{C}(i)$. The normal and tangential components of u_i are obtained by a simple rotation

$$\begin{pmatrix} u_t \\ u_n \end{pmatrix} = \begin{pmatrix} n_y & -n_x \\ n_x & n_y \end{pmatrix} \begin{pmatrix} u_i \\ v_i \end{pmatrix}$$

Then the normal components are negated, and the resultant values are rotated back and added to the wall velocities to give the halo point values

$$\begin{pmatrix} u_h \\ v_h \end{pmatrix} = \begin{pmatrix} n_y & n_x \\ -n_x & n_y \end{pmatrix} \begin{pmatrix} u_t \\ -u_n \end{pmatrix} + \begin{pmatrix} n_x & 0 \\ 0 & n_y \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$$

The density and pressure halo values are defined as the average of the values of the points in $\mathcal{C}(i)$, so

$$\begin{aligned} \rho_h &= \rho_i \\ p_h &= p_i \end{aligned}$$

this is equivalent to zeroth order extrapolation.

The Navier Stokes equations include viscous terms, which contain second order derivatives. As a result, we must increase the number of boundary conditions so that the equations remain well posed. For viscous flow we therefore impose the additional condition that the tangential velocity on the boundary wall is zero, this is the no slip condition, which means that

$$u_i = \dot{x} n_x + \dot{y} n_y$$

on the solid walls for unsteady calculations. This boundary condition is imposed by setting the velocity values at the halo point for the element i to be of the opposite sign to that of the average of $\mathcal{C}(i)$. The pressure and density at the ghost points are determined by the same zeroth order extrapolation of the values as for the slip condition.

Now that each halo for its corresponding element i has the appropriate flow values, we add it to the stencils of each of the points belonging to $\mathcal{B}(i)$. The use of halo points (in the same way as the use of the ghost points used in[29]) is advantageous in that it actually improves the quality of the stencils on the boundaries. They ensure that there will then be points in all directions and thus the conditioning of the least squares matrix improves dramatically

III. Temporal Discretisation

III.A. Time integration

The meshless method is used in the spatial discretisation to find the flux derivatives, which form the residual vector consisting of the right hand side of Eqs. (8) and (12). Its definition provides us with a set of global differential equations to solve

$$\frac{d\mathbf{w}}{dt} = -\mathbf{R}(\mathbf{w}) \tag{13}$$

the sign of the definition of the residual is convention in CFD. To solve this system we need to perform an integration with respect to time. Most meshless codes in the literature use explicit methods (with appropriate convergence accelerators) to solve Eq. (13), in which the residual is calculated at the current time step n, to be solved for the vector $\mathbf{w}$ at time $n+1$. However in this work we use a fully implicit time discretisation, which is described in this section.

III.B. Implicit methods

The main advantage of an implicit method is that the time step can be chosen for time accuracy without worrying about numerical stability restrictions. Implicit methods are characterised by the residual being evaluated at the unknown, new time level, creating a system of non-linear algebraic equations to be solved, so Eq. (13) becomes

$$\frac{d\mathbf{w}}{dt} = -\mathbf{R}(\mathbf{w}^{n+1}) \tag{14}$$

where the superscript $n+1$ denotes the time level $(n+1)\Delta\tau$ in pseudo time. There are in general two methods that can be used to solve this system:

- Solve at each step using a non-linear sub-iteration

- Choose some linearisation of $\mathbf{R}$ and reduce the non-linear system to a linear algebraic equation

The former method can be seen used with meshless solvers in.[14,30,31] In this work, the latter method is used in which we linearise the global residual vector $\mathbf{R}$ in pseudo time τ such that:

$$\begin{aligned}\mathbf{R}(\mathbf{w}^{n+1}) &= \mathbf{R}(\mathbf{w}^n) + \frac{\partial \mathbf{R}(\mathbf{w}^n)}{\partial \tau}\Delta\tau^n + O(\Delta\tau^2) \\ &= \mathbf{R}(\mathbf{w}^n) + \frac{\partial \mathbf{R}(\mathbf{w}^n)}{\partial \mathbf{p}}\frac{\partial \mathbf{p}^n}{\partial \tau}\Delta\tau^n + O(\Delta\tau^2) \\ &= \mathbf{R}(\mathbf{w}^n) + \frac{\partial \mathbf{R}(\mathbf{w}^n)}{\partial \mathbf{p}}\Delta\mathbf{p}^n + O(\Delta\tau^2)\end{aligned}$$

where $\mathbf{p}$ is the vector of primitive variables, $\Delta\mathbf{p}^n = \mathbf{p}^{n+1} - \mathbf{p}^n$ and $\frac{\partial \mathbf{R}}{\partial \mathbf{p}}(\mathbf{w}^n)$ is the Jacobian matrix with respect to the primitive variables at each point. The reason for choosing the Jacobian to be with respect to primitive variables is to make the differentiation simpler. Hence in addition to constructing the discrete residual vector of the scheme $\mathbf{R}$, the Jacobian matrix of the discrete residual is required for this method. There are three choices that need to be made in how an implicit method is classified:

- Temporal discretisation formula (time derivative approximation)
- The approximation (if any) to the Jacobian of $\mathbf{R}$
- The method of solving the linear system and to what accuracy

Each of these choices will be addressed in this section.

III.C. The time derivative and choice of solution method

The time derivative in Eq. (14) can be discretised with a simple backward differential operator. For time independent calculations for which the convergence is to steady state, the solution is independent of the choice of temporal discretisation and so the simplest, first order option is taken. Using this option in pseudo time and performing the linearisation outlined above, we obtain the approximate local system

$$\begin{aligned}\frac{\Delta\mathbf{w}^n}{\Delta\tau} &= -\left(\mathbf{R}(\mathbf{w}^n) + \frac{\partial \mathbf{R}}{\partial \mathbf{p}}\Delta\mathbf{p}^n\right) \\ \frac{\Delta\mathbf{w}^n}{\Delta\tau} + \frac{\partial \mathbf{R}}{\partial \mathbf{p}}\Delta\mathbf{p}^n &= -\mathbf{R}(\mathbf{w}^n) \\ \left(\frac{1}{\Delta\tau}\frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \frac{\partial \mathbf{R}}{\partial \mathbf{p}}\right)\Delta\mathbf{p}^n &= -\mathbf{R}(\mathbf{w}^n)\end{aligned} \tag{15}$$

where $\frac{\partial \mathbf{w}}{\partial \mathbf{p}}$ is the transformation matrix of conservative and primitive variables. For time dependent, unsteady calculations we need to increase the order of accuracy in time.[32] As a result, a second order form of Eq. (14) in real time is taken:

$$\frac{3\mathbf{w}^{n+1} - 4\mathbf{w}^n + \mathbf{w}^{n-1}}{2\Delta t} = -\mathbf{R}(\mathbf{w}^{n+1}) \tag{16}$$

This is a non-linear system of equations, which can be solved by introducing an iteration through pseudo-time τ to the steady state

$$\frac{\mathbf{w}^{n+1,k+1} - \mathbf{w}^{n+1,k}}{\Delta\tau} + \frac{3\mathbf{w}^{n+1} - 4\mathbf{w}^n + \mathbf{w}^{n-1}}{2\Delta t} = -\mathbf{R}(\mathbf{w}^{n+1})$$

Then we seperate the $(n+1)$ terms and compute a sequence of k approximations to the solution at time step $(n+1)$ starting from the solution at time step n to give

$$\left(\frac{1}{\Delta\tau} + \frac{3}{2\Delta t}\right)\Delta\mathbf{w}^k = -\left(\frac{3\mathbf{w}^k - 4\mathbf{w}^n + \mathbf{w}^{n-1}}{2\Delta t} + \mathbf{R}(\mathbf{w}^{n+1})\right)$$

Then using the same linearisation method in pseudo time for the residual gives us the following system we need to solve

$$\left(\left(\frac{1}{\Delta\tau}+\frac{3}{2\Delta t}\right)\frac{\partial \mathbf{w}}{\partial \mathbf{p}}+\frac{\partial \mathbf{R}}{\partial \mathbf{p}}\right)\Delta\mathbf{p}^k=-\left(\frac{3\mathbf{w}^k-4\mathbf{w}^n+\mathbf{w}^{n-1}}{2\Delta t}+\mathbf{R}(\mathbf{w}^n)\right) \tag{17}$$

We can write Eqs. (15) and (17) in a simpler global matrix form

$$A(\mathbf{p}^n)\Delta\mathbf{p}=-\mathbf{R}(\mathbf{w}^n) \tag{18}$$

where A represents the implicit system matrix. This equation encapsulates the implicit scheme. The ideal choice of A is such that it is relatively inexpensive to construct and solve the linear system for, but will still give an efficient convergence rate. The method which gives the most efficient convergence rate is the Newton-Raphson method for which we require:

- A large enough time step such that $\Delta\tau\to\infty$
- An exact Jacobian of $\mathbf{R}$
- Exact solution of the linear system

In the limit of close proximity to the exact solution the convergence is quadratic and we can achieve convergence to machine accuracy in about 8 iterations. Figure 1 shows a convergence plot for such an example, in this case 200 explicit iterations are first used to smooth out the flow field before the implicit solver starts. Despite these superior convergence properties over linear convergence methods, in practical CFD calculations it is unnecessary to solve the equations to such high accuracy. Solving the linear system exactly (for example with a direct solver) is both extremely costly in both memory requirements and arithmetic operations (which are often of the order $\mathcal{N}^3$). The fact that quadratic convergence is only achieved near the exact solution means that much effort is expended in reducing the norm of the residual to this region of convergence. If we also take into account the high cost of formulating and storing the exact Jacobian, particularly if the Jacobian is second order, it is clear that it is impractical to use this method.

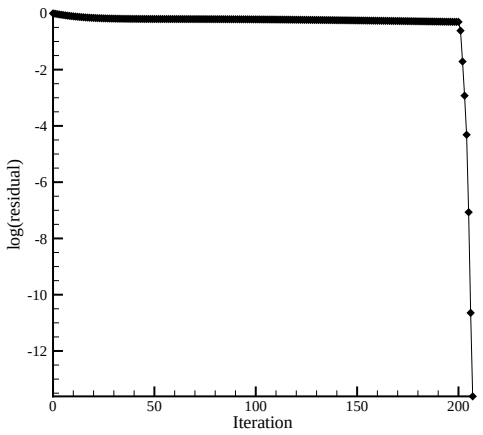


Figure 1. An example of quadratic convergence (after 200 explicit iterations)

Instead, a more efficient method is obtained by choosing an approximate Jacobian with finite values of $\Delta\tau$ (for additional stability) and an inexact linear system solver (such as an iterative method). Despite losing the highly desirable quadratic convergence, such methods tend not to have the undesirable properties of Newton-Raphson methods. In addition, they are much easier to formulate as the Jacobian does not need to be computed exactly thus reducing memory requirements and computational costs incurred when calculating terms that could otherwise be neglected as insignificant in increasing the convergence of the system using an inexact solver. Using an approximate Jacobian will also improve the preconditioning of the system and help resolve issues with parallel computations that will be addressed at a later date.

III.D. Constructing the approximate Jacobian

If we consider the structure of the Jacobian, it has local contributions consisting of the partial derivatives of the residual at i, with respect to the primitive variables at every point j within the domain Ω. In principle $\mathbf{R}$ may be a function of $\mathbf{p}_j\forall j\in\Omega$ so the global Jacobian can be written as an $N\times N$ block matrix. Since since $\mathbf{R}_i$ and $\mathbf{p}_j$ are vectors of size $(D+2)\times 1$ then each block (denoted $\frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j}$) will consist of a matrix of dimension $(D+2)\times(D+2)$ formed by differentiating the $(D+2)$ components of $\mathbf{R}_i$ with respect to the

$(D+2)$ components of $\mathbf{p}_j$. Each block matrix can be found from differentiating the right hand side of Eq. (8) or Eq. (12) so

$$\begin{aligned}\frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j} &= \frac{\partial}{\partial \mathbf{p}_j}\left\{\sum_{k=0}^{n_i} b_{ik}\mathbf{f}_{k-\frac{1}{2}} + \sum_{k=0}^{n_i} c_{ik}\mathbf{g}_{k-\frac{1}{2}}\right\} \\ &= \sum_{k=0}^{n_i} b_{ik}\frac{\partial \mathbf{f}_{k-\frac{1}{2}}}{\partial \mathbf{p}_j} + \sum_{k=0}^{n_i} c_{ik}\frac{\partial \mathbf{g}_{k-\frac{1}{2}}}{\partial \mathbf{p}_j} \qquad \forall i,j \in \Omega \end{aligned} \tag{19}$$

The final statement can be made since the shape functions are found purely from geometrical considerations and are not dependent on the flow variables. Using Eq. (9) we can write the partial derivatives of the midpoint flux as

$$\frac{\partial \mathbf{f}_{k-\frac{1}{2}}}{\partial \mathbf{p}_j} = \frac{1}{2}\frac{\partial}{\partial \mathbf{p}_j}\left\{\mathbf{f}(\mathbf{p}_L) + \mathbf{f}(\mathbf{p}_R) - |\tilde{A}(\mathbf{p}_L,\mathbf{p}_R)|(\mathbf{p}_L,\mathbf{p}_R)\right\} \tag{20}$$

It is clear from Eq. (20) that $\mathbf{R}_i$ is not a function of $\mathbf{p}_j \forall j \in \Omega$ but only on the points that belong to $\mathcal{M}(i)$ that are used to form $\mathbf{f}_{k-\frac{1}{2}}$ and $\mathbf{g}_{k-\frac{1}{2}}$. As a result most of the elements of the Jacobian are zero resulting in a sparse matrix. Just how sparse the matrix is depends on the order of spatial discretisation.

For a first order scheme Eq. (20) is dependent only on the $\mathcal{M}(i) = n_i$ points that make up the stencil. As a result, for every Riemann problem solved there are two contributions to the Jacobian, one for the star point at the diagonal (since $i = j$) and one for the neighbouring point. The second order scheme has reconstructed left and right sides of the Riemann problem which are each dependent on the gradients of the left and right side variables as well. As a result, there are additional non-zero contributions to the Jacobian as can be seen from Eq. (20), corresponding to each point within the neighbouring stencils that form the least squares gradients (recall $\mathcal{M}(i) > n_i$ for second order) for the higher order reconstruction.

It is desirable to keep the number of non-zero entries as low as possible for several reasons. Firstly, the memory requirements are reduced. Secondly each inner iteration is faster to perform since the matrix-vector multiplications require less operation counts. Finally, the omission of the additional terms will decrease the stiffness and ill-conditioning of the matrix making it easier to solve since the Jacobian will be more diagonally domainant.

To avoid these difficulties, we omit the additional second order block contributions to the Jacobian. The fill-in is then the same as that for an exact first order Jacobian, though with the reconstructed second order variables Eq. (10) used. The viscous flux Jacobian is approximated in the same way. Despite this mismatch between the Jacobian and residual, the corresponding reduction in the sparse fill-in means that this approach will be much more cost effective when used in conjuction with the inexact linear solver. Experience has shown that a relaxation factor $\omega = 0.5$ aids the convergence considerably.

The Jacobian must also contain contributions that account for the boundary conditions. Recall that the boundary conditions are imposed on the star point by the halo points which have flow variables obtained from their corresponding interior points. Obviously the halo points are not updated by the linear solver as they do not belong to the domain Ω, and as there is not a residual vector for the halo points there will not be any additional rows or columns in the Jacobian. Instead, for the star point i on the boundary we can use the chain rule to calculate the Jacobian contributions due to the halo points, so

$$\frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_j} = \frac{\partial \mathbf{R}_i}{\partial \mathbf{p}_h}\frac{\partial \mathbf{p}_h}{\partial \mathbf{p}_j} \tag{21}$$

where the matrix $\frac{\partial \mathbf{p}_g}{\partial \mathbf{p}_j}$ is non-zero only when j is for the star point on the surface i, and the corresponding interior point for the ghost point. As such the fill in of the Jacobian is not altered by the boundary Jacobians. If there are any "blanked points" (i.e. any points that lie outside Ω but are part of the linear system) then to prevent the matrix becoming singular, the block matrix on the diagonal is replaced with the identity matrix. The residual is set to zero so has no effect on the convergence.

The full approximate implicit system matrix A can be written in the following way

$$A = A_{diag} + A_{uns} + A_{mean} + A_{BC} + A_{visc}$$

where A_{diag} is the diagonal contribution seen in Eq. (15), A_{uns} is the additional contribution to the diagonal for unsteady calculations seen in Eq. (17), A_{mean} is the mean flow Jacobian formed by the blocks in Eq. (19), A_{BC} are the boundary Jacobians Eq. (21) and A_{visc} is the Jacobian due to viscous contributions.

Each of these matrices can be treated separately and summed at the end to ease the formulation if needed. The Jacobian is sparse, but due to the often random distribution of the points that are associated with meshless methods, it does not have a fixed pattern that can be readily exploited in storing the matrix and solving the system. Instead, the Jacobian will have sparsity patterns similar to those of unstructured grids. There is no need to store all of the zero blocks and we implicitly know the dimension of the non-zero blocks, so we can exploit the sparse, block nature of the Jacobian through the use of a Block Compressed Sparse Row (BCSR) format to explicitly store the matrix within a data structure.

III.E. Linear solver method

In this sub-section we will write the linear system Eq. (18) in the more conventional mathematical form

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

An iterative method is used to solve this system inexactly using several preconditioned Krylov steps. This is done by making successive approximations to an initial solution vector using matrix-vector multiplications until the solution converges to an acceptable level. The space which spans these approximations is called the Krylov subspace and generally only contains a good approximate solution if the system is relatively well conditioned. As such, we modify the original problem to obtain a better Krylov subspace by using a preconditioner M, which results in effectively solving the modified problem

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$$

The preconditioner is designed so that $M^{-1}A$ in some sense approximates the identity, so $M^{-1}(\mathbf{b} - A\mathbf{x}_k)$ can be expected to approximate the error in an approximate solution $\mathbf{x}_k$. The next approximate solution $\mathbf{x}_{k+1}$ can then naturally be obtained by taking

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k) \tag{22}$$

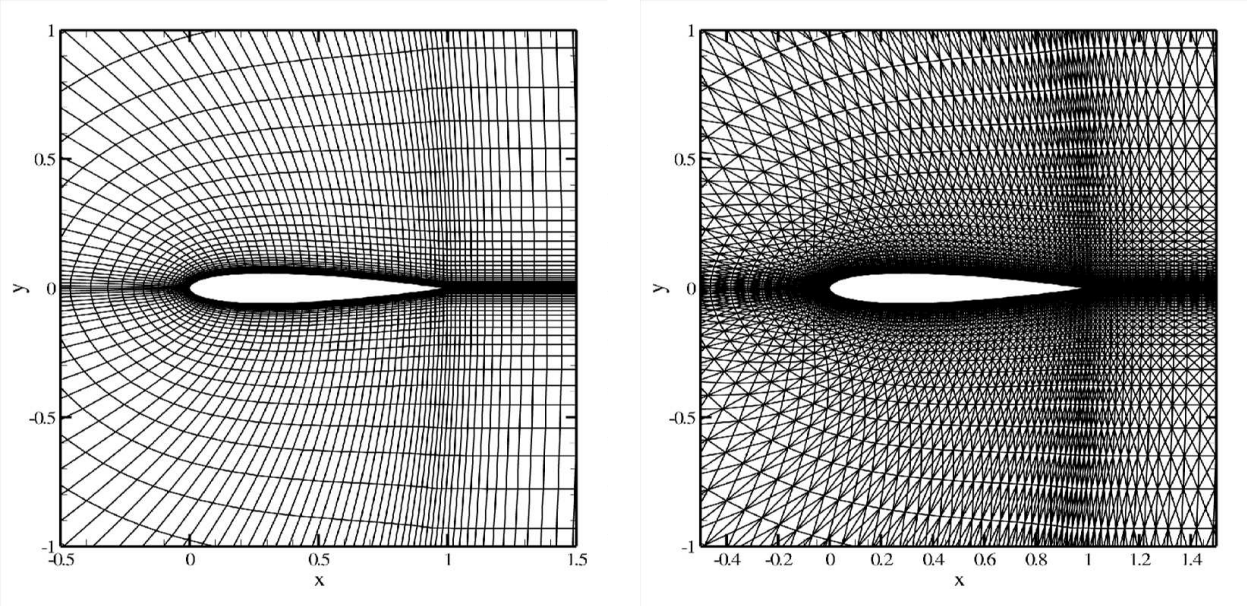
For M equal to the diagonal of A, this is the Jacobi iteration, for M equal to the lower triangle of A it is the Gauss-Seidel iteration, and for M of the form $\omega^{-1}D - L$, where D is the diagonal of A, L is the strict lower triangle of A and ω is a relation parameter, it is the successive over-relaxation (SOR) iteration. Such methods are used in the meshless literature in,[9,10] which both use split Jacobians and Gauss-Seidel methods.

This work however uses a modified version of Eq. (22) with additional parameters introduced into the iteration. It is called the generalised conjugate residual (GCR) method[33] as the successive approximations to the solution vector are formed by minimising the norm of the residual vector. The matrix A is not required to be symmetric and positive-definite, making it advantageous in that the method depends on A rather than the normal equations formed by A.

Various other algorithms have been tested for finite volume codes[34] and it was concluded that the choice of method is not as crucial as the preconditioning. The preconditioning strategy is based on a Block Incomplete Lower-Upper (BILU) factorisation,[35] which computes upper and lower triangular matrices whose product gives the original matrix. Furthermore we use the method with zero fill-in (BILU(0)) which means that the sparsity pattern on the Lower and Upper matrices is the same as those in the original matrix.

IV. Solver validation

Results are now presented for validation and to build confidence in the solver for when it is to be used for cases with stencils constructed using the method described in section V, with the results of those cases presented in section VI. Standard NACA 0012 aerofoil test cases, with non-dimensionalised chord length $c = 1.0$ for inviscid and viscous flows are presented. Also included is an unsteady simulation and a 3D problem. Comparisons are made with the parallel multiblock (PMB) flow solver,[36] which is an established research code, so further validation for the steady state results is not considered here. The PMB calculations



(a) Finite volume grid used by PMB

(b) Meshless connectivity used by PML

Figure 2. NACA0012 aerofoil used for 2D calculations

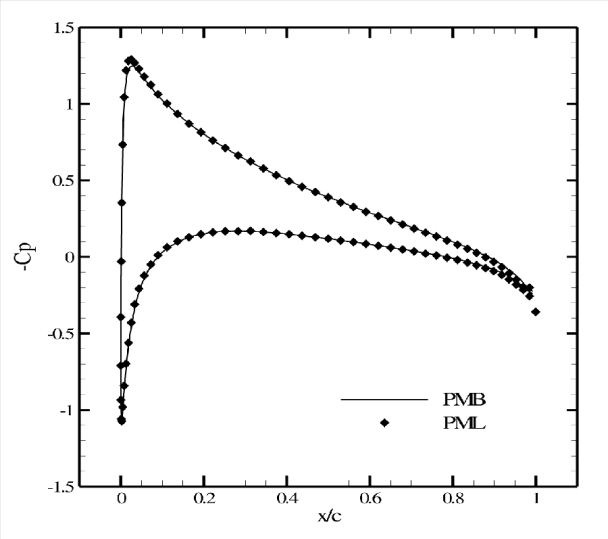
are made using a structured grid of 9408 points, 36 of which are on the solid wall boundary, with a mimimum grid spacing of 10^{-3}. This point distribution is also used by the PML solver, with stencils consisting of the points as shown in Fig. 12. Both the finite volume and meshless connectivities are shown in Fig. 2. In these results, the order of the polynomial used to approximate the function in Eq. (1) is $m = 4$, so it is linear with an xy cross term added for stability. The convergence histories are presented as a function of the number of iterations. For each test case the flow field is first smoothed out by 200 explicit iterations, after which the implicit scheme described is used. The implicit time step in Eq. (15) is fixed at $\Delta\tau = 10.0$ for all steady cases. There is yet to be an effective local time step scheme implemented.

IV.A. Steady state 2D inviscid

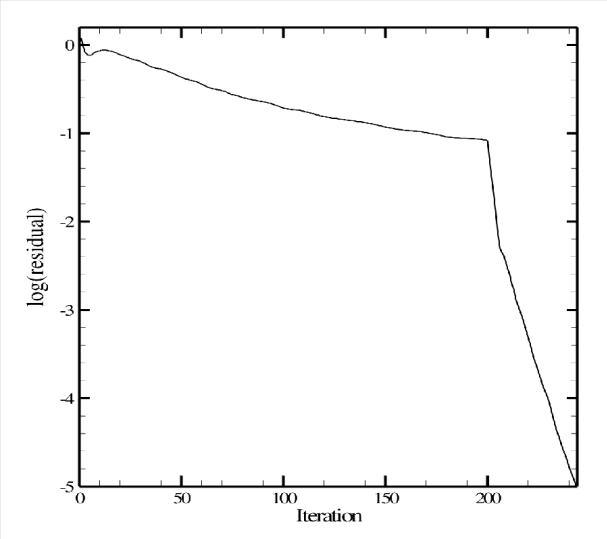
The first case in Fig. 3 is a subsonic flow ($M_\infty = 0.5$, $\alpha = 3.0°$) and we can see that the method is very efficient once the implicit scheme begins. Convergence is achieved in 250 iterations total. The transonic case Fig. 4 ($M_\infty = 0.8$, $\alpha = 1.25°$) is more expensive due to the two shock waves of moderate strength on the upper and lower surfaces of the aerofoil. Convergence is still achieved rapidly in less than 300 iterations. The supersonic case Fig. 5 ($M_\infty = 1.2$, $\alpha = 7.0°$) also shows the efficiency of the method. Apart from the spike at the 200th iteration the plot shows greatly accelerated convergence. This spike represents the increase in the residual that occurs in the one iteration when the method changes from explicit to implicit.

IV.B. Steady state 2D viscous

Viscous results are presented in Fig. 6 at $M_\infty = 0.5$, $Re = 5000$ and $\alpha = 3°$. For further validation, velocity profiles along the aerofoil are shown in Fig. 7. The angle of attack is at $\alpha = 0°$. Comparisons are made at $x = 0.135c$ (where maximum suction occurs), $x = 0.4c$, $x = 0.65c$ and $x = 0.9c$, which all show good agreement. The separation point is also at the same location 0.82c.

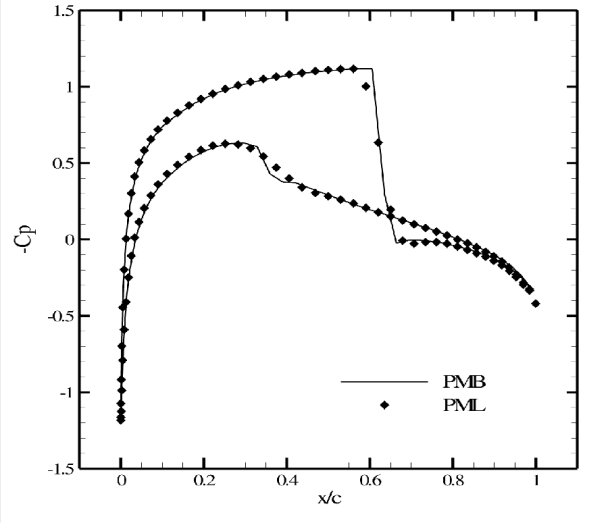


(a) Surface pressure distribution

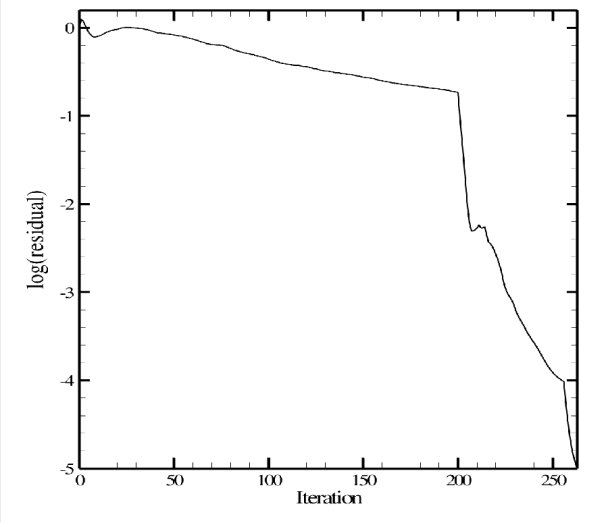


(b) Convergence history

Figure 3. NACA0012 $M_\infty = 0.5$, $\alpha = 3.0°$

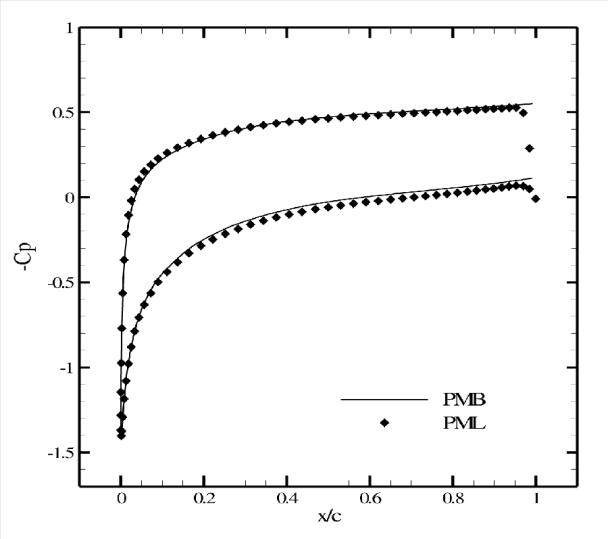


(a) Surface pressure distribution

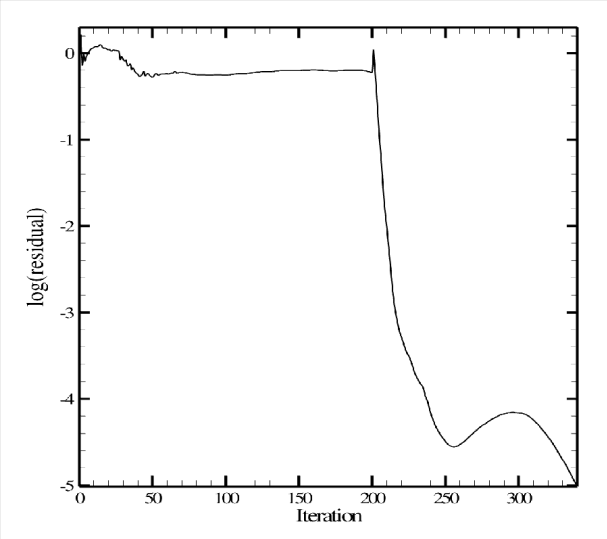


(b) Convergence history

Figure 4. NACA0012 $M_\infty = 0.8$, $\alpha = 1.25°$

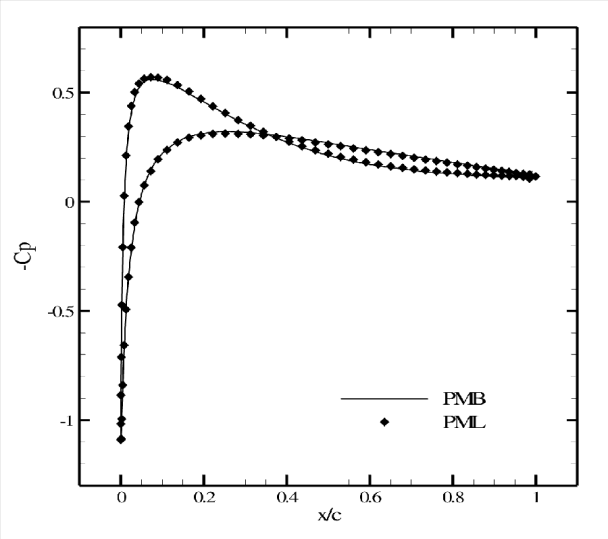


(a) Surface pressure distribution

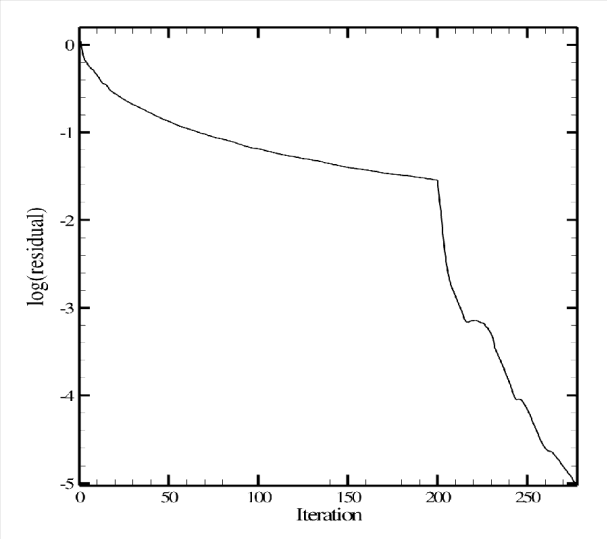


(b) Convergence history

Figure 5. NACA0012 $M_\infty = 1.2$, $\alpha = 7.0°$

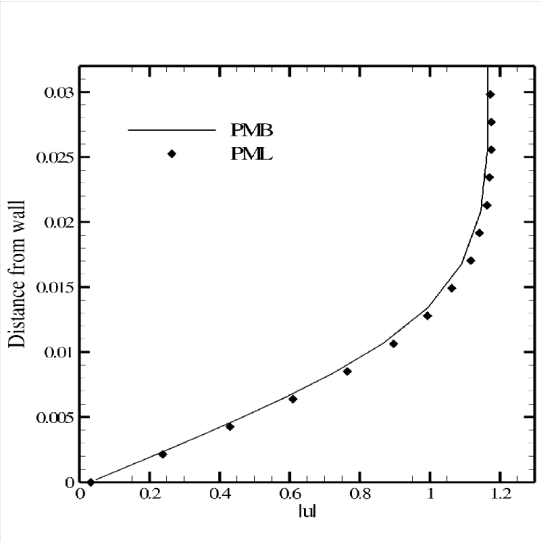


(a) Surface pressure distribution

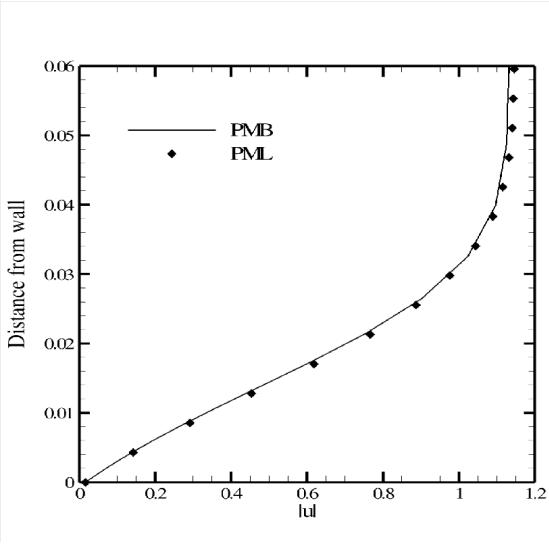


(b) Convergence history

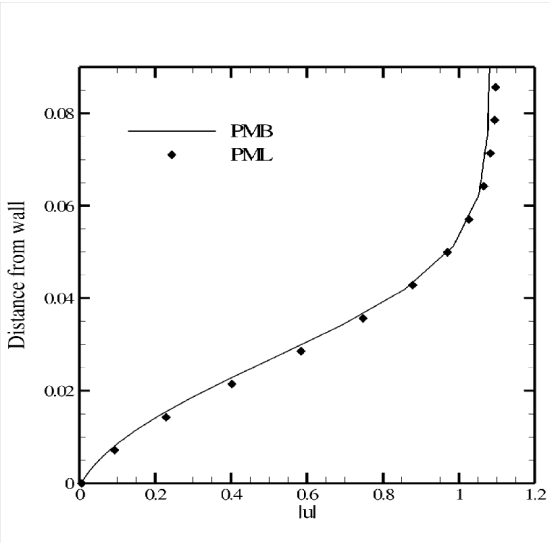
Figure 6. NACA0012 $M_\infty = 0.5$, $\alpha = 3.0°$, $Re = 5000$



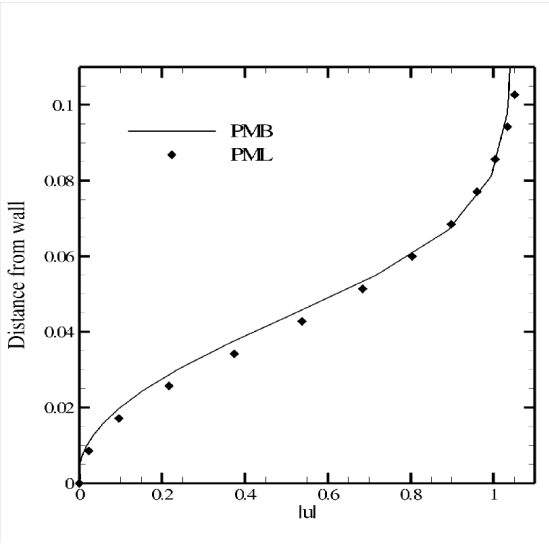
(a) x=0.135c



(b) x=0.4c



(c) x=0.65c



(d) x=0.9c

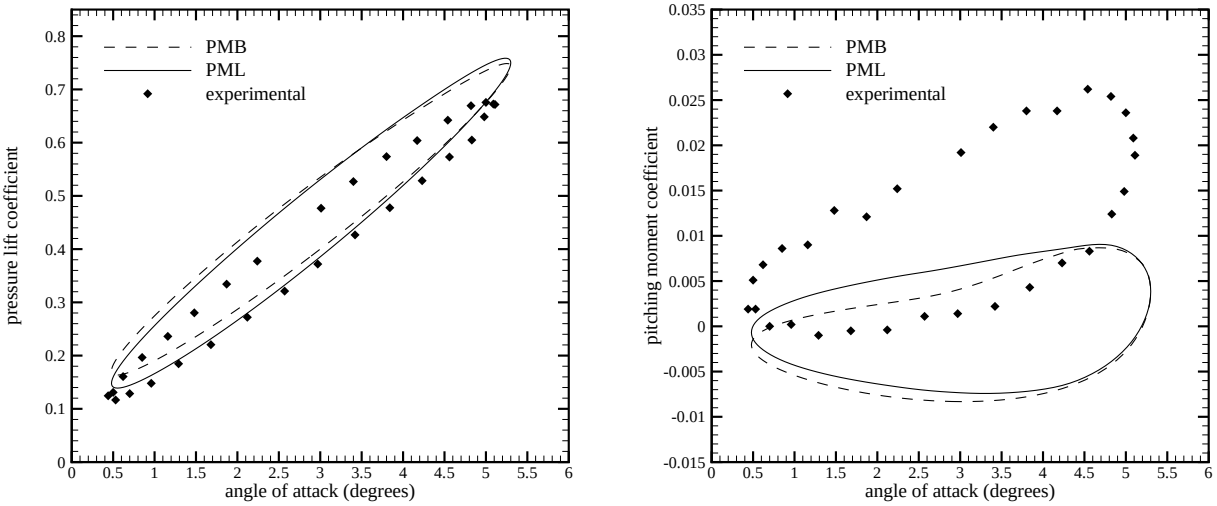
Figure 7. NACA0012 $M_\infty = 0.5$, $\alpha = 0.0^\circ$, $Re = 5000$

IV.C. Unsteady 2D inviscid

Figure 8 presents results of forced pitching simulations around the quarter chord $c = 0.25$ of the aerofoil, prescribed by a sinusoidal motion

$$\alpha(t) = \alpha_0 + \alpha_a \sin(2kt)$$

Comparisons are also made with the experimental data of the AGARD CT1 test case.[37] The configuration CT1 exhibits intermittent shock motion during the cycle, at freestream Mach number 0.6 with a reduced frequency of $k = 0.0808$, a mean incidence of $\alpha_0 = 2.89°$ and a pitch amplitude of $\alpha_a = 2.41°$. The experimental data is at Reynolds number 4.8 million, though the CFD comparisons are inviscid. Five motion cycles with 128 steps in each were simulated. Pressure lift and pitching moment coefficients are shown in Figs. 8(a) and 8(b) respectively in which the overall agreement between the 2 codes can be determined.

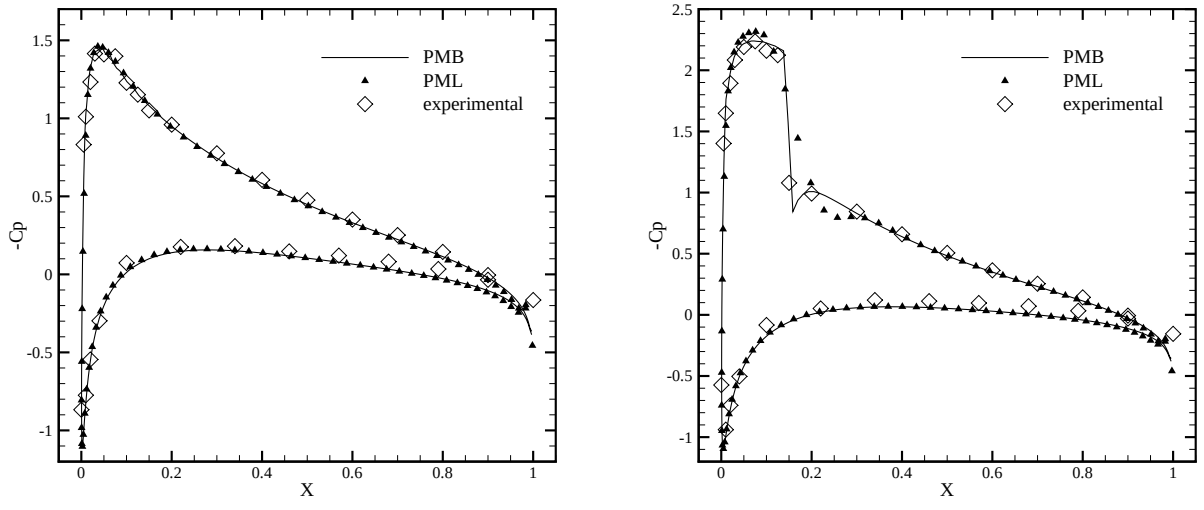


(a) Pressure lift coefficient

(b) Pitching moment coefficient

Figure 8. Lift coefficients for moving NACA0012 aerofoil

Fig. 9 presents unsteady pressure distributions at two angles of attack, the comparisons are excellent except at the shock location, with the PMB solver better able to resolve the shock cleanly.



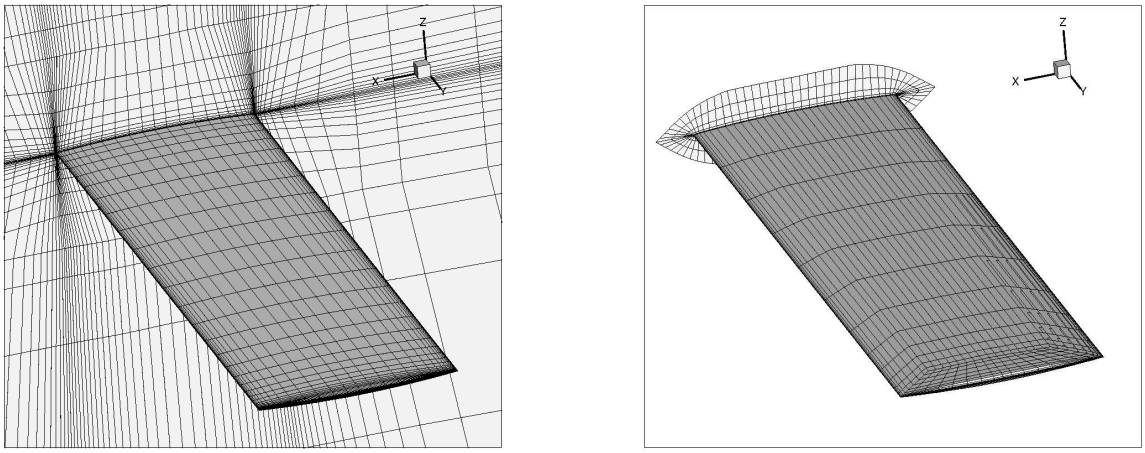
(a) $\alpha = 3.84°$ (upstroke)

(b) $\alpha = 4.82°$ (downstroke)

Figure 9. Unsteady pressure distributions for forced pitching motion test case of AGARD CT1

IV.D. Steady state 3D inviscid

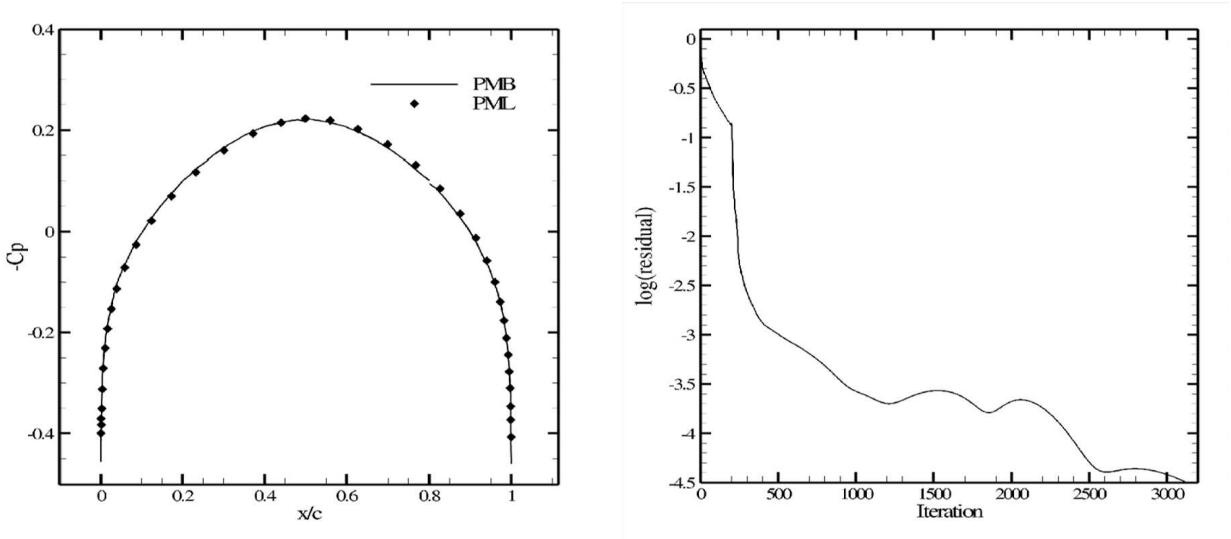
Finally 3D results are presented. This case is for a Goland wing with 188,000 points in the domain. The surface of the wing can be seen for both the PMB and PML cases in Figs. 10(a) and 10(b). Note that the surface used for the meshless calculation is made up of quadrilateral elements, which were found to be much better for imposing the necessary boundary conditions in PML than triangular elements.



(a) Wing surface from PML (b) Wing surface from PMB

Figure 10. Goland wing surface

The surface pressure comparison with PMB at $M_\infty = 0.85$ and $\alpha = 0°$, is good as can be seen in Fig. 11(a). However Fig. 11(b) shows the slow convergence rate, which is most likely due to the lack of an efficient local time step within PML.



(a) Surface pressure distribution (b) Convergence history

Figure 11. Goland wing $M_\infty = 0.85$, $\alpha = 0°$

V. Selecting the stencils for the meshless solver

V.A. Scheme requirements

In this section we outline a preprocessor method, that will give us the stencils required for the flow solver described in the previous sections. Despite some of the point generation methods in the literature,[11,12] we use points obtained from structured grids generated around individual components/bodies, which are allowed to overlap to cover the physical domain. Moving bodies are accomodated for by moving the component grids of each body separately. Even though some user input is required to generate the points, the scheme is much simpler than the generation of a full finite volume grid within the domain, and the conditions on the quality of the grid are a lot less strict.

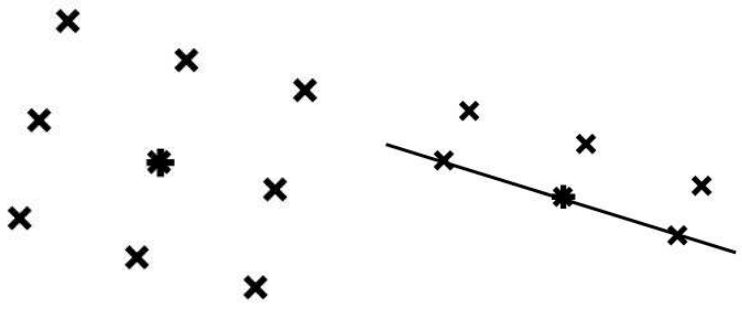
The meshless preprocessor is designed so that it fits a number of requirements. First, the stencil selection has to be fully robust and give stencils that will allow the flow solver to be as accurate as possible. Second, the selection has to be fully automatic - if it requires a large amount of user input then the advantages of a meshless method are lost. Third the selection has to be as fast as possible for repeated use in unsteady calculations.

For a first attempt the stencil selection method was to ignore the component grid connectivities and view the domain only as a distribution of points. A quadtree search (in 2D) was used to find point candidates and quadrant[6] and Delaunay techniques[18] were used to select the points for the stencils. The resultant stencils were tested by comparing the analytical gradient of a function against the least squares gradients formed using the stencils. It was found however that the choice of search regions was too heuristic, leading to great expense as search regions were increased/decreased depending on whether too few/many points were found. It was also difficult to determine the correct number if anisotropic point distributions were used, as a large number of the nearest neighbours are collinear. The selection steps were also found to be inaccurate with these point distributions and it was found that testing the stencils is only worthwhile if we know the form of the function that will need to be reconstructed in the flow computation, which clearly we do not. This is because the testing function may have a low gradient in the direction of a high flow gradient.

As a result of these problems, the notion of a "pure" meshless scheme was abandoned for the preprocessor stage, and the original grid connectivities are used as an aid to select the stencils. Computational time in the search algorithms is saved and the quality of the stencils improved using this aid. Through making this choice, several more requirements as well as the previous three are placed on the scheme. Fourth, despite using the grid connectivity as a guide, we still want to retain the simplicity of the meshless method, so we want to avoid techniques such as cutting out holes in the domain, and having to put precedence on a single grid, or regions of several grids when constructing the stencils. Fifth, the selection should be global, but ideally the meshless stencils will be the same as the original grid stencils in regions with no overlap.

The full user input consists of:

1. A list of points with coordinates and their neighbours in the original grid connectivity (for both interior and boundary points, see Fig. 12).
2. A list of boundary elements, and the boundary type, with a reference point for each edge that lies on the inside of the domain in that grid.



(a) Interior point (b) Boundary point

Figure 12. Structured grid connectivity

V.B. Preprocessor steps

It is with the considerations outlined above that the following scheme was developed and is discussed here for the 2D case. The process should generalise to 3D immediately. There are four primary operations that the code performs to construct the stencils required by the meshless flow solver.

1. Check if boundaries from separate grids overlap in any way. This can be the case if we wish to construct a body with complex geometries using individual component grids. If overlap occurs then the boundaries must be altered accordingly.

2. Identify the points that lie within a solid body, which can occur when the grids overlap. These points are "blanked" and removed from the flow computation.

3. Select the point stencils for the remaining non-blanked points.

4. Make sure that the resultant stencils respect the boundaries, that is, to check that stencils are not formed with points that lie on the opposite side of a solid body.

Each of these operations will be described in the following sub-sections.

V.C. Redefining the boundaries

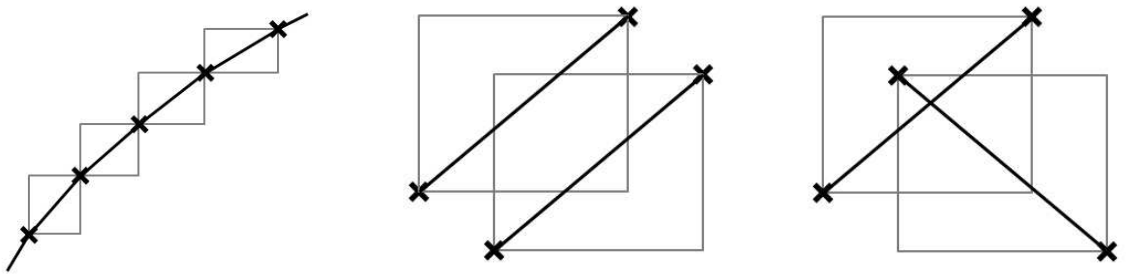
This first step is to identify and fix any boundary overlap. To do this we check each boundary edge of one component grid with each edge boundary edge from the other grids to see if they intersect. This is done through the use of a geometric intersection algorithm. As this process can be expensive, we use the method described in reference.[38] Bounding boxes are formed around each boundary edge so that the edge coordinate limits form the corners of the box Fig. 13(a). We then compare each bounding box from one grid with those from the other grids to see if the boxes intersect. This is done by seeing an object in D-dimensional space as a point in $\mathbb{R}^{2D}$ with the point coordinates as the limits of the object. So a box in 2D would be seen as a point in 4D with coordinates

$$\mathbf{x} = \{x_{min}, y_{min}, x_{max}, y_{max}\}$$

We can then construct a search tree[39] containing each of these higher dimensional points. If all the boxes lie within a domain Ω with coordinate limits $\mathbf{x}_{\Omega,max}$, $\mathbf{x}_{\Omega,min}$, then two box elements a and b will intersect if they satisfy

$$\begin{pmatrix} x_{\Omega,min} \\ y_{\Omega,min} \\ x_{a,min} \\ y_{a,min} \end{pmatrix} \leq \begin{pmatrix} x_{b,min} \\ y_{b,min} \\ x_{b,max} \\ y_{b,max} \end{pmatrix} \leq \begin{pmatrix} x_{a,max} \\ y_{a,max} \\ x_{\Omega,max} \\ y_{\Omega,max} \end{pmatrix} \tag{23}$$

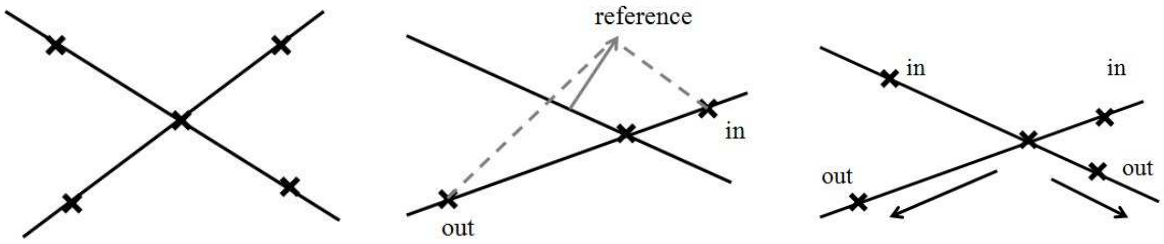
The tree is then traversed with the bounding boxes from grid 1 to see if any of the bounding boxes from the remaining grids overlap. If this is the case then we must explicitly check if the edges within intersect using a simple segment intersection algorithm to distinguish between the cases in Figs. 13(b) and 13(c).



(a) Edge bounding boxes (b) Boxes intersect - edges do not (c) Boxes intersect - edges intersect

Figure 13. Items in the tree search

If edges intersect then it is necessary to redefine the boundaries. This is done by adding a new point where the intersection occurs as in Fig. 14(a). We then use another ray algorithm to decide which points



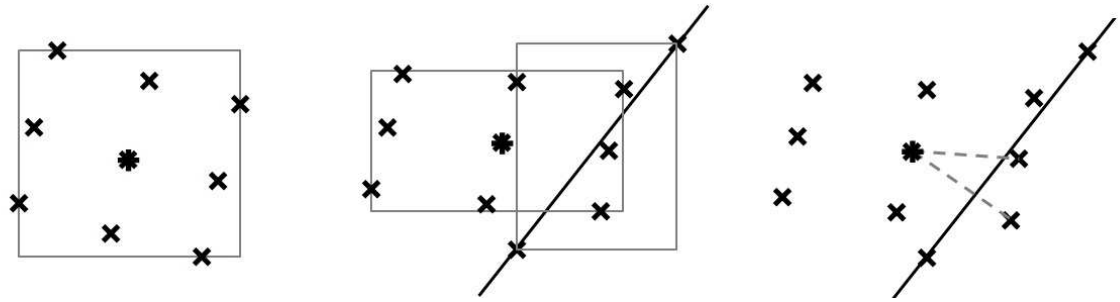
(a) Adding a point at the intersection (b) Identifying inside/outside edges (c) Flooding the edges

Figure 14. Boundary intersection diagrams

that form the edges lie inside the new boundary and need to be flagged (blanked) as such. The ray is formed from each end point of the intersecting edge and a reference point that lies above the other edge inside the domain. If the ray intersects anywhere along the infinite line lying collinearly to the other edge then it lies outside the intersection, if not it lies inside as in Fig. 14(b). If the boundaries are solid walls then a point lying outside the intersection is inside the boundary wall and needs to be blanked. Flood operations are performed to reassign the other edges that form the same boundary as in Fig. 14(c). If there are more than two grids that will form the final domain then grids 1 and 2 are compared, then the resultant domain is compared with grid 3, then the resultant domain is compared with grid 4 and so on.

V.D. Blanking points

It is relatively simple to modify the search step described above to identify any other points that must be blanked. This is done by checking each grid individually as before with a similar geometric intersection technique. This time we form boxes around the new boundary edges of one grid, and perform a search with a tree, that is made up of the boxes formed around the grid stencils in every other grid Fig.15(a). Then if a stencil overlaps with a boundary edge i.e. it satisfies Eq. (23), then it could be that some of the points lie behind a boundary wall and possibly even inside a solid body.



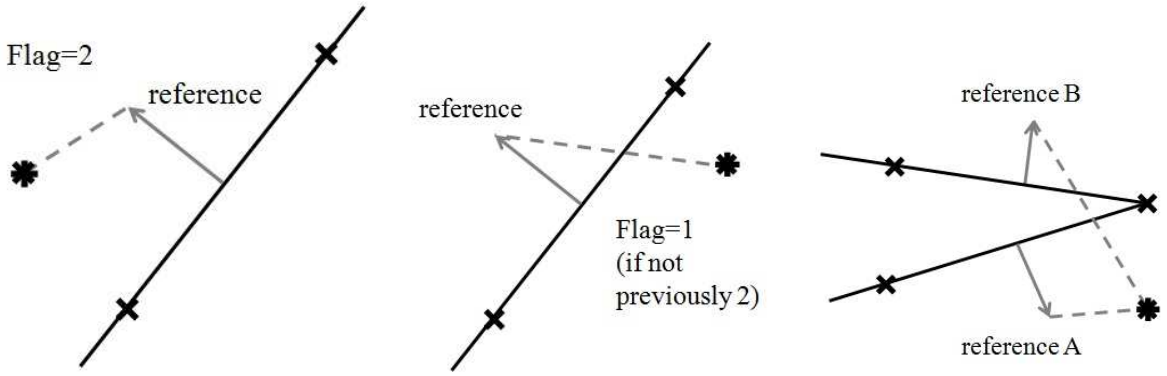
(a) Stencil bounding boxes (b) Box intersection (c) Remove points from stencil

Figure 15. Items in the tree search

If such an intersection occurs as in Fig.15(b), then for each stencil in the tree we form rays between the star and every neighbouring point within. If a ray intersects with the boundary edge then its point is removed from the stencil as it must lie behind a boundary Fig.15(c). After this step is performed globally we are left with a separate set of points that lie inside each solid body, and a set for the remaining points in the domain. The sets are not connected.

It only remains to identify which sets lie inside the solid bodies. For each boundary edge we check the star points of the stencils that overlap. A ray is formed from the edge reference point and the star. If the ray does not intersect the edge as in Fig.16(a) then the star is given the flag 2, if it does intersect and the flag is not already set to 2 then the flag of the star is set to 1 as in Fig.16(b). This system of flags is needed to prevent the blanking of points that lie behind a boundary wall but are still inside the domain, which can occur as in Fig.16(c) in which reference point B would flag the point as outside the domain (i.e. flag=1) if it were not for reference point A flagging the point as inside the domain (flag=2). As this is a global flag,

then if the flag is at any stage set to 2 then it cannot be a blanked point. When each stencil is checked in the grid, all of those points that are flagged as 1 lie within a boundary. A flood operation is then performed in which the neighbouring points of a blanked point are similarly blanked.



(a) Point is inside (b) Point is outside (c) Situation with possible multiple flaggings

Figure 16. Blanking points

V.E. Stencil selection

We now have a domain with properly defined boundaries, and points interior to solid bodies blanked as such. The interior stencils however are still grid dependent, the next step is to form stencils that effectively link the grids together to form the final meshless domain connectivity.

It is important to note that structured grids are generated so that regions where rapid variation of the flow variables is expected, such as near boundary walls or in an aerofoil wake have cells that are anisotropic to capture these effects. This means that the cells are highly stretched, with very fine spacings in the direction in which these effects will occur. The grid in Fig. 2(a) is an example of a typical structured grid designed to capture laminar flow around an aerofoil.

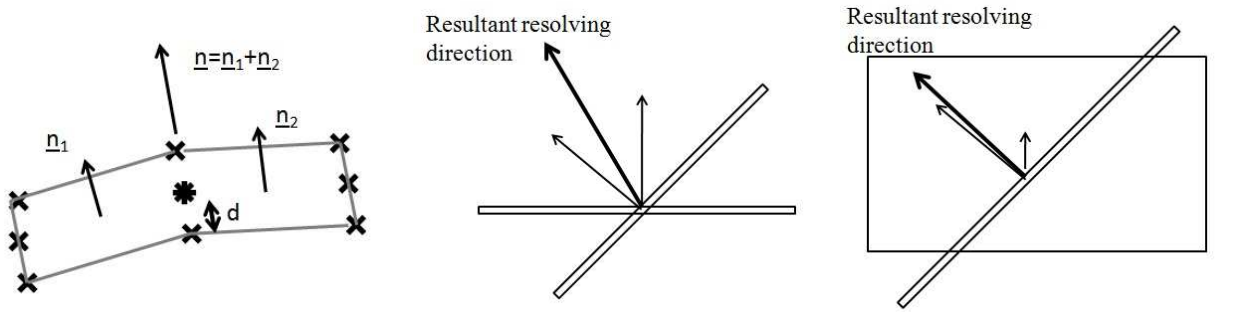
For conventional finite volume methods, the engineer designs the grid using his own intuition so that anisotropic cells are constructed in these regions. If we obtain the points for a meshless computation by overlaping two or more grids, we need a way to automatically construct meshless stencils that will reflect the anisotropic nature of each of the grids to capture the flow accurately. As a result nearest neighbour algorithms will not be sufficient. It is instead necessary to determine a direction in which we expect the flow variables to change the most. We can then best reconstruct these functions by selecting the stencils so that the points are fine in this direction. The original grid connectivities are used to determine this direction and the level of anisotropy required by assuming that the direction in which the grid stencils are finest is where the flow needs to be resolved the most.

With this, the resolving vector $\mathbf{v}$ of a grid stencil is defined as the direction in which the flow gradients are expected to be highest, with its length reflecting the strength of the gradients. To account for the topology of the original stencils, for structured grids this direction can be taken as the sum of the normals from the star to the points that form the edges that lie furthest away in the original stencil ($\mathbf{n}$ in Fig. 17(a)). This will ensure that the correct direction is defined for boundary points or points that lie near the boundary. We then define the length of this vector by first making it a unit vector $\hat{\mathbf{v}}$, and then dividing by the distance d (or some power p of the distance) from the star to the closest point in the original stencil

$$\mathbf{v} = \frac{\hat{\mathbf{v}}}{d^p} \tag{24}$$

This means that a small distance d (like for a boundary layer grid stencil point) will give a large resolving vector, and a point in the far field will give a small resolving vector. When two or more grid stencils overlap, we can sum the resolving vectors of the overlapping stencils (if more than one stencil from the other grid overlaps we only consider that with the lowest values of d - i.e. the finest stencil) to give a resultant resolving direction that gives an esimation of the flow direction considering all of the grids. The higher the value of p in Eq. (24), the more influence the finer stencils have. In this work the value $p = 4$ is used. Some examples

of the summation of the resolving vectors with representations of the overlapping original stencils are shown in Figs. 17(b) and 17(c). Points are then chosen for the meshless stencil that assume a high gradient in the resultant resolving direction. This will then allow us to select the points that will best capture the gradients that will be expected in this region from the overlap of the grids. To do this define a coordinate system as



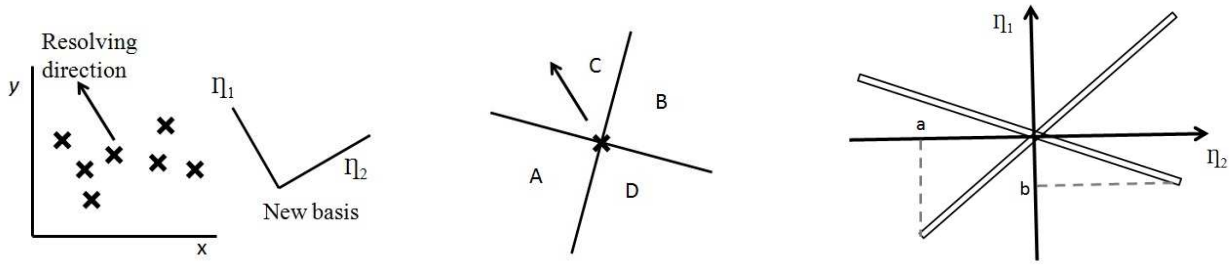
(a) Resolving vector for a grid stencil (b) Two anisotropic stencils overlap (c) An anisotropic and a regular stencil overlap

Figure 17. Defining the resolving direction

shown in Fig. 18(a). The basis η is chosen so that η_1 lies collinearly to the characteristic direction and η_2 lies orthogonal. We then form four quadrants around these axes as shown in Fig. 18(b) and search for the points in each quadrant. Hence quadrants A and B contain a list of points that lie to the left and right of the η_1 axis. We then choose the three points from each list that lie along η_2 (but not too far away). To balance the orthogonality and distance we define a merit function

$$\psi = \frac{\eta_2^2}{a^2} + \frac{\eta_1^2}{b^2}$$

This equation is similar to the equation for an ellipse, with the constants a and b as the semi-major axis and semi-minor axis. The values of a and b are determined from the projection of the various lengths in each of the overlapping stencils within the original connectivity onto the η_1 and η_2 axes. The smallest projection on η_1 is used as the value of b, and the smallest projection on η_2 is used as the value of a, Fig. 18(c). In this way we keep the stencils fine, but well conditioned.



(a) Defining the new coordinate system (b) Quadrants around resolving direction (c) Choosing a and b using stencil projections

Figure 18. Results for the steady store release case

One point is selected from quadrants A and B, each with the smallest values of ψ from their respective lists. Then from each quadrant A and B we select another two unique points, one with $\eta_1 > 0$ and one with $\eta_1 < 0$. Thus we have three points from these quadrants, which are relatively spread and not collinear, which would result in an ill-conditioned least squares matrix. The maximum value of ψ from these six points is noted ψ_{max}. To improve accuracy we then take one point from each of the quadrants C and D, each with the lowest value of ψ provided these values are less than ψ_{max}. This condition is to preserve the "stretched" nature of the stencils as it is possible that the nearest points within C and D are very far away as in Fig. 19. Here, the ellipse is at ψ_{max} and since points C and D exceed this they are not selected, the stencil would remain at 6 points.

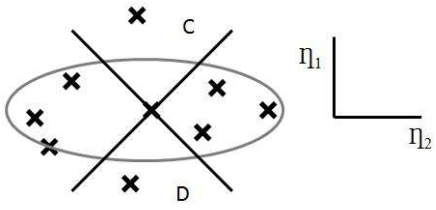


Figure 19. Points are not taken from quadrants C and D

This merit function is beneficial in that the stengths of the orthogonality and distance of the points from the star are dependent on the original connectivities. For highly stretched stencils near the boundary $a >> b$ and so the orthogonality of the points will have more of an influence on ψ. On the other hand if the overlapping stencils are more regular, then $a \approx b$ and the distance has more influence. In fact, for the special case $a = b$, then ψ resembles the equation for a circle and the orthogonality has no influence at all. It is for this case that the method effectively becomes a nearest neighbour algorithm.

V.F. Final boundary check

Once the stencils are created, the final step is to once again check that all of the stencils formed respect the boundaries. This step is required again because the previous boundary check used stencils from the original grid connectivity (and was partly to identify blanked points), while the current stencils are formed using points from all of the grids. It is possible for the points of a stencil to lie on the opposite side of a boundary wall where there are corners or sharp edges such as the trailing edge of an aerofoil. This is done using the same method as in subsection V.D, though now each boundary edge must be checked with all of the stencils in the domain, not just those that do not share the same grid as the boundary edge. Consequently this step is more expensive than the previous step.

VI. Results

Two simple test cases based on a store release that demonstrate the preprocessor are presented. They consist of a NACA 0012 aerofoil, with another at half the size acting as the store located beneath. Two grids with 9408 points are used, one for each aerofoil, which are allowed to overlap so the full domain consists of 18816 points. The pressure contour plots have been created by using a triangulation, so some parts of the plot may appear to be unsmooth as a result of this.

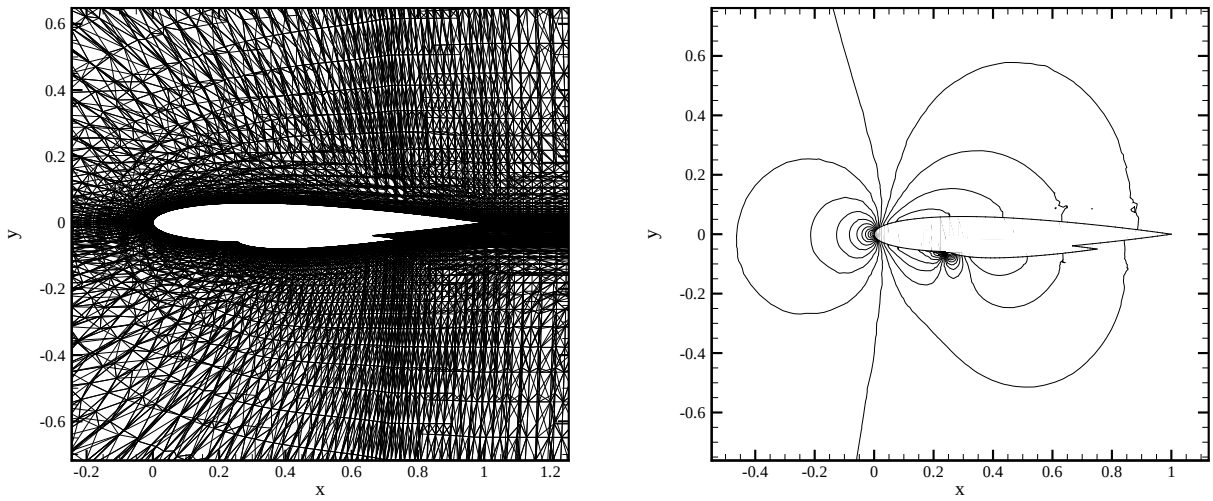
VI.A. Steady state overlapping body

The first case is when the leading edge of the aerofoil is at (0.25,-0.05), so that the bodies overlap. The resultant meshless connectivity generated by the preprocessor is shown in Fig. 20(a), and the pressure contours from a steady, laminar calculation at $M_\infty = 0.5$, $\alpha = 0°$, $Re = 5000$ is given in Fig. 20(b). It is clear that boundary overlap is accomodated for.

VI.B. Steady state overlapping body

The second case is an unsteady calculation in which the store is set in a forced descent from the main aerofoil. The initial location of the store leading edge is at (0.25,-0.15) and the descent veclocity is $\dot{y} = -0.1$. There are 200 time steps performed with $\Delta t = 0.05$ so the full time simulation takes place over 10 non-dimensionalised time units. The meshless connectivity and surface pressure contours at times 0, 5 and 10 are presented in Fig. 21. Vortex shedding occurs from the lower surface of the top aerofoil trailing edge. The moving of the bodies is done by moving each grid separately over one another. The full solver algorithm is as follows:

1. Construct the stencils for the initial time.
2. Perform the CFD calculation as a steady state calculation (till the residual is below 0.00001).
3. Change the point locations by moving the grid that the body to be moved belongs to.



(a) Stencil selection connectivity

(b) Pressure distribution

Figure 20. Results for the overlapping body case

4. Recalculate the stencils using the underlying grid connectivity.

5. Perform the CFD calculation as an unsteady time accurate calculation (till the residual is below 0.001).

Steps 3, 4 and 5 are repeated until the simulation finishes. When the 3 repeating steps begin, the stencil selection stage (steps 3 and 4) on average takes between $5\% - 10\%$ of this time.

VII. Conclusions and Outlook

A meshless solver that uses a Krylov subspace iterative method and a meshless stencil selection procedure are presented. The implicit scheme vastly improves the convergence of the equations and is used for steady and unsteady CFD calculations. The stencil selection scheme is used to construct clouds of points that are used by the meshless solver, and a preliminary unsteady test case with moving bodies shows promise for this method.

Future plans include developing the preprocessor so that it can accomodate 3 dimensional point distributions, a 6DoF model will be added too so that the exact movement of the bodies during the unsteady simulations can be determined using the aerodynamic forces.

The code will also be improved so that it can calculate turbulent flows (using a Spalart-Allmaras turbulence model) and perform parallel computations to speed up future, larger unsteady test cases.

Acknowledgments

This research has been supported by the Engineering and Physical Sciences Research Council and BAE SYSTEMS.

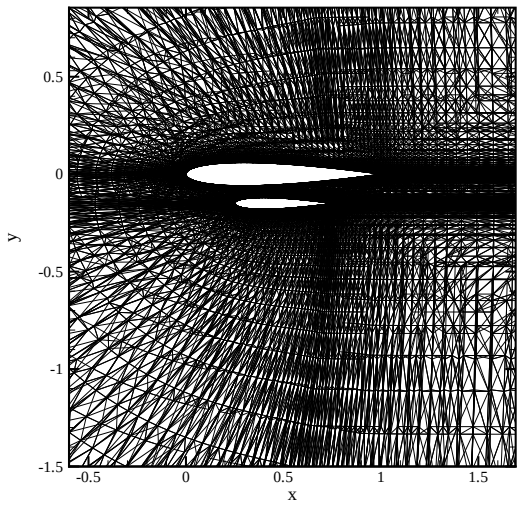
References

[1]Belytschko, T., Krongauz, Y., Organ, D., Fleming, and M., Krysl, P., "Meshless methods: an overview and recent development," Comput Methods Appl Mech Eng 139:347

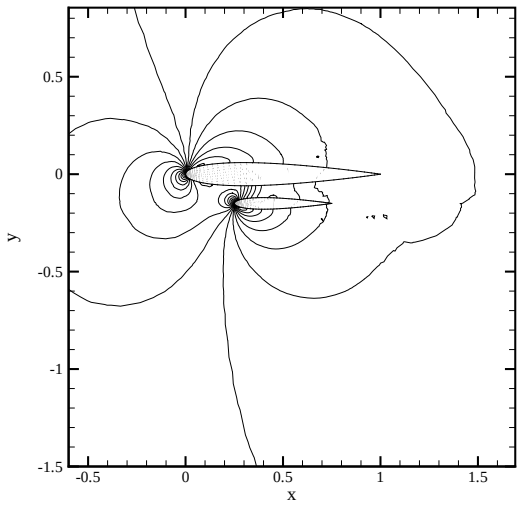
[2]Fries, T. and Matthies, H., "Classification and overview of meshfree methods," Department of Mathematics and Computer Science, Technical University of Braunschweig. Inf 20033

[3]Mavripilis, D.J., "Revisiting the least-squares procedure for gradient reconstruction on unstructured meshes," NASA CR-2003-212683, 2003

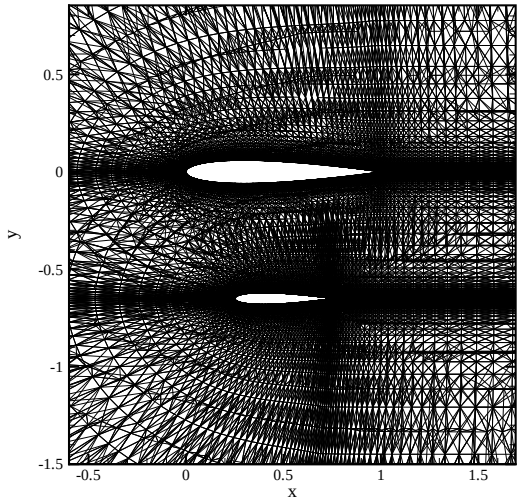
[4]Batina, J.T., "A gridless Euler/NavierStokes solution algorithm for complex two dimensional applications," NASA Technical Memorandum 107631, June 1992



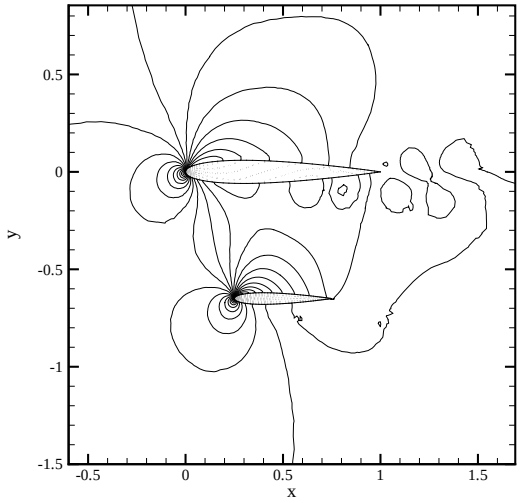
(a) Meshless connectivity at 0.0 sec



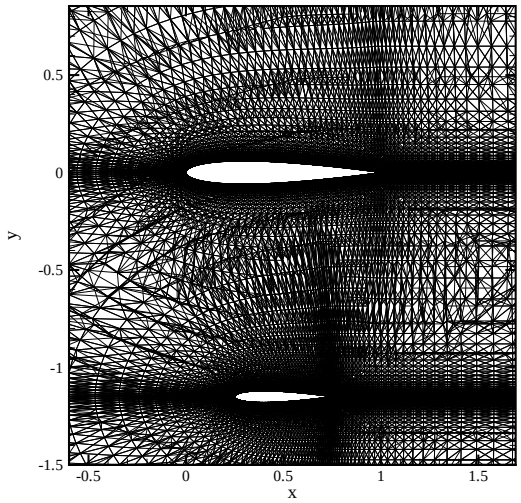
(b) Pressure contours at 0.0 sec



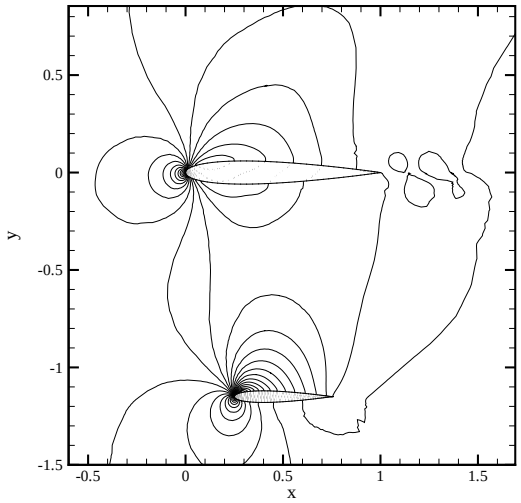
(c) Meshless connectivity at 5.0 sec



(d) Pressure contours at 5.0 sec



(e) Meshless connectivity at 10.0 sec



(f) Pressure contours at 10.0 sec

Figure 21. Unsteady store release case

[5]Oñate, E., Idelson, S., Zienkiewicz, O.C., and Taylor, R.L., "A finite point method in computational mechanics. Applications to convective transport and fluid flow," Int. J. Numer. Methods Engrg. v39. 3839-3866.

[6]Liszka, T.J., Duarte C.A.M., and Tworzydlo, W.W., "hp-Meshless cloud method," Comput Methods Appl Mech Eng 139: 263288

[7]Sridar, D., and Balakrishnan, N., "An upwind finite difference scheme for meshless solvers," Journal of Computational Physics 189 (2003) 1-29

[8]Katz, A., and Jameson, A., "Edge-based Meshless Methods for Compressible Flow Simulations," AIAA Paper 2008-699, 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, January 7-10, 2008.

[9]Sridar, D., and Balakrishnan, N., "Convergence Acceleration of an Upwind Least Squares Finite Difference based Meshless Solver," AIAA Journal Vol. 44, No. 10, October 2006

[10]Chen, H.Q., Shu, C., "An Efficient implicit mesh-free method to solve two-dimensional compressible Euler equations," International Journal of Modern Physics C, Vol. 16, No. 3, 2005

[11]Löhner, R., and Oñate, E., "An advancing point grid generation technique," Commun Numer Methods Eng. v14. 1097-1108.

[12]Lee, C.K., "A new finite point generation scheme using metric specifications," Int. J. Numer. Meth. Engng 2000; 48:1423-1444

[13]Idelsohn, S.R., Oñate, E., "To mesh or not to mesh. That is the question" Computer Methods in Applied Mechanics and Engineering, Volume 195, Issues 37-40, 15 July 2006, Pages 4681-4696

[14]Wang, H., Chen, H., and Periaux, J., "A study of gridless method with dynamic clouds of points for solving unsteady CFD problems in aerodynamics," International Journal for Numerical Methods in Fluids, Vol. 64, 98-118. (2009)

[15]Katz, A., Jameson, A., and Wissink, A., "A Multi-Solver Scheme for Viscous Flows Using Adaptive Cartesian Grids and Meshless Grid Communication," AIAA Paper 2009-768, 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition, Orlando, Florida, Jan. 5-8, 2009.

[16]Munikrishna, N., Balakrishnan, N., "Turbulent flow computations on a hybrid cartesian point distribution using meshless solver LSFD-U," Computers and Fluids 40 (2011) 118-138

[17]Luo, H., Baum, J., and Löhner, R., "A hybrid Cartesian grid and gridless method for compressible flows," J. Comput. Phys. 214 (2006), pp. 618632

[18]Löhner, R., Sacco, C., Oñate, E. and Idelsohn, S., "A finite point method for compressible flow," International Journal for Numerical Methods in Engineering. v53. 1765-1779.

[19]Kamruzzaman, M., Sonar, Th., Lutz, Th., and Krämer, E., "A New Meshless Collocation Method for Partial Differential Equations," Journal of Comm. in Num. Meth. in Eng., 24(4): 1617-1639, 2007.

[20]Seibold, B., "Minimal positive stencils in meshfree finite difference methods for the Poission equation," Comput. Methods Appl. Mech. Engrg. 198 (2008) 592-601

[21]Chiu, E. and Jameson, A., "An Edge-Averaged Semi-meshless Framework for Numerical Solution of Conservation Laws," AIAA Paper 2010-1269, 48th AIAA Aerospace Sciences Meeting, Orlando, Florida, January 4-7, 2010.

[22]Ortega, E., Oñate, E., and Idelsohn, S., "An improved finite point method for tridimensional potential flows, Computational Mechanics," Vol. 40 (6), pp. 949-963, 2007

[23]Stewart, G., "Matrix algorithms vol 1" Philadelphia : Society for Industrial and Applied Mathematics 1998

[24]Duarte, A.C., and Oden, J.T., "An hp adaptive method using clouds," Comput. Methods Appl. Mech. Engrg. 139 (1996), pp. 237262.

[25]Toro, E.F., "Riemann solver and numerical methods for fluid dynamics" Springer-Verlag. Second Edition, June 1999

[26]Roe, P.L., "Approximate Riemann solvers, parameter vectors and difference schemes," Journal of Computational Physics 1981; 43:357372.

[27]Barth, T.J., and Jespersen, D.C., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA paper 89-0366, Jan. 1989.

[28]Mavripilis, D.J., "Unstructured Mesh Discretisations and Solvers for Computational Aerodynamics," AIAA Paper 2007-3955, 18th AIAA Computational Fluid Dynamics Conference, Miami, Florida, June. 25-28, 2007

[29]Katz, A., "Meshless Methods for Computational Fluid Dynamics," Ph.D. Dissertation, Stanford University, January 2009.

[30]Hashemi, M. Y., and Jahangirian, A., "An efficient implicit mesh-less method for compressible flow calculations," International Journal for Numerical Methods in Fluids 2010

[31]Hashemi, M. Y., and Jahangirian, A., "Implicit fully mesh-less method for compressible viscous flow calculations," Journal of Computational and Applied Mathematics 2010 doi:10.1016/j.cam.2010.08.002

[32]Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows Past Airfoils and Wings," AIAA Paper 91-1596, AIAA 10th Computational Fluid Dynamics Conference, Honolulu, June 1991.

[33]Eisenstat, S., Elman, H., and Schultz, M., "Variational Iterative Methods for Nonsymmetric Systems of Linear Equations," SIAM Journal of Numerical Analysis, Vol 20, No. 2,1983, pp. 345-357.

[34]Badcock, K.J., L.Dubuc, X.Xu, and Richards, B.E., "Preconditioners for high speed flows in aerospace engineering," Numerical Methods for Fluid Dynamics, volume 5, pages 287-294

[35]Axelsson, O., "Iterative Solution Methods," Cambridge University Press, 1994

[36]Badcock, K.J., Richards, B.E., and Woodgate, M. A., "Elements of computational fluid dynamics on block structured grids using implicit solver," Progress in Aerospace Sciences, Vol. 36, 2000, pp. 351-392.

[37]Landon, R.H., "NACA 0012. Oscillatory and transient pitching," Tech. Rep. AGARD-R-702, 1982

[38]Bonet, J., Peraire, J. "An alternating digital tree (ADT) algorithm for 3D geometric searching and intersection problems," Int. J. Numer. Meth. Engng, vol 31, 1-17 (1991)

[39]Samet, H., "The quadtree and related hierarchical data structures," Comput Surv 2 (1984), pp. 187285.